A PROPOSAL FOR A DATA BASE FOR THE NATIONAL HEALTH SERVICE

by

Jonathan M. Kerridge, B.Sc., M.Sc.

A Thesis submitted

to the

Victoria University of Manchester

for the

Degree of Doctor of Philosophy

Department of Computer Science

January 1975

ABSTRACT

This thesis describes a data base system which could make the complete computerisation of the National Health Service possible.

The National Health Service is described from its beginnings in 1948 to the present day with its recent re-organisation in April 1974. The flow of information is described and the effects of a computer system are discussed.

The concept of a data base system is introduced and a discussion of present day systems and their shortcomings is presented.

A data base system called PATCOSY (PATient COmputer SYstem) is described which tries to overcome the difficulties of computerising the National Health Service. The syntax of PATCOSY is given in two appendices and an example of the use of PATCOSY is given in another appendix. An implementation and the operating environment of PATCOSY are also discussed.

ACKNOWLEDGEMENTS

I should like to acknowledge the Department of Computer Science for providing the facilities so that this thesis could be developed. In the Department many people have provided advice and guidance, but I should particularly like to thank Professor F. H. Sumner and Miss Hilary Kahn, who has been both mentor and taskmaster when necessary.

I also wish to acknowledge the help of Mr. Angus McNay, of the Newcastle Regional Health Authority, who put me on the correct lines where matters to do with the National Health Service and its re-organisation were concerned.

Financial assistance was provided by International Computers Limited, the Science Research Council and the University of Manchester for the William Kirtley Scholarship, which I acknowledge gratefully.

I should like to thank my wife, Sue, for the help and encouragement she gave me during the writing of this thesis.

Finally, to my mother who typed the thesis and to my father who proof-read the manuscript, the sincerest thanks for giving me the opportunity to be able to do this work.

CONTENTS

CHAPTER 1

THE NATIONAL HEALTH SERVICE

1.1 Introduction

The structure of the National Health Service (N. H. S.) is described from its inception in 1948 to the present time. The original system was constructed from three parts and this lead to deficiencies in the service provided.

The N. H. S. was re-organised in 1974 in an attempt to overcome the faults of the original system. A comprehensive management structure was defined which was dependent upon the creation of a centralised information system if the desired benefits of the re-organisation were to be realised.

1.1.1 The Tripartite National Health Service (1948 - 1974)

The National Health Service was instituted by the N. H. S. Act of 1948. This initiated a health care system through which a person could obtain free medical treatment from one or more parts of the service. The three parts of the service were:

(i) The hospital service

(ii) The general practitioner services

(iii) The local authority health services

In addition, there existed some health services which remained outside the N. H. S.

1.1.2 The Hospital Service (1)

The hospital service provided care for patients requiring the specialist treatment of a hospital environment. This provided specialist doctor and nursing facilities, and laboratory and radiographic facilities so that tests and investigations could be performed. In addition, these investigational facilities were made available to the local general practitioners so that they could send specimens for analysis.

The hospital service had a well defined management and planning structure set up by the N. H. S. Act. The country was divided into regions, each region being managed by a regional board. The regional board was responsible for the planning of new services and the efficient running of the services already provided. Their decisions were subject to the fiscal and policy guidelines set out by the Department of Health and Social Security.

1.1.3 The General Practitioner Services (1)

The N. H. S. Act set up in each local authority area Medical Executive Councils with which practising doctors, dentists and opticians, although they remained self-employed, made a contract of service. The Medical Executive Council also acted as a body to which the public could air their grievances.

Each practitioner had to provide his own equipment and premises and organise his work load to suit himself. The doctor was regarded as the first person to whom patients went when they were ill. The doctor could then treat the patient himself or send him to the local hospital for further investigations. The doctor was thus able to obtain a second opinion if necessary.

People went to their dentist if they required dental investigation and treatment. The dentist could still send the person to the local hospital if he thought this would be advisable. The same was true of opticians except that in order to get this service through the N. H. S. a patient had to get written permission from his doctor.

The general practitioner services had no well defined management structure because all the personnel were self-employed.

1.1.4 The Local Authority Health Services (1)

Under the N. H. S. Act and Local Authority Services Act (1970) local authorities were empowered to provide services at their own discretion depending on the priorities of their locality. Such services were; an ambulance service, epidemiological work and public health control, family planning guidance, health centres, health visitors, home nurses and midwives, maternity and child-care services, vaccination and immunisation schemes, and hostels for handicapped people.

The management structure varied from area to area because it was laid down individually by each local authority.

1.1.5 Other Health-Care Services (1)

Chiropody, speech therapy, the school medical service and other similar services were provided mainly by the local authorities but remained outside the N. H. S.

1.1.6 The Deficiencies of the Tripartite National Health Service

The three major faults of the tripartite system were:

(i) lack of communication between the parts

(ii) fragmentation of service

(iii) difficulty in obtaining reliable management information.

The lack of communication between the parts of the N. H. S. gave rise to the recording of redundant information. An example of this is given in the care of expectant and nursing mothers.

A mother and child would be receiving care from her doctor. The midwife would be in attendance around the time of the birth. If the birth took place in hospital yet another branch of the N. H. S. was involved.

The doctor, the midwife and the hospital would each make their own records thus duplicating parts of the information. Each branch retained its information and no interchange of information took place mainly due to the difficulty of communication.

The apparent fragmentation of service is discussed in (2). A patient receives a fragmented service when he is passed from one branch of the N. H. S. to another. This caused the feeling that the status of doctors in general practice was being undermined because patients themselves came to regard the general practitioner as a poor substitute for the local hospital. This was the result of doctors frequently sending patients to local hospitals for a second opinion. Outpatient departments of hospitals became overloaded because patients started going to the hospitals for treatments which could be provided quite satisfactorily by general practitioners. The hospitals consequently expanded to cope with the extra demand upon their services. As a result of the expansion those patients who consulted their general practitioner expected, as a matter of course, to be referred to the local hospital in all but the most trivial of cases. The result of this expectation, in itself, caused even more fragmentation.

The last major defect was the difficulty of obtaining reliable management information which could be used to plan future health-care services. It was difficult to identify trends because a person's medical record was maintained by many different medical personnel each in a different location.

The only feasible surveys were those organised in one part of the N. H. S. in one locality because most of the records would be easily available. However the final stumbling block was that records were kept on paper and no standard format was employed. To extract information entailed reading many files and much unwanted information.

As a result of these many paper files being read, information, which should have had restricted access, might be available. This would breach the confidentiality which must exist between a patient and the person who was being consulted. Consequently most surveys used information which was collected especially for that survey and not from information which was collected as a matter of course from a consultation.

1.2.1 The Re-organisation of the National Health Service (1974 onwards)

In April 1974 acts were passed which re-organised the N. H. S. in order that rectification of the deficiencies of the tripartite system might be possible. Any change in the running and organisation of the N. H. S. clearly should not result in the worsening of the service; rather, it was intended that the service should improve, thus justifying the upheaval which follows any re-organisation.

It is hoped that the general public will see an improvement by not having to waste as much time and by not having to repeat information which has already been obtained.

People who provide and administer the health service will see improvements in a different way. Medical staff will appreciate being able to provide better health-care services and the availability of more information. The administrator will note that he has more time to analyse and plan rather than just obtain facts. This in turn will give the medical staff more time to devote to their patients.

The initial requirements of the re-organisation are that planning throughout the N. H. S. should improve; other improvements will follow naturally from this. To be able to provide such information, all necessary records will have to be located at one central site, easily available to all users. One user requires only certain pieces of information and therefore the information must be recorded in a coherent manner so that it can be easily extracted.

The manner in which information is organised is governed by the criterion that any information relating to a patient's treatment should be easily accessible to those people who need it. Medical information can be defined as that which directly relates to the patient's treatment. However, such information is recorded in a haphazard way depending on who is recording it.

A record could be one or two words or a long piece of narrative none of which is held in a standard format. Planning and administrative information can be obtained either from a patient's record or from administrative information recorded directly. Examples of direct recording are; appointment and waiting lists, the control of consumables, requests for radiographic and laboratory investigations all of which are recorded by medical or non-medical staff.

Planning information will be obtained by analysing the above administrative information and collating the results with other information such as illness trends and population movements so that an overall picture of future requirements can be obtained.

1.2.2 A Management Structure

The above type of re-organisation is required by the N. H. S. Re-organisation acts (1974): in addition in (5), a management structure is defined within which the N. H. S. will operate. However such structures are liable to alter with political changes. Therefore the information system constructed has to be able to cope with changes in the management structure. Such alterations do not necessarily affect the information required but they do change the information flow.

The re-organisation provides for the combination of the three parts of the original N. H. S. into one in order that management can be centralised. In such a centralised management system it is sensible to provide a centralised information system.

Epidemiological work will be of more widespread use and influence because a larger statistical sample will be possible. However, the data collection problem will only be solved by the use of standard format records into which all information is formed.

Medical staff will be able to see all information which has been previously recorded about a person which will reduce the amount of redundant information. This becomes more relevant when (2) (3) (4) are considered. These reports emphasize that there should be a redistribution of staff towards the general practitioner services. In other words, midwives, home nurses, health visitors and social workers should be attached to one surgery and should operate from that surgery. This may result in a greater interchange of information, and the provision of a coherent service to the public. The general practitioner will then regain his role as the mainstay of the health-care services.

<u>1.2.3 Provision of a Centralised N. H. S.</u>

A computer based information system, in which data is held in an organised manner, is one solution to the problem of using information remote from a central location. In addition to making data more readily available, a computer system can process vast quantities of data selecting only those items which are required for a particular survey.

The N. H. S. is described (6) as being run on a system of crisis management; that is, wait until something goes wrong and then correct it. If such a method could be avoided by sensible planning then the staff of the N. H. S. would be able to see actual improvements in the service they are providing.

If a computer based system is to be considered, it is necessary to see how such a system could be implemented at both the local and national levels, remembering at all times that, despite the urgent need for useful management information the highest priority must be given to medical information about patients.

CHAPTER 2

THE DESIGN OF A COMPUTER BASED NATIONAL HEALTH SERVICE

2.1 Introduction

The design considerations of a computer based N. H. S. are discussed. Special consideration is given to the categories into which information falls, how the information is used by the different parts of the N. H. S. and who is authorised to access a particular piece of information. The physical realisation of a computer based information system is described at both the local and national levels.

2.2 Information Types

The first task when designing any information system is to define the main categories into which the data will fall. In the N. H. S. four categories can be identified:

(i) the patient's medical record (P.M.R.)

(ii) paramedical information

(iii) planning information

(iv) financial information

The relationships between these different categories are shown in fig. 2.1. The figure is divided into four areas each corresponding to a different category of information. The personal details of a patient are separated from the rest of

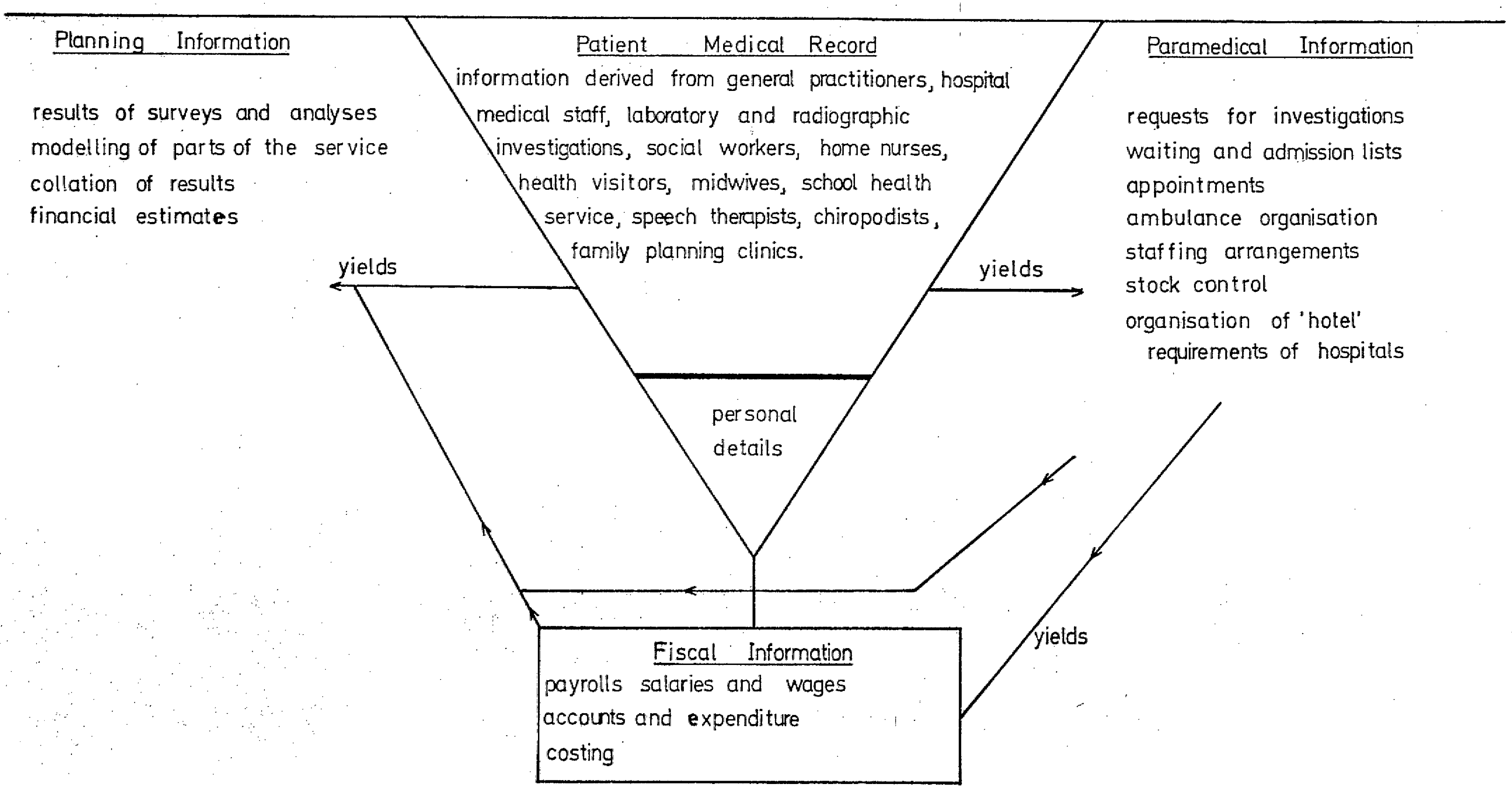

Planning Information
results of surveys and analyses
modelling of parts of the service
collation of results
financial estimates
Patient Medical Record
information derived from general practitioners, hospital medical staff, laboratory and radiographic investigations, social workers, home nurses, health visitors, midwives, school health service, speech therapists, chiropodists, family planning clinics.
Paramedical Information
requests for investigations
waiting and admission lists
appointments
ambulance organisation
staffing arrangements
stock control
organisation of 'hotel' requirements of hospitals
yields
yields
personal details
yields
Fiscal Information
payrolls salaries and wages
accounts and expenditure
costing

Fig2.1 Information Types and Flow

the medical record so that it is impossible for anybody to access the information unless he has the authority and the reason to do so. The arrows labelled 'yield' indicate where information from one category can be used to form information of a different category.

The figure shows the main information types which fall into each category. The P.M.R. can be used to yield other information which will be useful in the day-to-day running of the N. H. S. and also for the future planning of the N. H. S. Financial matters are handled separately and are used for both planning and budget control.

The example of stock-control provides an insight into the interaction of the different categories. First taking medical consumables; a patient is given a known quantity of a consumable and therefore the stock must have been reduced by that amount. By looking at the rate of usage the most economic re-ordering period can be defined. Also a record of expenditure can be easily maintained.

Secondly, non-medical consumables; these in the main are used by hospitals in their function as hotels. Examples of these consumables are; food, linen and building materials used for running repairs. By combining the stock-control information, expenditure, admissions-lists and population movement figures, future requirements can be better planned.

2.3 Information Privacy and Standardisation

In any system in which a person's medical information is maintained there must be absolute privacy so that no unauthorised person has access to the information. When designing a computer system it is necessary to define which users may read or write a particular piece of data. In some cases a person may be able to write information but not read information of the same type. This is especially true of the socio-economic status of a person. For example, a home nurse may note that a child has no shoes but does not need to know any other information which has been previously recorded. The person who reads all the information could be an almoner or social worker.

Information must be written in a standard way so that it can be quickly accessed extracting only those data items which are required. This is a great stumbling block where medical information is concerned because each medical person has traditionally kept his own records for his own use and not as part of an overall system where medical records will be used to plan for the future.

2.4 The Organisation of a Local Information System

The N. H. S. performs two tasks at the same time. First, it provides a service for each community and secondly, the N. H. S. provides a plan for the nation.

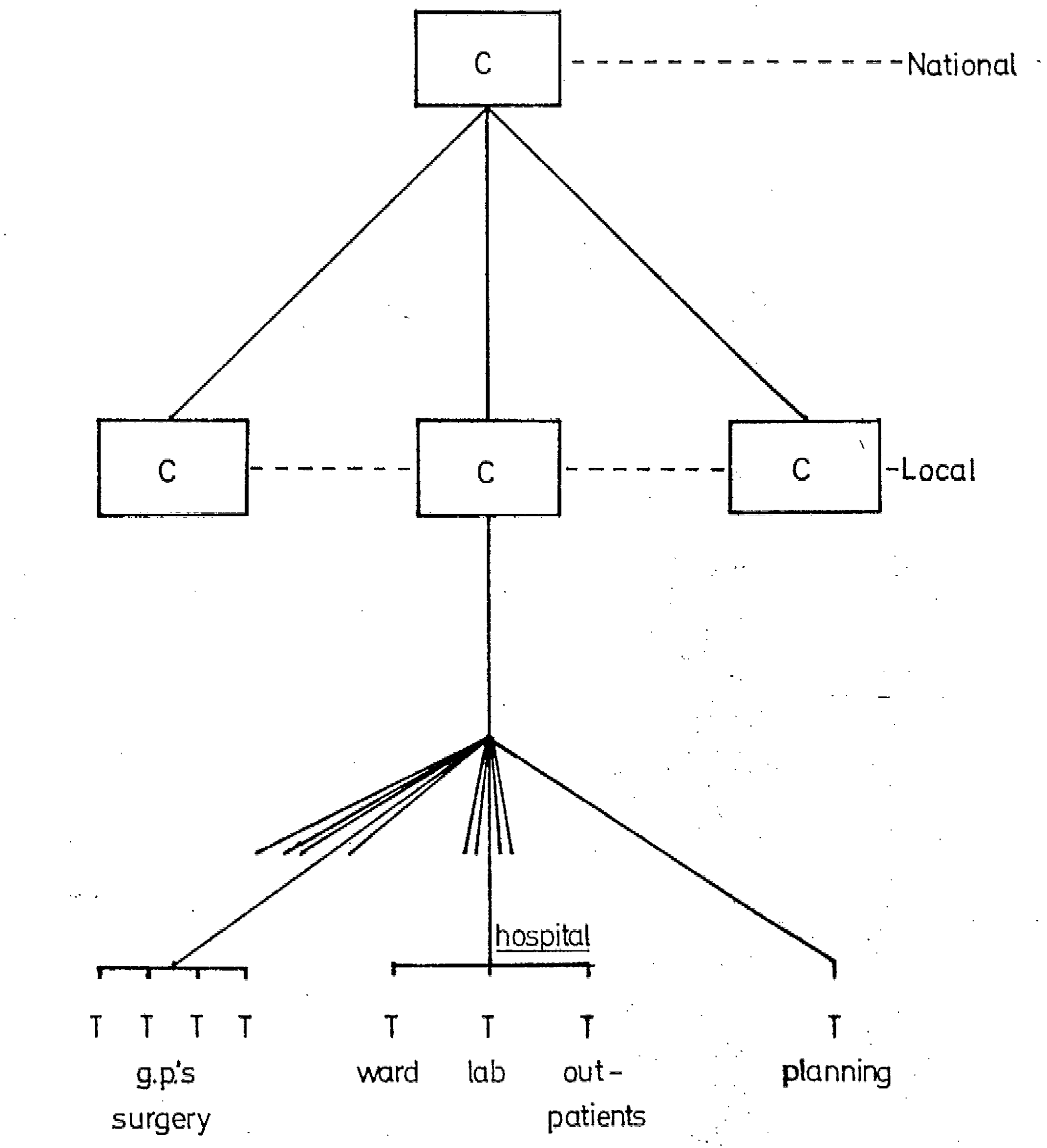
At the local level the computer system must serve the personnel of the N. H. S. That is, all personnel who read and write information must have an easily accessible means of communication with the computer.

A local computer could serve a district which is defined in (5) as the number of people who can make effective use of the facilities of a district hospital. In metropolitan areas it may prove more efficient for one computer to serve a larger population. This is because in such localities hospitals tend to specialise and people have greater freedom of movement and may frequently travel and work outside their locality.

The computer can handle all the para-medical information created by the locality. Appointments will become more certain because patients can be presented with a choice of appointments and the chosen appointment booked there and then. At the same time an ambulance can be booked if required. This will lead to the more efficient use of the ambulance service as journeys can be planned so that the maximum number of patients can be quickly transported provided one person is not in the ambulance too long.

2.5 The National Information System (see fig. 2.2)

A national information system is required for two reasons; first to provide inter-locality communication and secondly for the



T - Terminal
C - Computer

Fig 2.2 National Computer-Based Information System

planning of future N. H. S. requirements.

Inter-locality communication is required because information may be required in a locality different from the one in which the information was defined. With greater freedom of movement a person may be taken ill a long way from his 'home' locality. When a person moves from his home to another locality, his medical record is passed from one general practitioner to another. At present the transfer can result in the record being unavailable for a long period. In both cases a national computer based information system can overcome these problems. A national system enables medical staff to have access, at all times, to a full medical history of any person wherever they come from.

Most planning is carried out at the national level, even though many surveys are carried out for and by a locality. If there was no national system, data collection would be extremely difficult and one of the aims of a centralised system would be lost.

Further a national stock control system could be implemented in which purchasing and distribution of stock could be nationally controlled. At present stock-control is carried out mainly at the local or hospital level. This means that over the nation as a whole there is a large amount of stock being stored. If there were national control, the amount of stock could be reduced thus reducing capital outlay to the most economic possible.

A computer based information system would provide a very real possibility of improving the N. H. S. There are many different ways in which the system could be implemented, one of which is to look at the total system and see how relationships between data are formed. Such a method is called a data-base system.

CHAPTER 3

DATA BASE SYSTEMS

3.1 Definition of a Data Base

A data base is a means of recording information so that the following properties hold:

(i) Users can obtain information in whatever form they require.

(ii) Privacy and integrity of information is secured.

(iii) The recorded form of the information is independent of the storage device.

(iv) Relationships within the data can be created and maintained.

(v) There is a large degree of program/data independence.

(vi) No redundant information need be recorded.

It is also relevant to define the difference between data and information. Data becomes information when it is surrounded by other details which give it meaning. For example, to say a car is going at thirty-five has no meaning. The data thirty-five only becomes information when it is followed by miles per hour or kilometres per hour.

3.1.1 Information Forms

A particular piece of data may be used in many different ways and may yield different information when surrounded by different external symbols. This concept is extended to provide a means of recording data in one form and then, by altering the form and recombining the data in different ways, many information forms can be created at will.

One user may require parts of what other users regard as a whole, thus leading to a definition of physical and logical records. A physical record is data which is recorded as a unit at one time. A logical record is created from parts of one or more physical records.

3.1.2 Privacy and Integrity

With many different users accessing data from remote locations it is vitally important that privacy should be maintained so that no user is able to gain access to data for which he has no authorisation. When a user operates upon a data base he has to identify himself so that his privacy rating can be established and used throughout the operation.

The integrity of the data base has to be maintained; that is, data must be recorded correctly. In addition, the system must be able to recover from both machine and program faults without changing the data recorded in any way.

3.1.3 Recording Format

The recording formats throughout the system must remain the same regardless of the storage media. In other words, when data is moved from one storage device to another the recording format must not change. If it did, programs would have to be re-written several times to cope with the many different storage media.

3.1.4 Relationships

The concept of a relationship is one which does not occur in traditionally organised computer systems. A relationship is a means of grouping records together because some of the data items in the record have some property in common. A record in a data base stands by itself. In a traditional system a record is part of a file which is a collection of records of the same type. All access is through the file and the creation of relationships is implicit in the programs which use the files.

By allowing relationships, a data base allows the user a very powerful means of obtaining information. Instead of going from one record to a record of a similar type, a user can go where he wants provided a relationship has been created and maintained.

3.1.5 Program/Data Independence

A lot of expense is incurred by a traditional system in maintaining programs. Records are given a format and then each program is written with that format embedded in it. If the data format changes, an expensive re-write of programs may be needed. This problem is overcome in a data base system by the use of data maps.

A data map is a table which contains all the data definitions of a record. All data references are through the data map definitions which give the starting position of a data item relative to the start of the record. For each physical or logical record defined in the data base there is an associated data map.

If a definition is changed only the programs which reference that data item will have to be altered. All the other programs will remain the same thus saving programming effort.

3.1.6 Data Redundancy

Because parts of many records can be combined (see 3.1.1) to form another record, there is no need to define a data item more than once. In addition, the absence of repeated data, means that storage space can be saved. However, such a method does require a very flexible privacy system.

3.2 The Role of the Data Base Administrator (D.B.A.)

The design of such a complex entity as a data base requires that one person should have overall control. This person is the data base administrator. Initially when the system is defined relationships, record formats, privacy considerations and storage media characteristics all have to be taken into acount. Finding the right balance and mixture is the province of the D.B.A.

