A RESTRUCTURING MECHANISM FOR A CODASYL-TYPE DATA BASE

JAMES CARDEN

Ph.D.
University of Edinburgh
May 1983

Abstract

Data Base Management Systems are observed to operate in distinct environments within organisations. These environments are classified as 'Centralised' and 'Devolved' and different types of data base management systems are considered as more appropriate to each. The centralised data base is closely associated with an evolutionary model of the entire organisation whereas each devolved data base is a tool used by an individual or group to analyse information necessary to that person's function within the organisation.

The requirements for a Restructuring Mechanism to allow a centralised data base to be altered in structure to reflect alterations in the organisation and enhancements to the model are identified. Strategies to allow such a mechanism to operate concurrently with normal application program access to the data base are postulated. In particular Open Dynamic and Closed Dynamic restructuring techniques are described. Although no evidence has been found of a similar restructuring mechanism in published research or as implemented in a proprietary data base management system its relevance to other types of data base change which have been described elsewhere are considered.

Sixteen primitive restructuring tasks are described in detail and strategies for their execution are outlined. Application programs are classified as transparent or opaque to each task. The effect of each task on the routines processing data base access by application programs is also examined.

To illustrate the effectiveness of the restructuring tasks an implementation of a CODASYL-type Data Base Management System has been developed on the Edinburgh Multi Access System (EMAS). The implementation is then used to run application programs operating on a data base modelling a British Bank. Various types of restructurings may be carried out on the data base using Open or Closed Dynamic strategies.

Conclusions are drawn that a Restructuring Mechanism is an essential ingredient for a Data Base Management System to allow the data base to reflect the ever-changing structure of an organisational data model. The mechanism proposed in the thesis is considered to be functional, convenient and efficient for the population of users it is designed to serve - the data processing professionals.

Acknowledgements.

I would like to thank my wife Carol and my children Steven, Neil and Claire for surviving the rival attentions of my books and computer terminal for so many evenings over the last seven years.

Thanks are also due to my supervisors Dr. Geoff Stacey and Professor Sidney Michaelson.

Finally, I would like to thank my employers, the Bank of Scotland, for sponsoring me in this work.

Declaration.

This thesis has been composed by me.

It describes my own work carried out between
May 1976 and May 1983.

Index.

Chapter 1 - Introduction.

The Evolution of Data Base Management Systems.

Data processing made a dramatic impact on the operational procedures of industry and commerce during the 1950's and 1960's. Using fairly primitive techniques of data capture on unit record devices, computer systems were implemented to update master files of data held on magnetic tape by processing against the captured transactions. Suitable control and statistical reports were printed and distributed to appropriate users within the organisation.

The art of Systems Analysis, as it came to be known, was a new professional discipline which came into use within large organisations. The functions and procedures used by the organisation to achieve its commercial objectives would be examined in detail with a view to devising even better methods by performing the same or superior tasks using the new technology. The radical approach of the computer-oriented systems analysts (whose training encouraged them to have no preconceptions as to the value of even the most firmly established practices) in many cases led not only to the introduction of well defined procedures but also to the removal of unnecessary operations. A wide variety of the routine functions within large organisations (at that time only the largest could invest in the expensive computer equipment) were made more efficient in this way but by the start of the 1970s the proliferation of such computer-based systems within organisations began to reveal some of the limitations of this piecemeal approach.

Although the individual computer systems normally served their original purpose admirably, they tended to be rather inflexible when modified to serve other functions. Even if it were possible to design a system which was flexible enough to handle all future requirements it is unlikely that even the most innovative Systems Analyst would have the precognition to foresee all eventualities.

The development of discrete systems in isolation from each other also often led to the same data being maintained in different ways and this in turn inevitably led to inconsistencies between the output of different systems which should otherwise have been compatible.

Having identified the problems, the data processing community has searched for solutions in two general directions:

The computer programs themselves have been recognised as contributing to the inflexibility by being difficult to maintain by anyone except the original author. Particularly since computers have become faster and cheaper, and consequently program efficiency less of a constraint, this situation has been remedied by standardising programming practice to make programs more comprehensible and less error prone. Various methodologies of Structured Programming have been advocated and programming languages have been refined over the years.

Programs are not the only cause of system inflexibility and it has long been a goal to migrate from a set of disjoint computer systems to an integrated set of systems using the organisation's data as a common resource.

Once again various methodologies of Structured Data have been advocated and techniques have been developed for shared access to the data. The relative reduction in cost and increased storage capacity of direct access devices have contributed significantly to this development. Although magnetic discs have been available for many years their relatively high cost in comparison to magnetic tape led computer installations to use them prudently. Typically they would be used for specialised purposes such as operating system software, program libraries and sort workfiles with the storage of application data being restricted to the use of small 'index' files. In many cases the use of the medium has now expanded to such an extent that it is common to consider holding 'master' files of many hundreds of millions of characters on disc. Tape is often considered more as an archival medium. The inherent ability of disc to provide access to particular data records without having to examine the other records held on file made the medium particularly suitable for data capture and retrieval applications using on-line remote terminals. Such systems often have advantages over batch processing systems (because of the absence of input documents and printed reports) and have risen to prominence in recent years.

Definition of Data Base.

The view of an organisation's data as a shared resource has become known to the lay community of computer users as the organisation's Data Base. Although this definition of 'Data Base' may seem rather imprecise it is difficult to improve

on it. Indeed the term is now synonymous with the terms 'file' and 'collection of files'.
Data processing practitioners have allocated more specific objectives to the term which have implied a more narrow definition but differences in emphasis on individual objectives have led in the past to seemingly different definitions.

Rather than attempting to formulate a precise definition we will adopt an empirical approach by considering four generally agreed objectives of a Data Base.

1. To remove the difficulties of fragmented data caused by the development of ad-hoc computer systems.
This implies that some mechanism for a single central description of the data must exist and that this description is available either explicitly or implicitly for all applications to use when necessary. As a by-product it is implied that data bases provide centralised capabilities for control of quality and integrity of data while maintaining suitable privacy constraints (FRY & SIBLEY).

2. To provide a simple yet powerful mechanism to allow applications to interact with the data base. In particular to remove the bottleneck imposed by traditional systems in their inability to provide swift answers to unforeseen enquiries. (CODD & DATE).

3. To isolate the programs interacting with the database

from technological inovation in the form of improved data storage devices with different characteristics, different access methods and different techniques for linking data to reflect changing access patterns. This objective is termed providing resilience to reorganisation.

4. To isolate the programs interacting with the data base from changes in the description of the actual elements making up that data base and of changes in the relationships between these elements. Such changes are seen as inevitable as a result of corresponding changes in the organisation. This objective is termed providing resilience to restructuring.

The last two of these objectives have been particularly confused in the past since they have been described under the general heading of Data Independence. Some authors have attempted to bring out the distinction by using the terms Physical Data Independence and Logical Data Independence (FRY & SIBLEY) (LEFKOVITZ).

Some of these objectives have been addressed outwith the area normally considered to be the province of Data Base: Many organisations have examined the interrelationships between their computer systems and attempted to rationalise them by imposing rigid internal standards on the way in which programs are permitted to interact with data on file. Techniques for the standardisation of methods of file description have been introduced. In particular these are of value during the system design process to enable analysts to ascertain whether the data they wish to use is already collected by some other system.

Disc file access methods have been developed which allow concurrent access by more than one application and these are now fairly common (eg IBM-VSAM).

Operating systems have relegated the definition of device addresses, disc and tape serial numbers, record and block length and other device-dependent information to Job Control Language and Systems Catalogues.

Program libraries have become sophisticated, especially by providing facilities for straightforward methods of amending program source code via on-line terminals. Together with the widespread adoption of modular programming techniques it has become possible to localise the definition of data on existing files (and their input/output access routines) to single modules accessible by all programs (e.g. PANVALET). Data Dictionary packages are now coming into use which provide cross-indices between Programs, Files and Job Control Language Procedures in conjunction with statistics on data access patterns for conventional application programs as well as programs running under a Data Base Management System /XEPHON/.

Objectives of the Thesis.

The remaining chapters of this thesis address the following objectives:-

1. To observe that there are two distinct environments in which data base management systems may be used. The "Centralised" environment where the data base is a common resource providing a universally applicable model of the organisation and the "Devolved" environment where the data base reflects an information model held to be correct by some group of individuals within the organisation. Further, it will be noted that the different requirements of each of these environments may be better serviced by different types of data base management systems.

2. To consider that changes in the logical structure of the data base are likely to have more widespread implications throughout the organisation in the centralised environment than in the devolved environment because the same structure must serve a multiplicity of users. Consequently, although facilities to alter the structure of the data will be required for both centralised and devolved data bases, they will often be of only limited value for a devolved data base. On the other hand, the structure of the the centralised data base for many organisations can be expected to be relatively volatile, and a sophisticated restructuring mechanism will be essential to allow the data model to

evolve.

3. To propose a restructuring mechanism for a CODASYL data base management system. This type of network data base is considered to be a good example of a data base management system suitable for use by the professional data processing practitioners who are seen as retaining responsibility for maintenance of the centralised data base.

4. To demonstrate that a restructuring mechanism along the lines proposed can be implemented whereby alterations can be made to application programs to reflect the revised structure, where necessary, without interrupting the availability of these programs to service the demands of their end users.

Chapter 2 covers the process of accumulating a data structure to adequately model the organisation maintaining the data base. Not only will this highlight the differences between centralised and devolved data bases but it will also identify the iterative nature of the data model of the single centralised data base of the organisation.

Chapter 3 examines some types of data base management systems and discusses their relevance to either centralised or devolved data base environments.

Chapter 4 describes how researchers and implementors of data base management systems have considered the problem of allowing the structure of the data to change.

Chapter 5 considers how the structure of centralised data bases will alter through time. In particular it shows how changes to application systems will dictate changes to the logical structure of the data on which they operate. As a result, other applications must be in a position to react to the structural change.

Chapter 6 looks at how the network data base structure proposed by the CODASYL Data Base Task Group could support such a restructuring mechanism.

Chapter 7 gives details of the primitive restructuring tasks which are relevant to the clauses defining the CODASYL data base structure in its Data Definition Language.

Chapter 8 describes the implementation of a CODASYL-type data base management system and associated restructuring mechanism on the EMAS multi-access system.

Chapter 9 summarises how the proposed restructuring mechanism meets the objectives set out above. It also identifies areas where further research will be required.

Appendices are used to illustrate how the restructuring mechanism could be utilised for a centralised data base maintained by a British Bank. Several application programs and restructuring tasks have been written for the EMAS implementation in order to achieve this.

Chapter 2 - Data Analysis and Data Modelling.

Large organisations can be viewed in many different ways and it is often difficult to provide a single all-purpose picture of the entire enterprise. As business organisations in particular have become more sophisticated, efficiency has dictated that formal modelling techniques be devised. Models have been especially useful in areas of Management Services such as Operational Research, Organisation and Methods and Work Study and in many cases they have been adopted as the tools of the Systems Analyst.

Existing Modelling Techniques.

It is interesting to examine three typical modeling techniques each of which is designed to present a picture of the organisation in some way:

a) Organisational Chart.

This would be likely to show a pyramidal reporting structure within the organisation with individuals (or classes of individuals) shown as nodes on a hierarchical diagram. Other cross-hierarchy relationships denoted by additional links between nodes would reflect responsibilities outside line management. Relationships with individuals and other organisations in the "outside world" are also identified in such charts.

This model would mainly be of use in Personnel Departments where it could be viewed in conjunction with Job Descriptions and Staffing Level values to provide a mechanism for evaluating possible changes in working practice.

b) Flowcharts.

Prior to their use in describing the logic of computer systems and programs, flowcharts were (and still are) used to describe the transmission of information throughout the organisation and to and from the outside world.

Repetitive tasks (which are particularly suitable for flowcharting) were an early target for computerisation and systems analysts have often worked closely with O&M/Work-Study practitioners in this area. In many organisations virtually all the routine work interacts with computer systems in some way.

c) Critical Path Analysis Diagrams.

A Critical Path Analysis Diagram shows interrelationships between different activities carried out within the organisation in order to achieve some particular complex task.

There are numerous other models within organisations which do not have such an obvious visual manifestation. For example the profitability of a commercial organisation will be governed by its ability to plan its business and monitor its actual execution against the plan. The current plan is therefore a vital part of the organisational model as are such techniques as Management Accounting which are designed to measure actual performance against the plan.

Those models which do have a visual manifestation assist the observer in assimilating some aspect of the organisation. It is of interest to note also that there is a common feature in the preparation of such models in that the architect of the model must firstly identify discrete types of elements on which data is known or can be collected. These provide the nodes for the model. The differences between the models are found in the criteria used to establish relationships between these elements. By analogy various diagrams of the human body can be drawn showing blood circulation, central nervous system, skeleton etc. Just as a physician will choose to examine one of these diagrams of the body for some purpose he has in mind, one or other of the models can be used when it is required to consider the organisation from a particular viewpoint.

The distinction between the models and the organisation itself is one of a one-to-many mapping where each element in the model (often called an Entity) represents a number of actual occurrences of that classification of object within the organisation. Each individual object (such as each nut and bolt or each member of staff) will of course be unique but the generic term is sufficient to provide a meaningful model. Once it has been seen fit to define the existence of a class of objects in the model it may be inferred that each actual object has some properties which may be quantified in some way. The term Attribute has been used to describe such properties of entities.

For example an organisational chart of a Bank might be as follows:

```
General Manager
       :
       :
District Manager
       :
       :
Branch Manager
       :
       :
     Staff
```

Thus the single entry on the chart of "Bank Manager" would convey the impression of the few hundred people in control of the bank's branches. Equally (although not necessarily stated on the chart) the entity Bank Manager would have Attributes of Branch Name, Manager's Name, Lending Limit etc.

A General Purpose Data Model of an organisation must therefore provide for recording not only attributes which are associated with particular entities but also details of the patterns of associations between different occurrences of entities (including different occurrences of the same entity).

Large organisations tend to be complex evolutionary organisms and the task of preparing a model which contains all entity interrelationships is often formidable - thus the advantages of only considering particular classes of interrelationship as in the Organisational Chart, Flowchart and Critical Path Analysis diagram. Rather the preparation of the entire model is an iterative process with the possibility that a totally comprehensive model can never be achieved /FLORY and KOULOUDJIAN/.

Organisations differ widely in both size and complexity and it may be that there should be a difference in emphasis on different aspects of the general purpose model to take advantage of such differences. Thus for example the Bank is a large organisation with perhaps hundreds of branches and millions of accounts but nevertheless with relatively few entities and a correspondingly straightforward model. On the other hand a model of a manufacturing organisation even with a limited product-line might be complex and constantly changing to reflect a changing marketplace. In some circumstances time can be a fundamental consideration for all entities while in others the "current" situation is all that is required /SNUDGREN-75/.

Data models of different organisations will also vary significantly in the stability of the values of attributes in particular occurrences of entities. For example one could contrast a model of a library where once information on a book has been gathered it is unlikely to change (unless it is incorrect) against a model of a bank where the balance of an account is constantly altering to reflect the effect of financial transactions. Further the patterns of retrieval can be significantly different; where a book borrower may base his choice on Author, Title, Subject Matter or some more obscure criterion, in other environments (such as the bank) retrieval is often a straightforward matter based on a unique identification /KAM & ULLMAN/. For example the account number is all that is required in to obtain a bank balance and in most circumstances this would either be known or encoded on a plastic card.

The Three/Four Schema Data Model.

The process of developing a formal model of an organisation has been studied by several researchers. A significant contribution was made in 1975 by the report of the ANSI/SPARC organisation on the architecture of data base management systems /YORMARK/. In the present context the report proposed that a Conceptual Schema should exist which was maintained by a person or group known as an Enterprise Administrator. To emphasise the magnitude of the task of preparing and maintaining the schema as a true model of the organisation it might have been better to use the term 'Enterprise Analyst' to describe this individual or group of individuals. Without attempting to standardise the contents of this schema the report identified interfaces between it and more application-oriented and data-storage-oriented definitions in the form of an External Schema and an Internal Schema maintained by an Application Administrator and Data Base Administrator respectively. The personalisation of these functions by the allocation of a name reminiscent of a job title is most significant. It can be argued that by doing so it can be tacitly assumed that for these tasks to be performed well new skills will be required by the individuals performing the functions. It may be that many of the skills are a generalisation of those already possessed by the Systems Analysts but inevitably some will be entirely new, particularly in the area of

'Enterprise Analysis'.

The idea of a Conceptual Schema in particular has been developed in a report by an ISO Technical Working Party published in 1982 /ISO/. This report establishes the aims of a Conceptual Schema as :

(a) To describe the "Universe of Discourse" as a model of the entire enterprise in a manner suitable for easy interpretation. It follows that this model may not necessarily be concerned with the constructs of any particular implementation of a computerised DBMS.

(b) To control the information base used by computer systems.

The existing types of Data Base Management Systems are considered by the Report as somewhat inadequate to serve the first purpose of the Conceptual Schema but sufficient in many respects for the second.

A 'Four-Schema' philosophy is therefore suggested where the Conceptual Schema is written in a suitable language (for which the report draws up some basic requirements) and this is then subjected to a (manual) conversion to become the Data Base Schema in the terms supported by the Data Base Management System used by the organisation.

Preparation Of The Data Model.

Many approaches to the development of a methodology for establishing the data model have been advocated. Often these are extensions and standardisations of the established techniques for data analysis and fact recording developed over the previous two decades /JACKSON/.

Data Analysis is not an art practiced by the Systems Analyst in isolation and there must be an ongoing involvement by each end user such that a mutually acceptable view is achieved between him and the analyst rather than an artificial structure being imposed /SNUDGREN-78/.

The importance of information flow of the various processes operating on the data to the structure of the model has been highlighted in such techniques as User Task Analysis and Information System Architecture /IBM-UTAS/,/IBM-DA/.

Manifestations of the Data Model.

Diagramatic techniques have been devised to enable the Enterprise Administrator to assimilate the model of an organisation which he is in the process of creating and to allow analysts to judge the impact of proposed new systems and changes to existing systems. These provide a pictorial representation of the most general system of entities and the relationships between them /eg CHEN/. In particular they allow relationships to be represented where one element in one class may relate to many elements in another class where each of these may relate to many elements in the first class (ie a many-to-many mapping). Like the flowcharts and other diagrams referred to previously the manifestation of the data model as a chart does not reference individual occurrences of entities but uses generic terms for the various classes of entity and relationship.

Non-Diagramatic descriptions of the Data Model have also been proposed which use linguistic syntax to specify not only the structure of the model but also all other constraints which apply to it. The objectives of such representations are probably more oriented towards providing a precise description of the model than a representation which can be assimilated easily so that a particular area of interest can be readily identified /SHIPMAN/.

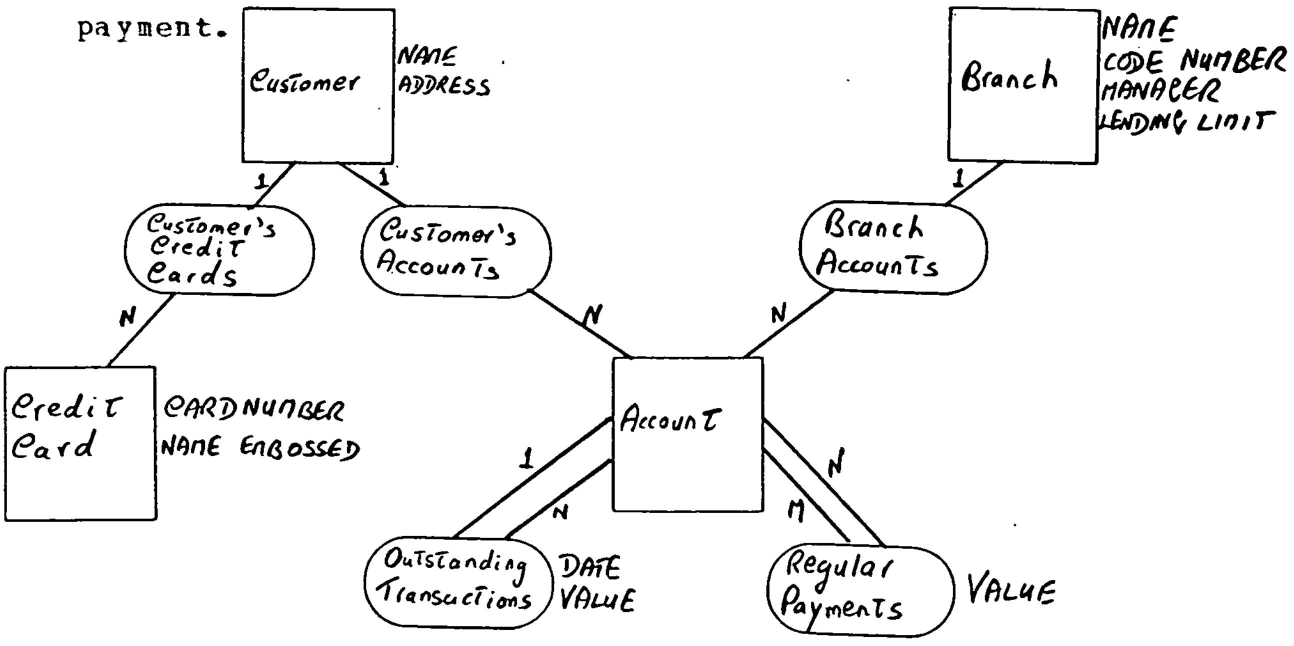
For example a simplified data model of a bank as shown below would illustrate that there were entities Customer, Branch, Account and Credit Card and these were related by the following associations:

Customer's Credit Cards - Where a customer could have zero or more credit cards (eg where the customer was a business or husband/wife)

Customer's Accounts - Where a customer may have one or more accounts at one or more branches.

Branch's Accounts - Where a branch may have one or more accounts of which some may belong to the same customer.

Regular Payments - Where funds are transferred between one account and another on a regular basis. The same account may be the payee of more than one payment and similarly the same account may be the beneficiary of more than one payment.



The effort required to produce a generalised data model in many organisations would be significant and there would be a constant requirement for refinement to cater for changes in the organisation and better perception of the accuracy of the model. In addition to the chart giving the structure of the data model there is often a more detailed manifestation in the shape of a Data Dictionary which provides a vehicle for recording all information known about the organisation's information - the Meta-Database. As well as describing all attributes of each entity in terms of their meaning, format and value range etc the Data Dictionary would also describe how attributes relate to other attributes, how entities relate to other entities, how application programs relate to both entities and attributes and how entities are distributed in terms of numbers of occurrences and frequency of access.

The Data Base Management System Model

We have seen that it is desirable to establish a general purpose data model of an organisation primarily to simplify human perception of that organisation for those individuals who wish to study and improve its operation. Given that a shared pool of data can be established on computer files in the form of a data base, an abstraction of that generalised data model can be made in the form of a Data Base Management System Model. In this model the various processes which are undertaken by the organisation are represented by executions of application programs which interact with the data base using the conventions of some database management system or systems. The requirement to produce an abstraction of the generalised data model stems from the data structures supported by existing data base management systems. The fairly simple structuring primitives of many data base management systems may tend to produce an apparently more complex model than the less formal requirements of the generalised data model but they have the significant advantage of providing straightforward access paths for the application programs. There is a parallel in this process to the abstraction which has been necessary to transform the rich language of speech and the written word into the rigidly structured programming languages necessary to communicate with the computer.

Classifications of DBMS users.

People interact with computers on different levels. Thus, for example, the following individuals may be identified.

The System Programmers who write and maintain Operating Systems and other hardware-specific software.

The Operating Staff who schedule program runs and optimise system throughput.

The Application Design and Programming Staff who transfer English-like requests from End Users into a series of accesses and manipulations of data.

The End Users who provide information to the computer system and obtain information from it. These have been further divided into various categories. Sophisticated Users are able to convert their own requests for information into meaningful instructions to the system. Casual Users can present a request for information in a "natural" fashion to the computer which interprets that request and processes it. Parametric Users provide data in standard formats and interpret output in a routine manner prescribed for them by the Systems Analyst. /LUCKING-74A/ /SMITH & SMITH/ /MINSKY/.

The categorisation of this "nest of symbiotic parasites" is not straightforward since individuals may fall into more than one category and the categories themselves may overlap. Thus, for example, several application computer systems have been designed which allow "parametric users" to specify fairly complex retrieval requests which would normally be considered as the province of the Sophisticated End-User or the Application Programmer.

Most professions have had many hundreds of years to evolve suitable breakdowns of tasks to be performed by well defined categories of individuals. The computer industry, on the other hand, has evolved the above groupings in only two or three decades. There is therefore a general feeling that this might not be the "right" breakdown at least in terms of the proportions of individuals in each category. This view is reinforced by the ever decreasing real cost of computer equipment with the corresponding loss of significance of program efficiency and by the ever increasing capabilities of programming languages and operating systems.

It was into this environment of fairly well established classes of computer user that the Data Base Management System Model emerged. The original requirement came from the application design and programming staff who were becoming aware of the inflexibility and mutual incompatibility of their existing systems. It was natural therefore that the first types of model were based on generalisations of the techniques used for data processing within these conventional application systems.

Classifications of DBMS Models

The constraints of sequential record processing imposed on magnetic-tape-based systems had allowed hierarchical record relationships to become commonplace. Almost universally magnetic tape master files would hold records of different formats (distinguished by a data field) in an order such that data common to a group of records would be held in a record of another type preceding that group. Thus for example a bank's master file might hold its records in branch order with the records for each branch in customer order and all account records for each customer following that customer record. Thus evolved the Hierarchical Data Base Management System /IBM-IMS/.

The amount of abstraction required to represent typical generalised organisational data models as hierarchies might well be significant and the eventual data base management system model might obscure some of the fundamental structure of the original data model. It is likely that there will be additional structure 'hidden' within the application programs.

While the relationship between records in a hierarchial data model is essentially specified by their order it is possible to define a more general data model structure by representing record relationships by a mechanism outwith the records themselves. Thus emerged the Network Data Base Management Systems /CODASYL-71/ /IDS-II/ /IDMS/.

In both of the above approaches the data base management system models were devised on an empirical basis with a knowledge of what types of computer systems could be produced by the Systems Programmers to be used in turn by the Application Programmers to run programs with reasonable efficiency. There is a parallel here with the evolution of programming languages such as FORTRAN and COBOL where enhancements tend to be a compromise between what is desirable and what is achievable.

A more radical approach to the structure of the computerised data model led to the development of the Relational Data Model where each type of record was disjoint and record relationships were established in an ad-hoc fashion by correspondence of value between attributes in the records. /CODD-70/ /ASTRAHAN/

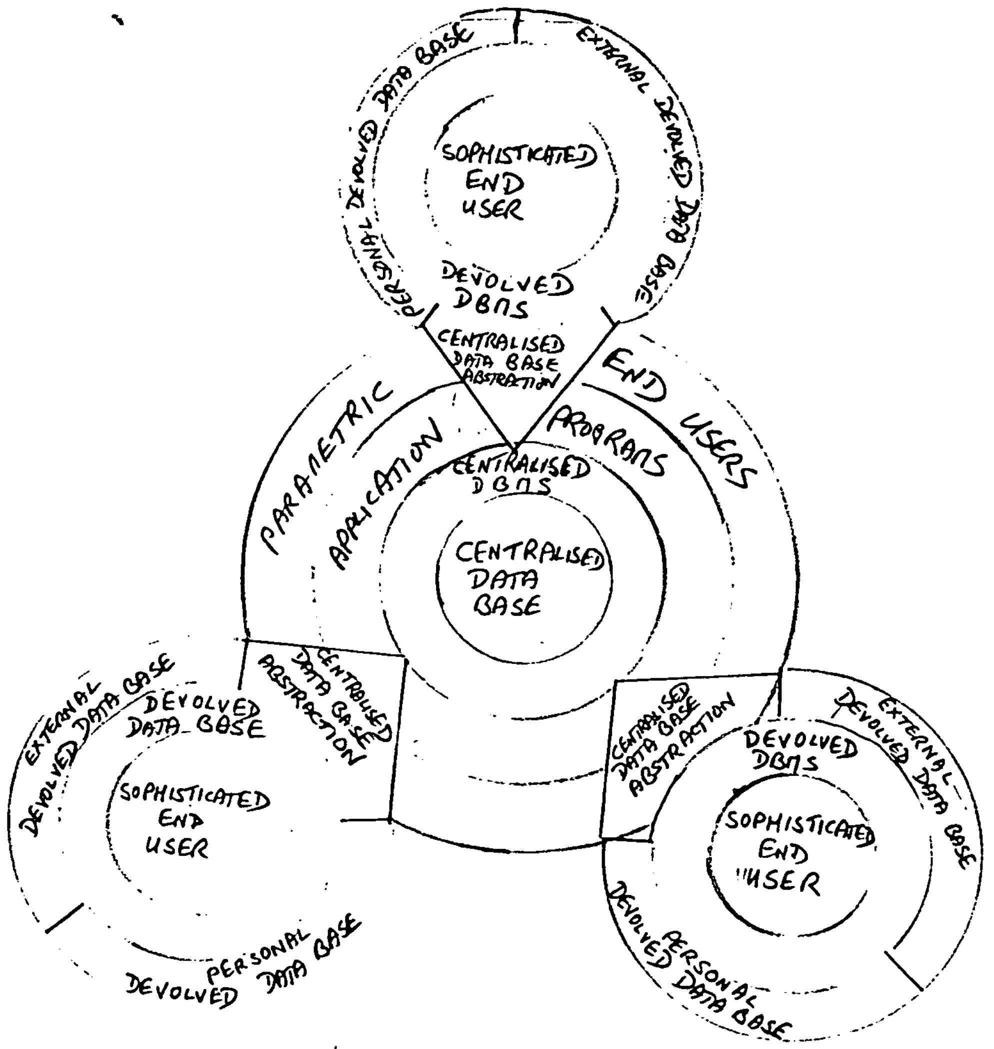
The differences in approach of these types of computerised data models may be seen to be associated with differences in emphasis on the priorities of the various objectives of a data base management system /STOCKER/. The "traditional" approaches of the network and hierarchical data base management systems are oriented towards the goal of providing the professional Application Programmer with a mechanism for interacting with the data base in a manner similar to that adopted for conventional files. At the same time the shared nature of the database is considered but left relatively transparent to the programmer. It is important to remember that the application programs which have constituted the bulk of computerised systems over the last 20 years have often been complex not so much because of their interaction with data held on secondary storage devices but more simply because they had complex functions to perform. This situation is perhaps best illustrated by reference to the program specifications from which the programs were written. After due analysis of the business functions these would typically take many weeks or months to write and would run to many pages of text. No matter how sophisticated programming languages may become in the future they cannot be expected to be simpler than today's program specifications.

The Relational Data Model has been more oriented towards a realignment of the functions of the various computer users to provide a more immediate access to the data base for End Users. In particular ad-hoc queries could be satisfied without the need to reference application programmers. Indeed this is a laudable objective but it has limitations since many of the benefits of computerisation have accrued from the thorough analysis of business practices and information requirements. Indiscriminate interrogation and update of a data base could lead to wasted resources within the organisation as a whole because individuals were not sufficiently disciplined to pursue activities specifically related to their jobs. Equally it would be more difficult to exercise control over individuals expending effort in retrieving information which had already been obtained by someone else.

Centralised and Devolved Data Bases.

One possible scenario for the future symbiosis of the different approaches to data base management systems is to postulate two distinct classes of DBMS which could have properties in common but which were designed to serve very different user requirements.

The two proposed classes will be termed Centralised and Devolved Data Bases and their interrelationship is shown in the following diagram:


SOPHISTICATED END USER
DEVOLVED DBMS
PERSONAL DEVOLVED DATA BASE
EXTERNAL DEVOLVED DATA BASE
CENTRALISED DATA BASE ABSTRACTION
END USERS
PROGRAMS
PARAMETRIC
APPLICATION
CENTRALISED DBMS
CENTRALISED DATA BASE
EXTERNAL DEVOLVED DATA BASE
DEVOLVED DATA BASE
CENTRALISED DATA BASE ABSTRACTION
SOPHISTICATED END USER
PERSONAL DEVOLVED DATA BASE
CENTRALISED DATA BASE ABSTRACTION
DEVOLVED DBMS
EXTERNAL DEVOLVED DATA BASE
SOPHISTICATED END USER
DEVOLVED PERSONAL DATA BASE

The CENTRALISED DBMS would be the province of the data processing practitioners and would support application programs where data is captured and updated in a constrained environment devised by Systems Analysts so that it can be guaranteed as correct and unambiguous to all current and future users within the organisation. The shared nature of this data places grave responsibilities on the designers of application systems to ensure that a single coherent data base structure exists. Furthermore the desire to satisfy the requirements of the entire organisation and not just particular sections who have identified known requirements implies that it would be advantageous for individual programs to view the data base as a strict subset of this mutually agreed overall structure.

The second class of DBMS may be termed a DEVOLVED Data Base in the sense that many such data bases could exist within a single organisation. The concept of Devolved Data Bases is very different from that of a Distributed Data Base where the physical placement of the organisation's one and only data base is scattered between a number of locations. Each Devolved Data Base, on the other hand, would be under the control of a (sophisticated) End User who would handle a complete spectrum of tasks which would be considered as the province of Data Base Administrator, Application Programmer as well as End User in a Centralised Data Base environment. Whether all Devolved Data Bases are held together at a central site or whether they are held on their own microcomputers is a matter for individual implementations and must take into account the computing resources available within the organisations. Nevertheless the concept of the Devolved Data Base reflects the different requirements of different classes of users within organisations and is valid however it is implemented.

The Devolved Data Base can be considered as comprising three sections as follows:

The first section would be an abstraction of the data held on the central data base as updated by the programs designed and written by the data processing professionals and consequently universally agreed as 'correct' throughout the organisation. The structure of this data could be similar to the single logical data base structure of the centralised data base of the organisation but there would be no requirement for it to be a strict subset. It may be that a comprehensive mapping language which converted from one structure to the other would be adequate to define the abstraction from one view to the other but it might be more appropriate for specialised application programs operating directly on the Centralised Data Base to provide the interface. Thus, for example, a bank's economist might 'see' the current financial position of the bank for his purposes as a table of average balances of each bank branch. The logical structure of each branch with its many accounts could be abstracted to this view either by defining a relationship as an algorithm for calculating the average balance per branch or by an application program being prepared which would perform the desired manipulation whenever this information is required. It might even be desirable to run the program at appropriate times to provide snapshots such as end of month positions.

In a similar vein, the second section would consist of data describing the world outside the organisation. Like the first section, this could be guaranteed as correct, this time by the source in the outside world which provided the information (ie an external Centralised Data Base). The data would not therefore be updated by the end user himself unless he was acting as an agent for the actual source. Abstraction of the data into a structure suitable to the end user would be the responsibility of the end user himself and to this extent he would perform the role of Data Base Administrator on his own devolved data base. Examples of this type of data within a devolved data base of a bank's economist would be census information or government statistics.

The final section of a devolved data base would be composed of information captured by the end user himself (or by parametric end users acting on behalf of a sophisticated end user) and guaranteed as correct only in the opinion of that end user. Since this responsibility for update lies solely with this individual user there is a possibility that the data will be ambiguous in that another user would be free to hold different values for the same data at the same time if he saw fit to do so. To continue with the example of a bank's economist, data of this type would be the type of information which he would supply to his model of the national economy which would dictate (in his opinion) how that economy would perform in the future. It is therefore possible that two economists, even in the same organisation, would have different opinions as to which factors would most influence the economy and what values these parameters would adopt. For this section of a devolved data base the role of Data Base Administrator would be almost entirely devolved to the End User himself. The intimate association between the user's meta-data-base and actual data base for this type of data would seem to advocate a method of programming where the two are virtually indistinguishable. Such techniques have been proposed /e.g. SHIPMAN/ but it is important to view them within this somewhat restricted context of one part of a devolved data base since their evident advantages in the ability to alter the meta-data just as easily as the actual data are less apparent when that data has to be shared between different users. The timescale for the implementation of changes which effect several users will necessarily be longer than that for changes to a single user's own data in his devolved data base. Inevitably some measure of consultation would be necessary in the centralised environment - the raison-d'etre for the Application Administrator.

In different environments it is likely that one or other of the three sections will dominate the devolved data base of a particular end user. For example some end users would operate entirely on their own self contained data base while others would operate entirely on an abstraction of the organisation's centralised data base. The Application Administrator will have an important role to play in any event although it would be more passive than some of his other tasks by ensuring that data held within the self-contained sections of devolved data bases was not, in fact, of more general interest throughout the organisation and therefore a candidate for inclusion in the centralised data base. If this were to be the case no doubt there would be debate between the End User and the Application

Administrator as to the desirability of this alteration since the ability to make changes without inconveniencing other users would be lost.

The identification of the two classes of data base leads to the consideration of the status of the output derived from the application programs operating on the centralised data base as against that obtained from end user programs or queries on devolved data bases. For the former it is realistic to expect the results to be accepted as correct universally throughout the organisation for if some individual disputes the results this should provoke an analysis of the cause of the objection so that the alleged anomaly can be resolved or at least explained. For the latter the results need only be accepted as the product of a single end user's analysis. Individuals within the organisation may disagree with them if they see fit and they may attempt to convince the originator of their inaccuracy. Thus, for example, a figure giving the total number of current employees of an organisation should be immutable but the projected figure for a year ahead could well differ depending on the criterion used to predict the extent of the organisation's activities.

It is the 'visionaries' within organisations who have the task of planning for the future who are the prime candidates for becoming sophisticated end users of their own devolved data bases. Although relatively few in number it is on the success of these forecasts that the future prosperity of the organisation will depend. They will not be computer professionals and the benefits of the 'user-friendliness' of self-contained, non-procedural, query-oriented Relational approach will be much more apparent.

Evaluation of the Data Base Management System Models.

The view that different types of Data Base Management Systems should be oriented towards different classes of user is supported by MICHAELS et al under the following headings:

(a) Convenience - the merits of the various approaches were partly subjective based on the users background and personal preferences. Relational for non-computer-specialists and Network or Hierarchical for application programmers.

(b) Selective Power - Relational languages are complete because they can express any query expressible in the relational calculus. The other approaches are complete in the sense that most programming languages are complete (i.e. it is always possible to express the query in a logical form with the suitable use of conditional expressions and other procedural techniques).

(c) Conciseness - the Data Selection Language of the Relational Data Model tends to involve fewer pen strokes or key depressions than the procedural approach. This is not to say that such queries are quicker to prepare since the formulation of the query may require fairly time consuming thought processes especially for non-mathematically oriented users.

(d) Language Level - The Relational non-procedural language may be said to be a higher level than that of the procedural approaches. It nevertheless remains a stylised language which must be learned by those who wish to use it. It does not allow the Casual End User to communicate with the computer in as free a way as that individual would expect to communicate with another human being.

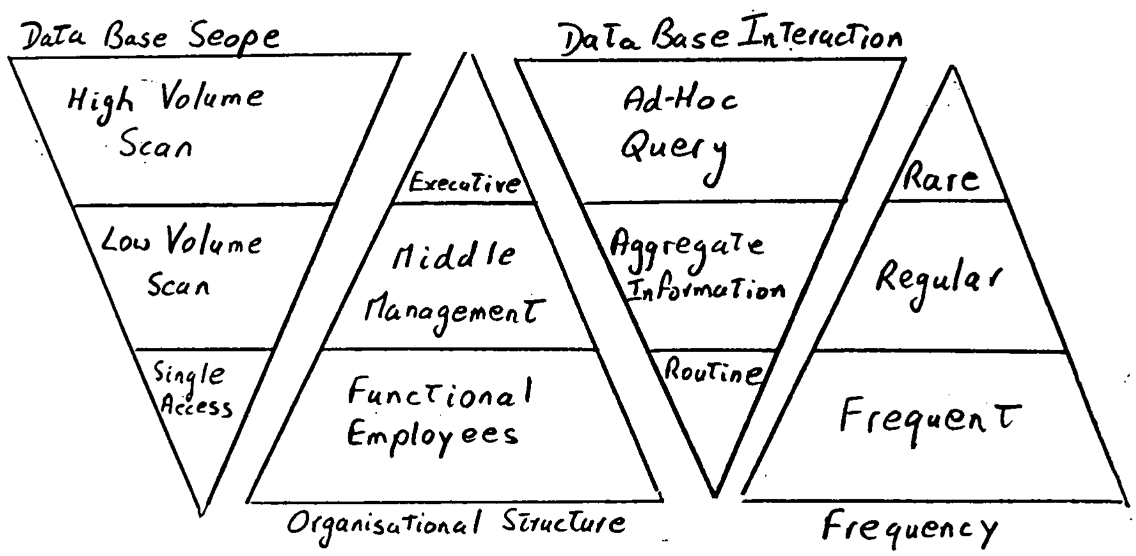
(e) Complexity - The ability to utilise details of data access and physical placement tends to lead to more complex and less logically data independent programs. Since the advent of the storage schema this criticism can now more properly only be aimed at particular implementations of the Network and Hierarchical approaches.

The relative future importance of the various types of Data Base Management Systems will depend to some extent on the eventual distribution of the various classes of data base user. Such predictions are difficult to make, especially in the rapidly changing technology of the computer industry. In 1974, for example, CODD & DATE predicted a vast increase in the population of sophisticated end users and casual users over the following 10 years to such an extent that they significantly outnumbered professional programmers. No such dramatic reversal is in fact apparent and the reasons for this trend may well highlight why the hierarchical and network approaches will continue to dominate data base usage techniques (albeit in the restricted community of data processing professionals).

Although the population of sophisticated end users may not have increased as much as expected, the population of parametric users has increased to such an extent that virtually the entire population can be said to interact with computers especially using such devices as self service banking terminals and Teletext keyboards connected to televisions. While the data processing community can satisfy the requirements of users without those users having to expand their skills to interrogate data bases by becoming sophisticated or casual end users there is little motivation for the lay community to learn such skills.

On the other hand the difficulties encountered by the professional programmers in modifying large computer application systems and the resultant long lead time for even the most trivial change have more and more been counteracted by users acquireing the skills of the application programmers. The availability of cheap microcomputers with easily tailored package programs or simple home-grown BASIC programs has allowed data to be maintained (often including data also handled by the mainframe computer) and analysed without access to the centralised data base. There is little evidence of non-professionals finding even 'old-fashioned' procedural languages of this type too complex once they have sufficient motivation to learn the language. Unfortunately this trend could eventually lead to the problems of inconsistency and incompatibility which data base was intended to overcome.

The accuracy of the data on the data base is vital for all applications but particularly for financial or personnel data bases. In such environments it is essential that information is only updated using recognised procedures. To allow even the most senior executive of a bank to alter the balance on an account at his whim would be ludicrous, as would be to allow personnel department staff to add a new member of staff without going through the recognised selection procedures. There are therefore tasks which must always be properly analysed and the current state of the art dictates that the end result of such an analysis will be programs written to support data base interaction on a record by record basis as handled by the hierarchial and network data models.



In the diagram above it can be seen that the needs of the different categories of employees vary widely. The large number of "shop floor" employees tend to require the same information over and over again on different entity occurrences, often so that they can supply new or updated information. They are likely, therefore, always to be classified as parametric users. On the other hand the executives of the organisation should from time to time require information which has never been required before and is unlikely to be required again. Perhaps surprisingly such individuals are also likely to remain parametric users since it would probably be a more efficient use of their time to communicate their request to a data processing professional (possibly one well versed in query languages) than for them to attempt to communicate directly with the computer. It is perhaps the middle management (particularly those involved in determining future strategy) who require the query language of the sophisticated end user but while management is functionally or organisationally structured these individuals are unlikely to require access to the entire data base. An abstraction of the centralised data base would always seem desirable in conjunction with the 'outside-world' and 'self-contained' sections of a devolved data base for each such end user.

Thus although the Relational approach with its emphasis on interacting with the data base via queries by sophisticated end users is perhaps a more fertile ground for novel academic research the other more traditional approaches are likely to grow rather the diminish in importance and therefore cannot be neglected. The primary functions of the network and the hierarchial data base models have been well understood for some time and there are many implementations. The more peripheral aspects such as the ability of these models to react to change have not as yet caused users much inconvenience and have therefore remained relatively unexplored. As the usage expands, however, such topics will assume greater importance and deserve detailed attention.

Chapter 3 - Current Implementations of Data Base Management Systems

The three most widely used types of Data Base Management Systems will be considered in this chapter. A more detailed description can be found in DATE's book "An Introduction to Database Systems". By describing both their differences and similarities, especially within the context of Centralised and Devolved Data Bases, the areas where Restructuring will be important will be highlighted so that they can be developed in later chapters.

The Hierarchical Data Model.

The Hierarchical data model is important mainly because IBM have established it as a de-facto industry standard by promotion of their product Information Management System (IMS) with its associated data manipulation language DL/1 /IBM-IMS/.

In a Hierarchical Data Base Management System a data base is defined as an inverted tree structure. Each node of the tree is termed a segment. The tree is headed by a root segment supported by a hierarchy of dependent segments. Hierarchies provide a simple, easily understood structure on which to establish a data base.

Thus in a banking example the hierarchy might be:

```
                     BRANCH
                        :
                        :
           ........CUSTOMER........
           :            :          :
           :            :          :
       CURRENT      DEPOSIT      LOAN
       ACCOUNTS     ACCOUNTS     ACCOUNTS
```

In this case the branch is the root segment and this has a single type of dependent in the form of the customer segment which has in turn 3 dependents: Current Accounts, Deposit Accounts and Loan Accounts. Since the dependent segments are on the same level of the tree they are called twin segments. This type of structure has an implied order which corresponds to that used in the past for conventional sequential files - each group of occurrences of a type of dependent segment is preceded by the occurrence of the segment on which it is dependent (i.e. its parent where it is the child). Where more than one type of segment is on the same level of the hierarchy there is a convention that all occurrences of each type of segment are presented together after their parent working from left to right in the diagram (twin segments in IMS parlance). Thus in the example above all current accounts for the same customer would be presented in order before all Deposit Accounts and Loan Accounts. Corresponding to this ordering is the concept of a concatenated key. The concatenated key of a segment consists of those data items which distinguish individual occurrences of that segment from other sibling occurrences of the same segment together with those data items which perform the same function for its parent segment and the parent's parent right up to the root.

Data bases are normally mirrored in the different types of files used to hold either the entire data base or particular types of segment on disc. It is likely therefore that the entire information maintained on an organisation will be held on a number of data bases with the responsibility of utilising consistent information from different data bases resting entirely with application programs. The impact of this situation can be minimised by the definition of Logical Data Bases which utilise segments from existing Physical Data Bases to establish a hierarchy of segments which was not previously apparent and which would not require the duplication of physical records.

Thus, for example, in the bank data base if customers could have accounts at more than one branch it would be possible to define the following structure of a distinct logical data base.

```
CUSTOMER
   :
   :
 BRANCH
```

Application Programs interact with data bases through Program Specification Blocks which are merged with the object versions of application programs by the Linkage Editor. These blocks present the programs with the data retrieved-from or to-be-inserted-as individual segment occurrences. It is the responsibility of the program itself to provide the definition of the data items in each segment in a format appropriate to the source programming language being used - this definition would typically be obtained from a library of such definitions maintained for the purpose.

A data base may be traversed using the data manipulation language DL/1 which allows the following operations:

A GET UNIQUE command allows the program to retrieve a particular occurrence of a particular type of segment together with its parents if required. Before the command is issued the name of the required segment and the names of any parent segments required together with the desired values for all data items in all concatenated keys must be placed in an area known as a Segment Search Argument.

The data base may also be traversed sequentially in the order described above by the use of a GET NEXT command. This process may either start at the beginning of the data base (the first occurrence of the root segment) or from the current position of the program on that data base as established by previously executed commands. All types of segment may be presented to the program in which case the command is said to be unqualified. Alternatively only segments of a particular type may be presented (still in the data base order) by placing the segment name in the Segment Search Argument area.

A restricted form of the get next command presents the program only with those segment occurrences which are children of the same parent occurrence. This is the GET NEXT WITHIN PARENT command.

New occurrences of segments may be added to the data base using the INSERT command. The full concatenated key for the segment must be provided in the Segment Search Argument area. If occurrences of that type of segment with that value of concatenated key already exist on the data base an "insert rule" (which must be specified when the data base is being defined by the data administrator) is invoked to determine whether the occurrence is to be placed before or after the existing occurrences or whether the insert is to be prohibited.

Existing occurrences of segments may be modified by use of the REPLACE command. Once again the new value of the segment must be provided, together with the concatenated key in the Segment Search Argument. An important constraint on the use of the Replace command is that the record to be modified must have previously been retrieved by the program using a Get Unique, Get Next or Get Next Within Parent command with a "Hold" option to indicate that the record is likely to be modified. This procedure avoids the problem of inconsistent update where two programs wish to update the same segment occurrence after both have retrieved the same "raw" version of the segment. In such a situation the second get with hold would be rejected by IMS and the application program would have to take appropriate action (possibly to try again in the hope that the segment had by now been updated by the first program).

Existing segment occurrences may be deleted from the data base using the DELETE command. Once again this must be preceded by a get with hold command.

Although IMS is limited in its ability to provide a satisfactory computerised model of an entire organisation in that such a model must be composed only of hierarchies it is probably today's most widely used data base management system. Despite the overheads that it places on the computer on which it runs it provides an unparalleled mechanism for the security of the data held on its data bases in the event of any type of hardware or software failure. As well as utility programs to back up data bases to magnetic tape from time to time (even while the data bases are being updated) /IBM-DBRC/ and restore these to disc when necessary, a Log Tape is constantly updated with data base changes. In the event of a catastrophic failure various levels of recovery may be undertaken including, if necessary, the most recent changes held in program buffer areas retrieved from the contents of memory at the time of the failure.

IMS is also widely used as a teleprocessing monitor. It supports a subsystem known as Message Format Services which makes it particularly convenient for the development of Visual Display Unit oriented applications /IBM-MFS/.

Although DL/1 is the recognised data manipulation language, higher level approaches are also available such as the Automated Development Facility /IBM-ADF/. Information retrieval facilities for non-professional programmers (sophisticated end users) are also available /EASYTRIEVE-IMS/.

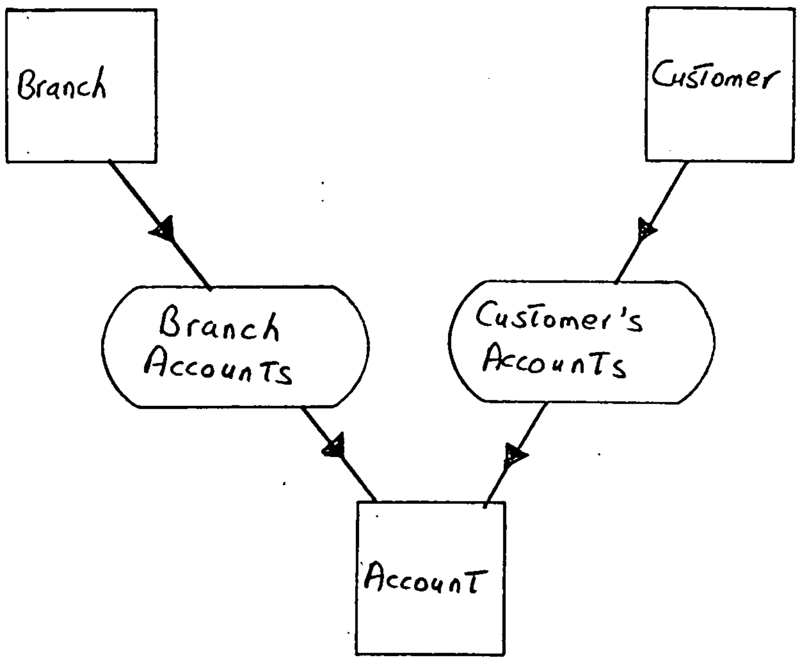
The Network Data Model.

While recognising the practical advantages of the Hierarchical Data Model its severe limitations of not normally permitting an organisation to be modeled as a single data base are apparent. At about the same time as IMS was evolving, other computer manufactures were developing systems which were less restrictive in the required pattern of record relationships. In 1969 the CODASYL organisation published a survey of current systems /CODASYL-69/ and later recognised the requirement for a uniformity of approach. In 1971 the CODASYL Data Base Task Group published a report which suggested a sophisticated method of providing a computerised model based on network rather than hierarchical record relationships /CODASYL-71/.

It is significant that the proposals contain various constraints which prevent the most general model of many-to-many record interrelationships from being computerised. A measure of compromise was reached between a totally flexible modeling technique and a mechanism which would make it unduly complex for application programs to navigate the data base and perhaps could not be implemented without imposing undue processing overheads.

In addition to the definition of the contents of classes of records the network data base management system permits the definition of relationships between individual record occurrences as "Sets" on a one-to-many basis. Thus one type of record is the Owner of the set and another (possibly the same) type of record is the Member. In fact there may be more than one type of member for each owner. Effectively Sets are a two level hierarchy which are used to link records to produce an arbitrarily complex network. The network, however, is of a special type since any individual record occurrence can only contribute to a maximum of one occurrence of each type of set of which it is an owner and a maximum of one occurrence of each type of set which it is a member.

A recognised way of diagramatically representing a network of this type was developed by Bachman for the system IDS which predated the CODASYL report /BACHMAN/. The bank data base would be represented as follows:



This structure is more versatile than the illustration given previously for the hierarchical data model since it allows individual customers to hold accounts at more than one branch.

The CODASYL report uses the term Schema for the formal description of the network. Here each record is described in terms of the data items it contains using a syntax not unlike that used by COBOL for the definition of the contents of the records on files. Further each set is described in terms of its owner and member records together with certain qualities which the set possesses such as whether all occurrences of the records must be participants in some occurrence of the set. Rules may also be supplied to determine the set occurrence appropriate to each individual record occurrence.

The original 1971 report also used the Schema to record information on how the records and sets would be structured on the secondary storage medium. At the time several commentators critisised the effect of this situation as it applied to Physical Data Independence. This is an example where the imprecise use of the term data independence has been a handicap. The authors of the report considered themselves to have tackled the problem of physical data independence by postulating a Device Media Control Language which they suggested would differ from implementation to implementation but would preserve the data base management system from the idiosyncrasies of different operating systems. The critics considered physical data independence to be resilience of applications to changes in physical data structure thus allowing overall performance to be tuned. Sadly this area of dispute overshadowed debate on the facilities provided by the proposed system and of the limitations imposed by the constraints inherent in it when considered against a totally abstract data model. In 1978, however, the criticisms were largely answered by a CODASYL Journal of Development which moved physical structure properties from the Schema to a Storage Schema designed to facilitate such definitions /CODASYL-78/. A further refinement was made in a later Journal of Development in 1981 /CODASYL-81/.

