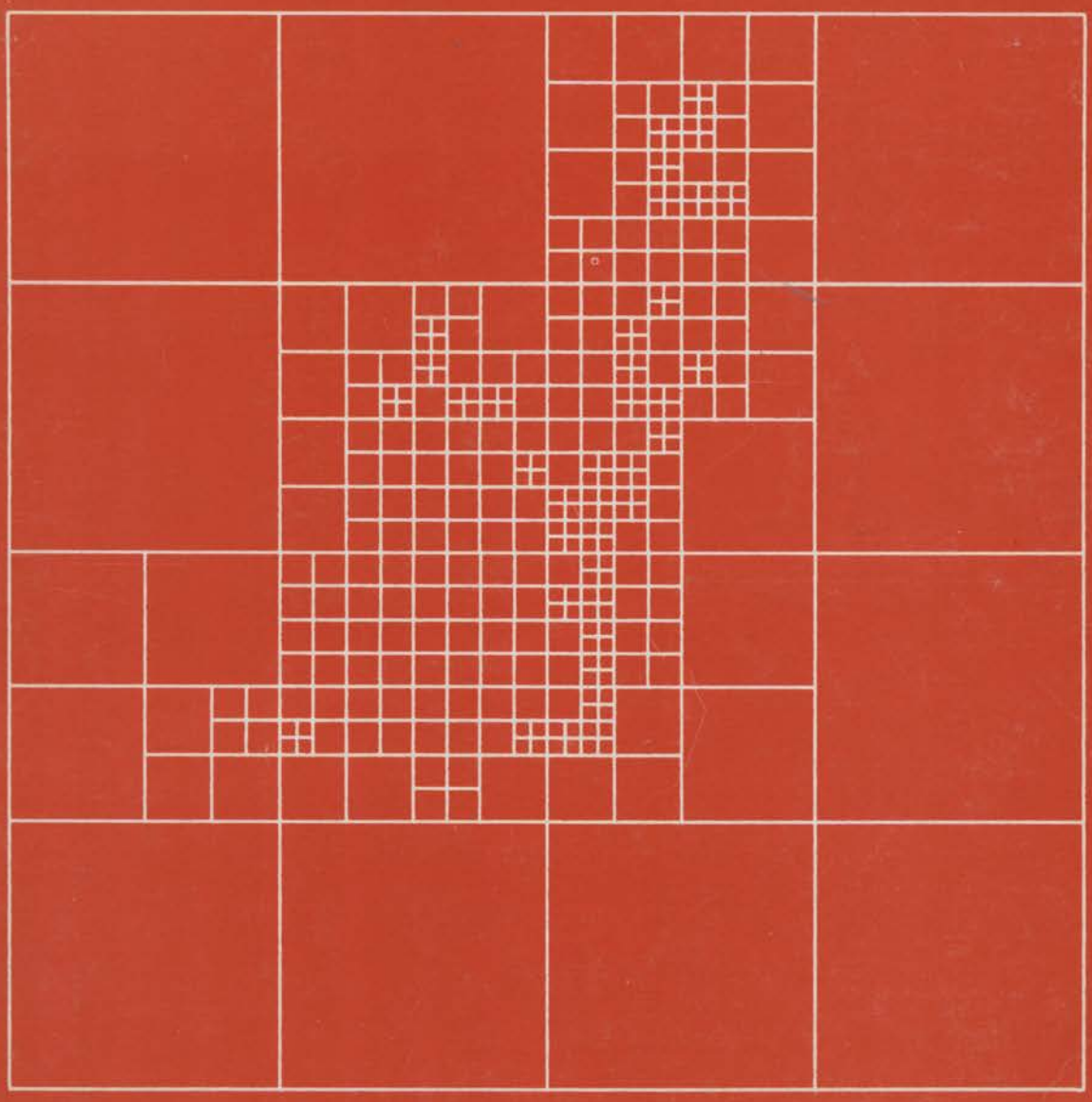
Data Base Management Systems

Edited by Garth L. Wolfendale



Data Base Management Systems

Proceedings of the joint ANU/ACS one-day seminar held at the Computer Centre of the Australian National University, 17th November, 1976.

Edited by Garth L. Wolfendale.

The Australian National University Press Canberra 1977

First published in Australia 1977
Printed in Australia for
the Australian National University Press, Canberra

National Library of Australia
Cataloguing-in-Publication entry

Data base management systems, Australian National University, 1976.
Data base management systems.

ISBN 0 7081 0826 1.

1. Data base management -- Congresses.
I. Wolfendale, Garth Leslie, ed.

001.6442

Southeast Asia: Angus & Robertson (S.E. Asia) Pty Ltd, Singapore
Japan: United Publishers Services Ltd, Tokyo
North America: Books Australia, Connecticut, U.S.A.

Preface

The seminar brought together users, manufactures, software vendors and data processing 'experts', to present the main current ideas and problems in data base management systems (DBMS), at tutorial level and to discuss them from the point of view of applications and user experiences.
The papers in these proceedings reflect the differing aspects of DBMS and also include a transcription of a vendors' forum held at the seminar, in which users questioned computer manufacturers and vendors of DBMS software. The first paper was not presented at the seminar as such but is included for completeness reasons, to provide a detailed survey of current DBMS work with especial emphasis on areas that have been neglected in existing surveys.

G.L.W.

Table of Contents

1 A Survey of Recent Work and Ideas in Data Base Management Systems

Garth Wolfendale

SUMMARY

The recent ACM Survey on Database Management Systems concentrated on discussion and comparison of the Relational, CODASYL and Hierarchic data models. There was hardly any mention of the rest of the dimensions of DBMS. The survey presented in this chapter is an attempt to correct this imbalance, which seems to prevail in the literature and in debates. It is argued that data models are related to DBMS as command and programming languages are to operating systems.

This chapter acts as a survey, whereas the next chapter by J.L. Smith presents a critical *introduction* to DBMS.

INTRODUCTION AND THEME

The so called 'grand debate' over the network (CODASYL) and relational data models is concerned with an issue that cannot be resolved by argument. What a user finds convenient for modelling a data application depends upon the application, how much control the user wants over storage and access mechanisms, how readily a model maps onto the date application, and so on. Blanket statements such as:– 'users are becoming increasingly oriented toward the *information content* of their data, and decreasingly concerned with its *representation details*'. (Chamberlin, 1976, p.43) are commonly made. I argue that users (i.e. general applications users) have always been interested in their applications. They have been forced, however, to be concerned with representation details because that is all that was available in programming and applications-oriented languages and packages. Users are *not* becoming increasingly oriented towards the information content of their data, it is the computer industry that is showing signs of waking up to a fact that has been around since computing began.

There is also the need to consider a large class of users concerned with data representation details. This class of 'data systems programmers' develop data systems, being intimately concerned with data organization, data control, data selection methods and data structures. The role of these users is to provide data systems for use by applications programmers and other users. In an ideal world, the latter would treat the data systems produced by the former in a similar way to operating systems. All the system dependent work would be done by the data system, and the user left only with application-dependent work to do. However, the world is not ideal, even in the area of operating systems which have been evolving over the last twenty years.

At the present time, data systems are available in the form of data base management systems which support a fixed type of data model and provide a set of data management capabilities available to the applications programmer, with varying degrees of system-oriented involvement. As in the evolution and

current state of the work in operating systems, there is a great deal still to be done in the area of data base management systems' theory, design and implementation. It must be recognized that most of the significant problems in data base management arise from applications which raise problems unthought of in the rarefied atmospheres of manufacturers and universities.

The papers in this book are intended to raise issues, to let applications 'speak for themselves' and to explore possibilities, rather than to provide 'answers' to questions. In this spirit four applications are presented from:-

- The National Library
- The National Bureau of Statistics
- CSIRO
- The AMP Insurance Company

Also, in the paper on distributed data bases, the literature is explored from the point of view of an implementation at ANU of a network which will evolve to support a centralized data system coexisting with a distributed data system.

Finally, in keeping with the theme, a forum took place in which vendors of data base systems were faced with questions from attendees. The main theme of the forum was what had been provided by the industry for the solution of users' database problems. A transcription (editted to be readable) is included in this book.

The rest of this chapter attempts to provide a survey of work in database management systems in order to provide a context for the rest of the papers.

The work on data models is very important for the design of DBMS but is far from sufficient. It is rather like concentrating on command or programming languages in the study and design of operating systems. There is a vast area of study concerned with the structures, functions and mechanisms of such systems which are independent of the languages and language models supported by the system.

In fact most systems support multiple programing languages or language models, and, for example, one system in which I was involved, in at the design level (IAS, 1975) allowed multiple command language models to coexist as well as programing languages.

Therefore, this chapter is more concerned with design, theory and implementation of DBMS rather than with data models which are adequately treated by Creasy and McKenzie in this book. To illustrate what is being said here, it is worth noting that in the ACM Computing Surveys Special issue on data-base management systems (March, 1976), out of 156 pages (or approximately 110,000 words), we find approximately 4 pages concerned with protection, 2 pages on performance and approximately 1 page dealing with other aspects of data base management systems (e.g. selection mechanisms). Thus about 95.5% of the volume is concerned with either history or consideration of three data models (CODASYL, RELATIONAL, HIERARCHIC), the rest briefly touches on other aspects. The survey presented in this chapter is an attempt to consider those aspects of data base management systems so sadly neglected in the ACM survey.

Data models are adequately dealt with in papers in the rest of this book, and in the above survey.

DIMENSIONS OF DATA BASE MANAGEMENT SYSTEMS

Data base management systems have not evolved at the same rate on all fronts. For example, there has been real progress made in separating data descriptions from applications programs and in support of shared data through such

centralized descriptions, but there has been little *coherent* progress in the area of data control (integrity, recoverability, reliability) or efficiency (optimal data mapping and organizations).

The starting point of an analysis of a data base management system is to consider it as a black box, DBMS, with two interfaces–that with a data environment and that with a user or process environment.

The data environment is made up of 'data-entities' which may be short lived and highly fluctuating (e.g. the data contents of main memory) or may be long lived and relatively fixed (e.g. output devices like line printers). Data-entities have logical relationships. For example – files fall into related groups or sets, records fall into files, microfilms and documents are parts of libraries, disks and magnetic-tapes fall into peripheral constellations.

Thus we have so far a two-dimensional description of a data environment. One dimension, that of data-entity-level, ranges from the 'physical' to theoretically unlimited degrees of logicality. The other dimension, that of relationship or 'structure', represents the logical relationships existing between data entities *at the same level.*

Entities in the data environment at different entity levels, map onto each other or are transforms of each other. Data-items map onto main store or into records, records map onto blocks, blocks onto disks or magnetic tapes and so on. A DBMS is responsible for handling the data mappings in the data environment.

Entity descriptions, relations and mappings are only part of a complete definition of a data environment.

At all levels, data flow and access require control and synchronization. Controls on the access of data must be applied in order to maintain privacy and security. Checks must be made on data to ensure consistency with other data in the environment, error procedures and the conditions under which they are called must be defined. These aspects will be referred to under the general heading of *data-control.*

The management of data involves the *selection* of entities in the environment. This involves the seeking out of entities and the recognition of entities as those satisfying criteria of search.

The methods and criteria adopted by DBMS for selection of entities will be referred to as selection-mechanisms.

The flexibility of a DBMS will be determined by what freedom a user has in describing mappings, selection-mechanisms and controls as well as data-structure (entities and relationships). Existing systems have a mixture of user-defined and 'hard-coded' capabilities in the definition of a DBMS.

In summary, in order to specify a DBMS, the following four major dimensions are required:–

1. Data-structure specifications for data entities and their structural relations.
2. Data-control specifications for privacy controls, error conditions and associated actions, checks on data to maintain integrity, backup procedures and so on.
3. Data-selection specifications of rules for data searching and criteria for success.
4. Data organization specifications, for mappings between entities in the data environment, the software associated with the management of entities, structural relations and mappings, and the conditions under which this software is brought into effect.

All this is required to define a DBMS to manage a data environment and associated user-processes, which themselves require further support specific to applications. Thus in the most general scheme of things, a DBMS requires the continuation of the above four dimensions of specification for the support of application dependent processes. In the CODASYL (1971) proposals, for example, the SCHEMA plays the role of providing system-wide specifications, whereas the SUB-SCHEMA provides the application- oriented specifications. Wolfendale (1974) explored the possibility of providing one language system from which a DBMS could be derived, based on the compiler-compiler idea (Brooker et al). The ANSI-SPARC proposals (ANSI/X3/SPARC, 1975) attempt to provide a framework for the analysis of data base management systems. The present description is in basic agreement with those proposals.

DATA STRUCTURES IN DBMS AND OUTLINE OF A GENERAL APPROACH TO DBMS ARCHITECTURE

As was described in the last section, data structures define the data entities and logical relations in a data environment.

A DBMS manages the system-wide data structure and in the general case, the application or process-dependent data structures.

This section briefly guides the reader to literature on currently relevant data models. The paper by Creasy in this book presents the relational data model and that of McKenzie presents the CODASYL model. Papers by Chamberlin (1976), Taylor and Frank (1976) and Michaels, Mittman and Carlson (1976) provide excellent tutorial and discussion papers concerned with these models.

The aim of this section is to argue that there is a need to reconsider the architecture of DBMS to enable support of many data models (at the application and process level) on the same system.

What is the purpose of a data model? I argue that, like a programming language, its purpose is to enable a user to express his problem and effect operations at his problem level without being involved in any system dependent details. Thus, as for programing languages, there are potentially many data models well suited for classes of applications.

Consider the following arbitrary list of data applications:–

1. Economic modelling and forecasting
2. Medical history, patient record and nursing schedules
3. Computer aided design
4. Computer aided instruction
5. Interactive graphics
6. Library and reference information systems
7. Computer based book production and subsetting (e.g. dictionaries— long and concise)
8. On-line, real time data acquisition, data processing and modelling
9. Chemical structure identification and recognition
10. Automata and robotics applications.

and so on.

All of these applications involve at some level the storage, retrieval, updating and searching of large quantities of data.

I am not convinced that the CODASYL, relational or hierarchic data models are sufficient, convenient or necessary for all or any of the above valid data base applications. Of course, we could enforce or so constrain applications to conform to one model, but this is equivalent to demanding that all users program in FORTRAN. Of course it is possible to do anything in FORTRAN, since the

CALL instruction provides a way out of the language limitations, but I don't think that many users would agree with such an approach.

The approach I advocate is to liken DBMS to an operating system. An operating system today supports many languages (ALGOL, FORTRAN, COBOL, PL/I etc.) provides multiple modes of operation– real-time, time-sharing, multi-stream batch and possibly transaction processing.

It may also support a full virtual machine capability to enable concurrent and yet protected operations by radically different and independent processes.

We therefore should not baulk at the idea of a DBMS supporting a wide range of diverse data applications demanding support for radically differing data modelling capabilities.

Now, one consequence of the introduction of virtual machines, was the need to provide, at one level, a unified data system, since, although the processes themselves are free to view data as they wish, the necessary data storage, retrieval and updating operations must be carried out in a unified way to prevent data anarchy. Thus the details of data management, by necessity became transparent to processes.

I feel that it is useful and probably more powerful to consider DBMS as a virtual machine monitor rather than a handler of data requests and models for applications. Key papers in this area are Bairstrow (1970), Buzen and Gagliardi (1973), Goldberg (1973), Denning (1971).

Such an analysis demands a low level data model and data management system, which of *necessity* is transparent to all applications.

The application-specific data models, would be integrated with applications to form virtual machines providing CODASYL, RELATIONAL, HIERARCHIC etc. processes coexisting on the same system. DBMS would be responsible for control and capability mechanisms and other functions that demand inter process control and synchronization, whereas data-model dependent actions would be carried out in-process.

Thus the following is proposed:–

1. A breaking away of DBMS from particular data models
2. Progress towards DBMS as a virtual machine monitor with synchronization, integrity, recovery, privacy functions being centralized in DBMS.
3. Treatment of data models in a similar way to programing languages– supporting many on one system.

Perhaps such a scheme would be one step towards focussing concern on important issues of DBMS design, theory and implementation and away from non-productive bickering about models.

The nett result of such a scheme would be that the user of each process would be able to run an application as though the system were dedicated to the support of its data model. The data-description operations on such a system would enable the user to select from a supported set of models [in a similar way to the selection of programming languages] and as sophistication increased, the user would be able to create his own data models using a generative data description facility (e.g. see Wolfendale [1974]).

DATA-CONTROL MECHANISMS IN DBMS

This section considers the problems of protection, integrity, error detection and recovery in DBMS.

Protection mechanisms. Considering the range and extent of data systems by government, banks, credit bureaux, medical and social services, time-sharing service bureaux and so on, there is concomitant range of needs for personal and

organizational privacy and protection. All require some plan to ensure that the computer systems implement a control structure that corresponds to the protection needs of the organization.

The words 'protection', 'security' and 'privacy' are often used with abandon, and therefore will be defined in the most commonly used way:-

Privacy is the ability of an individual or organization to determine whether, when and to whom information about or owned by the person or organization is to be released.

Security refers to techniques for the access to or modification of information.

Protection refers to those security techniques that control the access of processes to stored information.

Key references in this area are:-
Westin (1967), Miller (1971), Turn and Ware (1975), Martin (1973), Anderson (1973) and Patrick (1974).

Anderson (1973) categorized security violations into three classes:

1. Unauthorized information release — an unauthorized user reads or takes advantage of information,
2. Unauthorized information modification — an unauthorized user makes changes to information,
3. Unauthorized denial of use — prevention of an authorized user from accessing or modifying information, even though the 'intruder' cannot refer to or modify the information.

Security violations can be thought of as *sabotage* whereas *privacy violations* can be thought of as *infringement.*

One of the main problems of computer-based information systems is that an intruder or saboteur may be an otherwise legitimate user of the system, and may carry out the sabotage or infringement quite unintentionally.

As was stated above, protection is a subset of security. For example, a valid security mechanism is the locking of the computer room, whereas it is not concerned with the access of processes to stored information.

The rest of this section will be concerned with protection mechanisms. Many different mechanisms have been proposed and implemented for the protection of information. These can be roughly ordered into five categories of increasing sophistication:-

Unprotected systems
These provide no mechanisms for preventing a determined user from having access to all the information or a system. A large number of currently available operating systems fall into this class. They handle mistakes to some extent, but not the onslaught of a determined saboteur.

Binary or all-or-none systems
In these systems a user has access to either private data or public data. Most first generation commercial time-sharing systems provide this level of protection.

Controlled systems
Access to data entities is monitored and is allowed if the requesting process/user is authorized (is an owner or given access by an owner) to access the data as requested. Only a few complete examples of controlled systems exist today, for example TENEX (Bobrow, 1972), CTSS (MIT, 1965), TOPSIO (Stone, 1970), ADEPT (Weissman, 1969).

User controlled systems
A user may wish to restrict access to a data entity dynamically (e.g. limiting access during key operations, or to times of day), or he may wish to restrict access

to a limited form of data (e.g. a subset or an average). The user can be provided with a means for defining protected objects and protected subsystems.

A protected subsystem is a collection of programs and data with the property that only programs of the subsystem have access to the data (i.e. the protected objects). Access to these programs is limited to calling specified entry points. By constructing a protected subsystem a user can develop any programable form of access control to created objects. Only a few of the most advanced systems have attempted to permit such dynamic control:- MULTICS (Corbato, 1972), CAL (Sturgis, 1973), UNIX (Ritchie, 1974), BCC-500 (Lampson, 1969), CAP (Needham, 1972) and HYDRA (Wulf, 1974).

It is interesting to consider the protection mechanisms provided in the CODASYL system. Since the SCHEMA and SUB-SCHEMA programmer is able to define locks and corresponding keys either explicitly or in terms of procedures, and is able to define entities which are virtual (i.e. generated at run-time from other data items) and encoded/decoded, then the proposals provide a dynamically controlled protection system. CODASYL also, by means of the above mechanisms, allows support for protection based on content (e.g. X has access only to items with value < N). Other work on content access control is given in Conway (1972), Reed (1973), Hsiao (1974) and Hoffman (1970).

Information defined control

The previous systems have been concerned with the access of information. Once accessed, however, there is no control applied. A user who successfully accesses information can then pass it on to any other user. An example of how this is undesirable is that of classified information. It remains classified even though accessed. Some work has been done in this area, in a limited form (e.g. ADEPT Weissman, 1969). Currently proposed data base management systems do not provide for such information oriented protection.

A complete survey of protection system implementations would fill a book in its own right, and therefore I will concentrate on the more recent and sophisticated ideas on the assumption that the reader is already aware of currently available schemes. If not, an excellent tutorial paper by Saltzer and Schroeder is available in Clark and Redell (1975). Two major systems for protection will be considered, viz: The capability system and the access control list system.

THE CAPABILITY SYSTEM

This system was suggested by Dennis and Van Horn (1966), partially implemented by Ackerman (1967) and analysed by Fabry and Yngve (1968).

First of all, a capability system relies on a descriptor based architecture (Wilkes, 1972; Watson, 1970; Dennis, 1965; Fabry, 1974). Consider memory tagged as either data/code or a protection-descriptor value. The latter can be loaded, by instruction, into a protection descriptor register, but cannot be directly altered by normal functions. Also it can be stored into memory from a protection descriptor register and tagged as a protection descriptor value. Thus there are protection and data-oriented functions and registers providing a wall that prevents values that are subject to general computational manipulation from ever being used as protection descriptor values.

This particular 'tagged' architecture is known as a *capability system*. A memory word that contains a protection descriptor value is known as a *capability*.

To apply to basic sharing, suppose that each processor has m protection descriptor registers, and that, a program A is in control of a processor. Program

A has initial access to a catalogue of capabilities, C, whose descriptor is in protection descriptor register, PDR1. A can therefore load capability, C(1), say, into PDR2 to gain access to a private data base (we are treating storage as one level or virtual to simplify discussion). Note that there is no way that a program B can access the private database of A, unless it has capability C(1) in an accessible segment. Now, how does the capability picture expand to tie in with an authentication system in a DBMS, say. Suppose that DBMS responds to an authentication request by creating a new process and starting it executing with a capability for a user access table.

If the user identified as X, supplies password Y, the access process can check identification and load into a PDR the capability for the catalogue associated with X's entry in the user access table. It can then run, say, program A for which X has a capability in its catalogue. So after verification, the process enters X's domain, starting at program A.

By providing for authentication, two protection systems have been tied:–

1. An authentication system that controls access of users to named catalogue capabilities.
2. The general capability system that controls access of the holder of the catalogue capability to other objects stored in the system.

The scheme described so far admits any desired *static* arrangement of accessing authorization. However, X may wish to authorize access to an entity for Z. This will be referred to as *dynamic authorization*.

Dynamic authorization of sharing

One might suppose that X could simply share Z's catalogue and load the capability that it wants to give to Z into Z's catalogue. However this has a major defect. X could also overwrite and destroy Z's other capabilities in its catalogue.

The only secure way is to forge a communication link (segment) between the two and to define a protocol for sending and reception of capabilities. For more details see Lampson (1971), Saltzer and Schroeder (1975).

There are several potential problems with the capability system.

It is extremely difficult of revoke a capability once it has been given to a process.

Capabilities themselves are not associated with any protection once removed from initial owner.

For example, if X gives capability C to Z, then Z can give that capability to another process without the knowledge of X.

It is extremely difficult to trace capabilities to predict consequences of capability ownerships.

In order to counter some of these problems, some systems implement a copy restriction bit, which prevents subsequent capability copying once initially sent and received. Also, some segments are associated only with capabilities so that it is relatively easy to trace and monitor capabilities.

The CAP (Needham, 1972), Plessey 250 (England, 1974) are organized in this way.

In the multics system (Bensonssan, 1972) capabilities are recognized by the hardware only if they are placed in special capability– holding segments, and the supervisor domain never gives out copies of capabilities for those segments to other domains. Also the supervisor maintains threads leading to every copy made to a capability, so that revocation is possible.

THE ACCESS CONTROL LIST SYSTEM

Consider the following scheme:– whenever a user requests access to a data en-

tity, for example a segment, actually a second entity is also allocated, linked to the target entity. This secondary object will be referred to as an *access controller* which contains an *addressing descriptor* for the associated data entity and an *access control list*.