Once the data base is in use, the D.B.A. has to maintain the data base, making improvements where possible. New users have to be assigned a privacy coding and any alterations to the data base must be thoroughly checked to make sure that the integrity of the data base is maintained.

3.3 Manipulation of the Data Base

Once a data base has been defined a means of accessing the data contained in it has to be provided. There are two ways of doing this:

(i) Host-language systems

(ii) Self-contained systems

3.3.1 Host-Language Systems

This method provides the programs with several commands which operate upon the data base making the required data available for the program. The data is then processed by the language within which the manipulation command was embedded. It is therefore possible for the data to be processed by programs written in many different languages provided an interface has been defined between the programming language and the data base system.

3.3.2 Self-Contained Systems

This method provides in one self-contained unit, both the commands which operate upon the data base and those which process the data. This means that data can only be processed by the facilities which are made available by the system.

There are some systems currently available (see 3.4) which are basically self-contained or host-language but which contain facilities and functions which are provided by the other method.

3.4 Currently Defined Data Base Systems

There are at present several data base systems which have been defined: some of these are:

(i) The Data Base Task Group recommendations of the CODASYL committee (7)

(ii) Systems developed by manufacturers and software houses.

(iii) The Relational data base system defined by Codd (8)

3.4.1 The Data Base Task Group Recommendations

The D.B.T.G. report defines a host-language system which comprises two main languages. These languages are a data description language and a data manipulation language. There are many forms of the data manipulation language because the commands are intended to be embedded in many different languages. The report (7) in fact only gives the form for a COBOL orientated data manipulation language.

The data is defined at two levels called the schema and the sub-schema. At the schema level records are defined as well as the relationships which occur between the records. A record is constructed from data items. Examples are, vectors, scalars and repeating groups. A record may occur any number of times in the data base. A data base can be defined in terms of one or more schemas.

A sub-schema is the part of the definition of a whole data base which is relevant to an application or program. At the sub-schema level the characteristics of data items and records can be changed as necessary. The privacy requirements of a piece of data can also be changed. The sub-schema data description language is orientated towards the host language in which the particular application is written, in the same way as the data manipulation language. The properties of a D.B.T.G. defined data base are the same as those given in 3.1.

The data manipulation language provides a means of accessing data either through relationships or by directly specifying a particular record occurrence. The data is obtained and passed to the user program which uses the appropriate sub-schema definition to process the data in the required host-language.

The D.B.T.G. proposals are the ones with which all other systems have to be compared because they are so wide-ranging.

3.4.2 Manufacturers' Data Base Systems

An analysis of manufacturers' data base systems was carried out by the CODASYL systems committee and its findings were published in a report (9). The systems analysed can be divided into self-contained and host-language systems.

3.4.2.1 Self-Contained Systems

The systems discussed in this part of the report were; Generalised Information System, Mark IV, National Military Command System, Time Shared Data Management System and User Language/1.

SYSTEM NAME	ABBREVIATION
Generalised Information System	GIS
Mark IV	Mark IV
National Military Command System	NMCS
Time Shared Data Management System	TDMS
User Language/1	UL/1

Fig. 3.1 Self-Contained System Names and their Abbreviations.

3.4.2.1.1 System Definition

All the systems allow the definition of large file orientated systems which have to be interrogated and updated. The manner of file definition is in one of two ways; either by a narrative English-like program sequence (GIS, NMCS, TDMS), or in a separator and keyword format. The latter is inherently less readable for the non-programming user. Once the system has been defined only GIS, TDMS and UL/1 allow subsequent redefinition of the data base.

3.4.2.1.2 Internal Structure

A GIS data base is made up from many files, each file containing an arbitrary number of entries of the same record type. A record consists of a master group and up to fifteen sub-ordinate groups. Facilities are provided so that access to a file or record can be restricted to authorised users.

Mark IV employs a tabular form so that retrieval and maintenance functions are easier. Dictionaries are maintained which hold the definitions of the files and records and also of the files which are used to update the main data files. The manner in which the update is to take place is also held in dictionaries.

NMCS was designed specifically for the military environment and allows data items to be defined as numeric or alphanumeric quantities or as geographic co-ordinates. The lengths of individual items can be defined and those items which comprise a larger unit can be grouped together. In addition conversion mechanisms and editing requirements can also be defined.

TDMS uses hierarchical data handling and storage structures enabling the user to define large data files having no recourse to the internal structure of the computer being used.

UL/1 allows the user to define files which contain records that can be constructed from up to 15 levels of nested groups. The groups and items of a record can be of variable length.

3.4.2.1.3 Mode of Operation

GIS was designed so that unanticipated and unstructured interrogations could be handled. The interrogation is performed in a narrative style. Data can be updated and deleted as well as completely new data being written to the data base. The self-contained facilities can be extended by the installation providing own code procedures which can be added to the GIS commands. The interrogations can take place in both batch and on-line modes.

Mark IV was designed to provide batch processing of commercial data. Information requests can initiate the extraction of parts of records, report writing and the construction of new files. Requests that are frequently used can be joined together to run as a batch of requests.

NMCS operates mainly in a batch mode but does allow on-line information requests. The system was required for processing large data files, on a repetitive basis, initiated by non-computer personnel. Requests are made in a narrative English-like procedural language. The system allows file processing to proceed at many different levels of computer know-how.

TDMS allows data to be written which can subsequently be altered at will. Data can be selectively retrieved and presented to the user in whatever form he requires. The large files are maintained by means of sort, merge and other data handling utilities which are provided by the system.

UL/1 provides a system which can interrogate the data base and then process the data in a procedural language. Data can be presented to the user in any form he requires. Sub-files can be extracted and records can be sorted on any specified key. Data can be updated at any level from the data item to the record. The system is designed to run in batch mode only.

3.4.2.1.4 Other features

GIS maintains a record of all transactions which alter the data base so that recovery can be effected after a hardware fault. GIS and UL/1 allow the user to interrogate files which are not maintained as part of the data base. GIS can handle IBM compatible file structures and UL/1 uses COBOL defined files.

3.4.2.2 Host Language Systems

The systems discussed in the report (9) were; Integrated Data Store, Information Management System, and System Control-1. In addition, the Total data base system is discussed.

SYSTEM NAME	ABBREVIATION
Integrated Data Store	IDS
Information Management System	IMS
System Control-1	SC-1
Total	Total

Fig. 3.2 Host Language Systems and their Abbreviations.

3.4.2.2.1 System Definition

IDS uses files which may be distributed across different direct access devices. The basic file relation is a chain which contains a master group which points to subordinate detail groups. The detail groups are only available from the associated master group. A master group can be the start of many detail group chains. In addition, a detail group can be the master group of a different detail group chain. Another record type, a calculated group, is also available.

A calculated group is retrieved by specifying the value of a data item from which the physical location of the group can be obtained.

IMS was designed as an augmentation of International Business Machines (IBM) Operating System/360 and as a consequence some of the facilities normally associated with a data base are not present. A data base is defined by many tree-structured files. An element of a file is able to have a relationship with any element of any other file. In addition, file definitions can be combined to form logical files and in fact all access is by means of logical file definitions. The user therefore has to define both logical and actual files.

SC-1 and Total both use a keyword and separator type definition language. IDS and IMS use a narrative type definition language. A SC-1 data base allows the definition of data items; the location, security and conversion mechanisms for the data can also be specified. Subsequently restructuring and redefinition of the data base is provided.

The Total system provides a means of defining networks using master records and dependent records which are chained to the master record. Records are of fixed length and contain fixed length items. A record can be divided into groups to a maximum of 32 nested levels.

3.4.2.2.2 Mode of Operation

IDS employs COBOL as the host language. The COBOL programs are augmented by IDS commands which make data available from the data base. A master group has to be retrieved by specifying a pointer which the user has to maintain.

IMS was designed so that many applications programmers could share the same data. Before the system can be used, programs have to be written, dependent upon the installation, which allow communication between the data base and the peripherals which are used in the installation. Application programs can be written in COBOL, PL/1 and 360 Assembler. IMS can only be used on IBM 360 and 370 series computers. Functions included in IMS are; the fetching of records which have specified item values, updating records, writing new records, obtaining dependent records. Depending upon the machine and the operating system the data base may operate in an on-line mode.

SC-1 was designed so that users with varying amounts of computer knowledge could use the system. Therefore, some self-contained interrogation functions are provided which allow interrogation and updating of the data base. Data can be tailored to fit a particular application by redefining the data into logical records. In a host-language environment, language elements are embedded into programs so that the required data can be accessed. SC-1 allows batch mode processing only.

Total processes the data base by means of a call function, one parameter of which specifies the type of operation to be performed. The data is then processed by a host-language program in which the call command was embedded. Records belonging to a chain are only available from the associated master record. The system can be implemented to run in both batch and on-line modes.

3.4.2.2.3 Other Features

IMS includes a check-point facility so that restart and recovery after a machine fault are possible. A log of data base usage is also available so that re-organisation can take place.

One final comment pertinent to both self-contained and host-language systems is that all the systems discussed except UL/1 and IDS were designed to run on IBM 360 and/or 370 series computers and therefore all are dependent upon those machines and their software. IDS was designed by Honeywell and UL/1 was designed for an R.C.A. Spectra 70 computer. Total can also run on Honeywell computers.

3.4.3 The Relational Data Base System

This system employs relational calculus and is solely based upon the user defining relationships which occur between the different data items in the data base. As yet no method of implementing the system has been defined but the system clearly provides a very powerful mathematical tool for obtaining data.

One important aspect of the system is that it forces the user to define a data base with regard only to the relationships between the data items. No consideration has to be given to storage allocation or how the relationships are to be maintained. Thus in a data base definition it is possible for one item to appear in many different relationships.

One way the system could be implemented is by using tables, as the relationships naturally assume a tabular form. Large tables are expensive to store, update and maintain and, when a large number of relationships has been defined many tables will be created each containing key items which will have to be repeated in other tables.

3.5 The Data Base Requirements of a National Health Service Computer System.

The nature of information in the N.H.S. requires that certain facilities are provided over and above those which are normally provided for commercially orientated data base systems of the type discussed in section 3.4. The necessary facilities are:-

3.5.1 Storage Control

Data in the N.H.S. has a rotational nature; that is, data is only created or interrogated when the patient is receiving care. At all other times the data lies dormant and unused except for analysis by management applications. Dormant data does not have to be immediately accessible; it must, however, be available when a patient starts a new episode. Therefore dormant data can be recorded on non-direct access devices provided that it can be quickly transferred to direct access devices when needed.

3.5.2 Privacy

All medical information is private to the patient and the medical staff in attendance. At no time must any unauthorised person be able to gain access to medical data. However, management must be able to access the data so that realistic planning can be carried out. This leads to a situation in which an hierarchic privacy structure becomes impracticable and therefore a privacy structure has to exist in which absolute values are used with no hierarchic connections.

3.5.3 Data Erasure

Medical information cannot be deleted during a person's life time and thereafter is held for a considerable period of time, usually one hundred years. This means that data must be archived but in such a way as to make it readily available for management applications.

3.5.4 Variable Data

Medical staff have traditionally written narrative to describe a person's condition and treatment. Although narrative is inherently unsuitable for processing and recording, the facility must still be provided. Even in a well structured medical record there is still a large amount of data which occurs in variable quantities and therefore a flexible data structure system is required.

3.5.5 Tasking of Operations

In hospitals many of the investigations carried out upon a patient are done at the request of another member of the medical staff. A means should be provided where the requests can be ordered and presented to the many laboratory technicians.

3.5.6 Research and Analysis

At frequent intervals research projects are initiated which require new relationships to be defined in the data base. It is impracticable to redefine the data base for what are transitory definitions. Therefore a means should be provided of creating a relationship, external to the medical record, which can be used for a short time and then discarded.

3.5.7 Data Base Redefinition

Over a period of time illness patterns change, new drugs become available and new treatments are developed. All these will require alterations to be made to the organisation or definition of the data base to reflect these changes. The data base must therefore be capable of being redefined.

As computer technology improves and new devices become available it must be possible to restructure the data base so that the new devices can be incorporated into the system.

3.5.8 Operating Mode

The N.H.S. provides an environment in which data is processed in batch mode and on-line. Medical data is produced and interrogated in real time whereas management information is extracted by analysis which can proceed in batch mode. Therefore the data base must be capable of efficient operation in both modes.

3.5.9 Introduction of a Data Base System

At present the N.H.S. has a large number of programs which have been developed. These programs deal mainly with financial and statistical details. If a comprehensive data base system is introduced there is a large amount of programming which would have to be completed before the system became operational. It would therefore ease this load if previously written programs could still be used after the introduction of the data base system.

3.6 Comparison of N.H.S. Requirements and the Provisions of Current Systems

The comparison is summarised by fig. 3.3 in which an 'X' indicates that the facility required by the N.H.S. is provided by the system.

3.6.1 Storage Control

None of the systems discussed provides any means of defining the transfer of data from one storage device to another.

3.6.2. Privacy

Only the DBTG proposals and GIS provide a means of assigning a privacy coding to the data items, records and relationships. However, the method of implementation is by procedure calls which will tend to lead to an hierarchical system. Many of the commercial systems provide a security feature in which a check is made to ensure that data is recorded correctly. DBTG has no provision for recovering from a machine or program failure which is a necessary feature of

SYSTEM / REQUIREMENT	GIS	Mark IV	NMCS	TDMS	UL/1	IDS	IMS	SC-1	Total	DBTG
Storage Control										
Privacy	X									X
Data Erasure										
Variable Data					X					X
Operation Tasking										
Research & Analysis						X	X		X	X
Redefinition	X			X	X		X			
Operating Mode	X		X			X				X
Introduction	X				X					

Fig. 3.3 Comparison of Systems and Requirements

any N.H.S. system. Many of the commercial systems provide this feature.

3.6.3. Data Erasure

All the systems discussed were defined for use in the commercial environment or a specialist military environment (NMCS). In the commercial environment data is not maintained for say at least one hundred years and the archiving problem is not on so large a scale. The facilities in these commercial systems are inadequate to deal with this requirement.

3.6.4 Variable Data

Only DBTG and UL/1 provide variable data facilities and the methods employed are relatively inflexible because these languages are specifically designed to interact with COBOL which has a very limited variable field facility.

3.6.5. Tasking of Operations

None of the systems directly states that a provision is made for the tasking of operations to be carried out. Although an applications programmer could meet such requirements using the facilities of the system.

3.6.6. Research and Analysis

Only IMS, IDS, DBTG and Total provide network facilities and in all cases the networks are defined at the same time as the data base. Therefore the requirement of a transient relationship cannot be met.

All the other systems do not provide network systems which are imperative for reasonable research and analysis to be carried out.

3.6.7 Data Base Redefinition

Only IMS, UL/1, TDMS and GIS allow redefinition.

3.6.8 Operating Mode

Only IDS, DBTG, NMCS and GIS allow both batch and on-line processing to proceed at the same time.

3.6.9 Introduction of the Data Base System

Only UL/1 and GIS allow any communication with data which is created outside the data base or allow processing of data created in the data base from outside the data base.

3.7 Summary

In conclusion therefore none of the systems is able to satisfy the very demanding requirements of the N.H.S. The system which comes up to the specification most closely is the DBTG proposal.

Another drawback to using manufacturers' software is that the N.H.S. could then be tied to one manufacturer's product. If the manufacturer updated his hardware without updating the software the N.H.S. would be left with a system which would become progressively more obsolete.

In the following chapters a data base system called PATCOSY (PATient COmputer SYstem) is described which attempts to satisfy the above requirements of the N.H.S. PATCOSY also provides a very flexible commercial data base system.

CHAPTER 4

INTRODUCTION TO PATCOSY

A PATCOSY data base system is defined by a definition program and the data is then manipulated by application programs. The PATCOSY definition language (PDL) provides a means of defining data structures, privacy requirements, relationships, storage requirements and data transfer. PDL is used to form a data definition program which is then compiled and run; the output from the program is in the form of data maps which can be manipulated by the PATCOSY manipulation language (PML). Appendices 1 and 2 give the full syntax and semantics of PDL and PML.

4.1 The PATCOSY Definition Language

A PATCOSY data base is defined at three levels; the data item, the record and the basis levels.

4.1.1 The Data Item

PATCOSY allows data structures to be defined which contain the following different types of data item; fixed and variable length items, fixed and variable length repeating groups, and items which are known as associate items.

A fixed length item contains a defined number of characters or bytes, which will remain constant throughout the life of the data base. In addition, a fixed length item can be further qualified as computational. A computational item always contains numeric data held in a floating-point format appropriate to the machine of implementation. An ordinary fixed length item is held as a string of characters.

A variable length item has the number of characters which it occupies defined at the time the data is written. A variable length item is always held as a string of characters.

A repeating group is an item which is itself constructed from other items. It is therefore possible for a repeating group to contain repeating groups to any level of nesting provided the final level contains only fixed or variable length items.

A fixed length repeating group is repeated a known number of times. A variable length repeating group has the number of repetitions defined at the time of writing.

An associate item assumes the value or values of other items depending upon the value of an item which is contained in the same data structure. An associate item provides a means of encoding information and, when the item is associated with more than one value, a means of reducing storage space.

4.1.2 Records

A PATCOSY data base is defined in terms of different classes of record. A data base may contain several occurrences of the same class being used in many different data structure definitions (record types). The different classes are; structure, sub-structure, list, dictionary, table, index, pointer-list, pointer-array and transfer-structure. All the different record types have their contents defined as a sequence of data items.

A structure provides the main means of recording prime data. Structures are formed into strings which are accessed by specifying a key value. A structure is the physical record of a PATCOSY data base.

A sub-structure is the logical record of the data base. That is, a sub-structure is created by taking parts of one or more structures and then considering these as a unit which can be used as required. Further, the definition of an item can be changed between the structure and the sub-structure.

The system can contain many different list occurrences; within each list there is no external key value which allows different list members to be distinguished. When a list is manipulated a record is kept of which list member was most recently processed. This allows tasking to start from a point somewhere within the list if necessary.

A dictionary is the mechanism which is used with associate items. A dictionary is a data structure which contains a sequence of elements each containing the same field definitions. Any specified fields can be used as key items so that any element of the dictionary can be uniquely identified by specifying a value for one of these items. A dictionary can also contain items which do not act as keys.

A table is similar to a dictionary except that the table elements are held as a contiguous sequence and the items of the table are all of fixed length. Normally a table will be used to provide a very rapid look-up mechanism and will be kept short. Up to two table items can be used as keys.

An index is the means by which different strings of the same structure type can be identified. The index contains the key value which differentiates between the differing structure strings. If a data base contains more than one structure definition, the index will point to a pointer-list.

A pointer-list contains a sequence of items each of which points to the start of a string of structures.

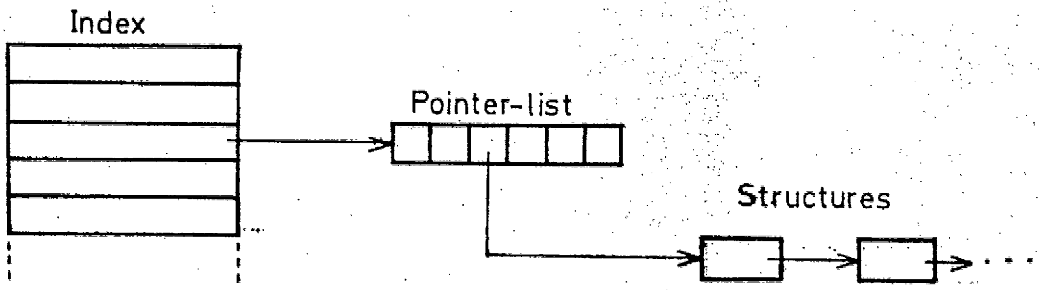


Fig 4.1 The Index/Pointer-list/Structure Construction

In some situations a structure definition could be used for many different applications, each application being differentiated by some key item. In this situation the pointer-list element would point to a pointer-array.

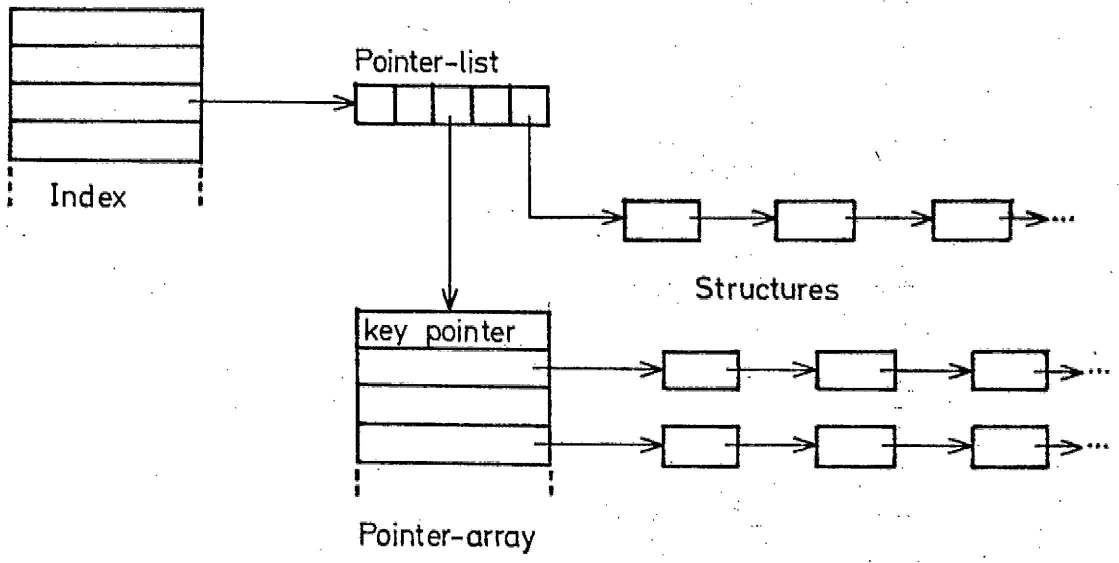


Fig 4.2 The Index/Pointer-list/Pointer-array/Structure Construction

A pointer-array element contains the value of the key item and a pointer which points to the start of the associated string of structures.

The transfer-structure is a means of processing data outside the data base. Data is collected from within the data base, re-organised as necessary, and then transferred to a file which is maintained outside the data base. The reverse process, whereby data is transferred into the data base is also provided.

At the record level relationships and storage control are also defined. There are two main types of relationship;

those that join similar records together (primary) and those that can join dissimilar records (secondary).

The primary type joins strings of structures or list elements. This type is also used to form the internal relationships of dictionaries, tables and indices (ACCESSED BY). The different relationships allow strings with 'pointer to next' (LINK) or 'pointer to next and prior' (DOUBLE-LINK) to be formed. A more complex binary tree structure (TREE) can also be formed in which an element of the relationship contains two 'pointers to next'. One of the pointers points to elements which have higher key values, the other points to those with lower key values.

The secondary type of relationship allows records which have some feature in common to be joined together. Such relationships can either have a starting point which is maintained by the data base system itself (GROUP) or a starting point which is maintained in another record (CHAIN).

Other relationships make it possible to order the elements of a repeating group on some key value or values. It is also possible to organise a dictionary so that the most frequently accessed records of the dictionary are the most readily available. (FREQUENCY).

Having established the different types of relationship provided, it is necessary to define the different mechanisms which can be used to implement the relationship.

The DIRECT mechanism uses the physical location of the record on a storage device for any relationships which require the record to be part of the relationship. The ALGORITHM mechanism is similar to the DIRECT mechanism except that the physical location of the record is calculated using one or more items which may or may not be part of that record.

The DISPLACEMENT mechanism creates pointers which use the displacement of the record occurrence from the start of a DIRECT primary relationship. If the definition of a record type indicates that DISPLACEMENT pointers are to be used then each occurrence of that record type is uniquely identified from all other occurrences in the particular DIRECT primary relationship to which the occurrence belongs. This is achieved by assigning sequence numbers to the record occurrences as they are written.

The advantage of DISPLACEMENT relationships is that they are independent of data transfers from one storage device to another. They are however, necessarily slow as they will inevitably include a search for the record occurrence required. For example, if structure occurrences are the members of a DISPLACEMENT relationship the pointers will be constructed from an index value, which indicates the DIRECT primary relationship required, and a number which identifies the occurrence required in that primary relationship. In Fig.4.3 three instances (P1, P2, P3) are shown of the same DIRECT primary relationship, crossing these are two distinct DISPLACEMENT secondary relationships (SA, SB).

It is obvious that a particular record type must have at least one DIRECT primary relationship defined, otherwise, there will be no means of physically locating record occurrences.