The CODASYL proposals also suggested how application programs would communicate with the data base. Each such program would contain a Sub-Schema which provided the program with a definition of that portion of the data base which was to be of interest. In this case the authors considered this procedure to provide a measure of logical data independence. Probably because of considerations on what could be implemented with reasonable efficiency the proposals restrict the sub-schema to be a strict subset of the schema conforming to the same pattern and same conventions for record, set and data item names. Aside from security benefits there would seem to be little to be gained in the way of convenience in this restriction of the sub-schema to be a subset of the schema as it would often be simple for the programmer to copy the entire schema directly into his program (assuming the language was compatible) rather than go to the trouble of preparing a sub-set of it. One advantage of minimising the scope of individual sub-schemas would be that so long as each sub-schema remained a subset, the schema itself could grow without alteration or recompilation of existing programs. A more sophisticated mapping between schema and sub-schema would be a feasible proposition but in the final analysis it is perhaps the arguments for a single data base structure within each organisation (as outlined in the previous chapter) which best support the CODASYL approach.

Individual record occurrences are transferred to and from programs via an area reserved by each program for this purpose - the User Work Area. A Data Manipulation Language is also proposed which is sufficiently powerful to permit application programs to interact with the data base at least as easily as they could store and retrieve data from conventional tape and disc files. The level of interaction, like that of the hierarchical data base management system, is designed to support the type of programs that had already proved capable of solving the problems of parametric end users as identified by Systems Analysts and implemented by professional Programmers.

The Data Manipulation Language for COBOL supports the following commands /CODASYL-COBOL-JOD/:

A FIND command effectively points the application program to a particular record in the data base. Numerous versions of the command are available which would allow a record to be retrieved "out of the blue" by supplying values of key data items or to be retrieved because of its relationship to another record previously accessed by the program. It is by this mechanism too that the set construct is used to allow the program to navigate through the network of data base record occurrences with a particular set occurrence.

Once a record has been retrieved from the data base the GET command allows one or more data items to be transferred to the application program User Work Area where they can then be manipulated like any other item of data. The separation of the Find and Get functions also seems to indicate that the CODASYL authors are protecting programs from the growth of additional data items in records in the schema. It is only rarely that a program could improve its performance by carrying out additional operations between executions of the two commands and this is therefore unlikely to be the justification. In fact there are implementations of the CODASYL proposals which provide an OBTAIN command which is a combination of a FIND followed by a GET.

The remainder of commands of the Data Manipulation Language allow records to be added to the data base and to particular set occurrences in a similar way to the hierarchical data base management system. Existing record occurrences can be modified or deleted. Deadlock and inconsistent update are prevented by the use of explicit HOLD and RELEASE commands within the language.

<u>The Relational Data Model.</u>

The Relational model arose as an alternative to the other traditionally-based data base management systems following the publication of a paper by E.F. Codd of IBM's San Jose laboratory in 1970 /CODD-70/. Motivated by the need for a tool to free end users from the frustrations of having to deal with the "clutter of storage representations" Codd reverted to first principles and suggested that entities were n-ary relationships of their n attributes. As such, the power of algebraic operators could be brought to bear on the data without the requirement of conventional record-at-a-time programming techniques.

Codd also proposed techniques for establishing the computerised model in a particularly desirable format by the process of normalisation. As outlined in Chapter 2 this process is now seen more as a tool for the data analysis exercise which is necessary for the conversion of an abstract organisational model to a data base management system model of whatever type. The technique is not a property of the relational data model in particular. Unfortunately many of the published papers on the Relational Data Model discuss both normalistion and the computer-oriented aspects of the model and the distinction between the two is not always made.

Thus the basis of the Relational Data Model is the partitioning of the data base into groups of occurrences of relations of various types. The model makes no attempt to establish how these relations will be held on storage media but presumably implementors could choose some point in the spectrum between disjoint "files" (one per relation) and a totally interrelated structure where all possible combinations between pairs of relations (based on equality of value of attributes with the same name in each relation) would be manifested by appropriate indices or pointers. Ideally the Data Base Administrator could determine how particular associations between relations would be represented. That is there would be some form of storage schema which indicated which relationships were to be supported by indices, which by pointers and which would be established when required at run time. This physical data independence is an important aspect of the model and is evident by the absence of a construct to permit permanent relationships between relations to be visible to application programs. It is the responsibility of application programs themselves to establish such relationships as part of their own logic especially by the use of the "Join" set-type operator. Codd has since suggested expansion of the model to allow relations to be defined which have more meaning to the eventual sophisticated end uses and data base designers /CODD-79/. Here the data base is considered to consist of "Base relations" which are defined without reference to other relations and "Derived relations" which provide more "natural" views of the data base. The Base Relations would have a direct physical representation on the data base as record occurrences. The Derived Relations, on the other hand, would be synthesised from the Base Relations by the data base management system (not the application programs) when required by the use of the projection and join operations as specified in the schema definitions of the Derived Relations.

The data definition language for the Relational Data Model consists of fewer constructs than either the Hierarchical Data Model or Network Data Model since only the relations (and not the relationships between them) have to be specified.

The relations have the following properties:

(1) There is no duplication in the rows of the relation (ie no two individual tuples or occurrences are identical).

(2) Row order is insignificant.

(3) Column (ie attribute) order is insignificant.

(4) All table entries (attribute values) are atomic.

Further, each relation must contain at least one set of attributes (sets of Candidate Key attributes) with the following properties:

(1) No two rows of the relation may have the same value for the concatenation of the attributes.

(2) If any attribute is dropped from the set of attributes then the uniqueness property of (1) is lost.

For each set of base relations one candidate key must be selected as the Primary key. The columns of the relation are referred to as domains and the domains of all primary keys are known as the Primary Domains of the data base.

Similarly to the other computerised data models the Relational Data Model supports a Data Manipulation Language. This language allows individual tuples to be Inserted, Modified or Deleted where the values of one of the candidate keys must be supplied by the application program wishing to operate on that tuple. In order to maintain data base consistency, however, the operation must not result in violation of any of the rules for base relations or candidate keys given above or of either of the following data base update rules:

(1) No Primary Key is allowed to be null or have a null component.

(2) Suppose an attribute A of a compound (i.e. multi-attribute primary key) of a relation R is defined on a primary domain D. Then at all times for each value v of A in R there must exist a base relation (say S) with a simple primary key (say B) such that v occurs as a value of B in S.

Where the Relational Data Model differs significantly from the Hierarchical and Network models is in the way in which the Data Manipulation Language allows application programs to establish which tuples of relations on the data base are of interest because they satisfy some selection criterion. The distinction is that the result of such selection will often be more than one tuple occurrence and conventional programming languages are not structured to process groups of records as a whole. They are structured to process each record in a group sequentially or based on the values of key data items. It may be, however, that the object of the selection is simply the display of selected tuples on a Visual Display Unit or Printer or the creation of a more conventional extracted file (which may be viewed as another relation) to be processed later by a program written in a suitable language. In such cases the ability to select tuples based on such a criterion is an important benefit.

Selection is made by the use of the following operators:

RESTRICT - Establish the set of tuples in a relation R where some attribute qualifies on the basis of a simple algebraic expression involving some constant (or table of constants) or some other attribute defined on the same domain.

PROJECTION - Drop all but certain columns of R and then drop redundant duplicate rows.

THETA-JOIN - LET THETA be one of the six binary operators EQ, NE, LT, GE, GT, LE and let the two relations R and S have common domains B1 and B2. The theta-join is the concatenation of the rows of R with rows of S whenever THETA holds between values of B1 and B2.

EQUI-JOIN (where the relation is equality) results in two identical columns in the resultant relation.

NATURAL-JOIN is the Equi-join where one of the redundant columns is removed.

The above operators may be said to constitute a relational algebra and implementations have utilised this situation /CHAMBERLAIN-76/.

Another technique for allowing sets of qualifying tuples to be established is in the mathematical notation of the relational calculus. Here the criterion is defined using two elements normally separated by a colon. The left hand side gives the target of the selection and the right hand side gives the qualification. Thus to select all bank accounts with a balance of over £10,000 would require an expression of the form.

GET ACCOUNT.NUMBER : Account. Balance LT 10000

Many implementors have used the solid mathematical basis provided by the relational calculus to devise more "natural" query languages to allow end uses to interact conveniently with the data base /PIROTTE/.

System R is perhaps the most significant implementation of the Relational Data Model /CHAMBERLAIN-81/ and this supports not only queries of the type described above but also the language SQL /IBM-SQL/ which supports the Insert/Update/Delete operations in addition to the relational algebra when embedded in a host programming language such as PL/1 or COBOL. SQL also makes a further distinction between ad-hoc queries (which are interpreted and processed as and when they are supplied to SQL in real time) and 'canned' programs (where SQL expressions are coded into conventional application programs written in high level languages). It is at this point that the similarities between the Hierarchical and Network Data Models on one hand and the Relational Data Model on the other become apparent: the act of compilation imposes a permanence on the interaction between program and data base which restricts data independence and this effect is common to all models. It is important to realise that this compilation exercise is not solely necessary to benefit program efficiency by avoiding continual interpretation but is demanded because the data base interface will typically form only a portion of an application program where a great deal of the logic is likely to be concerned with how data is manipulated and reported after it is retrieved or captured and formatted before it is stored. Currently compilation is essential to allow such programs to run repeatedly without undue use of computer processing time.

Internal Structure of Data Base Management Systems.

We have seen that there are a variety of items of computer software which fall under the general title of Data Base Management Systems - notably the three types described above. Despite each such system having its own unique external manifestation (primarily because each is oriented towards a specific class of user, be it end user or professional programmer) it is possible to infer the existence of a set of constructs which will form the basis of the internal representation of any Data Base Management System. This is notwithstanding the freedom of design which individual DBMS implementors will possess which will inevitably lead to peripheral differences between implementations. There is even some justification in suggesting that a standard internal data base format could be established which would be a target for all implementors such that divergence of external appearance would be solely directed towards different communities of users. Thus, for example, a centralised data base could be updated by the application programs written by the professional programmers for a network or hierarchial data base and at the same time accessed by the relational ad-hoc queries of sophisticated end users without the requirement for an abstraction process to incorporate this data into a section of each end user's devolved data base.

However the main reason for introducing internal data base structure into this thesis is so that the action of a Restructuring Mechanism can be considered in a fairly general context. Thus the following elements of the internal structure of a Data Base Management System are identified.

The Object Schema.

This element may be seen as a table with each entry corresponding to an occurrence of one of the constructs upon which the external representation of the DBMS is based. Thus in the CODASYL environment there would be one entry for each type of record specified in the Schema plus one entry for each type of set. Similarly in the Relational environment there would be one entry for each type of base relation and one for each type of derived relation.

Within each table entry there would be a number of sub-entries describing how that particular occurrence of that DBMS construct will be physically represented on a secondary storage medium by the DBMS. Thus for CODASYL record the position and format of each attribute would appear, together with details of which attributes can be found on which storage record and how storage records are linked together.

The distinction between the logical structure of a data base and its physical manifestation is less significant in the Object Schema than it is in the external representation of that schema. Schemas and Storage Schemas must assist in the human perception of the data base structure by Enterprise Administrator and Systems Analyst/End User alike and although it is valid to avoid references to physical storage details in order to achieve a more comprehensible data base model this argument is not applicable to the

Object Schema which is not viewed directly by a human being. The other advantage of a distinct Storage Schema - that it can be re-organised without altering the Schema - can be achieved equally well whether the Object Schema contains physical storage information or whether this information were to be held in an Object Storage Schema.

The Object Sub-Schemas.

This data base element is derived from the Sub Schemas associated with application programs when they are compiled. The sub-schema must provide the compiler with details of the structure of that subset of the data base to be viewed by that program. In particular the contents of the records transferred between program and data base via the user work area must be defined in terms compatible with the constructs used for data item representation in the the language in which the program is written. Thus in addition to using appropriate Data Manipulation Language commands to achieve the transfer the program must be capable of referring to the data items within the user work area anywhere in its logic using instructions in the normal syntax of the language. So that the compiler can create valid addresses when compiling such instructions (or at least delay their resolution to module linkage editor time or even start of run time) it is normal to bind the sub-schema to the program fairly strongly at compilation time. If the sub-schema alters for any reason it is likely that the program must be recompiled. Equally if the program is altered the sub-schema will re-bound when it is re-compiled.

But the Sub-schema is more than just the definition of some of the data items used by the program. When the program is eventually run the Data Base Management System must access the Object Sub-Schema so that it can determine what manipulations will be necessary in order to transfer data between data base and program user work area. The Object Sub-Schema may therefore also be considered as a table similar to the Object Schema. Entries in each table are matched each time the program is run and it is therefore essential that they are never incompatible. If either the object schema or object sub-schema is altered for any reason it must therefore be one of the functions of the appropriate software to check for compatibility. It is this inherent ability of allowing schema and sub-schema to alter without the requirement to change the other which provides much of the resilience to restructuring which is so desirable in any Data Base Management System.

Data Manipulation Language Execution Routines.

Operating Systems normally support routines which handle the transfer of data between programs and secondary storage in response to the Input/Output instructions embedded in conventional programming languages. Similarly Data Base Management Systems must provide routines which handle the transfer of data between programs and data base in response to Data Manipulation Language commands. As we have seen, however, these routines must be more sophisticated than those of the operating system since they must manipulate data items by reference to both object schema and object sub-schema.

Different implementations may bind the routines into the programs at compilation time or module linkage editor time. An example of this approach is the concept of Batch IMS programs as opposed to on-line Message Processing IMS programs working in conjunction with a central IMS Control Program. The alternative to binding the routines into the application programs is to have library copies of the routines available for execution when required. This type of binding has the advantage that the routines themselves may be amended if required without having to recompile the application programs.

Database Records.

The bulk of the data base will, of course, normally be occupied by occurrences of storage records. The 1981 CODASYL Journal of Development has described in some detail how these records would relate to the logical records defined in the Schema and it would seem that this approach would be relevant to the comprehensive implementation of any class of DBMS. Logical records themselves will not exist as distinct entities, they will simply be agglomerations of storage records.

Each occurrence of a storage record will be characterised by a unique storage address within the data base. The method of construction of the address will depend on the implementation but typically it would be composed of the volume number of some disc pack together with an address on that pack. The 1981 CODASYL Journal of Development suggests that all storage records will contain a version number so that they may be reorganised without impacting any application programs. This thesis proposes a further version number which is applicable to each logical record rather than any of its constituent storage records. Nevertheless this version number must be held somewhere for each record occurrence and implementors could hold it on each corresponding storage record or possibly only on the first such storage record for each logical record occurrence. Alternatively a mapping between storage record version numbers and schema record version numbers would be feasible such that a range of storage record versions were equivalent to the same schema record version number.

Peripheral Data.

The final element in the internal structure of a Data Base Management System covers several miscellaneous items of data which must be held. In addition to a library of Data Base Procedures, a table of Open Area Indicators and a Free Space Directory it is particularly important that a mechanism exists to allow application programs to navigate from one record occurrence to another using the constructs of the Data Base Management System being supported. The debate on procedurality of Data Manipulation Language is concerned with whether the DBMS or the program should perform the navigation - but for the internal organisation of any DBMS inter-record association is always essential if continual exhaustive searches of the data are to be avoided.

Physical juxtaposition is the simplest method of associating two records but data bases invariably require one record occurrence to be associated in different ways with different record occurrences and this approach is not feasible in this case. Further it presents problems if another record occurrence is to be inserted between two associated records. More practical inter-record relationships can be implemented using pointers embedded within data base storage records (like the schema record version number, either repeated on each corresponding storage record or only present on the first record) which can either point to an entry in an index containing record addresses or can themselves consist of an address. Alternatively indexes can be set up for each type of relationship containing the address of record occurrences with a cross index of record key to index entries.

Chapter 4 - Restructuring a Data Base.

Changes To Computer Systems.

The rapid growth of computerisation over the last 30 years has presented many challenges to the data processing professionals who have nurtured the science through its formative years. One particularly significant challenge has been to allow the computer-based application systems to react to change. Analysis of user requirements normally provided a sound basis for the development of systems to allow data to change in value in an orderly, efficient and controlled manner. It has become apparent, however, that changes to the systems themselves, and especially to the structure of the data they maintain, cannot be achieved without a great deal of effort on the part of the data processing staff.

The effort required to produce systems in the first place in terms of analysis, design and programming effort is often significant and can have a profound effect on the structure of each system and may even determine whether the system is developed at all. The process can be likened to the architectural and civil engineering effort required to create, say, a new building - investment in Research and Development at this stage should, in general, be reflected in the quality of the finished product. But where computer systems differ from buildings is that they tend to alter continuously throughout their lifetimes. Perhaps buildings, too, would alter to reflect changing patterns of usage if this were practicable. Perhaps it is inherent in human nature that if something can be altered then it will be altered but if alteration is impossible then methods will be devised of coping with the unaltered article. The 'soft' nature of computer programs makes them theoretically amenable to virtually any required change without actually scrapping and rewriting the original system. This is something of a new environment for any professional discipline - design of computer systems is an evolutionary process whereas other creative processes result in an immutable end product. There is, therefore, no established criterion with which to compare the effort of change with the effort of creation but it appears that such changes require a disproportionate amount of programmer time. Individual changes are often made on a piecemeal basis and are justified on their own merits, but it could well be that the objective of such changes could be achieved with far less effort if the change had been incorporated into the original system design. There is, however, a measure of creativity within the human thought process which is stimulated by the practical achievement of aims and it is therefore unlikely that the art of analysis can ever be refined to such an extent that all possible future directions can be foreseen and even if they are whether they can be incorporated into an economically feasible system design.

The problem of constant system maintenance therefore appears to be one that will plague computer installations for the foreseeable future. Any way in which its comsumption of programming resources can be reduced will be most desirable. As an example of the current extent of this problem the workload of the 80 programmers and systems analysts in the Bank of Scotland is such that only about 10 would be engaged in the development of new systems at any time. The remainder spend their time on the onerous task of amending existing programs or writing new programs to be incorporated into existing systems. Further, as the usage of its computer by any organisation increases over the years the design of existing systems imposes more and more constraints on the design of new systems since interface between systems is virtually unavoidable and is often desirable.

In the embryonic stage in the installation of a computer application there are often a multiplicity of changes required to correct errors in the system design or programming which become apparent only when the system has become operational. Essentially the system has not quite met the requirements for which it was designed even if some of these requirements were not identified as requirements by the eventual system user when he was being consulted at the design stage or were not even identified by the analyst when he was collating all relevant facts from all users. No matter how well analysed and carefully tested, the shortcomings of a system will not be apparent until the system has been put into practice as a tool for those human beings it is designed to serve - the "End Users". Although it is important to minimise the extent of these shortcomings, and thus the modifications required to correct them, the changes at this time are largely expected (at least in general terms) and the effort required to carry them out can be scheduled and costed as part of the system development plan.

Today many systems have long ago "bedded in" in this way but they are still changing. Often the users of the systems have identified areas where enhancements would allow them to perform their allotted task within the organisation more satisfactorily. The motivation for such proposed changes can stem from influences outside the user's immediate frame of reference which have altered the end user's job in some subtle way. Alternatively some individual may simply have a creative thought which puts a different perspective on the analysis on which the system was designed. Indeed the activities of any organisation are a moving target for analysis and it is only rarely that all future eventualities can be foreseen and taken account of in an original system design. It is this on-going system enhancement effort which is the major drain on valuable programmers time in many computer installations.

Difficulties Associated with Changing Computer Applications.

It is important to establish a perspective on the reasons why changes to computer systems are so time consuming. Once a suggested change to a computer system has been mooted the analyst must determine how it can best be incorporated into the system design. Typically this is not a source of inordinate analysis and design effort. Many changes consist of enhancements to the content and/or presentation of system input or output or even the addition of new forms of input and output. Thus the replacement of a punched-card-input/printed-output batch computer system by an on-line Visual Display Unit input/output system might involve little change to the system structure - only to the timescale of transaction processing by the system. A good system design will be amenable to such changes and it should be fairly obvious how the design philosophy can best be expanded to cater for the alteration.

Once a change to the design has been agreed it must be implemented by changes to the computer programs in the system. In the early days of programming it was frequently the case that the programs themselves were fragile in that they could not be changed easily. In particular this was the case where they were written in low level languages and used 'tricky' techniques to minimise program execution time or memory size. Often the clever trick of one programmer would become the millstone round the neck of a maintenance programmer several years later. Discipline in programming practices, especially with well defined programming standards together with the widespread use of structured and modular programming in high level languages, has significantly reduced this problem in recent years. Nevertheless the most fundamental systems of many organisations tended to be developed many years ago and there are many legacies of programs of this type.

The most time consuming source of maintenance effort comes from the requirement to ensure that each change is adequately tested and (perhaps most importantly) has no adverse effect on all of the existing functions of the system. The establishment of comprehensive system test beds are vital to this operation but each successive change to a system will require execution of an adequate set of tests to prove that no existing function of the system has inadvertantly been corrupted together with additional tests specific to the change being made. The time required to implement a change to a system is therefore dependant on the complexity of the system as well as the complexity of the change.

There is therefore most scope for reducing the time taken to amend computer systems by the introduction of techniques which minimise the effort to re-test the entire system. Techniques which compartmentalise the system such that the extent of testing is limited are desirable and Data Base Management Systems provide such a technique in the area of the storage of data on secondary media.

Change as Seen by Data Base Management Systems.

The use of a Data Base Management System does not remove the requirement to change computer systems. Exactly the opposite - the presence of the DBMS encourages users to think of the system as flexible and they are therefore even less reluctant to propose enhancements. One of the reasons why organisations install a Data Base Management System in the first place is with the intention of making change less traumatic for data processing staff and end users alike. Whether existing DBMS offerings achieve this objective is a matter of debate but is it evident that Data Base Management Systems should have some contribution to make to the smooth implementation of system changes. Preferably changes should be transparent to all programs with the exception of those programs whose changes dictated the data base change in the first place. Even these programs should be presented with a realistic migration path to ease the effort required for their modification.

The classes of data independence offered by Data Base Management Systems are intimately associated with the reasons why changes to application systems are postulated.

The requirement for Physical Data Independence stems from the motivation to alter systems because they consume computer resources in some way which is contrary to the interests of the computer installation as a whole. The pressure for change normally comes from the computer operations staff and in particular from the Data Base Administrator in his resource monitoring role. It may be that the monitoring has identified a system which consumes more than its share of processing or input/output resources and overtures would be made to ascertain whether changes could be made such that the system could operate in a different fashion and thereby consume less resources (possibly at the expense of overall run times). Date Base Management Systems permit certain of these types of changes without change to the programs themselves and therefore remove the requirement to re-test the system. Occasionally the users of a system may observe that critical response or deadline constraints are not being met and will therefore suggest a system change. Such a change might imply that the system could respond more satisfactorily if it utilised its processing or input/output resources differently. Once again the Physical Data Independence of Data Base Management Systems promises some measure of resilience of applications to the reorganisations necessary to accommodate such changes. Since the logic in the application systems is not considered to be unsatisfactory in either of the above situations it must be considered as a constraint on the design of any DBMS that as much flexibility in this area is provided as possible. This applies equally to Centralised Data Bases and to Devolved Data Bases. The mapping of the (conceptual) schema onto the (internal) storage schema is generally the vehicle for achieving this program transparency. The more sophisticated this mapping the more changes can be made without the knowledge of either the application programmers or the end users. The Data Base Administrator can monitor the usage of the DBMS and reorganise as necessary until an optimum operational environment can be achieved. It is feasible to design the data base management system such that it organises its own data storage and thus achieves this optimisation automatically /STOCKER & DEARNLEY/ /BATORY/.

There are other types of change which result from an inadequacy of computer resources and these often require stored data structure to be altered:

The computer used to process the data may change.

The operating system may change or be upgraded.

The Data Base Management System may change or be upgraded.

The medium used to store the data may be altered such that access has different characteristics.

The physical disc pack used to store the data may fail or require backup.

The data may be required in a more economical format (better clustered - less unutilised free space etc - garbage collection).

The techniques used to represent inter-record relationships may be revised in the light of usage.

Techniques to handle these types of change are common and although the causes of the changes are very different the processes for implementing them may use common logic. In particular some of the routines used by the DBMS for accessing data on the data base may be equally valid for reorganising that data. An example would be the logic to avoid deadlock which would be necessary if a record is being modified because it is being reorganised or if it is being updated by an application program.

Logical Data Independence is not concerned with the consumption of computer resources but, rather with the evolution of computer systems to reflect the changing requirements of the end users. Within this context it is apparent that two different philosophies have developed (either by accident or by design) and this is why the categorisations of Centralised and Devolved Data Bases have been suggested.

The Devolved Data Base approach tends to advocate a stable data base environment for each restricted community of users of each devolved data base. It is particularly important that queries can be formulated in a consistent manner which reflects the structure of the enquirer's perceived universe which is not envisaged as altering significantly as time passes.

The Centralised Data Base Management System cannot rely on a stable perceived universe and correspondingly stable programs since it must cater for a multiplicity of users. The types of DBMS designed for this environment limit the perception of each application program by the sub-schema construct but retain its universal applicability by ensuring that each sub-schema is a subset of the schema. Any record or set quoted in a sub-schema must have a corresponding entry in the schema albeit that that entry contains additional member record types (for a set) or additional data items (for a record). If an application system change is identified which requires a change in the sub-schema used by its programs it may therefore be necessary to alter the schema to maintain the continuity between the two. But altering the schema must imply also the alteration of any sub-schemas covering the area being changed and this in turn would require modification of the programs which used those sub-schemas. Restructuring in a Centralised Data Base environment is therefore a facility provided to aid the productivity of the application programmers when they modify computer systems in response to the changing requirements of the users of those systems. It should minimise the effort to modify the programs and by restricting the areas of change should reduce the extent of the retesting required to demonstrate that the new structure adequately supports all applications. Although the restructuring is performed by the Data Base Administrator it is not carried out at his behest and he personally receives no tangible benefit from it. It is a more complex subject than reorganisation and must be viewed in conjunction with other aspects of application systems evolution.

Restructuring a Devolved Data Base.

We have seen that the stability of the perceived universe of any particular user corresponding to his own devolved data base reduces the frequency of restructuring of that data base. The definition of the devolved data base ensures that the changing requirements of other individuals within the same organisation have no impact. The three sections of the devolved data base support this stability to different extents:

The abstraction of the centralised data base will alter only very rarely. This reflects the inherent stability of the corporate goal of the organisation. Although each section within it may be a microcosm of change the overall objectives of most organisations are well defined - to profit in a section of commerce where the organisation has expertise, to manufacture some class of artifact etc. - and the commonly used items of data within this framework will remain constant. Even when this data changes it will be possible to maintain the stability of the abstraction by altering the mapping from centralised to devolved data base to compensate for the change. Alternatively if an application program is providing the interface it can be altered by the data processing staff to reflect the new structure of centralised data base while retaining the same output to the devolved data base. This situation has a parallel in the way a centralised application would alter to reflect a structural change while retaining its existing output (e.g. printouts, displays etc.). In a banking example a program to identify all customers who had exceeded their overdraft limit and present their information to a devolved data base for the use of the lending control department would require change if the limit were to be applied on a customer (rather than account) basis but it must still identify the correct deviant accounts.

Occasionally a change to the central data base will necessitate an alteration to the devolved data base user's perceived universe and thus to the abstraction of the centralised data base. In such circumstances the end user must alter his programs if they are to reflect what he now considers to be the "true" structure of the organisation's data.

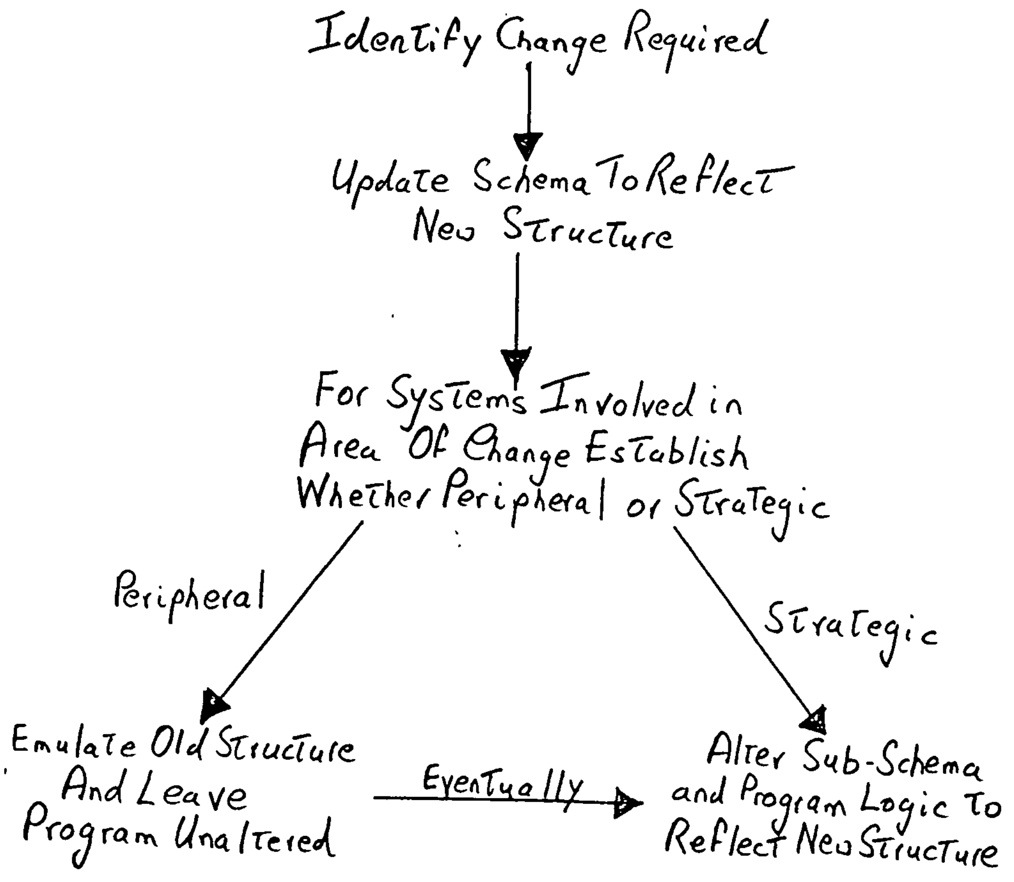
Devolved data base users are also entitled to expect a degree of stability for the section of their data base giving information on the "outside world". They must decide what information is to be considered within their universe and it is therefore their own prerogative to ignore any new information which becomes available if they do not consider it relevant. If the outside world does undergo an irrevocable change there is little alternative but to incorporate the change into the devolved data base and make appropriate changes to the end user's programs.

The third section of a Devolved Data Base, where the user captures and updates the data himself, is likely to alter from time to time at the user's discretion. This data, however, is the sole responsibility of the end user and is outwith the control of the Data Base Administrator. As such, its change cannot impact on any other users and change to structure with appropriate change to programs is an exercise which can be undertaken by the end user himself in isolation from all other end users.

Just as it was observed that many of the routines required by the software to reorganise a data base for any of a multiplicity of reasons would be commonly used it is likely that structural changes could be achieved within the general design philosophy of a devolved data base. It is the motivation behind the change which is different and this may indicate a shift in the priorities of the software.

Restructuring a Centralised Data Base.

This thesis is primarily concerned with the restructuring of the single centralised data base maintained as a common resource for a wide spectrum of end users by application programs written by professional data processing staff. The data processing department may be considered as a service centre within the organisation which is prepared to bear some inconvenience when the data base is restructured in the interests of keeping their entire library of programs operating on a single consistent data base structure. It would often be in the interests of the data processing department to view its data model of the organisation as a continuously altering but instantaneously accurate picture of the "current" state of the organisation. A universally applicable data base management system model is desirable at this level in preference to a plethora of such models which reflected the view of the organisation at times in the past when individual systems were implemented or modified. This approach is particularly true for the 'strategic' systems of the installation which carry out important roles such as data value update but as a consequence of their central position within the framework of systems require modification fairly frequently. Other more peripheral programs within systems might present less pressure for modification if this can be avoided. In addition to the provision of facilities to restructure the data base and to allow appropriate changes to programs the restructuring mechanism must therefore provide for emulating historical states of the data base structure against the current state. This emulation would allow decisions to be taken as to which programs to update but this is considered as a less desirable approach than modification of all systems at the time of the restructuring since it defers program change rather than dispenses with it. The decision as to whether a program should be considered as strategic or peripheral should be the province of the Application Administrator since long term convenience must be weighed against short term implementation cost and it is important that the result is not coloured by the interests of the individual entrusted with the system modification which necessitated the the change in the data base structure. To enable a phased implementation of system changes the restructuring mechanism should provide for the possibility of considering some strategic programs as non-critical such that they are modified at some convenient time after the restructuring.


Identify Change Required
Update Schema To Reflect New Structure
For Systems Involved in Area Of Change Establish Whether Peripheral or Strategic
Peripheral
Strategic
Emulate Old Structure And Leave Program Unaltered
Eventually
Alter Sub-Schema and Program Logic To Reflect New Structure

The timing of changes can be more critical in a centralised data base than in a devolved data base since rather than a single user or closely associated group of users there will be many users all with different claims on the availability of the data. Where enhancements to a system are the cause of the restructuring it is reasonable to expect users of that system to sacrifice availability of their data for some time but where a system is not altering in its function it is more difficult to present such arguments for unavailability. In some circumstances the option of making data unavailable is just not feasible because of the nature of the data base - one only has to consider Airline Reservation Systems and Bank data bases supporting 24-hour automated teller machines. Such considerations apply not only to the restructuring of these types of data base but also to virtually the entire spectrum of their reorganisation /LUCKING-74B/. Thus there may be no available time slot when a restructuring operation may be run on some centralised data bases to the exclusion of normal program access. Furthermore the nature of the programs and data involved in certain fundamental data base structural changes would often demand that a period of parallel running would be required to convince auditors that the change was having the desired effect.

Restructuring Strategies.

In situations where a data base can be made unavailable to its user(s) for some period of time the most straightforward approach to restructuring would be carry it out in discrete stages. This would normally be appropriate for any devolved data base and in many occasions for a centralised data base. The following steps would be typical:-

(a) Unload the data base in accordance with the corresponding schema to some backup storage medium (e.g. magnetic tape).

(b) Alter the schema to reflect the desired change.

(c) Alter the data base backup to ensure that it corresponds with the revised schema.

(d) Reload the data base from the backup in accordance with the revised schema.

Such a process is an example of a Static Restructuring Strategy. A variation may be obtained by operating on the data base in-situ rather than on a backup copy but this may present practical difficulties for certain types of change. Some of the operations in this procedure are convenient since they are required in any DBMS to facilitate the integrity of the data by occasional archive and recovery as required. Garbage collection, too, is convenient using this approach since it is easier to position data efficiently when it is reloaded in its entirety.

There are some types of static restructuring where data is still available to end users for the duration of the restructuring. The data base, for example, may be archived while it is still being updated /IBM-DBRC/ or certain access paths may be altered while the data base is on-line /VAO/.

Where a restructuring mechanism is implemented which allows the Schema, the application programs and the data base itself to alter while the data base continues to be accessed by application programs (including those being altered) that mechanism may be said to provide a Dynamic Restructuring Strategy.

Principles of Restructuring.

A Restructuring Mechanism (Static or Dynamic) must operate on the three elements of the Data Base Management System.

(a) The Schema.

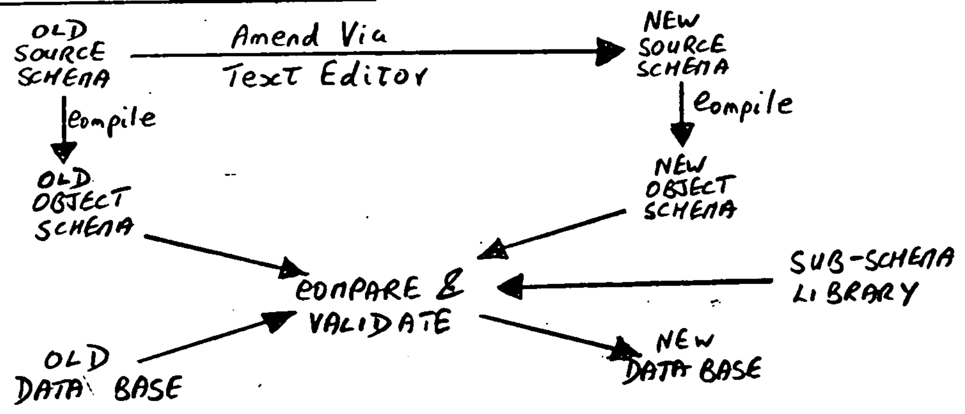
(b) The Data Base (possibly in the form of a back up copy).

(c) The Application Programs.

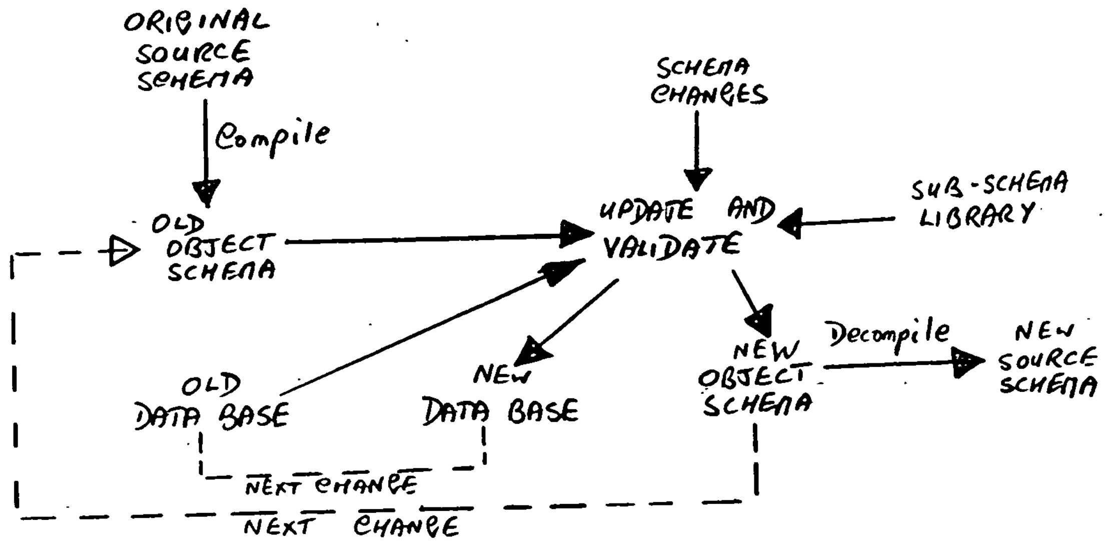
The cause of the restructuring is a requirement for program modification in one or more application systems which establishes the revised structure of the data base and, therefore, its description within the schema. The change in turn may require changes to other application systems. We have seen that the schema has both a source and object manifestation as do application programs. The established technique for altering application programs is to alter the source and recompile to produce a revised object version. The use of text editors has made this operation relatively straightforward. It does, however, imply a lack of continuity between each succeeding version of a program and it is this lack of control over program change which requires exhaustive system testing after each modification. The close relationship between the schema, sub-schema and data base makes the problem of continuity even more acute if the Schema is to be amended. The schema must always reflect the actual structure of the data base and the sub-schemas must always be subsets of the schema. No implementation of a data base management system can permit an error to be made such that there is an anomaly in this situation. But the more tightly the DBMS can preserve this continuity the less responsibility is left for demonstrating its accuracy by system retesting.

The following approaches might be used by data base systems to achieve this end.

Approach 1 - Schema Recompilation



Approach 2 - Object Schema Modification



The first approach therefore has the advantage that it utilises an existing item of software in the text editor in addition to two utilities which the DBMS must provide (viz: the Schema Compiler and a Compare and Validity utility). The second procedure uses 3 DBMS utilities (Schema Compiler, Update and Validate utility and a Decompiler). Note that schema compilation is only required once in the lifetime of a data base since the object schema it creates is subsequently updated by successive runs of the Update and Validate Utility. In the circumstances it is reasonable to suppose that the compiler would also be used to set up an initial data base (i.e. a data base with no records or inter-record relationships but with sufficient "hooks" to allow application programs to add record occurrences when they eventually run). The Schema Changes can be described in a language similar to that used for the original definition of the Source Schema and it is therefore likely that some of the logic for the Update and Validate Utility could be incorporated from the Schema Compiler.

When the implementor of a Data Base Management System decides how it will incorporate restructuring he must therefore choose one of these alternative approaches. He must consider the relative complexity of the various items of system software as cost constraints within the design of the DBMS but other factors such as the "user friendliness" of a system able to identify errors interactively will also be important. In the past the consideration of restructuring as a peripheral aspect of the overall data base system has tended to favour the "schema amendment and recompilation" approach but there are many attractions to the "object schema amendment" approach when restructuring is integral to the DBMS design.

Objectives for a Restructuring Mechanism.

A restructuring mechanism must be a versatile tool at the disposal of the Data Base Administrator to allow him to act on behalf of the application systems designers and programmers so that they can modify their systems as circumstances dictate while retaining a single centralised view of the organisation's data base as a shared resource. It must be powerful enough to allow a wide range of logical data structure changes to be performed. It may be that different mechanisms are appropriate for Centralised and Devolved data bases since the objectives for the stability of application programs are somewhat different. A choice of Static or Dynamic strategies must be available so that the Data Base Administrator can take the desirability of continuous availability of the section of data base to be restructured into consideration.

The mechanism must be functional. It must serve the practical needs of the Data Base Administrator in that it will permit him to perform common changes conveniently. Like the design of a programming language the design of the restructuring mechanism will be a compromise between the provision of a limited set of primitive constructs with many invocations of each construct required to describe a desired change and the provision of a complex set of operations which has the potential to specify restructurings concisely. In the latter case the required education of the Data Base Administrator in the language could become the critical factor.

The mechanism must be convenient. The Data Base Administrator must be able to specify his restructuring requirements in a straightforward manner while at the same time being allowed to exercise his discretion as to the most appropriate strategy for the change being made. He must also take into consideration the status of the data base at the time of the change.

The mechanism must be efficient. Restructuring is always a means to an end rather than an end in itself and it is therefore incumbent on the restructuring mechanism to consume as little computer resources as necessary. A Static Strategy must allow the restructuring to complete within some target timescale which is acceptable to the end users who have been deprived of access to their data base. A Dynamic Strategy must allow application programs to continue to access the data base while it is under way with any degradation of response during this period being kept within agreed limits.

Chapter 5 - Other Approaches to Data Base Change.

The process of continual evolution of the logical structure of a centralised data base together with the parallel process of evolution of the application programs operating on it is the particular aspect of data base change addressed in this thesis. From time to time the data base must be restructured so that it can continue to be an accurate model of the organisation which maintains it. In particular the CODASYL proposals for a data base management system suggest a definition of restructuring as the operation of altering the data base schema and the consequences thereof (as opposed to the alteration of the storage schema (reorganisation) or the sub-schemas).

Although little research work has so far been concentrated into this type of change within data base management systems there are several sources where developments have been described in closely related areas. This chapter considers those areas where there is some measure of commonality and notes where previously described techniques will be applicable. In some cases differences in approach between the proposals described here and those described elsewhere are highlighted.

The Spectrum of Data Base Change.

We have seen in Chapter 2 that a data base may consist of 4 types of schema:

a) The Conceptual Schema

b) The Data Base Schema

c) The Storage (or Internal) Schema

d) The Sub-Schemas (or External Schema)

In the hierarchy of mutual dependence of these schemas a change to the storage schema (within certain limitations) should not require changes to the conceptual schema, the data base schema or to any sub-schema. The motivation for such changes would be to reduce performance overheads.

Similarly, certain changes to sub-schemas should affect neither the data base schema nor the storage schema. They would be an inevitable consequence of certain types of enhancements to application programs where a change in the program's 'view' of the data base was required.

Changes to the schema, on the other hand, cannot guarantee to preserve the integrity of the storage schema or the sub-schemas since both are dependant on the structure of the schema for their own structure.

By the same token a change to the Conceptual Schema would normally dictate changes to the Data Base Schema (and consequently changes to the Storage Schema and Sub-schemas) since this is essentially a translation of the Conceptual Schema into the constructs of the Data Description Language of some Data Base Management System.

In a tutorial paper published in 1979, Sockut and Goldberg give an overview of those requirements which they view as necessary for handling changes to a data base

/SOCKUT & GOLDBERG/. They use the term 'reorganisation' as a generic description of the entire spectrum of change from 'restructuring' for changes to logical data structure to 'reformatting' for changes to the way in which data is held on storage media. The spectrum of change is illustrated by several examples ranging from (at the restructuring end) how the data base would be expected to react to changes in the reporting structure of the employees of the organisation modelled in the data base to (at the reformatting end) how clustering techniques for the data on disc could be altered to improve performance. By broadly equating the CODASYL Schema to the ANSI/SPARC Conceptual Schema, Sockut and Goldberg consider restructuring to relate to changes in the definitions of attributes and to changes in the relationships between them. They make the observation that the schema to sub-schema mapping will often imply that application programs need not alter if the schema alters, so long as the sub-schema which they use has remained unchanged. There is also some discussion of the reverse process of altering the sub-schema without changing the schema. The tutorial does not address the motivation behind data base restructuring and in particular does not discuss its close relationship to the process of application system evolution. This contrasts with the approach taken in this thesis which emphasises the need for program evolution in the centralised data base environment while at the same time identifying particular situations where application program changes will not be necessary.

The spectrum of data base changes was also described in the CODASYL Proposals when they were first published in 1971 /CODASYL-71/. In particular the distinction was drawn between changes in the logical data structure described in the schema and the changes necessary from performance or security/integrity considerations. In the subsequent Journals of Development this view has not radically altered /CODASYL-78, CODASYL-81/.

The proposals are mainly concerned with the description of a Data Definition Language (and, in the earliest version, a Data Manipulation Language which has subsequently been moved into particular language specifications) and do not claim to be a complete specification of a DBMS. Thus certain areas such as Restructuring have been mentioned but not described in detail. In particular the requirement of a DBMS to support restructuring is recognised at two points in the report.

a) In section 2.3.6. a System Support Function is postulated which "permits modification of a schema or sub-schema and causes the changes to be reflected in the data base itself. Without such a language, changes to the schema can only be made by developing an entirely new schema and restructuring the data base in accordance with the new schema".

b) In section 2.3.7. the report recognises that the Data Base Administrator must have facilities to allow him to "modify the schema and compile the changes into the object schema (and to) modify the data base to reflect changes in the schema and storage schema".

It has, therefore, been recognised that the schema and sub-schema will alter from time to time and also that it is probably more desirable to have a mechanism which changes the schema and data base together rather than one which alters the data base by examining consecutive editions of the schema. This approach is adopted in the individual restructuring tasks identified in Chapter 7. It is the identification of these tasks which has brought to light the limitations of a Data Definition Language which only permits the description of the data base structure at a single point in time. While recognising the applicability of the style of syntax for data definition described in the proposals, enhancements have been made here to provide a language which describes how the data structure is to alter. The requirement for a description of the current data base structure is considered as a problem of data retrieval from an object version of the schema and not directly related to the process of data capture of the definition of the structure. Although such changes to the proposed DDL are largely cosmetic it is important that the vital significance of a convenient mechanism to support changes to data base structure is made more apparent to data base management system implementors and users alike. The encapsulation of the DDL within a language which supports changing structure underlines the fundamental requirement for the Data Manipulation Language Execution Routines to operate in an environment of changing programs, changing data base structure and changing object schema.

The book "The Codasyl Approach to Data Base Management" /OLLE/ devotes a chapter to Restructuring. It is observed that restructuring is a vital requirement which has been given little attention and is only described in passing in the Data Base Task Group report. In common with the view expressed in this thesis, restructuring is defined (in the CODASYL context) as amendment to the schema while reorganisation is an amendment to the storage schema. Garbage Collection is considered in its own right as a utility which can operate on the data base without change to either the Schema or the Storage Schema. Elementary restructuring tasks are identified in much the same way as those of Chapter 7. They are classified as additive, subtractive and modificational but they are not described in any detail nor is their impact on application programs discussed. Olle admits that Restructuring is a topic which is "...often mentioned but never discussed in detail". This thesis is intended to provide such detail.

There is evidence to support the view that other researchers have also recognised the relationship between Restructuring and the information architecture aspect of the maintenance of a centralised data base /KAY/. Conversely the relevance of reorganisation to system performance (in both centralised and developed environments) is apparent. Facilities to generate the revised schema and to populate the data base using a mapping from old to new schema are required. Kay also suggests that research is needed into:

a) Impact on Sub-Schemas and Programs.

b) Impact on the Data Storage Description Language. In particular can this be automatically generated from changes in the schema?

c) Impact on the Data Base itself. What are the criteria for it to change in the event of particular types of restructuring?

The detailed consideration of the individual restructuring tasks in Chapter 7 particularly addresses (a) and (c) although the importance of (b) must also be recognised.

The CODASYL framework has been said to support an intuitive concept of entities, attributes and relationships /TAYLOR and FRANK/. They reflect the ideas which have evolved with system design over the years and in many practical situations it is useful to consider certain "things" as concrete entities and other "things" as abstract relationships. As a consequence this flexible model of the organisation, although easily understood by computer professionals and end users alike, is rather volatile and a volatile model must be amenable to change. Furthermore the facility to describe the same logical structure in more that one way (eg Repeating Groups and Sets) implies that there will be a requirement to move from one representation to another as the modeller's perception of the "best" approach alters.

Logical Changes and Structural Changes.

Restructuring is generally considered to cover changes in the Data Base Schema required as a consequence of changes in the Conceptual Schema in addition to changes on the Data Base Schema which reflect decisions to alter how this will be mapped from an unchanged Conceptual Schema. The restructuring as a consequence of the change of mapping has received attention from some researchers in the recent past. NAVATHE, for example, sees restructuring as "...a tool designed to increase the latitude of a database user with respect to the choice of structure. This class of change has been particularly attractive since it is evident that since the logical structure of the data has not changed there is no theoretical requirement for application programs to change. In practice, however, the close relationship between program and schema has implied changes and suggestions have been made for automatic program amendment techniques. Although such changes only cover part of the structural changes discussed in Chapter 7 they are reflected in semantically equivalent sequences of DML commands and the work of other researchers will undoubtedly be relevant in this context.

Sockut and Goldberg discuss this particular aspect of restructuring at some length in their tutorial. They suggest that the Conceptual Schema is the model which is free of all physical storage constraints and its amendment is therefore equivalent to "pure" restructuring. In individual implementations it would be difficult to formulate such a clear definition since constructs which would possibly be more appropriate at a lower level tend to be included at higher levels for the convenience of the implementation. Like Olle's approach, they suggest that attributes can be added, deleted, combined, split or renamed and they further categorise the important class of change where an attribute migrates from one record to another as "Reorganisation on the String Level". Examples are given of changes in structure which can be achieved by introducing new classifications of records and migrating data items to these records.

Where a set has an owner record of "Project" and a member record of "employee" it may be that each employee record contains a data item of "longevity" which indicates whether that employee is Permanent or Temporary. It is therefore feasible to alter this structure by giving the set two types of member records (viz "Permanent Employee" and "Temporary Employee") and thus dispense with the data item. This change can, of course, take place in either direction and a decision to undertake it is more likely to be influenced by programming considerations than changes in the organisational model. Typically it might become apparent that most application programs only require access to the permanent employees and it would be more convenient to have these grouped as distinct set members rather than always demanding a test on the Longevity data item to determine whether a particular data item is of interest.

In a thesis for the University of Aberdeen WILSON also identifies the requirement to switch between logically equivalent data structures. The relationship between pupils and teachers at a school may either be represented as a set

"Teaches" with owner of "Teacher" and member of "Pupil" where more than one pupil record may actually relate to the same teacher. Alternatively the same relationship may be represented by an "Is Taught By" set where the owner record is "Pupil" and the member record is "Teacher". In this case there will be more than one teacher record for the same pupil. A detailed example is also included for a data base of sheep and the multiplicity of relationships caused by their breeding. Although the Conceptual model in this instance is both simple and stable (viz:- a ram and a ewe are related by a mating to produce other sheep) there are several alternative data base models for converting this relationship into the set construct (IDS in this case). It is desirable to be able to switch from one data base model to another on the light of experience gained from the types of application programs required to analyse the data base.

Hierarchical Data Base Management Systems provide a structure which is amenable to certain types of change without impacting on the formulation of queries /DALE & DALE/. Thus if a query relates to data items in a cascade of nodes within the hierarchy some re-ordering of these nodes may take place without altering the query.

The criterion for allowable restructuring operations is based on the concept of a "Broom Set" of nodes for each node within the hierarchy. These are both the "Ancestor" and "Descendant" nodes of the node itself. Thus in the diagram the Broom set of B are A, D and E

$$\begin{tikzcd}
 & & & A \arrow[dll, dash, dotted] \arrow[drr, dash, dotted] & & & \\
 & B \arrow[dl, dash, dotted] \arrow[dr, dash, dotted] & & & & C \arrow[dl, dash, dotted] \arrow[dr, dash, dotted] & \\
D & & E & & F & & G
\end{tikzcd}$$

A restructuring is allowable if every broom set in the original structure exists at least as a subset of a broom set in the new structure.

NAVATHE and FRY have considered several equivalent structures where hierarchies are embedded into more general network structures. They consider such transformations as Compression (replacing two consecutive levels in a hierarchy by a single level), Assembly Merging (replacing twin segments at the same level by a single segment) and Inversion (the inherent relationship which exists between each lower level in a hierarchy and the levels above it to form another hierarchy). They recognise that schema modification is just one type of data base change and make the point that demarcation between restructuring and data processing is fuzzy.

Application Program Stability.

To allow application programs to remain intact in spite of a restructuring is a legitimate objective since it will eliminate the expensive and time consuming exercise of program modification and re-testing. However in a centralised data base environment this objective must be balanced against the sometimes contradictory objective of having all application programs operate on subsets of the same basic data structure. This objective of a unified data structure is less apparent when applied to the collection of devolved data bases linked to the same centralised data base. The objective here is more for the stability of each devolved data base (and the application programs which run on it) despite changes on the corresponding centralised data base.

The Proposals for restructuring tasks in Chapter 7 identify the criteria which may be applied to an application program to determine whether it will be stable under that task but, equally importantly, provide a route whereby application programs which must change are allowed to do so in a controlled manner without a moratorium on running them during the restructuring.

Other researchers have concentrated their attention on techniques with the principal objective of preserving application program stability.

An in-place restructure has been implemented at the University of Pennsylvania /GERRITSEN & MORGAN/ /BEAVER/ which allows the data base schema to evolve but which leaves any record occurrences existing at the time of the change in the format corresponding to the schema applicable when they were written (a process similar to that required for open dynamic restructuring). Programs, too, may be at various levels corresponding to schemas which were operative in the past. The system then carries out a continual emulation of records and program-access-paths by performing up to two translations each time a record occurrence is accessed during an application program run. The first translation is to a corresponding record (c-record) which is in the format defined in the current schema. The second translation is from the c-record to the generation in force when the program was compiled. Thus if n versions of the schema have existed in the past the number of possible transformations is 2(n-1) rather than n(n-1) which would be possible if a single translation were made from the actual generation of the record to the generation of the program which required it. The limitation on the complexity of transformation simplifies the implementation of the system but it does impose significant overheads each time a data base record not in the current schema format is accessed by a program compiled under a previous schema. Limitations are placed on the types of change which are permitted by use of the primitive operations of INCORPORATE, EXCISE and CHANGE on sets, records and data items.