An addressing descriptor for the access controller itself is assigned a unique identifier and placed in the map used by the memory access system. In order to access the data entity, the process must provide the unique identifier of the access controller's descriptor.

This scheme differs in the following ways from the capability system:

1. Allowing access to an entity has known auditable consequences. A program cannot make a copy of the addressing descriptor of an entity since it does not have direct access to it, since it is accessed via the access controller.

   The pointer to the access controller of the entity can be passed to anyone, but every use of the pointer is by means of the access controller which prevents unauthorized access.
2. The access control list implements the dynamic sharing protocol verifying that the requestor is authorized to use the object, *and remains with the entity*. In the capability system, verification was carried out only on a first capability copy and then protection was lost. The access control list is consulted on every access.
3. Revocation of access is manageable by altering access control lists.
4. It requires dynamic storage for access control lists, causing run-time problems that are not found in capability systems. Also list searching (or hashing, binary chopping etc.) adds to the overheads in the system.

PROTECTION REQUIREMENTS FOR A DBMS

The systems described so far, form the bases for protection systems, but as described are not adequate for a sophisticated DBMS. For instance, a user may be restricted in access to data concerning only himself and no other users, and even then in a summarized form.

The systems currently described are not adequate, since the objects of protection are not elements in the lower level system. A solution is to enhance the descriptor to enable complex entities to be described and related operations included in the capability or access control descriptor. A problem here is that such a scheme is beyond current hardware which at most can refer to segments in a one-level store. A solution is the use of *protected subsystems*, which are collections of programs and data entities which are self contained and cannot be interfered with by other programs. The data entities in a protected subsystem are referred to as *protected objects*. Progams in a protected subsystem act as caretakers for the protected objects and interpretively enforce necessary complex controls on access to them.

In the CODASYL sytem, protection of data-items, encoding/decoding etc., would be carried out by protected subsystems. Implementations of protected subsystems as virtual processes (one for each subsystem) was proposed by Dennis and Van Horn (1966) and discussed by Lampson (1971). A more economical scheme, using a single-virtual processor was proposed by Le Clerc (1966) and Evans and Le Clerc (1967), and explored in detail by Lampson (1969), Schroeder (1972), Needham (1972), Sturgis (1973), Jones (1973) and Rotenberg (1974).

CURRENT RESEARCH DIRECTIONS IN PROTECTION

Certification of the correctness of protection system designs and implementations

One must have a precise definition of what a protection system must achieve. For example if it is required that all users be completely isolated from each other, then virtual machine mechanisms per se are adequate. However if controlled sharing of information is required a model is much less easy to come by. Attempts to model the USA military security system have led to extremely complex formulations and difficult implementation problems (Bell and La Padula, 1973; Weissman, 1969). Given a precise model of the protection goals of a system and a working implementation of that system, the next problem is to verify that the implementation actually does what it is supposed to do. Robinson (1975) extends methods of proving assertions about programs to deal with constructs encountered in operating systems.

Information-oriented protection

This is concerned with the use information is put to after release to an executing program. Rotenberg (1973) proposed a 'privacy restriction processor' which traces the use and controls access to information throughout its 'life'. Another mechanism is to encipher information and then broadcast de-coding keys to only those programs that are allowed access to the data. Branstad (1973) considers this problem for multi-node computer networks, and an encipherment scheme is presented by Smith (1972).

Authentication mechanisms

Considerable research and development effort is going into better authentication methods. Work in progress can be found by reading Saltzer (1974), Lipner (1974), Mathis (1974) and a recent workshop IRIA (1974).

RESOURCE SHARING AND LOCKING, CONSISTENCY, INTEGRITY AND RECOVERY IN DBMS

Much work has been done on the problems of resource sharing in operating systems. However, when an integrated data base is considered, then the requirements for resource sharing, locking, integrity of data at recovery from errors are severe and more sophisticated in scope. The underlying mechanism, resource locking, for example, is the same for DBMS and operating systems, but the units of locking are fundamentally different. A database transaction may require locks on arbitary logical and related data entities, whereas most work in operating systems has centred round the locking and sharing of low level entities. Eswaran and Chamberlain (1975), Eswaran, Gray, Lorie and Traiger (1974), Everest (1974) and Coffman, Elphick and Shoshani (1971) give detailed papers in this area and what follows is an attempt to summarise their work.

Consistency is the discipline of preventing semantic errors which may arise due to the interaction of two or more processes operating concurrently on shared data. For example, suppose process X is transferring all elements in subset C of set B into set A, and concurrently process Y is processing all elements of set A. The result is not defined since it depends on the relative speeds and phases of the two processes. At the completion of process Y, there is no guarantee that all elements in set A have been processed. An *inconsistency* has arisen.

Informally, two concurrent transactions have to be processed consistently if the state of the database is the same as though transactions had been executed to completion in a predefined serial order. Solutions to consistency problems usually involve some locking scheme.

Integrity deals with the prevention of semantic errors made by users due to carelessness or ignorance. Integrity checking is primarily implemented by specialized code in programs to check for integrity infringements or by audits of data base contents (e.g. checking the sum of entities or balancing accounts). Auditing has problems, since it is impossible or very difficult to identify the exact error and the process which caused it. Furthermore other processes may be victims of the error before periodic auditing detected it.

This section is concerned with maintenance of consistency and integrity in a shared database, and the potential for recovery when they are infringed.

The unit of control is a transaction (Chamberlin, Boyce and Traiger, 1974). Performing a set of transactions concurrently, $t_1, t_2, \ldots t_n$ must be constrained to be equivalent to some sequence $(t_{i_1}, t_{i_2} \ldots t_{i_n})$ where $(i_1, i_2, \ldots i_n)$ is some permutation of (1,2,....,n). This last as stated earlier, is called consistency (Eswaran, Gray, Lorie and Traiger, 1974). It can be considered as a special integrity constraint establishing conditions under which transactions can run concurrently. Before a transaction is allowed to execute, it must have available all the resources it may need. For DBMS resources refer to entities in the database, which are required by locking and associating with the transaction. Most work to date on integrity has been concerned with consistency maintenance by operating systems. Locking techniques have been investigated extensively for operating systems, but are of limited applicability to DBMS and require significant enhancements.

LOCKING IN OPERATING SYSTEMS

The following is a list of the more common locking techniques developed for operating systems.

Pre-allocation of resources:

Before execution a process or transaction must claim all resources it will need. Typically these are specified on control cards preceding a job; and the process is not started until all resources have been allocated. In DBMS this is not a practical scheme, since resources themselves are often complex data entities involving searching through the database, and attempting to allocate shared and active resources. Often required resources are also data dependent.

Enforced sequencing of processes and transactions:

Processes on transactions potentially competing for resources are presequenced and executed sequentially. For DBMS, a prior knowledge of required data resources is often unknown, which means that most transactions or processes would be forced into sequential operation.

A posterior deadlock discovery and resolution:

This involves the discovery of deadlocks, terminates or returns to a checkpoint of one process involved in the deadlock and the resources locked by the process freed. Deadlock discovery, preemption of resources and backup to recover a state of integrity of a database are complicated problems, and are discussed later.

Deadlock anticipation and prevention:

The algorthms developed for operating systems rely on special properties of resources (e.g. Habermann's banker's algorithm, Habermann 1969) or on limited models of computation (e.g. Schroff 1974). Such restrictions are impossible to apply usefully to DBMS.

LOCKING SCHEMES IN DBMS

Locking schemes to maintain integrity and in particular consistency of a data base are very much 'proposals' at this stage.

In this section, two major contributions will be described; those of Chamberlin, Boyce and Traiger (1973, 1974) and Eswaran, Gray, Lorie and Traiger (1974).

The former system is concerned with the locking of data entities or objects ('individual object locking') whereas the latter system uses predicate locks ('predicate locking') for locking logical entities as well as individual data entities. Thus the latter allows locking of *potential* subsets of the database as well as existing subsets of data entities.

Locking in database environments has the following complications which are not present in a normal operating system environment:

1. *Non-unique resource names:* A resource may be described in many ways, so that it is often unclear whether requests for locking will lead to complications. For example, a transaction may require exclusive access to set B, and another transaction to set C. Since we assume that the elements of the two sets are distinct, then there should be no conflict for concurrent update. However, if B and C both happen to be subsets of set A which may impose constraints on relationships between elements in B and C, then locking should take place at level A, and all access be carried out through A and not at the B and C level. DBMS must be able to handle these problems, not normally found in simple object oriented environments.
2. *Dynamic resource types:* A transaction, by operating on a resource can change its nature. For example, a transaction can lock set B, remove some elements and insert them in set C before releasing them. It is also possible to create new sets at run-time, further complicating the picture.
3. *Interdependent locks:* A transaction may wish to lock some resources and examine them before issuing further lock requests which are dependent on the first. Thus, for example, a process may lock employee records for a particular organization, find a subset of them and then wish to lock corresponding 'department' records.
4. *Increased complexity:* In order to maximize concurrency among transactions, the granularity of locks should be as fine as possible. This results in an integrated database environment literally millions of individually lockable resources, introducing new orders of magnitude into the locking and integrity problem.

Normally acceptable techniques for conventional operating systems become inapplicable. For example, locking resources in a linear order is precluded by the interdependence of locks, advance declaration of resource requirements is precluded by non-unique resource naming problems. Detection of deadlocks is made significantly more difficult by the increased order of magnitude in the database environment. Chamberlin et al, consider transactions as atomic so that integrity of the database must be guaranteed at the beginning and at the end of a transaction. A transaction, t, is conceived of as being a sequence of *actions* denoted by $(a_1, a_2, \ldots a_3)$. Thus, although actions can and must disrupt integrity, t must be so managed as to guarantee it. The scheme proposed here, requires each transaction to lock all its resources (that it can identify and name at the time) during a 'seize' phase before starting its execution phase. During the seize phase the database must not be *modified* by the seizing transaction. However, the transaction can search the database in order to discover the resources to be locked. For such a transaction preemption of locked resources and backup to wait for the preempted resource are easy to manage.

Once a transaction has entered its execution phase, it is not allowed to claim more resources, thus no backup will be necessary due to deadlock. At the end of the execution phase, a transaction must free all its resources before starting a new seize phase.

The seize phase in a database environment may be a rather complicated task and seize phases for different transactions can and should run in parallel. Of course deadlocks can arise (the usual deadly embrace situation for example), and must be discovered by DBMS. A resource must be preempted from a transaction involved in the deadlock, causing it to wait for the resource. In the Chamberlin et al scheme, an ageing mechanism is associated with transactions to avoid indefinite delay. It is then shown that in this scheme each transaction will eventually be processed.

Thus the Chamberlin et al scheme is a combination of the previously listed simpler schemes:

1. Needed resources are preclaimed (however, there may be a fair amount of processing to be carried out in the seize phase).
2. If the seize phase leads to deadlock, this must be detected, resources preempted and transactions in seize phase backed up.
3. Presequencing of transactions by time stamping, enforcing or ageing mechanism and avoiding deadlock due to indefinite delay.

The deadlock discovery algorithm used by Chamberlin et al is described in King and Collmeyer (1973). However, it is limited since a major assumption is that transactions must wait for at most one other transaction. Bayer (1974, 1975 and 1976) describes a deadlock discovery scheme in which this restriction is relaxed. Also, in Bayer (1976), a scheme for breaking deadlock and preventing indefinite delay is also described.

In Eswaran, Gray, Lorie and Traiger (1974) a technique is presented using predicate locks for logical locking. This technique is proposed to solve the *'phantom problem'*. This problem occurs thus:–

Assume that D is the total data environment, and that B is a database, where B D. Two transactions t_1 and t_2 may have successfully locked all their required resources and be executing. t_1 may add a new object $r_1 \epsilon$ D to B and t_2 may add a new object $r_2 \epsilon$D to B, such that t_1 would have locked r_2 and t_2 would have locked r_1, if t_2 and t_1 could have seen these objects in their seize phases. r_1 and r_2 are called *phantoms' since they might, but not necessarily, appear in B while t_1 and t_2 are in execution phases (not allowed to lock).*

The appearance of a single phantom does not cause any problem, since the parallel operation t_1, t_2 is equivalent to serial operation. However when more than one phantom is involved, this equivalence cannot be guaranteed. To enforce consistency, each transaction t is required to lock (for read or write access) all data objects O ϵ D, irrespective of whether they are in B or are just phantoms, which might in any way influence or be influenced by the effect of the transaction on B. O is locked in the Eswaran et al scheme by specifying a predicate P defined on D such that O ϵ S(P) where S(P) is the subset of elements of D satisfying P. Two transactions are said to be in conflict, if for their predicates P_1 and P_2, it is true that $_r \epsilon$ S(P_1) S(P_2) and t_1 or t_2 performs a write action on r. Thus conflict can arise even if r is a phantom. In this case t_1 and t_2 must be run serially, but the order in which they are run is irrelevant for consistency.

For more details and discussion of coordidates for suitable locking predicates see the Eswaran et al paper.

Predicate locking and phantoms can be dealt with by more simple approaches, at the possible loss in some efficiency.

Transactions which do not contain any write actions (*readers*) do not need to lock phantoms. Any transaction with at least one write action is a *writer*. Consider two transactions t_1 and t_2. If both are readers, there cannot possibly be any phantoms. Let t_1 be a reader and t_2 be a writer. Assume that there is a phantom $r \epsilon S(P_1) \cap S(P_2)$ such that t_2 might perform a write on r. Then t_1 and t_2 could not run concurrently, if t_1 used predicate locking. If, however, t_1 uses object locking, and successfully terminates its seize phase, then t_1 can run in parallel with t_2 provided that the set of real data objects (without phantoms) in B which t_1 needs to lock for consistency is independent of the phantoms of t_2. In this case, if t_2 should materialize phantoms, then consistency is maintained. Thus t_2 can serialize its own locking in order to object lock its own materializing phantoms, without fear of consistency breakdown.

Hence, the problem of phantoms with its associated need for predicate locking only arises when t_1 and t_2 are both writers. Hence two simple strategies can be applied to replace predicate locking:–

Serialize Writers

Writers are not scheduled to run concurrently. Concurrency is possible for many readers and at most one writer. Consistency is guaranteed by individual object locking and by handling deadlocks and preemptions as described earlier.

Partition the Database

Which is an implementation of a simplified predicate locking scheme. This technique allows a high degree of transaction concurrency and avoids the phantom problem if transactions obey the proper protocols. A partitioning technique is described in Gray, Lorie and Putzolu and Traiger (1975), and has been implemented in system R, an experimental relational database system by Astrahan et al (1976).

Partitions form sets of data objects such that each object belongs to a unique block. There are three types of lock permitted on partitions:– X (exclusive), S (shared) and A (analysis), as well as data object locking within partitions.

In order to make partition locking still work, it must be known what sort of object locking is going on within partitions. This can be achieved by leaving traces at the partition level or equivalently by introducing additional lock modes at the partition level as in Gray et al papers.

The partition scheme can be generalized to a hierarchy, where partitions split into sub-partitions and so on up to a finite number of levels. Transactions can lock at a coarse or in a fine level depending on the tradeoff between low locking overhead and high concurrency requirements.

DBMS would be responsible for maintaining required levels of tradeoff using preemption and transaction sequencing to do so.

RECOVERY IN DBMS

The ability of DBMS to recover a consistent database in the event of errors, crashes, inconsistencies and so on depends heavily on the way data entities are locked, protected, preempted, backed-up and so on. For example, audit trailing is less than useful if, the recovery sequence itself cannot be guaranteed to leave a consistent database. Lorie (1976) discusses physical integrity in segmented databases and Bayer (1976) presents a coherent picture of a recovery scheme. This latter will now be briefly described. Backup of a database to an earlier stage of integrity may become necessary for such reasons as:– hardware failure, system failure, deadlock and deadlock resolution, prevention of indefinite delay, violation of integrity constraints during or at the end of a transaction, protection violations and so on.

For discussion purposes, it is assumed that all data for recovery of a previous state of integrity is available from some reliable backing store alone, since we are including central processor and main memory in our list of recovery failures. A failure has the same effect as halting operation at any random time of the whole system resulting in an undefined state of the central processor and main memory. Any read or write operations in progress on backup store are also considered undefined, without having *destructive* side effects. The recovery problem reduces to the study of the effects of and recovery from incomplete transactions, and the problem of undoing these effects on the contents of the database. Consider the updating of a single data entity, for example a partition. The key to a recovery scheme is the concept of a *shadow entity* (Lorie, 1976), which is the retained old entity when a transaction updates it. It is retained for

the contingency that if a failure occurs before the transaction completes, the shadow will be available for recovery. The shadow can be released (or backed up on a medium for the contingency that secondary storage fails) when the updated entity has been properly constructed on backup store (committed). However, the update-committment sequence which switches over from the shadow to the new entity (entry in directories etc.) may itself fail and recovery must be possible. After a new entity has been constructed by an updating transaction, two versions of the entity reside on backup store, each representinig part of a state of database integrity. To perform the update committment, the transaction handling this operation must first assure that the shadow is no longer used by other transactions by placing an exclusive lock on it. Then and only then can the new entity be validated and the shadow eventually freed. Since at any time there may be two versions of an entity, two address maps are required, defining for each entity the physical locations of the shadow and the new entities. Assume that each map entry is time-stamped (e.g. the system time of the moment of creation). Also associated with each map entry shall be a writeable Boolean *validation variable*. A value *true* means that the corresponding physical block represents part of a state of integrity of the database. The time stamp can be used to identify the state.

The data objects required for a committment sequence to be performed by transaction t to update entity E are:

FL: A list of free storage areas to place new blocks.

AS,AN: Two arrays held on backup store (of necessity for recovery) representing two address maps and the timestamps for physical blocks. AS[i] and AN[i] contain the addresses and timestamps of the shadow and new versions of E(i).

BS,BN: Two boolean arrays located on backup store to represent the two validation variables. Vc: An address map located in main or backup store. Vc[i] contains the address of the currently valid version of E(i) which is made available for read access to other transactions while E(i) may be updated by t. Vc is not needed for recovery.

The sequence for updating and committing of entity E(i) is as follows:–

1. Place an intent-to-update lock (A-lock) or E(i), allowing read access to 'old' version of E(i) for other transactions.
2. Get a free slot(s) for new E(i) from FL.
3. Prepare and write new entity E(i).
4. Update and timestamp AS[i] and AN[i].
5. Validate BS[i] or BN[i].
6. Invalidate BN[i] or BS[i].
7. Convert A-lock or E[i] to exclusive lock (X-lock).
8. Update Vc[i] with address of new E[i].
9. Release X-lock or E[i].
10. Free shadow and update FL or otherwise backup shadow with time stamp.

In this technique, no system quiescence for taking checkpoints is needed and no extra operations are needed on the backup store beyond mainteance of the structures AS, AN, BS and BN are required.

Now consider a system failure. It can occur according to the following four cases:–

Case 1. A failure occurs before step 5 (Validation of BS[i] or BN[i]). Then exactly one of BS[i] or BN[i] will be *true*. The corresponding address map entry AS[i] or AN[i] will point to a valid version of E[i] representing a state of integrity.

Case 2. If step 5 fails, we cannot know whether validation was successful or not, and we have two cases:-

Case 2a: The validation was not performed or failed, i.e. only one of BS[i] or BN[i] is *true*, so that recovery proceeds as in case 1.

Case 2b: Validation was performed successfully i.e. both BS[i] and BN[i] are true. Thus there are two valid versions of E[i]. Timestamps in AS[i] and AN[i] can be used in recovery to obtain the newest version.

Case 3: Step 6 (invalidation of the other boolean variable) fails. Depending on how this fails, there will be one or two validated versions of E[i], and recovery is performed as in case 2.

Case 4. A failure occurs after step 6. In this case there will be exactly one validated version of E[i] or backup store which can be used for recovery.

Instead of backing up a process completely for recovery it may be desirable to introduce additional intermediate safe points and to use partial backup. Overheads are introduced in maintaining information on how far transactions are backed up and how much work is necessary to recover (Astrahan et al, 1976; Lorie, 1976).

STORAGE AND SELECTION MECHANISMS IN DBMS

There has been a great deal of work in this area, and there exist excellent surveys, e.g. D'Imperio (1969), Maurer and Lewis (1975), McGee (1969), Nievergelt (1974) and Severance (1974) to name but a few. In this section I will briefly discuss recent work on the problems for DBMS in supporting a rich selection of structures for programs without the latter being dependent on the structure (data independence) and in the efficient and optimal use of storage and selection mechanisms by DBMS. Shapiro et al (1969) produced a handbook on file structuring which attempted to produce a coherent approach to the design and analysis of storage structures. The relative efficiencies of the different structures were discussed as a function of usage statistics.

Recent research has been concerned with the problem of processing high level access requests (in some external language) into an intermediate form on which optimisation can be carried out as far as file and storage accesses are concerned, and at the same time maintaining data independence. This general problem is referred to as the 'reduction' problem. An example of a reduction problem, is the case where the objects of search are in four domains A, B, C, and D of data entities related by R_1, R_2 and R_3 in the following way:- R_3 (R_1(A,B), R_2(C,D)). A straightforward evaluation would involve the construction of two 'inverted' domains F = R_2(C,D) and E = R_1(A,B) so that evaluation merely becomes R_3(E,F). However, such a solution would involve greatly increased quantities of secondary storage to store the inverted domains which themselves may require much more storage than the original domains. However, subsets of the domains can be inverted if usage statistics merit it. One of the earlier investigations in this area is given by Palermo (1972), in the context of a relational database. His analysis is also applicable to the general model of the database I have been using in this chapter. The main problem is to so reduce a query as to produce a *practical* solution. For example, if a user requests 'all' elements satisfying a predicate, Palermo shows that retrieval of the universe of the range (set of elements for which the predicate is true) is not necessary. Also, for the range of queries studied, Palermo showed that no triple (related set of entities) need be accessed more than once. This is achieved by the restriction of domains of variables, the building of indexes, and a least growth strategy (the order of forming joins domains is chosen so that the resulting subset tends to grow

slowly). Cartesian products are only indexed not actually formed and only the domains referenced in the query are retained. Palermo's proposals are an advance on the reduction algorithm of Codd (1971), involving much less storage and time overhead and thus increasing the workable size of the database supported.

A reduction algorithm for the DIAM system is described by Astrahan and Ghosh (1974), and by Ghosh and Senko (1974). Fehder (1973), developed a Representation Independent Language, RIL, for the DIAM system, and Astrahan et al described a heuristic algorithm which constructs a DIAM access path to a given RIL enquiry. Ghosh and Senko give an algorithm for the reduction of queries to access paths in a network which yields an access path of minimum path cardinality. Greenfeld (1974) and Rothnie (1975) discuss optimization techniques for a relational system like LEAP (Feldman and Rovner, 1969). Rothnie's strategy attempts to utilize the information gained with every triple-access for optimization purposes. Astrahan and Chamberlin (1975) describe the SEQUEL interpreter and the reduction algorithm employed by it to make use of secondary indexes for 'minimization' of data accesses. Wong and Chiang (1971) constrain each query to be a boolean expression over elementary queries. In this case the database can be optimized for query primitives and reduction becomes the problem of converting a boolean expression into some standard form.

Some work has been carried out on the reduction problem which is concerned with central processor usage which can become a real bottleneck in a database system. To reduce CPU time, less dynamic and interpretive search strategies are demanded, which can be implemented as CPU efficient search or access modules. Mehl and Wang (1974) give a proposal to increase data independence supported by IMS with the help of compiled routines which intercept the communication between application program and DBMS, and Taylor (1974) proposes a pre-processor to obtain data independence at precompile time.

Work apart from the reduction problem has been concerned with efficiency of various storage and search schemes. Shapiro et al (1969) has already been mentioned.

One of the frequently employed techniques is the acceleration of search with the help of an inverted file. If the inverted file is repeatedly inverted we obtain an hierarchical index organization known as a 'B-Tree' (Bayer and McCreight, 1972). B-Trees allow a logarithmic search time for retrieval, update and insertion. Cardenas (1974) carried out an analysis of the performance of inverted database structures.

Lum (1970) introduced multi-attribute indexes which allow quicker answers to queries of higher complexity. Finkel and Bentley (1974) describe an extension of binary trees to quad trees supporting logarithmic searches with two parameters (keys). Haerder (1974) proposes bit lists as an index organization and investigates when bit lists are superior to conventional methods of indexing. He analysed access with the help of simulation including a comparison of storage structures for indexes. Another method of reducing search times is hashing. Hashing has been extensively studied in connection with data management, e.g. Ghosh and Lum (1975), Lum et al (1971), Lum (1973).