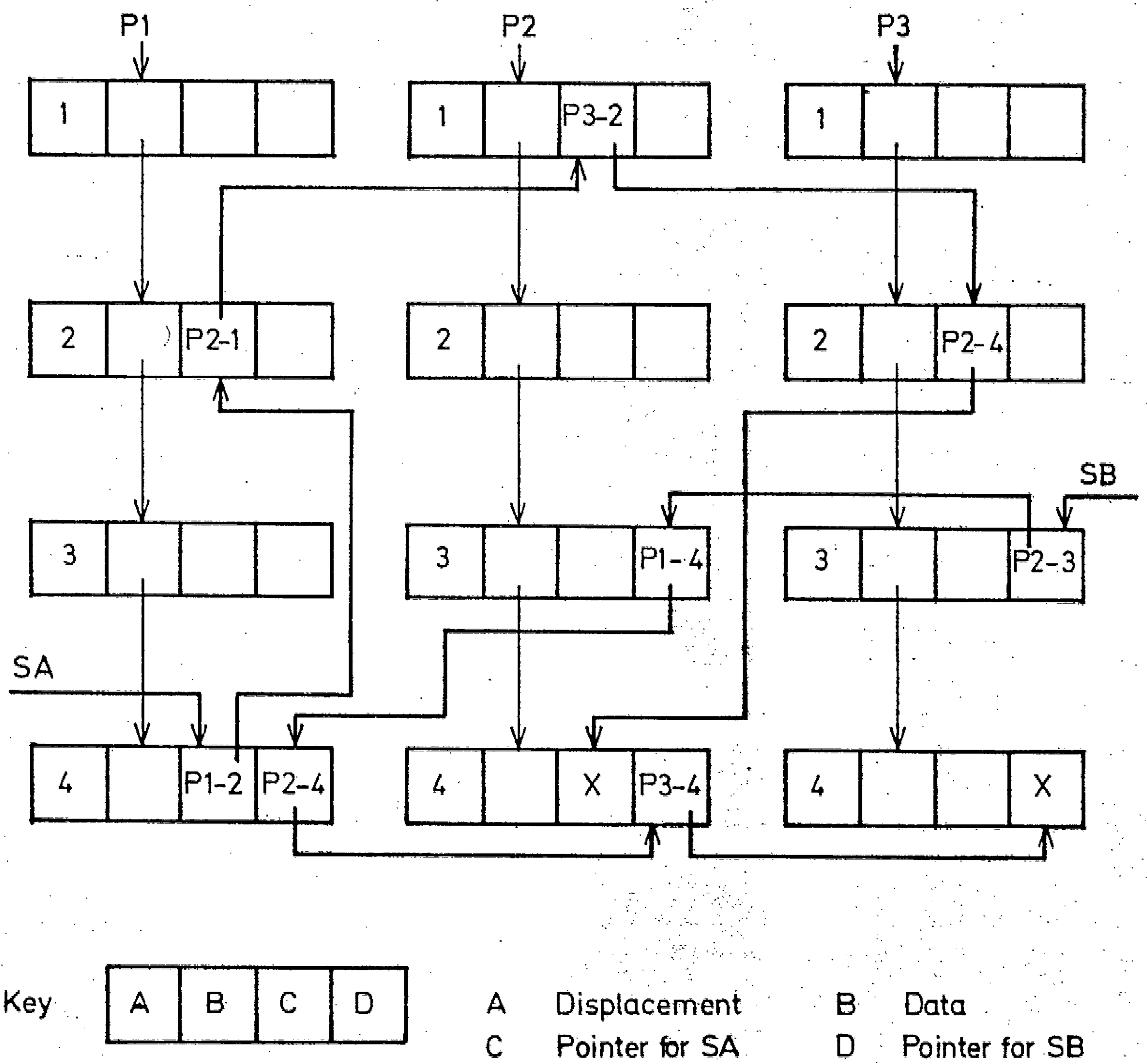


Fig. 4.3 A Direct and Displacement Pointer Example

Storage control is the means of allocating record occurrences to particular storage devices and further, providing a sequence of control so that records can be moved from one device to another in a sequence which finally either destroys the data or archives it on the required storage device. There are two ways of passing through the sequence of storage controls. The first is by a period of time elapsing since the record was moved to the device. The second is by setting a flag which is associated with a particular storage control transfer. It is also possible to combine these two ways.

Once data has been archived two possibilities can arise; either the data has to restart the sequence of storage controls or the data has to be made accessible for read access only so that research and analysis can take place more efficiently. This is especially true when the archive medium is tape and the required data could be distributed over many different tape reels. These types of data transfer are always initiated by setting a flag.

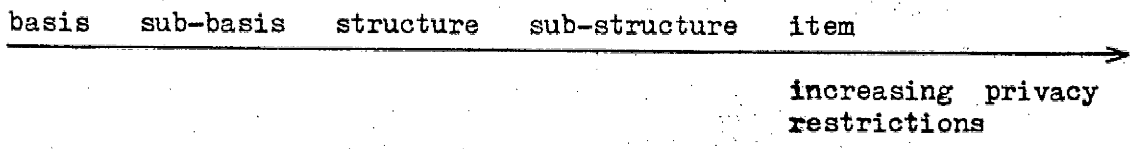
4.1.3 The Basis Level

A PATCOSY data base is defined in terms of one or more bases. Every different record defined must belong to only one basis. At a lower level the sub-basis is a mechanism which allows record definitions to be re-combined. Further, it is possible for parts of a record definition to be omitted from the sub-basis redefinition, allowing a large degree of control over data access. Each manipulation carried out upon the data base either explicitly or implicitly states the basis or sub-basis which is to be used for the manipulation.

4.1.4 Privacy Specification

A recurring facet of PDL is privacy which provides a means of restricting access to certain data items, records and bases. Privacy can either be specified as an overall feature of the quantity being restricted or as a specific to a particular manipulation.

Privacy of a quantity can always be altered between the different levels of the data base.



Levels at which privacy can be altered for an item which is part of a structure

4.2 The PATCOSY Manipulation Language

Most of the commands in PML have the following format. A key word which indicates the type of manipulation to be performed; an indication of the type of record to be manipulated; some specifiers which uniquely identify one occurrence of the record or may identify many similar records; and finally an indication of the basis or sub-basis which is to be used throughout the operation. Some of the commands allow sequences of host language instructions to be incorporated into the command. This feature permits application programs to define 'in situ' the processing which is to be carried out upon the requested data. The sequence will be performed for each record which satisfies the specifiers.

The commands provided fall into three main categories; those which operate upon prime data, those which operate upon relationships, and those which allow the Data Base Administrator to interrogate the system to make sure that the data base is running efficiently.

4.2.1 The Prime Data Commands

4.2.1.1 The WRITE Command

The WRITE command allows new data to be added to the data base. Any type of record can be added but the majority will be structures and lists. A new record will be positioned according to the value of the key item which is used in a direct or algorithmic relationship. If more than one such relationship is defined the first so defined will be used.

4.2.1.2 The READ Command

The READ command allows the application program to read data from the structures, sub-structures and lists within the data base. A particular READ command is associated with only one list, structure or sub-structure.

One form of the READ command allows lists to be used for operator and technician tasking. In this form the READ starts from the point where the last such READ finished. Thus if a technician is to be given three jobs at the same time the READ will obtain the next three elements from the list and a record will be kept of the point reached.

One alternative of READ allows relationships which may cross structure/sub-structure boundaries, to be identified for processing.

4.2.1.3 The DELETE Command

The DELETE command allows one record to be deleted from the data base at one time.

4.2.1.4 The ALTER Command

The ALTER command allows a record, which has been previously written, to be changed. Only one record can be ALTERed at a time. However, many items of the record may be changed at the same time. The format of the command requires specification of the value of the item which is to be changed as well as the value to which it is to be changed. This mechanism provides a means of safeguarding what has already been written.

If, instead of ALTERing a record, a particular application needs to keep adding similarly defined data to a record, the WRITE command can be used. The data to be added is defined as a sub-structure.

The related structure is the record to which data is repeatedly added. In the structure a variable repeating group is defined, the items of which are those defined as the sub-structure. When the WRITE command operates upon the sub-structure another occurrence of the variable repeating group is added to the record. If the sub-structure is used to READ from the record, only one occurrence of the repeating group will be released at one time. The whole of the repeating group can be made available either by READing the structure or defining a sub-structure which contains the identifier of the repeating group.

This facility is required to allow patient records to be built up in a form which was defined by Weed (10). Weed describes a 'problem-orientated record' to which information is repeatedly added after the initial consultation.

The doctor states what is wrong with the patient as a series of headings or problems. He also describes what treatment he is prescribing for each problem. If the treatment cures the problem the doctor can then state precisely what was wrong with the patient, which in due course will lead to more accurate illness types statistics. If the problem was not alleviated another course of treatment can be prescribed, possibly after redefining the problem.

4.2.1.5 The SET Command

The SET command is used to initiate the operation of a flag. This causes data to be moved from one storage device to another.

4.2.1.6 The FIND Command

The FIND command is similar to the READ command except that it has been extended in power. It allows logical combinations of specifiers to be formed (the READ command logically 'ands' all the specifiers together), more than one key index value to be specified, and the records found can be processed as found or saved for subsequent processing. These extensions mean that any program using FIND proceeds more slowly compared with one using READ even if the effect of each program, in a particular case, is the same.

4.2.1.7 The SORT Command

The SORT command allows a sequence of records to be ordered upon any item contained in the record. The SORTed order can either be saved for subsequent processing or the records can be processed as they are SORTed. Strings of records, relationships and files which contain information which has been saved can all be SORTed.

4.2.1.8 The DESTROY Command

The DESTROY command deletes any information which has been saved by the FIND or SORT commands, or by the commands which operate upon relationships.

4.2.2 The Relationship Commands

4.2.2.1 The ADD Command

The ADD command allows a record, which is currently being processed or which already exists, to be included in a particular relationship. The relationship must be a GROUP or a CHAIN. These are the only relationships which cannot be completed when a record is included in the data base. LINK, DOUBLE-LINK and TREE all join records of a similar type, and are dependent for their logical order upon the value of an item contained within the record. GROUP and CHAIN can join dissimilar records and therefore have no item which can act as a key for the logical ordering of the relationship.

4.2.2.2 The ERASE Command

ERASE allows the current record to be removed from a GROUP or CHAIN relationship.

4.2.2.3 The GIVE Command

The GIVE command allows the user to step his way through a relationship, without retaining any trace of the records which have been processed. GIVE also makes available any records which have been saved by other commands.

4.2.2.4 The FOLLOW Command

FOLLOW allows a user to process relationships, at the same time maintaining a trace of the path FOLLOWed so that he can return to any previous point. FOLLOW commands can be nested to as great a depth as necessary to reflect the data structure which is being processed. Other commands can be issued from within the FOLLOW sequence, although no trace will be kept of where these commands led. The trace created by a FOLLOW command is implemented by a first in, last out stack. The RETURN and RETRACE commands, which can only be issued from within a FOLLOW sequence, are used for back-tracking.

4.2.2.5 The RETURN Command

In a nested sequence, one option of the FOLLOW command allows the user to specify that a particular FOLLOW command in the subsequent program sequence should be more fully traced. This is achieved by the user associating a name with the particular FOLLOW command. Whenever this particular command is encountered during the dynamic program sequence, the name specified by the user and the information about the appropriate record occurrence are entered on the stack. The RETURN command allows the user to jump to the most recent occurrence of that name in the stack.

4.2.2.6 The RETRACE Command

The RETRACE command allows the user to process his way backwards through the stack until a particular record is found. The stack is not destroyed so that it is possible for the user to RETURN to the place from which the RETRACE was issued because the RETRACE command employs the same naming mechanism as the FOLLOW command.

4.2.2.7 The CREATE Command

CREATE allows the initiation of new relationships which are equivalent to GROUPS. The CREATEd relationship is intended to be used for a short period of time only and not as a redefinition of the data base. Pointers needed to maintain the relationship are external to any records which may have been ADDed to the relationship. The relationship can be subsequently DESTROYed by the user.

4.2.2.8 MEMBERSHIP Commands

These commands allow a user to ascertain of which relationships a particular record is the start or a member. One form of the command yields all such relationships; the other yields information on specific relationships.

4.2.3 Commands for the Data Base Administrator

These commands are the EXHIBIT and MAKE commands. EXHIBIT makes available the value of certain system maintained locations. Depending upon these values the D.B.A. can MAKE changes to increase the efficiency or alter the definition of the data base.

4.3 Data Maps

A data map is a contiguous sequence of elements which contain information about the items, relationships, and storage control commands associated with a record definition. There is one data map for each record type which is defined in the data base.

The information held by an element which refers to an item is:-

(i) type of item

(ii) identifier of item

(iii) length of item

(iv) privacy information

The information held in an element referring to a relationship is:-

(i) type of relationship

(ii) relationship-identifier

(iii) length of associated pointer(s)

(iv) pointer mechanism employed

(v) key item identifier

(vi) privacy information

The information held in an element referring to a storage control command is:-

(i) storage device

(ii) number of days and/or flags which control data transfer

or

(iii) type of transfer from an archive medium

(iv) flags which will cause the transfer from archive to direct access medium to take place.

4.4 Examples of Data Structures which can be formed by PATCOSY

4.4.1 Involving Items Only

```
structure  structure-1  privacy  1 to 10  contains
fixed  item-1  length  4  privacy  2,4 to 8
fixed  item-2  length  8  computational
variable  item-3  privacy  write  1 to 5;
     read  2,5 to 10;  7 to 10
fixed  item-4  length  8
```

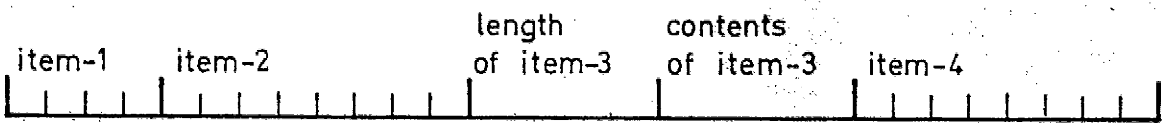


Fig 4.4 A Pictorial Representation of Structure-1

The associated data map is given in fig. 4.5.

The privacy rating of structure-1 restricts access to users who have a privacy coding between 1 and 10 inclusive.

Item-1 is four characters long, and access is restricted, for all commands, to users who have a privacy coding of 2 or 4 to 8.

Item-2 is eight characters in length and is held in a computational form. Access restrictions are the same as for the whole structure.

Item-3 is of variable length. Only users with a privacy rating of 1 to 5 may WRITE the item. Similarly only users with a privacy rating of 2 and 5 to 10 may READ the item. The privacy rating required for access by all other commands is 7 to 10.

Item-4 is a fixed length string of eight characters. As all syntactic forms of the privacy sentence are defined in this example, the privacy associated with subsequent examples will not be given.

4.4.2 Associate and Repeating Groups

```
structure structure-2 contains
fixed item-5 length 4
associate item-6 with t-item-1, t-item-4 of t-table
    for t-item-5 = item-7
fixed item-7 length 3
fixed item-8 length 2 (fixed item-9 length 2
                        variable item-10 (fixed item-11 length 1
                                          fixed item-12 length 2))
```

type	identifier	length	form	privacy			
				read	write	give	add
F	item-1	4	char	2,4 to 8	2,4 to 8	2,4 to 8	2,4 to 8
F	item-2	8	comp	1 to 10	1 to 10	1 to 10	1 to 10
V	item-3	-	char	2,5 to 10	1 to 5	7 to 10	7 to 10
F	item-4	8	char	1 to 10	1 to 10	1 to 10	1 to 10

Fig. 4.5 The Data Map Associated with Structure-1

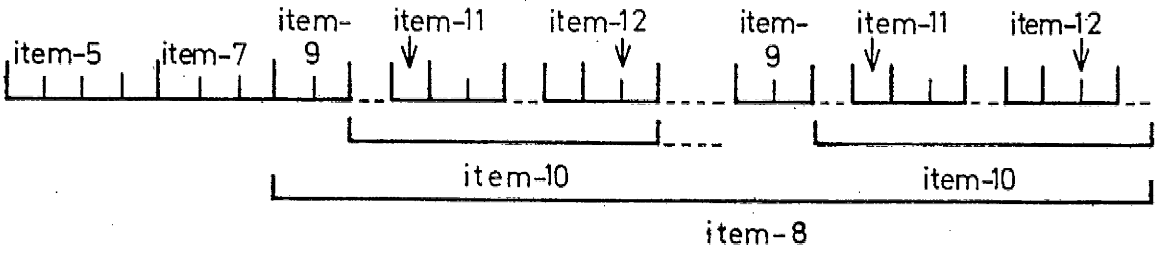


Fig 4.6 A Pictorial Representation of Structure-2

The associated data map is given in fig. 4.7.

Item-5 is a fixed length string of four characters.

Item-6 is an item which is associated with t-item-1 and t-item-4 from t-table. The required entry in t-table is found by using the value of item-7 for t-item-5.

Item-7 is a fixed length string of three characters.

Item-8 is a fixed length repeating group the items of which are repeated two times. The items of the repeating group are; the fixed length item-9 and the variable length repeating group item-10.

The items of the variable repeating group are the fixed length items 11 and 12.

Hence for each of the two occurrences of item-8 there is one occurrence of item-9 and an unknown number of occurrences of items 11 and 12.

4.4.3 Relationships

Using the example of structure-1 (4.4.1) the following relationships could be defined.

<u>double-link</u> d-link-1 <u>by</u> item-1 <u>ascending</u> <u>mechanism</u> <u>is</u> <u>direct</u>
<u>link</u> link-1 <u>by</u> item-2 <u>descending</u> <u>mechanism</u> <u>is</u> <u>displacement</u>

type	identifier	length	form	number	item-name	key-name	key-value	table-name	privacy	
									read	write
F	item-5	4	char							
A	item-6			2	t-item-1	t-item-5	item-7	t-table		
					t-item-4					
F	item-7	3	char							
FG	item-8	2								
FGE	item-9	2	char							
FGE	item-10									
VGE	item-11	1	char							
VGE	item-12	2	char							

fig. 4.7 The Data Map Associated with Structure-2

```
tree  a-tree  by  item-4  using  'ABCD1234'  as initial node
    mechanism is direct.
group  group-1  mechanism is direct  privacy  5;
    group-2  mechanism is displacement  privacy  4,6 to 10
member-chain  chain-1  mechanism is direct  privacy  erase  5
start-chain  chain-2  mechanism is direct
owned  by  index-1
```

The associated data map is given in fig. 4.8.

D-link-1 is a relationship which has pointers to both the next record occurrence and the prior record occurrence in the string of record occurrences. The next pointer of the relationship is ordered by ascending or increasing values of item-1. The direct pointer to mechanism is to be used.

Link-1 is a relationship which only has pointers to the next record of the string of occurrences. Record occurrences are ordered by taking the descending or decreasing values of item-2. The relationship is implemented by displacement pointers.

A-tree is a binary tree the key item of which is item-4, the value of the initial node is specified as a literal 'ABCD1234'. The pointers in the tree will be implemented by the direct mechanism. Note that the initial node value will always be as specified even though the record containing that value may not yet exist.

All the above relationships do not contain a privacy

type	identifier	length	form	key-name	tree-node	privacy		
						read	erase	add
DL	d-link-1		A/D	item-1		2,4 to 8	2,4 to 8	2,4 to 8
L	link-1		DE/DI	item-2		1 to 10	1 to 10	1 to 10
T	a-tree		D	item-4	ABCD1234	1 to 10	1 to 10	1 to 10
G	group-1		D			5	5	5
G	group-2		DI			4,6 to 10	4,6 to 10	4,6 to 10
MC	chain-1		D			1 to 10	5	1 to 10
SC	chain-2		D			1 to 10	1 to 10	1 to 10
O	index-1					1 to 10	1 to 10	1 to 10

fig. 4.8 The Data Map Associated with the Relationships

sentence, as the privacy of the relationship is defined by the privacy of the item which is used as the key-item of the relationship. In the following secondary relationships, where there is no dependence upon any item of the record, the privacy sentence can be specified, thus restricting access to the authorised users.

The record occurrence can be a member of both groups 1 and 2. The pointers associated with group-1 are implemented by the direct mechanism, those of group-2 by the displacement mechanism. Access to group-1 is restricted to those users who have a privacy rating of 5. For group-2 access is restricted to those users who have a privacy rating of 4, and 6 to 10.

The record occurrence can be a member of chain-1, which is implemented using the direct mechanism. For all commands other than erase the privacy restrictions are the same as for the structure as a whole. However, to erase the record from an occurrence of chain-1 the user must have a privacy rating of 5. It is possible for the record occurrence to be a member of more than one occurrence of chain-1, that is, there could be many chain-1's each of which is differentiated from the other in that each will be started in a different record. Chain-2 is an example of such a record occurrence as it is the start of a chain which employs the direct mechanism. It is impossible for the record occurrence to be the start of more than one chain of the same name.

The final, owned by, relationship indicates that a record of the index key value which gave rise to this record occurrence is to be maintained in the record occurrence. Note that in the data map, given in the fig. 4.8, the length of each relationship is not defined. The lengths of each of the different pointers will be calculated by the system when the data base is defined from information given in the system section of the definition program (see appendix 1).

4.4.4 Storage Commands

```
kept for 10 days on drum then
kept until flag-1 on fixed-disc then
kept for 7 days or until flag-2 on exchangeable-disc then
kept forever on tape
on-lined when flag-3 to index-1, flag-4 to index-2
read when flag-5 return immediately,
    flag-6 return after 6 days
```

The associated data map is given in fig. 4.9.

The above sequence of storage commands acts upon structure-1 (see 4.4.1).

Initially an occurrence of the record, structure-1, is kept for ten days on a drum. It is then transferred to a fixed-disc for an indeterminate period which is ended by

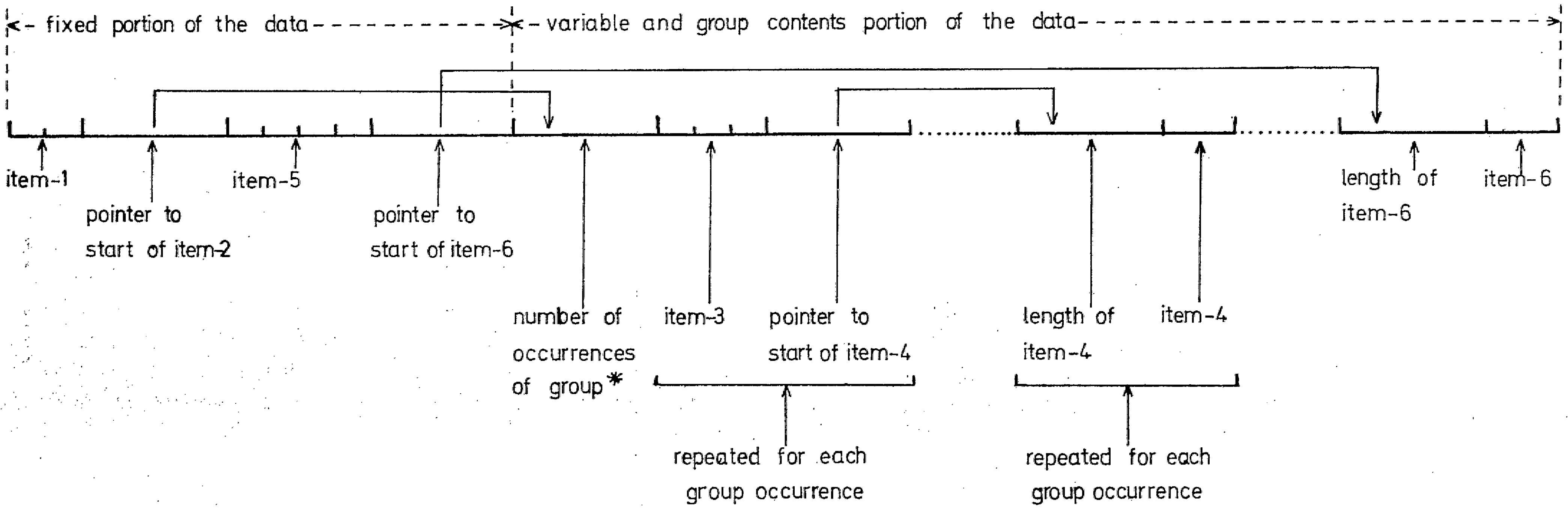
Definition

fixed item-1 length 2

variable item-2 (fixed item-3 length 3
variable item-4)

fixed item-5 length 4

variable item-6


fixed portion of the data
variable and group contents portion of the data
item-1
pointer to start of item-2
item-5
pointer to start of item-6
number of occurrences of group*
item-3
pointer to start of item-4
length of item-4
item-4
length of item-6
item-6
repeated for each group occurrence
repeated for each group occurrence

*not needed if group is fixed as number of occurrences is held in data-map

Fig5.1 To Show the Implementation of Groups

type	device	length	number	flag-name	index-name
KF	drum	10			
KU	fixed-disc		1	flag-1	
KFU	e-disc	7	1	flag-2	
E	tape				
OL			2	flag-3	index-1
				flag-4	index-2
R		0	2	flag-5	
		6		flag-6	

fig. 4.9 The Data Map Associated with the Storage Commands

flag-1 being SET. After which the record is kept on exchangeable-disc for seven days or until flag-2 is SET, whichever is the sooner. The record is then transferred to tape forever.

If flag-3 is SET, appropriate record occurrences will be transferred to drum, the first defined in the kept sequence, and will be accessible from index-1. Similarly if flag-4 is SET the record will be transferred to drum and will be accessible from index-2. When data is transferred by the on-lined command it always resumes the sequence of storage controls starting from the first. If an index had not been specified then the data would have been accessible from the only index which has been defined as the master index.

When flag-5 or 6 are SET a copy of the record will be taken and placed on a direct access device. The copy will either be destroyed immediately as in the case of flag-5, or it will be destroyed after 6 days.

Fig. 4.10 gives an indication of the complete data map which would be maintained for structure-1.