Relational data base management systems provide some measure of stability to the views of the entire data base held by application programs /ARORA & CARLSON/. Because of the essential simplicity of the concept of a relation and the lack of a construct corresponding to a set it is likely that the entire data base can evolve by continually expanding the scope of the base relations. The derived relations used by application programs would remain stable since they would be projections of the base relations. Certain classes of restructuring are said to possess a "Loss-less property" where any previous relation can be recreated by the natural join of relations created by the restructuring. That is any derived relations created from the previous base relation can still be created from the join of the new base relations.

Another proposed technique which would lead to the increased stability of programs is the concept of record sub-types /PALMER/. Where different types of entity have several properties in common it may be convenient to have a generic name which can describe their similar qualities in addition to the specific name for that type of record. Application programs would not reference the generic name and this would allow a more concise description of the data in the schema. It would also permit new sub-types to be added without disturbing the existing sub-types.

Changes to Physical Rather than Logical Structure.

The main thrust in proposing and implementing mechanisms for data base change has been in the area of reorganisation of the way in which data is held on its storage media without altering its logical relationships and therefore without altering the application programs which run on it. Thus the data base can be tuned to optimise its consumption of computer resources. The CODASYL proposals now provide a convenient distinction between restructuring (altering the schema) and reorganisation (altering the storage schema) but in other data base management systems the differentiation is less clear cut. Research in this area does form the basis of the techniques proposed for restructuring in this thesis since in both situations data base storage records must be manipulated to reflect the new storage schema. If the schema alters then the storage schema must also alter to correspond with it.

The EXPRESS project at IBM /SHU et al/ has tackled an exercise which many computer installations must undertake, that of converting a large number of well established conventional data files into the structures demanded by a data base management system. A source is quoted where 100 ad-hoc COBOL programs were required to convert 29 application files to data base structure. The effort in designing, coding and ensuring the accuracy of such programs is one factor which tends to discourage organisations from moving to data base systems in the first place, especially when existing applications have been running successfully for many years. EXPRESS requires forms to be completed giving a non-procedural description of how the data is to be converted. The structure of the source file and the target data base must be specified using a language 'DEFINE' which is similar in concept to the CODASYL DDL or the DBD of IMS but is more generalised than either so that it can handle the wide variety of structures on application files. The language is oriented towards the hierarchies typical of COBOL files. Within a hierarchy the items of data may be optional, variable length or self-describing and a user-extensible picture facility is provided.

A further language - 'CONVERT' is used to specify how the target structure is to be derived from the source. The primitives of the language are:-

(a) SLICE - Form a flat file from part of the hierarchy.

(b) SELECT - Provide the selection criteria to extract hierarchies from an existing hierarchy while retaining the same form.

(c) GRAFT - Form a large tree by joining two existing trees.

In addition, data may be manipulated by operations such as SUM, MAX, MIN, AVE, SORT.

EXPRESS translates the non-procedural description of the conversion into PL/1 programs which are run in 3 stages as follows:

(a) The READ step

This stage checks for inconsistencies between the specified structure of the source files and the actual structure and also transfers the source files to an internal file in a standardised format.

(b) The CONVERT step

This derives a further file in the standardised format based on the input file and the conversion procedures.

(c) The LOAD step

This uses the facilities of the target system (eg IMS) to load the output of (b) onto the target data bases.

Although the read and convert steps may be combined it is suggested that this might be unwise because of a potentially high error rate.

EXPRESS is therefore designed as a tool to assist in the process of setting up a data base but could also be used to convert from one structure to another or even from one type of data base management system to another (eg IMS to Relational).

The CODASYL Journals of Development in 1978 and 1981 not only describe a Data Storage Description Language for the storage schema but also indicate how that storage schema may be altered to facilitate reorganisation. Like the main text of the journal of development, the appendix describing the DSDL does so on a clause by clause basis but it still provides for a static picture of the data base storage structure rather than a language which allows the Data Base Administrator to indicate how that structure is to alter from its former state to a new (reorganised) state. The significant change in this area between the 1978 and 1981 JOD's is that storage records have been given version numbers much like those proposed here for the main schema. It is not proposed, however, that there must be a connection between schema version number and storage schema version number. They are logically distinct and any connection would be set up purely for the convenience of the implementation. Equally, strategies for reorganisation as Static, Background and Incremental have been identified much like the Static, Open Dynamic and Closed Dynamic strategies for Restructuring proposed here. These alternative strategies all have some merits in different circumstances so that the Data Base Administrator has the ability to determine which will be most appropriate for a particular reorganisation. By mixing the strategies, and in particular by inclusion of the open dynamic (incremental) strategy the data base may contain many versions of the same storage record at the same time. The object version of the storage schema will therefore contain multiple descriptions of each type of record (one for each possible version) each of which must map onto the object version of the schema.

It is evident, therefore, that any implementation of a CODASYL DBMS which supports reorganisation in accordance with the proposals of the JOD could utilise the same techniques at the physical data storage level for restructuring of the schema. But reorganisation is a far more straightforward (if less frequent) procedure as the Journal of Development recognises "Although the schema may change because of changes in the organisation's data or functional requirements, such changes are likely to occur much less frequently than changes to the storage schema made for performance and other reasons". The Data Base Administrator may carry out any type of reorganisation - "in the knowledge that his activities have no effect on the Schema itself and therefore on the application programs running against the data base".

Storage records may differ from logical schema records in either of two ways or by a combination of both:

(a) The schema record may be divided into several subsections with each being held as a unique storage record. Thus if certain data items are generally referenced more frequently than others they can be grouped together and placed on a storage device with a more rapid response time. Storage records for the same schema record are interconnected by means of pointers.

(b) Different occurrences of schema records may require different response times and their partitioning into storage records may reflect this situation.

The discussion on restructuring in this thesis treats schema records as being entire since, where a schema record is represented by more than one storage record, these must be linked so that the schema data structures are preserved. The precedent for this view is taken from the JOD itself where the discussion of set types and the reorganisation of the way in which they are physically represented considers schema records to be entire (ie effectively they have a single address).

The principal tasks which can be undertaken for a reorganisation are:

a) To alter the way in which schema records are partitioned into storage records.

b) To alter the representation of a set from direct pointers to indirect pointers via an index (and vice versa).

c) To alter the method of indexing the storage records for access by key or via set and for clustering or sequencing the records in storage.

Thus, although the work described here was carried out independently of the Data Base Administration Working Group which produced the DSDL, there is a great deal of similarity in the two approaches. Nevertheless restructuring of the schema is undertaken for very different reasons from reorganising the storage schema and, far from always being transparent to application programs, restructuring will often be a consequence of a change to some program and will require changes to other programs to maintain a consistent sub-schema to schema mapping. It is most likely that an implementation which supported both reorganisation and restructuring would use the same routines to modify the contents of the stored data in both situations.

Other research work has addressed the problem of portability of data between data base management systems /FRY & JERIS/. This activity would probably be as traumatic as conversion to data base in the first place but as more sophisticated systems are developed in years to come it will undoubtedly be necessary to convert from time to time. The Data Translation Project has concentrated on the development of a language for the definition of any data structure. With such a language at least the vehicle will exist for mapping from the old structure to a target (standard) definition and from there to the new structure. The sequence would be Read-Restructure-Write /MERTEN & FRY/.

Restructuring Techniques.

The techniques proposed for handling the various restructuring tasks under both open and closed dynamic restructuring strategies in Chapter 7 are based on the allocation of a version number to each record occurrence and corresponding sub-lists in the object schema for each version. For each task, sequences of operations are described which allow application programs to continue to operate during restructuring albeit that some of these programs will alter to reflect the revised structure of the data.

The desirability of continued application program availability during both reorganisation and restructuring has been widely recognised. Sockut and Goldberg devote a proportion of their tutorial to the discussion of such techniques. They firstly examine possible techniques for static reorganisation as either "in place" or "unload and reload" but they recognise that application program access to the sections of data base being updated is likely to be prohibited. The possibility of concurrent reorganisation is then discussed and although it is recognised as feasible the authors identify it as a possibly excessive consumer of computer resources. They make the distinction that a very large data base is "one whose reorganisation by reloading would take longer than the users can afford to have the database unavailable". Such data bases are typical of the centralised data bases where restructuring will be a vital requirement.

Sockut's studies of the performance of concurrent reorganisations /SOCKUT-78/ predict some degradation in user response time. For an Incremental Reorganisation Sockut has used the Seek, Latency and Read/Write times of a disc drive to measure the degredation for a single-disc data base. By assuming a Markov Chain Queueing Model, Sockut has used a Stochastic Process to measure the Expected User Response Time and Expected Reorganisation Time for different User Arrival Rates. His conclusions are:

1. Results generally agree with intuition as follows:

2. As user utilisation increases then user response time and reoganisation time increase. Therefore, if the user load varies, (eg it is low at night) then reorganisation should be performed during slack periods.

3. As the amount of work performed in one reorganisation step increases (ie the number of records reorganised before an interrupt caused by a user access is serviced) the user response time increases but the time taken to reorganise the entire data base decreases.

4. For typical values of User Arrival Rate and Work Performed by Each Reorganisation Step both User Response Time and Reorganisation Time have values which can be considered as reasonable in many situations.

Degradation is inevitable since overheads such as amendments to pointers and locations of records are being incurred which would not exist if the reorganisation were not being performed. The important aspect is not so much that there are overheads but that they can be channeled in such a way as to utilise resources which would otherwise be wasted and at the same time do not appreciably degrade response time for application programs. The techniques of open and closed dynamic restructuring do just that. For an open restructuring there is an additional processing overhead in manipulating data to the up-to-date format if it is retrieved but there is no additional Input/Output overhead. For a closed restructuring it is important that the restructuring always has a lower priority than any application program so the the additional Input/Output operations of progressing through the data base in the preferred direction does not delay an Input/Output access request by an application program. Restructuring is essentially an ad-hoc operation carried out in response to an identified change in data structure. So long as the resultant overhead is minimal (even if it is apparent) the Data Base Administrator should be able to persuade users that it is the price they will have to pay (for a limited period of time) for the structural enhancement. Reorganisation, on the other hand, is identified by Sockut and Goldberg as an ongoing requirement to tune the overall performance of the data base with no obvious benefit to individual users. The Data Base Administrator would therefore find it more difficult to "sell" any degradation of response time.

The system which has been implemented at the University of Pennsylvania supports concurrent reorganisation. Although it is claimed to be "restructuring" in its literature this is something of a misnomer at least in the terminology of the CODASYL proposals since the essential structure of programs and data base is maintained. The system supports nothing akin to Closed Dynamic Restructuring and appears to make no distinction between programs which are transparent to the change and those which must alter as a consequence.

Wilson also describes techniques which would be used to provide a restructuring mechanism. The precepts of his approach, however, are rather different. By recognising that data base management systems tend to be massive items of software he utilises the facilities provided by the DBMS itself to provide source, transitional and target schemas. Given that a source schema exists and that a target schema can be identified which will describe the desired data base structure, Wilson proposes a Transition Schema Synthesis Language (TSSL). This would enable the Data Base Administrator to describe an interim transition schema which encompasses both structures and at the same time would generate code to convert from the source schema to the transition schema and from the transition schema to the target schema. A further Restructure Control Language would then be used to group the generated code into restructure programs and to generate a stream of runs of these programs to effect the restructuring.

The Data Base Control System can therefore remain largely intact since the programs generated for the restructure are executed as run units just like application program run units. By judicious use of established data base concepts such as sets with owner of SYSTEM, data base procedures to alter data item format, and SOURCE and RESULT data items to migrate data between owner and member records of sets, an environment is set up which guarantees the integrity of the data by relying on the existing facilities provided by the DBMS for this purpose.

By demonstrating that coexistent restructuring is feasible and can be implemented without prohibitive degradation to existing applications by the expedient of sectionalising the activities into a number of discrete run units, Wilson has underlined the relevance of research in this area. The approach of this thesis, however, is somewhat different in that it does not assume any contraints imposed by the structure of existing systems. Rather it assumes that, to allow application programs to evolve so that they reflect the new structure while at the same time continuing to operate for the duration of the restructuring, significant DBMS redesign would be required.

Facilities for Change in Existing Implementations.

Both Sockut And Goldberg and Wilson have described the reorganisation facilities provided by several existing implementations of data base management systems. They also recognise that very little is currently provided by way of restructuring although the distinction between the two items is particularly vague for existing software since implementations do not tend to support a separate storage schema but rather have many of their storage concepts defined in the schema. I have written to several manufacturers recently (late 1982) and their response indicates that Wilson's conclusions are still valid that

"...it is necessary to perform certain restructures in stages interposing one-off application programs between each stage" /IDMS/,/ADABAS/,/TOTAL/,/IBM-IMS/.

The techniques used by IMS for "off-line" restructuring will now be described as a typical example of how far existing implementations facilitate change.

IMS supports hierarchical data base structures and an organisational model will therefore normally consist of a number of data bases. The full network can be modelled by superimposing logical data base structures which transcend the physical data base structures by means of pointers between them. The structure of both the logical and physical data bases is defined on the Data Base Description (DBD) library and this is therefore roughly equivalent to the CODASYL Schema.

The process for restructuring an IMS data base operates as follows:-

An Image Copy unload program is run which uses the Data Base Description to write the physical data base (together with its pointers for logical data bases) to a standard format of sequential file. The DBD Library Modification Program is then used to modify the Data Base Description as required. Finally the Reload Program uses the new DBD to carry out any necessary modifications to the data base before recreating it on disc. No updates may be carried out on the data base by application programs for the duration of this exercise.

Functions which may be performed on the data base are as follows:-

(a) An existing segment (ie Record) type can be deleted from the DBD provided all segments of this type were deleted from the data base prior to the execution of the Unload Utility.

(b) New segment types can be added to the DBD provided they do not change either the hierarchic relationship among existing segment types or the concatenated keys of logically related segments.

(c) Any field (ie Data Item) except the one for the sequence field of a segment can be changed, added or deleted. No attempt is made by IMS, however, to alter the data content of a segment.

(d) Existing segment lengths can be changed. IMS cannot alter the data content, however, except to truncate data if the segment is made smaller.

These restructuring functions are consistent with the level of transparency offered to application programs by IMS when they access a data base. Each program must have a corresponding Program Specification Block (PSB) on the PSB Library. Each PSB consists of a number of Program Communication Blocks (PCB) with one such PCB corresponding to each data base accessed. Thus the PSB roughly corresponds to the CODASYL Sub-Schema. The PSB gives a view which limits the program's ability to access data (eg it could indicate that the program may read but not write or update the data base) - thus the data base could still be accessed during restructuring.

Application Programs need only be 'sensitive' to certain of the segments of the data base. This allows the restructuring to delete and insert segments transparently to the program. Programs may also be sensitive to segments when they issue subroutine calls to access the data base using the DL/I Sub-language. This is done by qualifying the DL/I Call by Segment Search Arguments giving the name of the segment to be retrieved.

Further, the program need only be 'sensitive' to certain fields in a segment. This allows fields to be deleted and inserted transparently to the program. Unlike CODASYL there is no definition of data item format on the DBD or PCB and there is therefore no question of amendment to data item format.

The process of unloading and reloading a data base can be very swift (perhaps taking only a few minutes) but the usage of pointers to cater for logical data base structures may extend the reload process out of all proportion. The pointers on the reloaded records (i.e. to other physical data bases) will not present any problems but any pointers on other data bases to the data base being reloaded will have to be altered to reflect the new addresses of records on that data base. If this were to be done by direct access to the records containing the pointers there would probably be a significant overhead in disc head movement and therefore in overall execution time. Sorting and overwriting the pointers sequentially could reduce the overhead.

It is overheads such as this that make installations reluctant to reorganise their data base but so long as the overhead is inevitable they would be more prepared to accept (and even encourage) change if it were to be carried out by the consumption of otherwise spare resources and without degradation in response times.

Chapter 6 - A Proposed Restructuring Mechanism For a CODASYL Data Base

We have seen that there is a spectrum of approaches to the provision of a data base management system from those primarily directed to the devolved data bases under the control of their own sophisticated end users to those directed towards the professional data processing community controlling the single centralised data base of the organisation. We have observed that there is a corresponding spectrum in the requirements for a Restructuring Mechanism from one extreme where the desire for application program stability is the prime concern to the other where there is an attraction in an instantaneously accurate but constantly altering data model of the organisation.

The Data Base Management System Model proposed by the CODASYL Data Base Task Group is oriented towards the professional data processing community for use on the centralised data base of each organisation. The structure of the Data Definition Language encourages a single data base within the organisation and the procedural Data Manipulation Language embedded in common host languages like COBOL and FORTRAN is in keeping with the languages which are typically used by data processing professionals.

It is therefore proposed that a Restructuring Mechanism to support a data base set up in accordance with the CODASYL proposals should be directed towards the same community of users. In particular the mechanism must facilitate the maintenance of an evolutionary schema which provides a single comprehensible description of the central data base supported by the application programs of the data processing department. It must assume that there is a Data Base Administrator who is responsible for the accuracy and universal applicability of that model since it is this individual who will perform all restructurings. While retaining as much application program stability as possible, the mechanism must cater for as wide a scope of change as possible, if necessary while allowing an orderly set of changes in application programs.

Structure of the Proposed Restructuring Mechanism.

The proposed mechanism is not a single identifiable item of software in addition to the already defined facilities offered by the CODASYL DBMS. Rather it is a series of refinements to the proposals such that they are enhanced to support an evolving data base rather than the instantaneous picture they currently paint. The object elements of a data base management system have already been described. The proposed restructuring mechanism involves enhancements to the following aspects of these object elements in the CODASYL environment.

(a) The Schema Amendment Language.

The CODASYL proposals describe in detail the Data Description Language used to define the logical structure of the data base. It is proposed that this language be replaced by a Schema Amendment Language which would retain the spirit of the original DDL in both its syntax and level of user (i.e. Data Base Administrator) involvement.

The kernel of the language will continue to consist of clauses similar to those already detailed in the CODASYL Journal of Development. These clauses will specify the structure of the records in the data base in terms of the data items they contain together with details of the network of inter-record relationships using the Set construct. However, rather than describing a fixed picture of the data structure each group of clauses will be prefixed by a further clause describing how the existing data base structure is to alter to accommodate the structure defined by the clauses to follow. Only on the very first run of the Schema Amendment Facilities will there be no schema to update but on this occasion the language would still be meaningful if the syntax were considered to operate on a null schema (i.e. only additive operations would be valid).

Built round the syntax for data structure amendment will be instructions which the Data Base Administrator can supply to determine the order in which the sequence of changes is to be carried out and how each such change will relate to the other changes to be made at the same restructuring run.

Further, the language will allow the Data Base Administrator to specify the strategy he is to adopt for a particular restructuring run. In addition to a simple static strategy it is proposed that two types of dynamic strategy would also be available. This choice will allow the Data Base Administrator the widest possible scope for conducting any particular restructuring in the manner most appropriate to the prevailing circumstances.

(b) The Data Manipulation Language Execution Routines.

It is envisaged that there will be circumstances when application programs are made aware that a restructuring is taking place while they are running. In particular, programs will be classified as transparent or opaque to a particular restructuring depending on whether there is a requirement to alter their logic as a consequence of the restructuring or whether they can remain intact.

When a restructuring is under way there is a requirement for the Data Manipulation Language Execution Routines to be aware of its existence no matter whether the routine is being executed as a result of a DML call from a transparent or an opaque program.

Open and Closed Restructuring Strategies.

The CODASYL approach to a network Data Base Management System proposes a procedural data manipulation language embedded in a host language such as COBOL or FORTRAN. This language supports interaction between program and data base on a record-at-a-time basis. This situation permits the support of two types of dynamic restructuring strategies which can be used at the Data Base Administrator's discretion.

An Open Dynamic Strategy allows the Restructuring Mechanism to effect the required changes to data base records only when the records have been retrieved from the data base for update by a Data Manipulation Language Execution Routine in response to a DML call by some application program. Similarly when new record occurrences are being added to the data base by application programs the restructuring will be taken into account and the appropriate DML execution routine will write the records in the revised format. This strategy has the advantage that it takes the opportunity to restructure individual record occurrences after these have been retrieved from the data base on its secondary storage medium (an action that was necessary in any case) to be updated and replaced on that medium (a further action required in any case). The strategy therefore means that the Restructuring Mechanism imposes no additional Input/Output overhead on the DBMS as a whole and this may therefore act as a powerful justification for its use in situations where I/O efficiency is at a premium. Typical instances would be where a large data base was subject to fairly frequent and widespread update by application programs or alternatively in data bases where access was so infrequent that the majority of records are never likely to be accessed by application programs and the act of restructuring them is not considered worth while. It is evident that at any moment in the duration of an open dynamic restructuring the data base will contain record occurrences in different formats (i.e. those which happen to have been updated and those which do not). It is feasible that more than one restructuring will be under way at the same time since each such restructuring will not be complete until all record occurrences of the types being restructured have been updated by some application program. A convenient method of identifying such differences in format is to allocate a version number to each record occurrence such that all occurrences in the same format have the same value of version number. Although the open strategy imposes some processing overheads at the time at which it is initiated (ie the schema must be amended), and no additional input/output overhead at any time, it may well impose small processing overheads on the DBMS Data Manipulation Language Execution Routines for a considerable period of time. This prolonged overhead is due to the continuing necessity to examine version numbers on records as they are retrieved from the data base even if only to confirm that the record occurrence is at the "current" version.

A Closed Dynamic Strategy on the other hand operates rather like an application program in that it modifies each record occurrence in turn to reflect the required restructuring. The process would be a background function to the DBMS which would always interrupt the restructuring when necessary to allow an application program to access the data base whenever it requires to do so. The Closed Dynamic Restructuring therefore operates like the lowest priority application program under the control of the DBMS. Like an application program, the restructuring operation must not be permitted to give rise to a deadlock or inconsistent update situation in the data base and the DBMS must therefore take steps to lock each record occurrence from application update between its retrieval and update by the restructuring.

Similarly the restructuring must wait for any record it requires if that record has been locked by the DBMS because an update by an application is pending. The objective of the restructuring mechanism would be to limit the duration of these periods of locking to timescales comparable with those normally experienced due to the interaction of different application programs. For most restructuring tasks the locking period will be the time taken to retrieve, process and update a single record occurrence although in some cases an entire set occurrence will be locked for the time taken to retrieve, process and update each of its constituent record occurrences. It is envisaged that the locking mechanism employed by the DBMS to protect application programs will be adequate to also handle the closed restructuring since it imposes no limitations which do not already apply to application programs using CODASYL DML.

Under the Closed Strategy all record occurrences involved in the restructuring are therefore updated to their revised format within a finite timescale (i.e. the duration of the restructuring run). For the period between the initiation of the restructuring and its completion the data base is said to be in a Transitional State. That is, some record occurrences will have been restructured while others will not.

A further feature of the open strategy is that it may support a period of parallel running while both the old and new formats of data base can co-exist. This interim period takes place after the transitional period and allows the

Data Base Administrator to run Audit Programs to confirm that the data base has indeed been restructured as required and that no data has been lost as a result. When The Data Base Administrator (or more likely the organisation's computer auditors) is satisfied that all is well the parallel running state can be terminated and the data base can resume a stable state - the restructured state. The period of parallel running may also provide an opportunity for the analysts to confirm that their application programs are either producing identical results to those produced prior to the restructuring or that any deviation can be attributed to known factors.

For the Restructuring Mechanism to progress through the desired record occurrences in an orderly fashion in a closed restructuring it is essential that a "Preferred Direction" exists for each record type. This is a path through the record occurrences of that type such that the DBMS and the restructuring mechanism are always aware of how far the restructuring has progressed. One such suitable path would be that provided by a set with the owner of the SYSTEM and member records consisting of all occurrences of the record type in question based on some suitable key data items. An alternative would be simply to progress through the entire data base in physical address sequence and detect each instance of the primary storage record occurrence of the required type. A marker indicating the point along the preferred direction that the restructuring has currently reached would serve many of the functions of the version number in an open strategy. That is, during the transitional phase and the parallel run state the DBMS can ascertain whether a particular record occurrence (which it is required to read or write from the data base in response to a DML command from an application program) has been restructured or not. Certain types of restructuring, however, require the modification of more than one type of record concurrently and the appending of a version number to each record occurrence permits modification to be carried out on one of these types of record in an order different from that record's preferred direction. We shall see in Chapter 6 that this facility is convenient for restructuring tasks involving entire set occurrences.

Application Program Interface.

The objective of the proposed restructuring mechanism for a CODASYL data base is to provide the community of data processing professionals, for whom this type of DBMS is primarily directed, with a tool by which their application programs can evolve in parallel with the evolution of the data base itself. The effect of change on this community has already been discussed in Chapter 1 and it is important that the mechanism for data base restructuring in such an environment tackles the problem of change in such a way as to minimise the activities at the application level while recognising that it is changes to applications which necessitate restructurings. Some measure of involvement at the application level is therefore inevitable for any restructuring.

Nevertheless the most desirable situation for an application program is to remain stable throughout a restructuring and the proposed mechanism ensures that programs will change only when it is essential for them to do so. Programs which have this desirable property of operating during and after a restructuring in exactly the same way as they did beforehand and are classified as TRANSPARENT to that restructuring. It may be that the way in which the DBMS interprets DML commands from the programs is different as a result of the restructuring but this situation does not affect their transparency. The CODASYL Schema to Sub-Schema mapping is the main vehicle for achieving this transparency since it is evident that the formulators of the proposals were well aware of the advantages of restricting the view of the data base for each application program to that subset of the data base that is of interest to that program. Where some part of the data base structure outwith the area viewed by a particular application program and its sub-schema is altered it follows that that program will be transparent. More importantly, however, the proposed restructuring mechanism identifies a wide variety of circumstances in which application programs which operate within the sector of data base being restructured are still transparent. This can be attributed to a combination of the DML commands they issue and the type of restructuring being performed.

Other programs may be OPAQUE to the restructuring being carried out. That is, the logic of these programs must change in some way to reflect the revised data base structure. It is in the amendment to these programs that the analysis and programming effort will be required and where a thorough study of the criterion for and extent of any necessary charge will pay the highest dividend. In the proposed mechanism a pragmatic approach to this subject is adopted such that the types of restructuring which can be undertaken are identified by considering a number of primitive restructuring tasks and for each such task a criterion for transparency (and therefore opacity) is established. It is, of course, possible that some programs will be required to change as a consequence of the restructuring because they are to use the enhanced data base structure in some way. In general, however, changes of this type are less problematical to the programmer since the program can be upgraded at any convenient point in time after the restructuring is complete.

Given that an opaque program must change because of the restructuring, that program must assume at least two states during the process - the original and final versions. For some restructuring tasks the transfer from one state to another is an instantaneous event coinciding with some identifiable event in the execution of the restructuring (when it is initiated, when it is complete, or some point in time in between). For other restructuring tasks, however, there is a period of time during the restructuring when neither the initial or first state of the program is applicable and for which a third (intermediate) state is required. The smooth implementation of a closed restructuring is dependent on the simplicity of transition between these states. It is not proposed to describe how the operating system can schedule the correct version of each program at the correct time (or how this operation can be subjected to satisfactory controls) since such procedures are already well established /PANEXEC/.

However it is worth stressing the importance of adequate testing of application systems prior to this point since the restructuring mechanism does not control the validity of the program change. Testing is still seen as a manual operation requiring the skill of the systems analyst since the input and/or output of the system are also likely to change because of the restructuring in addition to the changes in the data base interaction. Insight is therefore required as to which tests will be necessary to prove that the final version of the program is 'correct'. In some circumstances, however, the program must alter while retaining the same input and output for its community of parametric users. In such situations it may be that one sequence of DML commands in the original version of the program can be replaced without ambiguity by another sequence of DML commands in the final version. The second sequence of commands can be said to be semantically equivalent to the first sequence within the context of the restructuring being preformed. The criterion for semantic equivalence is not straightforward since, in practice, sequences of DML commands in programs may be executed in different ways by use of normal conditional program instructions. Where sequences are unbroken by such instructions it may be possible to replace them by a semantically equivalent set without a manual change to the program and even without any re-testing. Semantic equivalence is alluded to from time to time in the following chapter where the individual restructuring tasks are described.

It is envisaged that Audit Prógrams will be required to assist in the restructuring function so that those within the organisation who are responsible for scrutinising changes to the application systems may be given the opportunity to satisfy themselves that no information has been lost and that no data has been deliberately or accidentally corrupted. These programs will be written specifically for this purpose and they must therefore have available to them the data base in both the old and new structures. The period of parallel running as described previously provides this environment but the programs will differ from normal application programs in that their Data Manipulation Language must provide the ability to specify which structure is to be accessed for a particular command. Even in the parallel running phase it is desirable for an application program to view only one structure - whether that structure is the old or the new will depend on both the type of restructuring and the logic of the program. Occasionally, however, it may be that the requirements of the program in conjunction with the type of processing being performed imply that neither structure is universally applicable and a mixture of both is required - this is particularly true in the transitional phase where the processing required from a program may well depend on whether the record it has accessed has been restructured or not. To allow for these somewhat specialised requirements to view both data base structures concurrently it is proposed that the Data Manipulation Language of the CODASYL Proposals be enhanced to allow the version number of records to be specified in certain instances (notably the FIND command). The default formats of commands would still not require a version number. Thus, for example, an Audit Program could issue a FIND for a version 5 account record with a number 123456 followed by a FIND for a version 6 account record with the same number together with a FIND for the version 6 customer record which if the 'owner' of that account. If the restructuring from version 5 to version 6 was to migrate the balance from the account record to the customer record the program would demonstrate that both values are identical. Where application programs utilise these enhancements to the DML to enable them to quote version numbers the programs are classified as "Version Specific".

Emulation of Previous Schema Versions.

The approach taken by the proposed restructuring mechanism recognises that restructurings are generally necessary as a result of changes to one or more application systems. Although the requirement to change other application systems to cater for the revised structure is less obvious it is argued that it is desirable in order to preserve a single organisational data model for the use of all application programs maintained by an organisation's Data Processing

Department. It must be conceded, however, that the decision not to change an application program (but instead to allow it to operate on an emulated version of the structure which was applicable when it was written) can be justified in some circumstances and must therefore be supported by the proposed restructuriing mechanism. In the context of the terms already defined, this emulation of a previous schema is equivalent to an extended period of parallel running where certain programs operate on the old schema structure.

The proposed restructuring mechanism supports emulation of previous schemas by allowing application programs to be "Implicitly Version Specific". A program is said to be implicitly version specific when the DBMS associates each one of its DML commands automatically with the version number of the record or set type to which it relates which was applicable when the program was compiled (or to be more precise, when it was first put into production). Although the DML commands within the programs do not therefore contain an explicit version number in the way that audit programs must, it is possible for the DBMS to append the numbers applicable at compilation time. So long as a period of parallel running is in operation the same mechanism used for explicit version specific DML commands will ensure that the required old data base structure is presented to such programs.

An extended period of parallel running could be a most expensive overhead for a computer installation in terms of storage capacity and computer processing resources and it is unlikely that emulation in this way would be countenanced for long. A Data Base Administrator would view emulation as an aid to providing a "breathing space" to allow modifications to less critical programs to be scheduled when time is available from the relevant programmers.

If a large number of programs were to be in this category and it was not considered desirable to amend them another approach to emulation would be necessary. One possibility would be to hold record occurrences at their "current" version and translate individual occurrences to previous versions as and when necessary in response to the DML commands from the implicitly version specific programs. Translation from the old structure to the new would also be required to enable data base records to be updated by the version specific programs. Although this approach has been investigated by Wilson at the University of Aberdeen and to some extent by the University of Pennsylvania (See Chapter 5) and might well have to be incorporated into any commercially viable restructuring mechanism, it will not be considered further in this thesis because it caters for a long term multiplicity of program views of the same data base. This is considered as generally undesirable for the Centralised Data Base Management System model used by the systems analysts and programmers to maintain the single centralised data base of the organisation.

Chapter 7 - The Primitive Restructuring Tasks

Introduction

This chapter gives details of 16 primitive restructuring tasks which would be available to the Data Base Administrator to allow him to alter the logical structure of the CODASYL data base schema and the corresponding data records. The tasks have been identified by consideration of each clause in the CODASYL Data Description Language in turn with a view to establishing how the structural element defined by that clause may be expected to change. On this basis it is claimed that the set of tasks is a complete one in the same sense that the CODASYL DDL is complete - (i.e. experience has shown that a language at this level satisfied the majority of requirements for the definition of a data base structure in a concise manner and the primitives are at a sufficiently low level to allow any structure to be defined).

The sufficiency of the set of tasks to define any desired change in structure is satisfied by the inclusion of tasks to add and delete data items, records and sets (since these are the primitives of the CODASYL DDL). It will always be possible to create any desirable structure by the addition of new data items, records or sets followed by the deletion of the data items, records and sets which they supercede. Allocation of actual occurrences of the new data can be achieved by interposing application programs between the addition and deletion exercises to use the about-to-become-redundant data to populate the new data. Such an approach is, of course, less than satisfactory in that specific application programs will have to be written and the timescale of adding, populating and deleting may prohibit runs of normal application programs. Further, since the mapping between schema and sub-schema is achieved by equality of data item name, record name and set name in both, either a mechanism to allow names to be retained would have to be supported or sub-schemas would have to alter. This subject is discussed in further detail under each specific task.

Syntactic Considerations

The CODASYL Proposals describe a Data Definition Language which allows the logical structure of a data base as it exists at some point in time to be defined. The tasks identified in this chapter are described in terms of a Schema Amendment Language which is essentially the same as the DDL but is encompassed with further clauses which describe how the structure is to change - the retained DDL elements will describe the new structure where relevant.

The first example of such a clause is the requirement to give each restructuring exercise carried out on a data base a unique identification - "The Restructure Name". It is envisaged that this would be a useful reference for an Audit Trail whereby at any time in the future the exercise which resulted in a particular section of the entire data base structure can be established. In addition to an identification of the restructure run the Data Base Administrator must indicate the name of the data base he wishes to restructure (this is in keeping with the Data Base Name clause in the DDL) and which strategy he wishes to be used for its implementation. The previous chapter describes alternative strategies.

Thus it is proposed that the initial clause of each run to restructure a data base will be the following

```
Restructure - Name - 1
RESTRUCTURE   Data-Base-Name

USING   ( STATIC                                  )
        (                                         )
       -( OPEN DYNAMIC                            )- STRATEGY
        (                                         )
        ( CLOSED DYNAMIC (WITH PARALLEL RUN)      )
```

Subsidiary Control of Restructuring

The concepts of Open and Closed Dynamic Restructuring, and Parallel Running after a restructuring, create an environment for the Data Base Administrator where he has a legacy of restructurings which were initiated in the past but which still have an impact on the current contents of the data base. The restructuring mechanism must therefore permit him to indicate that these on-going operations are to be considered as complete.

If a Closed Dynamic Strategy with Parallel Run has been used then the Data Base Administrator may indicate that the period of parallel running is to be terminated by presenting the following text to the Restructuring Mechanism.

TERMINATE RESTRUCTURE PARALLEL RUN Restructure-Name-1

In response to this the Restructuring Mechanism must alter the Object Schema by removing all reference to data held in the old structure (this would be done as a matter of course if the closed restructuring were being performed without parallel running as described in detail under the individual tasks later in this chapter). Furthermore all occurrences of data in the old structure which were being maintained in conjunction with corresponding occurrences in the new structure (again see details under individual tasks) would have to be deleted from the data base.

Similarly, if an Open Dynamic Restructuring has been carried out, the Data Base Administrator must reach a point where he considers that all relevant data has been restructured (i.e. it has been updated by some application program). Even if this assumption is correct, the schema will still contain references to this structure which are now, in fact, redundant and these must be removed. This activity can be initiated by specifying.

TERMINATE OPEN RESTRUCTURE Restructure-Name-1

It would be realistic for the Restructuring Mechanism to check that there were actually no occurrences of data records in the old structure before it took the irrevocable step of amending the object schema.

However the Data Base Administrator must be in a position of being confident that there are no remaining old format records and he must be provided with sufficient information to reach this conclusion. In addition to other statistical information on the distribution of data records on the data base it is therefore proposed that the number of occurrences of each version of each (or selected) record defined on the schema can be displayed by specifying

DISPLAY COUNTERS (OF Record-Name-1)

Conditional Execution of Tasks and Concurrent Tasks

The remainder of this chapter gives details of the individual restructuring tasks. For any particular restructuring to be performed on a data base, however, it is likely that a number of these tasks will be required for different data items, records etc. The syntax of the restructuring language must therefore be such that tasks are completed in a predefined sequence. Normally a task would not commence until its predecessor was complete but in some situations (detailed under the individual tasks) it will be convenient for the restructuring mechanism to handle tasks relating to the same record type(s) at the same time. The restructuring language must therefore permit the Data Base Administrator to indicate which tasks he wishes to have executed concurrently. It may also be the case that certain tasks should only be executed if a previously executed task has been successful. The language must also allow the Data Base Administrator to specify such conditions.

Techniques for specification of concurrent and conditional execution of tasks are not described in detail here since they are common in other classes of operating system software (e.g. Job Control Language).

Note that there is no suggestion that more than one closed restructuring should be permitted to take place at the same time (although certain records may not have been restructured despite an open dynamic restructuring on them in the past) since it is considered that the Data Base Administrator should always be in control of this operation and he can therefore schedule tasks as he sees fit.

Task 1 - Addition of a new data item to an existing record

Data Base Administrators will often require to add a new data item to an existing record. It may be that the physical entity to which the record relates has changed in some way and that the new data item is to apply to some new property of the entity. It may simply be that the data item was omitted when the record was created.

Program Categorisation

Transparent Programs - Programs which do not require the new data item and therefore do not include it in their sub-schema definition of the record if they have one.

Opaque Programs - Programs which wish to take advantage of the new data item and include it in their processing.

Concurrent Tasks

More than one data item may be added to the same record concurrently.

Syntax

Two elements are required when defining the task:-

a. Details of the new data item as described in the Codasyl Proposals.

b. An indication of where the data item is to appear within storage records and in the logical schema record structure.

The proposed syntax is as follows:-

```
ADD DATA ITEM(BEFORE) Data-Item-1 in Schema-Record-Name-1
             (AFTER )

level number Data-Item-2 etc. (as in CODASYL Proposals)

⎧STORAGE ⎧⎧BEFORE⎫     Date-Item-3 in⎫Storage Record 1⎫
⎨        ⎨⎩AFTER ⎭,                  ⎬                ⎬...
⎩        ⎩IN NEW RECORD              ⎭                ⎭
```

Without attempting to give a rigorous definition of the syntax we can see that the new data item (Data-Item-2) is placed logically before or after an existing data item (Data-Item-1) in the schema record (Schema-Record-Name-1). It will be stored before or after nominated data items (Data-Item-3) in existing Storage Records (Storage-Record-1) or will be the only data item in a new Storage Record. (Repetition of the last clause is necessary to cater for records with alternative storage structures).

The definition of the data item itself must cover all the Schema and Storage Schema entries as described in the Codasyl Proposals. Note that source and result data items with storage allocated may not be added directly by using this task. They must be added without the storage clause and this clause added later (possibly as a subsequent step in the same restructuring) using Task 4.

Population of the Data Base

An optional clause may be added to the basic syntax which will allow the data item to be populated as it is created. The clause is a COBOL 'compute' statement which may reference other data items in the same record as well as literal constants. If this clause is not included the new data item will be allocated null values for each occurrence.

An Open Dynamic Restructuring Strategy

If an Open Dynamic Strategy is adopted for this task no data base records are modified at the time of the restructuring. Only the object schema entry for the record containing the new data item is amended to hold an additional sub-list giving details of this new (version n) format of record. The list for the record may already contain details of other previous versions and these must remain.

The alteration to record occurrences will therefore take place from the time of the restructuring when record occurrences are added to the data base or existing occurrences are modified by application programs. All such record occurrences will be written to the data base in the version n format. Where a new occurrence is to be written, the values for the data items will have been supplied in the normal way via the application program's user work area. If the new data item is not included in the sub-schema for such a program the data item on the data base record will be allocated null values. Where an existing record occurrence is being modified and the new data item has not been included in the sub-schema the population algorithm will be used to evaluate it using the data items on the original version of the record.

The data base management system must also handle requests by application programs to retrieve record occurrences from the data base. Individual occurrences may be at any one of a number of versions but since the version number is held with the record the DBMS can establish which object schema sub-list to access to determine the format of the record. If the program sub-schema references the new data item and the record version retrieved is n (or greater) the DBMS can perform the normal process of transferring the data item from data base record to User Work Area.

If the retrieved record version is less than n the value of the data item must be calculated using the Population Algorithm. It may be necessary to also evaluate other data items since data items quoted in the algorithm may themselves have been added in some previous restructuring.

When a sub-schema is compiled it is likely that an implementation will ensure that all data items defined in the sub-schema also exist in the schema. Opaque programs referring to a new data item cannot therefore be scheduled (in their revised form) until after the restructuring since their sub-schemas could not compile until after the restructuring had altered the schema. From that point onwards, however, the Data Base Administrator can choose an appropriate point for their introduction (for example it will often be convenient to synchronise the introduction of enhancements to the data capture logic to allow the new data item to be given values). There is some justification for implementations to allow data items to be defined in a sub-schema which do not exist in the schema (see note in Task 2). In this case any reference to such data items would result in null values being returned and any attempt to give such data items values would not be effective. On this basis, however, the Data Base Administrator would be free to schedule opaque programs <u>before</u> restructuring took place if he so desired.

<u>A Closed Dynamic Restructuring Strategy</u>

A Closed Dynamic strategy cannot be considered as an instantaneous event. The restructuring mechanism must access all record occurrences in turn by progressing along the Preferred Direction until all have been processed. The Data Base is in a Transitional State while this takes place. When it has read each record occurrence the restructuring mechanism will upgrade it from version m to version n as described above for an open strategy. The new data item will be included by allocating null values or executing the Population Algorithm. Finally, the record will be written as version n to the data base and, if there is not to be a period of parallel running, the version m record occurrence will be deleted.

Transparent programs can operate on the data base before, during and after the restructuring since any retrieved record can be presented to the program using the object schema sub-list containing the definition of that version of record. Any records written to the data base during and after the restructuring will be at version n (if the restructuring mechanism subsequently tries to restructure such a record it can establish that it does not have to do so by referencing the version number). Once again, in a parallel running situation version m records will also be written where m is the previous version of the record.

New versions of Opaque Programs will be introduced by the data base administrator at some point in time either during or after the restructuring based on some external factor such as the scheduling of amended data capture procedures. In either case the DBMS can handle Data Manipulation Langauge Commands as follows:-

Record occurrences written to the data base will be version n as above. If the new data item appears in the sub-schema of the program writing the record it will be allocated the value given to it by the program. Otherwise if the record is being written by a program which does not include the new data item in its sub-schema the Data Base Management System will firstly determine where the record is to be stored relative to the position in the Preferred Direction currently being processed by the Restructuring Mechanism. If the record is to be stored before this position and it is a modified version of an existing record the data item will retain its value from that record (which must have been version n). If the record is a new record being stored (once again before the Current Restructuring Mechanism position) the Population Algorithm will be used. After the current position both existing and new records will have the data item evaluated using the Population Algorithm. After the Transitional Period during Parallel Running the data item will always be stored as null values since this is how the Data Base Management System will eventually operate.

Records retrieved from the data base may be at version m if the restructuring mechanism has not yet reached that point and the DBMS must therefore execute the Population Algorithm to provide a value for the new data item before supplying this to the application program's User Work Area. If the record retrieved is at version n the value can be moved directly to the User Work Area.

Task 2 - Deletion of a Redundant Data Item from an existing record

Function

It is likely that certain data items will become redundant during the life of a data base. For example, the entity to which the data item relates may have changed in such a way that the data item is no longer relevant or even meaningful. This situation must be distinguished from that in which a data item is temporarily not accessed by application programs.

More specifically a data item is redundant if it is not referenced by any sub-schema or from elsewhere within the schema (other than in its own data item entry) and is to remain so.

The function of the Restructuring Task is two-fold:-

a. to ensure that the data item is redundant.
b. to avoid allocating space for the data item in record occurrences.

Syntax

The proposed syntax is as follows:-

```
DELETE data-item-1 FROM Schema-Record-Name-1
```

Task Validation

The Restructuring Mechanism must firstly ensure that the task is valid.

The deletion of the data item must not alter the self-consistency of the data base. The lists for all records in the Object Schema must be examined to ensure that the data item in question is not referenced in a SOURCE or RESULT clause in some other record. The list for the record containing the data item must also be examined to ensure that it is not a key data item or does not appear in a Set Occurrence Selection Clause for any set for which it is a member.

Further, the deletion of the data item must not lead to inconsistency between object schema and sub-schemas. That is, the data item should not appear in the corresponding record description of any sub-schema. As we have seen in Task 1, consistency between schema and sub-schema will normally be checked as each new sub-schema is added to the sub-schema library and it should be feasible to access the library for each sub-schema referencing the record in question to re-execute the check against the revised schema. This check is not necessary for additive tasks such as Task 1 because the inherent transparency of the schema to sub-schema mapping ensures that the sub-schemas remain consistent with the schema in these cases. If an invalid task has been specified the Restructuring Mechanism will not alter the data base in any way.

It is, however, feasible that implementors will not insist that data items in the sub-schema exist in the schema - perhaps they will generate a warning message. This situation could be advantageous in this task since application programmers would be likely to utilise sub-schemas which had already been defined (e.g. those referencing all data items in a record) for convenience.

Program Categorisation

Transparent	- All Programs (although in some cases it might be necessary to recompile sub-schemas and rebind programs to them).
Opaque	- None. As we have seen the task should only be executed if no sub-schema (and therefore no program) references the data item to be deleted.

Concurrent Tasks

A Data Item can be deleted from a record concurrently with other Data Items being added to the same record although deleted data items may not be referenced in Population Algorithms of subsequent data item additions. More than one data item can be deleted from the same record concurrently.

Implementation Strategies

The strategies for the deletion of a data item are similar to those for the addition of a new data item but there are fewer inherent problems since there are no Opaque Programs to cater for.

There is a great deal to recommend a Closed Dynamic Strategy in this case since it allows the DBMS to take immediate advantage of the free space. Once again, the data base will be in a Transitional State with different versions of records on the data base but the DML execution routines can use the version number on any retrieved record to determine whether the record is in the old or new format (as we have seen the data item itself cannot be required by any program).

Re-use of Data Item Names

If successive Open Dynamic Restructurings are carried out on a record such that a data item is deleted and a new data item with the same name is later added, there is a potential ambiguity in that some record occurrences will hold the old version of the data item and others will hold the new depending on how they have been accessed by application programs. In particular, if an application program accesses a storage record occurrence it must be provided with null values rather than the redundant value.

The Technique adopted might simply be to prohibit the re-use of data item names and there is therefore no primitive task to allow a data item to be renamed.

Task 3 - Amendment to the Format of a Data Item

Function

When the Data Base Administrator originally defines a data base he must decide on the most appropriate format to hold each data item. This decision cannot be taken lightly and must take into consideration the range of values the data item might take at that time and in the future. The Data Base Administrator must also take into account whether the predominant likely usage of the data item will be for arithmetic or display.

Circumstances might be such that the original decisions on certain data items eventually prove inappropriate and the format of these data items must be altered to reflect the new range of values or revised usage.

The Codasyl Proposals are structured to cater for variations in data item format as held on the data base and as processed by application programs. A conversion between data base format and program format (as defined in the Sub-Schema) is carried out by the Data Manipulation Language Execution Routines when necessary.

Syntax

The proposed syntax of this task is as follows:-

```
AMEND FORMAT OF Data-Item

TO ( PICTURE  Picture-Clause - 1 )
   (                             )
   ( TYPE     Type-Clause - 1    )
```

The format of the Picture and Type Clauses are given in the Codasyl Proposals.

<u>Expansion or Contraction of Data Item Format</u>

It is possible that an alteration to the format of a data item will result in the revised format being suitable for holding all existing data item occurrences (e.g. where a data item originally defined as a 6-digit number is amended to become a 7-digit number). We shall term this type of amendment as an <u>expansion</u> of the data item's format.

On the other hand, certain changes in format might mean that some existing occurrences cannot be held under the new format (e.g. where a data item originally defined as a 7-digit number is amended to become a 6-digit number any occurrences with a value greater than 999999 could not be held). This type of amendment we shall term as a <u>contraction</u> of format.

<u>Program Categorisation</u>

<u>Transparent Programs</u>

(a) Programs which do not refer explicitly to the data item.

(b) Existing Programs which read (but do not write) a record where the data item is being contracted in format.

(c) Programs which write (but do not read) a record where the data item is being expanded in format.

<u>Opaque Programs</u>

(a) Programs which write a record where the data item is being contracted in format.

(b) Programs which read a record where the data item is being expanded in format.

Transparent programs can operate during and after the restructuring since the type of format change is compatible with the type of data item transparency incorporated into the sub-schema to schema mapping. For example, if a program reads a 7-digit numeric data item with a sub-schema format of PICTURE 9(8) this is equally transparent if the data item on the data base record is contracted to 6-digits. It is significant that the sub-schema format is not compatible if the program attempts to write a record to the data base in either case (i.e. a value of over 9999999 or 999999 could have been given to the data item by the program but such a value could not be written to the data base record).

Concurrent Tasks

The format of a Data Item can be amended while other data items are being deleted from or added to the same record.

The format of more than one data item in the same record may be amended concurrently.

An Open Dynamic Implementation Strategy

In an Open Dynamic Strategy all existing occurrences will remain at their present version but when records are added or replaced on the data base by application programs they will be written as version n with the data item in the new format. When a record is retrieved from the data base, the DBMS must use its version number together with the object schema sub-list for that version of the record to perform a suitable schema to sub-schema mapping.

All records written to the data base will be at version n. In a contraction of data item format this may mean that the DBMS discovers incompatible record occurrences at this point (e.g. if a data item is being reduced from 7 to 6 digits any occurrence with a value more than 999999 cannot be written). Such eventualities can onl- be reported to the Data Base Administrator as and when they take place rather than at the time of the restructuring if this type of strategy is used. Furthermore, the Data Base Management System must write something to the data base. Null Values would seem to be the best choice although some implementors might consider truncation appropriate.

This strategy would only be applicable to certain opaque programs operating on the data item being amended:

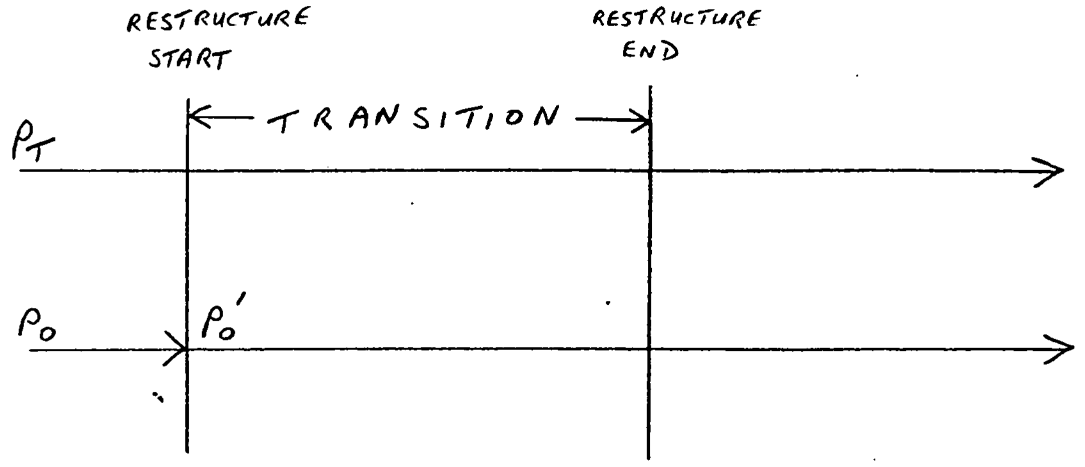
(a) Programs which read (but do not write) the record where the data item is being expanded in format.

(b) Programs which write (but do not read) the record where the data item is being contracted in format.

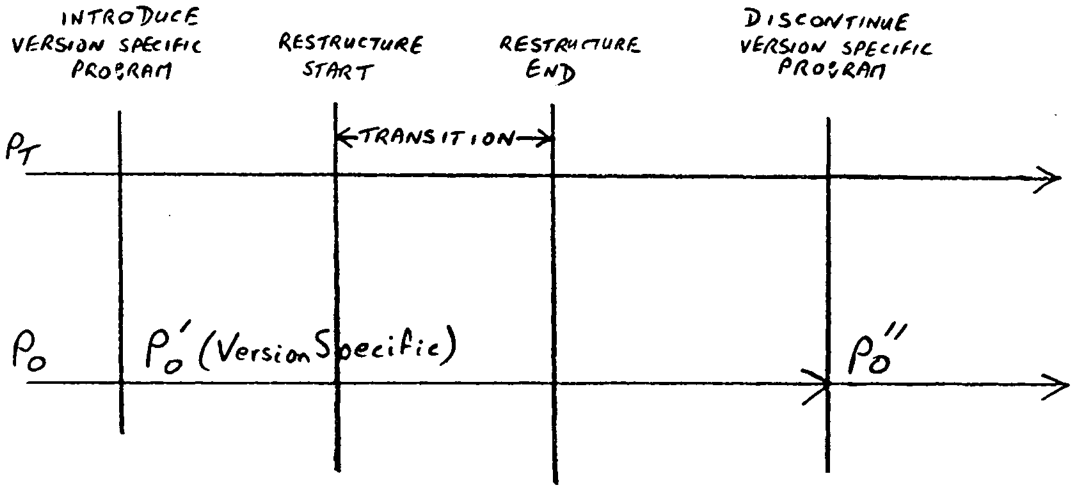
In these cases the revised version of the program must be scheduled before the restructuring takes place since the sub-schema to schema mapping will allow them to operate successfully during that time. For example, if a data item is expanded from 6 digits to 7 digits any programs which read (but do not write) the record with a sub-schema definition of PICTURE 9(6) must be altered to refer to it at least as PICTURE 9(7).

A Closed Dynamic Implementation Strategy

For an expansion of data item format a Closed Dynamic Strategy is straightforward. The Restructuring Mechanism can proceed through the data base in the Preferred Direction and transform each occurrence to the new format (this always being possible for an expansion). As for task 1 the DBMS can retrieve version m and n records from the data base and transfer the data to the program User Work Area as version n. This procedure implies that the new version of opaque programs must be scheduled to coincide with the start of the Transitional State.



Alternatively, a Version-Specific version of the program can be used to cover the Transitional State. This could operate indefinitely but it would probably be replaced by the revised version at some convenient time after the restructuring is complete.


INTRODUCE VERSION SPECIFIC PROGRAM
RESTRUCTURE START
RESTRUCTURE END
DISCONTINUE VERSION SPECIFIC PROGRAM
TRANSITION
P_T
P_O
P_O' (Version Specific)
P_O''

For a contraction in format the Restructuring Mechanism might encounter certain data items which are incompatible with the new format as described above for the open strategy. In this case the Data Base Administrator can opt to "Roll Back" the Data Base to its original format if such an occurrence is detected.

Even if he opted to have all such occurrences reported to him this would be more satisfactory than the open strategy where discrepancies were reported at arbitrary times in the future determined by the frequency of record modification by application programs. In other respects this type of amendment can be carried out in the same way as an expansion of format.