Ghosh (1976) discusses the CPU service and I/O requests are derived for a Direct Access Method, Index Seeeqqquuential Access Method and a Hierarchic Index Access Method. Performance of the different access methods under different conditions are derived for the models.

DATA ORGANIZATION IN DBMS

This dimension of DBMS is concerned with the mapping of data entities onto each other and onto host computer and peripheral mechanisms. There has been much work attempting to organize the area of computer architectures (e.g. Thurber and Wald, 1975; Hollander, 1967; Flynn, 1972). In the area of DBMS there has been no such development. However in the paper on Distributed Databases in this book, Wolfendale, gives a taxonomy for the description of DBMS comparing it with the computer communications network taxonomy of Anderson and Jensen (1975). This taxonomy is used as a framework for the discussion of the organization of DBMS for distributed databases.

REFERENCES

Ackerman, W. and Plumer, W. 'An Implementation of a Multiprocessing Computer System'. Proc. ACM Symposium on Operating System Principles, Oct. 1967.

Anderson, J. 'Information Security in a Multi-User Computer Environment'. Advances in Computers, 12, New York: Academic Press, pp. 1-35, 1973.

Anderson, G.A., Jensen, E.D. 'Gomputer Interconnection Structures: Taxonomy Characteristics and Examples'. ACM Computing Surveys, 7, No.4, Dec. 1975, pp. 197-213.

ANSI/X3/SPARC. Interim Report: Study Committee on Data Base Management Systems. ACM SIGMOD Newsletter, 1975.

Astrahan, M.M., Blasgen, M.W., Chamberlin, D.D., Eswaran, K.P., Gray, J.N., Griffiths, P.P., King, W.F., Lorie, R.A., Mc.Jones, P.R., Mehl, J.W., Putzolu, G.R., Traiger, I.L., Wade, B.W. and Watson, V. 'System R: A Relational Approach to Database Management'. ACM Transactions on Database Systems, 1, No.2, 1976.

Astrahan, M.M., Chamberlin, D.D. 'Implementation of a Structured English Query Language'. Communications of the ACM, 18, pp.580-588, 1975.

Astrahan, M.M. and Ghosh, S.P. 'A Search Path Selection Algorithm for the Data Independent Accessing Model (DIAM): Proc. 1974 ACM SIGFIDET Workshop, ACM, New York, 1974.

Bairstow, J.N. 'Many from One: The Virtual Machine Arrives'. Computer Decisions, p.29-31, Jan. 1970.

Bayer, R. 'Aggregates: A Software Design Method and its Application to a Family of Transitive Closure Algorithms'. TUM Math. Report No.7432, Technische Universitat Munchen, September 1974.

Bayer, R. 'On the Integrity of Data Bases and Resource Locking'. in Data Base Systems, Lecture Notes in Computer Science series, Hasselmeier and Sprith (Eds), 1975, pp.339-361.

Bayer, R. 'Integrity, Concurrency and Recovery in Data Bases'. in ECI Conference 1976, Samelson (Ed), in Lecture Notes in Computer Science Series, Berlin: Springer-Verlag, pp.79-106, 1976.

Bayer, R., McCreight, E. 'Organization and Maintenance of Large Ordered Indexes'. Acta Information 1, pp.173-189, 1972.

Bell, D., La Padula, L. 'Secure Computer Systems'. Mitre Corp. Technical Report MTR 2547, vols. I, II, and III, November 1973.

Bensonssan, A., Clingen, C., and Daley, R. 'The Multics Virtual Memory: Concepts and Design'. Communications of ACM, vol.15, No.5, pp.308-318, May 1972.

Branstad, D. 'Privacy and Protection in Operating Systems'. Computer, vol.6, No.1, pp.43-46, January, 1973.

Brooker, R., Morris, D., and Rohl, J.S. 'Experience with the Compiler'. Computing Journal, vol.9, pp.345-349, 1967.

Buzen, J.P., Gagliardi, V.O. The Evolution of Virtual Machine Architecture'. AFIPS Proceedings, vol.42, pp.291-300, NCC 1973.

Cardenas, A.F. 'Analysis and Performance of Inverted Data Base Structures'. Communications of ACM, 18, pp.253-263, 1975.

Chamberlin, D.D. 'Relational Database Management Systems'. ACM Computing Surveys, Special Issue: Data-Base Management Systems, vol.8, No.1, March 1976, pp.43-66.

Chamberlin, D.D., Boyce, R.F. and Traiger, I.L. 'A Deadlock Free Scheme for Resource Locking in a Data Base System'. Information Processing, 74, pp.340-343, North

Holland, Amsterdam, 1974.
Chamberlin, D.D., Gray, J.N., and Traiger, I.L. 'Views, Authorization and Locking in a Relational Data Base System'. 1975 AFIPS NCC Proceedings, vol.44, pp.425-430, 1975.
Clark, D.D. and Redell, D.D. (Eds) 'Protection of Information in Computer Systems'. IEEE Tutorial, COMPCON 75, 1975.
CODASYL Programing Language Committee. DBTG-report, 1971.
CODASYL Programming Language Committee. DBLTG proposal. February 1973.
CODASYL System Committee. 'Feature Analysis of Generalized Data Base Management Systems'. Technical Report, May 1971.
Codd, E.F. 'A Relational Model of Data for Large Shared Data Banks'. Communications of ACM, 13, pp.377-387, 1970.
Codd, E.F. 'Further Normalization of the Data Base Relational Model, and Relational Completeness of Data Base Sublanguages'. In Data Base Systems, R. Rustin (Ed), Englewood Cliffs: Prentice-Hall, 1971.
Coffman, E.G., Elphick, M.J. and Shoshani, A. 'System Deadlocks'. ACM Computing Surveys, Vol.3, No.2, June 1971, pp.67-78.
Conway, R., Maxwell, W. and Morgan, H. 'On the Implementation of Security Measures in Information Systems'. Communications of ACM, vol.15, No.4, April 1972, pp.211-220.
Corbato, F., Saltzer, J., and Clingen, C. 'Multics-The First Seven Years'. AFIPS Conference Proceedings, 40, pp.571-583, SJCC, 1972.
D'Imperio, M.E. 'Data Structures and their Representations in Storage'. Annual Review of Automatic Programming, vol.5, Oxford: Pergamon, 1969.
Denning, P.J. 'Third-generation computer systems'. ACM Computing Surveys, vol.3, No.4, pp.175-216.
Dennis, J. 'Segmentation and the Design of Multi Programmed Computer Systems'. Journal of ACM, vol.12, No.4, pp.589-602, October 1965.
Dennis, J. and Van Horn, E. 'Programming Semantics for multi programmed computations'. Communications of ACM, vol.9, No.3, pp.143-155, March 1966.
England, D. 'Capability Concept Mechanism and Structure in System 250'. IRIA International Workshop on Protection in Operating Systems, pp.63-82, August 1974.
Eswaran, K.P. and Chamberlin, D.D. 'Functional Specifications of a Subsystem for Data Base Integrity'. IBM Research Report RJ, 1601, 1975.
Eswaran, K.P., Gray, J.N., Lorie, R.A. and Traiger, I.L. 'On the Notions of Consistency and Predicate Locks in a Data Base System'. IBM Research Report RJ 1487, December 1974.
Evans, D. and Le Clerc, J. 'Address Mapping and the Control of Access in an Interactive Computer'. AFIPS Conference Proceedings, 30, pp.23-30, SJCC, 1967.
Everest, G.C. 'Concurrent Update Control and Data Base Integrity'. Data Base Management, Proceedings of IFIP workshop, Cargese, Corsica, pp.241-270, April 1974: Amsterdam: North Holland, 1974.
Fabry, R. 'Capability-Based Addressing'. Communications of ACM, vol.17, No.7, pp.403-412, July 1974.
Feldman, J.A. and Rovner, P.P. 'An ALGOL based Associative Language'. Communications of ACM, vol.12, pp.439-449, 1969.
Finkel, R.A. and Bentley, J.L. 'Quad-Trees: A Data Structure for Retrieval or Composite Keys'. Acta Information, vol.4, pp.1-9, 1974.
Flynn, M.J. 'Some Computer Organizations and their Effectiveness'. IEEE Transactions on Computers, Sept.1972, pp.948-960.
Ghosh, S.P. 'Performance of File Access Methods'. IBM Research Report, RJ 1712, January 1976.
Ghosh, S.P. and Lum, V.Y. 'Analysis of Collision when Hashing by Division'. Information Systems, vol.1, pp.15-22, 1975.
Ghosh, S.P. and Senko, M.E. 'String Path Search Procedures for Data Base Systems'. IBM J.Res.Dev., vol.18, pp.408-422, 1974.
Goldberg, R. 'Architecture of Virtual Machines'. AFIPS Conference Proceedings, vol.42, pp.309-318, NCC 1973.
Gray, J.N., Lorie, R.A. and Putzolu, G.R. 'Granularity of Locks in a Large Shared Database'. IBM Research Report, RJ 1606, June 1975.
Gray, J.N., Lorie, R.A., Putzolu, G.R. and Traiger, I.L. 'Granularity of Locks and

Degrees of Consistency in a Shared Database'. IBM Research Report, RJ 1654, September 1975.
Greenfeld, N.R. 'Quantification in a Relational Data System'. 1974 AFIDS NCC Proceedings, vol.43, pp.71-75, 1974.
Habermann, A.N. 'Prevention of System Deadlocks'. Communications of ACM, vol.12, No.7, pp.373-377, July 1969.
Haerder, T. 'Zugriffszeitverhalten bei der Auswahl von Sactzen aus einer Datenbank'. Tech. Hochschule, Darmstadt, Berichte der Inf.-Forsch, DV74-3.
Hoffman, L. (Ed) 'Security and Privacy in Computer Systems'. Los Angeles: Mellville, 1973.
Hollander, G.L. 'Architecture for Large Computing Systems'. AFIPS proceedings SJCC, 1967, pp.463-466.
Hsiao, D., Kerr, D. and Stahl, F. 'Research on Data Secure Systems'. AFIPS proceedings, NCC, 1974, pp.994-996.
IAS: Digital Equipment Corp: 'Interactive Applications Supervisor'. Documentation, 1976.
IRIA, International Workshop on Protection in Operating Systems, Rocquencourt, France, August 1974.
Jones, A. 'Protection in Programmed Systems'. Ph.D. Thesis, Carnegie Mellon U., 1973.
King, P.F. and Collmeyer, A.J. 'Database Sharing – An Efficient Mechanism for supporting Concurrent Processes'. AFIPS, NCC, 1973, pp.271-275.
Lampson, B. 'Dynamic Protection Structures'. AFIPS, vol.35, FJCC, pp.27-38, 1969.
Lampson, B. 'Protections'. Proc. Fifth Princeton Symp. on Inf. Sci. and Sys. pp.437-443, March 1971.
Le Clerc, J. 'Memory Structures for Interactive Computers'. Ph.D. Thesis, U. of Calif., Berkeley, May 1966.
Lipner, S. (Chairman) 'A Panel Session-Security Kernels'. AFIPS proceedings, vol.43, NCC 1974, pp.973-980.
Lorie, R.A. 'Physical Integrity in a Large Segmented Database'. IBM Research Report, RJ 1767, April 1976.
Lum, V.Y. 'Multi-Attribute Retrieval with Combined Indexes'. Communications of ACM, vol.13, 660-665, 1970.
Lum, V.Y. 'General Performance Analysis of Key-to-Address Transformation Methods Using an Abstract File Concept'. Communications of ACM, vol.16, pp.603-612, 1973.
Lum, V.Y., Yuen, P.S.T. and Dodd, M. 'Key to Address Transform Techniques, a Fundamental Performance Study on Large Existing Formatted Files'. Communications of ACM, vol.4, 1971.
Martin, J. 'Security, Accuracy and Privacy in Computer Systems'. Englewood Cliffs: Prentice-Hall, 1973.
Maurer, W.D. and Lewis, T.G. 'Hash Table Methods'. ACM Computing Surveys, vol.7, pp.5-19, 1975.
Mathis, R. (Chairman) 'A Panel Session-Research in Data Security–Policies and Projects'. AFIPS proceedings, vol.43, pp.993-999, NCC 1974.
McGee, W.C. 'Generalized File Processing'. Annual Review in Automatic Programming, vol.5, Oxford: Pergamon, 1969.
Mehl, J.W. and Wang, C.P. 'A Study of Order Transformations of Hierarchic Structures in IMS Data Bases'. ACM SIGFIDET Workshop, ACM, New York 1974.
Michaels, A.S., Mittman, B. and Carlson, C.R. 'A Comparison of Relational and CODASYL Approaches to Data-Base Management'. ACM Computing Surveys, Special Issue: Data-Base Management Systems, vol.8, No.1, March 1976, pp.125-151.
MIT-Computation Center, CTSS Programmer's Guide, 2nd Ed., MIT Press, 1965.
Miller, A. 'The Assault on Privacy'. U. of Michigan Press, 1971 and New York: Signet, 1972.
Needham, R. 'Protection Systems and Protection Implementations'. AFIPS Proceedings, vol.41, I, pp.571-578, FJCC 1972.
Nievergelt, J. 'Binary Search Trees and File Organization'. ACM Computing Surveys, vol.6, No.3, 1974.
Palermo, F.P. 'A Data Base Search Problem'. IBM Research Report RJ 1072, 3July 1972.
Patrick, R. 'Security Systems Review Manual'. Montvale; AFIPS Press, 1974.
Reed, I. 'The Application of Information Theory to Privacy in Data Banks'. Rand Corp. Technical Report, R-1282-NSF, 1973.

Ritchie, D. and Thompson, K. 'The UNIX Time-Sharing System'. Communications of ACM, vol.17, 7, pp.365-375, July 1974.

Rotenberg, L. 'Making Computers Keep Secrets'. Ph.D. Thesis, MIT Dept. of Elec. Eng., Sept. 1973, and MIT Proj. MAC Tech. Rept. TR-116.

Rothnie, J.B. 'Evaluating Inter-Entry Retrieval Expressions in a Relational Data Base Management System'. AFIPS Proceedings, vol.44, pp.417-423, NCC 1975.

Saltzer, J. 'Protection and the Control of Information Sharing in Multics'. Communications of ACM, vol.17, No.7, pp.388-402, July 1974.

Saltzer, J.H. and Schroeder, M.D. 'The Protection of Information in Computer Systems'. In IEEE Tutorial 'Protection of Information in Computer Systems', 1975, pp.57-199.

Schroeder, M.D. 'Cooperation of Mutually Suspicious Subsystems in a Computer Utility'. Ph.D. Thesis, MIT, 1972 and MIT Project MAC Technical Report TR-104.

Schroff, R. 'Vermeidung von totalen Verklemmungen in bewerteten Petrinetzen'. Dissertation, Technische Universitat Munchen, 1974.

Severance, D.G. 'Identifier Search Mechanism: A Survey and Generalized Model'. ACM Computing Surveys, vol.6, No.3, 1974.

Severance, D.G. 'A Parametric Model of Alternative File Structures'. Information Systems, vol.1, pp.51-55, 1975.

Shapiro, R.M., Saint, H., Millstein, R.E., Holt, A.W., Marshall, S., and Sempliner, L. 'A Handbook on File Structuring'. Technical Report, RADC-TR-69-313, vol.1, 1969.

Smith, J., Notz, W. and Osseck, P. 'An Experimental Application of Cryptography to a Remotely Accessed Data System'. ACM Proceedings of 25th Nat. Conf., pp.282-298, 1972.

Stone, D. 'PDP-10 System Concepts and Capabilities'. PDP-10 Applications in Science, vol.II, Maynard: DEC, pp.32-55, 1970.

Sturgis, H. 'A Postmortem for a Time Sharing System'. Ph.D. Thesis, U. of Calif., 1973, also Xerox Corp., Palo Alto, Technical Report CSL74-1.

Taylor, R.W. 'Data Administration and the DBIG Report'. Proc. of ACM SIGMOD-SIGFIDET Conf., 1974, NY:ACM, pp.431-444.

Taylor, R.W. and Frank, R.L. 'CODASYL Data-Base Management Systems'. ACM Computing Surveys, vol.8, No.1, pp.67-104, March 1976.

Thurber, K.J. and Wald, L.D. 'Associative and Parallel Processors'. ACM Computing Surveys, vol.7, No.4, pp.215-255, December 1975.

Turn, R. and Ware, W. 'Privacy and Security in Computer Systems'. American Scientist, 63, 2, pp.196-203, 1975.

Watson, R. 'Timesharing System Design Concepts'. NY: McGraw-Hill, 1970.

Weissman, C. 'Security Controls in the ADEPT-50 Time Sharing System'. AFIPS Proceedings, vol.35, pp.119-133, FJCC 1969.

Westin, A. 'Privacy and Freedom'. NY: Atheneum, 1967.

Wilkes, M.V. 'Time Sharing Computer Systems'. 2nd Ed., NY: American-Elsevier, 1972.

Wolfendale, G.L. 'A System for the Definition of Syntax and Semantics of Data Description Languages'. International Computing Symposium, 1973, Gunther and Lipps (Eds), NY: American-Elsevier, 1974, pp.517-526.

Wong, E. and Chiang, T.C. 'Canonical Structure in Attribute Based File Organizations'. Communications of ACM, vol.14, pp.593-597, 1971.

Wulf, W. et al 'HYDRA: The Kernel of a Multiprocessor Operating System'. Communications of ACM, vol.17, No.6, pp.337-345, June 1974.

2 A Critical Introduction to Data Base Management Systems

Dr. J.L. Smith

The requirements of Data Base Management Systems (DBMS) are discussed from the viewpoint of information processing in an enterprise. The main requirements are identified as: (1) facility of access to information, (2) control. Three roles in the management of data in an information system are defined in relation to the DDMS: (1) information administration (2) application administration (3) data base administration. The evolution and current trends in DBMS are described in this light. The paper is written to serve as a perspective overview for nine other papers on aspects of DBMS, to be given jointly at the one-day seminar.

INTRODUCTION

In order to provide an introduction to Data Base Management Systems (DBMS) [1] which will serve as a background to the subject matter of this seminar, the area has to be viewed from the wider environment of the computer based information system [2]. In the broad view each of us is involved in an enterprise whose business is the control of certain commodities. The basis of this control is an Information System (IS), which in many enterprises today contains a Computer System (CS) and within this operates a DBMS (see Figure 1). The DBMS provides many access and control functions for the files of the stored data base and interfaces to the computer's operating system. The most important components of the CS exterior to the DBMS are the Application Program (AP) subsystems, which are developed to serve the particular IS.

The following definition has been given for many years in an educational course: 'A DBMS is one that allows multiple independent users to have concurrent access to a central repository of information'. To this it must be added that the users are guiding the processing functions of an IS, and these functions are linked in an often very complex information flow. Therefore two important capabilities of the combined system are required.

1. facility of access to information
2. control

Facility of access has two prerequisites concerning data (data is defined to be any symbolic or computer representation of information):

1. the data must be organized
2. the user must be able to express his data access request easily.

A number of different types of control are required; these pertain to:

1. the integrity of data
2. the privacy of data
3. the meaning of data
4. the processing and flow of data
5. the evolution of the IS.

The prerequisities for building satisfactory control modules are:

1. appropriate description of the entities to be controlled

2. facility and control of access to these descriptions
3. facility for change.

Referring to Figure 1, as we progress from the enterprise to the DBMS, it is always true that the scale of information becomes progressively much less and the level of control becomes progressively much greater. The reason for this is that appropriate facility of access and control are being provided, according to the importance of the information, cost constraints and the limitations of the technology. While the technology has progressed greatly in the twenty years since computers were first applied to data processing, its limitations are still very apparent when project performance and goals sought for computer based IS are examined.

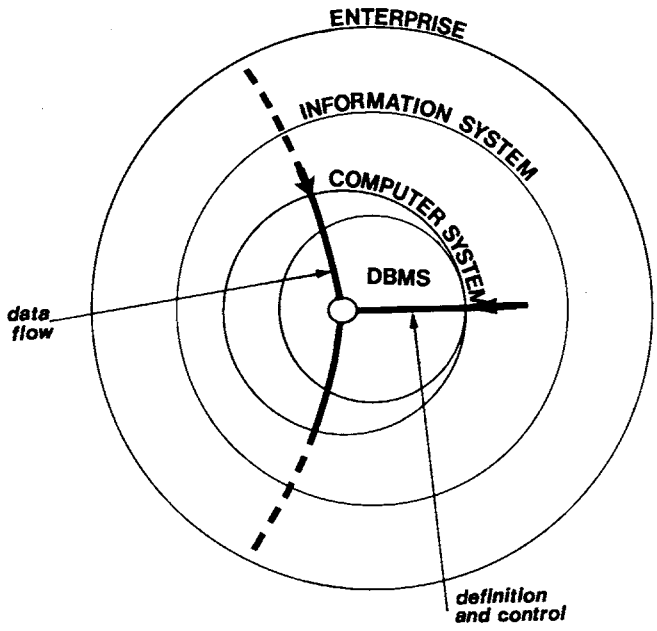


Fig. 1.0 Computer based information system

In evaluating the state of the art in DBMS there is always considerable discrepancy found between what has been demonstrated in research and development projects and what is available for IS. However it can be said that the drive towards good organization of data has in some cases left the user with a much too difficult task to express his information request; there is insufficient choice of data model, data models are too different from acceptable models of information, and in general the data model is complicated by DBMS organizational features. Presently available controls are mainly oriented towards the integrity and privacy of data. More recently the meaning of data and the properties of AP have been documented in data dictionaries but the coordination of control information is still lacking. Facility to change in the face of evolving IS requirements is aided by provisions for data dependence, but many problems exist for management, planning and control in this area.

The following sections expand on these features by considering DBMS evolution and trends, and the subject matter of this seminar.

MANAGEMENT OF DATA

When considering the interface from the IS to the CS and its DBMS, a number of different functions in the management of data can be identified. The report by the ANSI/SPARC Data Base Study Group [3] has classified these under three administration roles. The first is called enterprise administration, but in terms of Figure 1 it is better thought of as IS administration. This involves the definition of all the data to be embraced by the IS, whether it is to be committed to the computer media or not. Formal languages for this purpose are mentioned in subsequent sections. It is clear that in this role the definition of the various types of data processing function (computer and non computer) is also important, and the usual management functions of operation, planning and evaluation occur. Formal facilities for these purposes are far less mechanized.

The second role is called application administration. In [3] this is identified as being solely concerned with AP sub-systems which perform the data processing functions, but undoubtedly similar administration is necessary for the complementary set of non computerized functions. Particular CS problems which have to be dealt with in this role are: the definition of the subset of data for a particular function, function specification for programmers, provision of the appropriate data model, development of programs to operational level, and implementing change.

The third role, termed data base administration, is concerned with the organization of the data base for timely and economic access, from the global viewpoint of the IS. It is clear that this involves interaction with enterprise administration and application administration. The paper by Unsworth: 'Australian Mutual Provident Society — Case Study' is an example of particular decisions which were taken in these roles.

At this point it is appropriate to mention some data limitations of DBMS. If data is essentially text, but its retrieval is contingent on the analysis of the text, special AP sub-systems must be built (see Simmons: 'Information Retrieval Problems in the National Library of Australia'). It is also unusual to find a DBMS suited to handling geographical data. This is because the data organization and processing required for graphical and topographical data must be specially tailored. The combination of geographical data bases with DBMS controlled data bases is discussed by Cook: 'Geographic Data Bases in Land Use Research'.

DBMS EVOLUTION

A substantial treatment of this topic is given in [1].

In a similar fashion to any evolving technology, DBMS were first developed as superior systems for performing the automatic data processing tasks previously handled using data management facilities. In the former systems the limited file and record structures led to much data redundancy with the resultant lack of coordination of programs and assimilation of data. Thus the drive was towards efficiency at the data management level, with the result that numerous independent data files were replaced by an *integrated data base*, which removed all unnecessary data redundancy by using complex record and file structures. At the same time the development of data communications technology and much faster computers allowing on-line access, provided for interactive, transaction and batch modes of use of data base. This potentially powerful basis for an IS still required the mechanisms for control and facility of

access, and it is these features which are of major concern in this paper. The continued development of better technology for data base organization and communication is of considerable importance to the future of computer based IS, but it is a specialist area which should be transparent in this context.