Appendix 3 contains a fuller example of a data definition program and the data maps which would be produced.

type	identifier	length	form	number	item-name flag-name	key-name	key-value	table-name	privacy read	write	add	erase
S	structure-1								1 to 10	1 to 10	1 to 10	1 to 10
KF	drum	10										
KU	fixed-disc			1	flag-1							
KFU	e-disc	7		1	flag-2							
E	tape											
OL				2	flag-3	index-1						
					flag-4	index-2						
R		0		2	flag-5							
					flag-6							
DL	d-link-1		A/D		item-1				2,4 to 8	2, 4 to 8	2,4 to 8	2,4 to 8
L	link-1		DE/DI		item-2				1 to 10	1 to 10	1 to 10	1 to 10
T	a-tree		D		item-4		ABCD1234		1 to 10	1 to 10	1 to 10	1 to 10
G	group-1		D						5	5	5	5
G	group-2		DI						4,6 to 10	4,6 to 10	4,6 to 10	4,6 to 10
MC	chain-1		D						1 to 10	1 to 10	1 to 10	5
SC	chain-2		D						1 to 10	1 to 10	1 to 10	1 to 10
O	index-1											
F	item-1	4	char						2,4 to 8	2,4 to 8	2,4 to 8	2,4 to 8
F	item-2	8	comp						1 to 10	1 to 10	1 to 10	1 to 10
V	item-3		char						2,5 to 10	1 to 5	7 to 10	7 to 10
F	item-4	8	char						1 to 10	1 to 10	1 to 10	1 to 10

fig. 4.10 The Complete Data Map for Structure-1

CHAPTER 5

AN IMPLEMENTATION OF PATCOSY

This chapter describes how PATCOSY could be implemented. Initially the construction of the data maps is considered and this leads to a discussion of how the PML features could be implemented.

Finally a description of a pilot scheme is presented which employs the implementation methods already described.

5.1. An Implementation of the PDL

The data base administrator defines a data base by writing a sequence of PDL commands. These commands are then processed by running the sequence as a data definition program, the output from which is a sequence of discrete data maps. One data map is produced for each record which has been defined. Therefore a discussion of an implementation of the PDL devolves into a description of the way in which constructs of the PDL are converted into data maps.

In the implementation described, a data map contains five fields each of fixed length as shown below. An entry in the data map is therefore called a quintuple.

type	identifier	length	position	number

Fig. 5.1 The Fields of a Data Map Quintuple.

In the following discussion all lengths of items and positions of fields within records are given as a number of characters or bytes as this is the only standard which will allow transportability.

5.1.1 The Item Quintuples

For each item quintuple there are two forms; the first is used when the item is part of the whole record, the second when the item is part of a repeating group.

If the item is not part of a repeating group the position field in the quintuple gives the location of the item relative to the start of the record. If the item is part of a repeating group the position field indicates the start of the item relative to the start of the repeating group which contains the item.

To take account of the different quintuples the type field assumes the positive integers in sequence indicating the particular quintuple. The notation 'identifier-5' means the identifier field which belongs to a type 5 quintuple.

5.1.1.1 Fixed Length Items (Types 1 to 4)

Types 1 and 3 refer to fixed length items held as strings; types 2 and 4 to computational numeric items. Types 2 and 4 may only be used when the item is contained in a repeating group.

The length field contains the number of characters which the item occupies. The number field indicates how many privacy quintuples follow the defining quintuple. The identifier field holds the name by which the item will be referenced.

In an implementation the length of computational fields may be restricted to certain prescribed values due to the nature of the floating-point arithmetic provided.

5.1.1.2 Variable Length Items (types 5 and 6)

The identifier field holds the name by which the item will be referenced. The length field is not used. The position field indicates the start of a pointer which is internal to the record. The pointer contains the relative location of another field in the record, which contains the length of the item and the contents of the item (see fig.5.1). Using this construction the start of each item is well defined although one or more pointers may be required to obtain the data itself. Type 6 is used when the item is part of a repeating group.

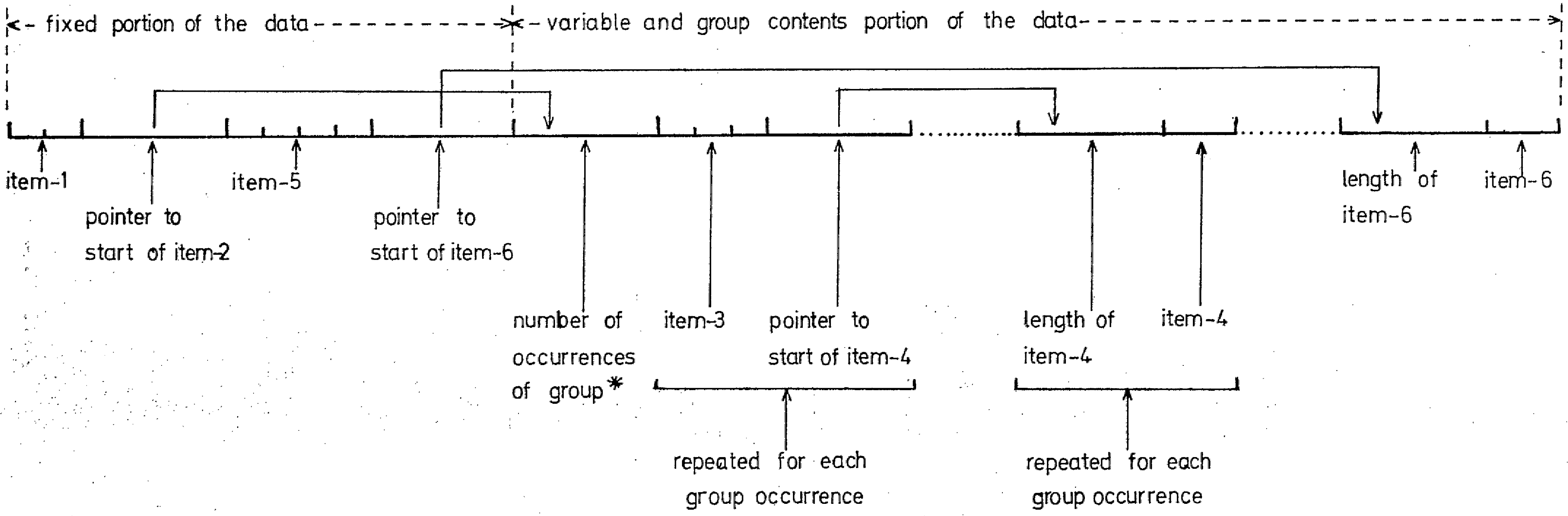
5.1.1.3 Pointer Items (types 7 and 8)

Pointers can only occur in indices, pointer-lists and pointer-arrays and cannot be contained in repeating groups.

Identifier-7 gives the name by which the pointer will

Definition

fixed item-1 length 2
variable item-2 (fixed item-3 length 3
variable item-4)
fixed item-5 length 4
variable item-6



fixed portion of the data
variable and group contents portion of the data
item-1
pointer to start of item-2
item-5
pointer to start of item-6
number of occurrences of group*
item-3
pointer to start of item-4
repeated for each group occurrence
length of item-4
item-4
repeated for each group occurrence
length of item-6
item-6

*not needed if group is fixed as number of occurrences is held in data-map

Fig 5.1 To Show the Implementation of Groups

be referenced.

Length-7 gives the length of the pointer; this is dependent upon the operating environment and the pointer mechanism and is therefore defined internally during the data definition program.

Position-7 gives the position of the pointer relative to the start of the record.

Number-7 indicates how many privacy quintuples follow the type 8 quintuple.

Identifier-8 gives the identifier of the object pointed to.

Length-8 will contain the location of the data map of the object pointed to.

Position-8 and Number-8 are not used.

5.1.1.4 Mechanism Quintuple (type 9)

The mechanism quintuple indicates whether the preceding pointer is implemented by the direct, displacement or algorithmic mechanism.

The type 9 quintuple follows either a type 8 quintuple or any privacy quintuples.

Identifier-9 indicates direct displacement or the name of an algorithm.

In the latter case Length-9 indicates the location of the algorithm. Otherwise all other fields are not used.

5.1.1.5 Repeating Group Items (types 10 to 13)

Types 10 and 12 refer to fixed length repeating groups. Types 11 and 13 refer to variable length repeating groups. Types 12 and 13 refer to repeating groups which are items of other repeating groups.

In all cases the identifier indicates the name by which the group as a whole will be referenced.

Lengths 10 and 12 indicate how many repetitions of the group there are.

Lengths 11 and 13 are not used.

The position fields indicate the location of a field which contains an internal pointer to the actual data similar to variable items (see fig. 5.1.)

The number fields indicate how many quintuples have to be skipped before an item is defined at the same level. Each time a repeating group is defined as an item of another repeating group the level is increased by one until the end of that repeating group definition.

5.1.1.6 The Associate Item (types 14 to 23)

Types 14 to 18 are used when the item is not part of a repeating group.

Types 19 to 23 are used when the item is part of a repeating group.

Identifier-14 indicates the name by which the item will be referenced.

Length-14 indicates how many type 15 quintuples follow.

Position-14 indicates the starting location of the item within the record. In fact the item, as such, occupies no space, and so the following item will have the same position.

Number-14 indicates the number of privacy quintuples which immediately follow the type 14 quintuple. If present the privacy quintuples are then followed by the type 15 quintuples.

Identifier-15 indicates the item of the table or dictionary which is to be associated with the item as a whole.

The remaining type 15 fields are not used.

Identifier-16 identifies the table or dictionary which is to be used.

Number-16 indicates the location of the data map of that table or dictionary.

Length-16 and Position-16 are not used.

Identifier-17 indicates which of the table or dictionary items is to be used as the key item.

The remaining type 17 fields are not used.

Identifier-18 indicates the item within the record which contains the value of the key item of the table or dictionary.

Number-18 indicates the relative location of the definition of the item in the data map.

Length-18 and Position-18 are not used.

5.1.1.7 Privacy Quintuples (types 24 to 43)

Many different privacy quintuples are required to cope with a privacy specification relating to all commands, privacy derived from sub-basis definitions, and privacy ratings relevant to one command only. (For further explanation see Appendix 1.5.2.).

It is presumed that the identifier field will be sufficently long to be split into two fields, thus yielding four fields which can be used to hold privacy codings. The other identifier field will indicate whether the four fields hold four discrete codings or two pairs giving two ranges of privacy codings.

The exception to this rule is in privacy quintuples following repeating group quintuples (types 10 to 13) because these quintuples have no field indicating how many privacy quintuples follow. In this case, if there are any privacy quintuples, the number field of the first will indicate how many more privacy quintuples follow.

In the preceding discussion no assumptions have been made about the identifier field. If a maximum length identifier is implemented then the identifier field can be of fixed length within the quintuples. If variable length identifiers are allowed the identifier field will have to be a pointer which points to the actual identifier.

5.1.2 The Relational Quintuples

5.1.2.1 Link and Double-link Relation (types 44 to 46)

Type 44 indicates a link and type 45 a double-link relation.

Identifiers 44 and 45 give the name by which the relation will be referenced.

Lengths 44 and 45 give the number of characters which the relation occupies. This length is caloulated by the system.

Positions 44 and 45 give the starting location of the pointer relative to the start of the record.

Numbers 44 and 45 are set to '2' indicating that two more quintuples (types 46 and 9) follow in the sequence relating to these relationships.

A type 44 or 45 quintuple is then followed by a type 46 quintuple and indicates the key item of the relationship.

Identifier - 46 holds the name of the key item.

Length - 46 indicates whether the relation is 'ascending' or 'descending'.

Position-46 is not used.

Number-46 indicates the relative location of the definition of the key item in the data map.

A type 46 quintuple is always followed by a mechanism (type 9) quintuple.

5.1.2.2 The Tree Relation (types 47 to 49)

Type 47 gives general information about the relation.

Identifier-47 gives the name by which the relation is referenced.

Length-47 indicates the number of characters used by one of the relation's pointers.

Position-47 gives the starting location of the relation relative to the start of the record.

Number-47 is set to '3' indicating that three more quintuples (types 48, 49 and 9) follow in the tree relation sequence.

The type 48 quintuple gives details about the key item of the relation.

Identifier-48 gives the name by which the key item is referenced.

Number-48 gives the relative location of the definition of the key item in the data map.

Length-48 and position-48 are not used.

The type 49 quintuple indicates the initial node of the relation.

If the value of the initial node is short enough then the value will be contained in identifier-49, otherwise identifier-49 will contain a pointer to the actual value. The type 49 quintuple is always followed by a mechanism (type 9) quintuple.

5.1.2.3 Group and Chain Relations (types 50 to 52)

A type 50 quintuple indicates that the record can be a member of the indicated group. A type 51 quintuple gives the name of a chain of which the record can be a member. A type 52 quintuple gives the name of a chain of which the record is the start.

Identifiers 50 to 52 give the name of the relation used.

Lengths 50 to 52 give the length of the pointer used to form the relation.

Positions 50 and 51 give the location of a field which points to a string of pointers. It is possible for one record to be a member of the same group or chain more than once.

Position-52 gives the starting location of the relation relative to the start of the record.

Numbers 50 to 52 give the number of privacy quintuples which follow the defining quintuple.

The privacy quintuples are then followed by a mechanism (type 9) quintuple.

5.1.2.4 Accessed by Relation (type 53)

Identifier-53 gives the key item of the record which is to be used in the relation.

Length-53 indicates the order in which the key item and its associated record are to be maintained ('ascending' or 'descending').

Position-53 indicates the relative location of the definition of the key item in the data map.

Number-53 is '1' when the accessed by is used in a table index or dictionary to indicate the order of records, in which case the type 53 quintuple is followed by a mechanism (type 9) quintuple. Number-53 is '0' if the relation is used to signify the order of elements of a repeating group contained in the record. In this case the key item must be defined in a repeating group. No other quintuple is required to form the relationship definition.

5.1.2.5 The Frequency Relation (types 54 and 55)

Identifier-54 contains the number of days which are to elapse before the first re-organisation.

Length-54 contains the number of days which are to elapse between subsequent re-organisations.

Position-54 contains the location of the counter field relative to the start of the record.

Number-54 indicates how many type 55 quintuples follow.

A type 55 quintuple is used to indicate each different level of the dictionary and how many records are to be held at that level.

Identifier-55 is not used.

Length-55 gives the number at the particular level.

Position-55 gives the level number.

5.1.3. Storage Command Quintuples

5.1.3.1. The KEPT Command (types 56 to 61)

Type 56 is used when KEPT FOREVER is specified. Type 57 indicates KEPT FOR a number of DAYS. Type 58 indicates KEPT UNTIL a flag is set.

Identifiers 56 to 58 indicate the device type, upon which the data will be kept.

The remaining type 56 fields are not used.

Length-57 indicates for how many days the data will be kept upon the indicated device.

Position-57 is not used.

Number-57's use is explained later.

Length and position 58 are not used.

Number-58 indicates how many type 59 quintuples follow.

Type 59 indicates the flag which when set will cause data movement.

Identifier-59 gives the name of a flag.

The remaining type 59 fields are not used.

If number-57 is zero no type 59 quintuples follow, otherwise number-57 indicates how many type 59 quintuples follow. This provides the KEPT FOR a number of DAYS OR UNTIL a flag is set option.

Type 60 is used to indicate that the data is to be DESTROYED.

None of the other type 60 quintuple fields are used.

Type 61 is used only when FREQUENCY is specified for a dictionary.

Identifier-61 indicates the device type.

Length-61 indicates the level number.

Position-61 indicates the relative location in the data map of the definition of that level.

Number-61, in the first type 61 quintuple, indicates how many more type 61 quintuples follow, otherwise it is not used.

5.1.3.2. The ON-LINED Command (types 62 and 63)

Identifier-62 contains the name of a flag which when set will cause the data movement.

Length and position-62 are not used.

Number-62 if '1' indicates a type 63 quintuple follows, otherwise it will be '0'.

Identifier-63 is the identifier of an index from which the data will be accessible.

5.1.3.3. The READ Command (types 64 and 65)

Identifier-64 contains the name of a flag which when set will cause a copy of the data to be taken.

Length-64 indicates how many days the copy is to be maintained. If it is '0', it implies the copy will be destroyed immediately after the process has been completed.

Position-64 is not used.

Number-64, if '1', indicates a type 65 quintuple follows.

Identifier-65 indicates the index from which the copy will be available if length-64 is greater than '0'.

5.1.4. Heading Quintuples (types 66 to 74 and 75)

Types 66 to 74 are used to identify the record at the start of each data map. One quintuple is used for each type of record; structure table, dictionary, list, etc.

Identifiers 66 to 74 hold the name by which the record will be referenced.

Lengths 66 to 74 hold the number of storage command and relational quintuples in the data map.

Positions 66 to 74 hold the number of item quintuples in the data map.

Numbers 66 to 74 hold the length of the fixed portion of the record.

Type 75 is used to provide a cross-reference between a record and its data map. The cross-reference system used is; as each data map is defined it is assigned a number. If a definition is changed then the redefined data map takes the next number in sequence, so that the records written with the original data map are still accessible.

Identifer-75 is not used.

Length-75 holds the data map number.

Position-75 holds the location of the most previous data map which is identified by the same identifier. Thus it is possible to find quickly previous definitions of the same record.

Number-75 indicates how many privacy and sub-basis quintuples follow.

Privacy quintuple 24 is used to indicate the sub-bases to which the record belongs. Instead of recording the identifier of a sub-basis, each sub-basis is allotted a code so that recording space is not wasted. If a type 24 occurs at the item level it means that the item is not part of the sub-basis.

In a large system the number of data maps will be many, and therefore to promote easy access an index will have to be maintained of record identifiers and the location of the appropriate data maps. Similarly an index of algorithms may have to be set up. In all places where such a location has been specified the reference has been to the index entry and not to the actual location of the data map or algorithm.

5.2 Implementation of PML

The basic building blocks of PML are routines which search for a particular quintuple and are then able to 'read' or 'write' the data associated with the quintuple, regardless of the type of quintuple. For example, routines will be provided which can 'read' and 'write' fixed items, the pointers associated with the tree relationship and the details associated with storage commands. Every manipulation command can be broken down into a sequence of 'reads' and 'writes' joined together by comparisons and loops.

A command has to be dissected into its constituent parts before the manipulation can proceed. This process is shown in fig. 5.2.1. When the data map is found the class of record (structure, list, sub-structure etc.) can be established so that any requirements for the different types can be checked. For example many of the types will require the key item of an index to be given a value.

The main difference between the operation of commands is whether or not the record is a structure or sub-structure. If it is a structure or sub-structure an extra sequence of instructions will have to be performed which access the index pointer-list and possibly the pointer-array associated with the structure or sub-structure. This sequence is described by fig. 5.2.2. The first requirement of the system is that it should keep a directory of frequently used identifiers and the location of the data map. This is especially true of indices where all access will be through the key-item rather than the name of the index. The manner in which the index will be searched is defined by the particular implementor. In the implementation described in 5.3 a 'binary chop' mechanism was used, in which the basic unit was 128 entries. The only further complication is when a pointer-array is used, as this entails searching through the pointer-array elements until the required key-item value is found.

When other classes of record such as list, table and dictionary are accessed the starting point of the required

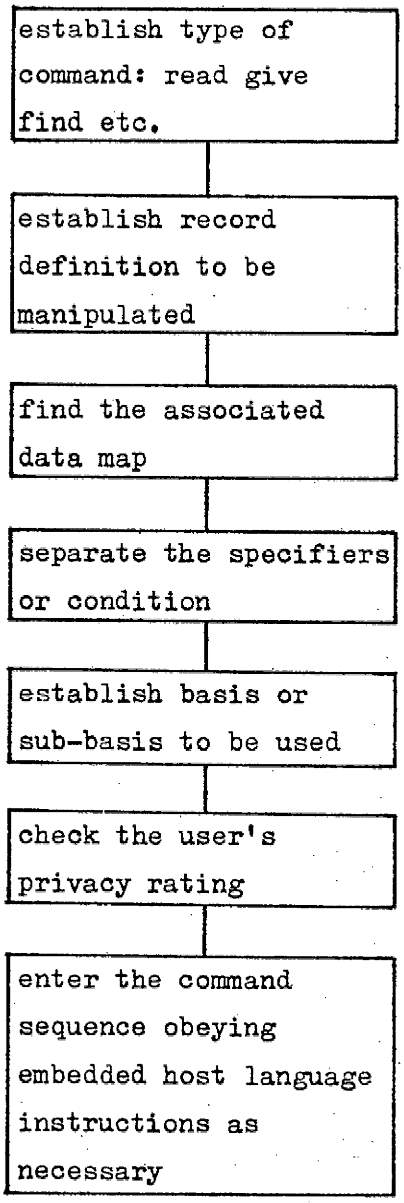

establish type of command: read give find etc.
establish record definition to be manipulated
find the associated data map
separate the specifiers or condition
establish basis or sub-basis to be used
check the user's privacy rating
enter the command sequence obeying embedded host language instructions as necessary

Fig. 5.2.1.

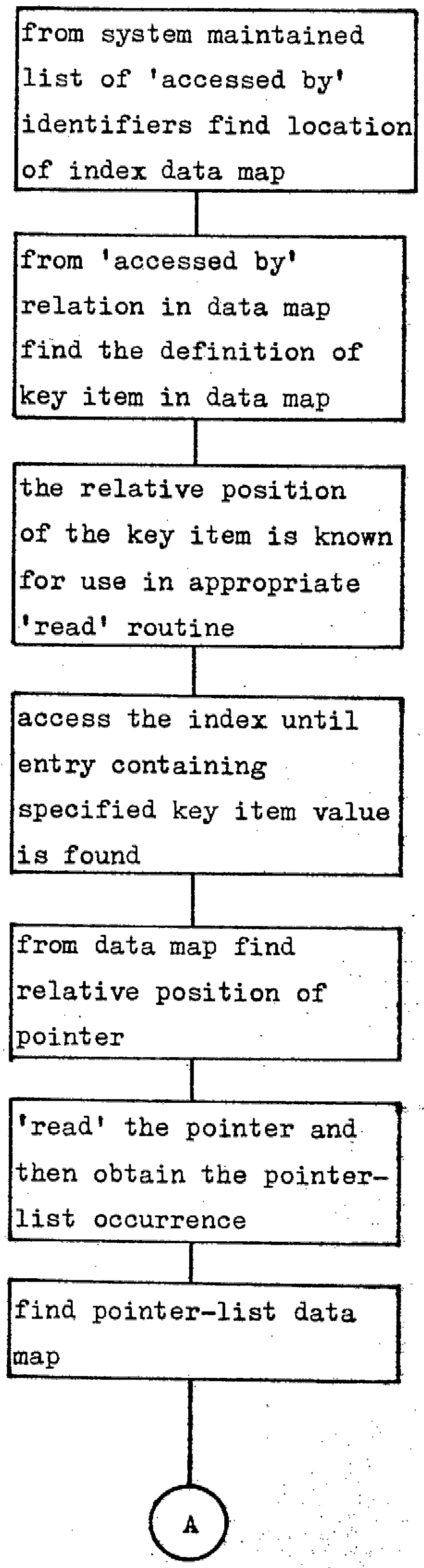

from system maintained list of 'accessed by' identifiers find location of index data map
from 'accessed by' relation in data map find the definition of key item in data map
the relative position of the key item is known for use in appropriate 'read' routine
access the index until entry containing specified key item value is found
from data map find relative position of pointer
'read' the pointer and then obtain the pointer-list occurrence
find pointer-list data map
A

Fig. 5.2.2.

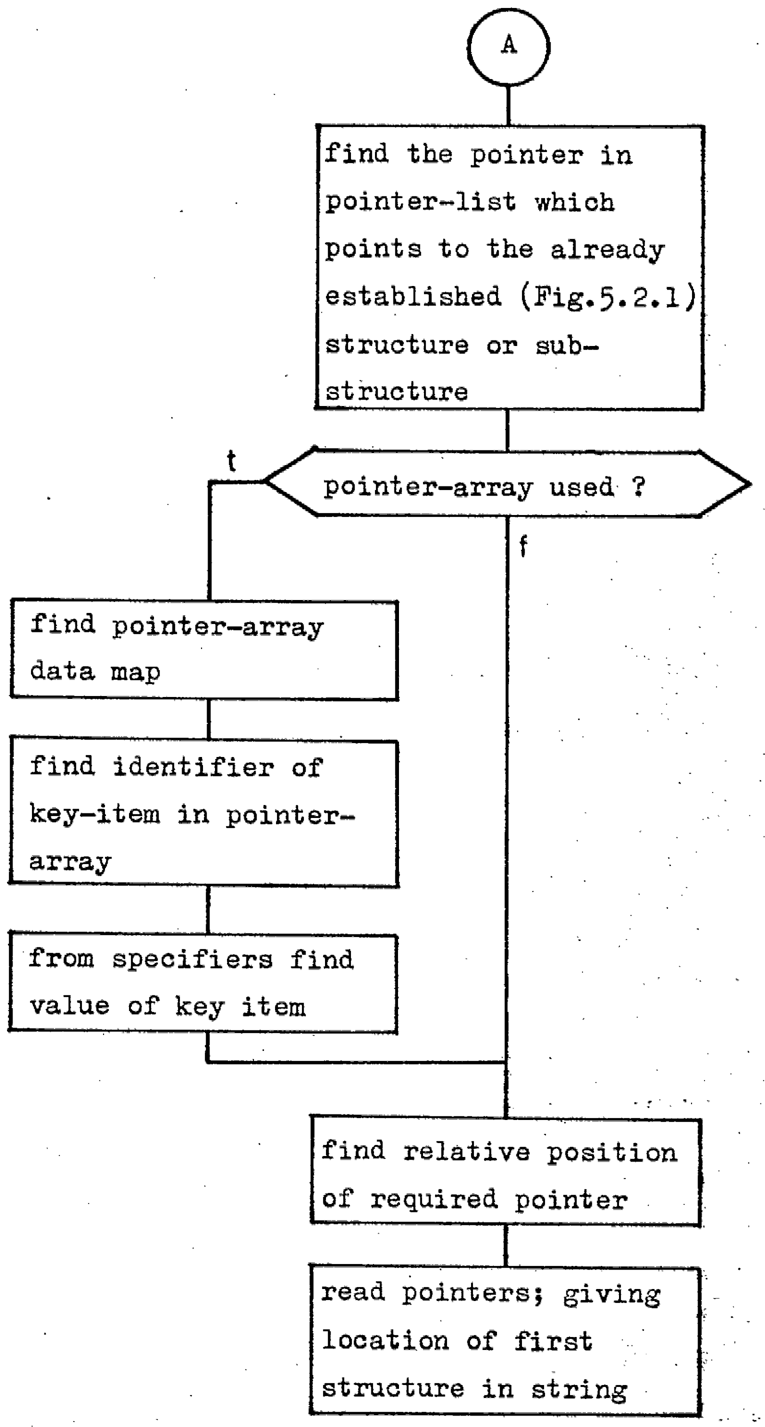

A
find the pointer in pointer-list which points to the already established (Fig.5.2.1) structure or sub-structure
pointer-array used ?
t
f
find pointer-array data map
find identifier of key-item in pointer-array
from specifiers find value of key item
find relative position of required pointer
read pointers; giving location of first structure in string

Fig. 5.2.2. cont.

records is always maintained by the system and so the search described above is not needed. The above procedure will have to be gone through each time a structure belonging to a different index key value is required. This is a necessary consideration when displacement mechanism pointers are used to join structure occurrences together.