Data Items Referenced Elsewhere in The Schema

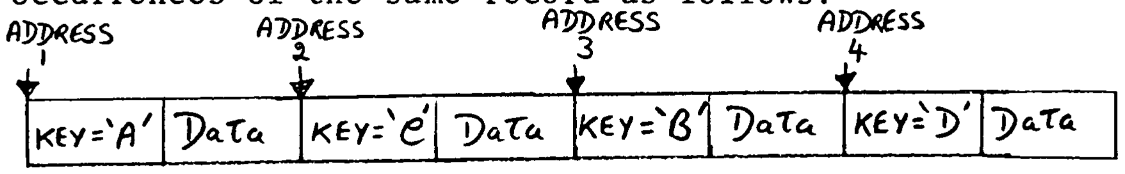
If the data item whose format is being amended is referenced elsewhere in the Schema the Restructuring Mechanism must take this into account.

The data item may be referenced in the SOURCE clause of another data item. If the STORAGE IS NOT REQUIRED clause has been specified for this data item the DBMS can cater for the change of format since it must access the original data item any time the source data item is required. We have already seen how this can be accomplished. If the STORAGE IS NOT REQUIRED clause is not included the Restructuring Mechanism must amend the source data item as well as the original data item. In a Closed Dynamic Strategy the source record can be amended immediately after the original record and its version too amended to n. The source records are not necessarily amended in their "Preferred Direction" but they should nevertheless all be altered when the restructuring is complete since there is a one-to-one relationship between source and original data items.

In an Open Dynamic Strategy a similar situation applies and the DBMS can alter the source record whenever the original is modified. Suppose, however, that the source record has to be modified for some other reason. The DBMS in this case would be required to also modify the corresponding original record.

The Data Item being amended may be referenced in the KEY clause for its record and in this case the format change must be reflected in the corresponding index in addition to the record itself. When a record alters from version m to version n in either Open or Closed Dynamic Strategy with one or more data items involved in the change being key data items the DBMS must set up a new index for the record. This can be done as the restructuring takes place by removing entries from the existing index to the new index as the corresponding records are altered. If a Parallel Running State is required old index entries will not be removed.

For example, suppose the data base consists of four occurrences of the same record as follows:



The original index would be:

A	1
B	3
C	2
D	4

Now if the "Preferred Direction" for the record is address sequence and Closed Dynamic Restructuring has amended the records at addresses 1 and 2 there would be two indices as follows:

VERSION m
INDEX

B	3
D	4

VERSION n
INDEX

A	1
C	2

That is the original index has lost two entries (for the amended records at addresses 1 and 2) and the new index has gained entries for these records.

In the Transitional State of a Closed Dynamic Restructuring and after an Open Dynamic Restructuring the DBMS may have to access both indices to satisfy a request from an application program to FIND a record based on the value supplied for a key data item. The sub-schema to schema mapping can be used to format the index key appropriately in each case. If, however, the DBMS has to alter an index entry because a record occurrence key data item has been MODIFIED it can access the correct index immediately by reference to the version number held in the record occurrence itself.

Task 4 - Allocating Storage to a Source or Result Data Item

Certain Data Items may be derivable from other data items either on the same record occurrence or on other occurrences of records related by a set occurrence.

The addition of such data items to a record is straightforward if no storage is to be reserved for them within record occurrences since this only involves an amendment to the schema entry to indicate how the data item is to be evaluated if it is required by an application program. If storage is required, however, the restructuring mechanism must operate as in the addition or deletion of a new data item but the population must be carried out using the Source or Result Algorithm.

Syntax

```
                 (INTRODUCED  )
STORAGE REQUIRED (DISCONTINUED)
```

Concurrent Tasks

This task may be executed concurrently with any of the previous tasks operating on data items in the same record.

Program Categorisation

Transparent - All programs including those which refer to the source or result data item.

Opaque - None.

Population of the Data Base

The following method of population of the data item can be carried out when a version n record is written to the data base in either an Open or Closed Dynamic Strategy (see Task 1).

For a SOURCE data item the corresponding data item in the owner record occurrence must be accessed and this value allocated to the new data item.

For a RESULT data item the value can be derived by executing the appropriate procedure using data items in the record itself and in corresponding member record occurrences.

Task 5 - Amendment to the Value Range of a Data Item

The Codasyl Proposals allow many of the functions of data item validation to be performed by the DBMS rather than the application programs themselves. The range of values which a data item may take is likely to alter from time to time either in a predictable fashion (e.g. the acceptable range for an employee number data item might expand or contract to reflect the size of a company's workforce) or because of unforeseen changes of the data item's usage.

In the schema the value range for a data item may be specified in one of two ways - as a number of discrete ranges of values or as a procedure to be executed to give a 'VALID' or 'INVALID' result. This may, therefore, be amended by adding, deleting or amending ranges or by introducing or discontinuing a data base procedure.

Syntax

There are a number of alternative formats to this task all of which can be carried out concurrently if required.

Format (a) - (Only one such entry per task)

```
                (INTRODUCED  )
CHECK NONNULL   (DISCONTINUED)
```

Format (b) - (Only one such entry per task or two entries with the first indicating DISCONTINUED and the second INTRODUCED)

```
CHECK procedure-name-1 (INTRODUCED  )
                       (DISCONTINUED)
```

Format (c)

```
CHECK VALUE RANGE INTRODUCED AS
[NOT] literal-1 THRU literal-2
```

Format (d)

```
CHECK VALUE RANGE DISCONTINUED WAS
[NOT] literal-1 THRU literal-2
```

Format (e)

```
CHECK VALUE RANGE AMENDED FROM
[NOT] literal-1 THRU literal-2
TO [NOT] literal-3 THRU literal-4
```

For simplicity we have assumed that data base procedures may not be altered without a change in procedure name.

Classification of Amendment

The net effect of the set of entries to alter the acceptable value range of a data item may either expand or contract that range (where some discrete ranges have been expanded and others contracted we shall consider the net effect as a contraction).

The constraints imposed by the ranges and procedures may make the validation procedures (performed by the DBMS when it modifies or stores records on the data base) either more or less rigorous.

Program Categorisation

Transparent - All Programs whether they reference the data item in question or not.

Opaque - None.

Implementation Strategies

We shall consider separately the strategies required where the data item validation procedures are to become more rigorous and less rigorous.

For an amendment to make the procedures less rigorous the Restructuring Mechanism need only alter the object schema to reflect the new criteria. The data base itself need not change and the record version number will not be altered. All existing data item occurrences will still be valid under the new criterion.

For an amendment to make the procedures more rigorous there is the possibility that certain existing occurrences of the data item in question will no longer be valid under the new criterion. There are two strategies which the Data Base Administrator can adopt to detect such occurrences as follows:

An Open Dynamic Technique would result in the validation being carried out when a record is retrieved from the data base by the DBMS in response to a request by an application program - this would be in addition to the normal procedure of carrying out the validation when the record is modified or stored. The validation need only be performed once on each record occurrence. This could be ensured if each successfully validated record was modified to version n as previously described for other tasks. Validation would then only be performed on reading version m records. On the other hand, if a data item occurrence did not satisfy the validation criterion the situation would be reported to the Data Base Administrator and the data base record would remain at version m.

A Closed Dynamic Technique would allow the Data Base Administrator to obtain a report of all occurrences with an unacceptable value at the time of the restructuring. The Restructuring could take appropriate action if any such occurrences were found. Possible actions are:-

(a) To leave the occurrence unchanged and not implement the revised value range.

(b) To set the occurrence to null values and continue to examine other occurrences.

In this case the DBMS would use the new (more rigorous) criterion for data item validation as required for record occurrences being written by application programs during the Transitional State. This means that existing record version numbers need not be altered.

Concurrent Tasks

All of the previous tasks may be executed concurrently with the amendment to the value range of a data item in the same record.

The value range for more than one data item in the same record can be altered concurrently.

Task 6 - Intra Record Structure Amendment

Introduction

In the Codasyl Proposals the structure of a record in the schema and storage schema is defined in much the same way as Cobol defines records on traditional files.

The record is divided into elements of three types:-

Data Items
Data Aggregates
Vectors

Each element is allocated a level number and the nesting of elements within each other allows the most general format of record to be defined.

Function

The principal operation required for an amendment to record structure is to allow the number of repetitions of a data item to be altered. Repetitions of a data item may be contiguous (a Vector) or separated by corresponding repetitions of other data items (part of a data aggregate). It is evident that this task will be necessary since assumptions made on the number of repetitions of data items when the data base was created will often have to be revised in the light of experience.

Syntax

A variety of formats are proposed for this task to allow the variation in the number of repetitions of the data item to manifest itself in the number of occurrences of a data aggregate or vector.

In particular, since an elementary data item is a vector with a single occurrence a data item may be elevated to become a vector or a vector may be reduced to a data item.

Format (a)

```
                         (Vector-1          )
AMEND OCCURS VALUE OF    (Data Aggregate-1) IN Record-Name-1
                         (Data Item-1       )

FROM (Data Identifier-1) TO (Data Identifier-2)
     (Integer-1         )    (Integer-2         )
```

Format (b)

```
INTRODUCE Data-Aggregate-2 OCCURS n TIMES
TO COVER Data-Item-2 TO Data-Item-3
IN Record-Name-2
```

(The data aggregate will assume the lowest level number of the covered data items and all covered data item levels will be increased by 1)

Format (c)

```
                         (BEFORE)
MOVE Data-Item-4 to      (AFTER )  Data-Item-5
IN Record-Name-3
```

(where Data-Item-5 is part of a data aggregate this will have the effect of adding Data-Item-4 to that data-aggregate and amending its level accordingly).

There is little point in moving a data item to a new position in a record while retaining the same number of repetitions since the schema to sub-schema mapping ensures that data item position within data base records is transparent to application programs.

Program Categorisation

Transparent	- All programs which do not reference the record whose structure is being amended. Programs which reference the record but do not reference data items whose number of repetitions are being altered.
Opaque	- Programs which reference data items whose number of repetitions are being altered.

Concurrent Tasks

The structure of a record may be altered concurrently with any of the preceding tasks in any of its data items whether their number of repetitions is being amended or not.

Population of the Data Base

Where the number of repetitions of a data item is being amended by altering the record structure, the restructuring mechanism must determine which existing repetitions must be removed from the record and which new repetitions (possibly with allocated values) must be added to the record.

Often if additional repetitions are being created they will become the 'last' in the record and will be allocated null values or some other suitable value. But if the new repetitions are to be inserted elsewhere, an additional clause will be required to specify their position as well as their value.

Equally, if repetitions are to be removed, these too will often be the 'last' on the record but other repetitions could be removed using an additional clause.

If the number of repetitions varies from one record occurrence to another (depending on the value of another data item) it may be necessary to quote both types of clause to cater for all possible changes in format.

An Open Dynamic Implementation Strategy

As for previous tasks an Open Dynamic Strategy involves an amendment to the object schema to incorporate an additional list giving the new structure of the record under version n.

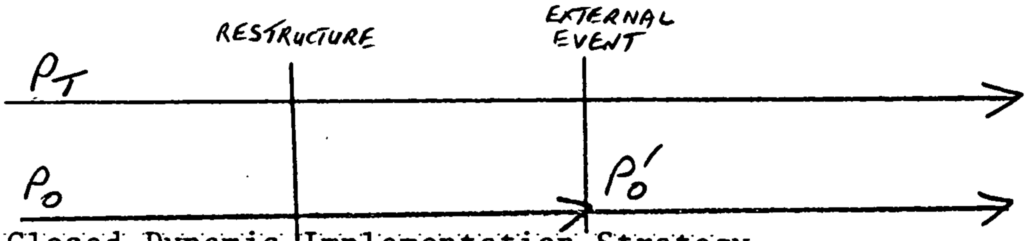
Any new records stored or modified on the data base will be written in the new structure while records read from the data base to satisfy a GET from an application program may be either version m or n. The DBMS must continue to allow application programs to "see" the previous record format even if the record retrieved is version n. It can do this by having a version number implied in each of its DML commands based on the version of data base operative when the program was compiled.

For example, suppose the data base holds a record with a vector of 5 elements. A restructuring could be carried out to add a 6th entry (with null values initially) before the existing 5 to produce a 6-element vector.

If an old version application program were to STORE or MODIFY an occurrence of the record it will be written with all 6 elements even though the program only "sees" it with 5. If a record with 5 elements is retrieved for a GET by an application program the DBMS can provide these values directly to the program but if a record with 6 elements is retrieved the program must take the restructuring into account and only provide the program with those 5 that it requires.

Eventually, however, application programs will wish to take advantage of the new record structure. In these circumstances, the DBMS must therefore be aware of which structure is required by a particular program and this can be done by associating a version number with the program (or sub-schema). Any program with a version number of n or greater will be provided with the new structure while programs with versions lower than n will be provided with the old structure.

In our example above, a version n program could be written referencing the vector with its full 6 elements. This could access existing version n occurrences and be provided with the correct values. If it were to access occurrences with a lower version the DBMS would have to transfer the five elements from the record to the program as the last 5 elements in its vector with the first element being set to null values. Records written by the version n program will always have the full 6 elements in the vector.

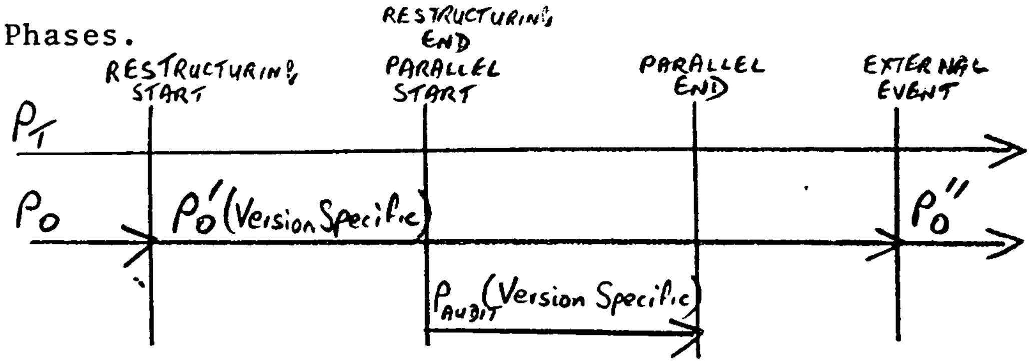


A Closed Dynamic Implementation Strategy

In a Closed Dynamic Strategy all existing record occurrences will have their structure amended during the restructuring. Any programs referencing the data items having their repetitions altered must be modified at some point during this process since the original versions must operate before the restructuring and the final version afterwards. If possible, the Data Base Administrator would arrange that such programs were not run during the Transitional Period.

Suppose, however, that there is a requirement to execute a program during the Transitional Period. The approach adopted above for Open Restructurings could be taken to allow the program to "see" the old structure or the new structure for the duration of the restructuring. Even this might not be sufficient and it may be that the program would have to be modified twice - once to become Version Specific so that it could take appropriate action depending on the version number of each record retrieved and once more to adopt its final form. It should be noted that the Version Specific program would continue to operate successfully after the restructuring although it would contain redundant coding which it would be advisable to remove eventually.

Another consideration is where the restructuring is to involve a period of parallel running once all records have been modified. In this case, the revised records supplement rather than replace the existing records. The old record structure continues to exist during the restructuring and the old versions of the programs can therefore continue to operate on these record versions during the Transition and Parallel Running Phases.



In the diagram P_T is a Transparent Program which can operate before, during and after the restructuring since it does not reference the data item whose number of repetitions is being altered.

P_O is an Opaque Program which is to take advantage of the change in the number of repetitions. A new version of the program (P_O') is scheduled for introduction to coincide with the start of the transitional period. If it finds a record which has had the number of repetitions altered it can recognise it by its version number and process its data accordingly. If it finds a record in the original form it can proceed as before. The program is therefore now version specific. Once the Transitional Phase is complete all records will be read and written in the new format (during the Parallel Run they will also be written in the old format but this is irrelevant to this program). Some time later, therefore, the logic to process records in the old format can be removed and the program will be no longer Version Specific (i.e. it is version P_O'').

P_{AUDIT} is an ad-hoc program which ensures that the Restructuring has been performed successfully by examining both old and new formats of the same record occurrence and checking their compatibility. Once this program has been run successfully, the Data Base Administrator can decide to terminate the period of parallel running.

Semantically Equivalent Programs

As we have seen, opaque programs will be required to alter in their data base access only in so far as they use GET and MODIFY to transfer data between User Work Area and data base record. In general these changes must be co-ordinated with other changes to the logic of the programs to determine how the amended number of occurrences of the data item in the record are to be handled.

One specific instance, however, leads itself to a process of automatic conversion of the DML commands issued by a program to a corresponding set of commands which are semantically equivalent under the new structure. This equivalence takes advantage of the fact that a data item within a data aggregate or vector which has only one repetition is equivalent to a simple data item once it has been transferred to the User Work Area. Thus GET (VALUE) will be equivalent to GET (VALUE (1)) if the data item VALUE is being embedded in a vector with one repetition. Although the frequency of this type of structural change may well be limited (it could be said that it does not actually alter the structure), the existence of a mechanism for automatic program conversion would encourage Data Base Administrators to approach alterations to the number of repetitions of data items in a phased manner with the intermediate state being that data items within vectors or data aggregates had only one repetition.

Task 7 - Migration of a Data Item Between Records

Function

When the Data Base Administrator originally defines a Data Base, he will form the data base records by grouping data items which he considers to relate to the same entities from the real world. For certain data items it might be difficult to decide to which entity (and, therefore, to which record) they relate. Later, it may well become apparent that certain data items would more properly be part of other records than those in which they were originally included.

If it is to allow data items to be moved from one record to another the restructuring mechanism must have some criterion to use to determine which occurrence of the destination record is to receive each occurrence of the data item from the source record. It is proposed that two means of establishing this criterion will be available:-

(a) The source and destination records are owner and member or member and owner of a set already defined in the data base.

(b) The source and destination records are related by equality of value of certain specified data items. This relationship may be considered as a special case of the set relationship where the record association is implicit in the data item values rather than explicitly defined by means of set pointers or indices. In the description which follows the set terminology is therefore used except where special consideration must be given to this type of implicit set.

Both types of record relationship allow for the possibility of one-to-many and many-to-one relationships between record occurrences although a simple one-to-one relationship is not excluded. In both cases, both source and destination record occurrences must actually exist to which the data item will be removed and added respectively.

In normal circumstances, the data item will be removed from each occurrence of the source record and added to each occurrence of the destination record. If necessary, however, the data item may be retained in the source record.

<u>Syntax</u>

The proposed syntax for this task is as follows:-

```
MOVE Data-Item - 1 FROM Record-Name - 1
                   TO   Record-Name - 2
                  (ALONG Set-Name - 1                 )
                  (USING EQUALITY OF (data-item 1 ...))
{DISTRIBUTION BY REPETITION              }
{DISTRIBUTION BY EQUALITY                }
{DISTRIBUTION BY Procedure-Name - 1      }
[RETAIN SOURCE]
```

The distribution clause is necessary to define how the data item is to be populated in the destination record.

For Distribution by Repetition the data item can be a Vector (or part of a repeating group) in the source record and move (as an elementary data item) to the destination record with each repetition of the data item assuming a value in one of the destination member records in turn. The number of repetitions of the data item in the source record must therefore exactly equal the number of occurrences of the destination record or occurrences can be created to accommodate them. Similarly, Distribution by Repetition may take place from member to owner where an elementary data item becomes a vector in the owner record.

For Distribution by Equality the value of the data item in the Owner record (if this is the source of the migration) is allocated to all occurrences of the data item in the destination (member) record occurrences. For migration from member to owner all member records in the same set occurrence must have the same value for the migrating data item in which case this value is allocated in the destination (owner) record.

For Distribution by Procedure, the specified procedure is executed to give value(s) for the data item in the destination record(s) based on value(s) in the source record(s).

The Clause 'RETAIN SOURCE' is optional. If included the data item is retained in each occurrence of the source record in addition to being copied to the destination record.

Program Categorisation

Transparent	- Programs which do not reference the data item being moved.
Opaque	- Programs which reference the data item being moved.

Concurrent Tasks

This task can be executed concurrently with any of the previous tasks where they are operating on data items in the source record.

More than one data item may be moved concurrently from the Source to the Destination Record.

A Closed Dynamic Implementation Strategy

In a closed dynamic strategy the restructuring mechanism must move a data item from an owner record occurrence to one or more member record occurrences or from one or more member record occurrences to an owner record occurrence.

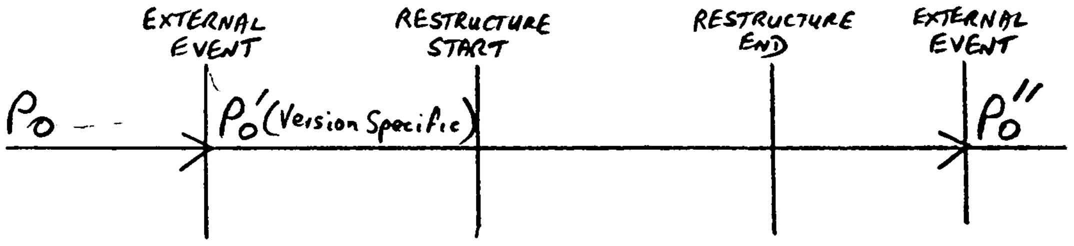
The restructuring mechanism therefore operates on all the record occurrences in a set occurrence to perform the necessary migration and access to all record occurrences in the set by application programs will therefore be precluded while this is done.

Each set occurrence is established by progressing through occurrences of the owner record in the preferred direction. Each such record will be at some version m $(<n)$. Using the appropriate set pointers all member record occurrences can then be retrieved. These will be at some other version p $(<n)$. Following reference to the Distribution Clause the restructuring mechanism can thenallocate values to the data item in the destination record(s) using values found in the source record(s). When this has been done all records can be written back to the data base in their new format at version n. Since the set relationship precludes one member record occurrence being related to more than one owner record and assuming membership of the set is mandatory (i.e. all member records must be related to some owner) when all owner records have been processed the restructuring is complete.

The use of version numbers allows the DBMS to process accesses to the source and destination records by Transparent Programs before, during and after the restructuring in the same way that it handles the addition and deletion of data items in Tasks 1 and 2.

As we have seen, there may be some delay in servicing accesses to records while the set occurrence in which they participate is being processed by the Restructuring Mechanism. Opaque Programs will see the data item as held in the source record before the restructuring and the destination record thereafter. There are various methods whereby opaque programs can operate during the transitional period:-

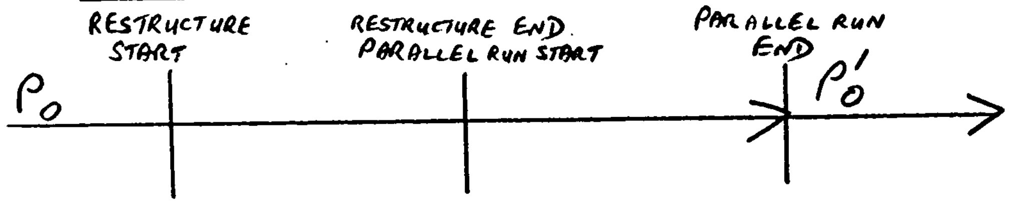
(a) <u>The Opaque Program can be made Version Specific</u>



In the knowledge that the restructuring is to take place the program can be amended to process records in different ways depending on their version numbers. If their version is less than n then the data item will appear in the source record but not in the destination. If their version is greater than or equal to n the data item will appear in the destination record but not in the source. Any new records written to the data base will be written at either version n or their original version depending on whether the corresponding set owner record would have been processed by the Restructuring Mechanism.

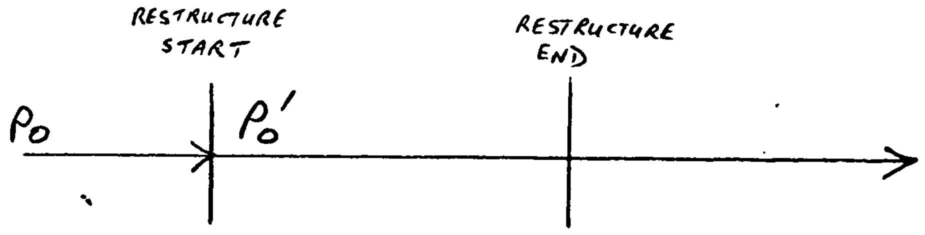
The introduction of the Version Specific Program need not coincide with the start of the restructuring since it will operate satisfactorily before this point where no version n records exist. Similarly, it need not be discontinued after the restructuring since it will operate satisfactorily when all records are at version n. Introduction is likely to be linked to some external event such as the introduction of new data capture procedures. Amendment to become non-Version Specific is likely to be influenced by feasibility of expending programming effort to make the program more efficient to run.

(b) <u>The Original Version of the Opaque Program can continue to run where a period of parallel running is required</u>



In this case occurrences of the source record containing the migrating data item will continue to exist up to the end of the parallel run and there is therefore no difficulty for the DBMS to satisfy requests from application programs to retrieve the data item from the source record. During and after the restructuring there may also exist versions of the source and destination records in the new format and the DBMS must handle any new or updated records by performing or reperforming the data item migration for these records.

(c) <u>The amended version of the Opaque Program can be introduced to coincide with the start of the restructuring</u>



During the restructuring, application programs may require to retrieve, update or insert records which have or have not been restructured. If the record has been restructured (or is to be added at a position prior to the current point of the restructuring in the preferred direction) there is no difficulty since the migrating data item will be in the destination record as expected by the program. If, however, the restructuring has not yet reached the point occupied or to be occupied by the record the DBMS must perform the migration on that set occurrence before servicing the request by the application program. This is similar to the technique described in Task 1 (Data Item Addition) where the restructuring mechanism may encounter already restructured records but it can recognise these as being at version n and progress onto the next occurrence.

As values for the destination data item are calculated by the restructuring mechanism it is possible that inconsistencies are detected between the Distribution Clause and the actual values on the data base record. For example, if the clause indicates distribution by repetition the number of occurrences of the member record may not correspond with the number of repetitions of the data item in the source record. Similarly, if the clause indicated distribution by equality and migration is from member to owner all values of the data item in the member records may not be equal. As for previous tasks the Data Base Administrator has the option of "rolling back" the restructuring or allocating some value to the migrating data item and reporting the situation.

An Open Dynamic Implementation Strategy

The object of an open strategy will be to migrate the data item only when this is necessary because of access to individual record occurrences by application programs. To accomplish this, it is essential that the set or equivalent data items used as the basis for the migration remain stable (i.e. are not themselves restructured) while occurrences of the old format of records remain on the data base.

The mechanism for handling each migration is similar to the type (c) Closed Strategy described above where a record to be retrieved, updated or added to the data base had not (yet) been restructured. To obtain this effect it is essential that any opaque programs are scheduled in their revised form to coincide with the start of the restructuring.

In this task the requirement to restructure a complete set occurrence as a unit may impose significant limitations on the use of an Open Dynamic Strategy. Accesses by application programs may have to wait for an appreciable time until the complete set has been processed. Considerations must be given to the feasibility of restructuring a set even if the migrating data item only has to be retrieved to service an application program request. Unlike other tasks it may be desirable to perform the migration in this case to avoid subsequent accesses to the complete set occurrence.

Once again an open strategy may impose difficulties on the Data Base Administrator in monitoring errors detected when a data item is migrating (e.g. a discrepancy in the actual number of occurrences of the destination record). Such errors can only be detected when application programs happen to access records and the DBMS therefore restructures them, and the timing of this will often be outwith the control of the Data Base Administrator.

Semantically Equivalent Programs

In general, opaque programs will alter significantly if a data item which they use migrates from one type of record to another. In particular if distribution is by procedure it is not likely that it will be possible to automatically convert such programs so that they can operate on the revised structure. Furthermore where the logic of a program permits complex paths with a variety of combinations of DML commands (i.e. where a program may branch to an instruction between DML commands) no simple logic path may reflect the new structure. Nevertheless in certain instances it may be possible to convert programs to semantically equivalent versions.

For example if distribution is by equality (or by repetition where the data aggregate has one repetition) programs with an un-interrupted flow of

```
     FIND OWNER
     FIND CORRESPONDING MEMBER OF MIGRATORY SET
or   FIND MEMBER
     FIND CORRESPONDING OWNER OF MIGRATORY SET.
```

may be modified automatically such that the GET for the migrating data item appears after both FINDS.

Task 8 - Adding a New Record to a Data Base

Function

One of the principal areas of expansion of a data base is where the organisation modelled in the data base identifies a new entity which must be represented by a corresponding data base record.

Program Categorisation

Transparent Programs	- Programs which do not require the new record and do not include it in their sub-schema.
Opaque Programs	- Programs which are to be amended to include the new record in their processing.

Concurrent Tasks

No other tasks may operate concurrently with this task since it is instantaneous.

Syntax

The proposed syntax for this task is as follows:-

ADD Record-name - 1

DEFINITION	- As for the Schema Entry in the Codasyl Proposals
STORAGE	- As for the Storage Schema Entry in the Codasyl Proposals.

Implementation

The task results in a list giving details of the new record being added to the Object Schema. At this time, no record occurrences will exist on the data base since these will be added by later tasks or by application programs.

Task 9 - Splitting Data Items from an Existing Record to form a New Record

Function

Where a number of related data items constitute a record it may become apparent that certain of these data items are related in some special way which could be more easily utilised if they were separate from the remaining data items. Thus, the original record will be split with each occurrence becoming two distinct records.

Note that this task is distinct from Task 7 (Data Item Migration) in that a new record occurrence is created when the data items are split where the requirements for a relationship by set or data item equality implies that record occurrences must exist to receive migrating data items.

Program Categorisation

Transparent Programs	- Programs which do not reference data items being split.
Opaque Programs	- Programs which reference the data items being split.

Syntax

The proposed syntax for this task is as follows:-

```
   SPLIT Data-Item-1 FROM Record-Name-1
                     To   Record-Name-2
or COPY Data-Item-1  FROM Record-Name-1
                     To   Record-Name-2
```

The difference between the clauses is that the second form results in the data item remaining in the original record as well as being included in the new record. This is particularly useful for data items which identify record occurrences and provides for relating the split records using a set.

Concurrent Tasks

This task can be executed concurrently with any of the previous tasks where they are operating on the data items of the original record.

More than one data item may be split or copied from the same original record to one or more new records.

A Closed Dynamic Implementation Strategy

By progressing through the original record occurrences in turn the restructuring mechanism can create new versions of the split records and write each to the data base at version n.

As for Task 7 (Migration of a data item between records) there are three possibilities for the scheduling of the new versions of opaque programs.

A Version Specific version of the program can be used to handle data base access during and immediately after the restructuring.

The new version can be scheduled for introduction to coincide with the end of a period of parallel running with the original version operating on the old record versions during the restructuring and parallel run.

The new version can be scheduled for introduction to coincide with the start of the restructuring. An attempt by an application program to access a record which has not yet been restructured will result in the DBMS performing the restructuring on that record before servicing the request.

An Open Dynamic Implementation Strategy

As for a Closed Strategy the techniques used to service accesses to the data items split from the main record can be handled by performing the restructuring if this has not already been done. In this case, this can be ascertained since the absence of a record occurrence implies that the restructuring has not taken place.

Semantically Equivalent Programs

The opportunity exists for the conversion of programs to semantically equivalent forms operating on the revised data base structure if the splitting of data items from a record to a new record is combined with the creation of a new set combining the two records (see Task 12). The one-to-one relationship between the records will imply that each set of occurrences has one owner and exactly one member. Any programs accessing the original record by the FIND command can have this DML callextended by addition of a further FIND MEMBER command for the new record. Similarly a STORE or MODIFY command can be replaced by two STORE commands or by a MODIFY, FIND MEMBER, MODIFY command sequence respectively.

Where data items are copied to the new record there should be no requirement to modify programs which do not STORE or MODIFY the data items. If STORE or MODIFY is used, however, the program must be altered to reference the copied data items in both types of record.

Task 10 - Deletion of a Redundant Record

Function

It may be that an entity on which data has previously been maintained is no longer relevant to an organisation's needs. The Data Base Administrator must therefore have the ability to delete all occurrences of such a record from the data base to free the storage space for useful records.

Before a record can be deleted the restructuring mechanism must ensure that it is indeed redundant. A redundant record is one which is not referenced in any sub-schema and is not referenced in the Object Schema other than in its own record definition. In particular, the record must not be the owner or member of any set (member records can be removed from sets using Task 15 and sets with this record as owner can be deleted using Task 13).

Program Categorisation

Transparent Programs - All programs

Opaque Programs - None

Concurrent Tasks

No tasks may be executed concurrently with this task.

Syntax

The proposed syntax for this task is as follows:
DELETE Record-Name - 1

Implementation

The restructuring mechanism has two functions to perform to execute this task:-

(a) To remove the list for the deleted record from the Object Schema.

(b) To delete all occurrences of the record from the data base.

The record occurrences can be deleted by progressing through them in the preferred direction and using the mechanism adopted by the DBMS for the deletion of individual records by application programs. One method of doing this is to maintain a "free space directory" and the storage occupied by the record occurrences can be added to this directory. Another technique is to mark records as due for deletion to be physically deleted later by a "garbage collection" utility.

Task 11 - Amendment to a Record Key

Function

A number of keys (a key being a collection of data items) may be defined for a record. For each key there is a corresponding index which allows an individual occurrence of the record to be addressed directly when provided with the required values of the data items in that key.

The use of the key is to some extent up to the host programming language in which the DML commands are to be embedded. It may be that a program can retrieve a record by providing values for any of its data items (if no key exists for these data items the appopriate record will be obtained by sequential searching). More likely, however, the program must quote the values of data items for one of the keys defined in the schema for the record.

Program Categorisation

Transparent Programs	- Programs which do not reference that key
Opaque Programs	- Programs which use the key being amended to address records on the data base. New versions of each program can only be scheduled when the restructuring is complete.

Concurrent Tasks

The task may be performed concurrently with any of the previously defined tasks operating on data items of the record to which the key relates where the restructuring is being done using a Closed Dynamic Strategy.

Syntax

The proposed syntax for this task is as follows:-

Format (a)

ADD NEW KEY TO Record-Name-1

Key-Name-1 (Key Clause as defined in Codasyl Proposals)

Format (b)

DELETE KEY Key-Name - 2 FROM Record-Name - 2

Format (c)

ADD $\left\{\begin{matrix}\text{ASCENDING}\\ \text{DESCENDING}\end{matrix}\right\}$ Data-Item - 1 TO KEY Key-Name - 3

IN Record-Name - 3 $\left\{\begin{matrix}\text{BEFORE}\\ \text{AFTER}\end{matrix}\right\}$ Data Item - 2

Format (d)

DELETE Data Item - 3 FROM KEY Key-Name - 4 IN Record-Name - 4

Format (e)

DUPLICATES IN KEY Key Name - 5 OF Record-Name - 5

$$\left\{\begin{matrix}\text{INTRODUCED AS } \left\{\begin{matrix}\text{FIRST}\\ \text{LAST}\end{matrix}\right\}\\ \text{DISCONTINUED}\end{matrix}\right\}$$

Implementation Strategy/

Implementation Strategy

The amendment to record keys does not require alteration to record occurrences on the data base and their version number, therefore, need not alter. The restructuring mechanism need only operate on the index corresponding to the key being altered.

Where a key is deleted the restructuring mechanism need only delete the corresponding index.

Where a key is being added or amended it will be necessary to set up a new index and this can be done as follows:-

All occurrences of the record in question can be processed in the preferred direction and the key data items extracted so that an index entry can be made. It will probably be more efficient to create the index by sorting the index entries after they have all been created rather than by creating the index directly by adding each entry as it is processed. In any event such an operation can continue without hindering application programs which reference the record since opaque programs which are to use the new key cannot be scheduled until it is complete. Some application programs will add new occurrences of the record and amend and delete existing occurrences. These will continue to result in any existing indices being updated. The DBMS need take no additional action if the restructuring mechanism has not yet reached this occurrence when progressing along the preferred direction. If the restructuring mechanism has passed the position occupied or to be occupied by the record occurrence the DBMS must take the necessary steps to add, amend or delete the entry from the index being set up.

If an index is being amended the last step in the restructuring will be to alter the Object Schema entry for the key to the record to provide the address of the new index rather than the old.

If a new or amended key includes a DUPLICATES ARE NOT ALLOWED clause it is possible that duplication of occurrences of the record are detected when the index is set up. The restructuring will not be successful in this case and the new index will not come into operation (the old index remaining if there was one). The offending occurrences will be reported to the Data Base Administrator.

If the DUPLICATES Clause is discontinued it is not even necessary for the restructuring mechanism to set up a new index. All that is required is for the Object Schema definition of the key to be altered to indicate that there is no restriction on duplicate values of the key.

Special consideration must also be given to the introduction of a DUPLICATES ARE FIRST/LAST Clause since this implies that the distinction between record occurrences with equal values for the key in terms of their presentation to application programs will be the chronological or inverse-chronological order of their insertion onto the data base. This implies some form of date/time stamping of records as they are added to the data base.

Task 12 - Addition of a New Set to the Data Base

Function

The network of relationships between records on the data base is established using the Set construct. In this way one type of record (the owner) is associated with one or more other types of record (the members). In any set occurrence there will be one occurrence of the owner record and zero or more occurrences of each of the member records.

It may be necessary for the Data Base Administrator to alter the network for many reasons during the lifetime of the data base. The introduction of new records relating to new entities, unforeseen inter-record relationships and the correction of errors in the initial network definition are typical examples.

The following tasks consider various methods of network modification. In this task we consider the addition of a new set to define the association between existing records.

Program Categorisation

Transparent - All programs which do not require the set in question including those which reference the owner and member records without using the set.

Opaque - Programs which wish to take advantage of the new set relationship.

Concurrent Tasks

No tasks may operate concurrently with this task.

Syntax

The proposed syntax for the task is as follows:-

```
ADD SET Set-Name - 1
DEFINITION (Definition of the Set as described in
            the Codasyl Proposals).
STORAGE     (Definition of the set storage mechanism
            as described in the Codasyl Storage Schema
            Proposals).
(INITIAL SELECTION OF Member-Record - 1 IS BY Procedure - 1) ..
```

The Initial Selection Procedure is required to provide a means of allocating record occurrences to particular set occurrences where the set has no SET SELECTION procedure (e.g. where record occurrences would normally be added by application programs using the INSERT DML command). The procedure differs from the Set Selection Procedure since it must have the ability to provide a result of "no owner" where membership of a set is not mandatory to provide for occurrences of a member record which are not members of any set occurrence.

Implementation Strategy

The principal objective in implementing this task is to set up the pointers (and, if necessary, indices) which identify occurrences of the set. The Codasyl Proposals describe First and Last Owner Record Pointers and Next, Prior and Owner Member Record Pointers. Each Pointer will be associated with one of the storage record occurrences corresponding to each record occurrence. They may point to another storage record occurrence or to an index entry depending on whether direct or indirect pointers are used.

Each storage record will already have other pointers associated with it corresponding to sets where the record is an owner or member. If the pointers for the new set are added after all existing pointers there is no requirement to alter the existing version number of the record since all existing programs can continue to operate using these pointers. It may be more convenient, however, to position the new pointers at some point within the storage record which displaces existing pointers and in this case the version number of the records must be increased to n. For the duration of the restructure, therefore, some record occurrences will be at version n and others will be at lower versions and the DML Execution Routines must take this into account. From the start of the restructuring any new occurrences of the records will be written as version n with the correct pointers. In particular, if the set being added has mandatory membership the pointers in the new record will be set to the value determined by the Set Selection Procedure. Similarly, if a member record occurrence at version n is deleted from the data base the pointers in its corresponding owner record occurrence and adjacent member record occurrences will be reset appropriately (these too must be at version n as described below).

To perform the restructuring dynamically the restructuring mechanism would operate as follows:-

Each set occurrence is associated with a unique owner record occurrence and the restructuring mechanism must firstly establish the existence of each such set by progressing through all occurrences of the owner record in the "Preferred Direction". Each record will be written back to the data base as version n, with the pointer for the new set allocated a value indicating that no member records currently exist for that set.

When all owner record occurrences have been processed the Restructuring Mechanism will set the pointers on each occurrence of each member record by once again progressing through these in the preferred direction. The appropriate owner record is identified using the Initial Selection or Set Selection Procedure.

Addition of the member to the set can then use the techniques used by the DML Execution Routines for adding records to sets when requested to do so by an application program. For a chronologically ordered set (INSERTION IS FIRST, LAST) the Restructuring Mechanism is not in a position to determine the order of the set which must, therefore, be random for existing records although any records subsequently added will be positioned correctly.

Once this process is complete, the set has been fully established and the modified versions of the Opaque Programs can be scheduled for execution.

Task 13 - Deletion of a Redundant Set

Function

The Data Base Administrator may also require to amend the network by deleting redundant sets to free the space occupied by unused pointers and indices. For sets where insertion is automatic the deletion would remove the requirement of the DML Execution Routines to update the pointers when record occurrences are added to or removed from the Data Base.

Before a set can be deleted the restructuring mechanism must ensure that it is indeed redundant. A redundant set is one which is not referenced in any sub-schema and the restructuring mechanism must reference all sub-schemas to ensure that this is the case.

Program Categorisation

Transparent	- All programs
Opaque	- None (since no sub-schema can contain the set no program can reference it).

Concurrent Tasks

No task may operate concurrently with this task.

Syntax

The proposed syntax for this task is as follows:-

DELETE SET Set-Name - 1

Implementation Strategy

The Restructuring Mechanism must firstly free the space allocated to pointers in the owner and member record occurrences. It can do this by progressing through thoese records in the preferred direction and writing version n records to the data base which do not contain these set pointers.

Once this operation is complete the index for the set can be deleted if the set uses indirect pointers.

Task 14 - Addition of a New Member Record to an Existing Set

Function

Since a Set links one type of owner record to a number of types of member records the Data Base Administrator may require to add a new type of member record to those present for an existing set.

Program Categorisation

Transparent	-	Programs which do not refer to the set. Programs which always refer to explicit record names (other than the record being added) when 'Finding' records via the set.
Opaque	-	Programs which wish to take advantage of the set membership of this new record. Programs which do not explicitly refer to member record name when 'Finding' via the set.

Concurrent Tasks

No task may operate concurrently with this task.

Syntax

The proposed syntax for this task is as follows:-

```
ADD MEMBER Record-Name - 1 TO SET Set Name - 1
DEFINITION (definition of set membership as described
            in the Codasyl Proposals)
INITIAL SELECTION IS BY Procedure - 1
```

Implementation Strategy

The strategy used for this task is identical to that used for the second phase of task 12 (adding a new set) except that the DML Execution Routines must be capable of handling requests for access to the set being modified by application programs whose view of the set does not reference the new record. This is possible because at all times the pointer chains for the set are self-consistent. It may be that certain pointer chains have occurrences of the new record at version n but the DML Execution Routines can pass through these pointers to the next occurrence of the type of record they require.

Once the restructuring is complete the new version of opaque programs which reference the set as containing the new record can be scheduled.

Task 15 - Removal of a Member Record from an Existing Set

Function

A further method whereby the Data Base Administrator could alter the network would be to remove an existing member record from participation in an existing set. By implication, other members would still remain in the set since if this were not the case the set could be deleted as detailed in Task 13. The removal of the record from the set might be necessary as a prelude to the deletion of the record or might reflect a loss of the relationship between the entity represented by that record and that represented by the owner record of the set.

Program Categorisation

Transparent	-	Programs which do not refer to the set.
		Programs which always refer to explicit record names (other than the record being removed) when 'Finding' records via the set.
Opaque	-	Programs which do not explicitly refer to member record name when 'Finding' via the set.

Concurrent Tasks

No tasks may operate concurrently with this task.

Syntax

The proposed syntax for this task is as follows:-

REMOVE Record-Name - 1 FROM Set-Name - 1

Implementation Strategy

This task can be implemented by the restructuring mechanism in progressing through all occurrences of the record in question in the preferred direction. Each occurrence will be read as version m and written back as version n (without the pointers for the set). In addition the adjacent record occurrences in the set occurrence must be accessed and their pointers amended to point to each other rather than the removed record (this is similar to the operation carried out to operate the DML Remove command).

As for the previous task this operation maintains set integrity at all times whether occurrences of the removed record are at version n or m. Transparent program can, therefore, continue to operate during the restructuring.

Where a program uses the FIND command to navigate through the set without explicitly referencing record names, its results would be unpredictable during the restructuring since it may or may not be presented with an occurrence of the removed record depending on whether the restructuring mechanism has reached that point. A modified version can be scheduled for introduction at some convenient point before the restructuring which ignores any occurrences of the removed record if they are presented. This procedure is similar to that used in previous tasks for Version Specific programs although in this case only the name of the retrieved record need be examined rather than its version. Similarly, it may be useful to schedule a further version of the program after the restructuring is complete which does not contain the logic to examine the record name since none will now be found.

Task 16 - Amendment to the Order of a Set

Function

The order of a set will often be a significant factor in the processing efficiency and storage requirements of the application programs which use it. Once again the Data Base Administrator must originally define an order which he considers the most appropriate for each set on the data base. Later it may become apparent that access to the records in the set by application programs would overall be more efficient if the set were in a different order.

It is, of course, possible to define a completely new set (Task 12) which is identical to the old in all aspects except the set order. The Data Base Administrator may not consider the overheads associated with maintaining the indices and pointers for both sets acceptable.

Program Categorisation

Transparent	- Programs which do not reference the set. Programs where the order of the set is irrelevant to the logic performed.
Opaque	- Programs where the set order affects the logic performed.

Concurrent Tasks

No tasks may operate concurrently with this task.

Syntax

The proposed syntax for this task is as follows:-
AMEND ORDER OF SET Set-Name-1
This is followed by the Order Clause and any necessary Key Clauses as defined in the Codasyl proposals.

Implementation Strategy

The following strategy may be used to process the task while allowing application programs access to each set occurrence in one order or the other during the restructuring. An attempt to access a set occurrence which is actually being restructured at that time will be delayed until this has been done.

The restructuring mechanism will progress through the occurrences of the owner record of the set. For each it will retrieve all occurrences of associated member records by following the pointer chain for the set in question. When this has been done it will use the Order and Key Clauses specified on the task to re-order that set occurrence. The pointers will be re-set appropriately and all records written back as version n. When all occurrences of the owner record have been processed so have all set occurrences and the task is complete.

Transparent programs where the set order is irrelevant may, therefore, be supplied with records in either the old or new order for a particular set occurrence depending on whether it has been processed by the restructuring mechanism. However, a consistent set occurrence is always presented.

Opaque Programs can continue to operate during the restructuring in much the same way as in Task 7 (Migration of a Data Item). As in that case there are three possible techniques:-

(a) The Opaque Program can be made Version Specific

Since the Version Number of any retrieved tenant record of a set occurrence determines the order of that set (Version n for the new order, some lower version for the old order) the program can take whatever action is appropriate for the order of the set retrieved.

Any records added to the set will be at the version appropriate to that set occurrence - thus they may be restructured later when the restructuring mechanism reaches that point.

The version specific program can be introduced at some convenient point before the restructuring and it will operate on the old set order until the restructuring commences. Similarly, it can be replaced by a non-version-specific program at some convenient point after the restructuring is complete.

b. <u>The original version of the Opaque Program can operate during the restructuring where a period of parallel running is required.</u>

The restructuring mechanism will leave occurrences of owner and member records of the set on the data base as well as writing new records at version n if a period of parallel running is required. Up until the end of the parallel running period there will therefore always be set occurrences in the old order and these will be presented to the opaque program since it is implicitly version specific as previously described. Any attempt to add a record to a set occurrence in the old order must be examined by the DBMS and if there is a corresponding set occurrence in the new order the record must also be added there at the appropriate point.

An amended version of the opaque program using the set in the new order must be scheduled for introduction to coincide with the end of the period of parallel running.

c. <u>The amended version of the opaque program can be introduced to coincide with the start of the restructuring.</u>

If the program requires access to a set occurrence prior to the point currently reached by the restructuring mechanism the DBMS has no difficulty in servicing the request since the set is in the desired order.

If access is required to a set occurrence after the current restructuring point the DBMS must perform the restructuring on that set before servicing the request. The restructuring mechanism will eventually reach this point but it can determine that that set occurrence has already been restructured since the owner record is at version n.

Semantically Equivalent Programs

It may be possible to convert programs which use a set, whose order is being changed where all member records are processed in a consistent manner. In general a sequence of "FIND then process" for each member record occurrence could be replaced by a series of FINDs followed by a sort of the records retrieved with the processing being performed in the exit from the sort.

Chapter 8 - An Implementation of a Data Base Management System Supporting Restructuring

Introduction

A Data Base Management System supporting Restructuring has been implemented on the Edinburgh Regional Computing Centre Multi-access System EMAS. The system is designed as a teaching aid to allow those interested in the interaction of restructuring tasks and DML calls by application programs to experiment and create situations where they consider that contention for shared resources might pose problems for the logic of the restructuring mechanism. The basic system cannot therefore be considered as a practical data base management system since it is interactive with the user on an operation-by-operation basis such that it continually prompts the user to establish which course of action to take and also informs the user of the results of each action. However, a further version of the system has also been created which interacts with the user at a higher level such that he must indicate only which application programs and/or restructuring tasks he wishes to initiate. In this case the usage of computer resources is reported when each task is complete so that the user can guage the overheads associated with the operations performed.

System Structure

Both versions of the system consist of a single FORTRAN program of some 3,000 instructions. There is a primary entry point where data is accepted from the interactive terminal and this dictates which one of a number of alternative paths the program should follow based on a code which the user must key.

There are three main sections of the program.

a. <u>The Application Programs</u>

The first section of the program contains a set of sequences of instructions each of which would normally be performed by an application program in a fullscale data base management system. Those which have been written so far illustrate the types of function which would be required from the application programs operating on a data base maintained by a British Bank.

Each application program has a corresponding code number and the user may initiate a run of any desired program by keying its code number when prompted to do so by the DBMS. In addition, the application programs may request input from the user to simulate the process of data capture. The programs will prompt the user by indicating the type of data required.

b. <u>The Data Manipulation Language Routines</u>

Application programs communicate with the data base by issuing calls to FORTRAN subroutines which directly correspond with the DML commands identified in the CODASYL proposals. Details of the commands are given in Appendix 1.

The DML routines require a Schema to provide them with a description of the data base and a Sub-schema to indicate how that program is to restrict its view of the entire data base. The Schema is an array held in the COMMON area between the subroutines and the application programs. The subschema is an array containing data item names and is passed from program to subroutine as an argument to the CALL instruction.

The Data Base itself is held as a FORTRAN Direct Access file such that each record occurrence has a unique address given by its relative position on the file. Records are not clustered with others of the same record type or with others in the same set occurrence but are simply added at the next available position on the file when they are STORED. To allow records of certain types to be retrieved by their key the prime data file is also supported by a number of index files. One such file is maintained for each version of each one of a number of record types (viz. Customer Records and Account Records at the present time). These files too are FORTRAN direct access files with the relative position of the record indicating the key of the corresponding record in the prime data file. This approach was necessary to overcome the absence of file indexing facilities in FORTRAN.

c. <u>The Restructuring Tasks</u>

In addition to initiating a run of an application program, the user of the system may also choose from a number of codes which allow him to initiate Restructuring Tasks. Those currently available correspond to actions which would realistically be expected in a Bank Data Base but in general are similar to the primative tasks described in Chapter 7. Tasks may use either an open or closed strategy at the user's discretion.

Like application programs, the restructuring tasks are, in fact, sections of the main FORTRAN program although in a fullscale data base they would be system routines at the same level in the system hierarchy as application programs. This similarity of interface level is reflected in the structure of the demonstration system.

Interaction between Application Programs and Restructuring Tasks

The implementation illustrates to the user how the Restructuring Mechanism can operate concurrently with application programs.

For tasks using Closed strategies the user is offered the choice of initiating application programs or continuing with the restructuring after each record (or set of records) has been restructured. Thus, the user can experiment with the effects of initiating different application programs operating on records which have been restructured or are as yet to be restructured as the case may be. This contrasts with the situation for most small data bases in a "live" data base system where a closed restructuring might be completed so quickly that there would be no opportunity to experiment with its various types of interaction with the application programs.

Since the system always informs the user of the actions it is taking as a result of DML commands by application program the user can observe the algorithms being carried out to cater for the handling of records and sets affected by the restructuring.

As a particular case of a Closed Restructuring it is possible to restructure using Parallel Running where not only does the new data base structure gradually materialise but the old structure remains until such time as it is deemed to be no longer necessary.

For an Open Restructuring only the schema is initially altered to reflect the change in data base structure. The user can then observe how the data base itself subsequently alters as application programs add new records to the data base and modify existing records.

The Data Manipulation Language Commands

Application Programs may use the following Data Manipulation Language Commands to access the Data Base.

FIND (Type 1)

Records may be retrieved from the data base by supplying values for up to three concatenated key data items. Thus, for example, an Account record could be retrieved by providing the relevant account number.

The system retrieves the record by access to the appropriate record on the appropriate index file. The index corresponding to the most recent version of the record is searched first followed by the previous version if necessary until all have been exhausted.

FIND (Type 2)

Records may also be retrieved by navigating along the inter-record relationships in the data base using the SET construct. Thus, if a program is currently processing an Account record, the corresponding Customer record could be obtained by navigating to the owner of the corresponding occurrence of the Customer's Accounts set. This navigation is achieved by the use of NEXT,PRIOR,OWNER,FIRST and LAST pointers held with each record occurrence and corresponding to each set of which it is a member or owner.

GET

Data items from the record currently being processed (possibly having been previously retrieved from the data base by use of one of the FIND commands above) are made available to the application program in the User Work Area. Thus, if an Account Record is being processed the Balance of that account may be made available in the User Work Area by using the GET command.

DELETE

The record currently being processed may be removed from the data base by using the DELETE command. The corresponding entry on an index file is also removed.

STORE

New records may be added to the data base by supplying values for their constituent data items in the application program User Work Area. The record will be allocated the next available position on the prime data file. An entry will also be added to the appropriate Index File.

MODIFY

The value of a data item in the record currently being processed can be altered by an application program supplying a revised value in its User Work Area. Thus, if an Account record is being processed, the balance can be modified by adding the value of any transaction to the previous figure.

INSERT

The record currently being processed can be added to a set of which that type of record is a member. The schema determines the occurrence of the corresponding owner record with which the new member is to be assocated. This is established by equality of value between data items in the records.

The set pointers in the record being inserted must be updated by the system in addition to those in the corresponding member record and those of an adjacent member record if such a record exists.

REMOVE

The record currently being processed can also be removed from any set of which it is a member. It is a necessary pre-requisite before a record can be deleted (see above) that it has been removed from all sets of which it is a member.

The set pointers must be deleted from the record being removed and those of the owner record and any adjacent member records must be modified accordingly.

The Bank Data Base

The initial structure of the hypothetical bank data base is as follows:-

The Account is the basic unit of operation of the bank and a data base record is maintained for each bank account. An account is identified by its account number, is held at a specific branch of the bank and belongs to a particular customer. Any customer may hold any number of accounts at one or more branches (indeed this is common practice for business customers).