The control required to preserve the integrity of data in the face of fallible machines, programs and humans has always existed in rudimentary form. In DBMS these *restart and recovery* techniques have evolved to match the reliability and complexity of hardware, software and data. The essence of these techniques is the duplication and preservation of data of various types associated with the data base [4,5]. DBMS provide alternatives in the way recovery can be planned for and executed after a particular type of failure. The challenge, in both DBMS design and its operation for a particular data base, is for the data base manager to be able to make the optimum decisions in the trade offs between costs, data base availability and data base response. The problem has been compounded by the development of complex communications networks and the payoffs for improvements in the area of recovery are great.

Another type of integrity threat, which has often been controlled using recovery techniques, arises from multiple access to a data base when update is involved. The concurrency conflict of two or more programs accessing common data under this circumstance is typically resolved by using recovery and restart for all but one program. An even more conservative approach sometimes used is to prevent conflict by controlling the types of concurrent activity. Better detection facilities and use of semantic information on processes would lead to much more efficient techniques.

Centralized definition of data, the most important single mechanism for control, arose with the integrated data base. This definition facility was first provided by a family of data definition languages (*DDL*) typified by the specification of the Codasyl body [6,7] (see Mackenzie: 'Codasyl Data Base Management Systems). One language called the DDL for the Schema was designed for the specification of the entire data base. The data model consisted of hierarchically structured records and functional relations between records, called sets. In addition there was provision for defining data item types and procedures to control data conversion, data compaction, data materialization, validation of data items and relations, and privacy. The language also provided for the specification of physical organization in the form of access methods and indexes, and for the specification of access paths which would be exploited. The other languages of the family were called DDL for the Sub-Schema. These languages had two main purposes (1) to specify a subset of the total schema view with additional privacy procedures (2) to specify some changes in the way stored data was to be presented to an AP, particularly to take account of the requirements of individual programming languages such as Cobol. A *data directory* was formed by the translation of the collection of schema and sub-schema into an object version.

Complementing the DDL described above was a data manipulation language (*DML*) designed for storing and retrieving records and data items from a data base conforming to a defined schema. The DML was designed to be used as an extension to any procedural programming language, e.g. Cobol, Fortran and PL/1, and hence the term host language system. Self contained systems provided higher level stand alone data languages. In [6] the INVOKE verb had to be executed prior to any data base access; the verb specified the names of the schema and sub-schema to be used and their privacy locks, there-

by gaining access for the program to the specified portion of a particular data base (the question of binding time will be ignored here). This satisfied certain basic control requirements: (1) a considerable contribution to the integrity of data was achieved because all programs derived their data description from the central schema definition provided by some administrator; the DBMS augmented by centrally stored procedures handled all data transformations (2) subject to the security of the installation and operating system, there were numerous effective privacy controls.

Considerably later than when the above control mechanisms were first prescribed and implemented, experience in the operation of large computer based IS indicated the need for other types of control concerning the semantics of data and programs. The IS performed a large number of interdependent processing functions on information, many existing as AP sub-systems using the data base. The correctness of programs could be jeopardized if the meaning of data items and relations was not understood by the programmer. (The same situation applied for the processing of data outside the CS). The syntax of DDL's was not sufficient for this purpose and so, in addition to the data directory, a *data dictionary* [8,9] (see Erbacher: The Place of Data Dictionaries in Large Scale Data Management) providing additional and expanded descriptions of data was devised. The description of processes and AP sub-systems, their data associations and their interdependencies, is now also being included in the data dictionary.

It is necessary for the data dictionary/directory function to be closely integrated with the DBMS. Extension of this function to include more of the semantics of data and processes has obvious pay off for control both inside and outside the CS.

The most difficult requirement of control is the facility for change. A major criticism of the Codasyl DDL schemas [6] is that programs associated with this type of schema are too *data dependent*. In particular they have *physical data dependence* because certain commands in the DML are dependent on the continued existence of declared access methods and access paths. They also have *logical data dependence* in that the permissable sub-schema data models must conform to the network hierarchical record model of the schema. This type of data model inevitably results in the explicit representation of only some of the useful relations amongst the data. More versatility in mapping to the sub-schema and other simpler forms of data model are desirable.

Assuming that the DBMS and the specialist responsible for the particular data base could provide the appropriate internal organization of the data base, facility of access to information is dependent on the data language and data model available to the user. Certain processes require a procedural language and data model of the type described, but these are becoming a smaller fraction of the total, thus leading to use of the relational data model [10,11,12] (see Creasy: Relational Data Base Management Systems). The most important characteristic in a retrieval facility is that the relational model provides a tabular view for any portion of the data base. Additional semantic properties have to be included in the tabular view when updating. In addition the relational model has been the most common one to be used with the latest non-procedural data languages [13,14]. These languages are a most important development towards facility of access. However they are still highly structured and require a degree of skill for correct use. An additional advantage is that they offer potential for less user error, by interactive resolution of ambiguities and error situations.

Closer approaches to a natural language interface are desirable for the casual data base user.

TRENDS IN COMPUTER BASED INFORMATION SYSTEMS

A large number of high level data languages using hierarchical and relational data models have now been developed for DBMS. The architecture described in the ANSI/SPARC report [3] allows that all such data models be supported as external schema and mapped to the internal data base by the DBMS. This significant step in logical data independence is within the capability of the technology.

The recognition of the information system in an enterprise and the need for formal models at this level is a recent and noteworthy development [15] (see Firth: The Infological Approach to Data Bases and their Management). An enterprise embraces a staggering amount of knowledge of information, but only a tiny portion of this is needed for its successful operation. This information must be properly defined, organized and controlled in the IS, to ensure its effective processing. This requires very high level definition of information, also expressing the way it is to be stored and the way it is to be processed.

It is essential to commit information to the data level before it can be processed by formal channels. Of course not all useful information is processed by the CS, some is not even committed to data media. Human intelligence must be called upon in the situations where our puny models and machines cannot cope with the complexities of the enterprise. Thus the processing functions, of the IS are a mixture of human decision making and data processing using computer and non-computer systems. In this context a so called reflective approach to IS is proposed [2] in which user, designer (controller) and CS interact as an evolving system in an environment of external information and data.

New developments in the area of control concern semantics and the facility for change. The ANSI/SPARC report [3] specifies a close integration of the data dictionary function with the DBMS control (see Figure 2), thus ensuring a common source of the semantics of data for all concerned. Recently a number of important extensions to the semantic definition of data have been defined [16], such as relational and transitional constraints, and it has been noted that the requirements for a definition language are very similar to those of the high level access languages [17].

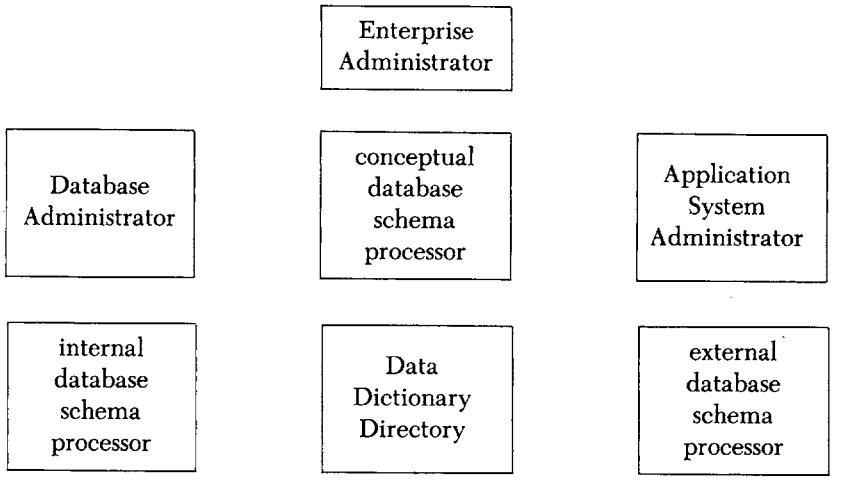


Fig. 2.0 ANSI/SPARC data definition architecture

The need for change in the information processing requirements of the IS still offers a difficult challenge to DBMS research. AP sub-systems can be protected against some changes external to themselves by being data independent. Extensions to the data dictionary concept in terms of process and data description are needed to provide mechanized assistance in planning additions and changes to information processing functions. An approach to reduce the modification necessary for AP sub-systems involved with changed data requirements is discussed in Hawryszkiewycz: 'Semantic Analysis for Data Bases'.

The underlying data base organization and access mechanisms of a DBMS have an important bearing on facility of access in terms of response time. A number of developments at the hardware level can be identified, all aimed at exploiting parallelism in some component [1]. In addition the changing economics of data communication and computer hardware has raised the prospects of geographically distributed data bases [18] (see Wolfendale: 'Distributed Data Bases'). When an enterprise is geographically distributed and the majority of data used locally is generated locally, a strong case can be made in their favour. However the basic requirements for facility of access and control place considerable demands on this technology.

REFERENCES

1. Fry, J.P., and Sibley, E.H., 'Evolution of Data Base Management Systems' ACM Computing Surveys 8, 1 (March 1976), 7–42.
2. Swanson, E.B., Information System Approaches : Directions for Research and Practice, Management Informatics, 5, 4 (Aug. 1976), 155–164.
3. ANSI/X3/SPARC Study Group — Data Base Systems, 'Interim Report', ACM/SIGMOD Newsletter : fdt 7, 2 (Dec. 1975).
4. Drake, R.W., and Smith, J.L., 'Some Techniques for File Recovery'. Australian Comp. J. 3, 4 (Nov. 1971), 162–170)
5. Davenport, R.A., 'Database Integrity' Computer J. 19, 2 (May 1976), 110–116.
6. Codasyl Data Base Task Group, April 1971 Report, (available from ACM).
7. Codasyl Data Description Language Committee, Codasyl Data Description Language Journal of Development, (June 1973), NBS Handbook 113, (Jan. 1974).
8. Uhrowczik, P.P., 'Data Dictionary/Directories', IBM Systems J. 12, 4 (Dec. 1973), 332–350.
9. Canning, R.G. (Ed.), 'The Data Dictionary/Directory Function', EDP Analyzer, 12, 10 (Nov. 1974).
10. Codd, E.F., 'A Relational Model of Data for Large Stored Data Banks', Comm. ACM 13, 6 (June 1970) 377–397.
11. Codd, E.F., 'Understanding Relations', continuing series of articles published in FDT, the quarterly bulletin of ACM-SIGMOD, beginning 5, 1 (June 1973).
12. Chan Peter Pin-Shan, 'The Entity-Relationship Model — Toward a Unified View of Data', Trans. ACM on Database Systems, 1, 1 (March 1976) 9–36.
13. Chamberlain, D.D. and Boyce, R.F., 'SEQUEL : A Structured Englished Query Language', Proc. ACM-SIGMOD Workshop on Data Description Access and Control, May 1974, ACM New York, 249–264.
14. Senko, M.E., "Specification of Stored Data Structures and Desired Output, Results in DIAM II with FORAL, Proc. International Conf. on Very Large Data Bases, Sept. 1975, ACM, New York, 1975, 557–571.
15. Sundgren, B., "Theory of Data Bases", Petrocelli/Charter, New York, 1975.
16. Hammer, M.M. and McLeod, D.J., "Semantic Integrity in Relational Data Base System", Proc. International Conf. on Very Large Data Bases, Sept. 1975, ACM, New York, 1975, 25–47.
17. Senko, M.E., "The DDL in the Context of a Multilevel Structured Description: DIAM II with FORAL" in Data Base Description, Douque and Nijssen (eds), North Holland, 1975, 239–257
18. Booth, G.E., "Distributed Information Systems", Proc. of National Computer Conference, June 1976, AFIPS, 789–794.

3 Codasyl Database Management Systems

H. McKenzie

The title 'CODASYL Database Management Systems' refers to database management systems, or DBMS, which follow the specifications first published in 1969 [1] and in 1971 [2]. Since 1971 the work has been taken over by the Data Description Language Committee, and the Database Language Task Group, and several revised reports have been issued, e.g. [3], [4], [5].

In this paper I shall cover four topics. Firstly the basic components of a CODASYL Database Management System shall be described. Next, the basic tools available in the CODASYL architecture for information and data modelling shall be discussed. Thirdly, the other facilities available in CODASYL Database systems shall be described, together with an example of the way that a database system might be constructed using the CODASYL facilities. Finally a brief outline of some of the criticisms that have been levelled at the CODASYL Database architecture will be presented.

COMPONENTS OF A CODASYL DATABASE MANAGEMENT SYSTEM

Database Management Systems must provide a language for accessing data, often called a Data sub-language. The Data sub-language may be augmented with other language constructs. The system thus formed is called a *self-contained system*, all the facilities available to the user being provided by the database management system. Such systems typically provide a range of language features limited to retrieval and report generation.

The data sub-language may also be embedded in an existing higher level language. The system formed is called a *Host Language System*, and the user has all the facilities of the host language available for manipulating the data. This is the approach taken by CODASYL.

A host language user interface and a self contained user interface may both be provided by the same underlying database management system.

COMPONENTS OF A CODASYL DATABASE MANAGEMENT SYSTEM

Most high level languages include the ability to declare data objects as having particular properties, or as being of a particular type with defined properties. For example in the COBOL Data Division one can declare data objects to have particular properties, and these properties will determine the way that the data item behaves when manipulated with a procedural statement. This notion of packaging commonly used procedural information into a declarative, non procedural form is common in computing. Manipulations involving data objects will have the effect previously defined by the declarative information.

The CODASYL DBTG specifies a number of different languages. One group, which is used to describe data in a declarative form, can be thought of as an extension to the COBOL Data Division. This group consists of a number of 'Data Description Languages,' or DDL's.

One of these is used to provide a centralized definition of all the data in the database. It defines the *structure, model or schema* to be manipulated and is

called the Schema Data Description Language, or Schema DDL. The model defined has some of the qualities of the Data Dictionary system defined elsewhere in these proceedings.

There is also a language used for the definition of that view of the total structure seen by an individual application, called the *Sub-Schema Data Description Language*, or Sub-Schema DDL. One sub-schema DDL would have to be provided for each host language which was to be able to interact with the DBMS, that is one for COBOL, one for PL/I, one for FORTRAN etc. Each sub-schema DDL would have to describe the data defined in the schema in terms of the data constructs allowed in each host language.

It would be advantageous if the transformation between sub-schema and schema was able to change the way that the user could view the data — his *data model*. This however is not possible under the CODASYL specification. The transformation possible between sub-schema and schema is mainly subsetting, with some changes to the way that data items are represented, to accommodate the different host language conventions.

Another group consists of a set of data sub-languages, called *Data Manipulation Languages*, or DML by CODASYL. There would be one of these for each host language supported by the DBMS. As indicated previously, these consist of statements which augment the high level host language, and manipulate the structures defined in the sub-schema DDL.

These Data Manipulation Language statements perform their functions in conjunction with a run-time library and a translated form of the structure specified in the sub-schema DDL, called the 'object sub-schema'. This situation is diagrammed in figure 1.

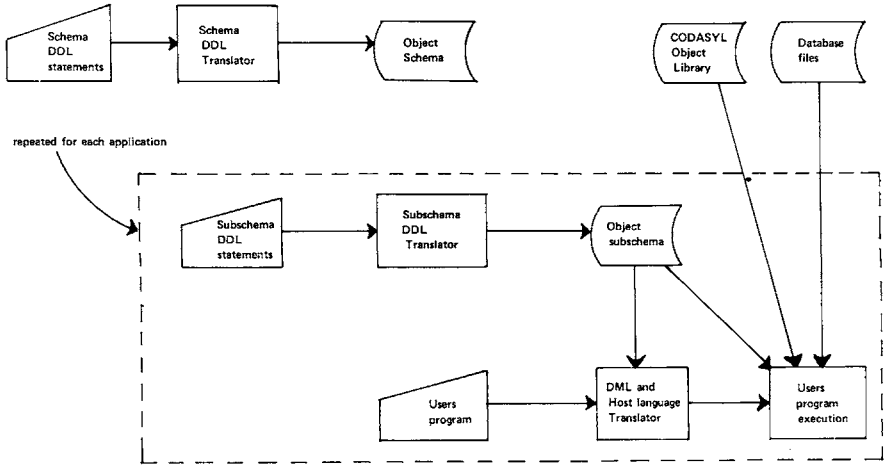


Fig. 1 Data manipulation language system

The CODASYL specifications also refer to a 'Device-Media Control Language', or DMCL which specifies the physical aspects of the database such as the devices on which particular files reside. In addition there must be a language to specify such utility functions as restructuring, copying and backup, and recovery from system malfunction. These languages were mentioned by CODASYL but not specified. Fig. 2 shows the relationships of these constructs.

Some of the facilities required by them could be provided by the operating system job control language.

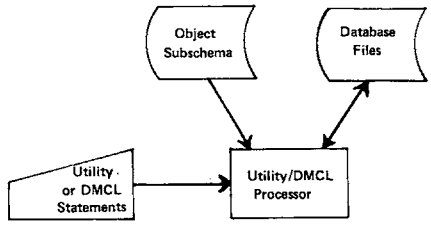


Fig. 2 DMCL and utility specifications

Broadly defined, the problem in database management is to model some features of a real world system, the 'object system', in such a way as to facilitate a user querying and changing the state of the system. Important characteristics of database systems are firstly that the amount of data describing the system is large, and secondly that the data is structured in such a way as to model the salient features of the object system, and simultaneously to make access and update efficient as possible.

One of the aims of the CODASYL DBTG was to specify a system that could be efficiently implemented using current hardware and software technology. Thus the CODASYL DDL includes both features for modelling the object system, and features which specify storage structure options to make particular sorts of operation more efficient.

When modelling an object system, one must identify *objects*, or *entities* from the object system to be included in the model. *Attributes* of those entities to be included must then be identified. Then the *relationships* occurring between entities must be defined. This process must be iterated until a model which will produce the required information outputs is obtained. Then an efficient representation must be devised for the final model and specified using the schema DDL.

The objects used in the DDL for describing data structure are data-items, records, and sets, corresponding to attributes of entities, entities and relationships in the object system model. Relationships between entities may be represented either by sets or by identical key attributes in different entities.

Data items correspond to fields in COBOL and are the most primitive objects manipulated by the database management system. Each data item type has a name. Data items may be assembled into groups, as in COBOL or PL/I.

Records are collections of data items or groups. Each record type defined in the schema has a name which refers to all occurrences of that particular structure. There is sometimes a certain amount of confusion between the *type* of an object and the collection of *occurrences* or *instances* of that object. This is analogous to the difference between the type INTEGER, defined for a fortran program and the collection of integer variables occurring in that program.

Thus a record type can be thought of as a pattern defining the structure of a large number of record instances or occurrences. The words 'type', 'instance' and 'occurrence' will be used in this way in places where there seems to be a possibility of ambiguity.

Sets define one-to-many relationships between record types. *Sets* in the CODASYL sense are not the same as sets in the mathematical sense, and are sometimes referred to a 'cosets', or 'owner coupled sets' or 'data structure sets'.

A set instance consists of a collection of record instances, comprising one 'owner' record instance and arbitrarily many 'member' record instances. The owner instance must be the only record of its particular record type in the set. The member instances may be records of more than one type, but it would be more usual to have only one member type as well.

For example in figure 3 there is a record type representing a state, and a record type representing a town. The set joining them could in this case represent the relationship 'is-within'. An instance, or occurrence, of this set structure would consist of a particular occurrence of a state record associated with a number of occurrences of town records.

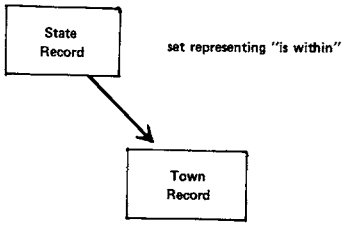


Fig. 3 Example set structure

Two possible occurrences of this set are shown in figure 4.

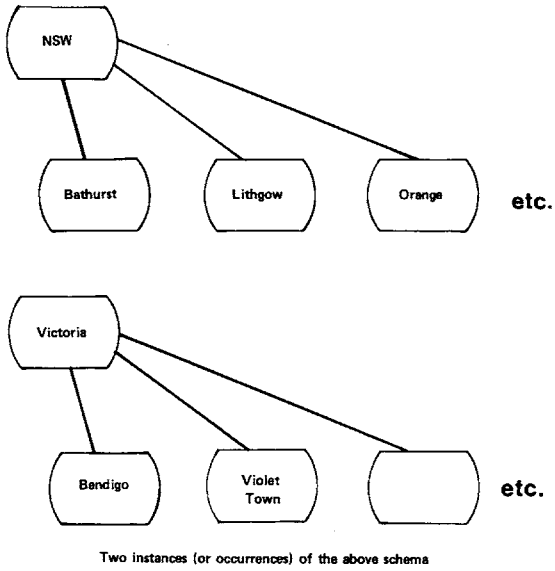


Fig. 4 Set occurence

A model of the object system may be built up using sets. For example a record type may be a member in more than one set, as shown in figure 5.

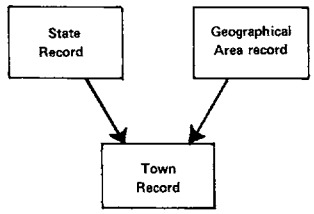


Fig. 5 Example of a record as member of more than one set

A semi-standard formalism used to represent this situation is the 'data structure diagram' or 'schema diagram', due to Bachman [8]. In this formalism rectangles are used to represent record types, and an arrow representing the set relationship is drawn from the owner record type to the member record type of the set. If there was more than one member type in the set, the arrow would split and point to each of the member types. This possibility is the distinguishing mark of the so called 'network' data structure. A record type may be the owner type in one set, and a member type in another.

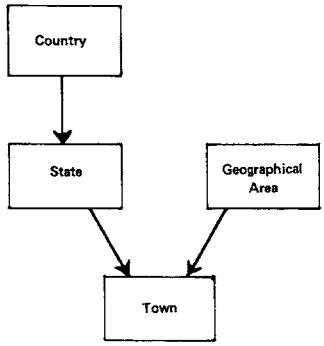


Fig. 6 A record type as owner in one set and member of another

There are cases where it is desirable for the same record type to be both owner type and member type in a set as in figure 7.

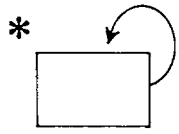


Fig. 7 Record type as owner and member of same set.

This however is not allowed in the CODASYL architecture. There is no practical reason why this should not be allowed, although the definition of certain data manipulation language commands and certain set currency conventions (to be described later) would have to be changed.

There is at least one obvious deficiency in the building blocks proved so far: that is the relationships between owner and member records are one-to-many.

For example where data is to be stored on a collection of journal articles, where each article might have been written by a number of authors, and each author might have written a number of articles, there is a many-many relationship between authors and articles. Representing the entities 'article' and 'author' by records of the same names, we can non-redundantly represent the mapping as shown in figure 8.

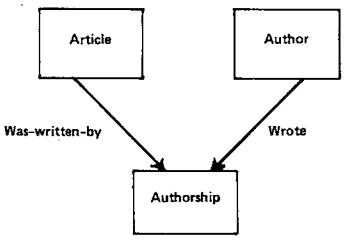


Fig. 8 One-many relationships

Two sets and an extra record type is used to represent the many-many relationship. The extra record, in this case called 'authorship' is called a relational, or relationship record, as it represents a relationship rather than an entity in the object system. The relationship may itself have attributes, for example in this case the authorship record could contain data on whether the author was the major author for that particular article etc.

By coalescing the Article and the Author records in figure 8, the structure illustrated in figure 9 is produced. This can be used to represent the situation

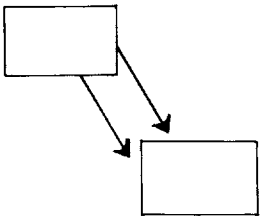


Fig. 9 Many-many relationships

where an entity has a many-many relationship with similar entities. An example is the so called parts explosion problem, where one of the sets could be called 'is-a-subcomponent of', and the other 'has-as-components'. The record owning the sets could represent the 'part' entity, and the member record the 'component' relation. This situation could be represented more naturally using the structure of figure 7, if it were not illegal.