The remainder of this section discusses the WRITE, READ, ADD, FOLLOW, SET and ALTER commands. Other commands are either extensions or the reverse of the process described. For example, DELETE is the reverse of WRITE, ERASE is the reverse of ADD, FIND is an extension of READ. Finally the GIVE command is a simpler kind of FOLLOW.

5.2.1 The WRITE Command (see fig. 5.2.3.)

The physical location of the record is determined by the operating system under which the data base system operates and is a function of what the implementor requires and what is provided by the data base. After the actual storage has been allocated to the record occurrence the logical position of the record in a primary relationship is dictated by the value of a key item field associated with the particular primary relationship. Previously written records of the same relationship are accessed until the correct logical position is found. This is carried out by successively 'reading' the value of the key item field in each record occurrence of the relationship.

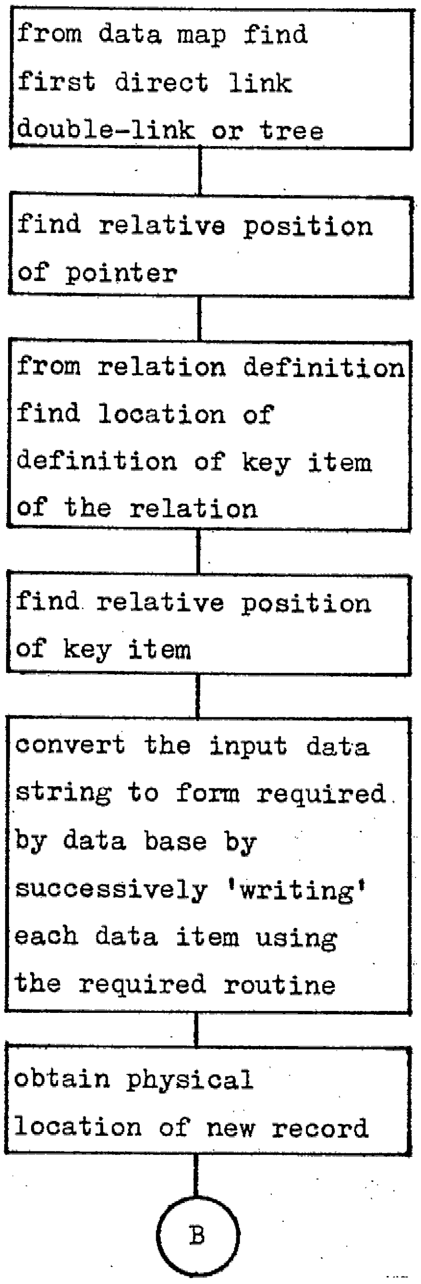

from data map find first direct link double-link or tree
find relative position of pointer
from relation definition find location of definition of key item of the relation
find relative position of key item
convert the input data string to form required by data base by successively 'writing' each data item using the required routine
obtain physical location of new record
B

Fig. 5.2.3.

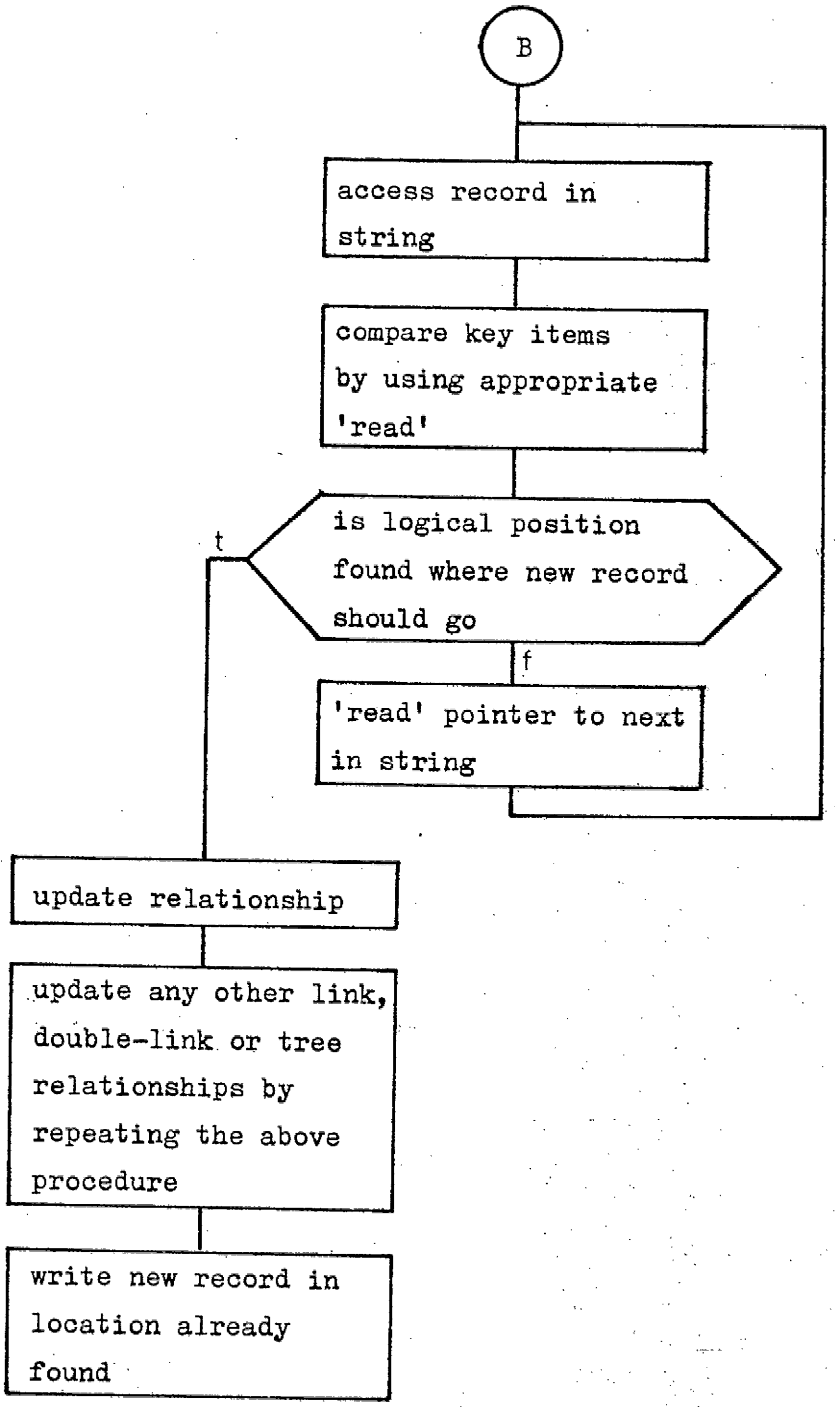

B
access record in string
compare key items by using appropriate 'read'
is logical position found where new record should go
t
f
'read' pointer to next in string
update relationship
update any other link, double-link or tree relationships by repeating the above procedure
write new record in location already found

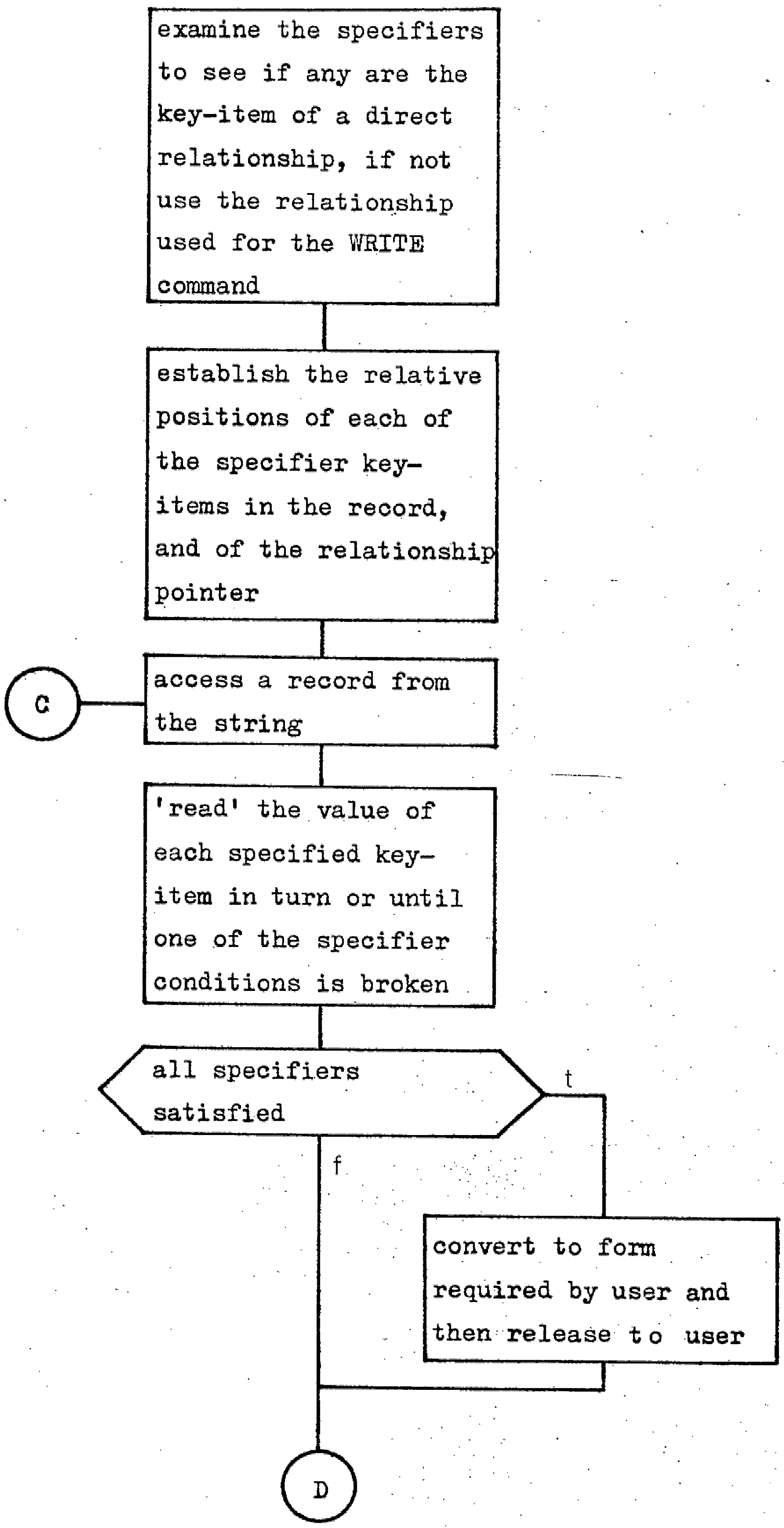
Fig. 5.2.3. cont.

The input data string associated with the WRITE command is converted to the form required for storing in the data base. This is effected by taking each item quintuple in turn and by applying the appropriate 'write' routine to the character string. In the case of variable and repeating group items the current end of the record has to be maintained so that the data base form of the data can be built up. The initial position of this marker is the end of the fixed portion of the record.

The final consideration of the WRITE command is when it is applied to a sub-structure which can be constructed from parts of one or more structures. Hence, if, when a sub-structure is written, a check is not made to ascertain whether any previously written structures could contain the sub-structure items relating to that structure, a great deal of storage space could be wasted. Obviously when data is added to an already existing structure great care must be taken to ensure that all common items have identical values.

5.2.2 The READ Command (see fig. 5.2.4.)

Initially, so that processing time can be reduced, it is advisable to check that none of the key items of the specifiers are also the key items of a direct link, double-link, or tree relationship. If this is the case then this relationship can be used to process the structure string so that processing can stop as soon as a value, outside the



examine the specifiers to see if any are the key-item of a direct relationship, if not use the relationship used for the WRITE command
establish the relative positions of each of the specifier key-items in the record, and of the relationship pointer
C
access a record from the string
'read' the value of each specified key-item in turn or until one of the specifier conditions is broken
all specifiers satisfied
t
f
convert to form required by user and then release to user
D

Fig. 5.2.4.

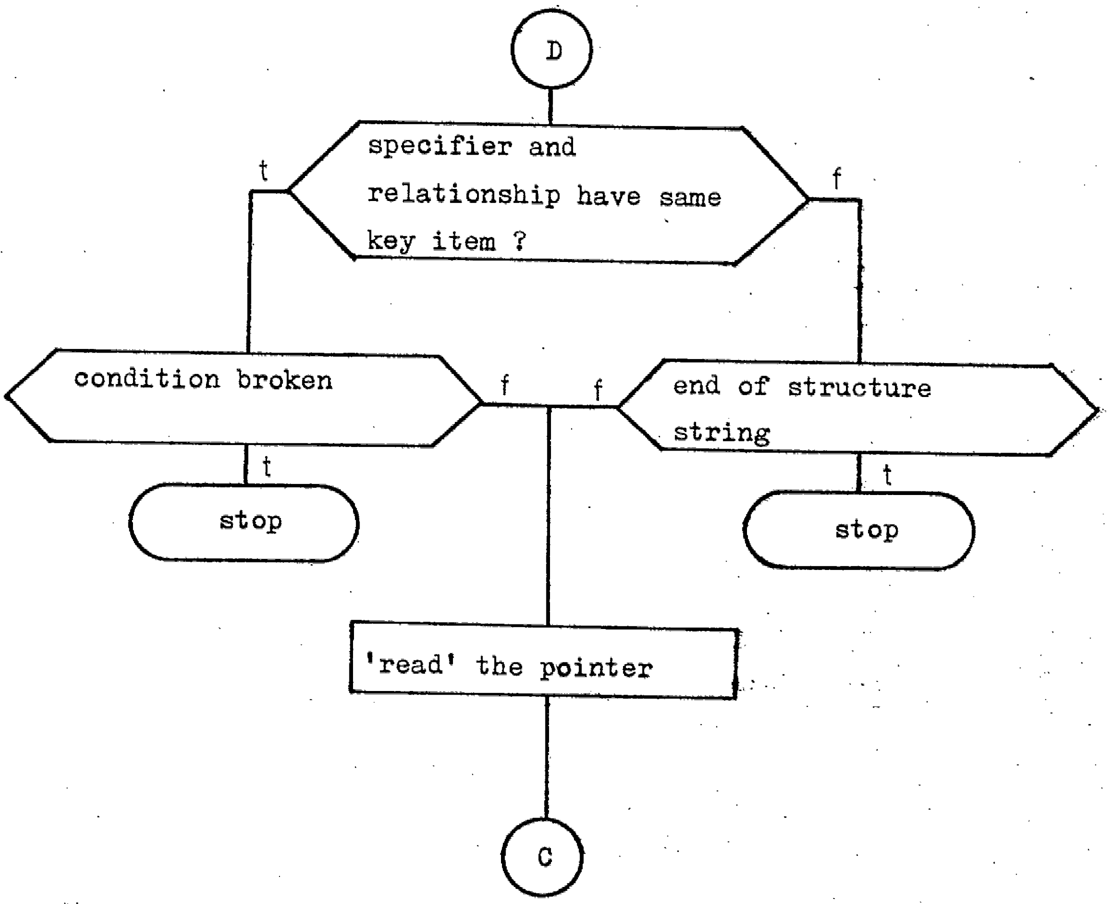

D
specifier and relationship have same key item ?
t
f
condition broken
f
t
stop
end of structure string
f
t
stop
'read' the pointer
C

Fig. 5.2.4.cont.

value specified, is encountered.

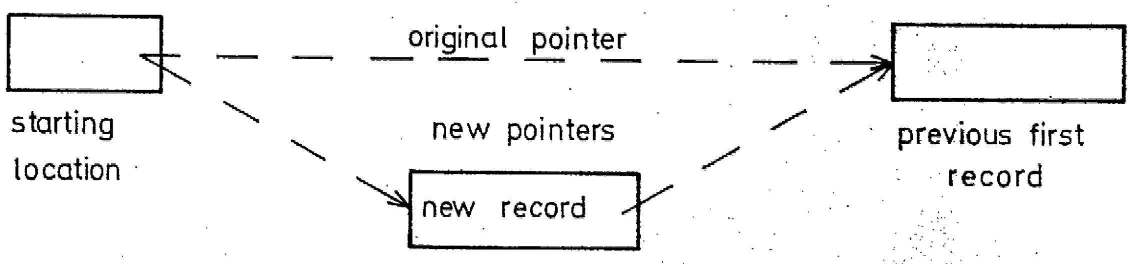
Before a record is released to the user the data must be converted to the form required by the user. This involves applying the appropriate 'read' routine to each item quintuple in turn, until the whole record has been transformed.

When sub-structures are READ it must be noted that many logical combinations can be created from a few physical records and therefore the user must ensure that the READ will not release too many sub-structure occurrences.

5.2.3 The ADD Command (see fig. 5.2.5.)

The ADD command always operates upon a record which is identified by another command, for example, READ or WRITE, and therefore the data map of the record is immediately available. The definition of the GROUP or CHAIN is found in the data map, and the starting point of the relationship is also found.

A new record is ADDed to the relationship by placing a pointer to the new record in the starting location and by placing in the new record a pointer to the record which was previously the first in the relationship.



ADDition of a New Record to a Group or Chain Relationship

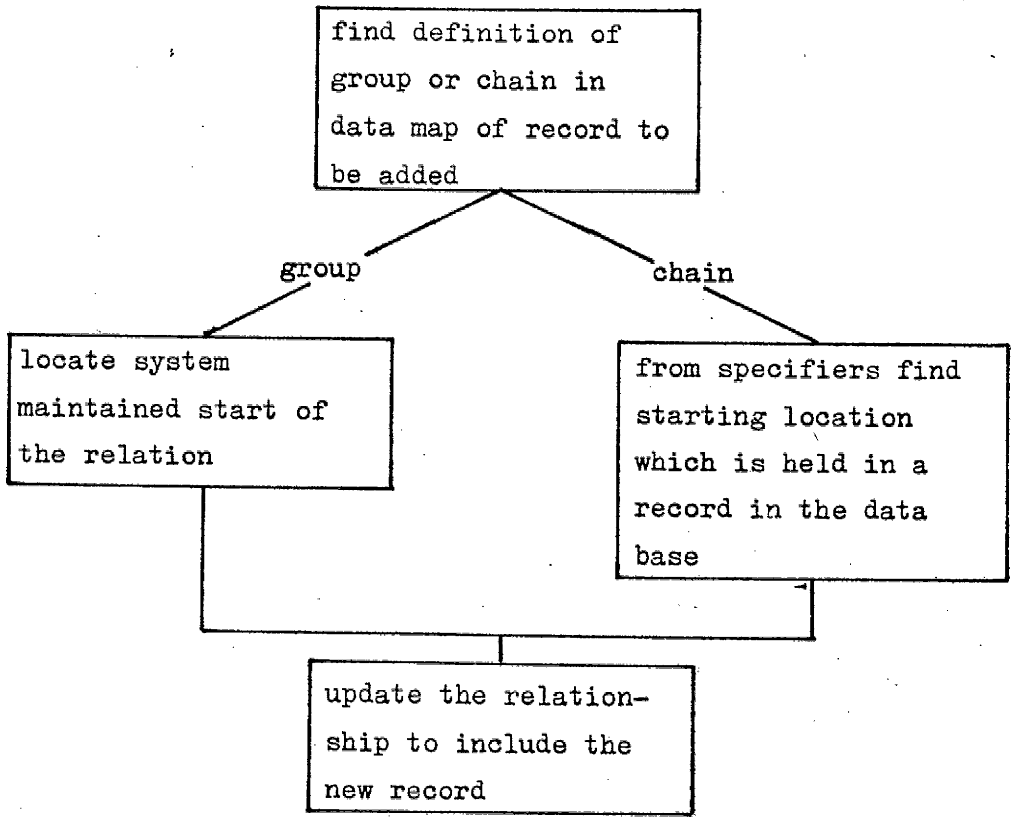

find definition of group or chain in data map of record to be added
group
chain
locate system maintained start of the relation
from specifiers find starting location which is held in a record in the data base
update the relationship to include the new record

Fig. 5.2.5.

5.2.4 The SET Command (see fig. 5.2.6.)

Initially the records which are to be transferred have to be established. This may be a lengthy process as the SET command contains a condition which further qualifies the records which can be transferred. Records which have been transferred must always be available in the normal way, for example, structures must always be accessible from the associated index. This means that when data is moved pointers must be set up so that all the data is still accessible. The same pointing mechanism is used as was used for new ADDitions to relationships.

5.2.5 The ALTER Command (see fig. 5.2.7.)

The ALTER command operates upon one record at a time and the single record occurrence is established by the specifiers. If the current values of the items to be ALTERed do not agree with the values given in the body of the command, the command process is halted. If the changed record does not fit in the space allocated to the previous form of the record, a new physical location for the record will have to be found. This will mean that all relationships which use direct or algorithmic pointers will have to be updated.

5.2.6 The FOLLOW Command (see figs. 5.2.8 and 5.2.9)

The FOLLOW command can be broken down into two operations:

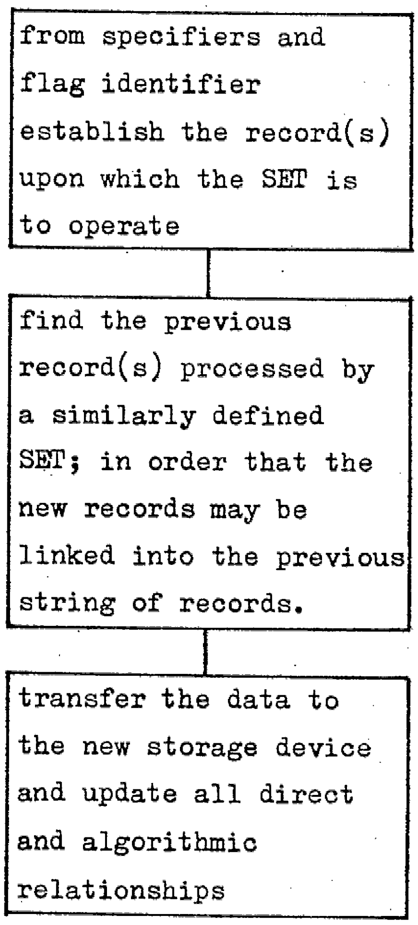

from specifiers and flag identifier establish the record(s) upon which the SET is to operate
find the previous record(s) processed by a similarly defined SET; in order that the new records may be linked into the previous string of records.
transfer the data to the new storage device and update all direct and algorithmic relationships

Fig. 5.2.6.

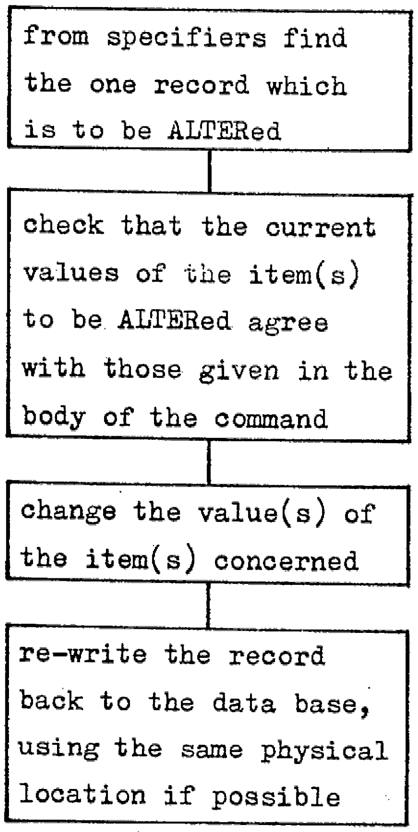

from specifiers find the one record which is to be ALTERed
check that the current values of the item(s) to be ALTERed agree with those given in the body of the command
change the value(s) of the item(s) concerned
re-write the record back to the data base, using the same physical location if possible

Fig. 5.2.7.

(i) the action as a result of encountering another FOLLOW

(ii) the action as a result of encountering an END FOLLOW.

In both cases the action taken is dependent upon the level which is currently being processed. The level is a count of the number of FOLLOW commands which were needed to give rise to the record under consideration.

When a FOLLOW is encountered the level is increased by one, the start of the appropriate relationship is found and processed until the required records are obtained. The levels in the command sequence each correspond to a different relationship being followed.

As each record is processed a record is kept in a first in, last out stack. The information kept in the stack is; the level, the addresss of the record which is being processed, and if the entry is the first after a change of level, an indication of the last such similar level of processing. The same level number can appear in many different command sequences and therefore this linking is not just a case of joining all similar level numbers together. Therefore each FOLLOW has associated with it the stack entry number of the last place where that relationship was processed.