The following data is therefore held for each Account:-

Account Number
Branch Number
Customer Number
Account Type (Current, Deposit, Loan, etc.)
Balance

A data base record is also maintained for each Branch of the bank containing the following data:-

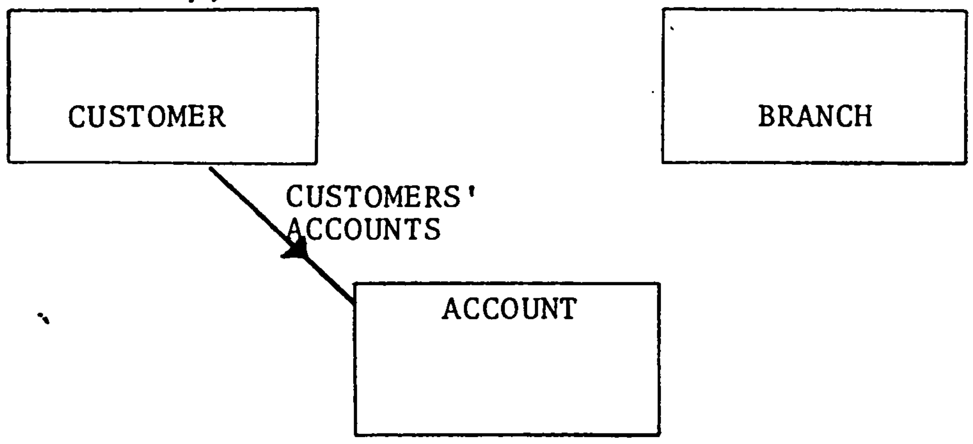
Branch Number
Branch Name
Designation Codes (Urban, Rural, East, West, etc,)

Further, a record is maintained for each Customer of the Bank containing the following items of data:-

Customer Number
Customer Name

The inter-relationship between Customer and Account records is represented by the Customers' Accounts set with an owner record of Customer and a member record of Account.

Diagramatically, the data base is therefore as follows:-



Data Base Programs

Initially, the bank has a number of application programs operating on the data base as described below. Full details of the programs are given in Appendix 2.

The Balance Calculation program calculates the net balance for any specified customer by navigating through his account records using the customers accounts set. When initiated, the program firstly requests the customer number for which the balance is required and the user types this through his terminal. The program then uses the customer number as a parameter to the first type of FIND command (i.e. Find a record given the value of a key data item). This establishes the record for the correct customer as the current of run unit. GET is then used to retrieve the customer's name. A loop of instructions is then carried out to FIND the first and subsequent account records of the customer's accounts set. For each account record found the command GET is used to transfer the contents of the Balance data item to the program user work area. The program then uses the balance to increment a running total of the net balance. When all account records have been found the program prints the final net balance on the user's terminal.

Table 1 - Balance Calculation Record and Set Usage

RECORD/SET \ ACTION	FIND 1	FIND 2	GET	STORE	MODIFY	INSERT
ACCOUNT		X	X			
CUSTOMER	X		X			
BRANCH						
CUSTOMERS ACCOUNTS		X				

The Transaction Posting program allows the balance of an account to be modified by the value of a Debit or Credit transaction (ie the transaction is posted to the account). In a live banking data base this would form part of a double entry book-keeping system handling transactions from various sources such as branch terminals, cash dispensing machines and cheques remitted by other banks. In this implementation the program requests the user to enter the account number and value to be posted on his terminal. A FIND command of the first type is then used to establish the correct account record as the current of run unit. This is followed by a GET to transfer the balance to the user work area.

The value of the transaction is then added to the balance (still in the user work area) and a MODIFY is then carried out to update the account record with the revised balance. Before the run terminates the program prints the new balance on the user's terminal.

Table 2 - Transaction Posting Record and Set Usage

ACTION / RECORD /SET	FIND 1	FIND 2	GET	STORE	MODIFY	INSERT
ACCOUNT	X		X		X	
CUSTOMER						
BRANCH						
CUSTOMER'S ACCOUNTS						

Existing customers may open any number of new accounts at any of the bank branches. The Open New Account program is used in such cases to set up a new account record with an initial balance of zero. It also adds the new account record to the set of such records for that customer. The program does this by firstly requesting the branch, account number, customer number and account type from the user via his terminal. These data items are then moved to the user work area together with a value of zero for the balance. The STORE command is then used to add the new record in the user work area to the data base. The INSERT command is then used to add the new record to the Customer's Accounts set.

Table 3 - Open New Account Record and Set Usage						
ACTION / RECORD /SET	FIND 1	FIND 2	GET	STORE	MODIFY	INSERT
ACCOUNT				X		X
CUSTOMER						X
BRANCH						
CUSTOMER'S ACCOUNTS						X

The <u>Statistics Print</u> program provides statistics on the bank's customers. The first type of FIND command is used to retrieve each customer record in turn from the data base. As each record is found it is counted. When all records have been found the program prints the number of customers on the user's terminal.

Table 4 - Statistics Print Record and Set Usage						
ACTION / RECORD /SET	FIND 1	FIND 2	GET	STORE	MODIFY	INSERT
ACCOUNT						
CUSTOMER	X					
BRANCH						
CUSTOMER'S ACCOUNTS						

The <u>Add New Customer</u> program allows a new customer record to be added to the data base. When the program is run the customer will have no accounts since these will be added later by one or more runs of the Open New Account program. The user enters the customer number and name of the new customer on his terminal and the program moves these to its user work area. The record is then added to the data base when the program issues the STORE command.

Table 5 - Add New Customer Record and Set Usage

RECORD/SET \ ACTION	FIND 1	FIND 2	GET	STORE	MODIFY	INSERT
ACCOUNT						
CUSTOMER				X		
BRANCH						
CUSTOMER'S ACCOUNTS						

The <u>Amend Customer Details</u> program is used to alter the customer's name as currently held on a customer record. The user enters the customer number and amended name on his terminal. The program uses the first type of FIND command to retrieve the appropriate customer record from the data base. It then moves the revised name to the user work area and issues a MODIFY command to alter the data base record.

Table 6 - Amend CustomerDetails Record and Set Usage

ACTION / RECORD /SET	FIND 1	FIND 2	GET	STORE	MODIFY	INSERT
ACCOUNT						
CUSTOMER	X				X	
BRANCH						
CUSTOMER'S ACCOUNTS						

The Restructuring Tasks

The implementation allows a number of restructuring tasks to be carried out on the data base. These tasks are a subset of those described in Chapter 3 and the syntax used has been simplified for ease of use on the interactive terminal. Full examples of the restructuring tasks are given in Appendix 3.

Adding a Customer's Age Group to the Customer Record

Let us suppose that the bank has found that customers of certain age groups have tended to be attracted to its services. Certain marketing strategies will be oriented to these classes of customers while others will be directed to the remaining age groups. To measure the effectiveness of these revised marketing techniques the bank requires statistics on the distribution of its customers by age group. Since this data is not currently maintained on the data base, before any statistics can be produced a new data item (age group) must be added to the customer record.

The Data Base Administrator has an option open to him as to which strategy he is to use to carry out this restructuring task. Both open or closed strategies would be applicable and these are now considered separately.

The first step in an Open restructuring is to define a new version of the customer record on the schema which is identical to the previous version except that it also contains age group. For example, the age group can be added adjacently to the Customer Number data item. Thus immediately after this operation has taken place the schema has a definition of both the old and new versions of the customer record. The data base is unchanged by the restructuring and all occurrences of the customer record therefore remain at the old version.

From the point of the restructuring the 'Add New Customer' and 'Amend Customer Details' programs will write new version records to the data base. These programs have not altered and therefore do not contain the age group in their definition of the customer record - the sub schema to schema mapping for the new version record ensures that null values are written for this data item. The 'Amend Customer Details' program must also read customer records from the data base before it modifies them. These records may be at the old version but may also be at the new version if they have just been written by the 'Amend Customer Details' or 'Add New Customer' programs. Once again the sub-schema to schema mapping for each version ensures that since the programs do not reference the age group the difference in version is not apparent to the programs. At some convenient point in time after the restructuring the Data Base Administrator must ensure that amended versions of the 'Add New Customer' and 'Amend Customer Details' programs which cater for the new data item are scheduled in preference to the previous version. This point in time is dictated by the introduction of revised procedures for branch staff submitting data to the programs such that they include age group. From this point the new version customer records written by the programs will include real values for age group. This is done by the 'Add New Customer' including the age group in its sub-schema. The 'Amend Customer Details' program must also include the age group in its sub-schema but since the record to be modified is accessed using FIND with no GET to the age group there is no difficulty in accessing old versions of the customer record for modification.

The 'Statistics Print' program does not reference the Age Group data item initially but since the objective of the restructuring is to provide the bank Executive with statistics on the distribution of customers by age group the program must be modified to take this new data item into account. As for the 'Amend Customer Details' program a run of the 'Statistics Print' may encounter both old and new versions of the customer record. In this case, however, where a new version record is found the program must GET the new data item so that it can use it in its analysis. For old version records it must not attempt to GET age group since this data item does not exist at that version. To overcome this problem the program must examine the version number of each record accessed and take appropriate action (i.e. the program must be made version specific). It would be likely that the Data Base Administrator would continue to run the existing version of the program for some time after the restructuring until age groups had been supplied for a fair proportion of the customers by the 'Add New Customer' and 'Amend Customer Details' programs. Only from this time will the revised version come into use. Eventually, all customer records might contain the age group data item but this may be a considerable period of time later. It would then be possible to introduce a third (non-version-specific) version of the program which always used GET to obtain the age group data item from each customer record accessed.

The 'Calculate Balance' program accesses the customer record but only uses GET to access the customer name, there being no requirement to ever access the new data item by this program. The sub-schema to schema mapping ensures that the program will operate correctly whether old or new version records are accessed.

The 'Transaction Posting' and 'Open New Account' programs do not reference the customer record and are therefore transparent to this restructuring task.

Table 7 - Chronological Chart of Open Restructuring to Add Age Group to Customer

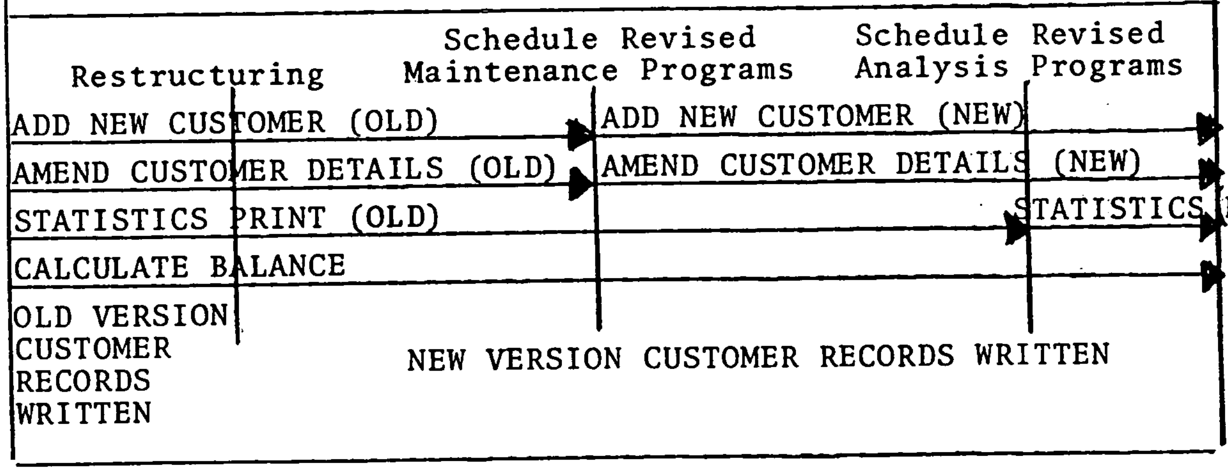

Restructuring
Schedule Revised Maintenance Programs
Schedule Revised Analysis Programs
ADD NEW CUSTOMER (OLD)
ADD NEW CUSTOMER (NEW)
AMEND CUSTOMER DETAILS (OLD)
AMEND CUSTOMER DETAILS (NEW)
STATISTICS PRINT (OLD)
STATISTICS
CALCULATE BALANCE
OLD VERSION CUSTOMER RECORDS WRITTEN
NEW VERSION CUSTOMER RECORDS WRITTEN

Table 8 - Program Transparency for Open Restructuring to Add Age Group

TRANSPARENCY / PROGRAM	DO NOT REFERENCE	TRANSPARENT	OPAQUE	VERSION SPECIFIC
ADD NEW CUSTOMER			X	
AMEND CUSTOMER DETAILS			X	
STATISTICS PRINT				X
CALCULATE BALANCE		X		
TRANSACTION POSTING	X			
OPEN NEW ACCOUNT	X			

Now consider a Closed strategy to add the age group to the customer record. The Data Base Administrator would firstly choose some time when the data base was not very volatile (e.g. overnight) and would schedule the restructuring to run at this time. After adding details of the new version of the customer record to the schema the restructuring mechanism will progress through each customer record occurrence and expand it at the appropriate point to accommodate the new age group data item by inserting null values. During this process the data base is in a transitional state. The 'Add New Customer' and "Amend Customer Details' programs will continue to operate during the transitional period in the form that they did beforehand. If the 'Amend Customer Details' program encounters a restructured record the sub-schema to schema mapping for the new version record will ensure that the program operates correctly. As for an open strategy new version records will always be written by both programs with the sub-schema to schema mapping allowing null values to be allocated to age group. Then it may be that the restructuring mechanism encounters a new version record as it is progressing through the occurrences but it can recognise this as having been restructured and take no further action. The 'Statistics Print' would probably not be run during the Transitional Restructuring Period (if it were the old version could continue to run as for an open strategy.)

After the restructuring is complete a revised version of the program can be introduced which GETs the age group from each customer record. This version of the program will consider the age group of a customer with null values in this data item as unknown. The 'Calculate Balance', 'Transaction Posting' and 'Open New Account' programs are not affected by this restructuring since they do not reference the new data item and operate satisfactorily whether they access the new format or old format of customer record.

Table 9 - Chronological Chart of Closed Restructuring to Add Age Group to Customer

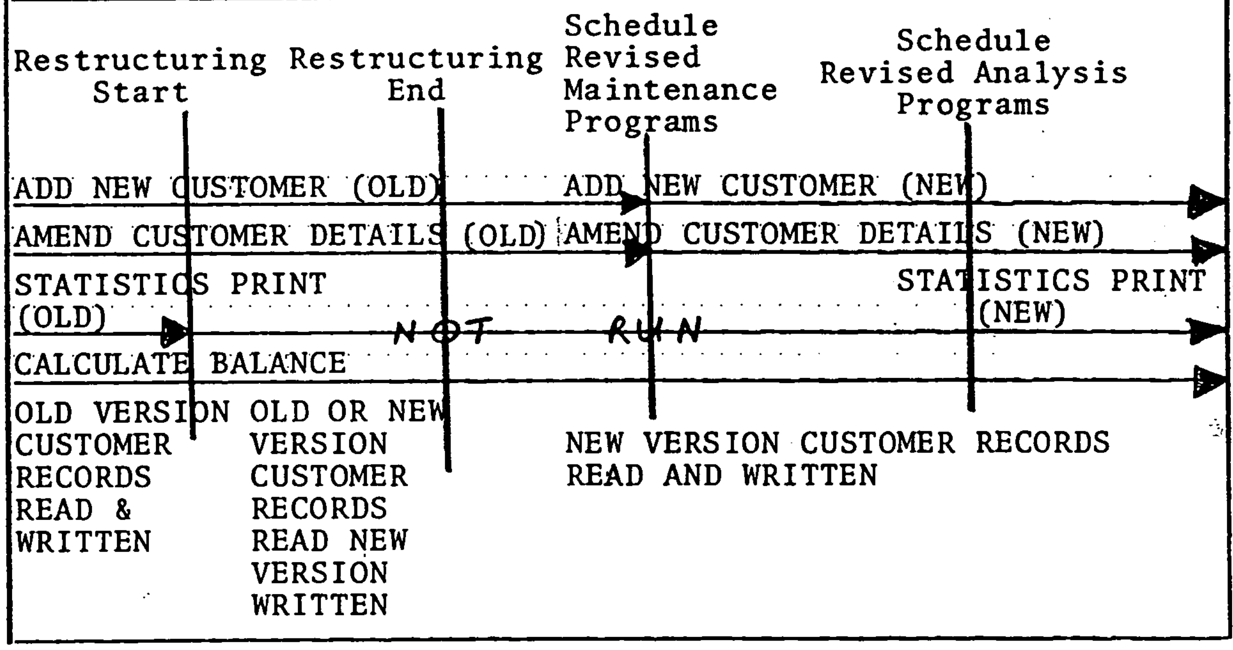

Restructuring Start
Restructuring End
Schedule Revised Maintenance Programs
Schedule Revised Analysis Programs
ADD NEW CUSTOMER (OLD)
ADD NEW CUSTOMER (NEW)
AMEND CUSTOMER DETAILS (OLD)
AMEND CUSTOMER DETAILS (NEW)
STATISTICS PRINT (OLD)
NOT RUN
STATISTICS PRINT (NEW)
CALCULATE BALANCE
OLD VERSION CUSTOMER RECORDS READ & WRITTEN
OLD OR NEW VERSION CUSTOMER RECORDS READ NEW VERSION WRITTEN
NEW VERSION CUSTOMER RECORDS READ AND WRITTEN

Table 10 - Program Transparency for Closed Restructuring to add Age Group

TRANSPARENCY / PROGRAM	DO NOT REFERENCE	TRANSPARENT	OPAQUE	VERSION SPECIFIC
ADD NEW CUSTOMER			X	
AMEND CUSTOMER DETAILS			X	
STATISTICS PRINT			X	
CALCULATE BALANCE		X		
TRANSACTION POSTING	X			
OPEN NEW ACCOUNT	X			

Deleting a Customer's Age Group from the Customer Record

Just as the bank could find it necessary to add age group to the customer record, it could then decide that this data was no longer relevant to its statistical reports and that the data item need therefore no longer be captured and maintained. This process would most likely be carried out using a closed restructuring to allow the Data Base Administrator to take advantage of the free space thus made available.

Firstly the Data Base Administrator must arrange that revised versions of the'Add New Customer' and 'Amend Customer Details' programs are brought into use which do not reference the age group data item. This must be synchronised with the introduction of revised procedures within bank branches such that age group is no longer supplied as data to these programs. Both before and after this point the programs will write customer records with age group but latterly this data item will have null values because of the sub-schema to schema mapping. The Data Base Administrator will then schedule the restructuring for some time when it would have minimal impact on normal data base processing (e.g. overnight).. The restructuring will progress through each customer record occurrence and will write a new version record without the age group data item. As for adding a data item the 'Add New Customer' and 'Amend Customer Details' programs can continue to operate while this is done and they will always write new version customer records which will be ignored by the restructuring mechanism if they are subsequently encountered. The 'Statistics Print' must also be modified so that its sub-schema does not reference the age group. This revised version can be run at any time before, during or after the restructuring as desired. The 'Calculate Balance', 'Transaction Posting' and 'Open New Account' programs are, once again, unaffected by the restructuring.

Table 11 - Chronological Chart of Closed Restructuring to Delete Age Group

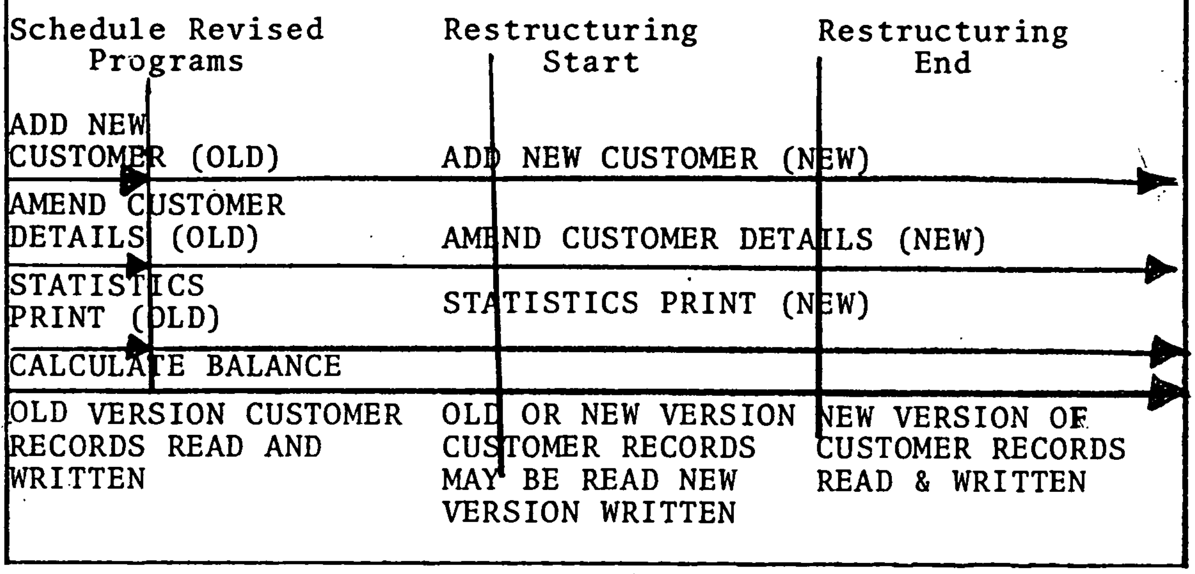

Schedule Revised Programs
Restructuring Start
Restructuring End
ADD NEW CUSTOMER (OLD)
ADD NEW CUSTOMER (NEW)
AMEND CUSTOMER DETAILS (OLD)
AMEND CUSTOMER DETAILS (NEW)
STATISTICS PRINT (OLD)
STATISTICS PRINT (NEW)
CALCULATE BALANCE
OLD VERSION CUSTOMER RECORDS READ AND WRITTEN
OLD OR NEW VERSION CUSTOMER RECORDS MAY BE READ NEW VERSION WRITTEN
NEW VERSION OF CUSTOMER RECORDS READ & WRITTEN

Table 12 - Program Transparency for Deleting Age Group

PROGRAM \ TRANSPARENCY	DO NOT REFERENCE	TRANSPARENT	OPAQUE	VERSION SPECIFIC
ADD NEW CUSTOMER			X	
AMEND CUSTOMER DETAILS			X	
STATISTICS PRINT			X	
CALCULATE BALANCE		X		
TRANSACTION POSTING	X			
OPEN NEW ACCOUNT	X			

Expanding the Balance Data Item

The bank originally considered £99,999 as the largest balance which could be maintained for an account and the balance data item on the account record was formatted accordingly.

Inflation took its toll and this figure eventually no longer provided a realistic maximum and a format sufficient to hold £999,999 is now required. This task will now be illustrated for a closed strategy since many of the considerations for an open strategy are also covered in this case.

The Data Base Administrator must firstly arrange for a version specific version of the 'Transaction Posting' program to be brought into use at some convenient point prior to the restructuring. The program would examine the version number of each account record retrieved and handle the balance accordingly (for old version records the balance would be printed as 5 digits, for new version records the balance would be printed as 6 digits). Thus, the branches supplying data to the 'Transaction Posting' program would not be aware that the program had been amended since all records would be at the old version when retrieved and would be written back as the old version since the restructuring would not yet be under way.

The restructuring itself will then commence at a later time when the data base is not volatile (e.g. overnight). The restructuring mechanism will progress through each account record in turn, expanding the record to accommodate the expanded balance and then writing it back to the data base as the new version. If a 'Transaction Posting' run is required during this transitional phase it may encounter either an old or new version of the account record but, as we have seen, since the program is version specific, either type of record will be handled satisfactorily, although now new version records will always be written.

The 'Calculate Balance' program also accesses the balance data item from the account record but unlike the 'Transaction Posting' program it does not modify the data item. Therefore, assuming that the program will cater for the larger of the two formats (i.e. 6 digits) the program is transparent and there is no requirement to make it version specific.

Table 13 - Chronological Chart to Expand Balance Data Item

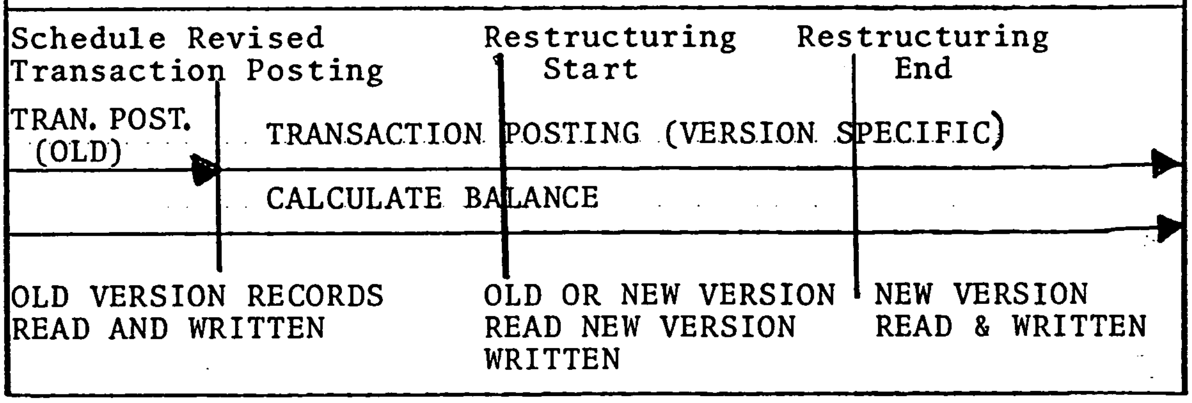

Schedule Revised Transaction Posting
Restructuring Start
Restructuring End
TRAN. POST. (OLD)
TRANSACTION POSTING (VERSION SPECIFIC)
CALCULATE BALANCE
OLD VERSION RECORDS READ AND WRITTEN
OLD OR NEW VERSION READ NEW VERSION WRITTEN
NEW VERSION READ & WRITTEN

Table 14 - Program Transparency for Expanding Balance Data Item

PROGRAM \ TRANSPARENCY	DO NOT REFERENCE	TRANSPARENT	OPAQUE	VERSION SPECIFIC
ADD NEW CUSTOMER	X			
AMEND CUSTOMER DETAILS	X			
STATISTICS PRINT	X			
CALCULATE BALANCE		X		
TRANSACTION POSTING				X
OPEN NEW ACCOUNT		X		

Contracting the Balance Data Item

As a converse to the previous task the bank might require to reduce the space allocated to the balance data item on the account record from 5 digits to 4 digits. In this case, there is no question of a version specific 'Transaction Posting' program since the existing program will continue to operate on the reduced size of data item because of the sub-schema to schema mapping. One effect of the change, however, would be that any attempt to store or modify a record with a balance of more than £9999 would fail after the restructuring was under way. Equally, any record with a balance of more than £9999 could not be restructured successfully.

Table 15 - Program Transparency for Contracting Balance Data Item

TRANSPARENCY / PROGRAM	DO NOT REFERENCE	TRANSPARENT	OPAQUE	VERSION SPECIFIC
ADD NEW CUSTOMER	X			
AMEND CUSTOMER DETAILS	X			
STATISTICS PRINT	X			
CALCULATE BALANCE		X		
TRANSACTION POSTING		X		
OPEN NEW ACCOUNT		X		

Intra-Record Structure Amendment

Although the implementation does not support vectors or repeating groups of data items some of the characteristics of altering intra-record structure are illustrated by allowing the position of data items to be interchanged. In fact, the position of data items is irrelevant to application programs because of the sub-schema to schema mapping. All programs are therefore transparent to the transposition of data items within records (say the Customer Number and Branch Number within the Account Record).

Table 16 - Program Transparency for Inter-Changing Customer Number and Branch Number

TRANSPARENCY / PROGRAM	DO NOT REFERENCE	TRANSPARENT	OPAQUE	VERSION SPECIFIC
ADD NEW CUSTOMER	X			
AMEND CUSTOMER DETAILS	X			
STATISTICS PRINT	X			
CALCULATE BALANCE		X		
TRANSACTION POSTING		X		
OPEN NEW ACCOUNT		X		

Migration of Balance from Account to Customer Record

The bank may make a policy decision to consider a customer's net balance as the primary indication of his solvency rather than the separate balances in his individual accounts. A corresponding restructure of the data base to migrate the balance data item from the account record to the customer record would then be in order.

This task, like those discussed previously, could be carried out using a closed strategy at some time when the data base was not volatile. Each customer record would be accessed in turn and for each such record accessed the restructuring mechanism will use the set pointers to access all corresponding account records. New versions of the customer record and the corresponding account records are then written back to the data base having added the values in the balance data item from each of the account records to produce the value for this data item in the new version customer record. If parallel running is being carried out the old version records will remain on the data base. Note that application programs have no opportunity to access the data base while this operation is being carried out on a particular set occurrence.

Before the restructuring commenced the Data Base Administrator must have ensured that the "Transaction Posting" program was made version specific so that it could update the balance of an old version Account record if it encountered one or find the appropriate customer record and then update the balance on this if a new version account record was encountered. If parallel running is taking place and a new version record is encountered the program must not only update the balance on the associated new version customer record, but also on the corresponding old version account record which will still be on the data base (addressed by a pointer on the new version record). Similarly, a version specific version of the 'Calculate Balance' program must be brought into use before the restructuring so that the balance is derived directly from new version customer records and by summation of balances on account records for old version records. In a parallel running situation an Audit program will also be run which will not only obtain the balance from the customer record but will also access the old version account records and recompute the balance. The two values will be compared and an error reported if they are not equal.

After the restructuring and period of parallel running (if applicable) are complete, further versions of the 'Calculate Balance' and 'Transaction Posting' programs will be introduced which operate only on new version customer and account records.

Table 17 - Chronological Chart for Migration of Balance to Customer Record

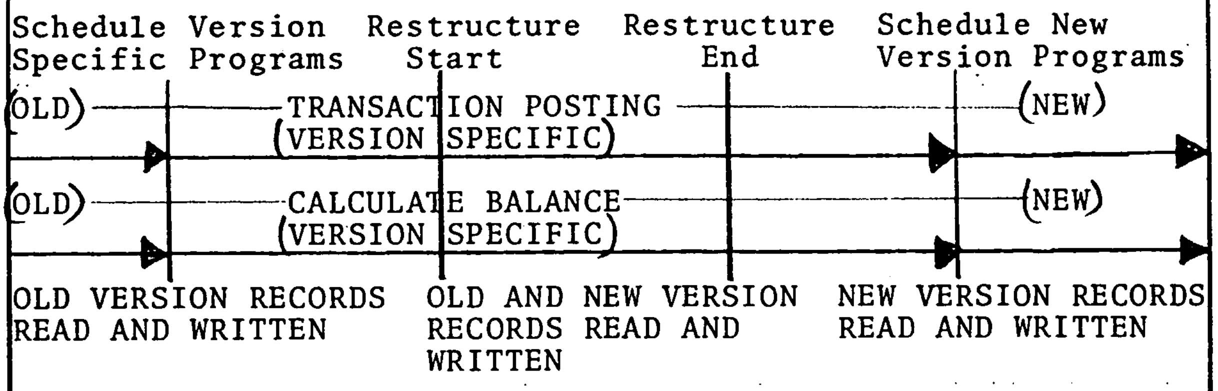

Schedule Version Specific Programs
Restructure Start
Restructure End
Schedule New Version Programs
(OLD)
TRANSACTION POSTING (VERSION SPECIFIC)
(NEW)
(OLD)
CALCULATE BALANCE (VERSION SPECIFIC)
(NEW)
OLD VERSION RECORDS READ AND WRITTEN
OLD AND NEW VERSION RECORDS READ AND WRITTEN
NEW VERSION RECORDS READ AND WRITTEN

Table 18 - Program Transparency Balance Data Item Migration

TRANSPARENCY PROGRAM	DO NOT REFERENCE	TRANSPARENT	OPAQUE	VERSION SPECIFIC
ADD NEW CUSTOMER		X		
AMEND CUSTOMER DETAILS		X		
STATISTICS PRINT		X		
CALCULATE BALANCE				X
TRANSACTION POSTING				X
OPEN NEW ACCOUNT		X		

Amendment to the Key for the Account Record

The key for the account record is originally a two-digit account number. It may be that the restriction of a two-digit number proves inadequate and this must be expanded to three digits in a similar way to the expansion of the balance data item above. The mechanism to carry out this expansion is as for a normal data item but the DBMS must cater for the existence of both old and new format records during the transitional phase. Thus if the 'Transaction Posting' program is run during the transition there ar two possible ways of selecting the desired account record - either characters 1 to 2 of an old version record or characters 1 to 3 of a new version record being equal to the specified account number. When the restructuring is complete all account records will be at the new version and once again a single criterion (the three-digit account number) will be used if the 'Transaction Posting' program is run at this time.

If the key is being altered to encompass more than the existing account number data item there is no requirement to alter record occurrences since the implementation uses set pointers rather than set indices and the restructuring is therefore an instantaneous event. Suppose that the account type is added to the key - all that is required is that the schema entry for the old version of the account record is amended to record this fact. This change to the schema requires alterations to application programs. For example, after the restructuring the 'Transaction Posting' program must provide the type of account in addition to the account number for each account record to be found. The scheduling of the new version of the program and the associated changes to branch procedures must therefore coincide exactly with the restructuring.

Table 19 - Chronological Chart of Add Account Type to Record Key

Restructuring	
TRANSACTION POSTING (OLD)	TRANSACTION POSTING (NEW)
ACCOUNT NUMBER SUPPLIED TO ACCESS RECORD	ACCOUNT TYPE AND NUMBER SUPPLIED TO ACCESS RECORD

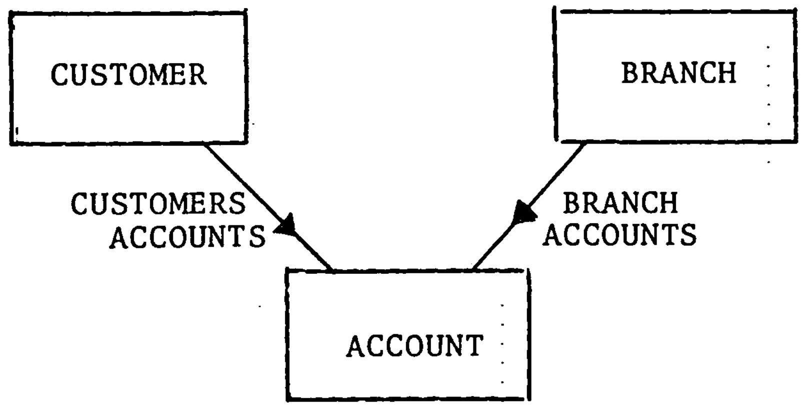
Table 20 - Program Transparency for Amending Account Record Key

PROGRAM \ TRANSPARENCY	DO NOT REFERENCE	TRANSPARENT	OPAQUE	VERSION SPECIFIC
ADD NEW CUSTOMER	X			
AMEND CUSTOMER DETAILS	X			
STATISTICS PRINT	X			
CALCULATE BALANCE		X		
TRANSACTION POSTING			X	
OPEN NEW ACCOUNT		X		

Addition of the Branch Accounts Set

There is a relationship between the Branch record and the account record (the Branch number is held on the account record) but let us suppose that the Data Base Administrator neglected to define the set when the data base was created since there was no obvious requirement to use the relationship in any application program. Eventually a requirement for the relationship arose in that a program was required to calculate the net balance of all accounts at a Branch.

The diagrammatic representation of the data base is therefore as follows:



The Data Base Administrator therefore has to set up the new set using a Closed restructuring strategy. The restructuring mechanism can do this by firstly progressing through the Branch records so that they have their version number increased to reflect that they have had pointers allocated as First and Last Owner Record Pointers for the new set. Then the restructuring mechanism can progress through the account records, and having derived the appropriate set occurrence for each member record, it can add Next, Prior and Owner Member Record Pointers to each record and modify the pointers of the owner record appropriately. Once again, the record version number is also increased.

From the start of the restructuring a new version of the 'Open New Account' program must be scheduled so that if any new records are written to the data base, they will also be inserted into the 'Branch accounts' set in addition to the 'Customer's accounts' set. The 'Calculate Balance' program is transparent to this restructuring task since, although it too uses pointers in the account record, the DBMS can ensure that the correct pointers are used whether an old or new version account record has been accessed.

A new program will be required after the restructuring is complete. The 'Calculate Branch Balance' program will navigate through all account records in an occurrence of the Branch Accounts set and calculate the net balance. When all such account records in the set have been accessed the program prints the balance on the user's terminal.

Table 21 /

Table 21 - Chronological Chart for adding Branch Accounts Set

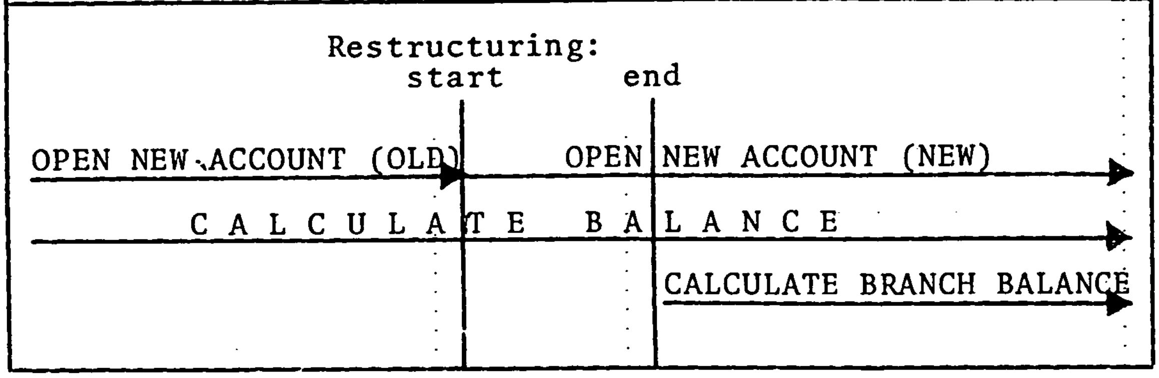


Table 22 - Program Transparency for adding Branch Accounts Set

TRANSPARENCY / PROGRAM	DO NOT REFERENCE	TRANSPARENT	OPAQUE	VERSION SPECIFIC
ADD NEW CUSTOMER	X			
AMEND CUSTOMER DETAILS	X			
STATISTICS PRINT	X			
CALCULATE BALANCE		X		
TRANSACTION POSTING	X			
OPEN NEW ACCOUNT			X	
CALCULATE BRANCH BALANCE	NEW PROGRAM			

Comparison Between Restructuring Strategies.

In order to demonstrate that it is feasible to implement the concepts of Open and Closed Dynamic Restructuring Strategies within a practical Data Base Management System designed to meet the requirements of the community of users of a shared centralised data base several runs of the version of the EMAS implementation which reports resource consumption rather than individual operations were also made. Like the tutorial version, he resource-monitor version is based on a monolithic FORTRAN program which simulates application programs in sections of its coding. These 'programs' call DML subroutines where appropriate. Restructuring operations are also simulated by sections of the program in conjunction with a general-purpose restructuring routine.

In fact, those instructions which are used in the tutorial environment to illustrate to the user the activities being performed on the data base and schema by the DML routines have been removed. Similarly the activities being carried out by the restructuring routine are no longer reported. In their place, logic has been added to inform the user of the extent of the computer resources being expended on the executions of application programs which he has initiated. The resultant system therefore gives a similar interface to the end user to that which he would expect from a full scale implementation and in addition it incorporates a resource usage monitor.

EMAS measures resource consumption in terms of the following parameters:

a) C.P.U. Time

b) Connect Time

c) Page Turns

d) Allocated Charge

The values of these parameters for the current EMAS session are displayed to the user both before and after the application program run. Thus the differences will provide a measure of the resources used. The number of DML calls and records restructured during the run are also provided to give a perspective on the activities performed. The number of reads and writes to the prime data file and index files are also given as a measure of the Input/Output activity.

The EMAS multi-user environment is very different from that which would apply to a DBMS supporting a large centralised data base. For example all files used by a program are effectively an extension of that program's virtual memory at run time. Thus Input/Output accesses are reflected in Page Turn activity although this must also cover paging caused by execution switching between different sections of the program object code and also contention for the use of resources with other applications running at the same time. It is important, therefore, that the measurements produced by EMAS itself are treated only as relative to each other in so far as this is possible. The absolute values of the figures can in no way serve as a guide as to how efficient a full implementation of such a DBMS might be. Nevertheless they do provide a basis for comparison (of CPU time in particular) both in normal circumstances and during a restructuring. It is well known that Data Base Management

Systems of this type (e.g. IMS) consume significant amounts of computer resources and it is the objective of this enhancement to the EMAS implementation simply to demonstrate that both Open and Closed Dynamic Restructurings would not impose further overheads which were of an order of magnitude greater than those imposed by the DBMS itself.

The usage of Index Files gives a particularly deceptive picture of potential resource consumption since page turns are often required in switching between prime data and index whereas straightforward sequential searching through the prime data is relatively inexpensive. This is only true, of course, when the volume of data is small and retrieval by sequential search could not be tolerated in a practical implementation of a DBMS.

Appendix 4 gives details of activities on the Bank data base which illustrate the consumption of resources by "Off-Line" Static, In-place Static, Closed Dynamic and Open Dynamic strategies.

Chapter 9 - Summary and Conclusions.

Summary

This thesis has identified a problem which will have a profound effect on the rate of acceptance of data base management systems by the data processing community. The ability to manipulate the structure of a centralised data base will be a vital element in enabling a satisfactory data model to be maintained.

The approach to the problem differs from that of other researchers in the motivation for the provision of a restructuring mechanism. Existing implementations provide reorganisation facilities to allow the consumption of computer resources to be tuned when the placement of data on secondary storage devices has led to inefficiency. Other implementors have observed that the same logical data structure may be represented in different ways within a data base schema and it is therefore possible to transfer from one representation to another. The desirability of application program stability is considered as paramount in both situations although it is more difficult to achieve in the second than in the first.

When restructuring is viewed as a tool to allow the data base administrator to maintain a continuously evolving data model the desire for application program stability must be weighed against other factors. If stability can be achieved it is beneficial, but it must be viewed in the context that it was changes to the application systems which probably dictated the data structure change in the first place. It is also likely that maintenance of application programs will be more straightforward if they all operate on subsets of the same data base structure. The approach here is therefore, to propose a restructuring mechanism which will allow the organisational data model to evolve while providing facilities to permit application programs to alter to reflect the new structure as conveniently as possible.

This chapter discusses how well this objective has been achieved.

The Functionality of the Restructuring Mechanism.

The adequacy of the restructuring mechanism to perform the tasks which data base administrators will demand of it can only be demonstrated by empiric evidence. The same comment can also be made of the CODASYL Data Base Task Group proposals and indeed of any programming language. The distinction between a 'good' and a 'bad' language or data base management system tends to be based on subjective considerations such as the background of the user (the data base administrator in this case), the amount of data which must be supplied and the ease of assimilation of the final result. After due consideration the data base task group adopted the record and set construct and a syntax similar to that of COBOL. Undoubtedly, the reason for this was that is was seen as an extension of the techniques used for data definition in conventional data processing. For this reason the restructuring mechanism is based on a clause-by-clause analysis of the DBTG proposals and the syntax retains the style of the Data Definition Language. To this extent it is therefore maintained that the proposed mechanism will be acceptable within the professional data processing community. It may be that the mechanism (and indeed the record and set concept) would not be acceptable outside this community but the contention is that a data processing profession will continue to exist and it is they who will be responsible for the maintenance of the single centralised data base of each organisation.

Given this restriction on the population likely to come into direct contact with the restructuring mechanism the experiments described for realistic restructuring tasks on a Bank data base illustrate its applicability to many practical situations.

It must also be possible to implement the proposed restructuring tasks and in Chapter 7 at least one implementation strategy (in addition to the process of unloading and reloading the data base) has been described. The interaction between restructuring mechanism and the data base management system can be somewhat complex but varies from task to task. The protocols to be observed for each task have been described in Chapter 7 such that application programs are always presented with a consistent data base structure even while the data base is in the process of restructuring. The strategies postulated in general allow restructurings to take place with minimal impact on those application programs operating on the data base elements being restructured and with little or no impact on other application programs. For each task the level of impact has been identified and criteria have been established for the categorisation of application programs as transparent or opaque.

The absolute requirement for accuracy in the data held on a centralised data base has been catered for by the provision of version specific programs and a parallel running state where both the old and new structures can co-exist. The potential for computer fraud could become so rampant as the sophistication of access to stored data increases that it is vital that a controlling mechanism is provided so that auditors can satisfy themselves that restructurings have been carried out satisfactorily. Similarly some measure of control must be exercised over the initiation of runs of new versions of application programs.

The Convenience of the Restructuring Mechanism.

A centralised data base will inevitably be a vital resource for the organisation which uses it. It follows that the data base must be available whenever possible and in many cases must be available continuously. Restructuring by unloading and reloading the data base (or even part of it) is therefore more than an inconvenience and tends to become an impossibility. The techniques for dynamic restructuring outlined in Chapter 7 are therefore essential to the success of a restructuring mechanism for a centralised data base. On the other hand, such considerations are not so relevant to devolved data bases because of their restricted communities of users and often because of the size of devolved data bases. When their users are absent from their terminals will lie dormant (e.g. at weekends) and may be restructured using a static mechanism at these times.

The interrelationships between the restructuring mechanism and data base management system (particularly the Data Manipulation Language Execution Routines) described in Chapter 7 and demonstrated in Chapter 8 serve to draw the attention of future DBMS implementors to the central position which restructuring must occupy in their system design. Only if such considerations are taken into account at an early stage in the design process will a DBMS be produced which will provide Data Base Administrators with a sufficiently versatile tool with which to restructure their data bases on a timescale consistent with the rate at which they are presented with changes in the real world which their data bases model. Existing data base management systems such as IMS, IDS/II and IDMS would require significant redesign to incorporate a satisfactory restructuring mechanism but this could be achieved without compromising existing application programs and is therefore considered to be eminently desirable. Only when sophisticated restructuring mechanisms are available will the concept of a data base really be exploited by the data processing industry. Although data base management systems have been available for several years there is little evidence of them being widely used as the basis for the most fundamental applications within organisations - those applications which were tackled in the early days of computerisation. In many cases the primitive data storage techniques of early application systems have been perpetuated because the advantages of flexibility of structure are not apparent in today's data base management systems and consequently there has been little motivation for system redesign. A sophisticated restructuring mechanism provides such flexibility and should encourage data processing departments to undertake such redesign of their basic application systems. Great care must be taken in assessing statistics on the current usage of data base management systems since in many cases the philosophy of data base is not being used to best advantage with several disjoint applications because of the limited way in which existing data base management systems can respond to external change.

The decomposition of a restructuring into a sequence of primitive tasks as postulated in Chapter 7 also has an element of convenience for the Data Base Administrator. More complex tasks may be devised by combinations of the primitive tasks and no doubt these could be retained on a library for possible subsequent use. The use of concurrent tasks would provide benefits in terms of the total elapsed time of a restructuring and the use of conditional tasks would provide for an escape mechanism in the event of a failure of a previous task. Overall, a language will be provided in which the Data Base Administrator will be expected to become proficient and once he has done so he will be able to conduct even the most complex restructuring with ease and confidence. Further, the use of a specialised language does not compromise the process of exploration of the structure of the centralised data base by interested parties other than the Data Base Administrator himself. Systems analysts, casual users and sophisticated end users are examples of individuals who would have a requirement to navigate the meta data base (the object version of the schema) in a manner appropriate to their own background and knowledge of the data structure.

The Efficiency of the Restructuring Mechanism.

The efficiency of a restructuring mechanism will be an important constraint on its acceptance by the data processing community. Data Base Management Systems have often been notorious in their consumption of computer resources when compared with those required for conventional techniques for data storage and retrieval. Nevertheless, the consumption of computer resources should be an acceptable price to pay for all of the advantages which a DBMS provides - so long as the additional overheads are not out of all proportion to the resources required to run the application system in the first place. Similarly, a restructuring mechanism which operates dynamically in conjunction with the DBMS itself must be expected to impose further overheads. Some measure of overhead will be acceptable but there will come a point at which restructuring techniques would impose intolerable overheads.

Static Restructuring is considered to impose intolerable overheads for many centralised data bases. Open and Closed Dynamic Restructurings, however, are considered to be compatible with existing data base access patterns since they normally operate on a record by record basis and any degradation in system response time should be limited to that of a single record access. In many cases such degradation could be expected in normal operation where one run unit is required to wait on the initiation of a data access by another run unit. For both open and closed techniques the restructuring is a subsidiary operation and should not unduly degrade response during periods of intense application program activity. An open strategy will impose additional CPU activity since each record occurrence may have to be restructured once it has been retrieved from the secondary storage medium. In the EMAS implementation this increase in activity is significant but in a full scale implementation (where the DBMS and restructuring mechanism were written in a low level language) it could be expected that this ratio would be considerably reduced. Nevertheless, so long as this were to be an overhead to be incurred for a fairly short period of time (say a day), and even then only by those programs which were updating the records to be restructured, many installations would be able to carry the overhead without appreciable degradation of response time for application programs. Processing power is becoming a much more freely available commodity and many installations normally run at far less than peak capacity.

A closed strategy will impose additional Input/Output overheads and will imply a different pattern of disc head movement than the norm (viz: continual movement back to the sequence of records currently being restructured after each interruption due to a DML call by an application program). Judicial choice of the timing of the restructuring will dictate its overall impact on response time. So long as it is scheduled to coincide with periods of relative inactivity of application programs it should be completed in a reasonable timescale without adversely affecting the response times.

The nature of the implementation of the DBMS and restructuring mechanism has precluded the collection of adequate statistical information to permit a meaningful conclusion to be reached on the overheads which could be expected from a full-scale restructuring mechanism. The 100% increase in CPU time which is evident from the experiments quoted in Appendix 4 must be considered more as an upper bound than a practical value. Since the DBMS was written in FORTRAN (as were the restructuring operations and the application programs themselves) the total CPU overhead tends to dwarf any attributable to one particular source. The lack of such statistics however, are not considered to imply that a restructuring mechanism would be prohibitively expensive in processing power - on the contrary the work of

Sockut (in his case to reorganise the storage records to give better overall data base performance) supports the idea that concurrent alterations to data base records together with access by application programs will be feasible. Similarly the Data Base Administration Working Group assume that a concurrent reorganisation mechanism can be implemented without unduly degrading application program response. The restructuring mechanism proposed here would use the same techniques as a reorganisation mechanism such as that suggested by Sockut - indeed it would use the same techniques as the DML execution routines to protect against deadlock etc.

The efficiency of the proposed restructuring mechanism stems from the ability of the Data Base Administrator to choose a strategy appropriate to the section of the data base being restructured and the characteristics of the application programs operating on it. It does not preclude restructuring using a static strategy of unload-amend-reload nor does it restrict restructuring by the population of new records and sets using ad-hoc application programs where this is seen to be the most convenient approach.

The Future of Data Base Management Systems.

The debate on the relative merits of different types of data base management systems is likely to continue within the data processing community for many years to come. Just as in the field of programming languages, the ingenuity of researchers will probably yield increasingly sophisticated ways in which human beings can store and retrieve data.

It is the contention of this thesis, however, that the most fertile round for such research will be in the area of devolved data bases where emphasis can be placed on the retrieval of information (be it information captured by some central incontrovertible source or data captured by the user of the devolved data base himself). Not only will such developments be concerned with the formulation of the retrieval requests in a concise, unambiguous manner but they will also address ways of directing the end user (particularly casual users) towards information which may be relevant to his request but the existence of which he may be unaware.

Equally, research will identify techniques for the capture of data to be held on devolved data bases and for mapping data from the centralised data base into a more easily assimilated form on the devolved data bases (e.g. by snapshotting at convenient times such as end of month). Nevertheless, data captured directly by end users (without formal controls having been established by a Systems Analyst) cannot be universally agreed as correct within the organisation. At best the data can be considered as likely to be correct because its values fall within previously defined constraints.

The equation of research into retrieval and capture of data on devolved data bases with research into techniques for the maintenance of centralised data bases cannot be considered as valid. It would be naive to assume that the maintenance of centralised data bases will be in the hands of anyone other than data processing professionals for the foreseeable future (if only because this would ensure that the organisation has some control over the accuracy of the data used for its own data model). The direction of research in this area should be directed towards an gradually increasing level of sophistication of the software tools available to the Data Base Administrator and Application Programmer. The enormous investment in procedural programs, for example, presents a formidable barrier to anything but an enhancement to an existing procedural language. The concepts of centralised and devolved data bases should contribute to the categorisation of research work towards particular types of individual.

Coincidentally, computer hardware appears to be developing in a direction which would tend to support the concepts of centralised and devolved data bases. More and more powerful mainframe computers are being developed which will permit more parametric users to interact with a centralised data base in a manner prescribed for them by the systems analyst. On the other hand microcomputers with large scale storage available on disc are now commonplace. These will probably be the ideal vehicle for holding devolved data bases where the end users of the data can have control of its structure as well as its contents so long as any information derived from it is viewed purely within this context.

This thesis considers that the CODASYL proposals are oriented towards the centralised data base maintained by the data processing professionals. The restructuring strategies are particularly relevant in this area as are the techniques for allowing several application programs to alter in a controlled manner to reflect the revised structure. The work recorded here has little relevance in the devolved data base environment.

Future Developments of the Work Recorded Here.

The groundwork has been laid in this thesis for a restructuring mechanism which would be compatible with a commercially available CODASYL Data Base Management system such as IDS/II or IDMS. A significant investment in redesign would be required by the implementors of such an item of software and it is probable that the Data Base Administration Working Group proposals for a storage schema would have to be implemented (together with a corresponding reorganisation mechanism) before the provision of a restructuring mechanism. Perhaps the impetus for such an enhancement to CODASYL data base implementations should come from the inclusion of a restructuring mechanism in a future CODASYL Data Base Journal of Development. This thesis should not only identify the requirement for the committee to consider proposals for a restructuring mechanism but also it should provide the basis of what those proposals would eventually contain.

Certainly the professional data processing community is particularly amenable to software produced as a result of published standards since it results in compatibility of view between different computers and operating systems. Conversely, such software tends to have a long gestation period because of its formulation in committee and subsequent consideration by interested parties. Once again this can be contrasted with the type of software which might be produced as a result of research into some area of devolved data bases where standardisation might well be of little value especially if the end product could be used by end users with the minimum of training.

Appendix 1 - The Structure of the EMAS Implementation of A Data Base Management System.

The implementation of a Data Base Management System (DBMS) on the EMAS multi-access system is based on the CODASYL proposals. In its basic form it is a teaching aid and all actions carried out on the data base are described to the user as they are performed.

In its second, more realistic, form the interactive user may initiate runs of application programs and restructuring tasks and the system will respond by indicating how much computer resources these operations have consumed.

Storage of Data

The data base itself is held on a FORTRAN direct access file such that each record is addressed by its relative record number within the file. Each record contains the following sections:

(a) The Record Type. The 4-character mnemonic identifying the record as an occurrence of a particular type of record is held.

(b) The Record Version. Similarly the version number of each occurrence is held on record.

(c) The Data. All data is held in character format for ease of reference. To allow numeric data items to be subjected to arithmetic these are converted to binary when held in an application program User Work Area.