PROGRAM COMMUNICATION WITH THE DBMS

USER WORKING AREA, SYSTEMS LOCATIONS, CURRENCY

As previously stated, an application programmer manipulates data stored in the database by writing a program in a host language augmented with Data Manipulation Language statements. This program also contains a record loading and unloading area called the 'user working area', which has space allocated in it for each data item defined in the sub-schema being used. Values for

data items in a particular record type are set up in the user working area before an instance of that record is stored. and when the data items from a particular record instance are retrieved, the values are put into the corresponding locations in the user working area.

The application program also contains a number of 'system communication locations' which provide status information to the program after the execution of any DML command. One of these locations is used to indicate that the last DML command has encountered an error condition. When this occurs the location is set with a value indicating the type of error, and the database restored to its state before the attempted execution of the DML command.

Each record instance is allocated a unique identifier when it is created. This identifier, called the 'database-key' of the record, is readily available to the programmer. It is a logical identifier, but because of the way that it is used, it should correspond fairly closely to the physical address of the record. This physical correspondence makes it difficult to see how a reorganization, that is change to the storage structure of the data, could be carried out without changing the database keys of many records.

When processing a sequential file, the 'current position' or place member in the file is determined by the last record read. The next record and the previous record are well defined intuitively. When 'navigating' one's way through a CODASYL data structure, as Bachman describes it, current positions need to be kept in many dimensions at once. CODASYL database systems keep the database-keys of the last record of any type accessed by the application program, called the 'current of run unit', and the database key of the last record accessed in each record type, set type or area. These are called the 'current record' of the particular record, set or area type. In general, DML commands will affect these currency status indicators, however it is possible to suppress updating all except the current of run unit.

The currency status indicators are available to the application program and may be used by it to control record retrieval.

The existence of the currency status indicators has been criticised by Engles in [9], as being a potential source of programming error. However, in the vast majority of situations the programmer does not need to worry about their existence. It is worth noting that hierarchically structured database systems such as IMS [10], while they must maintain the equivalent of status indicators to provide tree traversing commands, do not make them available to the programmer. Similarly CODASYL program operating on hierarchical structures does not need to worry about currency. It only needs to operate on them when manipulating structures such as the illustrated earlier for the parts explosion problem which are difficult to represent in hierarchically based systems.

The DDL contains some aspects concerned with the data structuring or model building activities described earlier, and somewhat overlapping these, aspects which are concerned with the way that the data is physically stored.

If the schema designer knows the different operations to be performed on the data, he can, through the DDL, give the database system directions on how to store the data to optimise overall performance. This can be illustrated by describing some further properties that can be declared in the DDL. Some of these, while improving performance, also make application programs less independent of storage structure, and have been attacked on that ground. [10], [11], [12].

AREAS

The database can be divided into a number of logical subdivisions called 'areas'. The most obvious way of implementing these is to make each area a file. Areas can be used as purely logical subdivisions, so that different applications access only a small subset of the areas comprising the database, or as physical subdivisions to make database dumping, etc more manageable.

One particular type of area, a 'temporary' area is used as scratch storage for the duration of one run-unit only.

In the COBOL subschema DDL, areas are known as 'realms'.

SETS

The member records of each set occurrence are ordered, so that an application program may iterate through the set occurrence, and the positional adjectives 'first', 'last', 'next' and 'prior' will have a consistent meaning. This order is declared in the DDL and is maintained by the system.

Record instances are inserted in a particular position when the record goes into a set instance. This position may be 'first', 'last', or 'next' or 'prior' (relative to the current record of the set). It may also be determined by values of data items in the member records ('sorted by defined keys'), in which case it is possible to specify that records with duplicate values of those keys shall be inserted 'first' or 'last' or shall be rejected altogether. The insertion position may be sorted within the set by the database key allocated to the record. All records of a given type in the set may be grouped together, and within that record type, other keys may be specified. A sort order may be specified within records of a given type in the set.

Sets which are sorted may also be specified as 'indexed'. If this is done then extra mechanisms are invoked so that when values of the sort keys are supplied in the user working area, a member record occurrence with key data items equal to the supplied values can be found rapidly, using a DML command. The same DML command can be used if the set is not indexed, but would in that case involve accessing consecutive records in the set sequentially.

One can also specify that set instances should be traversable backwards from last to first member in an efficient way.

The owner record type for the set and each possible member record type must be specified. If the owner of the set is declared to be SYSTEM, the database system supplies a single owner record instance, and there can be only one occurrence of this set in the database.

When inserting a record instance into a set, the system must be able to select a particular occurrence of the set to put it into. (It then uses the ordering specification to find the logical insertion point within that set occurrence).

The mechanism for selecting the set occurrence is described in the DDL. The system could simply take the last occurrence of the set that had been referenced, and insert the record in that. This is indeed an option ('THRU CURRENT OF SET'), but more powerful techniques are available.

CODASYL allows the schema designer to specify an 'access path' starting at some root record instance. The root record must be identifiable using the values of some of its data items set up in the user working area (see LOCATION MODE CALC), or in some other way not depending on the participation of the record as a member of a set.

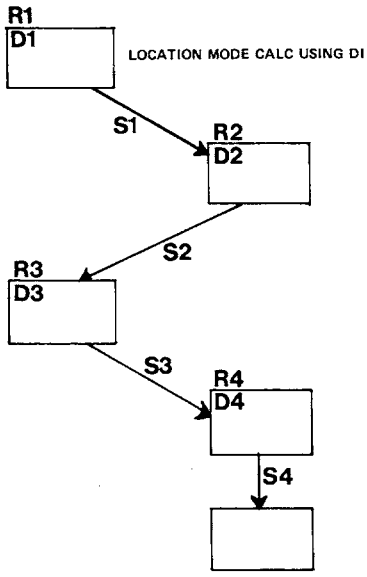


Fig. 10 SCHEMA example

The access path is defined by specifying sets and data items along a path from consecutive owner to member records in the data structure diagram. For example, in figure 10, the path to select an occurrence of S4 would be declared as follows.

SET SELECTION IS THRU SI, OWNER IDENTIFIED BY CALC KEYS
THEN THRU S2, OWNER IDENTIFIED BY D2
THEN THRU S3, OWNER IDENTIFIED BY D3
THEN THRU S4, OWNER IDENTIFIED BY D4.

The values of the data items specified on the access path i.e. D1,D2,D3 and D4 are set in the user working area at run time before the set occurrence selection is to occur. The system would first identify an R1 instance using the value of the CALC key D1. Then it would scan the instance of set S1 owned by that instance of R1 until it found an instance of R2 with the item D2 equal to the supplied value. The instance of R2 would define an instance of S2 to be scanned for a match on D3. This would occur all the way down the hierarchy, for as many levels as necessary, (in this case, three) unitl an instance of set S4 had been identified.

SET MEMBER PROPERTIES

A set type may contain more than one type of record as member. Each of these member types has the following properties defined.

The property AUTOMATIC or MANUAL specifies when a record of this type is inserted into an occurrence of the specified set.

If it is AUTOMATIC, the set selection specified for the set is invoked to select a set when the record is created, and the record is inserted into that set occurrence. If it is MANUAL, the application program must execute a specific DML command before a set is selected and the record inserted.

The property MANDATORY or OPTIONAL specifies whether a record of this type can exist without being a member of an occurrence of the specified set. If it is MANDATORY, it must always be a member of *some* occurrence of the

set, although the program can move it from one occurrence to another. If it is OPTIONAL, it may be removed from a set occurrence with a DML command.

The inclusion of a pointer to facilitate finding the owner of the set efficiently starting from a member can be specified.

If the set is sorted, the keys must be specified for each member type.

In the same way as indexing may be specified on sort keys, other keys called search keys may be specified. An index is set up for each set instance and for each set of search keys specified, to facilitate searching the set for a match with values of those keys supplied in the user working area.

RECORDS

Each record type has a property called 'location mode' declared in the DDL. This property determines the method by which an instance of the record will be stored. There are three location modes, 'direct', 'calc' and 'via set'.

If the location mode is direct, the application program must supply a database key at the time a record instance is created. The record is allocated a database key equal to, or as close as possible to, the supplied database key. That is, it is stored as physically close as possible to the specified position.

If the location mode is calc, certain data items in the record are designated as key data items. Values for these data items must be supplied when the record is stored. The essential feature of location mode calc is that the record instance can be efficiently retrieved at a later stage by setting values for the calc data items in the user working area and issuing an appropriate DML command.

The mechanism specified by CODASYL for calc record placement is that the values of the key data items are submitted to a repeatable randomising, or 'hashing' algorithm to produce the record address. The algorithm used must of course be able to handle 'synonyms', that is records with different keys which produce the same address. There is however, no reason why location mode calc should not be implemented using another suitable access method, for example some form of balanced tree structured index [13].

If the location mode of a record is via a specified set, the record must be a member of the set. When a record instance is inserted into an instance of the set, its physical position in the database is as close as possible to its logical insertion point in the set. This tends to make serial scanning of the records in the set efficient, as they are physically close together and accessed to secondary storage are minimised.

DATA ITEMS

A record's data items usually occupy space in each record instance, and are moved between that record instance and the item's location in the user working area when the record is stored or the data item is retrieved.

There are several cases where this is not true. The schema designer can specify that records which are members of sets can contain an item whose source is a similar item contained in the set owner. If the item is specified as 'actual', i.e. actually stored in the member record, a copy is obtained from the owner and put into the member at the time that the member is inserted into the set. That is, its source is the owner record, not the user working area. If the item is specified as 'virtual', it is not stored in the member record. When the application program retrieves the member record and requests that the virtual data item be transferred to the user working area, the system actually retrieves the owner of the set to obtain the data item. This is provision aimed at maintaining integrity.

The schema designer may also specify that a data item be obtained as the

result of a procedure. In this case, if the item is specified as virtual, it is not stored in the record but is produced as the result of a specified 'database procedure' when the application program requests that the item be transferred to the user working area. If the item is specified as actual, it *is* stored in it's record, and the database procedure is called each time a DML command is issued which could change the value of the item.

A simple example would be a procedural data item in the owner of a set which contained the total number of members in the set. If it was virtual, the procedure would iterate through the set and count the members when the item was to be transferred to the user working area. If it was actual the procedure would be called each time that a record was inserted into or removed from the set, and would either increment or decrement the data item as appropriate.

PRIVACY

Passwords and privacy procedures may be defined to apply to structures in the schema. A program may have to be validated against these procedures before being allowed different sorts of access to the structures that they protect.

ON CONDITIONS

The schema designer may specify the procedures to be executed when exception conditions arise. It would be equally possible for the application program to check the condition and execute the procedure, however specifying this in the schema provides an insurance against program error.

DML COMMANDS

The commands provided in the Data Manipulation Language fall into a number of categories.

1) Area Oriented Commands
   - INVOKE
   - OPEN or READY
   - CLOSE or FINISH
2) Record Updating Commands
   - STORE
   - DELETE or ERASE
   - MODIFY
3) Retrieval Commands
   - FIND (FETCH)
   - GET
4) Manual Set Manipulation
   - INSERT or CONNECT
   - REMOVE or DISCONNECT
5) Control and Status
   - IF
   - MOVE or ACCEPT
6) Set Reordering
   - KEEP
   - FREE

The commands defined first are those given in [1]; some of these conflicted with already defined COBOL reserved words, and alternatives were defined in [4]. Particular implementations of the CODASYL specifications may contain more commands.

Of the area oriented commands, the INVOKE command selects a particular subschema of a schema. The OPEN command prepares an area or areas for processing. It may define the type of access required, and the program may

have to supply passwords to be granted this access. The CLOSE command relinquishes control of one or more areas when processing is complete.

The STORE command specifies a record type. A new instance of that record is created, using the values of the items set in the user working area. The record is placed in the area using the record's location mode. For each set for which the record is an automatic member, a set occurrence is selected. The record is inserted in that set occurrence in the position specified by the set order clause in the DDL.

A record instance may be destroyed using the DELETE command. This may also destroy records which are members of sets owned by the target record.

Data items in the current record of the run unit may be changed using the MODIFY command. If any of the changed data items define a path used in set selection, then the record is moved into the set selected by the new data item values.

There are a number of different varieties of the FIND command, used for retrieving a record instance and establishing appropriate currency status indicators. The GET command is used in conjunction with the FIND command, and transfers specified data items from the current record of the run-unit (i.e. the one just found) to the user working area. These two commands are combined in one implementation into the FETCH command.

One of the FIND varieties retrieves a record instance when given it's database-key. Another retrieves an instance of a record whose location mode is calc when the values of the calc keys are set in the user working area.

Another scans through a set occurrence (possibly selected using the set selection path specified in the DDL) and searches for a record containing values for specified data items equal to values set up in the user working area.

Another variety moves from record to record backwards or forwards through a set occurrence, or through an area.

It is also possible to retrieve the owner record of a set instance, given a member record.

The INSERT and REMOVE commands insert and remove record instances from sets for which the record is MANUAL (for INSERT) or OPTIONAL (for REMOVE).

The IF command branches depending on whether a set instance is empty or not. The MOVE command enables the program to manipulate the currency status indicators directly.

A set may be reordered in the environment of the run unit issuing the ORDER command. Other concurrent or later run units will see the set order as being that defined in the DDL.

If an applications program has opened an area which can be accessed by other programs, and it wishes to update a record instance in that area, it must gain exclusive access to the record to be updated and to all other records which were used to establish that record occurrence. This is to prevent interference from those other programs which would either cause them or the updating program to derive or introduce incorrect information. Exclusive access is gained by the KEEP command, and relinquished by the FREE command. This mechanism puts what some [9] consider to be an overly heavy burden on the application programmer to be aware of all of the places where this mechanism must be invoked. It is probable that if the primitives used by the programmer were higher level and less oriented towards physical implementation, then the management of concurrent access could be taken out of the programmer's

hands to a greater extent.

A simple example may give some of the flavour of writing DDL specifications and DML programs using the CODASYL specifications. Take the data structure diagrammed in figure 8, in which a relationship between journal articles and their authors is being represented. This could be augmented by a set called ARTSET with member type ARTICLE and owner type SYSTEM. This could be specified as follows in schema DDL.

```
SCHEMA NAME IS LIBRARY SCHEMA.
AREA NAME IS LIBRARYJ
RECORD NAME IS AUTHOR
        LOCATION MODE IS CALC USING ANAME
        DUPLICATES ARE NOT ALLOWED,
        WITHIN LIBRARYJ
DATA
        ANAME PICTURE X(20).
RECORD NAME IS ARTICLE
        LOCATION MODE VIA ARTSET SET
        WITHIN LIBRARYJ
DATA
        TITLE PICTURE X(200),
        JOURNLCODE PICTURE X(4)
        DATEWRITTEN PICTURE X(8)
        JOURNLREF PICTURE X(10)
        SUBJECTS PICTURE X(12)
        OCCURS 10 TIMES.
RECORD NAME IS AUTHORSHIP
        LOCATION MODE IS VIA WROTE SET
        WITHIN LIBRARYJ
DATA
        AUTHORTYPE PICTURE 9.
SET NAME IS ARTSET
        OWNER IS SYSTEM
        ORDER IS SORTED INDEXED BY DEFINED KEYS
        DUPLICATES ARE ALLOWED
MEMBER IS ARTICLE MANDATORY AUTOMATIC
        KEY IS ASCENDING TITLE
        SEARCH KEY IS JOURNLCODE
        DUPLICATES ARE ALLOWED
        SET SELECTION IS THRU CURRENT OF SET.
SET NAME IS WAS-WRITTEN-BY
        OWNER ARTICLE
        ORDER IMMATERIAL
        MEMBER IS AUTHORSHIP MANDATORY AUTOMATIC
        SET SELECTION IS THRU CURRENT OF SET.
SET NAME IS WROTE
        OWNER AUTHOR
        ORDER IMMATERIAL
        MEMBER IS AUTHORSHIP MANDATORY AUTOMATIC
        SET SELECTION IS THRU CURRENT OF SET.
```

NOTES

1) The set ARTSET, which groups all ARTICLE records in the database, is

indexed for fast search on the data items TITLE (on which the set is also sorted in ascending order) and JOURNLCODE.

2) The LOCATION MODE of the authorship record is VIA WROTE SET. Because of the resulting physical placement, this will tend to make it more efficient to access an author and then access all his articles, as in example 1 below, rather than the other way around, as in example 2.

EXAMPLE 1

'Given an authors name, output the articles that author has written since the beginning of 1974, with their journal references'.

1) Set ANAME in the UWA to the author's name.
2) FIND AUTHOR RECORD.
3) If status=326, error 'no such author'
4) FIND FIRST AUTHORSHIP RECORD OF WROTE SET.
5) If status=326 error 'he hasn't written anything' go to step 7.
6) FIND NEXT AUTHORSHIP RECORD OF WROTE SET.
   if status=307, finish.
7) FIND OWNER RECORD OF WAS-WRITTEN-BY SET.
8) GET ARTICLE ; DATEWRITTN.
   if DATEWRITN less than '01/01/74' go to step 6.
9) GET ARTICLE ; TITLE, JOURNLREF.
10) Output TITLE, JOURNLREF
    go to step 6.

The status 326 indicates that the find statement failed to find a record and 307 indicates that it has come to the end of the set.

EXAMPLE 2

'Find all authors who have written for the Australian Computer Journal since the beginning of 1974'.

1) Set JOURNLCODE in the UWA to the code for the Australian Computer Journal.
2) FIND ARTICLE VIA ARTSET USING JOURNLCODE.
3) if status=326, error 'nobody in this database has written for the ACJ'. go to step 6.
4) FIND NEXT DUPLICATE WITHIN ARTSET USING JOURNLCODE.
5) if status=0, finish
6) GET ARTICLE ; DATEWRITTN.
   if DATEWRITTN less than '01/01/74' go to step 4.
7) FIND FIRST AUTHORSHIP RECORD OF WAS-WRITTEN-BY SET.
   if status=326 error 'this article seems to have no authors' go to step 9.
8) FIND NEXT AUTHORSHIP RECORD OF WAS-WRITTEN-BY SET.
   if status=307 go to step 4.
9) FIND OWNER RECORD OR WROTE SET.
10) GET AUTHOR ; ANAME.
11) output ANAME
    go to step 8.

In example 1, the program identifies an occurrence of the AUTHOR record. It then iterates through the WROTE set instance owned by that AUTHOR record. For each member AUTHORSHIP record, the owner of the WAS-WRITTEN-BY set is found, and the target data items extracted.

Example 2 is similar except that it traverses the structure in the opposite direction, from ARTICLE to AUTHOR. It must also be enclosed by another

loop to iterate through all the articles for a particular journal.

CRITICISM OF THE CODASYL ARCHITECTURE

The CODASYL Database architecture has been endorsed by the computing community to the extent that most of the larger manufacturers produce database systems based on that architecture. One major exception, IBM and its user-group, has indicated various areas where it feels that the CODASYL architecture is deficient, in [9] and [11].

In [9], Engles states the Guide-Share requirements as being Data Independence, Data Integrity and Application Oriented programming. By Data Independence he meant that the language used by the programmer should be independent of the underlying storage organization, and that the storage organization should be alterable without affecting any application programs. In Data Integrity he includes the concurrent update problem. By Application Oriented programming he means that the user's data sub-language should contain only constructs related to the application itself. To some extent this restates the data independence requirement.

The criticisms essentially come down to saying that the CODASYL DML is too low level. It is said to be needlessly complex, and that to force the programmer to think about such things as AREAS, location mode, currency, set selection, concurrent update conflicts, and the correct use of database keys mitigates against satisfying the requirements outlined.

These criticisms appear to have some justification. However higher level data sub languages have the disadvantage that it is not obvious to see how to implement them efficiently, avoiding too much data redundancy. A certain amount of work has been done on providing the programmer with a relational view of his data, while implementing it using a CODASYL like storage structure [14], [16].

A further attempt to get over the difficulties pointed out in [9] and [11] and by others has been made by ANSI in [18]. It proposes a gross architecture in which object system model, storage structure and the user programmer view is completely separated. It is too early yet to say whether this represents a useful contribution, however it seems safe to say that it poses major implementation challenges.

CONCLUSION

The CODASYL architecture provides a fairly low level, storage structure oriented way of tackling the data storage and retrieval problem. It has been criticized on these grounds, and on tying application programs too closely to particular representations. Nevertheless the DBTG did try to produce an architecture which could be efficiently implemented using todays techniques, and this does not allow one to completely divorce storage structure from data structure. In my opinion they have succeeded over a wide spectrum of applications, certainly better than any other type of system.

REFERENCES

[1] CODASYL Data Base Task Group April 1971 Report. (superseded).
[2] CODASYL Database Task Group April 1971 Report.
Obtainable from ACM order Dept.
[3] CODASYL Programming Language Committee,
CODASYL COBOL Journal of Development.
[4] CODASYL Database Language Task Group
CODASYL COBOL Database facility proposal
Canadian Government Department of Supply and Services, 1973.
[5] CODASYL Data Description Language Committee

Data Description Language Journal of Development
Document C13.6/2:113
U.S. Government Printing Office, Washington, D.C.

[6] ACM Computing Surveys
Special Issue: Data-Base Management Systems
Vol 8, Number 4, March 1976
(contains many extensive reference lists)

[7] Taylor, R.W. and Frank, R.L.
CODASYL Data-Base Management Systems
in 6, pp.76-103.

[8] Bachman, C.W.
Data Structure Diagrams
Data Base 1,2 (1969) pp.4-10

[9] Engles, R.W.
An Analysis of the April 1971 Data Base Task Group Report
Proc. 1971 ACM SIGFIDET
ACM NY pp.69-91

[10] IBM, IMS/VS Manuals
General Information GH20-1260-3
Application Programming SH20-9026-2
System Programming SH20-9027-2

[11] Guide-Share
Database Management System Requirements
Guide-Share Inc., N.Y. 1970

[12] Nijssen, G.M.
Set and Codasyl set or coset in [15], pp.1-72

[13] Bayer, R. and McCreight, E.
Organization and maintenance of large ordered indexes
Acta Information 1,3 (1972), pp.173-189.

[14] Kay, M.H.
'An assessment of the CODASYL DDL for use with a relational sub schema'.

[15] Donque, B.C.M. and Nijssen, G.M.
Data Base Description
Proceedings of the IFIP TC-2 Working Conference on Data Base Description
Data Base Description
North Holland 1975.

[16] Tsichritzis, D.
A Network framework for relation implementation in 15, pp.269-282.

[17] Everest, G.C. and Sibley, E.H.
Critique of the GUIDE-Share DBMS Requirements in Proc. ACM Sigfidet Workshop on Data Description, Access and Control, 1971, pp.93-112.

[18] Steel, T.B. (chairman)
ANSI/X3/SPARC Study Group on DataBase Systems Interim Report.
In ACM Sigmod Bulletin Vol. 7, No. 2, 1975.

4 Relational Data Base Systems

P.N. Creasy

INTRODUCTION

Since the late 1960's when data base usage started to expand rapidly one of the main aims of data base designers has been data independence. The problem of data dependency was instrumental in the development thrust given to the relational data model by Codd in 1970 (Codd 1970). Much had been done in the area of physical data independence, in particular in terms of device independence. But many data dependency problems still existed.

Application programs may be required to know implementation details. For instance the use of the indexed sequential access method (or say direct access in the CODASYL schema) results in the application programmer being required to supply an index and have some knowledge of the file sequence. The index is a performance oriented component and is redundant from an informational standpoint. An optional index does not improve the situation as it will not permit any change in the logical organization of data. Likewise any dependency upon ordering where say the application program relies upon the order of presentation being the same as the stored ordering will fail if any re-ordering takes place.

The logical organization of data (and the method by which it is accessed) to a large extent is determined by the requirements of the application. Any change in the logical structure (of the files) will result in failure when accessing if the application program is not aware of the current logical structure. Bachman's idea of the programmer as a navigator (Bachman 1973) traversing the data base (represented by the CODASYL proposals) will certainly require the programmer to have a knowledge of the logical structure of the data base leading to one of the above possible failures.

The advent of the 'casual user' who only requires 'one-off' answers and does not want to be concerned with the structure of the data base has hastened the evolution away from concern with representation details towards information content. To represent the data base in a form familiar to the 'end-user' is highly desirable. The user may consider networks an unnatural formulation and thus may find the CODASYL model unsuitable.