When an END FOLLOW is encountered the level is decreased

FOLLOW

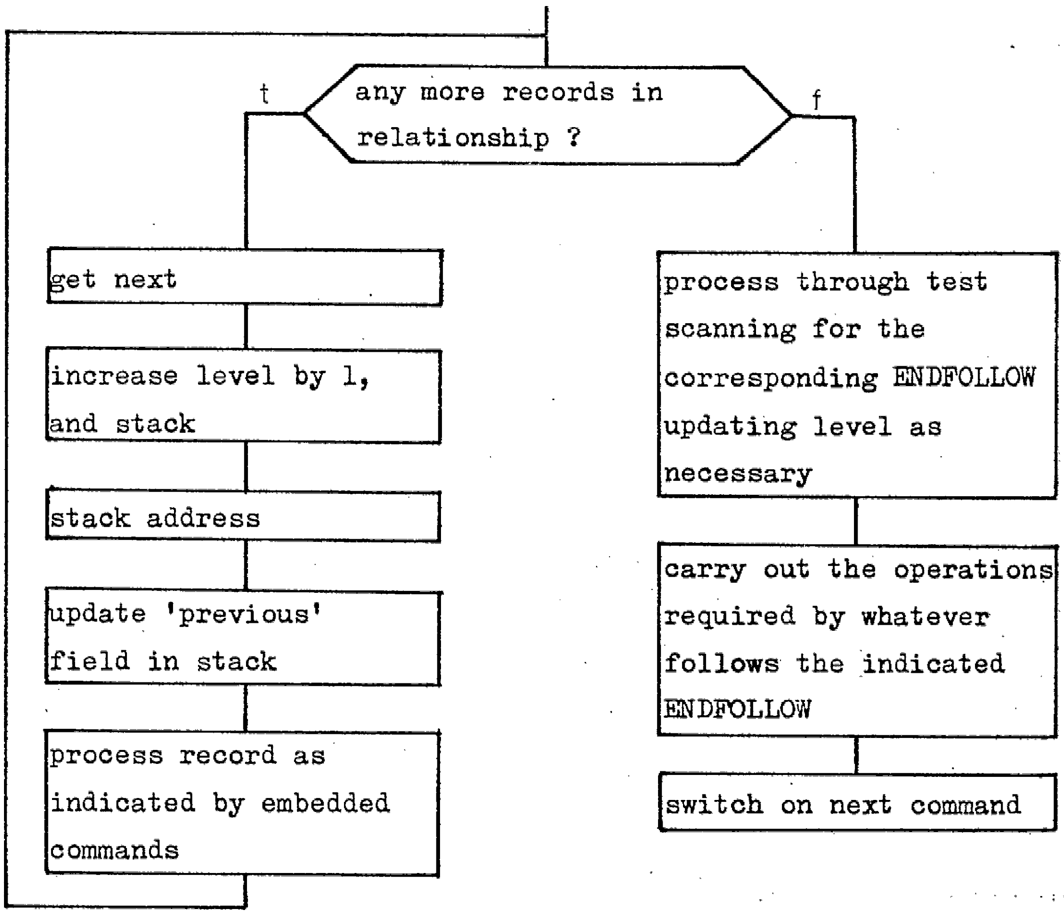

any more records in relationship ?
t
f
get next
increase level by 1, and stack
stack address
update 'previous' field in stack
process record as indicated by embedded commands
process through test scanning for the corresponding ENDFOLLOW updating level as necessary
carry out the operations required by whatever follows the indicated ENDFOLLOW
switch on next command

Fig. 5.2.8.

ENDFOLLOW

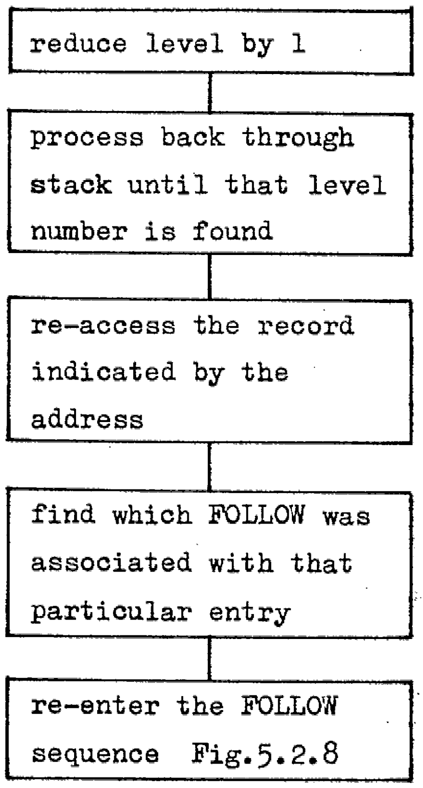
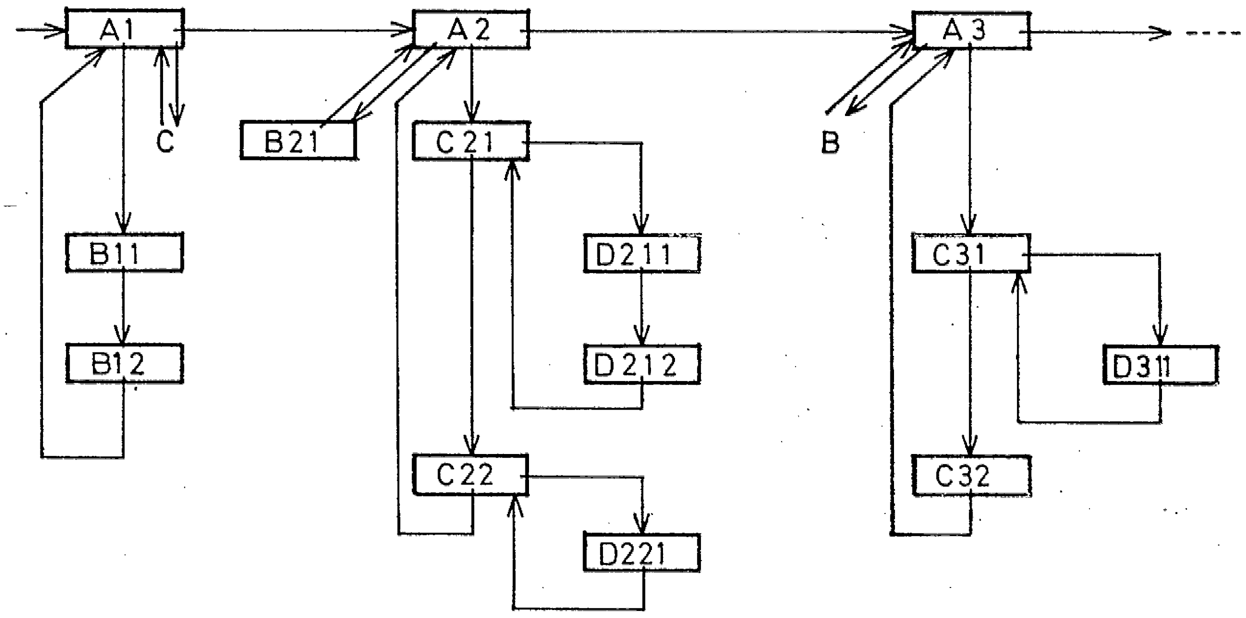

reduce level by 1
process back through stack until that level number is found
re-access the record indicated by the address
find which FOLLOW was associated with that particular entry
re-enter the FOLLOW sequence Fig.5.2.8

Fig. 5.2.9

by one and the entry on the stack which corresponded to the same level of processing is found. The FOLLOW which corresponded to the ENDFOLLOW is re-activated and the next member of the relationship is processed. When the end of the relationship is found the statement following the ENDFOLLOW is processed.



If the above sequence of relationships is processed by the commands:

```
FOLLOW A
  FOLLOW B
  ENDFOLLOW
  FOLLOW C
    FOLLOW D
    ENDFOLLOW
  ENDFOLLOW
ENDFOLLOW
```

the stack given below will be produced. Note that the stack mechanism does not destroy entries as it processes back through the stack, so that reverse processing of the stack is possible using the RETURN and RETRACE commands.

stack entry	level	address	previous (number is stack entry)
1	1	A1	-
2	2	B11	-
3	2	B12	-
4	1	A2	1
5	2	B21	3
6	2	C21	-
7	3	D211	-
8	3	D212	-
9	2	C22	6
10	3	D221	8
11	1	A3	4
12	2	C31	9
13	3	D311	10
14	2	D32	12

5.3. A Pilot Implementation

An implementation of parts of PATCOSY has been carried out. This uses the quintuples which were defined in 5.1. The parts implemented were: the WRITing and READing of structures, including repeating groups; the SET, FOLLOW and GIVE commands were also implemented for restricted cases.

The suite of programs, which were written in COBOL, operated upon data maps which had a fixed length (15 characters) identifier in the identifier field of the quintuples. The commands were implemented as a one-off sequence, containing no host language processing. Data which would have been processed was always output into what would have formed the processing space of the host language program.

WRITE was implemented so that any structure containing any feature, except associate items, could be handled. Repeating groups were handled as if they were a recursive structure, although COBOL has no recursive facilities. Structures, once written, could then be ADDed to any group or chain.

READ was implemented so that all structures satisfying the specifiers were released. Any item of the structure could act as a specifier and not just those which were keys of relationships.

GIVE could operate upon any chain or group.

FOLLOW operated upon a group which contained nested chains. This, although not necessarily the easiest to implement, avoids any problems of trying to write recursive programs in COBOL.

The SET command was implemented so that all data currently pointed to by a pointer-list element was placed on tape storage. Both direct and displacement pointers were implemented using link and double-link relationships.

The use of this type of data map makes for a very flexible system, although it does incur overheads in processing.

Each time an item is required, for whatever reason, access is through the data maps which have to be searched. In a full system an index of the identifiers used and the related data map and location of the definition in the data map could be maintained as an index-sequential file. This increases the storage space required and, in implementing a full system, consideration will have to be given to these factors.

CHAPTER 6

THE OPERATING REQUIREMENTS OF A PATCOSY DATA BASE

It is pertinent to discuss the way in which data base systems in general, and PATCOSY in particular, interact with an operating system, and further, become part of the operating system.

A data base which is constructed to operate in a real-time multi-programming environment will employ an operating system which can cope with the problems which arise. These problems include; making sure that program files are not overwritten accidentally, access is restricted as required, users are made to identify themselves to the system, and data is not corrupted. The operating system also controls the execution of programs so that data transfers are completed before the program is allowed to recommence execution.

A PATCOSY data base, in addition to requiring these standard operating system features, will require additional facilities to cope with; storage commands, the frequency relationship with its implicit data re-organisation and the SET command. It is at this point that the data base management system (DBMS) and the operating system (OS) functions overlap.

The DBMS will maintain a table of the identification of every user together with his associated privacy rating. When a user requires access to the data base the OS and the DBMS will have to interact. The OS will obtain the user identification, which will then be passed to the DBMS so that the privacy rating of the user can be established. The privacy rating will then be transferred via the OS to the process space which has been allotted to the user by the OS.

Once the process is initiated the DBMS and OS will have to interact each time data is required, because pointers are defined and maintained in the DBMS but are manipulated by the OS which accesses the actual physical device. The DBMS and OS together have to solve the problem of 'concurrent update'. This problem is illustrated by the situation in which two users attempt to update the same piece of data at the same time. Each user could be given a copy of the original data and each could then start the update sequence. The user who finishes first might be permitted to write the new data over the old. When the other user finishes, if his update is allowed to run to completion, he will overwrite the new data.

Solutions to the problem might be that;

(i) once an update sequence has been initiated all other users are prevented from accessing the data for whatever type of command or

(ii) both users would be allowed to proceed with their updates, but the user who finishes second would be forced to read the updated data before being allowed to overwrite it and the first user would have to be informed of the situation.

With the large amounts of data transfers implied by PATCOSY either the DBMS will assume control of data placement using the OS as a tool, or the OS will have to be suitably altered to cope with the problems which PATCOSY presents. The major problems are those of store fragmentation and garbage collection.

Store fragmentation occurs if data is simply placed in the next available free space in sequence; this means that when data is deleted or transferred an isolated area of store remains. The likelihood of finding a similar sized piece of data as the next to be written will be remote and so small pieces of storage space become unavailable. Garbage collection is the reverse process; that is, pieces of unused store are identified and concatenated so that they may be used again.

One way of overcoming this problem is to maintain lists of all elements of store which are unused. In this case, a list is maintained for each storage element length which is supported by the DBMS or OS. When new data is written the storage space indicated at the top of the appropriate free list is used; the free list is then suitably updated. When data is deleted or moved to another storage location, the storage space location is added to the top of the appropriate free list. Thus at all times data storage space which is unused is locatable. This method does involve wastage in that the storage element lengths which could be supported would be limited by the number of free lists the system is able to maintain. Hence data would be recorded in the storage space which minimises wastage of space.

Through the EXHIBIT and MAKE commands the D.B.A. has control over the proportions of different storage lengths used. It is therefore possible that searches of the free lists will have to be carried out so that areas of contiguous storage space can be created and made available for re-use. This will, in general, lead to a situation in which data will have to be moved to enable such re-organisations to take place. This type of system is most probably best implemented by considering the storage space associated with a device type as one long string of characters. These strings are then divided into the correct proportions for each storage length. The number of free lists would, however, be increased as separate ones have to be maintained for each device type.

The problems which PATCOSY will cause in operating systems design are immense. There are two possible solutions; first, if the operating system can handle many of these functions the interface between the DBMS and the OS can be well defined and then implemented. Secondly, and most probably the more likely situation, the OS and DBMS functions will intermingle and the data definition part of the data base will be seen as an extension of the operating system.

CHAPTER 7

CONCLUSIONS

In this chapter the potential relationship between data base systems in general, PATCOSY and the National Health Service is discussed and summarised.

7.1 Technical Feasibility

When the N.H.S. is implementing a computer based information system it must ensure that :

(i) the system is transportable between different machines

(ii) the implementation is able to cope with any new devices which come on to the market.

This implies that the coding of the PATCOSY languages will have to be in a high level language supported by the majority of computer manufacturers. Therefore the data base created will not be tied to one machine type. A high level language implementation also makes it possible to add easily new coding to the system to cope with new devices.

There is an inbuilt mechanism in PATCOSY which allows new devices to be used even though much of the programming effort will still have to be made when a new device is used.

The underlying feature of any data base system is that the amount of work done by the central processor is very small in comparison to channel activity. Before the N.H.S. can consider a computer based information system the basic design philosophy of computers will have to be changed to cope with the higher channel activity.

The final requirement of the N.H.S. is that no information, relating to a living person, should be held on slow backing store, tapes and exchangeable-discs, for example. If it were, and the information were required quickly, the time taken to access the information is dependent upon the speed or slowness of human operator intervention. Therefore, to overcome this problem, information must be held on backing store to which access can be made without human intervention. This means that large capacity storage devices similar to those being developed by Ampex (11) will have to be generally available and of proven capability.

7.2 Capabilities of PATCOSY

PATCOSY has been designed specifically for use in the N.H.S. and, as such, reflects the structure and information flow of the N.H.S. However, the N.H.S. can be considered to be a large business in which the product for 'sale' is a service. Therefore PATCOSY could find uses in large commercial organisations.

The features which have been designed into PATCOSY are:

(i) an extensive variable length data capability

(ii) a non-hierarchical privacy structure

(iii) a means of assigning jobs or tasks to operators or technicians

(iv) the ability to control the storage media upon which the data is recorded.

(v) the possibility of redefining and restructuring a data base once it has been created

(vi) a lessening of the difficulties encountered when a computer system is introduced.

7.2.1 Variable Length Data

On the whole most of the data associated with the N.H.S. can be placed into fixed length fields. However, there are some instances, for example, when a data item or group of items may be repeated an unknown number of times, where a variable length facility is required.

If COBOL is used as the host language the full variable length facility will not be usable. Only when the more powerful languages such as PL/1 or Algol are used, will the full power of PATCOSY's variable length facility be realised.

7.2.2 Privacy Structure

PATCOSY has a privacy structure which is not hierarchical, although it can be used in that way if required. Generally, it is accepted that someone who writes information can also read it. This cannot be allowed in the N.H.S. because of the confidentiality of certain aspects of medical records. Thus the privacy structure is based upon absolute values in which a particular user is given a number of different privacy codings so that he can gain access to the data base as required and permitted. The privacy structure is made explicit when the data base is defined and not by subsequent program definition.

7.2.3 Operator Tasking

Many functions, whether in a commercial organisation or in the N.H.S., require that operators and technicians have their work load organised by some external means. PATCOSY has a built-in system which provides this facility.

7.2.4 Storage Control

The nature of data in the N.H.S. makes it imperative that some means is provided for controlling where, and for how long, a particular piece of data is to be recorded. These functions, because storage control can be visualised as an on-going process finishing with data either being archived or destroyed, are built into the definition when the data base is initially established.

The control sequences allow the data to remain on a device for a specific length of time, or until a condition arises which necessitates the data being moved. The condition is established by a user program setting a flag which will cause the data to be moved. Jones and Ould (12) suggest that for the N.H.S. "The requirement may be a data base where structure is partially controlled by users at terminals". PATCOSY fulfills this requirement.

7.2.5 Redefinition and Restructuring

PATCOSY provides a means of changing the definition of the data base whenever it may be required. It has also been designed so that only those programs accessing data items, which have been altered, will have to be changed.

Once a data base has been set up it may prove that the organisation of the data base is inefficient. In such a case the organisation of the data base can be changed by an in-built command.

7.2.6 Data Base Introduction

The N.H.S. has at present many computer systems which have all been designed for some specific purpose such as payroll and finance. A vast amount of programming effort has been put into these systems. This effort can be utilised when a data base system is installed because PATCOSY allows data transfer between the data base and files outside the data base. During the transfer the data can be changed in many ways so that the previously written programs can be utilised with obvious economy.

7.2.7 Other Features

PATCOSY allows relationships or networks to be created which can be processed in many different ways. For example, the path taken through the network can be processed in the reverse direction, so that it is always possible to return to where the process has already been.

Some of the manipulation commands contain a field for the host language instructions so that the operations upon particular records are well defined. This is especially useful when it is considered that one command could yield many different records satisfying the conditions defined in the command.

7.3 Summary

Many of the features outlined above were described by Ollé (13) as being useful developments in data base design, especially that of storage control and the introduction of a data base system. These features are also defined as a pre-requisite of the Guide/Share Report. A comparison of DBTG and Guide/Share was carried out by Ollé (14). PATCOSY compares favourably with both these proposals offering a way forward to computerisation of the N.H.S. and gaining the intended benefits of the 1974 re-organisation.

REFERENCES

1. A Review of the Medical Services of Great Britain - Medical Services Review Committee, Chairman, Sir. A. Porritt 1963 Social Assay

2. Doctors in an Integrated Health Service - Scottish Home and Health Department H.M.S.O. 1971. SBN 11 490664 5

3. Nurses in an Integrated Health Service - Scottish Home and Health Department H.M.S.O. 1972 SBN 11 490848 6

4. The Organisation of Group Practice - Central Health Services Council H.M.S.O. 1971 SBN 11 320429 9

5. National Health Service Re-organisation - England H.M.S.O. Cmnd. 5055 SBN 10 150550 7

6. Si Vis Pacem, B. Edwards, P. R. Walker - Nuffield Provincial Hospitals Trust by Oxford University Press 1973 ISBN 0 19 721374 X

7. Report of the Codasyl Data Base Task Group - British Computer Society - April 1971

8. Codd E. F. 1971, A.C.M. Sigfidet Workshop on Data Description, Access and Control - Pages 1, 20, 35

9. A Survey of Generalised Data Base Management Systems - Codasyl Systems Committee, Association for Computing Machinery, May 1969

10. Weed L. L., Medical records, medical education and patient care : the problem-orientated record as a tool. Case Western Reserve University, Cleveland Ohio 1971

11. A.F.I.P.S. - Volume 33, Part 2, Page 1381

12. Jones B. E. and Ould M. A., Computer Journal Volume 17, Number 4, Pages 295 to 301

13. Date Base Management - Infotech State of the Art Report Number 15 Pages 409-428

14. Ollé T.W., An Assessment of how the Codasyl Data Base Task Group Proposals meet the Guide/Share Requirements - Norwegian Computer Centre April 1972

APPENDIX 1

THE SYNTAX AND SEMANTICS OF THE PATCOSY DEFINITION LANGUAGE

A.1.1 Introduction

The formal syntax and semantics of the PDL are presented. This appendix is not intended as a user guide, the elements of which can be extracted from Chapter 4. The language specifications give rules for the formation of data structures and the exact meaning of the constructs.

A.1.2 Notation

Symbol	Meaning
< >	object indicated is defined elsewhere
[ ]	optionally
| |	at least one and any element repeated in any order
{ }	choose one only
|| ||	each element can be repeated zero or many times, in any order
aaaaaa	'a' an alphabetic character. a mandatory symbol
aaaaaa	an optional symbol of the language
.,();=	mandatory punctuation symbols
–	continued on next line

Symbol	Meaning
name-n	'n' a positive integer. A D.B.A. supplied identifier. An identifier is a string of alphanumeric characters and - which start with an alphabetic character.
value-n	A D.B.A. supplied number
literal-n	A D.B.A. supplied literal. A literal is a string of characters enclosed in primes. Note: '2' is not the same as 2.

A.1.3 General Layout of a Data Definition Program

Syntax

```
    start name-1.
      [system-section. <system sentence>.]
      [generating-section. |<generating sentence>|]
      [initialising-section. |<initialising sentence>.|]
      [editing-section. |<editing sentence>.|]
      [change-section. |<change sentence>.|]
      end name-1.
```

Semantics

1. Name-1 identifies the data base. This identifier will only assume importance when a computer is maintaining more than one data base.

2. Each section will be described in turn.

A.1.4 The System-Section

Syntax

```
<system sentence> is defined to be
[character is value-1 bits -
 core is value-2 characters -
|drum is value-3 units of value-4 characters| -
|fixed-disc is value-5 units of value-6 characters| -
|exchangeable-disc is value-7 units of value-8 characters| -
|tape is value-9 units of value-10 characters| -
|other [backing] is value-11 units of -
          value-12 characters|] -
[multiprogrammed] -
[maximum storage length is value-13 characters] -
[priority [read] [add] [erase] [alter] [give] [find] -
[delete] [follow] [retrace] [return] [sort] [set] [move] -
[transfer] [destroy] ]
```

Semantics

1. The system sentence gives details of the operating environment which has been allocated to the data base.

2. All storage amounts are given in number of characters (equivalently bytes). Values 3,5,7,9,11 give the number of units of each device type and values 4,6,8,10,12 give the number of characters on each unit. K equalling 1024 is allowed thus 10K = 10240.

3. A device type entry must be repeated if different units exist which contain a different number of characters.

4. Other is to be used when a different device is used. If the device is used as backing or archive storage then

backing must be used. In the sentence other is replaced by an identifier of the device

5. If multiprogrammed is specified the system defined will be one which allows more than one user to access the data base on a time sharing basis.

6. Maximum storage length indicates the length of the longest storage unit to be maintained by the data base. Value-13 must be of the form 2^n where $n \geq 5$ and integer.

7. The priority statement indicates the order commands will be dealt with if more than one command is issued for the same piece of data. The order shown is the default option which will be assumed if the statement is omitted.

Note: This is the only place where a default option has been specified, mainly due to the length of the statement. No other default options have been given, and it is intended that no others should be specified, because different installations may require different default options which may lead to non-standardisation with the consequent loss of transportability.

8. The whole sentence may be omitted in any program subsequent to the initial definition. The sentence can be changed by a change sentence.

A.1.5 Generating-Section

Syntax

<generating sentence> is defined to be

$$\left\{\begin{array}{l}\text{<basis sentence> [<privacy sentence>] |<class sentence>.|} \\ \text{<sub-basis sentence>.}\end{array}\right\}$$

A.1.5.1 Syntax

<basis sentence> is defined to be

$$\underline{\text{basis}}\ \text{name-1}$$

Semantics

1. A data base is defined in terms of one or more independent bases each identified by name-1.

2. All definitions of data structures occur at the basis level in the <class sentence>, changes of definition and data structure combination are allowed in the <sub-basis sentence>.

A.1.5.2 Syntax

<privacy sentence> is defined to be

$$\underline{\text{privacy}}\ \left\Vert\ [\text{<command-1>}]\ \right\Vert \left\{\begin{array}{l}\text{value-1}\\ \text{value-2}\ \underline{\text{to}}\ \text{value-3}\end{array}\right\}, \Big\Vert\ -$$

$$\left\{\begin{array}{l}\text{value-4}\\ \text{value-5}\ \underline{\text{to}}\ \text{value-6}\end{array}\right\}; \Big\Vert\ [\text{<command-2>}]\ \left\Vert \left\{\begin{array}{l}\text{value-7}\\ \text{value-8}\ \underline{\text{to}}\ \text{value-9}\end{array}\right\}, \right\Vert\ -$$

$$\left\{\begin{array}{l}\text{value-10}\\ \text{value-11}\ \underline{\text{to}}\ \text{value-12}\end{array}\right\}$$

<command> is defined to be

$$\left\{\begin{array}{l}\underline{\text{read}}\ \underline{\text{write}}\ \underline{\text{alter}}\ \underline{\text{delete}}\ \underline{\text{find}}\ \underline{\text{sort}}\ \underline{\text{add}}\ \underline{\text{erase}}\\ \underline{\text{give}}\ \underline{\text{follow}}\ \underline{\text{set}}\ \underline{\text{move}}\ \underline{\text{transfer}}\ \underline{\text{destroy}}\ \underline{\text{exhibit}}\\ \underline{\text{current}}\ \underline{\text{membership}}\ \underline{\text{make}}\end{array}\right\}$$

Semantics

1. The D.B.A. will assign to each user privacy codings which will be maintained in the privacy table.

2. Values 1 and 4, and 7 and 10 specify sequences of single codes. Values 2 and 3, 5 and 6, 8 and 9, 11 and 12

specify sequences of ranges of codes. The values are inclusive, values 2, 5, 8 and 11 are the lower bound of the range.