(d) The Pointers. Pointers are held which associate the record occurrence with other record occurrences in the same set occurrence. A number of pointers are held since the record may belong to more than one set occurrence. The value held in each pointer is the address of the associated record.

(e) The Parallel Record Tag. If parallel running is taking place this further pointer is used to associate a record occurrence with the address of that same occurrence at another version.

The Index Files.

A number of FORTRAN direct access files are maintained as Index Files for the prime data file. This approach is necessary to overcome the absence of file indexing facilities in FORTRAN.

The FORTRAN File number is used to indicate the type of record to which the index relates and also the version number of the records covered by that index. In a file number of ab the first digit (a) indicates the record type and the second digit (b) the version number. By convention Customer records are indexed by files 11, 12, 13 etc, Account records by files 21, 22, 23 etc and Branch records by files 31, 32, 33 etc.

Each Index File contains an entry for any record held on the prime data file (FORTRAN File 1) with the corresponding record type and version number. The only data on that record will be the address of the appropriate data record. Any entries with a value of zero and any entries absent from the file will imply that there is no record in the prime data file for that version of that record. Thus a value of 25 for the 3rd record on file 11 will indicate that the 25th record on the prime data file will contain a Customer record at version 1 for Customer Number 3.

Thus in a parallel running situation two indices for the same record type may indicate that the same prime record occurrence exists but these will actually be at different addresses since they have different versions.

Note that indexing in this way is limited to one data item per record type. If further data items are included in a key then synonyms must be distinguished by successive examination.

The Schema

The description of the data base is held as an array in the COMMON area between the various Data Manipulation Language Execution Routines.

The following information is held:-

(i) There are entries for each type of record (each having up to 10 possible current version numbers). Records are identified by a 4-character record name.

(ii) Each version of each record can have up to 10 data items each identified by a 4-character data item name. The schema holds the start and end position of the data item and its FORTRAN format.

(iii) The record may have up to 10 pointers corresponding to its set membership.. Owner (First and Last) and Member (Next, Prior and Owner) Pointers may be used.

(iv) Up to three concatenated data items may be used as the record key.

(v) Details of the sets forming the data base are also held in array. For each set the set name is held with the name of the owner record plus one of its data items which is to be used for Set Occurrence Selection. Similarly, the names of up to 4 member records are held with corresponding data item names for Set Occurrence Selection (i.e set occurrence selection is based on equality of value between the data item quoted in the member record with that quoted in the owner record). Finally, an indication of whether new member record occurrences are to be added first or last in the set occurrence is held.

Special Registers

The following 'Special Registers' are variables held in the program COMMON area which may be referenced by application programs. Under no circumstances should their value be modified by a program - this may only be done by the DML subroutines. All Special Registers are integer fullwords.

ICURTP - This is the type of record which is Current of Run Unit. The record name is held as four alpha-numeric characters in FORTRAN format A4.

ICURNT - This is the address of the record which is Current of Run Unit. It is an integer held in a binary fullword.

ICURVR - This is the version number of the record which is Current of Run Unit. It too is an integer held in a binary fullword.

The User Work Area

Data is transferred between application programs and Data Base via a location known as the User Work Area. The use of the area differs between DML commands and is described under each command. In all cases, the area is an integer 1-dimensional array of 10 entries held in the FORTRAN COMMON area, viz. INTEGER IUWA(10).

The Sub-Schemas

Certain DML commands reference a Sub-schema for the data base being accessed. The implementation does not require a sub-schema for all commands and where it is required only a list of data item names is necessary. This is provided in an integer 1-dimensional array of 10 elements with each containing the name of a data item as four alpha-numeric characters in FORTRAN format A4. The array is passed as a parameter to the relevant routine as described below,

```
viz. INTEGER ISUBSC(10)/'ANCO','CUNO','BRNO','ACTP','BALC'/
```

The Data Manipulation Language Commands

The CODASYL DDLC Proposals do not now extend to details of the various DML commands but the CODASYL COBOL 1978 Journal of Development contains specifications for COBOL DML commands and a COBOL-format Sub Schema. The commands used for the implementation generally follow this specification although they use a FORTRAN format.

Calling Sequence

DML Commands are executed by calling a FORTRAN subroutine for the desired command. The subroutine call contains a parameter list which is specific to each command.

Thus the following sequence of commands would be used to find a record

```
CALL FIND1(IREC,IKEY1,IKEY2,IKEY3,IERR)
```

In this case the command is FIND1 where the parameters

IREC, IKEY1, IKEY2, IKEY3 and IERR are described in detail below.

The FIND Command

FIND is used by application programs to navigate through the data base. There are two formats of the command to correspond to different methods of access to the data base records - by record key and by set relationship.

FIND does not transfer data from data base to application program but rather it establishes a record as the Current of Run Unit and the application program can then perform various actions on it (e.g. delete it, get data items from it).

FIND of the First Type - Using Key Data Items

The first format of Find allows the application program to provide values for key data items of a specified type of record and the DBMS will establish the currency of the corresponding record.

If more than one occurrence is present with the same key data item values the first such occurrence will become the current record. If no occurrence is present with these values of key data items an error will be indicated to the application program.

The DBMS executes the command by accessing the index file corresponding to the most recent version of the record using the first key data item. In a full implementation the index would be based on all key data items concatenated together but this has not been feasible within the limits of FORTRAN file accessing facilities. Nevertheless the system will be able to determine whether an occurrence of the required record with the required key value actually exists for that version. If no entry exists on the index for the most recent version the system will then access the index for the previous version. This process will continue until all indices for the record have been examined and only if no entry has been found in any, will the system return an error code to the application program indicating that the required record does not exist.

Another limitation in the above technique for record indexing is that only a nominated data item may act as key and it must be numeric since it will be the relative record number on the search of the index file. The basic interactive version of the system therefore supports a simplistic technique for retrieval of records based on keys - a sequential search through the prime data file. All 3 key data items are used in this search and, by successive examination in decreasing order of version number, different key data items may be used as the key depending on the version number.

The Parameter List to the Routine is as follows:-

Parameter 1 - The Record Name - This is a integer variable containing the name of the record to be found as four alphanumeric characters.

Parameter 2 - The First Key Data Item - This is an integer variable containing the value for the first key data item. Values for Numeric Data Items must be supplied in the normal FORTRAN binary format and values for Alphanumeric Data Items must be supplied as left-justified in character format.

Parameter 3 - The Second Key Data Item - This is an integer variable as for the First Key Data Item. If only one key data item applies to the record being accessed this parameter should be set to spaces.

Parameter 4 - The Third Key Data Item - This too is an integer variable. It has a value of spaces if no third key data item is applicable to the record to be found.

Parameter 5 - The Error Code - This is an integer variable. On return from the routine it contains a binary number indicating whether the command has been successful. A value of zero is used for success and one for failure. In particular, if no occurrences of the record exist on the data base with a value of the key data items equal to that supplied, a value of 1 will be returned.

For example, the following sequence of instructions would be used to FIND a record type CUST with two key data items; the customer number of 25 and the first for characters of the name equal to CARD.

```
INTEGER KCUST/'CUST'/,KCARD/'CARD'/,ISPACE/' '/
I = 25
CALL FIND1(KCUST,KCARD,I,ISPACE,IERR)
IF (IERR.NE.0) GO TO 99
```

__FIND of the Second Type - Using Set Relationships__

Given that the existing Current Record of Run Unit is the Owner or Member of the set quoted on the command, this format of FIND allows the program to navigate to the first or last member record (from the owner record), to the next or prior member records (from a member record) or to the Owner record (again, from a member record).

Since the DBMS has the current record of run unit, it can examine the set pointers associated with that record. By further examination of the schema entry for that version of the record it can determine which pointer to use to provide the address of the desired record.

The Parameter List to the Routine is as follows:-

Parameter 1 - The Record Name - This is an integer variable containing the name of the record to be found as four alphanumeric characters.

Parameter 2 - The Set Name - This too is an integer variable containing the name of the set along which navigation is to take place. Again, this is held as four alphanumeric characters.

Parameter 3 - Type of Navigation - This is an integer variable containing a mnemonic of four alphanumeric characters indicating which type of navigation is required. The following types are possible:

FRST - The current record of run unit must be the owner of the set an the found record will be the first member record occurrence in the corresponding set OCCURRENCE.

LAST - The current record of run unit must be the owner of the set and the found record will be the last member record occurrence in the corresponding set occurrence.

NEXT - The current record of run unit must be a member of the set and the found record will be the next member record occurrence in the same set occurrence.

PRIR - The current record of run unit must be a member of the set and the found record will be the prior member record occurrence in the same set occurrence.

OWNR - The current record of run unit must be a member of the set and the found record will be the owner record occurrence of that set occurrence.

Parameter 4 - The Error Code - This is an integer variable. On return from the routine it contains a binary number indicating whether the command has been successful. A value of zero is used for success and one for failure. In particular, if navigation of FRST or LAST is used and there are no member record occurrences in that set occurrence a value of one will be returned. If navigation of NEXT is used and the current record of run unit is the last in that set occurrence a value of one is returned. Similarly, if navigation of PRIR is used and the current record of run unit is the first in that set occurrence a value of one is returned.

For example if the current record of run unit is a Customer Record the following sequence of instructions would be used to find the first account record in the set of customers accounts for that customer.

```
INTEGER KACNT/'ANCT'/,KCUAC/'CUAC'/,KFRST/'FRST'/
CALL FIND2(KACNT,KCUAC,KFRST,IERR)
IF (IERR.NE.0) GO TO 99
```

<u>The DELETE Command</u>

Record occurrences may be removed from the data base by using the DELETE command. Before this can be done the record must be removed from all sets of which it is a member by using the REMOVE command (see below) The record to be deleted is the Current of Run Unit. Any index entry for the record in question will also be deleted.

The Parameter List for the Routine is as follows:-

Parameter 1 - The Record Name - This is an integer variable containing the name of the record to be deleted as four alphanumeric characters. A record of this type must be the Current of Run Unit.

Thus the following sequence of instructions would be used to delete an account record.

```
 INTEGER KACNT/'ACNT'/
 CALL DELETE (KACNT)
```

The GET Command

The GET Command transfers the value of a single data item in the Current Record of Run Unit from the Data Base to the User Work Area of the Application Program.

Knowing the version number of the current record of run unit (this having been established by FIND, etc.) the routine can access the appropriate entry in the schema for that version of the record. From this the position and format of the data item can be established and the DBMS can therefore carry out any necessary transformation of format to provide the data item in the user work area.

The Parameter List for the Routine is as follows:-

Parameter 1 - The Data Item Name - This is an integer variable containing the name of the data item to be retrieved from the data base as four alphanumeric characters. This data item must be defined in the schema entry for the current version of the Current Record of Run Unit.

The value of the retrieved data item is stored in the first element of the User Work Area Array (IUWA(1)). Alphanumeric Data Items are provided left justified in the fullword and Numeric Data Items are provided in FORTRAN Binary Format.

The following sequence of instructions would be used to GET the Balance from the Account record assuming it has already been established as Current of Run Unit:-

```
INTEGER KBALC/'BALC'/
CALL GET(KBALC)
IBAL = IBAL + IUWA(1)
```

<u>The STORE Command</u>

The STORE command is used to add a new record occurrence to the Data Base. The value of the data items in the record are provided by the application program in the User Work Area.

The routine will examine the schema entry for the record in question. If more than one version of the record is present it will use the description of the highest version number (i.e. that most recently defined) to establish the position and format of the data items. A comparison is made between the list of data items in the sub-schema and that in the schema and any not included in the sub-schema definition will be set to a default value of spaces. At this point the record does not belong to any set since the INSERT command must be used to establish set membership and all set pointers are, therefore, set to zero..

The system maintains a record of the next available position within the prime data file (this is held in the first record of the file which does not, in fact, hold data). When this free 'slot' is allocated by the STORE this 'Free Space Pool' indicator is updated to point to the following record. Note that the system does not re-utilise records freed as a result of DELETE commands. Since the record is stored at the most recent version the appropriate index entry is updated to point to the new record.

The Parameter List for the Routine is as follows:-

Parameter 1 - The Record Name - This is an integer variable containing the name of the record to be added to the data base as four alphanumeric characters.

Parameter 2 - The Sub Schema Name - This is an integer array of 10 elements with each element containing the name of a data item to be stored in the new record occurrence. Fewer than 10 data items may be specified in which case the rightmost elements are set to spaces.

The User Work Area must contain the values for each data item quoted in the sub schema with each entry corresponding element for element in each array. Alphanumeric Data Items must be supplied as left justified in the fullword and Numeric Data Items must be supplied in FORTRAN Binary Format.

The following sequence of instructions would be used to STORE a new Account Record (Number 07) for Customer 1, Branch 3, Account Type 'CA' and Balance of Zero:-

```
INTEGER KSUBSC/'BRNO','ACNO','ACTP','BALC',5*' '/
INTEGER KACNT/'ACNT'/,KCA/'CA'/
IUWA(1)=3
IUWA(2)=7
IUWA(3)=1
IUWA(4)=KCA
IUWA(5)=0
CALL STORE (KACNT,KSUBSC)
```

The MODIFY Command

The MODIFY Command is similar to STORE in that it results in a record being written to the data base but in this case the new record supersedes an existing record occurrence - the existing current record of run unit.

The record is modified by having one of its data items altered in value.

Where a record is to be modified and the version number of the existing current record of run unit is not the most recently defined version of that record the record occurrence is restructured to this latest version before it is modified. This allows the data base to support open restructurings by performing the restructuring in primary storage when a record has been retrieved and before it is written back to secondary storage by the modify thereby incurring no additional Input/Output operation.

In normal circumstances the Index Entry for the record being modified will not change (assuming that the value of the key data item has not altered) but where an open restructuring has taken place such that the version number of the record has altered then the index entry for the record must be transferred to the new index file for that new version. If parallel running is taking place the new version of the record will be added to the data base at the next available 'slot' as described previously for STORE and the existing version will also be retained in its original position. Furthermore the entry for the new version will be included in the appropriate index file and that for the original version will be retained in its index file .

The Parameter List for this routine is as follows:-

Parameter 1 - The Data Item Name - This is an integer variable containing the name of the data item to be modified as four alphanumeric characters. The data item must be defined in the schema entry for the current version of the Current Record of Run Unit.

The new value to which the data item is to be modified is supplied by the application program as the first element in the User Work Area Array (IUWA(1)). Alphanumeric Data Items are supplied left justified in the fullword and Numeric Data Items are provided in FORTRAN Binary Format.

The following sequence of instructions would be used to MODIFY the Balance of an Account record assuming that it has already been established as Current of Run Unit:-

```
INTEGER KBALC/'BALC'/
IUWA(1)  =  100
CALL MODIFY (KBALC)
```

The INSERT Command

INSERT is used to add the current record of run unit into a set of which it is a member. The actual occurrence of the set to receive the record and the point at which this record is to be added within the set occurrence is determined by the data supplied for that set in the schema.

The routine establishes the value of the data item in the Current of Run Unit record to be used for matching with a corresponding data item in the owner record in much the same way as an application program would get one of the data items in that record.

When this value has been established it is used by the routine in much the same way as an application program would use FIND1 to determine the address and version of the corresponding owner record. A further record may also be required (e.g. if a new 'last' member is being inserted the previous 'last' member is required) and the DBMS will alter the pointers of all records to reflect the new member. In doing so, the routine uses the insertion rule specified in the schema for the position of insertion of records into the set. In this implementation only positions of first or last in the member record occurrences for the set occurrence are permissible but in a full implementation indices would be required for each set occurrence to allow records to be inserted based on key data item at a particular point within the set occurrence. These indices would be distinct from those which have been established to reference records based on the values of key data items.

Parameter 1 - The Set Name - This is an integer variable containing the name of the set into which the Current Record of Run Unit is to be inserted as four alphanumeric characters. The set must be defined on the Schema with the record type of the Current Record of Run Unit as a member.

The following sequence of instructions would be used to INSERT the Current Record of Run Unit (An Account Record) into the Customer's Accounts Sets.

```
INTEGER KCUAC'/CUAC'/
CALL INSERT (KCUAC)
```

Note that all record occurrences must be added to sets using INSERT. The implementation does not support automatic set insertion.

The REMOVE Command

REMOVE serves the opposite function to INSERT. It allows an application program to end membership of a set occurrence for the current record of run unit.

The routine firstly examines the pointers for the set in question from the current record of run unit. From these the address of the owner record and adjacent member records can be determined by examination of the entry for that version of the current record on the schema. The pointers on the current record, adjacent member records and the owner record are then altered to set up a chain of pointers which no longer includes the current record.

The Parameter List for the routine is as follows:-

Parameter 1 - The Set Name - This is an integer variable containing the name of the set from which the Current Record of Run Unit is to be removed as four alphanumeric characters. The set must be defined on the Schema with the record type of Current Record of Run Unit as a member.

The following sequence of instructions would be used to REMOVE the Current Record of Run Unit (An Account Record) from the Customer's Accounts Set.

```
INTEGER KCUAN/'CUAC'/
```

```
CALL REMOVE (KCUAC)
```

Appendix 2

The Bank Data Base and Application Programs

The Account is the basic unit of operation of the bank and a data base record is maintained for each bank account. An account is identified by its account number and is held at a specific branch of the bank and belongs to a particular customer. Any customer may hold any number of accounts at one or more branches (indeed this is common practice for business customers).

The following data is, therefore, held for each account:-

Account Number
Branch Number
Customer Number
Account Type (Current, Deposit, Loan, etc.)
Balance

A data base record is also maintained for each branch of the bank containing the following data:-

Branch Number
Branch Name
Designation Codes (Urban, Rural, East, West, etc.)

Further, a record is maintained for each customer of the bank containing the following items of data:-

Customer Number
Customer Name

The inter-relationship between Customer and Account

Records is represented by the following set:-

Customer's Accounts - Owner Customer

Member Account.

The Initial Data Base and Schema

The Schema for this data base is shown in Figure 1.

Figure 2 shows the initial contents of the data base as follows:

> Customer Number 01 is A. JONES who holds Account Number 01 at Branch 01 which is a Current Account (CA) with a Balance of £11. He also holds Account 04 at Branch 01 which is also a Current Account with a balance of £44.
>
> Customer Number 02 is J. SMITH who holds Account Number 02 at Branch 01 which is a Current Account with a balance of £22. He also holds Account 03 at Branch 02 which is a Deposit Account (DA) with a balance of £33.
>
> There are also two branch records on the Data Base. Branch 01 is BIGTOWN Branch. Branch 02 is SMALLTOWN Branch.

Data Base Programs

The Bank has a number of application programs operating on the data base as described below.

Runs of the programs are initiated by the user keying an appropriate two-digit code on his interactive terminal. The user may then also key a further single digit code indicating the version of the program he wishes to run. This is a simplified way of demonstrating that various versions will be required at various points during different restructurings. Details of when the different versions will be required are given under the individual restructuring tasks in Appendix 3.

Figure 1

```
Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM ERNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE FRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM ENM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM ENM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM ENM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMEER RECCRD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST
```

Figure 2

```
Data:30

*** THE CURRENT DATA EASE ***

ADD RECORD VRSN DATA                               POINTERS              TAG
--- ------ ---- ---------------------------------  -------------------   ---
 1   CUST    1   1JONES,ALAN                       2 6 0 0 0 0 0 0 0 0    0
 2   ACNT    1   1 1 1    11CA                     6 0 1 0 0 0 0 0 0 0    0
 3   CUST    1   2SMITH,JAMES                      4 5 0 0 0 0 0 0 0 0    0
 4   ACNT    1   2 1 2    22CA                     5 0 3 0 0 0 0 0 0 0    0
 5   ACNT    1   3 2 2    33CA                     0 4 3 0 0 0 0 0 0 0    0
 6   ACNT    1   4 1 1    44CA                     0 2 1 0 0 0 0 0 0 0    0
 7   BRCH    1   1BIGTOWN      U                   0 0 0 0 0 0 0 0 0 0    0
 8   BRCH    1   2SMALLTOWN    R                   0 0 0 0 0 0 0 0 0 0    0
```

The 'Balance Calculation' Program

This program calculates the net balance for any specified customer by navigating through his account records.

The run is initiated by the user entering an input type of 01 on his interactive terminal. The program requests the Customer Number for which the balance is required and the user types this through his terminal. The customer number is then used by the program as a parameter to a FIND command of the first type (i.e. Find a record given the value of a key data item). This establishes the correct customer record as current of run unit. GET is then used to retrieve the customer's name. A loop of instructions is then carried out to FIND the 'First' and subsequent Account records in the 'Customer's Accounts' set using a FIND command of the second type (i.e. Navigate through a set based on the Current of Run Unit which is Owner or Member of that set).

For each Account record found the command GET is issued by the program to transfer the contents of the Balance data item to the program User Work Area. When all Account records in the set have been found the program prints the sum of the account balances on the user's terminal.

Figure 3 is a Listing of the Program
Figure 4 shows a run of the program to calculate the net balance for Customer 01.

Figure 3

```
C
C
C
C
   10 WRITE(6,1320)
      WRITE(6,2011)
      READ(5,2015)IPVER
      IF(IPVER.EQ.1)GOTO 5010
      IF(IPVER.EQ.2)GOTO 5020
 1320 FORMAT ('0***RUN OF BALANCE CALCULATION PROGRAM***')
 5010 WRITE (6,1005)
C  INPUT TYPE 01 REQUIRES EXECUTION OF THE CALCULATE BALANCE PROGRAM
C  USING A SPECIFIED CUSTOMER NUMBER
      READ (5,1002) ICUST
 1002 FORMAT (I2)
 1005 FORMAT (' TYPE CUSTOMER NUMBER')
      IBAL=0
C
C DML COMMAND FIND1 IS USED TO ESTABLISH A CURRENT RECORD OF RUN
C UNIT FOR A CUSTOMER RECORD WITH THE SUPPLIED CUSTOMER NUMBER
C
      CALL FIND1(KCUST,ICUST,ISPACE,ISPACE,IERR)
   11 IF (IERR.NE.1) GO TO 17
      WRITE(6,1007)ICUST
 1007 FORMAT (' CUSTOMER ',I2,' NOT FOUND ')
      GO TO 16
   17 CALL GET(KNAM1)
 8001 IWN1=IUWA(1)
      CALL GET(KNAM2)
 8002 IWN2=IUWA(1)
      CALL GET(KNAM3)
 8003 IWN3=IUWA(1)
      CALL FIND2(KACNT,KCUAC,KFRST,IERR)
   12 IF (IERR.EQ.1) GO TO 15
      CALL GET(KBALC)
   14 IBAL=IBAL+IUWA(1)
      CALL FIND2 (KACNT,KCUAC,KNEXT,IERR)
   15 IF (IERR.NE.1) GO TO 12
      WRITE (6,1040)ICUST,IWN1,IWN2,IWN3,IBAL
 1040 FORMAT (' BALANCE FOR CUSTOMER ',I2,' ',3A4,' IS ',I4)
   16 WRITE(6,1321)
 1321 FORMAT(' ***END OF BALANCE CALCULATION RUN***')
      GO TO 5
 5020 WRITE(6,1005)
      READ (5,1002) ICUST
      IBAL=0
      CALL FIND1 (KCUST,ICUST,ISPACE,ISPACE,IERR)
 5021 IF (IERR.NE.1) GO TO 5027
      WRITE (6,1007) ICUST
      GO TO 5026
 5027 CALL GET (KNAM1)
 5001 IWN1=IUWA(1)
      CALL GET(KNAM2)
 5002 IWN2=IUWA(1)
      CALL GET(KNAM3)
 5003 IWN3=IUWA(1)
      IF (ICURVR.EQ.1) GO TO 5004
      CALL GET(KBALC)
 5005 IBAL=IUWA(1)
      GO TO 5006
 5004 CALL FIND2(KACNT,KCUAC,KFRST,IERR)
 5022 IF (IERR.EQ.1)GO TO 5025
      CALL GET(KBALC)
 5024 IBAL=IBAL+IUWA(1)
      CALL FIND2 (KACNT,KCUAC,KNEXT,IERR)
 5025 IF (IERR.NE.1) GO TO 5022
 5006 WRITE (6,1040) ICUST,IWN1,IWN2,IWN3,IBAL
 5026 WRITE (6,1321)
```

Figure 4

```
Data:01

***RUN OF BALANCE CALCULATION PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE CUSTOMER NUMBER
Data:01

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  1 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  6
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1 ARE:-
ONE
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 10
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1 ARE:-
,AL
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS 11 TO 14
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1 ARE:-
N
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1
IS  2
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  2 ARE:-
  11
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  2
IS  6
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  6 ARE:-
  44
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  6
IS  0
***'FIND2' COMPLETED***
BALANCE FOR CUSTOMER  1 JONES,ALAN   IS   55
***END OF BALANCE CALCULATION RUN***
```

The 'Transaction Posting' Program

This program allows the balance of an account to be modified by the value of a Debit or Credit transaction which has been performed on the account (i.e. the transaction is posted to the account). In practice, such transactions could be obtained from various sources such as branch terminals, cash dispensing machines and cheques remitted by other banks.

The program requests the Account Number and Value to be Posted and the user enters these on his terminal. A FIND command of the first type is then used to establish the correct record as current of run unit followed by a GET to transfer the Balance to the User Work Area. The value of the transaction is then added to the Balance (still in the User Work Area) and a MODIFY is then carried out to update the account record with the correct balance. Before the run terminates, the new balance is printed on the user's terminal.

Figure 5 is a Listing of the program.
Figure 6 shows a run of the program to post a Credit of £10 to Account 02.

Figure 5

```
C
C
C
C
C
   20 WRITE(6,1322)
 1322 FORMAT ('0***RUN OF TRANSACTION POSTING PROGRAM***')
      WRITE(6,2011)
      READ(5,2015)IPVER
      IF(IPVER.EQ.1)GOTO 5030
      IF(IPVER.EQ.2)GOTO 5040
      IF(IPVER.EQ.3)GOTO 5050
      IF(IPVER.EQ.4)GOTO 5060
      STOP 77
 5030 WRITE (6,1042)
 1042 FORMAT (' TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED')
      READ (5,1045) IACCNO,IVAL
 1045 FORMAT (I2,I5)
      CALL FIND1(KACNT,IACCNO,ISPACE,ISPACE,IERR)
   21 IF(IERR.NE.1) GO TO 27
      WRITE(6,1046)IACCNO
 1046 FORMAT(' ACCOUNT ',I2,' NOT FOUND ')
      GO TO 28
C
C  FIND1 ESTABLISHES THE ACCOUNT SPECIFIED AS THE
C  CURRENT OF RUN UNIT
C
   27 CALL GET (KBALC)
   24 IUWA(1)=IUWA(1)+IVAL
      CALL MODIFY(KBALC)
   26 WRITE (6,1050) IVAL,IUWA(1)
 1050 FORMAT (' TRANSACTION OF $',I5,' POSTED - NEW BALANCE $',I5)
   28 WRITE(6,1323)
 1323 FORMAT (' ***END OF TRANSACTION POSTING RUN***')
      GO TO 5
      INTEGER KBRCH/'BRCH'/,KBNM1/'BNM1'/,KBNM2/'BNM2'/,KBNM3/'BNM3'/
      INTEGER KBRAC/'BRAC'/
 5040 WRITE (6,1042)
      READ (5,1045) IACCNO,IVAL
      CALL FIND1(KACNT,IACCNO,ISPACE,ISPACE,IERR)
 5041 IF (IERR.NE.1) GO TO 5047
      WRITE (6,1046) IACCNO
      GO TO 5048
 5047 CALL GET (KBALC)
 5044 IUWA(1)=IUWA(1)+IVAL
      CALL MODIFY(KBALC)
 5046 IF (ICURVR.EQ.1) GO TO 5049
      WRITE (6,1051) IVAL,IUWA(1)
 1051  FORMAT (' TRANSACTION OF $',I5,' POSTED - NEW BALANCE $',I6)
      GO TO 5048
 5049 WRITE (6,1050) IVAL,IUWA(1)
 5048 WRITE (6,1323)
      GO TO 5
```

Figure 5 (Cont.)

```
 5050 WRITE (6,1042)
      READ (5,1045) IACCNO,IVAL
      CALL FIND1 (KACNT,IACCNO,ISPACE,ISPACE,IERR)
 5051 IF (IERR.NE.1) GO TO 5057
      WRITE (6,1046) IACCNO
      GO TO 5058
 5057 IF (ICURVR.EQ.1) GO TO 5555
      CALL FIND3(IERR)
      IF(IERR.NE.1) GO TO 7001
      WRITE(6,7002)
 7002 FORMAT (' NO VERSION 1 ACCOUNT RECORD FOUND ')
      GO TO 5556
 7001 CALL GET(KBALC)
      IUWA(1)=IUWA(1)+IVAL
      CALL MODIFY(KBALC)
      WRITE(6,7003)IVAL,IUWA(1)
 7003 FORMAT(' TRANSACTION OF $',I5,' POSTED'
     *,' - NEW ACCOUNT BALANCE $',I6)
 5556 CALL FIND2 (KCUST,KCUAC,KOWNR,IERR)
 5052 IF (IERR.NE.1) GO TO 5055
      WRITE (6,1047)
 1047 FORMAT (' CORRESPONDING CUSTOMER RECORD NOT FOUND')
      GO TO 5058
 5055 CALL GET (KBALC)
 5054 IUWA(1)=IUWA(1)+IVAL
      CALL MODIFY (KBALC)
 5056 WRITE (6,7004) IVAL,IUWA(1)
 7004 FORMAT(' TRANSACTION OF $',I5,' POSTED'
     1*,' - NEW CUSTOMER BALANCE $',I5)
 5058 WRITE (6,1323)
      GO TO 5
 5555 CALL GET(KBALC)
      IUWA(1)=IUWA(1)+IVAL
      CALL MODIFY(KBALC)
      WRITE (6,7003) IVAL,IUWA(1)
      GO TO 5058
 5060 WRITE (6,1147)
 1147 FORMAT (' TYPE ACCOUNT NUMBER, ACCOUNT TYPE AND VALUE TO BE POSTED
     *')
      READ (5,1048) IACCNO,IACCTP,IVAL
 1048 FORMAT (I2,1X,A2,1X,I5)
      CALL FIND1 (KACNT,IACCNO,IACCTP,ISPACE,IERR)
 5061  IF (IERR.NE.1) GO TO 5067
      WRITE (6,1049) IACCNO,IACCTP
 1049 FORMAT (' ACCOUNT NUMBER ',I2,' TYPE ',A2,' NOT FOUND')
      GO TO 5068
 5067 CALL GET(KBALC)
 5064 IUWA(1)=IUWA(1)+IVAL
      CALL MODIFY (KBALC)
 5066 WRITE(6,1050) IVAL,IUWA(1)
 5068 WRITE(6,1323)
      GO TO 5
C
```

Figure 6.

```
Data:02

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:02 0010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
RECORD FOUND AT ADDRESS  4 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4 ARE:-
  22
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4
HAVE BEEN ALTERED TO
   32
***'MODIFY' COMPLETED***
TRANSACTION OF $    10 POSTED - NEW BALANCE $    32
***END OF TRANSACTION POSTING RUN***



Data:30

*** THE CURRENT DATA EASE ***

ADD RECORD VRSN DATA                                    POINTERS
--- ------ ---- ------------------------------          -------------------
 1   CUST    1   1JONES,ALAN                             2 6 0 0 0 0 0 0 0 0
 2   ACNT    1   1 1 1    11CA                           6 0 1 0 0 0 0 0 0 0
 3   CUST    1   2SMITH,JAMES                            4 5 0 0 0 0 0 0 0 0
 4   ACNT    1   2 1 2    32CA                           5 0 3 0 0 0 0 0 0 0
 5   ACNT    1   3 2 2    33CA                           0 4 3 0 0 0 0 0 0 0
 6   ACNT    1   4 1 1    44CA                           0 2 1 0 0 0 0 0 0 0
 7   BRCH    1   1BIGTOWN      U                         0 0 0 0 0 0 0 0 0 0
 8   BRCH    1   2SMALLTOWN    R                         0 0 0 0 0 0 0 0 0 0
```

The 'Open New Account' Program

An existing customer may open any number of new accounts at any of the bank branches. This program sets up a new account record for the account with an initial balance of zero. It also adds the new account record to the set of such records for that customer.

The run is initiated by the user entering an input type of 03 on his interactive terminal. The program requests the Branch, Account Number, Customer Number and Account Type for the new account and these are read from the user's terminal to the User Work Area where (together with a value of zero for the balance) they are used with the STORE command to add the new record to the data base. The INSERT command is then used to add the new record to the Customer's Accounts Set. Before the run terminates a confirmatory message is printed on the user's terminal.

Figure 7 is a listing of the program.
Figure 8 shows a run of the program to open a new account number 5 at branch 02 for Customer 01.

Figure 7

```
C
C
C
C
C
   30 WRITE(6,1324)
 1324 FORMAT('0***RUN OF OPEN NEW ACCOUNT PROGRAM***')
      WRITE(6,2011)
      READ(5,2015) IPVER
      IF (IPVER.EQ.1) GO TO 5070
      IF(IPVER.EQ.2) GO TO 5080
      STOP 77
 5070 WRITE (6,1070)
 1070 FORMAT (' TYPE BRANCH , ACCOUNT NUMBER , CUSTOMER , ACCOUNT TYPE')
      READ (5,1075) IUWA(1),IUWA(2),IUWA(3),IUWA(4)
 1075 FORMAT (I2,1X,I2,1X,I2,1X,A2)
      IUWA(5)=0
      CALL STORE (KACNT,KSUBSC)
      CALL INSRT (KCUAC)
   36 WRITE (6,1115)
 1115 FORMAT (' NEW ACCOUNT OPENED')
      WRITE(6,1325)
 1325 FORMAT('***END OF OPEN NEW ACCOUNT RUN***')
      GO TO 5
 5080 WRITE (6,1070)
      READ (5,1075) IUWA(1),IUWA(2),IUWA(3),IUWA(4)
      IUWA(5)=0
      CALL STORE (KACNT,KSUBSC)
      CALL INSRT (KCUAC)
      CALL INSRT (KBRAC)
 5087 WRITE (6,1115)
      WRITE (6,1325)
      GO TO 5
C
C
```

Figure 8

```
Data:03

***RUN OF OPEN NEW ACCOUNT PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE BRANCH , ACCOUNT NUMBER , CUSTOMER , ACCOUNT TYPE
Data:02 05 01 CA

***FOLLOWING INFORMATION IS FROM 'STORE'***
THE FOLLOWING NEW RECORD
IS BEING ADDED TO THE DATA BASE

RECORD TYPE ACNT
VERSION   1
ALL POINTERS ZERO
UNINITIALISED CHARACTERS SPACES
CHARACTERS   3 TO   4 SET TO :-
 2
CHARACTERS   1 TO   2 SET TO :-
 5
CHARACTERS   5 TO   6 SET TO :-
 1
CHARACTERS 12 TO 13 SET TO :-
CA
CHARACTERS   7 TO 11 SET TO :-
     0
RECORD HAS BEEN STORED AT ADDRESS   9
***'STORE' COMPLETED***

*** FOLLOWING INFORMATION IS FROM 'INSERT'***
CHARACTERS   5 TO   6
OF RECORD TYPE ACNT VERSION   1 AT ADDRESS   9 ARE :-
 1
DBMS IS SEARCHING FOR RECORD TYPE CUST VERSION   1
WITH CHARACTERS   1 TO   2 EQUAL TO :-
 1
RECORD FOUND AT ADDRESS   1
RECORD-TYPE AND VERSION OF RECORD AT ADDRESS   6 ARE :-
ACNT   1
POINTER   1 OF RECORD TYPE ACNT ADDRESS   6 VERSION   1
HAS BEEN ALTERED TO   9
POINTER   2 OF RECORD TYPE ACNT ADDRESS   9 VERSION   1
HAS BEEN ALTERED TO   6
POINTER   3 OF RECORD TYPE ACNT ADDRESS   9 VERSION   1
HAS BEEN ALTERED TO   1
POINTER   2 OF RECORD TYPE CUST ADDRESS   1 VERSION   1
HAS BEEN ALTERED TO   9
***'INSERT' COMPLETED***
NEW ACCOUNT OPENED
**END OF OPEN NEW ACCOUNT RUN***



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                                         POINTERS                  TA
--- ------ ---- -------------------------------------------  ------------------------  --
 1   CUST    1    1JONES,ALAN                                2 9 0 0 0 0 0 0 0 0      0
 2   ACNT    1    1  1  1      11CA                          6 0 1 0 0 0 0 0 0 0      0
 3   CUST    1    2SMITH,JAMES                               4 5 0 0 0 0 0 0 0 0      0
 4   ACNT    1    2  1  2      32CA                          5 0 3 0 0 0 0 0 0 0      0
 5   ACNT    1    3  2  2      33CA                          0 4 3 0 0 0 0 0 0 0      0
 6   ACNT    1    4  1  1      44CA                          9 2 1 0 0 0 0 0 0 0      0
 7   BRCH    1    1BIGTOWN      U                            0 0 0 0 0 0 0 0 0 0      0
 8   BRCH    1    2SMALLTOWN   R                             0 0 0 0 0 0 0 0 0 0      0
 9   ACNT    1    5  2  1       0CA                          0 6 1 0 0 0 0 0 0 0      0
```

The 'Statistics Print' Program

This program provides statistics on the bank's customers.

The run is initiated by the user entering an input type of 04 on his interactive terminal.

The first type of FIND command is used to retrieve all Customer Records in turn from the data base. The GET command is used to obtain the Customer Number from each found record to allow the program to progress onto the next record. As each record is found, it is counted. When all records have been found the program prints the record count and terminates.

Figure 9 is a Listing of the program.

Figure 10 is a run of the program.

Figure 9

```
C
C
C
C
   40 WRITE (6,2010)
 2010 FORMAT('0***RUN OF STATISTICS PRINT PROGRAM***')
      WRITE(6,2011)
 2011 FORMAT(' TYPE VERSION NUMBER OF PROGRAM')
      READ(5,2015)IPVER
 2015 FORMAT(I1)
      IF (IPVER.EQ.1) GOTO 41
      IF(IPVER.EQ.2) GOTO 45
      STOP77
   41 INOCST=0
      ICUST=1
      CALL FIND1(KCUST,ICUST,ISPACE,ISPACE,IERR)
   42 IF(IERR.EQ.1) GO TO 44
      CALL GET(KCNUM)
   43 IF(IUWA(1).EQ.99) GO TO 44
      ICUST=IUWA(1)+1
      INOCST=INOCST+1
      CALL FIND1(KCUST,ICUST,ISPACE,ISPACE,IERR)
      GO TO 42
   44 WRITE(6,2012)INOCST
 2012 FORMAT(' NUMBER OF CUSTOMERS - ',I1)
      WRITE(6,1327)
 1327 FORMAT(' ***END OF STATISTICS PRINT RUN***')
      GO TO 5
   45 INOCST=0
      IAGES(1)=0
      IAGES(2)=0
      IAGES(3)=0
      IAGES(4)=0
      IAGES(5)=0
      ICUST=1
      CALL FIND1(KCUST,ICUST,ISPACE,ISPACE,IERR)
   46 IF(IERR.EQ.1) GO TO 48
      CALL GET(KCNUM)
   47 IF (IUWA(1).EQ.99) GO TO 48
      ICUST=IUWA(1)+1
      INOCST=INOCST+1
      IAGE=1
C
C
C  NOTE THAT RECORD VERSION IS IN SPECIAL REGISTER
C
C
      IF(ICURVR.EQ.1)GO TO 49
      CALL GET(KAGE)
 8040 IAGE=IUWA(1)
      IF (IAGE.EQ.0) IAGE=1
   49 IAGES(IAGE)=IAGES(IAGE)+1
      CALL FIND1(KCUST,ICUST,ISPACE,ISPACE,IERR)
      GO TO 46
   48 WRITE(6,2012)INOCST
      WRITE(6,2013)IAGES(1)
 2013 FORMAT(' NUMBER WITH UNKNOWN AGE GROUP - ',I2)
      WRITE(6,2014)I2,IAGES(2)
      WRITE(6,2014)I3,IAGES(3)
      WRITE(6,2014)I4,IAGES(4)
      WRITE(6,2014)I5,IAGES(5)
 2014 FORMAT(' NUMBER IN AGE GROUP ',I1,' - ',I2)
      WRITE(6,1327)
      GO TO 5
```

Figure 10

```
Data:04

***RUN OF STATISTICS PRINT PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  1 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  1 TO  2
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1 ARE:-
1
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
RECORD FOUND AT ADDRESS  3 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  1 TO  2
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3 ARE:-
2
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 3
NO RECORD FOUND
***'FIND1' COMPLETED***
NUMBER OF CUSTOMERS - 2
***END OF STATISTICS PRINT RUN***
```

The 'Add New Customer' Program

This program allows a new customer to be added to the data base. At this stage, the customer has no accounts but these can be added by running the 'Open New Account' Program. The program is initiated by the user entering 08 on his interactive terminal.

The user enters the customer number and name of the new customer on his terminal and the corresponding record with these data items is added to the data base using the STORE command. Before the run terminates a confirmatory message is printed.

Figure 11 is a Listing of the program.
Figure 12 shows the program being run to add a New Customer 03.

Figure 11

```
C
C
C
C
C
   80 WRITE (6,1360)
 1360 FORMAT ('0*** RUN OF THE ADD NEW CUSTOMER PROGRAM ***')
      WRITE (6,1361)
 1361 FORMAT (' TYPE VERSION NUMBER OF PROGRAM')
      READ (5,1362) IPVER
 1362 FORMAT (I1)
      IF (IPVER.EQ.1) GO TO 81
      IF (IPVER.EQ.2) GO TO 85
      STOP 80
   81 WRITE (6,1363)
 1363 FORMAT (' TYPE CUSTOMER NUMBER AND NAME')
      READ (5,1364) IUWA(1),IUWA(2),IUWA(3),IUWA(4)
 1364 FORMAT (I2,1X,3A4)
      CALL STORE(KCUST,KSUBS2)
   82 WRITE (6,1365)
 1365 FORMAT (' NEW CUSTOMER ADDED')
      WRITE (6,1366)
 1366 FORMAT (' *** END OF NEW CUSTOMER RUN ***')
      GO TO 5
   85 WRITE(6,1367)
 1367 FORMAT (' TYPE CUSTOMER NUMBER,NAME AND AGE GROUP')
      READ (5,1368) (IUWA(II),II=1,5)
      CALL STORE (KCUST,KSUBS3)
 1368 FORMAT (I2,1X,3A4,1X,I1)
   86 GO TO 82
C
C
```

Figure 12

```
Data:08

*** RUN OF THE ADD NEW CUSTOMER PROGRAM ***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE CUSTOMER NUMBER AND NAME
Data:03 GREEN,MARY

***FOLLOWING INFORMATION IS FROM 'STORE'***
THE FOLLOWING NEW RECORD
IS BEING ADDED TO THE DATA BASE

RECORD TYPE CUST
VERSION  1
ALL POINTERS ZERO
UNINITIALISED CHARACTERS SPACES
CHARACTERS  1 TO  2 SET TO :-
 3
CHARACTERS  3 TO  6 SET TO :-
GREE
CHARACTERS  7 TO 10 SET TO :-
N,MA
CHARACTERS 11 TO 14 SET TO :-
RY
RECORD HAS BEEN STORED AT ADDRESS 10
***'STORE' COMPLETED***
NEW CUSTOMER ADDED
*** END OF NEW CUSTOMER RUN ***



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                                  POINTERS                T
--- ------ ---- ------------------------------------ ------------------- -
  1   CUST    1   1JONES,ALAN                         2 9 0 0 0 0 0 0 0 0
  2   ACNT    1   1 1 1    11CA                       6 0 1 0 0 0 0 0 0 0
  3   CUST    1   2SMITH,JAMES                        4 5 0 0 0 0 0 0 0 0
  4   ACNT    1   2 1 2    32CA                       5 0 3 0 0 0 0 0 0 0
  5   ACNT    1   3 2 2    33DA                       0 4 3 0 0 0 0 0 0 0
  6   ACNT    1   4 1 1    44CA                       9 2 1 0 0 0 0 0 0 0
  7   BRCH    1   1BIGTOWN      U                     0 0 0 0 0 0 0 0 0 0
  8   BRCH    1   2SMALLTOWN   R                      0 0 0 0 0 0 0 0 0 0
  9   ACNT    1   5 2 1     0CA                       0 6 1 0 0 0 0 0 0 0
 10   CUST    1   3GREEN,MARY                         0 0 0 0 0 0 0 0 0 0
```

The 'Amend Customer Details' Program

This program allows the user to alter the details currently held on the customer record (i.e. the Customer Name) for a particular customer. The run is initiated by the user entering an input type of 09 on his interactive terminal.

The user enters the customer number and the amended name on his terminal. The program uses the first type of FIND to retrieve the appropriate record from the data base. It then moves the revised name to the User Work Area and issues a MODIFY Command to alter the data base record.

Figure 13 is a Listing of the program.
Figure 14 shows a run of the program to amend the name of Customer 02 from J. Smith to W. Brown.

Figure 13

```
C
C
C
C
C
   90 WRITE (6,1380)
 1380 FORMAT ('0*** RUN OF AMEND CUSTOMER DETAILS PROGRAM ',
     *'***')
      WRITE (6,1381)
 1381 FORMAT (' TYPE VERSION NUMBER OF PROGRAM ')
      READ (5,1382) IPVER
 1382 FORMAT (I1)
      IF (IPVER.EQ.1) GO TO 91
      IF (IPVER.EQ.2) GO TO 991
      STOP 90
   91 WRITE (6,1383)
 1383 FORMAT (' TYPE CUSTOMER NUMBER AND AMENDED NAME')
      READ (5,1384)ICUST,NAM1,NAM2,NAM3
 1384 FORMAT (I2,1X,3A4)
  992 CALL FIND1(KCUST,ICUST,ISPACE,ISPACE,IERR)
   92 IF (IERR.NE.1)GO TO 97
      WRITE(6,1007)ICUST
      GO TO 98
   97 IUWA(1)=NAM1
      CALL MODIFY(KNAM1)
   93 IUWA(1)=NAM2
      CALL MODIFY(KNAM2)
   94 IUWA(1)=NAM3
      CALL MODIFY(KNAM3)
   95 IF (IPVER.EQ.1) GO TO 99
      IF (IPVER.EQ.2) GO TO 96
      STOP 90
   96 IUWA(1)=IAGE
      CALL MODIFY(KAGE)
   99 WRITE (6,1387)
 1387 FORMAT (' CUSTOMER DETAILS AMENDED')
   98 WRITE (6,1388)
 1388 FORMAT (' *** END OF AMEND CUSTOMER DETAILS RUN ***')
      GO TO 5
  991 WRITE (6,1385)
 1385 FORMAT (' TYPE CUSTOMER NUMBER, AMENDED NAME AND AGE GROUP '
      READ (5,1386) ICUST,NAM1,NAM2,NAM3,IAGE
 1386 FORMAT (I2,1X,3A4,1X,I1)
      GO TO 992
C
```

Figure 14

```
Data:09

*** RUN OF AMEND CUSTOMER DETAILS PROGRAM ***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE CUSTOMER NUMBER AND AMENDED NAME
Data:02 BROWN,WALTER

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
RECORD FOUND AT ADDRESS  3 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  3 TO  6
OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3
HAVE BEEN ALTERED TO
BROW
***'MODIFY' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 10
OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3
HAVE BEEN ALTERED TO
N,WA
***'MODIFY' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS 11 TO 14
OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3
HAVE BEEN ALTERED TO
LTER
***'MODIFY' COMPLETED***
CUSTOMER DETAILS AMENDED
*** END OF AMEND CUSTOMER DETAILS RUN ***



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                              POINTERS
--- ------ ---- ------------------------------    ------------------
 1   CUST    1    1JONES,ALAN                     2 9 0 0 0 0 0 0 0 0
 2   ACNT    1    1 1 1     11CA                  6 0 1 0 0 0 0 0 0 0
 3   CUST    1    2BROWN,WALTER                   4 5 0 0 0 0 0 0 0 0
 4   ACNT    1    2 1 2     32CA                  5 0 3 0 0 0 0 0 0 0
 5   ACNT    1    3 2 2     33DA                  0 4 3 0 0 0 0 0 0 0
 6   ACNT    1    4 1 1     44CA                  9 2 1 0 0 0 0 0 0 0
 7   BRCH    1    1BIGTOWN      U                 0 0 0 0 0 0 0 0 0 0
 8   BRCH    1    2SMALLTOWN    R                 0 0 0 0 0 0 0 0 0 0
 9   ACNT    1    5 2 1      0CA                  0 6 1 0 0 0 0 0 0 0
10   CUST    1    3GREEN,MARY                     0 0 0 0 0 0 0 0 0 0
```

Appendix 3

The Restructuring Tasks

In Chapter 7 various restructuring tasks were postulated for the bank data base.

In this appendix actual runs of the programs involved are demonstrated. For most of the restructuring tasks it is necessary to invoke different versions of certain of the application programs described in Appendix 2 since these programs are not transparent to the task being performed. The listings of the programs given in Appendix 2 show the differences in logic for the various versions of the programs.

Open Restructuring to add Age Group to the Customer Record

The user initiates an 'Add Data Item' restructuring by entering code 20 on his terminal. The DBMS responds by requesting details of the new data item and the position in the record to which it is to be added. Since in this example an open restructuring strategy is to be used, only the schema is altered at this point to reflect the existence of the new data item in the latest version of this record.

Thus when the schema is printed there are definitions for both version 1 and version 2 of the Customer record with version 2 containing the new AGE data item. The original version of the 'Add New Customer' program (version 1) will continue to run before and after the schema has been altered and thus there is no difficulty in adding customer 3 (Anne Black) to the data base. The record for this customer is at version 2 but since the program has not supplied a value for the AGE the DBMS defaults this data item to spaces.

Eventually a new version of the Add New Customer program is introduced and customer 4 (Carol White) can be seen being added using version 2 of the program. This version is similar to version 1 but contains logic to accept the age group and to store it on the new customer record. As can be seen the DBMS responds to this change in program logic by indicating that AGE has been given an explicit value.

Similarly, the Amend Customer Details program will run after the restructuring has been initiated. The run of this program shown gives details of the operations carried out to amend the name of customer 1 to Jack Jones and also to allocate him an age group of 2. Since the DBMS always searches for the most recent version of a record first, an attempt is made to find a version 2 record for this customer. When no such record is found a successful attempt is made to find a version 1 record. Each data item in the record is now modified in turn by the program. For the first modify the DBMS detects that the record is not at the most recent version and therefore restructures the record before actually performing the modification logic. This is not necessary for subsequent modifications since the record is now at version 2.

The Calculate Balance program requires the Customer Name from the Customer record and the run of this program at version 1 shows that the DBMS can detect the revised position of this data in providing it to the program via the 'GET' command.