The user is generally only interested in the information in the data base and what operations can be performed on it. Thus a user requires a view or model of the data (suited to the user) and a language to operate on the model as independently as possible of any storage and access changes which may occur. The relational model provides such a model. It provides a simple view which can be used as the model for a wide variety of users (from casual to the professional programmer/analyst).

RELATIONS

The underlying data structure of the relational model is the table (or flat file).

On the basis of many casual users being familiar with the use of tables this provides a suitable community view. Any representation can be reduced to a table (although perhaps with some redundancy). An example of a table (a rectangular array) is given in figure 1. In the terminology of the relational model the (normalized) table is referred to as a *relation* and each row of the table as a *tuple*.

COURSES

C#	CNAME	LEVEL	LPW
101	CALCULUS	1	3
103	ALGEBRA	1	3
107	BIOLOGY	1	2
204	OPTIMIZATION	2	3
209	ZOOLOGY	2	2
213	BOTANY	2	2

TEACHERS

T#	TNAME	TPOS
3217	ORBOT	LECTURER
4096	FLINN	PROFESSOR
4371	HANSIN	SEN. LECTURER
4589	DOREN	LECTURER
4672	PYKE	LECTURER

TEACH-COURSES

T#	C#	NSE
3217	101	87
3217	103	69
4096	213	35
4371	107	71
4371	209	47
4371	213	49
4589	101	117
4589	204	69
4672	107	27
4672	213	32

Fig. 1 The Course/Teachers data base

The relational approach is based on the mathematical theory of relations, providing a sound theoretical basis. (A rigorous definition has been given to the relational algebra and relational calculus by Codd (Codd 1971a).

In mathematical terms, a relation is defined over sets S_1, S_2, ..., S_n. R is a relation on these sets if it is a set of (ordered) n-tuples i.e. the ith element is from Si. The sets S_1, ..., S_n are called the domains of R. R is a relation of degree n and the number of tuples in R is the cardinality of R. (When using relations we are more concerned with attributes than domains. It is possible to have more than one attribute in a relation defined on the same domain. We should thus also define a 1-1 mapping of the set of attributes into the set of domains.)

In figure I there are three relations:

(1) COURSES, with attributes C# (course number), CNAME (course name), LEVEL (course level) and LPW (the number of lectures per week).
(2) TEACHERS, with attributes T# (teacher number), TNAME (teacher name) and TPOS (position).
(3) TEACH-COURSES with attributes T#, C# and NSE (number of students enrolled).

Viewing the relation as a rectangular array we assume the following properties:

(i) in any column all elements must be of the same appropriate domain,
(ii) each domain value is a single non-decomposable data item.

A *normal* table is defined to have the following additional properties,

(iii) all rows of the table are distinct,
(iv) the ordering of rows within the table is immaterial,
(v) the ordering of columns is immaterial (which can be achieved by assigning distinct names to each column and referencing the column by this name – the attribute name).

Each row can be uniquely identified by its content (property (iii) ensures this). Thus one or more attributes combined, whose value(s) are different for each row, could uniquely identify that row. An attribute (or combination of attributes) which achieves this is called a *candidate key* and the key chosen is called the *primary key*. We assume the primary key is non-redundant, i.e. each of the components is essential for the unique identification in at least one row. In the above relation C# or CNAME (assuming unique names) would be candidate keys. Choosing C# we write the relation as COURSES (C#, CNAME, LEVEL, LPW) with the primary key underlined.

Relations can be easily manipulated to form new relations. Codd (Codd 1971a) proposed a relational calculus and relational algebra for this purpose.

The *relational algebra* has a number of operators to select data from the data base. The traditional set operations union, intersection and difference are defined in the usual way but operate on two relations (with compatible sets of attributes) treating each relation as a set of tuples and producing a single relation.

The *projection* operation (operating on a single relation) essentially is a column selection process. For example

COURSES [C#, CNAME]

produces a new relation consisting of the columns C# and CNAME from COURSES as in figure 2. This relation may be named and is available to the user until deleted or until the end of the run. Under certain circumstances it may be possible to make the relation part of the data base. The attribute names of the created relation are inherited from COURSES and the columns are specifically ordered. Any duplicate tuple is eliminated.

C#	CNAME
101	CALCULUS
103	ALGEBRA
107	BIOLOGY
204	OPTIMIZATION
209	ZOOLOGY
213	BOTANY

Fig. 2 Sample projection

T#	TNAME	TPOS	C#	NSE
3217	ORBOT	LECTURER	101	87
3217	ORBOT	LECTURER	103	69
4096	FLINN	PROFESSOR	213	35
4371	HANSIN	SEN. LECTURER	107	71
4371	HANSIN	SEN. LECTURER	209	47
4371	HANSIN	SEN. LECTURER	213	49
4589	DOREN	LECTURER	101	117
4589	DOREN	LECTURER	204	69
4672	PYKE	LECTURER	107	27
4672	PYKE	LECTURER	213	32

Fig. 3 Sample join

The *join* operation has two relations as arguments. A tuple from one relation is concatenated with a tuple from the other whenever a given condition holds between them. All such concatenations form a new relation. The condition is of the form

A r B

where A and B are attributes of the relations defined on the same domain and r is a mathematical relation such as = or >. If r is restricted to = if the attributes are identical it is called the *natural join* and the duplicate column is eliminated from the concatenation. For example

TEACHERS [T# = T#] TEACH-COURSES

which is the natural join of TEACHERS AND TEACH-COURSES over T# produces the relation as in figure 3. Obviously if the join was always restricted to the natural join then a simpler specification could be used.

If the second domain is a constant then the second relation is not present and is assumed to be a system supplied relation containing that constant with a domain name the same as the first domain. For example

TEACHERS [TNAME = 'HANSIN']

produces the relation of figure 4(a), and

TEACHERS [TNAME = 'HANSIN'][TN#]

results in the relation of figure 4(b).

(a)

T#	TNAME	TPOS
4371	HANSIN	SEN. LECTURER

(b)

T#
4371

Fig. 4 Sample queries

The projection and join operators provide a powerful query facility. To find all the information on teachers taking course number 207 the following expression would be used:

TEACHERS [T# = T#]TEACH-COURSES[C# = '207']

and to find who teaches ALGEBRA:

TEACHERS[T# = T#]TEACH-
COURSES[C# = C#]COURSES[CNAME = 'ALGEBRA'][TNAME]

Other operations such as restriction and division are also allowed.

NORMALIZATION

To enable future changes to be made to the data base without the need to restructure current relations, and to overcome some anomalies concerned with adding to, deleting from and updating the data base, further changes must be made to the relations so far discussed in order to produce a canonical normalized form.

Codd (Codd 1971b) has proposed three levels of normalization, first, second and third normal forms. The normalized relations discussed so far are *first normal form*. Property (ii) of the last section essentially defines first normal form.

In the relation TC(T#, C#, TNAME, TPOS, BSR, NSE) of figure 5, BSR is the bottom of the salary range for the given position. It can be seen that information concerning a teacher cannot be added until a course is to be given by the teacher, unless a fictitious course is introduced – an insertion anomaly. A course cannot simply be a null value as it is part of the key. Likewise deletion of the only course a teacher takes will delete the teacher – a deletion anomaly. Any updating of information associated with the teacher (e.g. a promotion) will involve updating as many triples as there are courses taken by the teacher, leading to inconsistencies between the commencement and completion of the update – an update anomaly.

TC

T#	C#	TNAME	TPOS	BSR	NSE
3217	101	ORBOT	LECTURER	13100	87
3217	103	ORBOT	LECTURER	13100	69
4096	213	FLINN	PROFESSOR	25000	35
4371	107	HANSIN	SEN. LECTURER	18000	71
4371	209	HANSIN	SEN. LECTURER	18000	47
4371	213	HANSIN	SEN. LECTURER	18000	49
4589	101	DOREN	LECTURER	13100	117
4589	204	DOREN	LECTURER	13100	69
4672	107	PYKE	LECTURER	13100	27
4672	213	PYKE	LECTURER	13100	32

Fig. 5 The teacher-course (TC) relation

Removal of these irregularities involves splitting the relation. However before doing this it is best to first look at the causes of these problems – functional (in)dependencies.

An attribute B of a relation R is *functionally dependent* on an attribute A of R if (at any time) each value in A has no more than one value in B associated with it in R, i.e. A identifies B. We write A→B. The functional dependencies of the relation TC are shown in figure 6. An attribute B of a relation R is *fully functionally dependent* on an attribute (or collection of attributes) A if it is functionally dependent on A but not any subset of A.

Finally we say that an attribute is a *prime attribute* if it is a member of at least one candidate key.

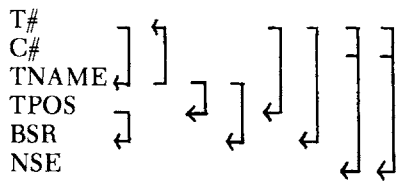


Fig. 6 Functional dependencies in the relation TC

A normalized relation (first normal form) is said to be *second normal form* if every non-prime attribute of R is fully functionally dependent on each (candidate) key of R.

It can be seen from figure 6 that TPOS and BSR are not fully functionally dependent on the key T#, C# or TNAME, C#. If we split TC into T(T#, TNAME, TPOS, BSR) (as in figure 7) and COURSES (T#, C#, NSE) then each of these relations is in second normal form as TPOS and BSR are fully functionally dependent on the key T#. Now information concerning a teacher can be added and deleted without a course being allocated to the teacher.

T

T#	TNAME	TPOS	BSR
3217	ORBOT	LECTURER	13100
4096	FLINN	PROFESSOR	25000
4371	HANSIN	SEN. LECTURER	18000
4589	DOREN	LECTURER	13100
4672	PYKE	LECTURER	13100

TEACH-COURSES

T#	C#	NSE
3217	101	87
3217	103	69
4096	213	35
4371	107	71
4371	209	47
4371	213	49
4589	101	117
4589	204	69
4672	107	27
4672	213	32

Fig. 7 Relations in second normal form

However the kind of anomalies mentioned above can still occur because of transitive dependencies due to the non-prime attributes not all being mutually independent. For example TPOS and BSR are functionally dependent on each other. This results in problems such as changing the salary range for a certain position and resulting in the update anomaly.

We now say that a normalized relation is *third normal form* if it is second normal form and each of the non-prime domains are mutually independent. Thus BSR should be removed from T and T split into TEACHERS (T#, TN, TPOS) and POSITIONS (TPOS, BSR) as in figure 8. The relations COURSES, TEACHERS, TEACH-COURSES and POSITIONS are in third normal form.

The data base designer, presumably the data base administrator, would need to be aware of the functional dependencies. As the data base was expanded any additional attributes to be added to any relation could only be added if the 'mutually independent', and 'fully functionally dependent' conditions held, otherwise a new relation would need to be created.

LANGUAGES AND IMPLEMENTATIONS

Previous to Codd's work proposals for relational-type systems had been presented. CODASYL (CODASYL 1962), for instance, proposed an 'information algebra'. A number of earlier systems certainly used binary relations in systems — particularly in the artificial intelligence area.

TEACHERS

T#	TNAME	TPOS
3217	ORBOT	LECTURER
4096	FLINN	PROFESSOR
4371	HANSIN	SEN. LECTURER
4589	DOREN	LECTURER
4672	PYKE	LECTURER

POSITIONS

TPOS	BSR
PROFESSOR	25000
READER	22500
SEN. LECTURER	18000
LECTURER	13100

Fig. 8 Relations in third normal form

The *MacAIMS data management system* (Goldstein and Strand, 1970) (an algebraic-based sublanguage) is recognized as the earliest system providing a full relational model. It was developed for use on the MULTICS system at MIT. It did not introduce any new theoretical concepts but rather implementation concepts. The system is logically divided into two parts — data and relations.

Essentially each 'data type' (domain type) is stored via a separate 'procedure segment' in a 'data element set' (i.e. a procedure and set for each domain). The names of the data element sets (domain names) were themselves stored in a data element set allowing for the system to be self descriptive. As each new data element entered the system a fixed length reference number was assigned to the element and this number was used in all subsequent operations on that element. The algorithm for assigning the number ordered the numbers according to the ordering of the data allowing for easier comparison — identical data items were given the same reference number.

The relations were stored in a similar manner — being identified by a reference number and allowing a variety of relation types (each handled by its procedure segment) which could be converted to a canonical form (the usual tabular form), if necessary, for operations.

Since Codd's first paper on relational models a number of models have been proposed. The first of these was Codd's data sublanguage ALPHA (Codd, 1971a) which was based on *relational calculus*. It is much more mathematically oriented than the relational algebra but is less procedural allowing greater scope for search optimization. The sublanguage allows for retrieval (with ordering etc.), storage and the use of library functions.

The basic format of the retrieval is

GET workspace target-list: qualification expression.

The target-list specifies the attributes of the relation to be returned (essentially as a relation) to the specified workspace. The qualification expression is a condition which is used to select the particular tuples. For example: 'get the number of the teacher Hansin'

GET W (TEACHERS, T#): TEACHERS. TNAME= 'HANSIN'

'Who teaches course 207?'

GET W (TEACHERS.TNAME): TEACH-COURSES (TEACH-COURSES.C#= '207'ATEACH-COURSES. T#= TEACHERS.T#)

This reads as 'get the names of all the teachers (TNAME from the TEACHERS

relation) such that there exists a TEACH-COURSE tuple with its C# value as 207 and its T# value the same as that in the TEACHERS tuple', i.e. we are restricting the teachers tuples to those where we can find a TEACH-COURSE tuple with a C# value of 207 and a T# value matching the T# value of the TEACHERS tuple.

(Strictly we should not use TEACH-COURSES with the existential quantifier but instead use a RANGE statement to specify a variable in its place.)

The following (non unique) ALPHA statement will retrieve the names of those teaching algebra.

GET W (TEACHERS.TNAME): TEACH-COURSES(TEACH-COURSES.T# = TEACHERS.T# ∧ COURSES (COURSES.C# = TEACH-COURSES.C# ∧ COURSES.CNAME = 'ALGEBRA')).

Once again range variables should be used for 'quantified variables'.

Codd (Codd, 1972a) defined 'relational completeness' of data base sublanguages. The completeness is in terms of selection capability (independent of any host language). Codd defined a sublanguage to be relationally complete if the expressions of the sublanguage permits definitions of any relation definable from the relational calculus. That is, the relational calculus was defined as the standard, and any sublanguage was relationally complete if it permitted equivalent queries. Codd also showed the relational algebra was complete allowing any sublanguage to be compared to the calculus or the algebra, whichever was most convenient. A number of the sublanguages since proposed have been shown to be complete, in particular SQUARE and SEQUEL.

The relational algebra and relational calculus both have some disadvantages, the algebra being somewhat procedural and the calculus being mathematically oriented. It is often particularly difficult to express a problem in the calculus. To remove these problems the so-called 'mapping languages' SQUARE (Specifying Queries as the Relational Expression) (Boyce et al, 1975) and SEQUEL (Structured English Query Language) (Chamberlain and Boyce, 1975) have been proposed. Intended for the 'casual user' they are non-procedural without the quantifiers etc. of the relational calculus.

The format of a SQUARE statement is

	relation name	(argument)
range	attribute name	

This will retrieve the set of 'range' values from the named relation whose associated attribute values (i.e. from the same tuple) match the argument. For instance

	TEACHERS	('HANSIN')
T#	TNAME	

will find HANSIN's number, while

	TEACHERS ∘	TEACH-COURSES	('207')
TNAME	T# T#	C#	

will return the names of teachers taking course 207. Thus the 'expressions' may be nested to any depth.

A similar approach is taken in SEQUEL which is a less terse and more readable language. The equivalent expressions in SEQUEL are

```
SELECT  T#
FROM    TEACHERS
WHERE   TNAME = 'HANSIN'
```

and

```
SELECT  TNAME
FROM    TEACHERS
WHERE   T# =
        SELECT T#
        FROM TEACH-COURSES
        WHERE C# = '207'
```

While many of the relational systems that have been proposed have essentially been relatively small sub-systems, several large experimental prototype data management systems are in the process of being implemented. One of these, *System R* (Astrahan et al, 1976), provides both storage and data interfaces which allow a number of sublanguages (e.g. SEQUEL) to be supported. The data interface consists of a set of operators which may be called from any host programming language. The supporting system contains an optimizer which selects a low cost acess path from those available. The storage interface provides access to invalid tuples with its supporting system providing all storage access facilities e.g. device management, space allocation, deadlock detection. The total system is intended to provide a complete support for many users with high volume requests.

The systems covered in this paper and in fact most of the relational systems written which have been based on Codd's ideas have been non-inferential. Despite the intention that they provide a facility for the casual user, they nevertheless require the users to have a knowledge of relations, the relations in the model and to be able to express their problems in a language which may be difficult to remember if they use it infrequently. Thus it seemed desirable to allow the users greater freedom in their requests to the system and be able to employ their native languages. Codd (Codd 1974) has proposed (and partially implemented) a system *(RENDEZVOUS)* which enables users to engage in a dialogue with a relational data base management system. The system allows the user to carry on the dialogue in the natural language until it cannot make enough sense of a statement at which point the system provides the user with a multiple choice question. The dialogue then resumes in the natural language until an 'agreement' has been reached as to both the user's and the system's requirements.

SUMMARY

This paper is intended to present a brief overview of the relational data base model and some of the systems that have been proposed and written. The paper has concentrated on the aspects of data base systems that are purely relational. Important aspects of data base systems such as integrity and security have received little or no mention. However an important feature of the relational model is that it allows separation of integrity and security constraints from the logical data structure. A number of ways (e.g. Date, 1975) have been proposed for specifying the integrity and security constraints.

The relational model has a number of advantages over other models that have been proposed or are in use. These include data independence, simplicity, non procedurality, ease of extension and symmetry (i.e. the data model struc-

ture doesn't make some questions easier to ask than others — as in say the hierarchical model).

Finally, relational data base systems haven't yet proved themselves. If some of the large system (e.g. System R) are showed to work well maybe they will be accepted by the industry. However it may need a change in the memory structure of computers for relational systems to come into their own.

REFERENCES

Astrahan, M.M. et al., 'System R: relational approach to data base management', *ACM Transactions on Data Base Systems* 1, 2 (June 1976), pp.97-137.

Bachman, C.W., "The programmer was navigator", *Comm ACM* 16, 11 (Nov. 1973), pp.653–658.

Boyce, R.F., Chamberlin, D.D., King, W.F. and Hammer, M.M., 'Specifying queries as relational expressions: the SQUARE data sublanguage', *Comm ACM* 18, 11 (Nov. 1975), pp.621-628.

Chamberlin, D.D. and Boyce, R.F., 'SEQUEL: A Structured English query language', *Proc. ACM-SIGMOD Workshop on Data Description, Access and Control*, May 1974, ACM, New York, 1975, pp.249-264.

CODASYL Development Committee, 'An information algebra', *Comm ACM* 5, 4 (April 1962), pp.190-204.

Codd, E.F., 'A relational model of data for large shared data banks', *Comm ACM* 13, 6 (June 1970), pp.377-397.

Codd, E.F., 'A data base sublanguage founded on relational calculus', *Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control*, Nov. 1971, ACM, New York, 1971, pp.35-68.

Codd, E.F., 'Relational completeness of data-base sublanguages', Courant Computer Science Symposia 6, 'Data Base Systems', New York, May 1971, Prentice Hall, Englewood Cliffs, N.J., 1972, pp.65-98.

Codd, E.F., 'Further normalization of the data base relational model', Courant Computer Science Symposia 6, 'Data Base Systems', New York, May 1971, Prentice-Hall, Englewood Cliffs, N.J., 1972, pp.33-64.

Codd, E.F., 'Seven steps to RENDEZVOUS with the casual user', *IFIP TC-2 Working Conference on Data Base Management Systems*, April 1974, North Holland, Amsterdam, 1974, pp.179-200.

Date, C.J., 'An Introduction to Database Systems', Prentice-Hall, 1975, pp.285-316.

Goldstein, R.C. and Strand, A.L., 'The MacAIMS data management system', *Proc. 1970 ACM-SIGFIDET Workshop on Data Description and Access*, Nov. 1970, ACM, New York, 1970, pp.201-229.

5 The Infological Approach to Data Base

J. Firth

THE WORD 'INFOLOGICAL' AND ITS SWEDISH ANCESTRY

The word 'infological' is an artificial or made-up word that has been used to describe a general systems approach to the information systems analysis and design problem. It emanates from Stockholm, Sweden and describes the very original solution to this problem suggested by Prof. Börje Längefors of the University of Stockholm and applied to data base system design by Dr. Bo. Sundgren of the Swedish National Bureau of Statistics. The principal initial results of this work are described in references [4] and [6]. What follows is attempt to describe some of the implications of these ideas.

This author must claim some responsibility for the suggestion that 'infological' sounded much better in English than either 'informatological' or 'information-logical' despite expert philological advice that it was meaningless. His connection with the infological approach was through spending part of a sabbatical year at the Swedish National Bureau of Statistics as a consultant to the Data Processing Methods Department.

THEORETICAL ANTECEDENTS

The basic approach is a general systems one first enunciated by Prof. Längefors in [3].

"Partition the systems work into separate tasks, 1 through 4,

1. *Definition of the system as a set of parts.*
   List all the parts from which the system is regarded as built up.
2. *Definition of system structure.*
   Define all interconnections which make up the system by joining its parts together.
3. *Definition of the systems parts.*
   For each single part (or group of similar parts) separately define its properties as required by the system work at hand and do this in a format as specified by the way the systems structure is defined (in task 2).
4. *Determination of the properties of the system.*
   Use the definitions as produced by tasks 1, 2 and all separate tasks 3, all taken together. Compare with specifications for the system and repeat, 1,2,3 and 4 until satisfied."(p. 51)

On the basis of this iterative and recursive (structured) prescription he goes on to develop a systems algebra involving precedence relations and matrices which he applies to the problem of design of information systems.

This approach has great theoretical appeal but has been largely ignored by information systems analysis and design practitioners. Perhaps this is because of the awkward style in which the theory is presented, and the apparent sophistication of the methodology. However, for anyone taking the trouble to inwardly digest the slightly turgid prose-style and the set theoretical

methodology, it will soon become clear that defining precedence relations between information sets in an information system is what information systems analysis is all about. Likewise, information systems design embodies the tasks of grouping data processes into programs and consolidating data resources into files on which these programs will operate.

THE REQUIREMENT FOR AN INTEGRATED FRAMEWORK TO DATA BASE DESIGN.

One of the largest data management problems has always been that of national statistical data processing — the paper by D. Erbacher addresses some of these problems. However, it was the need to develop an integrated framework for the design of a data base, or data bases, on this scale that led to the involvement of a National Statistics Bureau in the project. In a historical sense this is not so surprising. Herman Hollerith's mechanical tabulation using punched cards had its first large scale use in 1890 US Census processing. First specifically non-scientific data processing computer (UNIVAC I) was installed in the US Bureau of the Census.

In any case, this integrated framework had as its aims:

1. To enable people who were not data processing professionals to co-operate actively and constructively in data base design projects;
2. To make it possible to transform systematically the problems, desires, and requirements of those who are affected by a projected data base into problems that can be tackled by data processing specialists;
3. To enable data processing specialists to analyse the computer-oriented data base problems systematically and with sufficient precision;
4. To make it possible to design data bases with which decision makers, planners, and researchers within different specialised fields can interact constructively, even if the information needs of the interactors are complex, and even if they lack knowledge about computers and computing.

Authorities within the computing trade either implicitly or, more often, explicitly deny that it is feasible to cover all the above aspects within one framework. The infological approach rejects this assumption and attempts to give a methodology by which it may be done. Certainly, unless it can be done, no matter how good are the computer software systems to handle data bases, so-called, the lack of tools to ensure their integration into increasingly complex information systems will prevent the full realization of their potential as aids in information processing.

THE ANALYSIS OF INFORMATION SYSTEMS, DATA SYSTEMS AND DATA BASES

Any problem in systems analysis raises two kinds of questions:

1. *What* do we want the planned system to achieve?
2. *How* should the system be constructed?

It seems to be a flaw embedded in human nature that we tend to escape the 'what' questions in favour of answering only the 'how' questions. Even though it is logically clear that the former are logical precedents for the latter.