3. To limit the privacy code to a specific operation a <command> can precede one or many of the privacy specifications.

4. Before a user is allowed access to a part of the data base a check is made to ensure that at least one of the user's privacy codes and a privacy code attached to that part of the data base match.

A.1.5.3. Syntax

```
<sub-basis sentence> is defined to be
sub-basis name-1 [<privacy sentence-1>] contains -
|| [|| [not] name-2 [<privacy sentence-2>],|| -
[not] name-3 [<privacy sentence-3>] of] <type-name-1> -
[<privacy sentence-4>];||   [|| [not] name-4 [<privacy sentence-5>],|| -
[not] name-5 [<privacy sentence-6>] of] <type-name-2> -
[<privacy sentence-7>]
```

Note: A construction <something-n> is deemed to be defined in the same way as <something>

Semantics

1. Name 1 is the identifier of the sub-basis as a whole and privacy sentence 1 refers to the whole sub-basis.

2. A sub-basis contains data structures which have been defined in a basis. The data structures required are identified by type-names 1 and 2. Privacy sentences 4 and 7 referring to the privacy coding of the data structure as a whole in this sub-basis. If these or any other privacy sentences are

omitted the privacy will be the same as that in the basis of definition.

3. If only parts of the data structure are required then these can be identified by names 2, 3, 4 and 5. The privacy codings also can be altered by privacy sentences 2, 3, 5 and 6.

4. If the majority of a data structure is required then those items not required can be specified by inserting not as appropriate. However, once not has been used all occurrences of a name before of must be preceded by not.

A.1.5.4. Syntax

```
<class sentence> is defined to be
<class> <type-name> [<privacy sentence>] contains -
<item> [connections are <relationship>] [storage is -
<storage command>]


<class> is defined to be
{ [master] index                                          }
{ sub-structure pointer-list dictionary list              }
{ table pointer-array structure transfer-structure        }


<type-name> is defined to be name-1
```

Semantics

1. The class defines the data structure being defined. The data structure is identified by the type-name.

2. The privacy sentence refers to the type as a whole.

3. A definition of each class follows:

(i) Index

An index is an entity which holds a key by which other data is identified as belonging to a particular key value. An index is a contiguous sequence of elements each element constructed from one occurrence of the definition. An element of an index must be fixed length and must contain a pointer to a pointer-list. In a data base only one index can be preceded by master. Master can only be used when more than one index is defined. In such a system the master index always points to the defined pointer-list. The other indices defined may contain entries which point to the same pointer-list.

(ii) Pointer-List

In each basis defined a pointer-list must contain as many pointers as there are structures defined in the basis. One pointer pointing to each structure or a pointer-array.

(iii) Pointer-Array

If for one index key value one structure definition could be used many times each differentiated from the other by one key value then a pointer-array can be used. A pointer-array element contains a pointer and an item which contains the key value. More elements can be added to a pointer-array at any later time.

(iv) Structure

A structure is the data construct which holds information in the data base, when the information can be associated with a particular index key value. For each pointer in a pointer-list or pointer-array a string of occurrences of that structure is created. A structure constitutes a physical record.

(v) Sub-Structure

A sub-structure is a logical record. That is, it is made up from parts of one or more structures. The data format can be changed between a structure and a sub-structure.

(vi) List

A list is similar to a structure except that data is accessed using the list identifier instead of an index key value.

(vii) Dictionary

A dictionary provides a means of encoding data. A dictionary also provides a means of associating one item in a list or structure with more than one item from a dictionary element. A dictionary can be accessed therefore, by many different key items, values of which have to be maintained in the list or structure. The elements of a dictionary can be of variable length and therefore the elements are not contiguous. An element is one occurrence of the definition of the dictionary being recorded.

(viii) Table

A table is similar to a dictionary except that the elements are contiguous and of fixed length. Also only two key items can be specified.

(ix) Transfer-Structure

Initially a data base may be initiated into an environment where a computer information system already exists. To save on initial programming effort data can be transferred from the data base to a file in the format required by the previously written programs. The transfer-structure provides the mapping function. This function also allows for transfer between two different computers.

A.1.5.4.1 Syntax

<item> is defined to be

```
| fixed name-1 length value-1 [computational] -                   |
|     [<privacy sentence>]                                        |
| variable name-2 [<privacy sentence>]                            |
| associate name-3 with ||name-4,|| name-5 of name-6 -            |
|     for name-7 = name-8 [<privacy sentence>]                    |
| pointer name-9 points to name-10 [<privacy sentence>] -         |
|     <mechanism>                                                 |
|  {fixed name-11 length value-2}                                 |
|  {variable name-12            } (<item>) -                      |
|     [<privacy sentence>]                                        |
```

<mechanism> is defined to be

```
pointer mechanism is {direct displacement name-13}
```

Each class is restricted in the item(s) it may contain.

(i) Index may contain only fixed and pointer.

(ii) Pointer-list may contain only <u>pointer</u>.

(iii) Pointer-array may contain one <u>fixed</u> or <u>variable</u> item and a <u>pointer</u>.

(iv) Structure may contain everything but <u>pointer</u>.

(v) Sub-structure has its own defined items see A1.5.4.2.

(vi) List may contain anything but <u>pointer</u>.

(vii) Dictionary may contain everything but <u>pointer</u>.

(viii) Table may contain <u>fixed</u> and <u>associate</u> only.

(ix) Transfer-structure has its own defined items see A1.5.4.3.

<u>Semantics</u>

1. Name 1 is the identifier of a fixed length data item, the length of which is value 1 characters.

2. If <u>computational</u> is specified the item must be numeric and will be held in floating-point format. In an implementation the length of <u>computational</u> items may be restricted to prescribed values.

3. If <u>computational</u> is not specified the item will be held as a string of characters.

4. Name 2 is the identifier of a variable length data item, the length only being defined at the time of writing. The item is always held as a string of characters.

5. Name 3 identifies an item whose value(s) are obtained from a dictionary or table depending on the value of a key item contained in the type.

6. The item(s) to be associated are identified by names 4 and 5.

7. The dictionary or table is identified by name 6.

8. The item of the dictionary or table which is to be used as the key is identified by name 7.

9. The item from which the key value is obtained is identified by name 8. Name 8 must be defined in the same record type. If name 3 is defined in a repeating group (see 11 below) then name 8 must be defined in the same repeating group.

10. Name 9 identifies a pointer which points to the record type identified by name 10. Mechanism defines the manner in which the pointer will be implemented. Direct means that the actual physical address of the object being pointed to will be recorded as the pointer. Displacement can only be used in lists and structure relationships and is included here for completeness only. Name 13 identifies an algorithm which returns a physical address which will be recorded in the pointer field.

11. Names 11 and 12 identify repeating groups. A fixed repeating group is one in which the number of times the element is repeated is defined by value 2. In a variable repeating group this is defined at the time of writing.

12. An element of a repeating group is an occurence of <item>.

13. This definition allows a repeating group to contain another repeating group to any level.

14. A variable repeating group can have another element added to it at any time if a sub-structure is defined which

has the same definition as the repeating group element.

15. Any of the items may have a privacy sentence defined. If not, then the privacy will be the same as the whole record type. The lower the level, the more restrictive privacy becomes. The highest level being the basis, the lowest the item.

A.1.5.4.2 Syntax

```
<item> for a sub-structure is defined to be
|name-1 ||of name-2|| [to  {computational}  [using name-3]]
                           {characters   }
[{left } justified in value-1 characters -
 {right}
[{sign   }               ]]
 {space  }               ]]
 {zero   }  extension    ]]  [<privacy sentence>]|
 {literal}               ]]
```

Semantics

1. A sub-structure is constructed from one or more items from one or more different structures.

2. Name 1 is the identifier of the item required.

3. The last occurrence of name 2 identifies the structure where name 1 is defined.

4. If name 1 is an associated item then another occurrence of name 2 will be required to identify the associated item unless name 1 identifies the whole of the associated item.

5. If the whole structure is required, and needs no conversions then name 2 can be omitted and name 1 will identify the structure. The privacy sentence can still be used in this event.

6. If the item is to be converted from computational to characters or vice versa then the one specified in the sub-structure is that which the item is not in the structure.

7. If the conversion is to be carried out using a special algorithm table or dictionary then name 3 identifies the algorithm etc.

8. The justified clause allows data to be shifted within the new field.

9. The length of the new field is given by value 1. If value 1 is less than the original length then data will be lost from the right hand end if left justification is specified or vice versa.

10. If the new field is longer then the data will start with the left or right hand ends aligned with the left or right hand end of the new field. In this case the extension statement may be used. This identifies with what the extra space in the field is to be filled.

11. If the literal is not long enough it will be repeated until all space is filled. If left justified is specified the literal will always be written starting at the left side of the literal. Similarly if the literal is too long only part of the literal will be used. The part used depending on the justification.

12. If the item is held in the computational form, and the form is restricted to predefined values, then any change of length must be between these predefined values.

13. The privacy sentence refers to the item only and not to the whole sub-structure.

A.1.5.4.3. Syntax

```
     <item> for a transfer-structure is defined to be

  |name-1 || of name-2|| [associate name-1 with name-3 -
       of name-4 for name-5] -

  [from  {6 to 8 bit}  characters using name-6]  -
  [      {8 to 6 bit}                          ]

  [to  {computational}  [using name-7]]  -
  [    {character    }                ]

  [ {left }  justified in value-1  {characters}  -
  [ {right}                        {bits      }

  [ {space  }            ]] [after value-2 {characters}]  -
  [ {zero   }  extension ]] [              {bits      }]
  [ {sign   }            ]]
  [ {literal}            ]]

  [associate {name-1} with name-8 of name-9 for name-10]|
  [          {name-3}                                  ]|
```

Semantics

1. The statements which appear in the sub-structure have exactly the same meaning in the transfer-structure.

2. If associate is given then the value of the item will be used as the value of the key item name 5 or 10, using the dictionary or table identified by name 4 or 9. The new value will be obtained from the value of name 3 or 8.

3. If the associate with name 3 is specified then it is that value which will be converted and not the value of name 1.

4. In the second associate name 1 or name 3 can be specified depending on whether the first associate was used or not. In either case the value of name 10 used will be the one which has been subject to any conversions.

5. If the transfer is to take place between two different character length machines the from statement can be used.

If the data base is held on a 6 bit machine the option chosen will be from 6 to 8 bit character. Name 6 identifies a table, dictionary or algorithm where the conversion is held.

6. If a space is required in the transfer <u>after</u> can be used. In this case value 2 characters or bits will be left after the previous item before the new item is written.

A1.5.4.4 Syntax

<relationship> is defined to be

```
group ||name-1 <mechanism> [<privacy sentence>];|| -
    name-2 <mechanism> [<privacy sentence>]
{start }  chain ||name-3 <mechanism> [<privacy sentence>];|| -
{member}
    name-4 <mechanism> [<privacy sentence>]
owned by name-5 [<privacy sentence>]
accessed by ||name-6 {ascending } [<mechanism>],|| -
                     {descending}
    name-7 {ascending } [<mechanism>]
           {descending}
{link       } name-8 by name-9 {ascending } <mechanism>
{double-link}                  {descending}
tree name-10 by name-11 using literal-1 as initial node -
    <mechanism>
frequency [after value-1 days] every value-2 days -
||value-3 at level value-4,|| {value-5} at level value-6
                              {rest   }
```

Semantics

1. Groups and chains are relationships which provide connections between different record types. A record can only be a member of a group as the start of a group is maintained

by the system.

2. A chain is similar to a group except that a starting point must be defined. Only one occurrence of a record type can be the start of a chain though many can be members.

3. Names 1 and 2 identify the groups to which the record type being defined may belong. The mechanism defines the way in which the relationship's pointer will be maintained. The privacy sentence indicating any restrictions applicable to that group.

4. A record can be defined as being the start of a chain or a member of a chain. The remaining parts of the statement having the same meaning as in a group.

5. Owned by can only be specified for structures. This relationship maintains an indication of what index value is the origin of the particular structure occurrence. The privacy can be restricted by the privacy sentence.

6. Accessed by is the only relationship allowed for indices and must be present with name 6 occurrences omitted. Accessed by indicates which items of a table dictionary or index can be used as key items and the mechanism indicates the manner in which one element will be connected with another. Ascending or descending indicates the sequence in which the elements will be presented. Ascending means the smallest will be presented first, descending the largest first.

7. Accessed by can also be defined in types containing a repeating group data item. In this case mechanism must not be specified. This form of accessed by provides a means of ordering the elements of a repeating group depending on the

values of items contained within the repeating group. Therefore names 6 and 7 must be items within the repeating group. Items will be ordered by the value of the item indicated by the first occurrence of name 6. Other items only being used to break equalities.

8. Mechanism must not be specified for the <u>accessed</u> by when used in indices.

9. Link and double-link provide a means of connecting records of the same type. The records will be ordered by the values of the item identified by name 9. Name 8 is the identifier by which the relationship will be referenced. The mechanism indicates what connection is to be used to join the record occurrences into a string.

10. Tree is similarly a means of joining records of the same type except that a binary tree is formed instead of a string. Name 10 identifies the tree. Name 11 identifies the item within the record, which will be used as the key for the tree. Literal 1 specifies a value of that item which will form the first node of the tree even though a record with that value of key item may not yet exist.

11. In the accessed by, link and double-link, and tree relationships no privacy is specified. The privacy of these relationships is defined by the privacy associated with the key items. The items are identified by names 6, 7, 9 and 11. In addition, names 9 and 11 must not be part of a repeating group.

12. The frequency relationship can only be specified for a dictionary and provides a means of ensuring that the most

frequently accessed elements are the most readily available. A count is maintained of how many times an element is accessed. At the end of a period each element's count is accessed so that the dictionary can be suitably re-organised.

13. If value 1 is specified the first re-organisation will take place <u>after</u> that number of days. Otherwise or thereafter the re-organisation will take place <u>every</u> value 2 days.

14. Value 3 and 5 specify the count values which are to be held at each <u>level</u> of the dictionary or table. Values 4 and 6 assume the values of the positive integers in sequence. Instead of value 5 <u>rest</u> may be specified indicating that the remaining elements of the table or dictionary are to be held at the bottom level.

15. Pointer-lists, pointer-arrays, sub-structures and transfer-structures may have no relationships specified.

16. If, in a sub-structure or a sub-basis an item is not defined which is the key item of a relationship then that relationship is also unavailable.

17. In structures and lists at least one of the relationships link, double-link or tree must be specified. Further, at least one of the occurrences must be defined as <u>direct</u> or algorithmic.

A.1.5.4.5 Syntax

<storage command> is defined to be

```
| kept  {forever                                                       } -  |
|       {[for  value-1  days ] [or]  [until ‖ name-1,‖  name-2]        }    |
|                                                                           |
|      | {drum                          }                    |              |
|      | {fixed-disc                    }                    |              |
|  on  | {exchangeable-disc             }  [level  value-2]  |  then  -     |
|      | {{tape }    {randomly }        }                    |              |
|      | {{other}    {collected}        }                    |              |
|      | {core                          }                    |              |
|                                                                           |
|             <storage  command>                                            |
|                                                                           |
| on-lined  when  name-3 [to name-4],  name-5 [to name-6]                   |
| read  when  name-7 [to name-8],  name-9 [to name-10]  -                   |
|             returned   {immediately          }                            |
|                        {after  value-3  days }                            |
|                                                                           |
| destroyed                                                                 |
```

Storage commands must not be specified for sub-structures and transfer-structures and must be specified for all other classes.

Semantics

1. The storage commands give details of where data is to be recorded and for how long.

2. The kept command defines the initial and subsequent location of any data. Data is kept either forever or until some condition is satisfied on a particular hardware type. The condition takes one of three forms:

(i) If value 1 is specified then the data will remain on a device until that number of days has elapsed since the data was written.

(ii) The data will remain on a device until a flag identified by names 1 and 2 is set.

(iii) A combination of the previous two cases, in which the two cases are or-ed together.

3. The device is specified by choosing one of the storage media specifiable. <u>Level</u> is required only when <u>frequency</u> is specified, in which case a device must be specified for each level of the dictionary. Value 2 therefore, assumes the values of the increasing positive integers. <u>Kept</u> <u>forever</u> may only be specified in this case.

4. <u>Then</u> must be specified when <u>forever</u> is not specified and must not be specified when <u>forever</u> is specified. The following storage command indicates where the data is to be moved to.

5. Device types <u>tape</u> and <u>other</u> can be followed by <u>randomly</u> or <u>collected</u>. <u>Randomly</u> means that data will be written to the tape in the pieces as they are made available in the location on the tape which is next to be used. The pieces are linked together by pointers so that records are maintained. <u>Collected</u> means that data will be archived so that all data relating to a particular index key value will be contiguous.

6. Destroyed means that the data will be removed from the data base.

7. The other two commands will only become effective when the data has been archived to tape or other.

8. On-lined is used when data is required for processing and as such is required to pass through the complete sequence of kept commands. Data will be on-lined when flags identified by names 3 and 5 are <u>set</u>. The data will be assigned to the index identified by names 4 and 6. If name 4 or 6 is not specified than the data will be accessible from the index which

must be the only one defined in the sub-basis or basis which was used to set the flag. (see set command in Appendix 2.2.5).

9. Read is similar except that a copy from the tape is made available. The data can only be read and no other processing is allowed. The copy will be destroyed either immediately or after the specified number of days. Names 7 to 10 have the same meaning as names 3 to 6.

A.1.6 Initialising - Section

<initialise sentence> is defined to be

```
⎧ initialise <type-name> length value-1 with -                  ⎫
⎪     ‖data-entry-1,‖ data-entry-2                               ⎪
⎨ algorithm name-1 in name-2 using ‖name-3,‖ name-4 -           ⎬
⎪     begin <program> end into ⎧location⎫                       ⎪
⎩                              ⎩result  ⎭                       ⎭
```

A data-entry is values of items, in the correct character sequence, which constitutes an element of a dictionary, table or index. A <program> is a sequence of instructions.

Semantics

1. The initialise sentence allows data to be placed in dictionaries, tables or indices without the necessity of using many write commands. In addition the sentence also allows an algorithm to be defined.

2. Type-name identifies the table, dictionary or index and value 1 specifies how many elements are being defined, which are followed by each data-entry separated by a comma.

3. Name 1 identifies the algorithm and name 2 the programming language. Names 3 and 4 identify any items which will be used in the sequence of instructions.

4. If the algorithm is used in mechanism location must be specified otherwise result must be specified.

A.1.7 Editing-Section

Syntax

<editing sentence> is defined to be

$$\left\{\begin{array}{l}\underline{\text{precede}}\\ \underline{\text{supercede}}\end{array}\right\} \quad \text{name-1} \quad \underline{\text{by}} \quad \text{name-2}$$

Semantics

1. Names 1 and 2 identify applications programs which use the data base.

2. If a change has taken place, or some monitoring of the data base is required then the program name 1 can be performed before or after name 2 depending upon precede or supercede respectively.

A.1.8 Change-Section

Syntax

<change sentence> is defined to be

$$\underline{\text{change}} \quad \left\{\begin{array}{l}\text{sentence-1}\\ \underline{\text{empty}}\end{array}\right\} \quad \underline{\text{to}} \quad \left\{\begin{array}{l}\text{sentence-2}\\ \underline{\text{empty}}\end{array}\right\} \quad \left[\,\underline{\text{in}} \quad \text{name-1}\right]$$

Semantics

1. The change sentence allows a data base definition to be altered automatically assigning a new generation number to the new definition.

2. Name 1 identifies the basis in which the change is to take place. It is not needed in the case of a change to a sub-basis sentence.

3. A <u>sentence</u> is a string of symbols preceded by, but not including, a full-stop and terminated by, but not including, a full-stop.

4. If the <u>sentence</u> is completely new then sentence-1 can be replaced by <u>empty</u>.

5. If the <u>sentence</u> is to be deleted then sentence-2 can be replaced by <u>empty</u>.

APPENDIX 2

THE SYNTAX AND SEMANTICS OF THE PATCOSY MANIPULATION LANGUAGE

A.2.1 Introduction

The PML naturally divides itself into three parts:

(i) Commands which cause data to be read or written

(ii) Commands which operate upon and maintain relationships

(iii) Utility and statistical commands.

The first two types of command are intended for use by all users. The third type being used only by the D.B.A. in carrying out his role of maintaining an efficient data base.

The notation used is the same as is used in Appendix 1.

A.2.2 Prime Commands

A.2.2.1 The Write Command

Syntax

```
write <type-name> [for <unique-specifier>] from name-1 -
using name-2 {exclusively }
             {concurrently}
```

<unique-specifier> is defined to be

‖ name-1 = value-1,‖ name-2 = value-2

further

<specifier> is defined to be

‖ name-1 = value-1 [to value-2],‖ name-2 = value-3 to value-4

Semantics

1. Type-name is the identifier of the record to be written.

2. In order that a unique logical location can be found for the record, a unique specifier must be given. A unique specifier is made up of names of items together with a value of the item.

3. The unique specifier is not needed when a list occurrence is written as only one list of a particular name exists in the data base.

4. The comma separating each item and its value can be considered to be a logical 'and'.

5. Name 1 identifies the field in a user's program from which the data to be written can be obtained.

6. Name 2 identifies the basis or sub-basis which is to be used when the operation is performed.

7. If <u>exclusively</u> is specified then the user issuing the command will be the only one who has access to the named basis or sub-basis during the write operation.

8. The order in which a sequence of exclusive commands, each of which uses the same basis or sub-basis, is determined by the <u>priority</u> assigned to the command type in the system section of the data definition program.

9. If <u>concurrently</u> is specified then any number of users may access the basis or sub-basis at the same time.

10. <u>Exclusively</u> and <u>concurrently</u> may only be specified when <u>multiprogrammed</u> has been specified in the system section.

11. A specifier is a means of indicating a range of inclusive values to a particular item which is to act as a key item.

A.2.2.2 The Read Commands

Syntax

Format 1

```
read  {<type name-1>          }  [ for <specifier>] -
      {complete <type name>   }

[first value-1] [after <unique-specifier-1>] -

[last value-2] [before <unique-specifier-2>] -

into name-1 using name-2 {exclusively } <program> end read.
                         {concurrently}
```

Format 2

```
read <type-name-2> next value-3 into name-3 -

using name-4  {exclusively } .
              {concurrently}
```

Format 3

```
read last <type-name-3> into name-5 using name-6 {exclusively }.
                                                 {concurrently}
```

<program> is defined to be a sequence of host language instructions.

Semantics

1. Format 1 read makes available one or more occurrences of the type identified by type-name 1 which satisfy the conditions contained in the specifier. The type-name 1 identifies any class of record which has been defined. Read can only operate upon one index key-value at one time.

2. Only some of the records may be presented to the user. If value 1 is specified then only that number of records will be made available to the user provided there is sufficient to satisfy the value. If not, all records will be presented.

3. If the records, after some condition has been satisfied, are required, then unique-specifier 1 must be specified. First and after can be combined to yield value 1 records after unique-specifier 1 has been specified.

4. Similarly if records are required from the end of those to be realeased, last and before can be used. In all cases the manner of defining the last or first is by inspecting the direction of the link which joins the records together. If double-link is specified the order is that specified by the ascending or descending.

5. First last must not be specified for a tree.

6. Name 1 identifies the field into which each record will be passed for processing in turn. Name 2 is the basis or sub-basis to be used. Exclusively and concurrently have the same meaning as in write.

7. The program contains the host language instructions which will operate upon the fields contained in name 1.

8. End read signifies the end of these instructions so that the next record can be presented for processing. If the program jumps outside the read end read bracket it will not be possible to return, to continue processing.

9. Format 2 is used when a list is used to organise the tasking of personnel. Type-name 2 can only name a list.

10. Value 3 records are passed to the field identified by name 3. When the next similar read is issued, records will be passed from where the last read finished.

11. If the whole list is to be read then complete must be specified in format 1.

12. Format 3 is used, when a new element of a dictionary or table used for encoding is to be written, in order to find the last such written and thus the present highest value of the code field. The dictionary or table element is placed in name 5 field.

A.2.2.3 The Alter Command

Syntax

$$\underline{\text{alter}}\ \langle\text{type-name}\rangle\ \underline{\text{for}}\ \langle\text{unique-specifier}\rangle\ \underline{\text{from}}\ -$$

$$\left\| \text{name-1} = \begin{Bmatrix}\text{value-1}\\ \underline{\text{empty}}\end{Bmatrix}\ \underline{\text{to}} \begin{Bmatrix}\text{value-2}\\ \underline{\text{empty}}\end{Bmatrix}, \right\|\ -$$

$$\text{name-2} = \begin{Bmatrix}\text{value-3}\\ \underline{\text{empty}}\end{Bmatrix}\ \underline{\text{to}}\ \begin{Bmatrix}\text{value-4}\\ \underline{\text{empty}}\end{Bmatrix}\ \underline{\text{using}}\ \text{name-3}\ \begin{Bmatrix}\underline{\text{exclusively}}\\ \underline{\text{concurrently}}\end{Bmatrix}.$$

Semantics

1. Alter is used to change the value of an item which has already been written to the data base.

2. Type-name identifies the type which is to be altered and the unique-specifier indicates which record is to be altered. Only one occurrence of a type can be altered at one time but many items in that occurrence can be altered.