The Statistics Print has been deliberately enhanced to examine the version number of each Customer record accessed so that it can assume an unknown age group for all records which have not been restructured. The run shown of this program at version 2 demonstrates selective calls to the GET routine for AGE only for customers 1, 3 and 4 which as we have seen above have been modified or added since the restructuring was initiated. Note too that since customer 2 has a null value of zero for this data item the program assumes an unknown age group for this customer.

```
Data:28

***START OF ADD DATA ITEM RESTRUCTURING***
TYPE OPEN OR CLSD FOR TYPE OF RESTRUCTURING
FOLLOWED BY RECORD NAME , DATA ITEM NAME,LENGTH AND FORMAT
AND ADJACENT DATA ITEM OF DATA ITEM TO BE ADDED
Data:OPEN CUST AGE  01 ,I1) CNUM
***OPEN RESTRUCTURING NOW UNDER WAY***



Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  2
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM AGE  START AT  3 END AT  3 FORMAT ,I1)
DATA ITEM NAM1 START AT  4 END AT  7 FORMAT ,A4)
DATA ITEM NAM2 START AT  8 END AT 11 FORMAT ,A4)
DATA ITEM NAM3 START AT 12 END AT 15 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM ERNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM EALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE FRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM ENUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM ENM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM ENM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM ENM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM ENUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST
```

```
Data:0

*** RUN OF THE ADD NEW CUSTOMER PROGRAM ***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE CUSTOMER NUMBER AND NAME
Data:03 BLACK,ANNE

***FOLLOWING INFORMATION IS FROM 'STORE'***
THE FOLLOWING NEW RECORD
IS BEING ADDED TO THE DATA BASE

RECORD TYPE CUST
VERSION  2
ALL POINTERS ZERO
UNINITIALISED CHARACTERS SPACES
CHARACTERS  1 TO  2 SET TO :-
 3
CHARACTERS  4 TO  7 SET TO :-
BLAC
CHARACTERS  8 TO 11 SET TO :-
K,AN
CHARACTERS 12 TO 15 SET TO :-
NE
RECORD HAS BEEN STORED AT ADDRESS  9
***'STORE' COMPLETED***
NEW CUSTOMER ADDED
*** END OF NEW CUSTOMER RUN ***



Data:03

*** RUN OF THE ADD NEW CUSTOMER PROGRAM ***
TYPE VERSION NUMBER OF PROGRAM
Data:2
TYPE CUSTOMER NUMBER,NAME AND AGE GROUP
Data:04 WHITE,CAROL  3

***FOLLOWING INFORMATION IS FROM 'STORE'***
THE FOLLOWING NEW RECORD
IS BEING ADDED TO THE DATA BASE

RECORD TYPE CUST
VERSION  2
ALL POINTERS ZERO
UNINITIALISED CHARACTERS SPACES
CHARACTERS  1 TO  2 SET TO :-
 4
CHARACTERS  4 TO  7 SET TO :-
WHIT
CHARACTERS  8 TO 11 SET TO :-
E,CA
CHARACTERS 12 TO 15 SET TO :-
ROL
CHARACTERS  3 TO  3 SET TO :-
3
RECORD HAS BEEN STORED AT ADDRESS 10
***'STORE' COMPLETED***
NEW CUSTOMER ADDED
***
```

```
Data:[illegible]

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                                              POINTERS                TA
--- ------ ---- ------------------------------------------------ -------------------     --
 1   CUST    1   1JONES,ALAN                                       2 6 0 0 0 0 0 0 0 0     0
 2   ACNT    1   1 1 1     11CA                                    6 0 1 0 0 0 0 0 0 0     0
 3   CUST    1   2SMITH,JAMES                                      4 5 0 0 0 0 0 0 0 0     0
 4   ACNT    1   2 1 2     22CA                                    5 0 3 0 0 0 0 0 0 0     0
 5   ACNT    1   3 2 2     33DA                                    0 4 3 0 0 0 0 0 0 0     0
 6   ACNT    1   4 1 1     44CA                                    0 2 1 0 0 0 0 0 0 0     0
 7   BRCH    1   1BIGTOWN       U                                  0 0 0 0 0 0 0 0 0 0     0
 8   BRCH    1   2SMALLTOWN     R                                  0 0 0 0 0 0 0 0 0 0     0
 9   CUST    2   3 BLACK,ANNE                                      0 0 0 0 0 0 0 0 0 0     0
10   CUST    2   43WHITE,CAROL                                     0 0 0 0 0 0 0 0 0 0     0


Data:09

*** RUN OF AMEND CUSTOMER DETAILS PROGRAM ***
TYPE VERSION NUMBER OF PROGRAM
Data:2
TYPE CUSTOMER NUMBER, AMENDED NAME AND AGE GROUP
Data:01 JONES,JACK    2

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS   1 TO   2 EQUAL TO
 1
NONE HAS BEEN FOUND SO DBMS IS NOW SEARCHING FOR
RECORD TYPE CUST VERSION  1
WITH CHARACTERS   1 TO   2 EQUAL TO
 1
RECORD FOUND AT ADDRESS   1 VERSION   1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE CUST AT ADDRESS   1
IS BEING ALTERED TO   2
CHARACTERS   1 TO  2 WERE PREVIOUSLY
 1
CHARACTERS   3 TO   6 WERE PREVIOUSLY
JONE
CHARACTERS   7 TO 10 WERE PREVIOUSLY
S,AL
CHARACTERS 11 TO 14 WERE PREVIOUSLY
AN
CHARACTERS   1 TO   2 HAVE BEEN SET TO :-
 1
CHARACTERS   3 TO   3 ARE BEING SET TO SPACES
CHARACTERS   4 TO   7 HAVE BEEN SET TO :-
JONE
CHARACTERS     TO 11 HAVE BEEN SET TO :-
S,AL
CHARACTERS  12 TO 15 HAVE BEEN SET TO :-
CHARACTERS  4 TO  7
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
HAVE BEEN ALTERED TO
JONE
***'MODIFY' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  8 TO 11
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
HAVE BEEN ALTERED TO
S,JA
***'MODIFY' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS 12 TO 15
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
HAVE BEEN ALTERED TO
CK
***'MODIFY' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  3 TO  3
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
HAVE BEEN ALTERED TO
2
***'MODIFY' COMPLETED***
CUSTOMER DETAILS AMENDED
*** END OF AMEND CUSTOMER DETAILS RUN ***



Data:3u

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                              POINTERS              T
--- ------ ---- ------------------------------    -------------------   -
 1   CUST    2   12JONES,JACK                      2 6 0 0 0 0 0 0 0 0
 2   ACNT    1   1 1 1    11CA                     6 0 1 0 0 0 0 0 0 0
 3   CUST    1   2SMITH,JAMES                      4 5 0 0 0 0 0 0 0 0
 4   ACNT    1   2 1 2    22CA                     5 0 3 0 0 0 0 0 0 0
 5   ACNT    1   3 2 2    33CA                     0 4 3 0 0 0 0 0 0 0
 6   ACNT    1   4 1 1    44CA                     0 2 1 0 0 0 0 0 0 0
 7   BRCH    1   1BIGTOWN     U                    0 0 0 0 0 0 0 0 0 0
 8   BRCH    1   2SMALLTOWN   R                    0 0 0 0 0 0 0 0 0 0
 9   CUST    2   3 BLACK,ANNE                      0 0 0 0 0 0 0 0 0 0
10   CUST    2   43WHITE,CAROL                     0 0 0 0 0 0 0 0 0 0
```

```
Data:01

***RUN OF BALANCE CALCULATION PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE CUSTOMER NUMBER
Data:01

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  1 VERSION  2
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  4 TO  7
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
JONE
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  8 TO 11
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
S,JA
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS 12 TO 15
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
CK
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
IS  2
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  2 ARE:-
   11
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  2
IS  6
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  6 ARE:-
   44
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  6
IS  0
***'FIND2' COMPLETED***
[illegible]
```

```
Data:01
***RUN OF BALANCE CALCULATION PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE CUSTOMER NUMBER
Data:02

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
NONE HAS BEEN FOUND SO DBMS IS NOW SEARCHING FOR
RECORD TYPE CUST VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
RECORD FOUND AT ADDRESS  3 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  6
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3 ARE:-
SMIT
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 10
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3 ARE:-
H,JA
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS 11 TO 14
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3 ARE:-
MES
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3
IS  4
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4 ARE:-
   22
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4
IS  5
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  5 ARE:-
   33
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  5
IS  0
***'FIND2' COMPLETED***
BALANCE FOR CUSTOMER  2 SMITH,JAMES  IS   55
***END OF BALANCE CALCULATION RUN***
```

```
Data:4

***RUN OF STATISTICS PRINT PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:2

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  1 VERSION  2
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  1 TO  2
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
 1
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  3
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
2
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
NONE HAS BEEN FOUND SO DBMS IS NOW SEARCHING FOR
RECORD TYPE CUST VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
RECORD FOUND AT ADDRESS  3 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  1 TO  2
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3 ARE:-
 2
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 3
RECORD FOUND AT ADDRESS  9 VERSION  2
***'FIND1' COMPLETED***
```

```
***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  1 TO  2
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  9 ARE:-
 3
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  3
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  9 ARE:-

**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 4
RECORD FOUND AT ADDRESS 10 VERSION  2
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  1 TO  2
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS 10 ARE:-
 4
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  3
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS 10 ARE:-
3
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 5
NONE HAS BEEN FOUND SO DBMS IS NOW SEARCHING FOR
RECORD TYPE CUST VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 5
NO RECORD FOUND
***'FIND1' COMPLETED***
NUMBER OF CUSTOMERS - 4
NUMBER WITH UNKNOWN AGE GROUP -  2
NUMBER IN AGE GROUP 2 -  1
NUMBER IN AGE GROUP 3 -  1
NUMBER IN AGE GROUP 4 -  0
NUMBER IN AGE GROUP 5 -  0
***END OF STATISTICS PRINT RUN***
```

Closed Restructuring to add Age Group to Customer Record

The restructuring described above could also be carried out using a closed strategy. In this case the DBMS immediately searches for the first customer record on the data base in address sequence and, in fact, finds customer 1 at address 1. This record is then restructured by making space available for the new data item while not actually allocating it a value. The schema is also altered to reflect the existence of both version 1 and version 2 customer records. When the record is restructured the user is invited to either continue to allow the next customer record to be restructured or to run an application program.

As before, the Amend Customer Details program is run for customer 1 and since this record is at version 2 (having just been restructured) each data item can be modified in a straightforward manner. Similarly, when customer 3 (Anne Black) is added a version 2 record is created as before.

The restructuring is then allowed to continue and the record for customer 2 at address 3 is then restructured. When the restructuring mechanism then encounters the record for customer 3 at address 9 it discovers that it is already at version 2 and need not therefore be restructured.

Since no further customer records exist on the data base the restructuring is now complete and the schema can again be modified to remove the reference to the obsolete version 1 customer record.

```
Data:20

***START OF ADD DATA ITEM RESTRUCTURING***
TYPE OPEN OR CLSD FOR TYPE OF RESTRUCTURING
FOLLOWED BY RECORD NAME , DATA ITEM NAME,LENGTH AND FORMAT
AND ADJACENT DATA ITEM OF DATA ITEM TO BE ADDED
Data:CLSD CUST AGE  01  ,I1) CNUM
***CLOSED RESTRUCTURING NOW UNDER WAY***
RECORD TYPE CUST VERSION   1 FOUND
AT ADDRESS   1

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE CUST AT ADDRESS   1
IS BEING ALTERED TO   2
CHARACTERS   1 TO  2 WERE PREVIOUSLY
 1
CHARACTERS   3 TO  6 WERE PREVIOUSLY
JONE
CHARACTERS   7 TO 10 WERE PREVIOUSLY
S,AL
CHARACTERS 11 TO 14 WERE PREVIOUSLY
AN
CHARACTERS   1 TO  2 HAVE BEEN SET TO :-
 1
CHARACTERS   3 TO  3 ARE BEING SET TO SPACES
CHARACTERS   4 TO  7 HAVE BEEN SET TO :-
JONE
CHARACTERS   8 TO 11 HAVE BEEN SET TO :-
S,AL
CHARACTERS 12 TO 15 HAVE BEEN SET TO :-
AN
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21
```

```
Data:20

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  2
DATA ITEM CNUM START AT   1 END AT  2 FORMAT ,I2)
DATA ITEM AGE  START AT   3 END AT  3 FORMAT ,I1)
DATA ITEM NAM1 START AT   4 END AT  7 FORMAT ,A4)
DATA ITEM NAM2 START AT   8 END AT 11 FORMAT ,A4)
DATA ITEM NAM3 START AT  12 END AT 15 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT   1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT   3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT   7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT  11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT   1 END AT  2 FORMAT ,I2)
DATA ITEM BRNO START AT   3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT   5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT   7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT  12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT   1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT   3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT   7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT  11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT  15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST
```

```
Data:09

*** RUN OF AMEND CUSTOMER DETAILS PROGRAM ***
TYPE VERSION NUMBER OF PROGRAM
Data:2
TYPE CUSTOMER NUMBER, AMENDED NAME AND AGE GROUP
Data:01 JONES,JACK    2

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS   1 VERSION  2
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  4 TO  7
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
HAVE BEEN ALTERED TO
JONE
***'MODIFY' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  8 TO 11
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
HAVE BEEN ALTERED TO
S,JA
***'MODIFY' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS 12 TO 15
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
HAVE BEEN ALTERED TO
CK
***'MODIFY' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  3 TO  3
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
HAVE BEEN ALTERED TO
2
***'MODIFY' COMPLETED***
CUSTOMER DETAILS AMENDED
*** END OF AMEND CUSTOMER DETAILS RUN ***
```

```
Data:08

*** RUN OF THE ADD NEW CUSTOMER PROGRAM ***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE CUSTOMER NUMBER AND NAME
Data:03 BLACK,ANNE

***FOLLOWING INFORMATION IS FROM 'STORE'***
THE FOLLOWING NEW RECORD
IS BEING ADDED TO THE DATA BASE

RECORD TYPE CUST
VERSION  2
ALL POINTERS ZERO
UNINITIALISED CHARACTERS SPACES
CHARACTERS  1 TO  2 SET TO :-
 3
CHARACTERS  4 TO  7 SET TO :-
BLAC
CHARACTERS  8 TO 11 SET TO :-
K,AN
CHARACTERS 12 TO 15 SET TO :-
NE
RECORD HAS BEEN STORED AT ADDRESS  9
***'STORE' COMPLETED***
NEW CUSTOMER ADDED
*** END OF NEW CUSTOMER RUN ***
```

```
Data:21
RECORD TYPE CUST VERSION  1 FOUND
AT ADDRESS  3

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE CUST AT ADDRESS  3
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 2
CHARACTERS  3 TO  6 WERE PREVIOUSLY
SMIT
CHARACTERS  7 TO 10 WERE PREVIOUSLY
H,JA
CHARACTERS 11 TO 14 WERE PREVIOUSLY
MES
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 2
CHARACTERS  3 TO  3 ARE BEING SET TO SPACES
CHARACTERS  4 TO  7 HAVE BEEN SET TO :-
SMIT
CHARACTERS  8 TO 11 HAVE BEEN SET TO :-
H,JA
CHARACTERS 12 TO 15 HAVE BEEN SET TO :-
MES
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21
```

```
Data:21
RECORD TYPE CUST VERSION  2 FOUND
AT ADDRESS  9
THIS RECORD HAS ALREADY BEEN RESTRUCTURED AND NEED NOT BE ALTERED
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21
```

```
Data:21
*** CLOSED RESTRUCTURING COMPLETE ***
```

```
Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  2
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM AGE  START AT  3 END AT  3 FORMAT ,I1)
DATA ITEM NAM1 START AT  4 END AT  7 FORMAT ,A4)
DATA ITEM NAM2 START AT  8 END AT 11 FORMAT ,A4)
DATA ITEM NAM3 START AT 12 END AT 15 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM ERNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST
```

```
Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                                                  POINTERS                     T
--- ------ ---- ------------------------------------------------      ----------------------       -
 1   CUST    2   12JONES,JACK                                         2 6 0 0 0 0 0 0 0 0
 2   ACNT    1   1 1 1    11CA                                        6 0 1 0 0 0 0 0 0 0
 3   CUST    2   2 SMITH,JAMES                                        4 5 0 0 0 0 0 0 0 0
 4   ACNT    1   2 1 2    22CA                                        5 0 3 0 0 0 0 0 0 0
 5   ACNT    1   3 2 2    33CA                                        0 4 3 0 0 0 0 0 0 0
 6   ACNT    1   4 1 1    44CA                                        0 2 1 0 0 0 0 0 0 0
 7   BRCH    1   1BIGTOWN      U                                      0 0 0 0 0 0 0 0 0 0
 8   BRCH    1   2SMALLTOWN    R                                      0 [illegible] 0 0 0 0
 9   CUST    2   [illegible]                                          [illegible] 0 0 0 0 0
```

Closed Restructuring to delete the Age Group from the Customer Record

The opposite function to that described above can also be carried out using a closed strategy. To delete a data item the user enters a code of 22 on his terminal. In this case only the name of the data item to be deleted and the name of its record are requested.

Once again the first customer record is immediately found for customer 1 at address 1. This is then restructured to version 3 by re-allocating the position of subsequent data items.

Version 1 of the Amend Customer Details program can once again be run and in this case there is no difficulty in operating on the record which has just been restructured. Similarly, version 1 of the Add New Customer Program can be used to add a record for customer 4 (Carol White) at version 3.

The records for customers 2 and 3 are then restructured and that for customer 4 bypassed since it is already at version 3. Since the restructuring is then complete the obsolete reference to the version 2 customer record in the schema is then removed.

```
Data:27

***START OF DELETE DATA ITEM RESTRUCTURING***
TYPE OPEN OR CLSD FOR TYPE OF RESTRUCTURING
FOLLOWED BY RECORD NAME AND DATA ITEM TO BE DELETED
Data:CLSD CUST AGE
***CLOSED RESTRUCTURING NOW UNDER WAY***
RECORD TYPE CUST VERSION  2 FOUND
AT ADDRESS  1

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE CUST AT ADDRESS  1
IS BEING ALTERED TO  3
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 1
CHARACTERS  4 TO  7 WERE PREVIOUSLY
JONE
CHARACTERS  8 TO 11 WERE PREVIOUSLY
S,JA
CHARACTERS 12 TO 15 WERE PREVIOUSLY
CK
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 1
CHARACTERS  3 TO  6 HAVE BEEN SET TO :-
JONE
CHARACTERS  7 TO 10 HAVE BEEN SET TO :-
S,JA
CHARACTERS 11 TO 14 HAVE BEEN SET TO :-
CK
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21
```

```
Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  3
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE CUST VERSION  2
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM AGE  START AT  3 END AT  3 FORMAT ,I1)
DATA ITEM NAM1 START AT  4 END AT  7 FORMAT ,A4)
DATA ITEM NAM2 START AT  8 END AT 11 FORMAT ,A4)
DATA ITEM NAM3 START AT 12 END AT 15 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM
```

```
RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BRNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE CWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM ENUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM ENM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM ENM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM ENM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM ENUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST
```

```
Data:09

*** RUN OF AMEND CUSTOMER DETAILS PROGRAM ***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE CUSTOMER NUMBER AND AMENDED NAME
Data:01 JONES,ALAN

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  3
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  1 VERSION  3
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  3 TO  6
OF RECORD TYPE CUST VERSION  3 AT ADDRESS  1
HAVE BEEN ALTERED TO
JONE
***'MODIFY' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 10
OF RECORD TYPE CUST VERSION  3 AT ADDRESS  1
HAVE BEEN ALTERED TC
S,AL
***'MODIFY' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS 11 TO 14
OF RECORD TYPE CUST VERSION  3 AT ADDRESS  1
HAVE BEEN ALTERED TC
AN
***'MODIFY' COMPLETED***
CUSTOMER DETAILS AMENDED
*** END OF AMEND CUSTOMER DETAILS RUN ***
```

```
Data:0

*** RUN OF THE ADD NEW CUSTOMER PROGRAM ***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE CUSTOMER NUMBER AND NAME
Data:04 WHITE,CAROL

***FOLLOWING INFORMATION IS FROM 'STORE'***
THE FOLLOWING NEW RECORD
IS BEING ADDED TO THE DATA BASE

RECORD TYPE CUST
VERSION  3
ALL POINTERS ZERO
UNINITIALISED CHARACTERS SPACES
CHARACTERS  1 TO  2 SET TO :-
 4
CHARACTERS  3 TO  6 SET TO :-
WHIT
CHARACTERS  7 TO 10 SET TO :-
E,CA
CHARACTERS 11 TO 14 SET TO :-
ROL
RECORD HAS BEEN STORED AT ADDRESS 10
***'STORE' COMPLETED***
NEW CUSTOMER ADDED
*** END OF NEW CUSTOMER RUN ***



Data:21
RECORD TYPE CUST VERSION  2 FOUND
AT ADDRESS  3

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE CUST AT ADDRESS  3
IS BEING ALTERED TO  3
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 2
CHARACTERS  4 TO  7 WERE PREVIOUSLY
SMIT
CHARACTERS  8 TO 11 WERE PREVIOUSLY
H,JA
CHARACTERS 12 TO 15 WERE PREVIOUSLY
MES
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 2
CHARACTERS  3 TO  6 HAVE BEEN SET TO :-
SMIT
CHARACTERS  7 TO 10 HAVE BEEN SET TO :-
H,JA
CHARACTERS 11 TO 14 HAVE BEEN SET TO :-
MES
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21
```

```
Data:21
RECORD TYPE CUST VERSION  2 FOUND
AT ADDRESS  9

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE CUST AT ADDRESS  9
IS BEING ALTERED TO  3
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 3
CHARACTERS  4 TO  7 WERE PREVIOUSLY
BLAC
CHARACTERS  8 TO 11 WERE PREVIOUSLY
K,AN
CHARACTERS 12 TO 15 WERE PREVIOUSLY
NE
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 3
CHARACTERS  3 TO  6 HAVE BEEN SET TO :-
BLAC
CHARACTERS  7 TO 10 HAVE BEEN SET TO :-
K,AN
CHARACTERS 11 TO 14 HAVE BEEN SET TO :-
NE
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21


Data:21
RECORD TYPE CUST VERSION  3 FOUND
AT ADDRESS 10
THIS RECORD HAS ALREADY BEEN RESTRUCTURED AND NEED NOT BE ALTERED
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21


Data:21
*** CLOSED RESTRUCTURING COMPLETE ***
```

```
Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  3
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM ERNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                                        POINTERS
--- ------ ---- ------------------------------------------- -------------------
 1   CUST    3   1JONES,ALAN                                 2 6 0 0 0 0 0 0 0 0
 2   ACNT    1   1 1 1    11CA                               6 0 1 0 0 0 0 0 0 0
 3   CUST    3   2SMITH,JAMES                                4 5 0 0 0 0 0 0 0 0
 4   ACNT    1   2 1 2    22CA                               5 0 3 0 0 0 0 0 0 0
 5   ACNT    1   3 2 2    33CA                               0 4 3 0 0 0 0 0 0 0
 6   ACNT    1   4 1 1    44CA                               0 2 1 0 0 0 0 0 0 0
 7   BRCH    1   1BIGTOWN    U                               0 0 0 0 0 0 0 0 0 0
 8   BRCH    1   2SMALLTOWN  R                               0 0 0 0 0 0 0 0 0 0
 9   CUST    3   3BLACK,ANNE                                 0 0 0 0 0 0 0 0 0 0
10   CUST    3   4WHITE,CAROL                                0 0 0 0 0 0 0 0 0 0
```

Closed Restructuring to expand the format of the Balance Data Item

To amend the format of a data item the user must enter a code of 23 on his terminal. The DBMS responds by requesting the new FORTRAN format of the data item. In this case Balance changes from I5 to I6.

The run of the Transaction Posting program prior to the restructuring shows £10 being posted to account 02. Once the restructuring is initiated the first account record on the data base for account 1 at address 2 is restructured to version 2 by accommodating an additional character for the Balance data item. The schema is altered to hold details of both version 1 and version 2 Account records. Account 2 is then similarly restructured.

The Transaction Posting program at version 2 is then run to post a further £10 to account 2. The difference in version is necessary to allow a balance of up to six digits to be printed. The same version of the Transaction Posting program is then used to post £10 to account 3. Since this has not yet been restructured this is done prior to the record being modified.

The restructuring mechanism is then allowed to continue and on encountering the record for account 3 it detects that there is no requirement for restructuring. When account 4 is restructured the operation is complete and the entry for version 1 of the account record on the schema can be removed.

```
Data:00

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:2
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:02 0010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
RECORD FOUND AT ADDRESS  4 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4 ARE:-
   22
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4
HAVE BEEN ALTERED TO
   32
***'MODIFY' COMPLETED***
TRANSACTION OF $   10 POSTED - NEW BALANCE $   32
***END OF TRANSACTION POSTING RUN***



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                                POINTERS           TA
--- ------ ---- ----------------------------------  -----------------  --
 1   CUST    1   1JONES,ALAN                         2 6 0 0 0 0 0 0 0 0  0
 2   ACNT    1   1 1 1     11CA                      6 0 1 0 0 0 0 0 0 0  0
 3   CUST    1   2SMITH,JAMES                        4 5 0 0 0 0 0 0 0 0  0
 4   ACNT    1   2 1 2     32CA                      5 0 3 0 0 0 0 0 0 0  0
 5   ACNT    1   3 2 2     33DA                      0 4 3 0 0 0 0 0 0 0  0
 6   ACNT    1   4 1 1     44CA                      0 2 1 0 0 0 0 0 0 0  0
 7   ERCH    1   1BIGTOWN      U                     0 0 0 0 0 0 0 0 0 0  0
 8   ERCH    1   2SMALLTOWN    R                     0 0 0 0 0 0 0 0 0 0  0



Data:23

***START OF AMEND DATA ITEM FORMAT RESTRUCTURING ***
TYPE OPEN OR CLSD FOR TYPE OF RESTRUCTURING
FOLLOWED BY RECORD NAME, DATA ITEM NAME, LENGTH AND NEW FORMAT
Data:CLSD ACNT BALC 06 ,I0)
***CLOSED RESTRUCTURING NOW UNDER WAY***
RECORD TYPE ACNT VERSION  1 [illegible]
```

```
*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  2
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 1
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 1
CHARACTERS  7 TO 11 WERE PREVIOUSLY
    11
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 1
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 1
CHARACTERS  7 TO 12 HAVE BEEN SET TO :-
     11
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21
```

```
Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  2
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM ERNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 12 FORMAT ,I6)
DATA ITEM ACTP START AT 13 END AT 14 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM ERNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
```

```
RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOC. START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST



Data:21
RECORD TYPE ACNT VERSION  1 FOUND
AT ADDRESS  4

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  4
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 2
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 2
CHARACTERS  7 TO 11 WERE PREVIOUSLY
   32
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 2
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 2
CHARACTERS  7 TO 12 HAVE BEEN SET TO :-
    32
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21



Data:02

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:2
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:02 C010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DEMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
RECORD FOUND AT ADDRESS  4 VERSION  2
***'FIND1' COMPLETED***
```

```
***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 12
 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  4 ARE:-
    32
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 12
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  4
HAVE BEEN ALTERED TO
    42
***'MODIFY' COMPLETED***
TRANSACTION OF $   10 POSTED - NEW BALANCE $     42
***END OF TRANSACTION POSTING RUN***


Data:02
***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:2
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:03 0010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
  3
NONE HAS BEEN FOUND SO DBMS IS NOW SEARCHING FOR
RECORD TYPE ACNT VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
  3
RECORD FOUND AT ADDRESS   5 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  5 ARE:-
    33
**'GET' COMPLETED***
***FOLLOWING INFORMATION IS FROM 'MODIFY'***
*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  5
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
  3
CHARACTERS  3 TO  4 WERE PREVIOUSLY
  2
CHARACTERS  5 TO  6 WERE PREVIOUSLY
  2
CHARACTERS  7 TO 11 WERE PREVIOUSLY
    33
CHARACTERS 12 TO 13 WERE PREVIOUSLY
DA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
  3
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
  2
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
  2
CHARACTERS  7 TO 12 HAVE BEEN SET TO :-
    33
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
CHARACTERS  7 TO 12
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  5
HAVE BEEN ALTERED TO
    43
***'MODIFY' COMPLETED***
TRANSACTION OF $    10 POSTED - NEW BALANCE $    43
***END OF TRANSACTION POSTING RUN***


Data:21
RECORD TYPE ACNT VERSION  2 FOUND
AT ADDRESS  5
THIS RECORD HAS ALREADY BEEN RESTRUCTURED AND NEED NOT BE ALTERED
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21


Data:21
RECORD TYPE ACNT VERSION  1 FOUND
AT ADDRESS  6

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  6
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 4
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 1
CHARACTERS  7 TO 11 WERE PREVIOUSLY
   44
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 4
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 1
CHARACTERS  7 TO 12 HAVE BEEN SET TO :-
    44
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21


Data:21
*** CLOSED RESTRUCTURING COMPLETE ***
```

```
Data:2

*** TH  CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  2
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BRNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 12 FORMAT ,I6)
DATA ITEM ACTP START AT 13 END AT 14 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA
--- ------ ---- --------------------------------    POINTERS
                                                    ------------------   T
 1   CUST    1   1JONES,ALAN                        2 6 0 0 0 0 0 0 0 0
 2   ACNT    2   1 1 1     11CA                     6 0 1 0 0 0 0 0 0 0
 3   CUST    1   2SMITH,JAMES                       4 5 0 0 0 0 0 0 0 0
 4   ACNT    2   2 1 2     42CA                     5 0 3 0 0 0 0 0 0 0
 5   ACNT    2   3 2 2     43DA                     0 4 3 0 0 0 0 0 0 0
 6   ACNT    2   4 1 1     44CA                     0 2 1 0 0 0 0 0 0 0
 7   BRCH    1   1BIGTOWN     U                     0 0 0 0 0 0 0 0 0 0
 8   BRCH    1   2SMALLTOWN   R                     0 0 0 0 0 0 0 0 0 0
```

A second run of the same restructuring is also shown. In this case a run of the original version of the Calculate Balance Program takes place after the record for account 1 has been restructured. The DBMS responds to the GET command from the program for the Balance from both a version 1 and version 2 account record but it uses the schema definition of each to supply the correct data. Even though the program accesses the version 1 record for account 4 the DBMS does not restructure it since it has not been modified.

```
Data:[illegible]

***ST[illegible] [illegible]ATE ITEM FORMAT RESTRUCTURING ***
TYPE [illegible] FOR TYPE OF RESTRUCTURING
FOLL[illegible] BY [illegible], [illegible] ITEM NAME, LENGTH AND [illegible] FO[illegible]
Data:[illegible] [illegible] [illegible] ,I6)
***C[illegible] STRUCTURING NOW UNDER WAY***
RECORD TYPE ACNT VERSION  1 FOUND
AT ADDRESS  2

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  2
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 1
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 1
CHARACTERS  7 TO 11 WERE PREVIOUSLY
    11
CHARACTERS 12 TO 14 WERE PREVIOUSLY
C
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 1
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 1
CHARACTERS  7 TO 12 HAVE BEEN SET TO :-
    11
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
C
*** [illegible] HAS BEEN RESTRUCTURED ***
IF NO OTHER [illegible] IS OUTSTANDING TYPE REPLY [illegible]1


Data:[illegible]1

***RUN OF BALANCE CALCULATION PROGRAM***
TYPE VERSION NUMBER OF [illegible]
Data:1
TYPE CUSTOMER [illegible]
Data:[illegible]1

***FOLLOWING INFORMATION [illegible] 'FI[illegible]01'***
[illegible]
RECORD TYPE CUST VERSION  1
WITH [illegible] TO
 1
R[illegible] [illegible] SION  1
***'F[illegible] 1' [illegible]***

***F[illegible] [illegible]***
CHA[illegible] T[illegible]
   F [illegible] 1 [illegible] :-
J :
```

```
***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 10
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1 ARE:-
S,AL
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS 11 TO 14
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1 ARE:-
AN
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1
IS  2
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  2
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 12
 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2 ARE:-
    11
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2
IS  6
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  6 ARE:-
   44
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  6
IS  0
***'FIND2' COMPLETED***
BALANCE FOR CUSTOMER  1 JONES,ALAN   IS   55
***END OF BALANCE CALCULATION RUN***
```

```
Data:21
RECORD TYPE ACNT VERSION  1 FOUND
AT ADDRESS  4

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  4
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 2
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 2
CHARACTERS  7 TO 11 WERE PREVIOUSLY
   22
CH[illegible]   1[illegible]   [illegible]SLY
```

```
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 2
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 2
CHARACTERS  7 TO 12 HAVE BEEN SET TO :-
    22
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21
```

```
Data:21
RECORD TYPE ACNT VERSION   1 FOUND
AT ADDRESS  5

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS   5
IS BEING ALTERED TO   2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 3
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 2
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 2
CHARACTERS  7 TO 11 WERE PREVIOUSLY
    33
CHARACTERS 12 TO 13 WERE PREVIOUSLY
DA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 3
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 2
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 2
CHARACTERS  7 TO 12 HAVE BEEN SET TO :-
    33
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
DA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21
```

```
Data:21
RECORD TYPE ACNT VERSION   1 FOUND
AT ADDRESS  6

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS   6
IS BEING ALTERED TO   2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 4
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 1
CHARACTERS  7 TO 11 WERE PREVIOUSLY
    [illegible]
```

```
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 4
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 1
CHARACTERS  7 TO 12 HAVE BEEN SET TO :-
    44
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21



Data:21
*** CLOSED RESTRUCTURING COMPLETE ***




Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                                 POINTERS             TA
--- ------ ---- -----------------------------       -------------------- --
 1    CUST    1    1JONES,ALAN                       2 6 0 0 0 0 0 0 0 0  0
 2    ACNT    2    1 1 1     11CA                    6 0 1 0 0 0 0 0 0 0  0
 3    CUST    1    2SMITH,JAMES                      4 5 0 0 0 0 0 0 0 0  0
 4    ACNT    2    2 1 2     22CA                    5 0 3 0 0 0 0 0 0 0  0
 5    ACNT    2    3 2 2     33DA                    0 4 3 0 0 0 0 0 0 0  0
 6    ACNT    2    4 1 1     44CA                    0 2 1 0 0 0 0 0 0 0  0
 7    BRCH    1    1BIGTOWN     U                    0 0 0 0 0 0 0 0 0 0  0
 8    BRCH    1    2SMALLTOWN   R                    0 0 0 0 0 0 0 0 0 0  0
```

Closed Restructuring to contract the format of the Balance Data Item

The Balance is now shown as being contracted from format I5 to I4.

Before the restructuring commences £9999 is posted to account 01 to bring its balance to £10010. When the restructuring attempts to restructure this record it finds that the value will not fit into the 4 available characters and the normal FORTRAN overflow convention of substituting asterisks is used. All other account records are then restructured and in these cases the values of the balance can be accommodated in the available 4 digits.

When an attempt is made to post £9999 to account 02 there is once again an overflow since the new balance of £10021 will not fit in 4 digits.

```
Data:02

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:01 9999

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DEMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  2 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  2 ARE:-
   11
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  2
HAVE BEEN ALTERED TO
10010
***'MODIFY' COMPLETED***
TRANSACTION OF $ 9999 POSTED - NEW BALANCE $10010
***END OF TRANSACTION POSTING RUN***
```

```
Data:23

***START OF AMEND DATA ITEM FORMAT RESTRUCTURING ***
TYPE OPEN OR CLSD FOR TYPE OF RESTRUCTURING
FOLLOWED BY RECORD NAME , DATA ITEM NAME, LENGTH AND NEW FORMAT
Data:CLSD ACNT BALC (4 ,I4)
***CLOSED RESTRUCTURING NOW UNDER WAY***
RECORD TYPE ACNT VERSION  1 FOUND
AT ADDRESS  2

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  2
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 1
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 1
CHARACTERS  7 TO 11 WERE PREVIOUSLY
10010
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 1
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 1
CHARACTERS  7 TO 10 HAVE BEEN SET TO :-
****
CHARACTERS 11 TO 12 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21



Data:21
RECORD TYPE ACNT VERSION  1 FOUND
AT ADDRESS  4

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  4
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 2
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 2
CHARACTERS  7 TO 11 WERE PREVIOUSLY
   22
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 2
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 2
CHARACTERS  7 TO 10 HAVE BEEN SET TO :-
  22
CHARACTERS 11 TO 12 HAVE BEEN SET TO :-
CA
```

```
Data:21
RECORD TYPE ACNT VERSION  1 FOUND
AT ADDRESS  5

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  5
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 3
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 2
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 2
CHARACTERS  7 TO 11 WERE PREVIOUSLY
   33
CHARACTERS 12 TO 13 WERE PREVIOUSLY
DA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 3
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 2
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 2
CHARACTERS  7 TO 10 HAVE BEEN SET TO :-
  33
CHARACTERS 11 TO 12 HAVE BEEN SET TO :-
DA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21
```

```
Data:21
RECORD TYPE ACNT VERSION  1 FOUND
AT ADDRESS  6

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  6
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 4
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 1
CHARACTERS  7 TO 11 WERE PREVIOUSLY
   44
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 4
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 1
CHARACTERS  7 TO 10 HAVE BEEN SET TO :-
  44
CHARACTERS 11 TO 12 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21
```

```
Data:21
*** CLOSED RESTRUCTURING COMPLETE ***



Data:02

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:02 9999

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
RECORD FOUND AT ADDRESS  4 VERSION  2
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 10
 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  4 ARE:-
  22
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 10
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  4
HAVE BEEN ALTERED TO
****
***'MODIFY' COMPLETED***
TRANSACTION OF $ 9999 POSTED - NEW BALANCE $10021
***END OF TRANSACTION POSTING RUN***



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA
                                                     POINTERS              TAG
--- ------ ---- --------------------------------    -------------------   ---
 1   CUST    1   1JONES,ALAN
 2   ACNT    2   1 1 1****CA                         2 6 0 0 0 0 0 0 0 0   0
 3   CUST    1   2SMITH,JAMES                        6 0 1 0 0 0 0 0 0 0   0
 4   ACNT    2   2 1 2****CA                         4 5 0 0 0 0 0 0 0 0   0
 5   ACNT    2   3 2 2  33DA                         5 0 3 0 0 0 0 0 0 0   0
 6   ACNT    2   4 1 1  44CA                         0 4 3 0 0 0 0 0 0 0   0
 7   BRCH    1   1BIGTOWN     U                      0 2 1 0 0 0 0 0 0 0   0
 8   BRCH    1   2SMALLTOWN   R                      0 0 0 0 0 0 0 0 0 0   0
                                                     0 0 0 0 0 0 0 0 0 0   0
```

Restructuring to Interchange position of Customer Number and Branch Number on Account Record

To interchange the position of two data items on the same record the user enters a code of 26. The DBMS responds by requesting the name of the record and the data items to be interchanged.

As for previous examples only the schema is amended and existing records are only restructured when they are modified. In the example shown a new account number 5 is added to the data base. Although this record is added as version 2 with the data items interchanged the original version of the program can continue to be run. In particular, note that the Store command continues to process the data items in the sequence held in the sub-schema although this is not the sequence for the version 2 data base record.

```
Data:28

*** START OF INTERCHANGE DATA ITEM RESTRUCTURING ***
TYPE OPEN OR CLSD FOR TYPE OF RESTRUCTURING
FOLLOWED BY RECORD NAME AND NAMES OF DATA ITEMS
Data:OPEN ACNT CUNO ERNO
***OPEN RESTRUCTURING NOW UNDER WAY***



Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  2
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM CUNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM ERNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM ERNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST
```

```
Data:03
***RUN OF OPEN NEW ACCOUNT PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE BRANCH , ACCOUNT NUMBER , CUSTOMER , ACCOUNT TYPE
Data:02 05 01 CA

***FOLLOWING INFORMATION IS FROM 'STORE'***
THE FOLLOWING NEW RECORD
IS BEING ADDED TO THE DATA BASE

RECORD TYPE ACNT
VERSION  2
ALL POINTERS ZERO
UNINITIALISED CHARACTERS SPACES
CHARACTERS  5 TO  6 SET TO :-
 2
CHARACTERS  1 TO  2 SET TO :-
 5
CHARACTERS  3 TO  4 SET TO :-
 1
CHARACTERS 12 TO 13 SET TO :-
CA
CHARACTERS  7 TO 11 SET TO :-
    0
RECORD HAS BEEN STORED AT ADDRESS  9
***'STORE' COMPLETED***

*** FOLLOWING INFORMATION IS FROM 'INSERT'***
CHARACTERS  3 TO  4
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  9 ARE :-
 1
DBMS IS SEARCHING FOR RECORD TYPE CUST VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO :-
 1
RECORD FOUND AT ADDRESS  1
RECORD-TYPE AND VERSION OF RECORD AT ADDRESS  6 ARE :-
ACNT  1
POINTER  1 OF RECORD TYPE ACNT ADDRESS  6 VERSION  1
HAS BEEN ALTERED TO  9
POINTER  2 OF RECORD TYPE ACNT ADDRESS  9 VERSION  2
HAS BEEN ALTERED TO  6
POINTER  3 OF RECORD TYPE ACNT ADDRESS  9 VERSION  2
HAS BEEN ALTERED TO  1
POINTER  2 OF RECORD TYPE CUST ADDRESS  1 VERSION  1
HAS BEEN ALTERED TO  9
***'INSERT' COMPLETED***
NEW ACCOUNT OPENED
**END OF OPEN NEW ACCOUNT RUN***



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                            POINTERS                TAG
--- ------ ---- ------------------------------  ----------------------  ---
 1   CUST    1    1JONES,ALAN                    2 9 0 0 0 0 0 0 0 0     0
 2   ACNT    1    1 1 1    11CA                  6 0 1 0 0 0 0 0 0 0     0
 3   CUST    1    2SMITH,JAMES                   4 5 0 0 0 0 0 0 0 0     0
 4   ACNT    1    2 1 2    22CA                  5 0 3 0 0 0 0 0 0 0     0
 5   ACNT    1    3 2 2    33CA                  0 4 3 0 0 0 0 0 0 0     0
 6   ACNT    1    4 1 1    44CA                  9 2 1 0 0 0 0 0 0 0     0
 7   BRCH    1    1BIGTOWN      U                0 0 0 0 0 0 0 0 0 0     0
 8   BRCH    1    2SMALLTOWN    R                0 0 0 0 0 0 0 0 0 0     0
 9   ACNT    2    5 1 2     0CA                  0 6 1 0 0 0 0 0 0 0     0
```

Migration of the Balance data item from the Account to the Customer Record

Data Item Migration is performed by the user entering a code of 27 on this terminal. Only a closed strategy with parallel running is available in this example.

Each set occurrence is restructured as a single unit and the user therefore only has an opportunity to execute an application program when each set occurrence has been restructured. The DBMS accesses each set by searching for each customer record (the owner of the set) in address sequence. The first customer record is for customer 1 at address 1. The First Member record pointer on this record is used to establish the address of the first account in the set - account 1 at address 2. Before this record is restructured the value in the Balance data item is retained for later use. The restructuring of the account record to version 2 results in the balance data item being deleted. The Next-Member record pointer on the account record shows the next account in the set to be account 4 at address 6. Once again the balance is extracted before the record is restructured. In this case the value of the data item (£44) is added to that extracted from account 1 (£11) to produce a current net balance for the customer of £55.

The Next Member record pointer on this account has a value of zero indicating that there are no further member records. The original owner record is then restructured and the value of the net balance as calculated above is allocated to the new data item.

The Schema and Data Base are displayed at this point. Versions 1 and 2 of both the Customer and Account records are defined at this stage. The Data Base contains not only the new version 2 Customer and Account records but also the corresponding original version 1 records which have been moved to the first available spare record slots on the data base. The parallel record tag is used to associate each version 1 record with its version 2 counterpart and vice versa.

The Transaction Posting program is run to post £10 to account 2 which is still at version 1. Version 3 of this program is version specific in that it examines the version of the account record retrieved before it modifies the balance - in this case, as previously, on the account record. Note that in this example no restructuring takes place prior to the modify because of the type of restructuring taking place.

When £10 is posted to account 1 the program detects that the account record is at version 2. The corresponding customer record is then accessed and the balance on this record modified accordingly. Since parallel running is taking place the program then establishes the corresponding version 1 account record as the current of run unit using a third version of the FIND command. The balance on this record is then modified by the £10 posting.

Version 2 of the Calculate Balance Program is then run to provide the balances for both customer 2 and customer 1. Like the Transaction Posting program this program is version specific. Since customer 2 is at version 1 the program accumulates the customer balance by summation of the balances on the associated account records. For customer 1 (at version 2) the balance is derived directly from that record.

The restructuring is then allowed to continue and the set corresponding to customer 2 is restructured as before. Since no further customers exist the restructuring is complete after this point.

Parallel running, however, continues and in particular a Balance Calculation Audit program is run which compares the balance held on a version 2 customer record with the sum of the balances on the associated version 1 account records. This run is initiated by the user entering a code of 10. The run verifies that the balance for customer 1 is the same in both cases.

Finally, the user can enter a code of 24 which terminates the parallel running phase. This is done by removing the schema entries for version 1 of both the Customer and Account records, by setting the parallel run tag for all version 2 Customer and Account records to zero on the data base and by removing all version 1 Customer and Account records.

When version 3 of the Transaction Posting program is run after parallel running is complete the call to FIND3 returns a code indicating that no corresponding Account record exists and no attempt is therefore made to modify the balance on such a record.

```
Data:27

*** START OF MEMBER TO OWNER DATA ITEM MIGRATION ***
RESTRUCTURING USES CLOSED STRATEGY WITH PARALLEL RUN
TYPE SET NAME, SOURCE RECORD NAME,DATA ITEM NAME,
DESTINATION RECORD NAME AND ADJACENT DATA ITEM
Data:CUAC ACNT BALC CUST CNUM
***CLOSED RESTRUCTURING NOW UNDER WAY***
RECORD TYPE CUST VERSION  1 FOUND
AT ADDRESS  1
POINTER  1 GIVES ADDRESS OF FIRST MEMBER AS  2
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  2 ARE :-
  11

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  2
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 1
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 1
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 1
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 1
CHARACTERS  7 TO  8 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
POINTER  1 GIVES ADDRESS OF NEXT MEMBER AS  6
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  6 ARE :-
  44

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  6
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 4
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 1
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 4
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 1
CHARACTERS  7 TO  8 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE CUST AT ADDRESS  1
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 1
CHARACTERS  3 TO  6 WERE PREVIOUSLY
JONE
CHARACTERS  7 TO 10 WERE PREVIOUSLY
S,AL
CHARACTERS 11 TO 14 WERE PREVIOUSLY
AN
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 1
CHARACTERS  3 TO  7 ARE BEING SET TO SPACES
CHARACTERS  8 TO 11 HAVE BEEN SET TO :-
JONE
CHARACTERS 12 TO 15 HAVE BEEN SET TO :-
S,AL
CHARACTERS 16 TO 19 HAVE BEEN SET TO :-
AN
*** RECORD HAS BEEN RESTRUCTURED ***
CHARACTERS  3 TO  7
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
HAVE BEEN ALTERED TO
   55
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 28
```

```
Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  2
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BALC START AT  3 END AT  7 FORMAT ,I5)
DATA ITEM NAM1 START AT  8 END AT 11 FORMAT ,A4)
DATA ITEM NAM2 START AT 12 END AT 15 FORMAT ,A4)
DATA ITEM NAM3 START AT 16 END AT 19 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  2
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BRNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM ACTP START AT  7 END AT  8 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIP
SET NAME CUAC POINTER TYPE OWNP
KEY DATA ITEM ACNO
```

```
RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BRNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIP
SET NAME CUAC POINTER TYPE OWNP
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST



Data:30

*** THE CURRENT DATA BASE ***

                                                    POINTERS               T
ADD RECORD VRSN DATA                                
--- ------ ---- ----------------------------------  ------------------     --
 1   CUST    2   1    55JONES,ALAN                  2 6 0 0 0 0 0 0 0 0
 2   ACNT    2   1 1 1CA                            6 0 1 0 0 0 0 0 0 0    10
 3   CUST    1   2SMITH,JAMES                       4 5 0 0 0 0 0 0 0 0
 4   ACNT    1   2 1 2   22CA                       5 0 3 0 0 0 0 0 0 0
 5   ACNT    1   3 2 2   33DA                       0 4 3 0 0 0 0 0 0 0
 6   ACNT    2   4 1 1CA                            0 2 1 0 0 0 0 0 0 0    1
 7   BRCH    1   1BIGTOWN     U                     0 0 0 0 0 0 0 0 0 0
 8   BRCH    1   2SMALLTOWN   R                     0 0 0 0 0 0 0 0 0 0
 9   CUST    1   1JONES,ALAN                        2 6 0 0 0 0 0 0 0 0
10   ACNT    1   1 1 1    11CA                      6 0 1 0 0 0 0 0 0 0
11   ACNT    1   4 1 1    44CA                      0 2 1 0 0 0 0 0 0 0



Data:C2

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:3
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:C2 C010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
NONE HAS BEEN FOUND SO DBMS IS NOW SEARCHING FOR
RECORD TYPE ACNT VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
```

```
***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4 ARE:-
   22
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4
HAVE BEEN ALTERED TO
   32
***'MODIFY' COMPLETED***
TRANSACTION OF $   10 POSTED              -NEW CUSTOMER BALANCE     32
***END OF TRANSACTION POSTING RUN***



Data:C2

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:3
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:C1 0010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  2 VERSION  2
***'FIND1' COMPLETED***

*** FOLLOWING INFORMATION IS FROM 'FIND3' ***
DBMS IS FOLLOWING TAG TO EARLIER VERSION OF CURRENT RECORD AT ADDRESS
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
*** 'FIND3' COMPLETED ***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS 10 ARE:-
   11
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS 10
HAVE BEEN ALTERED TO
   21
***'MODIFY' COMPLETED***
TRANSACTION OF $   10 POSTED              -NEW ACCOUNT BALANCE     21

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 3
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS 10
IS  1
VERSION OF RECORD TYPE CUST AT THIS ADDRESS IS  2
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  7
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
   55
[illegible]
```

```
***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  3 TO  7
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
HAVE BEEN ALTERED TO
   65
***'MODIFY' COMPLETED***
TRANSACTION OF $   10 POSTED          -NEW CUSTOMER BALANCE     65
***END OF TRANSACTION POSTING RUN***


Data:01

***RUN OF BALANCE CALCULATION PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:2
TYPE CUSTOMER NUMBER
Data:C2

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
NONE HAS BEEN FOUND SO DBMS IS NOW SEARCHING FOR
RECORD TYPE CUST VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
RECORD FOUND AT ADDRESS  3 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  6
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3 ARE:-
SMIT
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 10
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3 ARE:-
H,JA
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS 11 TO 14
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3 ARE:-
MES
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE CUST VERSION  1 AT ADDRESS  3
IS  4
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4 ARE:-
   32
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4
IS  5
```

```
***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  5 ARE:-
   33
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  5
IS  0
***'FIND2' COMPLETED***
BALANCE FOR CUSTOMER  2 SMITH,JAMES  IS   65
***END OF BALANCE CALCULATION RUN***
```

```
Data:01

***RUN OF BALANCE CALCULATION PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:2
TYPE CUSTOMER NUMBER
Data:01

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO  -
 1
RECORD FOUND AT ADDRESS  1 VERSION  2
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  8 TO 11
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
JONE
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS 12 TO 15
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
S,AL
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS 16 TO 19
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
AN
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  7
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
   65
**'GET' COMPLETED***
BALANCE FOR CUSTOMER  1 JONES,ALAN   IS   65
***END OF BALANCE CALCULATION RUN***
```

```
Data:28
RECORD TYPE CUST VERSION  1 FOUND
AT ADDRESS  3
POINTER  1 GIVES ADDRESS OF FIRST MEMBER AS  4
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4 ARE :-
  32

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  4
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 2
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 2
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 2
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 1
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 2
CHARACTERS  7 TO  8 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
POINTER  1 GIVES ADDRESS OF NEXT MEMBER AS  5
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  5 ARE :-
  33

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  5
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 3
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 2
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 2
CHARACTERS 12 TO 13 WERE PREVIOUSLY
DA
CHARACTERS  1 TO  2 HAVE BEEN SET TO :-
 3
CHARACTERS  3 TO  4 HAVE BEEN SET TO :-
 2
CHARACTERS  5 TO  6 HAVE BEEN SET TO :-
 2
CHARACTERS  7 TO  8 HAVE BEEN SET TO :-
DA
*** RECORD HAS BEEN RESTRUCTURED ***
POINTER  1 GIVES ADDRESS OF NEXT MEMBER AS  0
```

```
*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE CUST AT ADDRESS   3
IS BEING ALTERED TO  2
CHARACTERS   1 TO  2 WERE PREVIOUSLY
 2
CHARACTERS   3 TO  6 WERE PREVIOUSLY
SMIT
CHARACTERS   7 TO 10 WERE PREVIOUSLY
H,JA
CHARACTERS  11 TO 14 WERE PREVIOUSLY
MES
CHARACTERS   1 TO  2 HAVE BEEN SET TO :-
 2
CHARACTERS   3 TO  7 ARE BEING SET TO SPACES
CHARACTERS   8 TO 11 HAVE BEEN SET TO :-
SMIT
CHARACTERS  12 TO 15 HAVE BEEN SET TO :-
H,JA
CHARACTERS  16 TO 19 HAVE BEEN SET TO :-
MES
*** RECORD HAS BEEN RESTRUCTURED ***
CHARACTERS   3 TO  7
OF RECORD TYPE CUST VERSION  2 AT ADDRESS   3
HAVE BEEN ALTERED TO
   65
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 28


Data:28
*** RESTRUCTURING COMPLETE - PARALLEL RUN CONTINUES ***


Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  2
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BALC START AT  3 END AT  7 FORMAT ,I5)
DATA ITEM NAM1 START AT  8 END AT 11 FORMAT ,A4)
DATA ITEM NAM2 START AT 12 END AT 15 FORMAT ,A4)
DATA ITEM NAM3 START AT 16 END AT 19 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM
```

```
RECORD TYPE ACNT VERSION  2
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BRNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM ACTP START AT  7 END AT  8 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BRNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                             POINTERS             TA
--- ------ ---- -------------------------------  -------------------  --
 1   CUST    2   1    65JONES,ALAN                2 6 0 0 0 0 0 0 0 0  9
 2   ACNT    2   1 1  1CA                         6 0 1 0 0 0 0 0 0 0  10
 3   CUST    2   2    65SMITH,JAMES               4 5 0 0 0 0 0 0 0 0  12
 4   ACNT    2   2 1  2CA                         5 0 3 0 0 0 0 0 0 0  13
 5   ACNT    2   3 2  2DA                         0 4 3 0 0 0 0 0 0 0  14
 6   ACNT    2   4 1  1CA                         0 2 1 0 0 0 0 0 0 0  11
 7   BRCH    1   1BIGTOWN     U                   0 0 0 0 0 0 0 0 0 0  0
 8   BRCH    1   2SMALLTOWN   R                   0 0 0 0 0 0 0 0 0 0  0
 9   CUST    1   1JONES,ALAN                      2 6 0 0 0 0 0 0 0 0  1
10   ACNT    1   1 1 1    21CA                    6 0 1 0 0 0 0 0 0 0  2
11   ACNT    1   4 1 1    44CA                    0 2 1 0 0 0 0 0 0 0  6
12   CUST    1   2SMITH,JAMES                     4 5 0 0 0 0 0 0 0 0  3
13   ACNT    1   2 1 2    32CA                    5 0 3 0 0 0 0 0 0 0  4
14   ACNT    1   3 2 2    33DA                    0 4 3 0 0 0 0 0 0 0  5
```

```
Data:10

***RUN OF BALANCE CALCULATION AUDIT PROGRAM***
TYPE CUSTOMER NUMBER
Data:01

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  1 VERSION  2
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  7
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
   65
***'GET' COMPLETED***
CUSTOMER BALANCE FROM VER 2 RECORD IS     65

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
IS  2
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  2
***'FIND2' COMPLETED***

*** FOLLOWING INFORMATION IS FROM 'FIND3' ***
DBMS IS FOLLOWING TAG TO EARLIER VERSION OF CURRENT RECORD AT ADDRESS 10
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
*** 'FIND3' COMPLETED ***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS 10 ARE:-
   21
***'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS 10
IS  6
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  2
***'FIND2' COMPLETED***

*** FOLLOWING INFORMATION IS FROM 'FIND3' ***
DBMS IS FOLLOWING TAG TO EARLIER VERSION OF CURRENT RECORD AT ADDRESS 1
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
*** 'FIND3' COMPLETED ***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS 11 ARE:-
   44
***'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS 11
IS  0
***'FIND2' COMPLETED***
SUM OF ACCOUNT BALANCES FOR VER 1 RECORDS IS    65
***END OF BALANCE CALCULATION AUDIT RUN***
```

```
Data:24
*** PARALLEL RUN COMPLETE ***



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                                     POINTERS              TAG
--- ------ ---- -------------------------------------- -------------------   ---
 1   CUST   2    1    65JONES,ALAN                       2 6 0 0 0 0 0 0 0 0   0
 2   ACNT   2    1 1 1CA                                 6 0 1 0 0 0 0 0 0 0   0
 3   CUST   2    2    65SMITH,JAMES                      4 5 0 0 0 0 0 0 0 0   0
 4   ACNT   2    2 1 2CA                                 5 0 3 0 0 0 0 0 0 0   0
 5   ACNT   2    3 2 2DA                                 0 4 3 0 0 0 0 0 0 0   0
 6   ACNT   2    4 1 1CA                                 0 2 1 0 0 0 0 0 0 0   0
 7   ERCH   1    1BIGTOWN     U                          0 0 0 0 0 0 0 0 0 0   0
 8   ERCH   1    2SMALLTOWN   R                          0 0 0 0 0 0 0 0 0 0   0



Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  2
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BALC START AT  3 END AT  7 FORMAT ,I5)
DATA ITEM NAM1 START AT  8 END AT 11 FORMAT ,A4)
DATA ITEM NAM2 START AT 12 END AT 15 FORMAT ,A4)
DATA ITEM NAM3 START AT 16 END AT 19 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  2
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BRNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM ACTP START AT  7 END AT  8 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE ERCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST
```

```
Data:C2

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:3
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:C1 0010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  2 VERSION  2
***'FIND1' COMPLETED***

*** FOLLOWING INFORMATION IS FROM 'FIND3' ***
DBMS IS FOLLOWING TAG TO EARLIER VERSION OF CURRENT RECORD AT ADDRESS  0
*** 'FIND3' COMPLETED ***
NO VERSION 1 ACCOUNT RECORD FOUND

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 3
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2
IS  1
VERSION OF RECORD TYPE CUST AT THIS ADDRESS IS  2
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  7
 OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1 ARE:-
   65
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  3 TO  7
OF RECORD TYPE CUST VERSION  2 AT ADDRESS  1
HAVE BEEN ALTERED TO
    75
***'MODIFY' COMPLETED***
TRANSACTION OF $   10 POSTED - NEW CUSTOMER BALANCE $   75
***END OF TRANSACTION POSTING RUN***
```

```
C
C
C
C
C
  100 WRITE(6,7320)
 7320 FORMAT ('0***RUN OF BALANCE CALCULATION AUDIT PROGRAM***')
      WRITE (6,1005)
      READ (5,1002) ICUST
      IBAL1=0
C
C DML COMMAND FIND1 IS USED TO ESTABLISH A CURRENT RECORD OF RUN
C UNIT FOR A CUSTOMER RECORD WITH THE SUPPLIED CUSTOMER NUMBER
C
      CALL FIND1(KCUST,ICUST,ISPACE,ISPACE,IERR)
  101 IF (IERR.NE.1) GO TO 107
      WRITE(6,7007)ICUST
 7007 FORMAT (' CUSTOMER ',I2,' NOT FOUND ')
      GO TO 106
  107 IF (ICURVR.NE.2) GO TO 109
      CALL GET(KBALC)
      IBAL2=IUWA(1)
      WRITE(6,7015) IBAL2
 7015 FORMAT (' CUSTOMER BALANCE FROM VER 2 RECORD IS ',I5)
      CALL FIND2(KACNT,KCUAC,KFRST,IERR)
  102 IF (IERR.EQ.1) GO TO 105
      CALL FIND3(IERR)
      CALL GET(KBALC)
  104 IBAL1=IBAL1+IUWA(1)
      CALL FIND2 (KACNT,KCUAC,KNEXT,IERR)
  105 IF (IERR.NE.1) GO TO 102
      WRITE (6,7040)IBAL1
 7040 FORMAT(' SUM OF ACCOUNT BALANCES FOR VER 1 RECORDS IS',I5)
  106 WRITE(6,7321)
 7321 FORMAT(' ***END OF BALANCE CALCULATION AUDIT RUN***')
      GO TO 5
  109 WRITE(6,7010)
 7010 FORMAT(' CUSTOMER RECORD IS NOT AT VERSION 2')
      GO TO 106
C
C
C
```

Amendment to the Format of the Account Number (Key to the Account Record)

Particular considerations apply when the format of a key data item is altered. In this case the account number in the account record is amended from format I2 to I3. As before, a closed restructuring strategy is used and each Account Record is accessed and restructured in address sequence.