The data processing field is no exception to this general pattern. In particular, the problem of data base management is the part of the iceberg that is currently visible. Every computer man and his robot dog (to mix the metaphors) is talking about, constructing, marketing data base management software. The infological framework is a first, perhaps primitive, attempt to bridge the gap between the gross, inexact management perspective in which the 'what' questions tend to be answered, and precise syntactic and semantic

description of the computer technical solution of the 'how' questions.

Although this paper will concentrate on suggesting how the infological approach provides an analytic means for answering the 'what' questions both Längefors and Sundgren show in some detail that the 'how' questions can also be answered using this infological approach as its basis.

When one begins to analyse the environment of a data base one is faced with the task of defining a homomorphic model of some 'slice of reality'. There are several stages in the process of creating a more or less permanently maintained digital data model of this slice of reality.

Firstly, one needs to *abstract* a subject matter model which contains all the parts (and the relationships between them) in which we are interested. Secondly, one should *specify* the information entities (and the relationships between them) to make up the infological model of the slice of reality in which we are interested. Thirdly, we need to *design* a data-logical model which contains or spans our infological requirements. Fourthly, we need to *implement* and *operate* our data-logical model as a data base which will reflect the slice of reality, and the way it changes.

Mostly computer-technical experts are concerned with the third and fourth stages of specification and implementation. The infological methodology is concerned primarily with the first and second stages and only then, in terms of these, with the third and fourth stages.

THE PROPERTIES OF A DATA BASE IN USER TERMS

In infological terms, a data base has the following properties:

1. A data base is a system for the storage and retrieval of data. It is a component subsystem of a particular information system, which itself models some 'slice of reality'. The definition of this slice of reality might itself be called an object system.
2. A data base serves more than one user-profile. A single user-profile may embody many users, but they will all have similar information requirements. Not all the user profiles may necessarily be identifiable at design time.
3. The purpose of a data base must be formulated in such a way that it is always possible to decide, for a particular request for the storage, processing or retrieval of some element of information, whether that request falls within this purpose.
4. Any request for information from the base must be expressible in a task oriented language in which each request can be formulated *solely in terms of the information required.* It should never be necessary to describe how the information is to be obtained from the data as part of the request.
5. For any request for information from the base, the response delay should be short enough for the requestor not to be hindered in his work by having to wait. This does not necessarily imply a real-time or instantaneous response; a week, or a month, or a year may well be a 'short enough' period!

Not all user required data bases will need each of these properties in its full generality. Indeed most working data base implementations are restricted in scope in one or more of the above dimensions.

The user — concepts and data structure independent access

One interesting implication of the human users' need being the basis for the data base design is that it is the conceptions and pre-conceptions in the mind of the user that determines what data is used to satisfy a particular information

need. Different data may have to be used to supply approximately the same information to distinct users. Some users may not be able to accept, let alone assimilate, specific information because they lack the necessary concepts.

Thus, a part of the interaction process between the user and the data base must be for the user and the data base to agree as to what information is supplied. This will mean reaching infological agreement as to what data-attributes supply the information as well as data-logical agreement as to what processing, file (and thus, time) resources are feasible. The need for this is recognized in computing literature when reference is made to the concept of a data dictionary.

Another implication of the approach is that users (or applications programs) should only have to specify requirements in terms that they themselves think natural, and the system will then retrieve or compute the data conveying this information *regardless* of how the data is stored or structured at a data-logical level. This is a very awkward and usually impossible requirement. The infological approach makes feasible the provision of such 'data structure independent access' (see p.xi of [4]) to information within a data base. For, providing the underlying infological model remains, the data structure may be changed without changing user's requirements (or programs), and vice versa.

BASIS OF THE INFOLOGICAL APPROACH TO DATA BASES

A full consistent definition of the infological approach is given in [6]. What follows is but a pale outline, interested readers are referred to at least the above two references.

The principal purpose of a data base is to receive, retain and produce information about some slice of reality.

This slice of reality:

> S contains different kinds of objects. At any point of time every object in S possesses a set of properties. Some of the properties of an object are local, i.e. they are independent of the existence and properties of other objects in S. Other properties are relational, i.e. they are dependent upon the object's relations to other objects in S. [5] p.61.

Thus, the approach rests on four fundamental concepts:

object
property
object-relation
time

Using these concepts, one can build an object system which reflects the slice of reality. For example, we can define a group of objects having a particular property, then derive a time slice of object group as those objects having the property at a particular time.

If we now add a fifth concept called 'reference', we can structure information or messages as references to entities in the object system. References and messages are non-physical entities, but they can be represented as data in file structures. Such datalogical structures can be allocated to particular storage devices (memories). The important point to note is that the data base, as stored, reflects both the infological structure of the represented information model as well as the physical storage and access-structure of the particular memory device(s).

Thus, there is a clear distinction in the infological framework between

—the object system
—information about the object system
—data representations of this information

—storage and access media to which the data has been allocated.

When a data base is to be systematically designed with the constructive participation of its users, the design tasks concerning the infological definition of the data base must be attempted *before* the datalogical aspects which are deliberately postponed until later steps. In a particular case, we must:

—define what objects in the 'slice of reality' are of interest, what properties they have, what relations between them are of interest and over what time span we want them considered.

—define what names we are going to use to refer to these objects system entities.

For example:

Suppose we define a set of objects as 'persons', a property 'age in years' and a relation 'owning a car'. We are now in a position to classify a set of references to members of the object group

'persons owning a car'

Thus, we can have corresponding information entities or messages referring to that object group with particular values of the property 'age in years'.

In general, messages will have the form of a triple $<\alpha, \beta, \gamma>$,

where α is a reference to an object or a set of objects

β is a reference to a property attribute or an object relation attribute. β may well be a compound reference : for example <attribute name, value>.

γ is a reference to a time point or interval.

A particular message (or information entity) can be represented by some data items, called a data entry. Sets of such data entries will make up a file. The structure the data entries may take can be decided by a data designer who may easily make that decision on the basis of particular storage or processing criteria he has or has been given. The important point is that he can make a decision to choose a particular data structure to suit the information structure the user has already specified.

Clearly the difference at this particular point is the analysis between the infological approach and the so-called 'relational' approach of Codd [1] is merely one of emphasis.

The Infological Data Base

The term 'data base' has become a most over-used word and now means almost anything the user has in mind. Perhaps it is necessary for data base users (whoever they are), information systems analysts, computer specialists to have slightly different opinions as to the meaning of the concept. The infological view of a data base is that basically it is a black box reservoir of information. This is a user's view. The ultimate user of a data base is an information consumer who uses the data base as a source among other information sources. He assumes (or should be able to assume) he is able to retrieve information contained in his own frame of reference, whatever that may be.

However, whenever a user requests/receives information from a data base, there are problems of both confidence in, and interpretation of, what is supplied. The information supplied may have a significant uncertainty associated with it. It could be deliberately or inadvertently biased. The assumptions of the infological model may not be clear, or what the user assumed them to be. Thus, the data base must also contain information on the information, on the infological model underlying the base, as well as information about the data

representations (for example, file descriptions).

The advantages of this infological view should be obvious. The data base designer is encouraged to interact with the user to resolve the 'what' question first and, in doing so, to focus on non-data-logical problems concerning quality and interpretation. 'How' questions such as implementation of file and data structures, selection of programming language or software package, storage devices to be used, can be postponed until user requirements are established.

The Environment of a Data Base

The environment of the data base may be defined as the identifiable subsystems that are more or less closely related to, but not necessarily under the control of, the data base. The arrows indicate principal information and control flows. In particular, inputs to and outputs from the data base are the data base transactions. From an infological point of view, these are conceptual messages or, perhaps more precisely, data representations and conceptual messages.

Observed and Controlled Object Systems

There are necessarily two images of the object system that are relevant. One that can be observed; one that can be controlled. The degree to which they overlap will dictate the kind of problems facing the data base designer.

If the two images do not overlap, there may be no motivation for an object in the observed system to supply relevant information. On the other hand, if they do overlap there will be motivation for an object to supply information that could distort the base to react favourably.

Information Supply and Consumption

These functions involve all kinds of resources and activities engaged in supplying or utilizing (consuming) information going to or coming from the data base.

Supply:	Observation and measurement of object system phenomena.
	Questionnaire design and interviews
	Coding and data registration
	Transmission
	Editing of input information and data
Consumption:	Identification of information needs (in the operational sense)
	Request formulation (in terms of the particular infological model)
	Transmission, receipt, interpretation of responses from data base
	Influencing and making decisions concerning the object system

Data Base Goal Stating, Decision Making and Administration

For the purposes of this exercise these are the 'real' *goal setters and decision makers* who may not necessarily be those with formal authority to do so. Quite often these real decision makers have considerable power to establish new, or reject existing goals, to produce or resist change. This function will be most active during planning and development stages in the life of a data base. One task of this system is to combine, reconcile and adjudicate among the interests of all other systems.

The *database administration* subsystem embraces all the planning, systemeering, programming, operating and supervisory activities connected with the design, construction and running of the data base. It includes both data-logical and infological restructuring. An example of data-logical restructuring might be a change in a file structure to speed up certain response times or to save secondary storage. An example of infological restructuring might be

the elimination of an object type or the changing of the secrecy status of a particular attribute.

The Subsystems of an Infological Data Base

The *Schema* is equivalent to a statement of the infological model underlying the data base. It has two distinguishable sub-functions.

Firstly, the Schema has a *deductive* mission to amplify and extend explicit information (stored in the nucleus) using various embedded derivation rules to generate implicit messages. These rules need not necessarily be deterministic. A sophisticated Schema may well be able to provide statistical estimates of the validity of messages provided by the base.

Secondly, the Schema has a *semantic* mission to convey the 'true' meaning of messages contained in the data base to the user. There are many different people in different roles who will interact with a data base. In general, they will have different frames of reference, and a particular person's frame of reference will probably vary over time. An example of how this role might work is the user who begins to interact with the data base, but has no clear problem definition in terms of the underlying infological model of the base. It is a part of the semantic mission of the Schema to converge on a mutually agreeable definition before the base responds with messages containing information.

The *Nucleus* of a data base is the set of messages which, in conjunction with the Schema, can generate the information contents of the data base.

Any message in the nucleus may be true, false, or meaningless. One task of the filter will be to prevent messages that are false or meaningless from entering the nucleus in the first place. Inevitably, they will get in since a filter function is unlikely to be perfect. It may also be true that the 'infological distance' of a particular false message from its true counterpart may well be quite short. So for practical purposes the false message may well be as useful as the true one. For example, in a statistical data base whose output messages are usually aggregates, it is quite often convenient to impute missing individual messages. Such imputed messages are almost certainly false but are still useful.

The *filter* function surrounds the data base in the sense that it protects the data base and its users from each other. It has a role as a quality controller on both input and output messages. It must prevent disclosure of messages to unauthorized users. In this capacity, the same physical data may appear infologically different to different user profiles.

Let us take an example. Suppose the nucleus of a particular data base contains the following elementary messages:

1. p is a well-known politician
2. q is a famous actress
3. p and r are brothers-in-law
4. s is q's fiance
5. r murdered s yesterday

It would not be difficult to devise a Schema for the base so that it also contains the following imputed message:

6. A well known politician's brother-in-law murdered the fiance of a famous actress yesterday.

The filtering function may well have allowed this message to be transmitted to some user interacting with the base. However, when a further request for the names of the objects referred to by message 6 were requested, perhaps the filtering function prevented the elementary messages being transmitted to this particular non-privileged user.

Data Base Interactions

One can begin considering the dynamics of a data base by considering how the subsystems of a data base may be changed.

The most frequent way a data base is changed is by changing the nucleus; by adding or deleting messages. Unless there is particular redundancy the information contents of the base must be changed. If we change the Schema, the information contents *may* be changed. Changing an object group definition will change the information contents; but adding a new attribute will not, until references to the new attribute arrive in the nucleus. Changing the filter will not have direct effects on the information contents, although changing the confidentiality criteria may well drastically change a particular user's frame of reference.

One could classify data base interactions in terms of the subsystems from its environment that communicate with it; in particular, the following functions:

information supply
information consumption
data base administration
computer
other data bases.

A more interesting classification consistent with our view of the data base as a black box is based on the so-called S-O-R model. An input transaction is the stimulus, the data base is the organism, and an output transaction is the response. Now we can classify the interactions according to whether they have:

Φ a null stimulus/transformation/response
1 a non-null stimulus/transformation/response
° any (null or not) stimulus/transformation/response.

Amongst *spontaneous* behaviour interactions might be internally generated warnings or low priority maintenance tasks. Amongst *triggered* behaviour will be transactions that modify or transform the data base, compared with *query* transactions which supply information of some kind.

A data base transaction will have two parts: an operator which will be a specific type of stimulus; and a parameter which provides whatever process is initiated by the stimulus with some input.

Modification Transactions

Operator	*Parameter*
ADD	message
DELETE	message
SUBSTITUTE	message 1, message 2

These are the transactions that would be expected. The traditional up-data situation would be to substitute message 1 of the form:

<object i, <attribute A, value x>, time t_n>
with a message 2 of the form
<object i, <attribute A, value y>, time t_{n+1}>

Query Transactions

The triggering stimulus of a query transaction will normally result in a non-trivial reply transaction from the data base to the interactor. It is possible to distinguish between at least three structurally different query transactions.

(i) Yes/No queries: whose parameter part will be a complete message the veracity of which is to be confirmed by the data base.

(ii) Retrieval queries: whose parameter part will be an incomplete message, the missing or ambiguous part of the message will be supplied by the data base.

(iii) Processing queries: whose operator part or parameter part will specify a set of messages to be retrieved; for example, a request for aggregation or statistical analysis.

CONCLUSION

The Data-logical Design Task

It was stated earlier there are two kinds of questions a systems analyst needs to answer:

1. *What* do we want the planned system to achieve?
2. *How* should the system be constructed?

This paper has attempted to show that the infological approach provides an embryonic tool to help with the *what* questions. Although it also provides a basis for answering the *how* questions. When applied to data bases many of the *how* questions revolve around data-logical analysis and design tasks. Ways in which this might be done are elucidated in Chapter 6 of [4] and in Chapters 7 and 8 of [5].

One can easily create a subsystem structure of the processes involved in a data base system. The list of subsystems might include:

—quality
—scheduling
—protection
—interaction
—accounting
—catalogues
—maintenance
—files.

Clearly many of the data-logical subsystems will have infological equivalents. For example, the quality and protection subsystems correspond to the infological filter function.

It should perhaps also be noted that the same entity in a particular object system can be referred to by different infological entities, each of these can be represented by different data-logical entities. Each data-logical entity can obviously be stored differently. Thus, although the infological requirements may have been established first of all, there is still considerable design work left to be done, this is properly the function of a computer-technical design team.

A particularly important data-logical task will be to design the file system for the data base. This task will involve the following ordered set of sub-tasks:

1. Formally define what information the base should contain — this will be at least partially done during the infological analysis.
2. Formally define the transaction types that the data base will encounter when operational. It will also be necessary to specify desired response times and expected frequencies for each transaction type.
3. Define the structure, capacity, and speed of the available memories.
4. Define files for 1 and map these into the available memories defined in 3.

Methods of defining the final file structure in terms of the initial object files, property files and relational files by using various structuring operators is worth another paper in itself and is not dealt with here.

The Usefulness of the Infological Approach

The idea of a data base is a very general concept. The infological approach to the analysis and design of a data base provides a beginning of a prescriptive theory for their design. Until such prescriptive methods are developed to involve many different users in their design, it may only be useful to implement

quite limited and relatively less complex data bases using computers.

BIBLIOGRAPHY

1. Codd, E.F. 'A Relational Model for Large Shared Data Banks' Comm. ACM June 1970, Vol.13, No.6, p.377.
2. Firth, J.W. 'System Influences on the Design of Large Scale Data Bases' 6th Aust. Computer Conference, Sydney, 1974.
3. Längefors, Börje 'Theoretical Analysis of Information Systems' 4th Edition, Averbach Publishers, Philadelphia, 1973.
4. Längefors, Börje and Sundgren, Bo 'Information Systems Architecture' Petrocelli/Charter, New York, 1975.
5. Sundgren, B. 'Conceptual Foundation to the Infological Approach to Data Bases' Invited paper to IFIP-TC2 Working Session, Corsica, 1974. (Published in Klimbie and Koffeman (Eds.), *Data Base Management*, North-Holland, 1974).
6. Sundgren, Bo 'Theory of Data Base' Petrocelli/Charter, New York, 1975.

6 A Semantic Approach to Data Base Design

Dr. I.T. Hawryszkiewycz

INTRODUCTION

An important component of any system is the information that supports systems operations. In a computer-based implementation this information is mapped onto some available computer facilities. In many cases such mapping, for performance reasons, places more emphasis on short term advantages provided by the software rather than the long term developments expected of the system. This then can result in systems that are over-reliant on current versions of software, which are available on some machines and which are not responsive to changing user requirements.

The claim of this paper is that, to design systems that are not over-dependent on software facilities, priority must be placed on the analysis of the information structure of the problem rather than the structures provided by the software. We distinguish these two structures as the information structure and the data structure and discuss an approach to system design based on information structures. Operations on the information structure then are defined by user semantics whereby those on the data structure by data semantics.

THE PROBLEM AREA

A problem facing system designers is the mapping from some information structure to the data structures that are available at some host machine. The problems encountered include those that are semantic in nature together with those associated with realizing satisfactory system performance.

Semantic problems arise as there is generally only a limited subset of data structures provided by most software systems. The user is then forced to map the information structure of the problem into a data structure within the constraints of the software system. Similar problems arise with mapping user operations, which are performed on the information structures, into commands provided by available software systems. These commands transform the data structures that represent the information structures. In this case one natural operation is generally mapped into a sequence of software commands that exploit the chosen data structure. Performance problems also arise as such mappings must ensure that:-

1. access paths are made available to allow anticipated queries to be satisfied,
2. there is little duplication of data, and
3. proper security and privacy constraints are established.

In most cases such design is a trial and error process. Designers use previous experience with similar information structures or data structures to decide on mappings that satisfy most design requirements.

One consequence of this can be that to meet performance criteria, unnecessary complexity is introduced in the mapping to the data structure as

the designer wishes to exploit those forms of data structure that yield the best performance. Thus more emphasis is placed on the data structure rather than on the information structure. This may result in a mapping between the information structure and data structure that is more complex than necessary. This, together with the fact that the mappings are one-to-many in nature, means that any changes to user requirements will be magnified many times in the underlying data structure semantics. Because of this the user's initial maping tends to become fixed and inflexible as changes to information needs are generally greatly magnified into corresponding changes to the data structure because of complex mappings. Consequently the system tends to become inflexible due to the cost involved in incorporating such changes. Nolan (1973) claims that such inflexibility inhibits the development of management information systems.

THE DATA BASE MANAGEMENT SYSTEMS SOLUTION

Data Base Management Systems (DBMS) provide a solution here by separating the user semantics, the data model and the physical structure with mappings between these becoming a function of data management rather than a user design problem. This is illustrated in Figure 1.

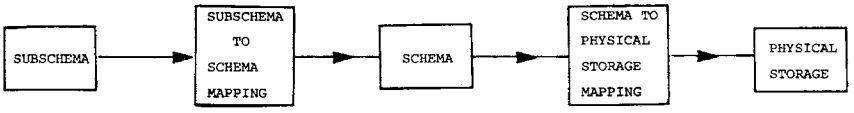


Fig. 1 Mappings in a DBMS

Here the information structure is expressed in terms of a schema. The manner in which the information is used is expressed by the subschema. There is a mapping from the subschema to the schema, which in turn is mapped onto a physical structure. Under these conditions facilities are provided where:–

1. performance problems are resolved at the schema to physical mapping and hence make user semantics of physical structure,
2. new ways of using the information can be expressed by the addition of new user subschema without the necessity of making changes to the schema or physical structure, and
3. the schema can be changed without effecting any existing user views of the information.

Although the above facilities are possible in principle, in practice, as discussed by the author (1976) only a limited subset of the facilities is available in existing DBMS. Bracchi and Paolini (1975) have proposed that this limitation is due to the fact that in most cases the record is chosen as the basic unit of the schema. Their proposal is that the binary relation should be used as the basic unit.

A user seeking flexibility must then choose the method of specifying the information structure in terms of the data structure with some care. The specification must be such that it can utilize those facilities of a given data base management system that allow some flexibility in the mappings to provide the flexibility desired by the user in the system.

The basic approaches here are to:–

1. develop some formalized procedure that maps the user information model into the data structure in a way that exploits changes permissable in the schema to realize changing user requirements, or

2. develop generalized definitions of the semantics of user operations in terms of sequences of program instructions that transform the data structure. Changes in user requirements can then be expressed in terms of the generalized semantics and new user operations can be accommodated by selecting existing sequences of program instructions that express these semantics.

For convenience in this paper we call these two approaches the data model approach and the semantic approach to data base design. The data model approach is perhaps self-explanatory as it is based on choosing data structures in the manner described earlier with additional consideration of possible future changes. One such formalized approach with the DBTG data model is described by Bubenko, et al (1976). The semantic approach requires some more explanation.

THE SEMANTIC APPROACH

This approach is illustrated in Figure 2.

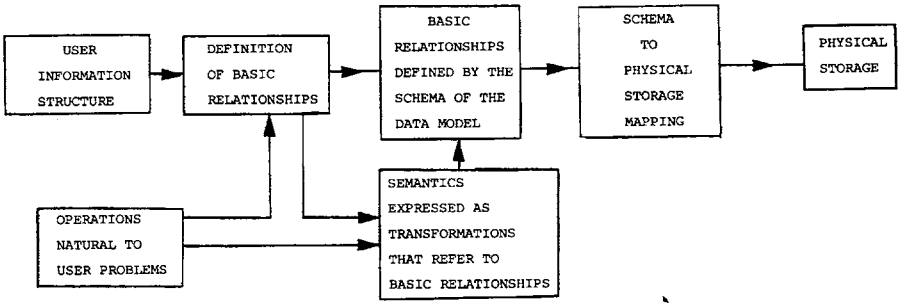


Fig. 2 The Semantic Approach

Here we have a data model that may take any of the forms of general class of models available with DBMS. These have been described by Date (1975) as being either relational, hierarchial or the network model. The mapping between the information structure and the data model is primarily determined by consideration of the user semantics. This is done by analyzing user operations on the information structure to determine any basic relationships that are inherent in the structure. The basic relationships are those that cannot be changed by changes to user operations. Various attributes are associated with these relationships. Thus, for example, the attributes "amount" and "price" can be associated with the relationship CONTRACT-ITEM. User operation can affect the values of the attributes. The basic relationships and attributes are mapped onto the data model and defined by the schema. User operations result in transformations that change the values of attributes or add new relationships or relationship occurrences. Changes to user requirements can result in changes to the ways in which attributes are affected. They may also result in the addition of new basic relationships or attributes but will not require changes to existing basic relationships. If the basic relationships are defined in the schema and transformations expressed in terms of the basic relationships, then the only changes in the schema will be the addition of new relationships or attributes but the existing relationships and attributes in the schema will not need to be changed.

In some cases where there is little similarity between the data model and the user information structure it may be advantageous to introduce an intermediate

stage of a conceptual data model as illustrated in Figure 3. This facilitates the determination of the basic relationships. The user semantics are expressed in terms of instruction sequences that express user operations in terms of transformations of the conceptual model. Formalized techniques are then provided for mapping from the conceptual model to the data model that is available on an available DBMS. Similarly formalized techniques are provided to translate from the semantic instruction sequences that express user operations in terms of transformations of the conceptual model to corresponding instruction sequences that express user operations in terms of transformations of the data model.