3. Names 1 and 2 identify the items which are to be altered.

4. Values 1 and 3 indicate the value of the items which were previously written.

5. Values 2 and 4 indicate the value to which the items are to be altered.

6. Empty is used when no value exists, or is to exist.

7. Name 3 indicates the basis or sub-basis to be used.

8. If exclusively is specified all outstanding higher priority commands will be completed and the alteration will then

take place. The users of the previous commands being informed of the imminent alteration.

9. If concurrently is specified commands of higher priority will be allowed to interrupt the alter but will be informed of the state.

A.2.2.4 The Delete Command

Syntax

```
delete <type-name> for <unique-specifier> -
using name-1 {exclusively } .
             {concurrently}
```

Semantics

1. The delete command will erase a record from the data base provided the record is not part of a group or chain.

2. Only one record can be deleted at one time. The record type is identified by type-name and the particular occurrence by the unique specifier.

3. Name 1 identifies the basis or sub-basis to be used. If exclusively is specified then the user will be the only one using the basis or sub-basis and the deletion will take place immediately. If concurrently is specified the command will be completed in the correct priority sequence, while other higher priority commands are allowed to proceed upon the record.

A.2.2.5 The Set Command

Syntax

$$\underline{\text{set}}\ \ \text{name-1},\ \ \text{name-2}\ \ [\underline{\text{for}}\ \langle\text{unique-specifier}\rangle]-$$

$$[\underline{\text{until}}\ \langle\text{condition}\rangle]\ \ \underline{\text{using}}\ \ \text{name-3}\ \ \left\{\begin{array}{l}\underline{\text{exclusively}}\\ \underline{\text{concurrently}}\end{array}\right\}\ .$$

<condition> is defined to be

$$[\underline{\text{not}}]\ \langle\text{specifier}\rangle\ \left\Vert\left\{\begin{array}{l}\underline{\text{and}}\\ \underline{\text{or}}\end{array}\right\}\ [\underline{\text{not}}]\ \langle\text{specifier}\rangle\right\Vert$$

In constructing a condition logical operator (and or) evaluation takes place from left to right, unless an expression is enclosed in parenthesis. An enclosed expression is evaluated first. When expressions are nested the most internal expression is evaluated first.

Semantics

1. Set initiates the movement of data from one storage device to another, by identifying flags indicated by names 1 and 2.

2. To specify which flag is indicated a unique specifier may be needed. It is not needed in the case of a list. All flag identifiers are unique and therefore the setting of one flag cannot initiate more than one data movement.

3. Names 1 and 2 identify flags which are qualified by the same unique specifier.

4. The basis or sub-basis to be used is identified by name 3. If exclusively is specified the data movement will take place immediately, and from then on the data will be made available from the new storage device. If concurrently is specified the data movement will take place when time

permits. The data being available from the old storage device right up to the time of data movement.

5. The amount of data transferred depends upon the location of the flag:

(i) Index All data belonging to the specified index value. If a sub-basis in use has restrictions on the pointer-list then only the data accessible from the sub-basis will be moved.

(ii) Structure The specified string of structures will be moved. If the structure is obtained through a pointer-array then two cases occur. First, if the key-item of the pointer-array is not specified then all structure strings will be transferred. Secondly, if one or more key item values are specified then all the indicated strings will be moved. Note: In the latter case the key item identifier will have to be repeated many times even though the item values could be written as a range as in a specifier.

6. The condition provides a means of specifying how much data is to be moved when the set is used for a flag defined in the on-lined or read when storage command.

A.2.2.6 The Find Command

Syntax

```
find from {<type-name>} all [for <condition>] using name-2 -
          {name-1     }

{saved in name-3        } end find
{into name-4 <program>  }
```

Semantics

1. Find is a more complex form of read, which always takes place concurrently. It is more complex in that logical expressions are allowed. However, only one record type can be processed at one time.

2. Name 2 identifies the basis or sub-basis to be used.

3. The records which satisfy the condition can be either processed by the program using the field identified by name 4 as working space, or the records can be saved in a file identified by name 3 for subsequent processing.

4. Name 1 can identify either a file in which data has been saved or a relationship. In the case of a relationship it is therefore possible for the find command to process more than one record type.

A.2.2.7 The Sort Command

Syntax

```
sort {<type-name>} [for <specifier>] using name-2 -
     {name-1     }

order by name-3 {ascending },  name-4 {ascending } -
                {descending}          {descending}

 {saved in name-5        }  end sort.
 {into name-6 <program>  }
```

Semantics

1. Sort re-orders a string of records identified by type-name or the contents of a saved file or a relationship identified by name 1.

2. Name 2 identifies the basis or sub-basis to be used. All processing will take place in the concurrently mode.

3. The order into which the records will be re-ordered is specified by names 3 and 4. The first item identified by name 3 will be used, subsequent items being used in case of equalities of value of name 3.

4. The order will be <u>saved</u> in a file identified by name 5.

5. The records will be processed in the new order from a field identified by name 6.

A.2.2.8 The Length Command

<u>Syntax</u>

<u>length</u> <u>of</u> name-1

<u>Semantics</u>

1. The length command makes available the number of elements or characters contained in a variable repeating group or item respectively.

A.2.3 Relational Commands

A.2.3.1 The Add Command

<u>Syntax</u>

add to ‖name-1 [for <unique-specifier-1>],‖ -
name-2 [for <unique-specifier-2>] .

<u>Semantics</u>

1. The add command is the only way in which a record occurrence can be added to a <u>group</u> or <u>chain</u> relationship which is identified by names 1 and 2.

2. The record added is the one which was last processed by any command except <u>delete</u>.

3. If the relationship needs specification names 1 and 2 may be qualified by the unique specifiers 1 and 2.

4. The basis or sub-basis of use is the one which was used to process the record initially.

A.2.3.2 The Erase Command

Syntax

```
erase from ||name-1 [for <unique-specifier-1>],|| -
name-2 [for <unique-specifier-2>].
```

Semantics

1. The erase command removes a record occurrence from the group or chain relations specified by names 1 and 2.

2. The record erased will be the one which was last processed by the process using the same basis or sub-basis for the erase as for the process.

A.2.3.3 The Follow Command

Syntax

```
follow name-1 [for <unique-specifier>] using name-2 -
 {into   name-3 [if <condition-1>] <program>            }  -
 {saving {those with <condition-2>}  in  name-4         }
 {       {all                     }                     }
[remember origin name-5] end follow.
```

Semantics

1. Follow allows a relationship to be processed in such a manner that the previous records which have been processed are always available.

2. The relationship is identified by name 1 and may be

qualified by the unique specifier.

3. The basis or sub-basis of use is identified by name 2. The processing is always carried out in the concurrently mode.

4. The records released can be processed immediately by the program or can be saved in a file for future processing.

5. The field used for immediate processing is identified by name 3.

6. The saved file is identified by name 4.

7. The conditions allow records to be selected for processing or saving.

8. Name 5 is an identifier by which it is possible to mark points in the processing so that returns can be made to these points.

A.2.3.4 The Return Command

Syntax

    return to name-1.

Semantics

1. The return command can only be issued from within a program of a follow command.

2. Processing will revert to the record which was processed when the specified origin was remembered. The origin is specified by name 1.

3. The return will be made to the last such origin remembered. If that is not the required origin further returns will have to be issued until the required origin is found.

A.2.3.5 The Retrace Command

Syntax

retrace to <condition> [remember origin name-1}.

Semantics

1. Retrace can only be issued from within the confines of the program in a follow command.

2. The records which had been previously processed by a follow command are released in the reverse order until the condition is satisfied.

3. The origin of the retrace can be remembered so that an immediate return can be made.

A.2.3.6 The Give Command

Syntax

give name-1 [for <condition>] using name-2 -
into name-3 <program> end give.

Semantics

1. The give command releases records of a relation identified by name 1, but retains no memory of previous record occurrences processed.

2. Give is also used to make available the records of a saved file.

3. Name 2 identifies the basis or sub-basis to be used in a concurrent mode only.

4. Name 3 identifies the field upon which the program will operate.

A.2.3.7 The Create Command

Syntax

   create name-1 using name-2.

Semantics

1. The create command allows relationships to be created. The relationship is identified by name 1 and can only be used in the basis or sub-basis identified by name 2.

A.2.3.8 The Destroy Command

Syntax

   destroy name-1 using name-2.

Semantics

1. The destroy command removes created relationships and saved files.

2. The destruction will only be allowed if the sub-basis or basis identified by name 2 matches the one which created the file or relation.

A.2.3.9 The Membership Commands

Syntax

1. current membership ‖name-1,‖ name-2.

2. what membership

Semantics

1. The membership commands are used to find out to which relationships the record being processed belongs.

2. Format 1 enquires whether the record belongs to relationships identified by names 1 and 2.

3. Format 2 obtains the identifier of all the relationships which the record belongs to.

A.2.4. Optimising Statistical and Other Commands

A.2.4.1 The Transfer Command

Syntax

$\underline{\text{transfer}}$ name-1 $\left[\,\underline{\text{for}}\ \langle\text{condition}\rangle\right]$ $\left\{\begin{matrix}\underline{\text{to}}\\ \underline{\text{from}}\end{matrix}\right\}$ name-2 -

$\underline{\text{using}}$ name-3.

Semantics

1. The transfer command makes data available to and from the data base in a different format for processing, outside the control of the data base.

2. Name 1 identifies the transfer-structure to be used.

3. Name 2 identifies the file outside the control of the data base.

4. Name 3 identifies the basis or sub-basis to be used.

5. If to is selected, data is transferred from the data base to the file, and vice versa if from is selected.

A.2.4.2. The Move Command

Syntax

$\underline{\text{move}}$ name-1 = value-1 $\underline{\text{to}}$ literal-1 -

$\left[\underline{\text{employing}}\ \|\text{name-2,}\|\ \text{name-3}\right]$ $\underline{\text{using}}$ name-4.

Semantics

1. The move command takes data from one computer and places it on another computer and can only be used in a multi-computer system. The transfer takes place as though from

tape to tape and therefore any data to be moved must have already been placed on tape.

2. Name 1 is the identifier of an accessed by identifier of an index, thus implying that only structures can be moved.

3. Literal 1 identifies the computer to which the data will be moved.

4. If any changes are needed in format then names 2 and 3 identify transfer-structures.

5. Name 4 identifies the basis or sub-basis to be used.

Note: If in a sub-basis not all pointers in a pointer-list are used then only the pointers available in that sub-basis will be used, thus the data will be split between two computers.

6. The amount of data transferred will be all that to which a pointer points.

7. Both the transfer and move commands operate concurrently, that is, any outstanding higher priority commands for any elements to be read are completed before the move or transfer takes place, the users being informed as to this state.
In the case of move this should provide no difficulties as a flag will have been set so that the data can be transferred to tape.

8. When only part of the data is moved a special move structure is pointed to by the pointers which pointed to the moved data. This structure holds details of when, and to where the move took place.

A.2.4.3. The Exhibit Command

Syntax

The exhibit command occurs in many different forms each of which is listed and the semantics of each is explained as a whole.

```
exhibit packing densities.
exhibit time of name-1.
exhibit number of accesses to name-2 [completely].
exhibit space of name-3.
exhibit balance of name-4 [for <unique-specifier>].
exhibit count of name-5 for name-6 = value-1.
exhibit frequency count for {||value-2,|| value-3} -
                            {all                 }
        {level } of name-7.
        {levels}
```

Semantics

Exhibit packing densities make available the following pairs of information; the total number of storage lengths and the total number of those storage lengths which are filled with data as described in Chapter 6.

Exhibit time makes available two details about the program identified by name 1, these are the total elapsed time of the program from starting to completion and the amount of time that the central processor was used by the program. This command must be made in another program which is executed immediately before the program name 1, by an editing section command.

Similarly exhibit space must be executed before the program, identified by name 3, is executed. This form of exhibit makes available the total number of characters which the program name 3 used in order to execute.

Exhibit number must also be placed in a preceding editing program but in this case name 2 identifies a structure or sub-structure or list.

Exhibit number makes available the number of times a command was issued for the required name 2. Each command will be assigned the number of times it was used.

If completely is specified then for each occurrence of any command being issued a count will be made available of how many elements had to be read before the required one or ones were found.

Name 4 is the identifier of a tree suitably qualified by a unique specifier so that only one tree is defined. Exhibit balance makes available the number of elements to the 'left' and to the 'right' of the initial node.

Name 5 identifies a dictionary for which the frequency relationship has been specified. The value given to the accessed by identifier name 6 will define one element of that dictionary. Exhibit count makes available the counter, which is held with every element of such a dictionary.

Name 4 identifies a similar dictionary, that is, one for which frequency is specified. In order to get the range of values of count for which an element will be held at a particular level within the dictionary, exhibit frequency either makes all ranges available, or the ranges of the levels specified by values 2 and 3.

A.2.4.4 The Make Command

Syntax

```
make total units ||value-1, value-2;|| value-3, value-4.
make initial node value-5 of name-1 -
    [for <unique-specifier-1>] .
make count of name-2 for <unique-specifier-2> -
    equal to value-6.
```

Semantics

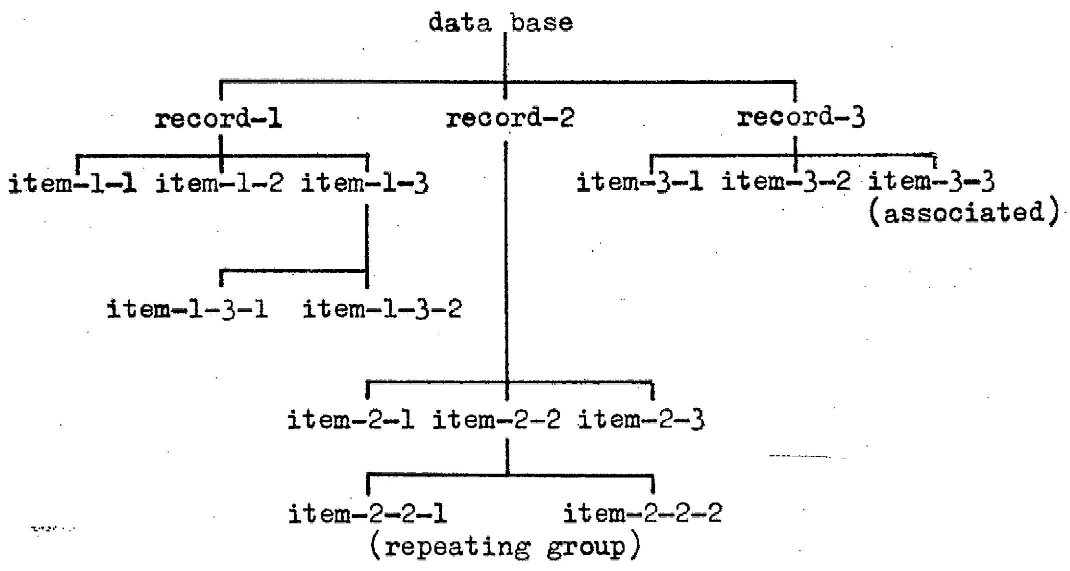
The make total command assigns new values to the total number of storage units assigned to each length of storage space. Values 1 and 3 are the length of storage space, values 2 and 4 are the total number of units assigned to that length.

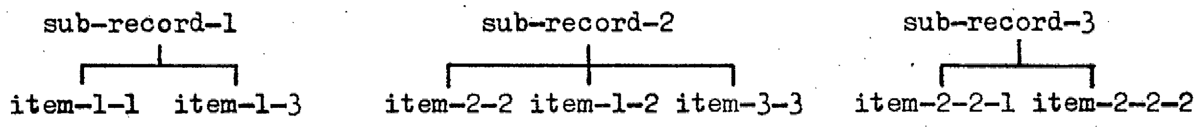
If a tree identified by name 1, possibly qualified by the unique specifier, is found to be unbalanced the initial node value can be changed by the make initial command. The new value of the initial node is specified by value 5.

If an element of a dictionary is known to be very variable in the frequency of use, namely it has periods of great use and periods of little use and these periods of great use are too short for the normal period re-organisation to have any effect, the make count command can be used. Name 2 identifies the dictionary, the element of which is specified by the unique specifier 2. The count associated with that element can be artificially altered to value 6. Thus effecting a change before the element comes into a period of great or little use.

APPENDIX 3

A data base is to be defined from the following diagram. Only the generating section is specified. Other information has been added to make the definition complete.





```
basis the-only-basis
  master index main-index contains
    fixed main-key length 6
    pointer main-pointer points to a-p-list mechanism is direct
   connections are
    accessed by main-key ascending
   storage is
    kept forever on exchangeable-disc.

index another-index contains
    fixed another-index-key length 6
    pointer another-pointer points to a-p-list mechanism is direct
   connections are
    accessed by another-index-key descending
   storage is
    kept forever on fixed-disc.

pointer-list a-p-list contains
    pointer pointer-1 points to record-1 mechanism is direct
    pointer pointer-2 points to record-2 mechanism is direct
    pointer pointer-3 points to record-3 mechanism is direct
   storage is
    kept forever on exchangeable-disc.

structure record-1 privacy 1 to 6 contains
    variable item-1-1 privacy 1,3 to 6
    fixed item-1-2 length 4
  fixed item-1-3 length 1 (fixed item-1-3-1 length 2 privacy 5
                           variable item-1-3-2)
 connections are
  double-link d-1-1 by item-1-2 ascending mechanism is direct
  link 1-1 by item-1-1 descending mechanism is displacement
  double-link d-1-1-A by item-1-3-1 ascending mechanism is direct
  member chain first-chain mechanism is displacement privacy 2,3 to 5; -
    second-chain mechanism is direct
 storage is
  kept until flag-1 on fixed-disc then
  kept until flag-2 on exchangeable-disc then
  kept forever on tape
  on-lined when flag-3 to main-index, flag-4 to another-index
  read when flag-5 returned immediately.

structure record-2 privacy 3 to 10 contains
  fixed item-2-1 length 8 computational privacy 6 to 10
 variable item-2-2 (fixed item-2-2-1 length 4
                    fixed item-2-2-2 length 2)
  fixed item-2-3 length 25 privacy 8, 10
 connections are
  accessed by item-2-2-1 ascending, item-2-2-2 descending
  link 1-2 by item-2-3 ascending mechanism is direct
  start chain second-chain mechanism is direct
  member chain first-chain mechanism is displacement
  group the-only-group mechanism is displacement
  storage is
   kept for 10 days on drum then
   kept forever on tape
   on-lined when flag-6 to main-index
   read when flag-7 returned after 6 days.

structure record-3 privacy 1 to 10 contains
   variable item-3-1
   fixed item-3-2 length 4 computational
   associate item-3-3 with dict-item-1, dict-item-2 of
     the-dictionary for dict-item-3 = item-3-2
  connections are
   double-link d-1-3 by item-3-1 ascending mechanism is direct
   link 1-3 by item-3-2 descending mechanism is displacement
   start chain first-chain mechanism is displacement
   member chain second-chain mechanism is direct
   group the-only-group mechanism is displacement
  storage is
   kept for 14 days or until flag-8 on fixed-disc then
   kept forever on tape
   on-lined when flag-9 to another-index, flag-10 to main-index.

sub-structure sub-record-1 privacy 5 contains
   item-1-1 of record-1 left justified in 15 characters space extension
   item-1-2 of record-1.
```

```
sub-structure sub-record-2 contains

    item-2-2 of record-2

    item-1-2 of record-1 to computational

    item-3-3 of record-3.


sub-structure sub-record-3 privacy write 6 to 10

    item-2-2-1 of record-2

    item-2-2-2 of record-2.


dictionary the-dictionary privacy write 5 contains

    fixed dict-item-1 length 25

    fixed dict-item-2 length 8

    fixed dict-item-3 length 4 computational

    variable dict-item-4

   connections are

    accessed by dict-item-2 ascending mechanism is direct

    accessed by dict-item-3 descending mechanism is direct

    frequency after 60 days every 30 days 50 at level 1, 100 at level 2,

      300 at level 3, rest at level 4

   storage is

    kept forever on

      drum level 1

      fixed-disc level 2

      exchangeable-disc level 3

      tape level 4.
```

sub-basis first-sub-basis privacy 5 contains

    master-index; the-dictionary; record-3; sub-record-3.

sub-basis second-sub-basis contains

    another-index; record-3; record-1.

sub-basis third-sub-basis contains

    master-index; another-index; record-1; record-2;

      not item-3-3 of record-3.

Points to Note

1. The third-sub-basis explicitly excludes item-3-3 and the second-sub-basis implicitly excludes item-3-3 because the-dictionary is not contained in the second-sub-basis.
2. Due to the privacy restrictions on 1-2 effected by the privacy restrictions of item-2-3, item-2-2 is only available from sub-record-3, unless the user has a privacy coding of 8 or 10.
3. Items used as key values for dictionary items must match the definition of the key item in the-dictionary.
4. On-lined and read when can only be used when the data is archived to tape.
5. When on-lined and read when are used if no index is specified, the one designated master index will be assumed.
6. The item-2-2, which is a repeating group, is ordered by the accessed by relation given in the connections part of the definition.

Key to symbols used in the following data maps

A	ascending
AB	accessed by
AS	associate
char	character
comp	computational
D	direct
DE	descending
DI	displacement
DL	double-link
DY	dictionary
E	kept forever
F	fixed
FG	fixed length repeating group
FGE	entry in a fixed length repeating group
FR	frequency
G	group
I	index
KF	kept for
KFU	kept for or until
KU	kept until
LK	link
MC	member chain
MI	master index
OL	on-lined
P	pointer
PL	pointer-list
R	read when
S	structure
SC	start chain
SS	sub-structure
SB	sub-basis
V	variable
VG	variable length repeating group
VGE	entry in a variable length repeating group

class	identifier or device	length	form	number assoc*	item-name flag-name	key-name	key-value	table-name	privacy read	write	erase	add
MI	main-index											
E	e-disc											
AB	main-key											
F	main-key	6	char									
P	main-pointer		D		a-p-list							
I	another-index											
E	fixed-disc											
AB	another-index-key											
F	another-index-key	6	char									
P	another-pointer		D		a-p-list							
PL	a-p-list											
E	e-disc											
P	pointer-1		D		record-1							
P	pointer-2		D		record-2							
P	pointer-3		D		record-3							

* associated

class	identifier or device	length	form	number assoc*	item-name flag-name	key-name	key-value	table-name	privacy			
									read	write	erase	add
S	record-1								1 to 6	1 to 6	1 to 6	1 to 6
KU	fixed-disc				flag-1							
KU	e-disc				flag-2							
E	tape											
OL	main-index				flag-3							
OL	another-index				flag-4							
R	main-index	0			flag-5							
DL	d-1-1		D/A		item-1-2							
LK	1-1		DI/DE		item-1-1							
DL	d-1-1-A		D/A		item-1-3-1							
MC	first-chain		DI						2,3 to 5	2,3 to 5	2,3 to 5	2,3 to 5
MC	second-chain		D									
V	item-1-1								1,3 to 6	1,3 to 6	1,3 to 6	1,3 to 6
F	item-1-2	4	char									
FG	item-1-3	1	char									
FGE	item-1-3-1	2	char						5	5	5	5
VGE	item-1-3-2		char									

class	identifier or device	length	form	number assoc*	item-name flag-name	key-name	key-value	table-name	privacy read	write	erase	add
S	record-2								3 to 10	3 to 10	3 to 10	3 to 10
KF	drum	10										
E	tape											
OL	main-index				flag-6							
R	main-index	7			flag-7							
AB	item-2-2-1		A									
AB	item-2-2-2		DE									
LK	1-2		D/A		item-2-3							
SC	second-chain		D									
MC	first-chain		DI									
G	the-only-group		DI									
F	item-2-1	8	comp						6 to 10	6 to 10	6 to 10	6 to 10
VG	item-2-2		char									
FGE	item-2-2-1	4	char									
FGE	item-2-2-2	2	char									
F	item-2-3	25	char						8,10	8,10	8,10	8,10

class	identifier or device	length	form	number assoc*	item-name flag-name	key-name	key-value	table-name	privacy read	write	erase	add
S	record-3								1 to 10	1 to 10	1 to 10	1 to 10
KFU	fixed-disc	14		1	flag-8							
E	tape											
OL	another-index				flag-9							
OL	main-index				flag-10							
DL	d-1-3		D/A		item-3-1							
L	1-3		DI/DE		item-3-2							
SC	first-chain		DI									
MC	second-chain		D									
G	the-only-group		DI									
V	item-3-1		char									
F	item-3-2	4	comp									
AS	item-3-3			2	dict-item-1							
					dict-item-2	dict-item-3	item-3-2	the-dictionary				

class	identifier or device	length	form	number assoc*	item-name flag-name	key-name	key-value	table-name	privacy			
									read	write	erase	add
DY	the-dictionary									5		
E	drum	1		4								
	fixed-disc	2										
	e-disc	3										
	tape	4										
AB	dict-item-2		DI/A									
AB	dict-item-3		DI/DE									
FR		60		2								
		30										
		50		4								
		100										
		300										
		Rest										
F	dict-item-1	25	char									
F	dict-item-2	8	char									
F	dict-item-3	4	comp									
V	dict-item-4		char									

class	item identifier	type identifier	form	size	exten-sion	justi-fied	privacy			
							read	write	erase	add
SS	sub-record-1						5	5	5	5
	item-1-1	record-1		15	space	left				
	item-1-2	record-1								
SS	sub-record-2									
	item-2-2	record-2								
	item-1-2	record-1	comp							
	item-3-3	record-3								
SS	sub-record-3							6 to 10		
	item-2-2-1	record-2								
	item-2-2-2	record-2								
SB	first-sub-basis						5	5	5	5
		master-index								
		the-dictionary								
		record-3								
		sub-record-3								