When the Transaction Posting program is run to post £10 to account 2 the DBMS uses the schema definition of the version 2 account record to establish that it must match using a three-digit number. The record is found at address 2 in this way and the balance is modified as normal.

When posting £10 to account 03 the DBMS firstly searches for a version 2 account record using a three-digit account number. In this case no match is found and the schema is examined again so that a second search can be made for a version 1 record using a two-digit account number. In this case the record is found. Since the record is to be modified it is restructured before this is done.

When the restructuring proceeds it finds Account 3 already restructured. Account 4 is then restructured and the restructuring is complete.

```
Data:25
***START OF AMEND DATA ITEM FORMAT RESTRUCTURING ***
TYPE OPEN OR CLSD FOR TYPE OF RESTRUCTURING
FOLLOWED BY RECORD NAME , DATA ITEM NAME, LENGTH AND NEW FORMAT
Data:CLSD ACNT ACNO C3 ,I2)
***CLOSED RESTRUCTURING NOW UNDER WAY***
RECORD TYPE ACNT VERSION  1 FOUND
AT ADDRESS  2

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  2
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 1
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 1
CHARACTERS  7 TO 11 WERE PREVIOUSLY
    11
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS  1 TO  3 HAVE BEEN SET TO :-
  1
CHARACTERS  4 TO  5 HAVE BEEN SET TO :-
 1
CHARACTERS  6 TO  7 HAVE BEEN SET TO :-
 1
CHARACTERS  8 TO 12 HAVE BEEN SET TO :-
    11
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21


Data:21
RECORD TYPE ACNT VERSION  1 FOUND
AT ADDRESS  4

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  4
IS BEING ALTERED TO  2
CHARACTERS  1 TO  2 WERE PREVIOUSLY
 2
CHARACTERS  3 TO  4 WERE PREVIOUSLY
 1
CHARACTERS  5 TO  6 WERE PREVIOUSLY
 2
CHARACTERS  7 TO 11 WERE PREVIOUSLY
    22
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS  1 TO  3 HAVE BEEN SET TO :-
  2
CHARACTERS  4 TO  5 HAVE BEEN SET TO :-
 1
CHARACTERS  6 TO  7 HAVE BEEN SET TO :-
 2
CHARACTERS  8 TO 12 HAVE BEEN SET TO :-
    22
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21
```

```
Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  2
DATA ITEM ACNO START AT  1 END AT  3 FORMAT ,I3)
DATA ITEM ERNO START AT  4 END AT  5 FORMAT ,I2)
DATA ITEM CUNO START AT  6 END AT  7 FORMAT ,I2)
DATA ITEM BALC START AT  8 END AT 12 FORMAT ,I5)
DATA ITEM ACTP START AT 13 END AT 14 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM ERNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                               POINTERS              TA
--- ------ ---- ------------------------------     -------------------   --
 1   CUST    1   1JONES,ALAN                        2 6 0 0 0 0 0 0 0 0   0
 2   ACNT    2    1 1 1    11CA                     6 0 1 0 0 0 0 0 0 0   0
 3   CUST    1   2SMITH,JAMES                       4 5 0 0 0 0 0 0 0 0   0
 4   ACNT    2    2 1 2    22CA                     5 0 3 0 0 0 0 0 0 0   0
 5   ACNT    1   3 2 2     33CA                     0 4 3 0 0 0 0 0 0 0   0
 6   ACNT    1   4 1 1     44CA                     0 2 1 0 0 0 0 0 0 0   0
 7   BRCH    1   1BRIGHTON      U                   0 0 0 0 0 0 0 0 0 0   0
```

```
Data:02

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:02 0010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  2
WITH CHARACTERS  1 TO  3 EQUAL TO
  2
RECORD FOUND AT ADDRESS  4 VERSION  2
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  8 TO 12
 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  4 ARE:-
   22
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  8 TO 12
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  4
HAVE BEEN ALTERED TO
   32
***'MODIFY' COMPLETED***
TRANSACTION OF $    10 POSTED - NEW BALANCE $    32
***END OF TRANSACTION POSTING RUN***
```

```
Data:02

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:03 0010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  2
WITH CHARACTERS  1 TO  3 EQUAL TO
  3
NONE HAS BEEN FOUND SO DBMS IS NOW SEARCHING FOR
RECORD TYPE ACNT VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 3
RECORD FOUND AT ADDRESS  5 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  5 ARE:-
   33
**'GET' COMPLETED***
```

```
***FOLLOWING INFORMATION IS FROM 'MODIFY'***

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS   5
IS BEING ALTERED TO   2
CHARACTERS   1 TO   2 WERE PREVIOUSLY
 3
CHARACTERS   3 TO   4 WERE PREVIOUSLY
 2
CHARACTERS   5 TO   6 WERE PREVIOUSLY
 2
CHARACTERS   7 TO 11 WERE PREVIOUSLY
    33
CHARACTERS 12 TO 13 WERE PREVIOUSLY
DA
CHARACTERS   1 TO   3 HAVE BEEN SET TO :-
 3
CHARACTERS   4 TO   5 HAVE BEEN SET TO :-
 2
CHARACTERS   6 TO   7 HAVE BEEN SET TO :-
 2
CHARACTERS   8 TO 12 HAVE BEEN SET TO :-
    33
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
DA
*** RECORD HAS BEEN RESTRUCTURED ***
CHARACTERS   8 TO 12
OF RECORD TYPE ACNT VERSION   2 AT ADDRESS   5
HAVE BEEN ALTERED TO
    43
***'MODIFY' COMPLETED***
TRANSACTION OF $    10 POSTED - NEW BALANCE $    43
***END OF TRANSACTION POSTING RUN***



Data:21
RECORD TYPE ACNT VERSION   2 FOUND
AT ADDRESS   5
THIS RECORD HAS ALREADY BEEN RESTRUCTURED AND NEED NOT BE ALTERED
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21



Data:21
RECORD TYPE ACNT VERSION   1 FOUND
AT ADDRESS   6

*** FOLLOWING INFORMATION IS FROM RESTRUCTURING ROUTINE ***
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS   6
IS BEING ALTERED TO   2
CHARACTERS   1 TO   2 WERE PREVIOUSLY
 4
CHARACTERS   3 TO   4 WERE PREVIOUSLY
 1
CHARACTERS   5 TO   6 WERE PREVIOUSLY
 1
CHARACTERS   7 TO 11 WERE PREVIOUSLY
    44
CHARACTERS 12 TO 13 WERE PREVIOUSLY
CA
CHARACTERS   1 TO   3 HAVE BEEN SET TO :-
  4
CHARACTERS   4 TO   5 HAVE BEEN SET TO :-
 1
```

```
CHARACTERS  8 TO 12 HAVE BEEN SET TO :-
   44
CHARACTERS 13 TO 14 HAVE BEEN SET TO :-
CA
*** RECORD HAS BEEN RESTRUCTURED ***
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 21


Data:21
*** CLOSED RESTRUCTURING COMPLETE ***



Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  2
DATA ITEM ACNO START AT  1 END AT  3 FORMAT ,I3)
DATA ITEM BRNO START AT  4 END AT  5 FORMAT ,I2)
DATA ITEM CUNO START AT  6 END AT  7 FORMAT ,I2)
DATA ITEM BALC START AT  8 END AT 12 FORMAT ,I5)
DATA ITEM ACTP START AT 13 END AT 14 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                                        POINTERS               TAG
--- ------ ---- ------------------------------------------ --------------------  ---
 1   CUST    1   1JONES,ALAN                                2 6 0 0 0 0 0 0 0 0    0
 2   ACNT    2    1 1 1   11CA                              6 0 1 0 0 0 0 0 0 0    0
 3   CUST    1   2SMITH,JAMES                               4 5 0 0 0 0 0 0 0 0    0
 4   ACNT    2    2 1 2   32CA                              5 [illegible] 3 0 0 0 0 0 0 0    0
 5   ACNT    2    3 2 2   43DA                              0 4 3 0 0 0 0 0 0 0    0
 6   ACNT    2    4 1 1   44CA                              0 2 1 0 0 0 0 0 0 0    0
 7   BRCH    1   1BIGTOWN    U                              0 0 0 0 0 0 0 0 0 0    0
     [illegible] 1   [illegible]
```

Addition of the Account Type to the Account Record Key

The key to a record may be amended by a restructuring initiated by the user entering a code of 25 on his terminal. The DBMS amends the schema entry for the record in question to reflect the revised key data items provided by the user. In this implementation this is all that is required, there being no need to increase record version numbers.

The example shows a run of the Transaction Posting program at Version 1 to post £10 to account 2. After the restructuring, version 4 of this program is introduced. This requests not only the account number but also the account type from the user to determine the account to receive the posting. The FIND1 command then carries out a sequential search of the data base for a record with the required values in both data items.

```
Data:02

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:02 0010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
RECORD FOUND AT ADDRESS  4 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4 ARE:-
    22
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4
HAVE BEEN ALTERED TO
    32
***'MODIFY' COMPLETED***
TRANSACTION OF $    10 POSTED - NEW BALANCE $    32
***END OF TRANSACTION POSTING RUN***



Data:25

***START OF AMEND RECORD KEY RESTRUCTURING ***
TYPE RECORD NAME AND REVISED KEY DATA ITEMS
Data:ACNT ACNO ACTP
*** RESTRUCTURING COMPLETE ***



Data:02

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:4
TYPE ACCOUNT NUMBER, ACCOUNT TYPE AND VALUE TO BE POSTED
Data:02 CA  0010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 2
AND CHARACTERS 12 TO 13 EQUAL TO
CA
RECORD FOUND AT ADDRESS  4 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4 ARE:-
```

```
***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4
HAVE BEEN ALTERED TO
    42
***'MODIFY' COMPLETED***
TRANSACTION OF $    10 POSTED - NEW BALANCE $     42
***END OF TRANSACTION POSTING RUN***
```

Data:29

```
*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BRNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO
KEY DATA ITEM ACTP

RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST
```

Data:30

```
*** THE CURRENT DATA BASE ***

                                                                  POINTERS
ADD RECORD VRSN DATA                                              -------------------
--- ------ ---- ------------------------------------------------  2 6 0 0 0 0 0 0 0 0
 1   CUST    1   1JONES,ALAN                                      6 0 1 0 0 0 0 0 0 0
 2   ACNT    1   1  1  1      11CA                                4 5 0 0 0 0 0 0 0 0
 3   CUST    1   2SMITH,JAMES                                     5 0 3 0 0 0 0 0 0 0
 4   ACNT    1   2  1  2      42CA                                0 4 3 0 0 0 0 0 0 0
 5   ACNT    1   3  2  2      33DA                                0 2 1 0 0 0 0 0 0 0
 6   ACNT    1   4  1  1      44CA                                0 0 0 0 0 0 0 0 0 0
 7   BRCH    1   1BIGTOWN        U                                0 0 0 0 0 0 0 0 0 0
     [illegible] 1  [illegible]SMALLTOWN  R
```

Addition of the Branch Accounts Set

To add a new set the user must enter a code of 17 on his terminal. He must then supply the following:

> The Name of the New Set
>
> The Name of the Owner Record
>
> The Name of a data item in the Owner Record to be used for matching
>
> The Name of the Member Record
>
> The Name of a data item in the Member Record to be used for matching
>
> The position at which new member records are to be added to the set.

In this example the new set is "Branch Accounts" with an owner of Branch (Matching Data Item Branch Number) and a member of Account (Matching Data Item Branch Number). New inserts are to be added first. The restructuring uses a closed strategy.

The first branch record on the data base is found as branch 1 at address 7. This is restructured to version 2 by the addition of two further pointers for the new set (this operation is trivial in this case since the original version of the record had no pointers defined) both with a value of zero. Thus at this point there are no member records in the Branch Accounts set for branch 1. Similarly, the second Branch record is restructured.

Since there are no further Branch records on the data base a search is now made of each Account record in address sequence. The first record is for account 1 at address 2. Like the branch record this record is then restructured to accommodate three additional pointers for the new set. Initially, these are set to zero but the DBMS retrieves the value of the Branch Number data item (in much the same way as it does for a GET) and uses this value to find the Branch record with the same value in its matching Branch Number data item. In this case the Branch Number is 1 and the Branch record is found at address 7. Pointers on both records are then modified to reflect their tenancy of the new set.

The Transaction Posting program can still be run at version 1 to post £10 to account 1 even though this record is at version 2. Similarly, a run of the Calculate Balance program has no difficulty in operating on the account 1 record (at version 2) and on the Account 4 record (at version 1).

Further account records are then accessed at addresses 4, 5 and 6 and these are restructured to version 2. Where necessary the pointers on any other existing version 2 account record which is the previous first member of the set to which the new account is to be added are also amended.

When the restructuring is complete a new program is run to calculate the net balance for a branch by progressing through the "Branch Accounts" set.

```
Data:17

*** START OF ADD NEW SET RESTRUCTURING ***
RESTRUCTURING USES CLOSED STRATEGY
TYPE SET NAME ,OWNER RECORD TYPE & MATCHING DATA ITEM
MEMBER RECORD TYPE & MATCHING DATA ITEM
AND POSITION OF NEW INSERTS
Data:BRAC BRCH BNUM ACNT BRNO FRST
***CLOSED RESTRUCTURING NOW UNDER WAY***
POINTERS 1 TO 8 OF RECORD TYPE BRCH VERSION  1
AT ADDRESS  7 ARE
 0  0  0  0  0  0  0  0
VERSION NUMBER OF RECORD TYPE BRCH AT ADDRESS  7 HAS BEEN SET TO  2
POINTER  1 HAS BEEN SET TO  0
POINTER  2 HAS BEEN SET TO  0
POINTER  3 HAS BEEN SET TO  0
POINTER  4 HAS BEEN SET TO  0
POINTER  5 HAS BEEN SET TO  0
POINTER  6 HAS BEEN SET TO  0
POINTER  7 HAS BEEN SET TO  0
POINTER  8 HAS BEEN SET TO  0
POINTER  9 HAS BEEN SET TO  0
POINTER 10 HAS BEEN SET TO  0
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 18


Data:18
POINTERS 1 TO 8 OF RECORD TYPE BRCH VERSION  1
AT ADDRESS  8 ARE
 0  0  0  0  0  0  0  0
VERSION NUMBER OF RECORD TYPE BRCH AT ADDRESS  8 HAS BEEN SET TO  2
POINTER  1 HAS BEEN SET TO  0
POINTER  2 HAS BEEN SET TO  0
POINTER  3 HAS BEEN SET TO  0
POINTER  4 HAS BEEN SET TO  0
POINTER  5 HAS BEEN SET TO  0
POINTER  6 HAS BEEN SET TO  0
POINTER  7 HAS BEEN SET TO  0
POINTER  8 HAS BEEN SET TO  0
POINTER  9 HAS BEEN SET TO  0
POINTER 10 HAS BEEN SET TO  0
IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 18


Data:18
POINTERS 1 TO 7 OF RECORD TYPE ACNT VERSION  1
AT ADDRESS  2 ARE
 6  0  1  0  0  0  0
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  2 HAS BEEN SET TO  2
POINTER  1 HAS BEEN SET TO  0
POINTER  2 HAS BEEN SET TO  0
POINTER  3 HAS BEEN SET TO  0
POINTER  4 HAS BEEN SET TO  0
POINTER  5 HAS BEEN SET TO  0
POINTER  6 HAS BEEN SET TO  1
POINTER  7 HAS BEEN SET TO  0
POINTER  8 HAS BEEN SET TO  0
POINTER  9 HAS BEEN SET TO  0
POINTER 10 HAS BEEN SET TO  0
CHAR[illegible] T[illegible] 2
```

```
DBMS IS SEARCHING FOR
RECORD TYPE BRCH VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO :-
 1
RECORD FOUND AT ADDRESS  7
POINTER  1 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2
HAS BEEN ALTERED TO  0
POINTER  3 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2
HAS BEEN ALTERED TO  7
POINTER  1 OF RECORD TYPE BRCH VERSION  2 AT ADDRESS  7
HAS BEEN ALTERED TO  2
POINTER  2 OF RECORD TYPE BRCH VERSION  2 AT ADDRESS  7
HAS BEEN ALTERED TO  2
 IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 19



Data:29

*** THE CURRENT SCHEMA ***

RECORD TYPE CUST VERSION  1
DATA ITEM CNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM NAM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM NAM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM NAM3 START AT 11 END AT 14 FORMAT ,A4)
SET NAME CUAC POINTER TYPE FRST
SET NAME CUAC POINTER TYPE LAST
KEY DATA ITEM CNUM

RECORD TYPE ACNT VERSION  2
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BRNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME BRAC POINTER TYPE NEXT
SET NAME BRAC POINTER TYPE PRIR
SET NAME BRAC POINTER TYPE OWNR
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE ACNT VERSION  1
DATA ITEM ACNO START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BRNO START AT  3 END AT  4 FORMAT ,I2)
DATA ITEM CUNO START AT  5 END AT  6 FORMAT ,I2)
DATA ITEM BALC START AT  7 END AT 11 FORMAT ,I5)
DATA ITEM ACTP START AT 12 END AT 13 FORMAT ,A2)
SET NAME CUAC POINTER TYPE NEXT
SET NAME CUAC POINTER TYPE PRIR
SET NAME CUAC POINTER TYPE OWNR
KEY DATA ITEM ACNO

RECORD TYPE BRCH VERSION  2
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
SET NAME BRAC POINTER TYPE FRST
SET NAME BRAC POINTER TYPE LAST
KEY [illegible]
```

```
RECORD TYPE BRCH VERSION  1
DATA ITEM BNUM START AT  1 END AT  2 FORMAT ,I2)
DATA ITEM BNM1 START AT  3 END AT  6 FORMAT ,A4)
DATA ITEM BNM2 START AT  7 END AT 10 FORMAT ,A4)
DATA ITEM BNM3 START AT 11 END AT 14 FORMAT ,A4)
DATA ITEM LOCN START AT 15 END AT 15 FORMAT ,A1)
KEY DATA ITEM BNUM

SET NAME CUAC
OWNER RECORD TYPE CUST MATCHING DATA ITEM CNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM CUNO
POSITION OF NEW INSERTS - LAST

SET NAME BRAC
OWNER RECORD TYPE BRCH MATCHING DATA ITEM BNUM
MEMBER RECORD TYPE ACNT MATCHING DATA ITEM BRNO
POSITION OF NEW INSERTS - FRST



Data:30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                                    POINTERS                 TAG
--- ------ ---- ------------------------------------   --------------------     ---
 1   CUST    1    1JONES,ALAN                          2 6 0 0 0 0 0 0 0 0       0
 2   ACNT    2    1 1 1    11CA                        0 0 7 6 0 1 0 0 0 0       0
 3   CUST    1    2SMITH,JAMES                         4 5 0 0 0 0 0 0 0 0       0
 4   ACNT    1    2 1 2    22CA                        5 0 3 0 0 0 0 0 0 0       0
 5   ACNT    1    3 2 2    33DA                        0 4 3 0 0 0 0 0 0 0       0
 6   ACNT    1    4 1 1    44CA                        0 2 1 0 0 0 0 0 0 0       0
 7   BRCH    2    1BIGTOWN     U                       2 2 0 0 0 0 0 0 0 0       0
 8   BRCH    2    2SMALLTOWN   R                       0 0 0 0 0 0 0 0 0 0       0



Data:02

***RUN OF TRANSACTION POSTING PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE ACCOUNT NUMBER AND VALUE TO BE POSTED
Data:01 0010

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE ACNT VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  2 VERSION  2
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2 ARE:-
   11
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'MODIFY'***
CHARACTERS  7 TO 11
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2
HAVE BEEN ALTERED TO
   21
***'MODIFY' COMPLETED***
TRANSACTION OF $   10 POSTED - NEW BALANCE $   21
***END OF TRANSACTION POSTING RUN***
```

```
Data:01

***RUN OF BALANCE CALCULATION PROGRAM***
TYPE VERSION NUMBER OF PROGRAM
Data:1
TYPE CUSTOMER NUMBER
Data:01

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE CUST VERSION  1
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  1 VERSION  1
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  6
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1 ARE:-
JONE
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 10
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1 ARE:-
S,AL
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS 11 TO 14
 OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1 ARE:-
AN
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE CUST VERSION  1 AT ADDRESS  1
IS  2
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  2
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2 ARE:-
   21
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 4
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2
IS  6
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  1
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  6 ARE:-
   44
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  6
IS [illegible]
***'FIND2' COMPLETED***
[illegible]
```

```
Data:19
POINTERS 1 TO 7 OF RECORD TYPE ACNT VERSION  1
AT ADDRESS  4 ARE
 5  0  3  0  0  0  0
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  4 HAS BEEN SET TO  2
POINTER  1 HAS BEEN SET TO  0
POINTER  2 HAS BEEN SET TO  0
POINTER  3 HAS BEEN SET TO  0
POINTER  4 HAS BEEN SET TO  5
POINTER  5 HAS BEEN SET TO  0
POINTER  6 HAS BEEN SET TO  3
POINTER  7 HAS BEEN SET TO  0
POINTER  8 HAS BEEN SET TO  0
POINTER  9 HAS BEEN SET TO  0
POINTER 10 HAS BEEN SET TO  0
CHARACTERS  3 TO  4
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  4 ARE :-
 1
DBMS IS SEARCHING FOR
RECORD TYPE BRCH VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO :-
 1
RECORD FOUND AT ADDRESS  7
RECORD-TYPE ACNT VERSION  2 FOUND AT ADDRESS  2
POINTER  2 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2
HAS BEEN ALTERED TO  4
POINTER  1 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  4
HAS BEEN ALTERED TO  2
POINTER  3 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  4
HAS BEEN ALTERED TO  7
POINTER  1 OF RECORD TYPE BRCH VERSION  2 AT ADDRESS  7
HAS BEEN ALTERED TO  4
 IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 19


Data:19
POINTERS 1 TO 7 OF RECORD TYPE ACNT VERSION  1
AT ADDRESS  5 ARE
 0  4  3  0  0  0  0
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  5 HAS BEEN SET TO  2
POINTER  1 HAS BEEN SET TO  0
POINTER  2 HAS BEEN SET TO  0
POINTER  3 HAS BEEN SET TO  0
POINTER  4 HAS BEEN SET TO  0
POINTER  5 HAS BEEN SET TO  4
POINTER  6 HAS BEEN SET TO  3
POINTER  7 HAS BEEN SET TO  0
POINTER  8 HAS BEEN SET TO  0
POINTER  9 HAS BEEN SET TO  0
POINTER 10 HAS BEEN SET TO  0
CHARACTERS  3 TO  4
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  5 ARE :-
 2
DBMS IS SEARCHING FOR
RECORD TYPE BRCH VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO :-
 2
RECORD FOUND AT ADDRESS  8
POINTER  1 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  5
HAS BEEN ALTERED TO  [illegible]
POINTER  3 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  5
HAS BEEN ALTERED TO  [illegible]
POINTER  1 OF RECORD TYPE BRCH VERSION  2 AT ADDRESS  [illegible]
HAS BEEN ALTERED TO  5
POINTER  [illegible] OF RECORD TYPE [illegible] VERSION  [illegible] AT ADDRESS  [illegible]
[illegible]
```

```
Data:19
POINTERS 1 TO 7 OF RECORD TYPE ACNT VERSION  1
AT ADDRESS  6 ARE
 0  2  1  0  0  0  0
VERSION NUMBER OF RECORD TYPE ACNT AT ADDRESS  6 HAS BEEN SET TO  2
POINTER  1 HAS BEEN SET TO  0
POINTER  2 HAS BEEN SET TO  0
POINTER  3 HAS BEEN SET TO  0
POINTER  4 HAS BEEN SET TO  0
POINTER  5 HAS BEEN SET TO  2
POINTER  6 HAS BEEN SET TO  1
POINTER  7 HAS BEEN SET TO  0
POINTER  8 HAS BEEN SET TO  0
POINTER  9 HAS BEEN SET TO  0
POINTER 10 HAS BEEN SET TO  0
CHARACTERS  3 TO  4
OF RECORD TYPE ACNT VERSION  1 AT ADDRESS  6 ARE :-
 1
DBMS IS SEARCHING FOR
RECORD TYPE BRCH VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO :-
 1
RECORD FOUND AT ADDRESS  7
RECORD-TYPE ACNT VERSION  2 FOUND AT ADDRESS  4
POINTER  2 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  4
HAS BEEN ALTERED TO  6
POINTER  1 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  6
HAS BEEN ALTERED TO  4
POINTER  3 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  6
HAS BEEN ALTERED TO  7
POINTER  1 OF RECORD TYPE BRCH VERSION  2 AT ADDRESS  7
HAS BEEN ALTERED TO  6
 IF NO OTHER REQUEST IS OUTSTANDING TYPE REPLY 19


Data:19
*** CLOSED RESTRUCTURING COMPLETE ***


Data:C6

*** RUN OF CALCULATE BRANCH BALANCE PROGRAM ***
YPE BRANCH NUMBER
Data:C1

***FOLLOWING INFORMATION IS FROM 'FIND1'***
DBMS IS SEARCHING FOR
RECORD TYPE BRCH VERSION  2
WITH CHARACTERS  1 TO  2 EQUAL TO
 1
RECORD FOUND AT ADDRESS  7 VERSION  2
***'FIND1' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  3 TO  6
 OF RECORD TYPE BRCH VERSION  2 AT ADDRESS  7 ARE:-
[illegible]
```

```
***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 10
 OF RECORD TYPE BRCH VERSION  2 AT ADDRESS  7 ARE:-
OWN
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS 11 TO 14
 OF RECORD TYPE BRCH VERSION  2 AT ADDRESS  7 ARE:-
   U
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE BRCH VERSION  2 AT ADDRESS  7
IS  6
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  2
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  6 ARE:-
   44
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  6
IS  4
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  2
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  4 ARE:-
   22
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  4
IS  2
VERSION OF RECORD TYPE ACNT AT THIS ADDRESS IS  2
***'FIND2' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'GET'***
CHARACTERS  7 TO 11
 OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2 ARE:-
   21
**'GET' COMPLETED***

***FOLLOWING INFORMATION IS FROM 'FIND2'***
VALUE IN POINTER 1
OF RECORD TYPE ACNT VERSION  2 AT ADDRESS  2
IS  0
***'FIND2' COMPLETED***
BALANCE FOR BRANCH  1 BIGTOWN    U IS    87
END OF BRANCH BALANCE CALCULATION RUN
```

```
C
C
C
C
   60 WRITE (6,1390)
 1390 FORMAT ('0*** RUN OF CALCULATE BRANCH BALANCE PROGRAM ***')
      WRITE (6,1391)
 1391 FORMAT ('TYPE BRANCH NUMBER')
      READ (5,1392) IBRAN
 1392 FORMAT (I2)
      IBAL=0
      CALL FIND1 (KBRCH,IBRAN,ISPACE,ISPACE,IERR)
   61 IF (IERR.NE.1) GO TO 62
      WRITE (6,1393)IBRAN
 1393 FORMAT (' BRANCH ',I2,' NOT FOUND')
      GO TO 66
   62 CALL GET (KBNM1)
   63 IWN1=IUWA(1)
      CALL GET(KBNM2)
   64 IWN2=IUWA(1)
      CALL GET(KBNM3)
   65 IWN3=IUWA(1)
      CALL FIND2 (KACNT,KBRAC,KFRST,IERR)
   66 IF (IERR.EQ.1) GO TO 68
      CALL GET(KBALC)
  660 IBAL=IBAL+IUWA(1)
      CALL FIND2(KACNT,KBRAC,KNEXT,IERR)
   68 IF (IERR.NE.1) GO TO 66
      WRITE (6,1394) IBRAN,IWN1,IWN2,IWN3,IBAL
 1394 FORMAT (' BALANCE FOR BRANCH ',I2,' ',3A4,' IS ',I5)
      WRITE (6,1395)
 1395 FORMAT (' END OF BRANCH BALANCE CALCULATION RUN')
      GO TO 5
C
C
C
C
   70 STOP 99
C
C   INPUT TYPE 08 ALLOWS A NEW CUSTOMER TO BE ADDED
```

Appendix 4 - Consumption of Computer Resources By the EMAS Implementation.

This appendix gives details of activities carried out on the version of the EMAS implementation of a Data Base Management System where the usage of computer resources are monitored and reported to the user. Resources required for different restructuring strategies may be compared.

The Demonstration Data.

The restructurings described here are based on a data base of 10 customers of the Bank. Each customer holds one or more account depending on the type of restructuring being demonstrated. In order to simulate a "worst possible" situation in terms of the clustering of the data all customer records are placed together on the data base but in the order 1,2,3,4,5,6,7,8,9,10. Thus a progression in key sequence will not correspond with a progression in physical placement sequence. Similarly, the account records are grouped together after all of the customer records. Once again this provides a useful distinction between logical and physical order but also (since customer 1 holds account 1, account 11, account 21 etc. and customer 2 holds account 2, account 12 etc.) the owner and member record occurrences of each set occurrence are invariably physically separate.

Figure 1 illustrates a 'Basic' Data Base where each customer has one account.

```
Data:30

 PLEASE TYPE START AND END ADDRESS
FOR DATA BASE PRINT
Data:    1  30

*** THE CURRENT DATA BASE ***

ADD RECORD VRSN DATA                                   POINTERS                              TAG
--- ------ ---- ---------------------------------------------------------------------------- ---
  1  CUST    1     1                                   11 11  0  0  0  0  0  0  0  0         0
  2  CUST    1     3                                   12 12  0  0  0  0  0  0  0  0         0
  3  CUST    1     5                                   13 13  0  0  0  0  0  0  0  0         0
  4  CUST    1     7                                   14 14  0  0  0  0  0  0  0  0         0
  5  CUST    1     9                                   15 15  0  0  0  0  0  0  0  0         0
  6  CUST    1     2                                   16 16  0  0  0  0  0  0  0  0         0
  7  CUST    1     4                                   17 17  0  0  0  0  0  0  0  0         0
  8  CUST    1     6                                   18 18  0  0  0  0  0  0  0  0         0
  9  CUST    1     8                                   19 19  0  0  0  0  0  0  0  0         0
 10  CUST    1    10                                   20 20  0  0  0  0  0  0  0  0         0
 11  ACNT    1     1  1  1     0CA                      0  0  1  0  0  0  0  0  0  0         0
 12  ACNT    1     3  1  3     0CA                      0  0  2  0  0  0  0  0  0  0         0
 13  ACNT    1     5  1  5     0CA                      0  0  3  0  0  0  0  0  0  0         0
 14  ACNT    1     7  1  7     0CA                      0  0  4  0  0  0  0  0  0  0         0
 15  ACNT    1     9  1  9     0CA                      0  0  5  0  0  0  0  0  0  0         0
 16  ACNT    1     2  1  2     0CA                      0  0  6  0  0  0  0  0  0  0         0
 17  ACNT    1     4  1  4     0CA                      0  0  7  0  0  0  0  0  0  0         0
 18  ACNT    1     6  1  6     0CA                      0  0  8  0  0  0  0  0  0  0         0
 19  ACNT    1     8  1  8     0CA                      0  0  9  0  0  0  0  0  0  0         0
 20  ACNT    1    10  1 10     0CA                      0  0 10  0  0  0  0  0  0  0         0
```

Pointer On Owner Record To FIRST Member Record Of Customer's Accounts Set

Pointer To LAST Member Of Set

Customer Number

Account Number

Branch Number

Customer Number

Balance (5 Digits)

Account Type

Pointer To NEXT Member Record Of Customer's Accounts Set (No Other Members As Yet)

Pointer To PRIOR Record (None As Yet)

Pointer To OWNER Record Of Customer's Accounts Set

Figure 1

"Off-line" Static Restructuring.

A program has been written to load the basic data base with each account record having a balance of zero. A simple, conventional application program, it uses the CODASYL DML commands to STORE the 10 customer records, STORE the 10 account records and INSERT each account record into the appropriate set occurrence of the customer's-accounts set based on the customer number on that record. The consumption of resources by the Load program gives some measure of what would be required by the reload phase of an "off-line" Static Restructuring. For such a restructuring the amendment to record format would take place on the sequential back-up copy of the data base. The unload and reformatting phase would also require computer resources

Figure 2 shows a listing of the program to store 10 customer records and the store 10 account records and insert them into the appropriate set.

Figure 3 shows a run of the program. The Input/Output activity consists of (a) 10 Stores of Customer records which require 10 Prime Data File Writes and 10 Index Writes, (b) 10 stores of Account records which will also require 10 Prime Data File Writes and 10 Index Writes and (c) 10 Inserts of the Account records into the Customer's Accounts sets which will require 10 Prime Data File Reads to retrieve the owner Customer records together with 20 Prime Data File Writes to update both Customer and Account records with modified set pointers.

Thus there are 100 I/O operations which are reflected in 287 page turns and 6.12 seconds of CPU usage.

Figure 2
(Program Listing)

```
 110 WRITE (6,7416)
     DO 9110 I=1,20
     ISTATS(I)=0
9110 CONTINUE
     CALL EMASFC('METER',5,'DUMMY',0)
7416 FORMAT('0*** RUN OF DATA BASE LOAD PROGRAM ***')
     DO 111 I=1,9,2
     IUWA(1)=I
     IUWA(2)=ISPACE
     IUWA(3)=ISPACE
     IUWA(4)=ISPACE
     CALL STORE(KCUST,KSUBS2)
 111 CONTINUE
     DO 112 I=2,10,2
     IUWA(1)=I
     IUWA(2)=ISPACE
     IUWA(3)=ISPACE
     IUWA(4)=ISPACE
     CALL STORE(KCUST,KSUBS2)
 112 CONTINUE
     DO 113 I=1,9,2
     IUWA(1)=1
     IUWA(2)=I
     IUWA(3)=I
     IUWA(4)=KCA
     IUWA(5)=0
     CALL STORE (KACNT,KSUBSC)
     CALL INSRT (KCUAC)
 113 CONTINUE
     DO 114 I=2,10,2
     IUWA(1)=1
     IUWA(2)=I
     IUWA(3)=I
     IUWA(4)=KCA
     IUWA(5)=0
     CALL STORE (KACNT,KSUBSC)
     CALL INSRT (KCUAC)
 114 CONTINUE
 135 WRITE(6,7417)
7417 FORMAT('0*** END OF DATA BASE LOAD RUN ***')
     CALL EMASFC('METER',5,'DUMMY',0)
     WRITE(6,9125)(ISTATS(I),I=1,8)
     GO TO 5
```

```
Data:11

*** RUN OF DATA BASE LOAD PROGRAM ***
09/05/83 22.07.22  CPU= 25.54 Secs CT= 9 Mins PT= 2644 Ch= 176p

*** END OF DATA BASE LOAD RUN ***
09/05/83 22.09.30  CPU= 31.66 Secs CT= 11 Mins PT= 2931 Ch= 215p
FIND1 CALLS    0
FIND2 CALLS    0
FIND3 CALLS    0
GET  CALLS    0
MODIFY CALLS    0
STORE CALLS  20
INSERT CALLS  10
RECORDS RESTRUCTURED    0
INDEX READS   0
INDEX WRITES  20
PRIME DATA READS  10
PRIME DATA WRITES  40
```

Figure 3

"Off-Line" Static Restructuring Reload Step

In-Place Static Restructuring.

A Closed Dynamic Restructuring to alter the Balance data item from 5 to 4 digits was run without an interruption by a request for access by an application program. Figure 4 shows the run.

This run required 20 reads from the Prime Data File (in physical placement sequence) to cover the 10 Customer and 10 Account records. The 10 Account records are modified and written back to the Prime Data File. A total of 30 I/O operations.

The run required 298 page turns and 5.16 seconds of CPU time. The same order of magnitude as the off-line reload operation.

```
Data:23

***START OF AMEND DATA ITEM FORMAT RESTRUCTURING ***
09/05/83 22.10.10  CPU= 31.69 Secs CT= 11 Mins PT= 3036 Ch= 216p
TYPE OPEN OR CLSD FOR TYPE OF RESTRUCTURING
FOLLOWED BY RECORD NAME , DATA ITEM NAME, LENGTH AND NEW FORMAT
Data:CLSD ACNT BALC  4 ,I4)
***CLOSED RESTRUCTURING NOW UNDER WAY***
*** CLOSED RESTRUCTURING COMPLETE ***
09/05/83 22.13.00  CPU= 36.85 Secs CT= 14 Mins PT= 3334 Ch= 250p
FIND1 CALLS    0
FIND2 CALLS    0
FIND3 CALLS    0
GET  CALLS    0
MODIFY CALLS    0
STORE CALLS    0
INSERT CALLS    0
RECORDS RESTRUCTURED  10
INDEX READS    0
INDEX WRITES    0
PRIME DATA READS   20
PRIME DATA WRITES  10
```

Figure 4
In Place Static
Restructuring

The "Add New Accounts" Program.

A demonstration application program has been written to add a new account record for each customer by STOREing at the next available free address on the data base and then INSERTing it into the appropriate occurrence of the Customer's Accounts Set.

Figure 5 gives a listing of the program and a run is shown in Figure 6.

The run required 180 page turns and used 3.18 seconds of CPU time. The consumption of resources is therefore approximately half of that required for a static restructuring of a similar number of account records.

```
C
  120 WRITE (6,7418)
      DO 9120 I=1,20
      ISTATS(I)=0
 9120 CONTINUE
 7418 FORMAT ('*** RUN OF ADD NEW ACCOUNTS PROGRAM ***')
      CALL EMASFC ('METER',5,'DUMMY',0)
      DO 115 I=11,19,2
      IUWA(1)=1
      IUWA(2)=I
      IUWA(3)=I-10
      IUWA(4)=KCA
      IUWA(5)=0
      CALL STORE (KACNT,KSUBSC)
      CALL INSRT (KCUAC)
  115 CONTINUE
      DO 116 I=12,20,2
      IUWA(1)=1
      IUWA(2)=I
      IUWA(3)=I-10
      IUWA(4)=KCA
      IUWA(5)=0
      CALL STORE (KACNT,KSUBSC)
      CALL INSRT (KCUAC)
  116 CONTINUE
      WRITE (6,7419)
 7419 FORMAT ('0*** END OF ADD NEW ACCOUNTS RUN ***')
      CALL EMASFC ('METER',5,'DUMMY',0)
      WRITE (6,9125)(ISTATS(I),I=1,12)
 9125 FORMAT (' FIND1 CALLS ',I3,
     */' FIND2 CALLS ',I3,
     */' FIND3 CALLS ',I3,
     */,' GET  CALLS ',I3,
     */,' MODIFY CALLS ',I3,
     */,' STORE CALLS ',I3,
     */,' INSERT CALLS ',I3,
     */,' RECORDS RESTRUCTURED ',I3,
     */,' INDEX READS ',I3,
     */,' INDEX WRITES ',I3,
     */,' PRIME DATA READS ',I3,
     */,' PRIME DATA WRITES ',I3)

      GO TO 5
```

Figure 5

Program Listing The "Open New Accounts" Program

```
Data:12
** RUN OF ADD NEW ACCOUNTS PROGRAM ***
03/05/83 22.00.03  CPU= 34.21 Secs CT= 34 Mins PT= 5503 Ch= 252p

*** END OF ADD NEW ACCOUNTS RUN ***
03/05/83 22.01.16  CPU= 38.02 Secs CT= 35 Mins PT= 5683 Ch= 277p
FIND1 CALLS    0
FIND2 CALLS    0
FIND3 CALLS    0
GET  CALLS    0
MODIFY CALLS    0
STORE CALLS  10
INSERT CALLS  10
RECORDS RESTRUCTURED    0
INDEX READS    0
INDEX WRITES  10
PRIME DATA READS  20
PRIME DATA WRITES  40
```

Figure 6

Run of The "Add New Accounts" Program

<u>The Closed Restructuring While Adding New Accounts.</u>

A Closed Restructuring to reduce the length of the balance data item on existing account records from 5 to 4 bytes was performed such that after each record was restructured an application program was invoked to add a new account record to the data base. This run is shown in figure 7.

The effect of this operation is therefore a combination of the "in-place" Static Restructuring and the "Add New Accounts" application program and the total consumption of resources is approximately equal to the sum of these individual tasks. (433 page turns and 8.93 seconds of CPU usage)

The situation demonstrated by this run is the worst possible which can be encountered by a Closed Restructuring - it is interrupted each time a record is restructured and must add a new record at another point in the data base. A queue of application program requests for access could be serviced during this interruption without further degrading the response time. The overall increase in consumption of resources between the run of the "Add New Accounts" program and the addition of the accounts during a closed restructuring is of the order of 120% and there is a corresponding increase in I/O activity (from 90 operations to 140). If this figure is considered as an upper bound of the increase in response time to which any application program would be subjected (because of the untypical nature of the implementation) this should give some credence to the view that a Closed Restructuring Strategy would not impose intolerable overheads on a full scale Database Management System.

```
Data:23

***START OF AMEND DATA ITEM FORMAT RESTRUCTURING ***
09/05/83 22.20.43  CPU= 67.64 Secs CT= 22 Mins PT= 5686 Ch= 455p
TYPE OPEN OR CLSD FOR TYPE OF RESTRUCTURING
FOLLOWED BY RECORD NAME , DATA ITEM NAME, LENGTH AND NEW FORMAT
Data:CLSD ACNT BALC  4 ,I4>
***CLOSED RESTRUCTURING NOW UNDER WAY***
*** CLOSED RESTRUCTURING COMPLETE ***
09/05/83 22.25.22  CPU= 76.57 Secs CT= 27 Mins PT= 6129 Ch= 512p
FIND1 CALLS    0
FIND2 CALLS    0
FIND3 CALLS    0
GET  CALLS   0
MODIFY CALLS   0
STORE CALLS  10
INSERT CALLS  10
RECORDS RESTRUCTURED  10
INDEX READS   0
INDEX WRITES  10
PRIME DATA READS  50
PRIME DATA WRITES  50
```

Figure 7
Closed Restructuring
while adding
new accounts

The "£1 Posting" Program.

The previous examples used a small data base of 10 customers and 10 accounts. In order to defray any overheads incurred by the small volumes the second example uses ten accounts per customer.

An application program has been written to increment the value of the balance data item of each account record by £1. The order of the account records ensures that this operation is not equivalent to the progression in physical placement sequence which would be undertaken by a Closed Restructuring (i.e. it is 1,3,5...99,2,4,6...100 and the program modifies the accounts in numerical sequence).

Figure 8 gives a listing of the program.

Figure 9 shows a run of the program. 658 page turns were required and the program ran in 16.37 seconds of CPU time.

Figure 8

Program Listing
The "£1 Posting" Program

```
C
C
C
C
C
C
C
 130 WRITE(6,9130)
9130 FORMAT('0***RUN TO POST £1 TO EACH ACCOUNT***')
     DO 9131 I=1,20
     ISTATS(I)=0
9131 CONTINUE
     CALL EMASFC ('METER',5,'DUMMY',0)
     DO 136 IJI=1,100
     CALL FIND4(KACNT,IJI,ISPACE,ISPACE,IERR)
     CALL GET(KBALC)
     IUWA(1)=IUWA(1)+1
     CALL MODIFY(KBALC)
 136 CONTINUE
     WRITE(6,9132)
9132 FORMAT ('0*** END OF £1 POSTING RUN ***')
     CALL EMASFC ('METER',5,'DUMMY',0)
     WRITE(6,9125)(ISTATS(I),I=1,12)
     GO TO 5

C
C
C
```

```
Data:13

***RUN TO POST £1 TO EACH ACCOUNT***
09/05/83 22.52.36  CPU= 134.01 Secs CT= 54 Mins PT= 10649 Ch= 895p

*** END OF £1 POSTING RUN ***
09/05/83 22.58.22  CPU= 150.38 Secs CT= 60 Mins PT= 11307 Ch= 999p
FIND1 CALLS 100
FIND2 CALLS   0
FIND3 CALLS   0
GET  CALLS 100
MODIFY CALLS 100
STORE CALLS   0
INSERT CALLS   0
RECORDS RESTRUCTURED   0
INDEX READS 100
INDEX WRITES   0
PRIME DATA READS 100
PRIME DATA WRITES 100
```

Figure 9

Normal Run Of

The "£1 Posting" Program

The Open Restructuring.

The restructuring to reduce the length of the Balance data item from 5 to 4 digits was carried out using an Open Dynamic strategy. In order to ensure that existing account records were accessed, the £1 Posting run was executed while the restructuring was under way.

Figure 10 shows the run while the restructuring is operational. There is a substantial increase in resource consumption to 2919 page turns and 65.49 seconds of CPU time. There is no increase in I/O activity on the data base, however, and if resources were available to provide processing capacity the open strategy could provide a viable alternative to a Static or Closed Dynamic Strategy for the Data Base Administrator in certain circumstances.

```
Data:23

***START OF AMEND DATA ITEM FORMAT RESTRUCTURING ***
09/05/83 22.59.00  CPU= 150.40 Secs CT= 60 Mins PT= 11411 Ch= 1000p
TYPE OPEN OR CLSD FOR TYPE OF RESTRUCTURING
FOLLOWED BY RECORD NAME , DATA ITEM NAME, LENGTH AND NEW FORMAT
Data:OPEN ACNT BALC  4 ,I4)
***OPEN RESTRUCTURING NOW UNDER WAY***



Data:13

***RUN TO POST £1 TO EACH ACCOUNT***
09/05/83 23.00.07  CPU= 150.46 Secs CT= 61 Mins PT= 11555 Ch= 1002p

*** END OF £1 POSTING RUN ***
09/05/83 23.25.26  CPU= 215.95 Secs CT= 87 Mins PT= 14474 Ch= 1420p
FIND1 CALLS 100
FIND2 CALLS   0
FIND3 CALLS   0
GET  CALLS 100
MODIFY CALLS 100
STORE CALLS   0
INSERT CALLS   0
RECORDS RESTRUCTURED 100
INDEX READS 100
INDEX WRITES   0
PRIME DATA READS 100
PRIME DATA WRITES 100
```

Figure 10
Run of "£1 Posting" Program With Open Restructuring

References.

ADABAS - "The ABABAS Introduction Manual" Published by Software-AG Ref. ADA-410-000.

ARORA & CARLSON - "The Information Preserving properties of Relational Data Base Transformations" 4th International Conference on Very Large Data Bases (1978).

ASTRAHAN _ "System R, A Relational Approach to Data Base Management" - Transactions on Database Systems June 1976.

BACHMAN - "The Data Structure Set Model" - ACM SIGMOD Workshop on Data Description Access and Control May 1974.

BATORY - "Optimal File Design and Reorganisation Points" Transactions on Database Systems March 1982.

BEAVER - "Dynamic Techniques for Restructuring a Conceptual Schema - an Implementation" University of Pennsylvania Working Paper Ref. 77-06-02 1977.

CHAMBERLAIN-76 - "Relational Database Management Systems" - ACM Computing Surveys March 1976.

CHAMBERLAIN-81 - "Support for Repetitive Transactions and Ad-hoc Queries in System R" - Transactions on Database Systems March 1981.

CHEN - "The Entity Relationship Model - Towards a Unified View of Data" - Transactions on Database Systems March 1976.

CODD & DATE - "Interactive support for Non-programmers: The Relational and Network Approaches" - ACM SIGMOD Workshop on Data Description Access and Control May 1974

CODD-70 - "A Relational Model for Data for Large Shared Data Banks" - Communications of the ACM June 1970.

CODD-79 - "Expanding the Relational Model to Capture More Meaning" Transactions on Database Systems December 1979.

CODASYL-69 - "A Survey of Generalised Data Base Management Systems" Published by the CODASYL Systems Committee.

CODASYL-71 - "The CODASYL Data Base Task Group Report" (1971) Available from the British Computer Society.

CODASYL-78 - "CODASYL Data Description Language Committee Journal of Development 1978" Published by the Canadian Government on behalf of CODASYL.

CODASYL-81 - "CODASYL Data Description Language Committee Journal of Development 1981" Published by the Canadian Government on behalf of CODASYL.

CODASYL-COBOL-JOD - "CODASYL COBOL Committee Journal of Development 1978" Published by the Canadian Government on behalf of CODASYL.

DALE & DALE - "Schema and Occurrence Structure Transformations in Hierarchical Systems" ACM SIGMOD International Conference on the Management of Data.

DATE - "An Introduction to Database Systems" Published by Addison-Wesley.

EASYTRIEVE-IMS - "The EASYTRIEVE-IMS Reference Manual" Published by PANSOPHIC Inc. Ref. 7809.

FLORY & KOULOUDJIAN - "A Model and Method for Logical Database Design" - 4th International Conference on Very Large Databases (1978)

FRY & SIBLEY - "Evolution of Data Base Management Systems" - ACM Computing Surveys March 1976.

FRY & JERIS - "Towards a Formulation and Definition of Data Reorganisation" ACM SIGMOD Workshop on Data Description Access and Control May 1974.

GERRITSEN & MORGAN - "Dynamic Restructuring of Data Bases with Generation Data Structures" University of Pennsylvania Working Paper Ref. 75-12-02 1975.

IBM-ADF - "The IMS Application Development Facility" Published by IBM Ref. SH20-1931.

IBM-IMS - "IMS Data Base Administration Guide" Published by IBM Ref. SH20-9025

IBM-VSAM - "Virtual Storage Access Method Planning Guide" Published by IBM Ref. GC26-3799.

IBM-DBRC - "Data Base Recovery Control Feature General Information Manual" Published by IBM Ref GH35-0010.

IBM-MFS - "IMS Message Format Service User's Guide" Published by IBM Ref. SH20-9053.

IBM-UTAS - "User Task Analysis System" From IBM-UK Course EA12 - Data Base Analysis and Design.

IBM-DA - "Data Analysis" Published by IBM Ref. UK26-8101.

IBM-SQL - "SQL Data System Application Programming" Published by IBM Ref. SH24-5018.

IDS/II - "IDS/II Data Base Administrator's Guide" Published by Honeywell Information Systems Inc.

IDMS - "IDMS Concepts and Facilities" Published by Cullinaine Corporation.

ISO - "Concepts and Terminology for the Conceptual Schema and the Information Base" Published by the International Standards Organisation (1982)

JACKSON - "The Design of Data Processing Systems" Michael Jackson Systems Ltd. 1980.

KAM & ULLMAN - "A Model of Statistical Databases and their Security" Transactions on Database Systems March 1977.

KAY M.H. - "Restructuring and Reorganisation" Working Paper to the BCS/CODASYL Data Base Administration Working Group (Feb. 1978).

LEFKOVITZ - "Data Management for On-Line Systems" Published by Hayden 1974.

LUCKING-74A - "A Descriptive Methodology for Multiple Views of an Information Processing System" IFIP Working Conference on Database Management 1974.

LUCKING-74B - "Advantages of the Data Structure Set Model" - ACM SIGMOD Workshop on Data Description Access and Control 1974.

MERTEN & FRY - "A Data Description Language Approach to File Translation (The Data Translation Project)"

MICHAELS MITTMAN CARLSON - "A Comparison of the Relational and CODASYL Approaches to Data Base Management" ACM Computing Surveys March 1976.

MINSKY - "On Interaction with Data Bases" ACM SOGMOD Workshop on Data Description Access and Control 1974.

NAVATHE - "Schema Analysis for Data Base Restructuring" Transaction on Database Systems June 1980.

NAVATHE & FRY - "Restructuring for Large Databases - 3 levels of Abstraction" Transaction on Database Systems June 1976.

OLLE - "The CODASYL Approach to Data Base Management".

PALMER - "Record Subtype Facilities in Database Systems" 4th International Conference on Very Large Databases 1978.

PANEXEC - "The PANEXEC System Reference Manual" Published by PANSOPHIC Inc. Ref 8205.

PANVALET - "The PANVALET System Reference Manual" Published by PANSOPHIC Inc. Ref 2379.

PIROTTE - "High Level Data Base Query Languages" International Seminar on Intelligent Question Answering and Data Base Systems 1977.

SHIPMAN - "The Functional Data Model and the Data Language DAPLEX" Transactions on Database Systems March 1981.

SMITH & SMITH - "Database Abstractions - Aggregation and Generalisation" Transaction on Database Systems June 1977.

SNUDGREN 75 - "Theory of Data Bases" Published by Carter 1975.

SNUDGREN 78 - "Data Base Design in Theory and Practice" 4th International Conference on Very Large Databases 1978.

SHU et al - "EXPRESS A Data Extracting, Processing and Restructuring System" Transactions on Database Systems June 1977.

SOCKUT and GOLDBERG - "Data Reorganisation Principles and Practice" ACM Computing Surveys Dec. 1979.

SOCKUT-78 - "A Performance Model for Computer Data Base Reorganisation Performed Concurrently with Usage" Operations Research Sept-Oct 1978.

STOCKER - "Efficient Organisation of Internal DBMS Structure" 4th International Conference on Vary Large Data Bases 1978.

STOCKER & DEARNLEY - "A Self Organising Data Base Management System" IFIP Working Conference on Data Base Management 1974.

TAYLOR & FRANK - "CODASYL Data Base Management Systems" ACM Computing Surveys March 1976.

TOTAL - "The TOTAL Information System" - Cincom Systems Inc. 1982.

VAO - "A Dynamic Database Reorganisation Algorithm" Transactions on Database Systems June 1876.

WILSON - "Data Base Restructuring - A Direction for Aberdeen" University of Aberdeen Computer Centre 1978.

XEPHON - "Buyer's Guide to Data Dictionaries" Published by Xephon 1982.

YORMARK - "The ANSI/SPARC DBMS Model' Published by North-Holland 1976.