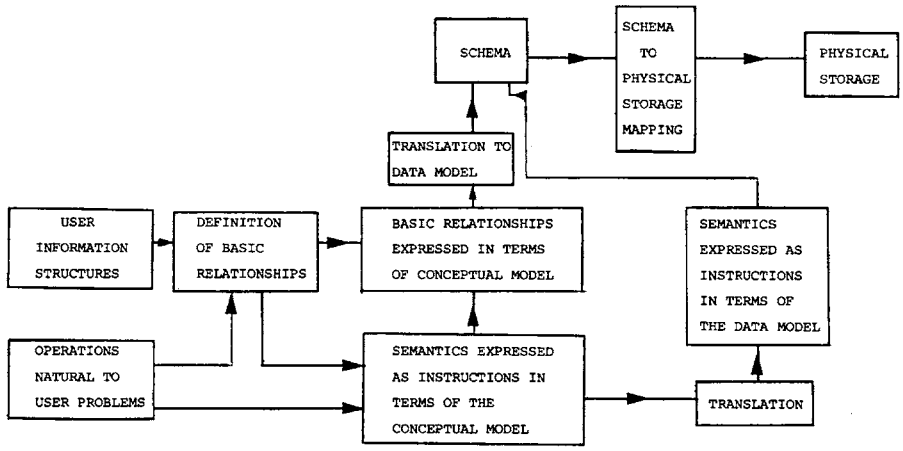


Fig. 3 The Semantic Approach Using a Conceptual Model.

The advantages here are two-fold. Firstly we now completely remove any influence that the existing DBMS may have on user semantics as these are now expressed in terms of the conceptual model. Secondly, by formalizing the interface we are now best able to take advantage of those features of the data model without consideration of user semantics. The design has thus been clearly divided into two independent parts.

The performance control still resides outside the domain of the data base user as it is implemented at the data model to physical interface.

In the remainder of the paper we illustrate this approach by considering a class of user systems, which we call service systems.

THE SERVICE INFORMATION MODEL

There are many situations in industry characterized by the interchange of services, goods or moneys between organizational entities. Furthermore, it can be shown that many systems that appear inherently different have many characteristics in common. For example, there is a similarity between a payroll system and a sales accounts system.

In the former an employee performs a service to the organization for which there is some eventual compensation. Similarly in a sales account the organization performs a service by supplying some goods to a customer. There is again some eventual compensation here.

The system that is described here is based on a service model, which is then used as the basis in describing user requirements.

The service model is basically characterized by two kinds of entities, namely, the *server* and the *receiver* and *transactions* between the two. There may be a

number of attributes associated with each server and receiver and with the relationships.

The values of the attributes can be changed following the occurrence of a *transaction* in the system. We can have:–

1. a *service* transaction, which is a transfer of some object from the server to the receiver, and
2. a *payment* transaction, which is a transfer of some object from the receiver to the server.

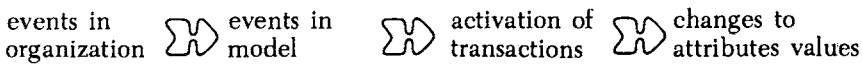
Such a transfer is effected by reducing the value of some attribute associated with one entity and increasing the value of an attribute associated with the other entity by the same amount.

A transaction is consequent upon some *event* occurring. Generally there is an event that causes some service transactions and this is subsequently followed by an event that causes a payment transaction.

There are a number of transaction types allowed and each event occurrence causes the execution of a subset of these transactions. The event defines the transaction type to be executed.

Thus one possible event is a sale. This transfers some amount of goods from the seller to the buyer while at the same time transferring some moneys from the buyer to the seller.

The general model philosophy is then as shown below



Here an event in an organization is defined formally as some event in the information model. The occurrence of that event in an organization results in an initiation of the corresponding event in the model. This in turn initiates a number of transactions which result in changes to attribute values. Each event is thus defined in terms of a number of transactions. Each transaction defines the change to some selected attributes.

Part of the activation of each event requires the specification of the server or receiver involved in the event and in some cases parameters that are used in computing the change to attribute values.

One step in allowing flexibility is by the manner in which transactions, events and attributes are defined. Such definitions should allow

(i) flexible combinations of events, transactions and attributes, and
(ii) various ways of computing the changes in attribute values.

Thus we require the ability to define new events and transactions, and to associate existing attributes with the transactions. Appropriate parameter structures must also be provided to allow the transaction to determine the changes to attribute values.

DESIGN USING THE DATA MODEL APPROACH

With existing data models based on a record as a basic unit we would in general expect each attribute to be mapped onto a record or a repeating group in the server or receiver record. An event 'e' would then result in some input transaction that activates some program. This program then updates those records that contain attributes, which are effected by the event, 'e'. Basically we then have the situation illustrated in Figure 4. Addition of a new event then generally re-

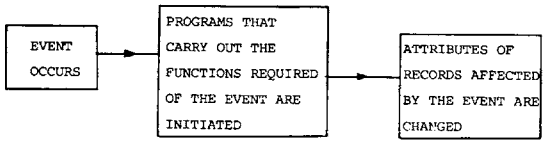


Fig. 4 The Data Model Design.

quires a new program. Similarly if a new attribute is created then the program is modified to update that new attribute. One possibility provided by existing DBMS is to allow new fields to be added. In this case we can, by using this facility, improve flexibility while at the same time being contradictory to user semantics. For example, suppose we have a service system where the servers are employees of some organization. Events, which are daily attendance records, result in changes to attributes such as net pay, or gross pay associated with each employee. These attributes are included in the employee record. Suppose we now wish to record employee attendance against a project that the employee has worked on. Given a DBMS that allows new fields to be added to a record we may implement the change by

1. adding a new field to employee records that corresponds to projects,
2. changing the program so that the new field is updated as necessary, and
3. providing summaries of project costs by scanning all employee records.

This in the first instance has the undesirable effect of requiring us to scan employee records when we have some query associated with project costs. Suppose, however, that any problems that arise are solved through privacy and security facilities available with the DBMS. However, suppose now we wish to further expand our cost control procedure by including material costs associated with projects and providing summaries including both wage and material costs.

If we already had an inventory system available then this could now be accommodated by adding fields to inventory records. Our project costs would now be distributed across two files, neither of which are basically concerned with project management. We would then either redesign the system creating a new set of records associated with projects and changing those programs that correspond to events that include withdrawal of items from store and the payment of employees in the organization. The problem of redesign has thus been simply postponed.

DESIGN USING THE SEMANTIC APPROACH

What we are looking for then is an approach that allows:–

1. definition of new kinds of server or receiver in the above case, a project
2. existing events to effect attributes associated with these new servers
3. new relationships between existing entities or between new and existing entities to be established, and
4. new events and transactions to be defined.

The conceptual model that can be used to describe this is an extension of

Codd's relational model (1970). Some modifications to the relational model have been discussed by Chen (1976). These modifications have been made to allow entities in the commercial environment to be defined in terms natural to this environment and the operations to be related to those entities in a similar way. Florentin (1976) has also used this approach to represent various application models.

ENTITIES AND RELATIONSHIPS

The conceptual model is based on *entities* which are elements that can be distinctly identified. Such entities are then generally grouped into several kinds. For example, one kind of entity may be a person whereas another may be a project in an organization. Each entity is identified by some element of a value set. The value set that identifies persons, for example, may be a set of personnel numbers and each such personnel number uniquely identifies some person. A number of attributes may be associated with each entity. These describe some properties of the entity. Thus attributes associated with a person entity may be age, colour of eyes, qualifications and so on. Each attribute has a name and takes some value from some value set. The attributes associated with one kind of entity will in general be different from those associated with another kind of entity. Thus the attributes associated with persons may be different from those associated with projects. The attribute that can be used to uniquely identify an entity is called the *entity identifier.*

An example of a description of an *entity set* is given below. The set illustrated is a set of persons. The attribute 'personnel id' uniquely identifies each entity in the set. 'Name' and 'age' are properties of the entity.

ENTITY	ATTRIBUTES		
Persons	Personnel id	Name	Age
p_1	678	J.K. Black	35
p_2	679	P. Hogan	43
.			
.			
.			
p_n	680	J.K. Brown	35

Entities in the model can be associated with each other in a number of relationships. Thus there can be a relationship in the system between the entity set persons and the entity set projects. This relationship indicates the people, who performed some task for the project.

Each entity in a relationship plays some role in the relationship. In the PERSON-PROJECT relationship the entity person will play the role of a 'worker'. Other roles that a person may take are 'manager' in a MANAGER-PROJECT relationship or 'supporter' in a SUPPORTER-DEPENDANT relationship.

An illustration of a relationship is given in the following figure.

Project-Person Relationship			
Entities		Attributes	
Person	Project	Hrs. Worked	Class of Work
Role Worker	Project		
678	A3	23	A
679	A5	22	A
.			
.			
.			
680	A7	23	C

Here the entities are 'person' and 'project' and the attributes are hours worked by the person on the project and the class of work carried out. Each entry in the relationship table is a relationship *tuple*.

The entities that uniquely identify each relationship tuple are called the identifiers of the relationship. In the above case each person and project uniquely identifies each tuple (provided that each person only does one class of work on the project).

Operations in the system are concerned with creating or deleting relationships, adding or deleting relationship tuples or changing the values of attributes associated with relationship tuples.

Given the conceptual model let us now return to the semantic approach as illustrated in Figure 3. What we need to define now are

1. mappings between the user semantics as described in the service model and the conceptual model, and
2. a mapping between the conceptual model and the actual data model.

SEMANTICS TO CONCEPTUAL MODEL

The mapping between the service information model and the conceptual model takes the following form

Service Information Model Elements	*Conceptual Model*
server	entity
receiver	entity
attribute	attribute
relationship between server and receiver	relationship

The semantics of the service system then require that events and transactions be defined and stored within the system memory. Generalized routines are then provided to:–

1. create events and transactions,
2. create new relationships, entities and their associated attributes,
3. add new relationships n-tuples, and
4. perform a transformation following the occurrence of an event.

The last of the above would be a generalized routine, which selects the transactions defined as part of the event, computes a value given transaction parameters, selects the attribute whose value is to be changed and effects the required change. A detailed example of such routines has been discussed by the author previously (1976).

Suppose now we wish to effect the kind of change discussed earlier, that is, recording employee attendance against a project. This can be done by activating generalized routines to:–

1. create a project entity and attributes associated with each project,
2. create a new transaction that effects attributes associated with the project, and
3. associate this new transaction with the event that results following employee attendance.

CONCEPTUAL MODEL TO DATA MODEL MAPPING

In this mapping we express the conceptual model in terms of a data model in some formal way. Thus, relationship types for example, may map to some record type in a data model. Creation of a new type of relationship create a new instance of this record. This record can then become the owner of a new set instance that contains relationship instances. We thus have the mapping shown in Figure 5. Semantic routines are now translations of the semantic routines that

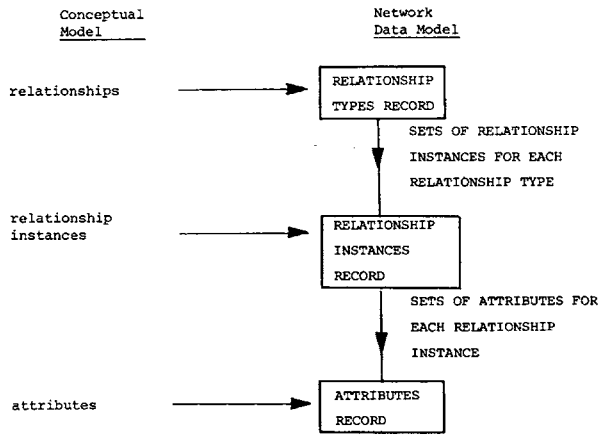


Fig. 5 Conceptual Model to Data Model Mapping.

transform the conceptual model into sequences of instructions that transform the data model. Each new operation results in the execution of one such sequence. The semantic routines thus, in effect, are an interpretive implementation of user operations.

It should be noted that the semantic routines can be thought of as program sequences in some interpreter. The kind of design discussed can be thought of as constructing a generalized interpretive system to perform operations that exist in service information systems.

CONCLUSION

This paper has placed emphasis on defining the semantics of user processing ahead of the consideration of data structures in a software data model. The

resultant method of implementation has then required an interpretive approach, converting user operations to corresponding operations on the data model. It may then be argued that such an implementation is unrealizable due to the high overheads involved in the interpretive process. This if of course true if there is a significant difference between the data model and the conceptual model. Should this not be so, as for example, where the data model is a relational structure, then the interpretive mapping may be a 1:1 mapping with little resultant overheads. Secondly, with the changing computer architecture it may be possible to include the semantic interpretive routines as firmware in some back end processor making this kind of design approach attractive.

REFERENCES

BUBENKO, J.A., BERILD, S, LINDENCRONA-OHLIN, E., NACHMENS, S. (1976) 'From Information Structures to DBTG Data Structures' ACM SIGPLAN Notices, Vol.8, No.2, 1976. Special Issue on Proceedings of Conference on Data: Abstraction, Definition and Structure.

BRACCHI, G., PAOLINI, P. 'Multilevel Relational Data Representations' Management Datamatics, Vol.4, No.5, 1975.

CHEN, P.P. (1976) 'The Entity-Relationship Model: Towards a Unified View of Data' ACM Transactions on Data Base System, March, 1976.

CODD, E.F. (1970) 'A Relational Model on Data for Large Data Banks' Communications of the ACM, June, 1970.

DATE, C.J. (1975) 'An Introduction to Database Systems' Addison-Wesley, 1975.

FLORENTIN, J.J. (1976) 'Data Base Representation of Application Models' The Computer Journal, Vol.19, No.1, 1976.

HAWRYSZKIEWYCZ, I.T. (1976) 'The Role of Data Bases in Realizing System Flexibility' Proceedings 7th Australian Computer Conference.

NOLAN, R.L. (1973) 'Computer Data Bases; the Future is Now' Harvard Business Review, September-October, 1973.

7 Distributed Databases: DBMS Design Issues

Garth Wolfendale

SUMMARY

There are many dimensions to DBMS design, especially in the management of distributed databases. The problems of structure, selection, control and organization of data are many and hardly touched on as yet in the literature. This paper attempts to cover a few issues in DBMS design for the support of distributed databases. An example of a system currently being implemented is given.

A DECLARATION OF INTEREST

At ANU a network involving non-homogeneous systems and supporting a wide range of applications, is under development. One of the prime goals is to support a kind of anarchy in which users are free to create files anywhere on the net (within implementation limits) and free to access them. As part of the research effort we are studying the unification of the network data system providing a coherent data system and command language independent of the host system differences. Any user will be able to describe data, have available selection mechanisms and controls which will be dependent only on the user (or process) and not *where* he or it is operating. Some host systems will wish to maintain solely a 'local' system whereas others will want to have currently active data local, with other less active data held on another system supporting more on-line and off-line storage. This data can be considered 'distributed' to some extent but 'local' in the sense that only a single user (or group of users) has access or interest in it. Some applications will involve distributed data and shared access, along with possibly concurrent different logical or application-level data views.

Thus, on the one network we have a range of distributed database requirements, and in order to break these down in some ordered fashion, a taxonomy is required for the classification of distributed databases. Anderson and Jensen (1975) presented a taxonomy of interconnected computer systems, and attempt to provide a taxonomy for database systems is presented in this chapter, with discussion of their mapping on to computer systems. In the literature on distributed databases there has been little attempt to organize database system problem areas. However the number of combinations of data and computer systems is great, and there is a need to develop such an organization to be able to identify the database management system requirements for the support of such systems.

A TAXONOMY FOR DATABASE SYSTEMS

As was stated in chapter 1 of these proceedings, apart from the 'external world' of a database system (outside the data environment managed by a DBMS), there are four major dimensions of a DBMS involved in the management of a database. These are:–

1. Data structure – data entities and their logical relationships.
2. Data Control – protection, integrity and recovery mechanisms.
3. Data Selection – storage and search mechanisms.
4. Date Organization – mappings between entities themselves and between entities and elements of the host computer system.

Data systems like computer systems onto which they map, can be centralized or distributed. Similarly, each dimension can be placed on the centralized-distributed dimension. A centralized logical view is one in which there is one system wide view, with very restricted subsetting allowed, whereas a distributed logical view is one in which the same physical data can be viewed in many ways, depending on differing applications and their data models. These views need not be strict subsets of global or system-wide views. In a similar way there can be centralized or distributed controls. A distributed control system may only rely on 'local' integrity or protection mechanisms, or a hierarchy of controls, as opposed to single central control. It is possible, and common, to map distributed logical data views and data controls on a centralized host computer system, and centralized logical views and controls on a distributed computer system. Selection mechanisms can also be centralized or distributed. File organizations can be centralized on one machine and funnelled through one process organization, or can be distributed with the logical view and associated searches being managed on distributed file systems and concurrent processes running on possibly different computer systems.

Data organization, in these terms, therefore becomes the problem of mapping the other dimensions of DBMS onto computer systems, which themselves can be centralized or distributed according to the taxonomy.

In the design and systems analysis of a DBMS, it is therefore necessary to define all aspects of the system, identifying where they fit in the taxonomy for the data system as well as that for the computer system on which DBMS is being implemented.

The following sections briefly discuss the problems involved in designing and implementing a DBMS supporting distributed data views, controls, selection mechanisms and organizations.

PROBLEMS OF DISTRIBUTED ORGANIZATIONS

Why not centralize the whole database, access and control mechanisms, and selection algorithms? Surely this would be simpler than distributing data and its management, but what would be the *efficiency* of the final system? A DBMS is a system designed for the optimal allocation, update, maintenance and control of data resources, thus in the area of distributed data systems there has been an increasing amount of work concerned with optimal data resource (e.g. files, file-sections, records) allocation.

Casey (1972), Chang (1975), Chu (1969), Mahmoud and Riordon (1976), Eswaran (1974), Levin and Morgan (1975) and Whitney (1970) are key papers in this area. Casey developed a file allocation model which distinguished between update and query traffic transactions in a distributed system. The model was used to find the minimum cost allocation of file copies to modes in a communications network which was full connected (DDC in the taxonomy of fig.1). Chu investigated optimal allocation of files in a DDC network with a fixed number of copies of each file, whereas Whitney investigated the problem of assigning file copies to nodes of a linear graph while minimizing traffic induced along the edges of the graph.

Mahmoud and Riordon, take more parameters into consideration. They treat

the problems of file allocation and of internodal link capacity allocation simultaneously in order to achieve a minimum cost design, subject to constraints of file availability and network delay.

They identify the following parameters:–

1. Capacity allocations and associated costs for communications,
2. File storage allocation costs and placement,
3. Constraints such as link reliability, delays in the network due to message lengths, channel capacities, traffic flow, access times and file availability.

The explicit solutions to the derived equations are in the class of non linear programming problems, and Mahmoud and Riordan give solutions for small problems. For large problems, however, they present a heuristic technique which consists of two routines: the starting routine and the optimization routine. The starting routine allocates files and capacities that satisfy delay and availability constraints, and generates a number of feasible initial solutions. Starting with one of the initial solutions, the optimization routine attempts to improve the total cost by successive additions and deletions of file copies. When a feasible solution has been found, the routine then iterates.

Eventually a feasible solution is reached, the cost of which cannot be significantly reduced by further reallocation of resources. This is now treated as a *local* optimum solution. The final solution arrives at a distribution of files or host nodes, and allocation of channels and capacities given cost factors and parameters such as:–

1. The number of network nodes
2. The number of unidirectional links
3. The number of link capacity alternatives per link
4. The number of files on the system
5. A capacity/cost matrix
6. Query traffic specifications
7. Update traffic specifications
8. Storage costs
9. File lengths
10. Link reliability specifications
11. Node reliability specifications
12. Message lengths
13. Maximum acceptable average delay per message
14. Minimum acceptable file availabilities (probabilities of being available when requested).

The problem being to choose file allocation and capacity allocation matrices to achieve the minimum of a measure of system cost.

DISTRIBUTION OF FUNCTION

The storage, selection and access of data can be centralized in a DBMS, or can be distributed. One way of achieving this, is to define a utility in a network which is responsible for access and search requests, which can run concurrently with the rest of DBMS activities. Since access and search activities account for a high proportion of the response delays in a database system, then again performance and efficiency are good reasons for exploring such a scheme.

Marill and Stern (1975) presented a paper concerned with a specialized data utility for networks which they called the 'datacomputer'. It is designed to provide facilities for data sharing among dissimilar machines, rapid access to large on-line files, storage economy through shared use of a trillion-bit store, and improved access control. Within a resource-sharing network there is a

natural tendency towards specialization of network nodes. The medium scale, time sharing system tends to become the interactive node, others become nodes for heavy scientific computation and so on. In the limit, specialized network nodes become what can be called 'utilities' and the datacomputer is a utility in this sense. A database managed by the data computer is shareable by all computers having access to the system. Thus a database on the utility is shared not only among users of different interest, but among users of different host computer systems.

At a low level, character codes, floating point representations and word sizes vary from user to user. At a higher level, data structures vary, and these problems will be discussed later. The datacomputer is required to perform translations, between various hardware representations and data structuring concepts. To access data on the data utility a network-wide data language is used as a protocol. In most approaches to DBMS (e.g. CODASYL, protection discussions), the assumption is made that the DBMS is closely coupled to the application program. Chapter 1 of these proceeding proposed a DBMS approach that countered this assumption. In the distributed environment, this need is even more urgent. Close coupling is certainly not the case in a network environment where bandwidth between the application program and DBMS is low. Thus a data language must be defined to allow self-contained requests to be shipped to the datacomputer to be executed there.

Consider an application to increase all salaries in a Department by 5%. In a conventional or centralized system the application will access the whole file record by record to update it. In a network, the node doing the processing would require the whole file to be shipped over to it from the central database node. The node doing the processing could possibly not have enough local storage capacity to hold it, and anyway it would involve a lengthy (possibly unreliable) transfer from and to the node owning the file. Also, if the file organization is such that auxillary files (indexes for example) are involved, these would also have to be shipped backwards and forwards between nodes. Therefore the data language is designed so that requests can be sent to the datacomputer from the application program, and the datacomputer itself performs the indicated functions, signalling the application program upon completion. Of course, the user program can request a file transfer if necessary, so that the datacomputer can be used as a file manager in the style of the TABLON system (Gentile and Lucas, 1971), which has been in use since 1973, with terabit memory being added in 1975. Canaday, Harrison et al (1974) also present a dedicated utility, referred to as a back-end computer for database management.

DISTRIBUTION OF DATA CONTROL

Chapter 1 of this book discusses protection, integrity and locking procedures in some detail. In this section, the distributed control of multiple copies of data will be discussed. The presentation will be based on a paper by Mullery (1975). The control of data in a network consists of routing of requests for data to the proper location and protecting that data from unauthorized or consistency-threatening requests. Control can be centralized or distributed (e.g. Farber and Heinrich, 1972), where data is distributed with each datum uniquely located at one location which is also the site of the lock(s) for the datum. As a whole, all locks and locking mechanisms are distributed, but all requests for a given datum are channelled through one lock. Even data that exist as distributed multiple copies can be handled through a central controller (Booth, 1972), however the distributed control of multiple copies presents serious problems. One ques-

tion to ask is whether it is ever necessary to have multiple copies of data. A file or set of data entities that is mainly read, but by multiple users or different nodes is a network, is most efficiently stored and accessed at the systems that are making most use of it. To centrally maintain and control such a set would lead to tremendous overheads. A catalogue of the location of data sets (a data directory as opposed to a data dictionary) is a good example of a data set that could be most efficiently distributed as multiple copies since a catalogue will have a high proportion of reads to writes. If it is assumed that the directory at a given node is accurate for the data on the node then a special type of distributed control for all copies, called 'loose control' can be applied. This level of control does not guarantee that all entries in all copies are always accurate, but only accurate for some acceptable percentage of the time. Updates are performed periodically at a rate sufficient to keep inaccurate entries below a given criterion. In loose control, data set accesses must be checked locally, but in the case of the data directory this can be done without network cost since access is normally made to the location of the data anyway. Unless data has properties to enable loose control (inaccuracies determinable and checking done at low cost) then strict control is necessary. Mullery (1975) presents a strict control mechanism which will now be described. It has the feature that the control can be centralized in the degenerate case as a special case. There are two requirements for the method to work:-

1. Each control must resolve ties in a uniform way and
2. The controls must exist at locations at which the locks exists, although not necessarily vice versa.

The method is based on the use of a lock with three possible states: available, prepared and locked. The prepared state is an intermediate state into which data is put pending the resolution of any conflicts by the protocol. When these have been resolved, the data is then locked, and updating can be performed. When in a prepared state, who or what the data is prepared for is also stored. An efficient implementation of the lock is simply an integer with the following significance:-

Lock = 0 means available

Lock = m + 1 means locked, where m is the number of nodes in the net that can take part in the lock control mechanism

Lock = $1 \leqslant j \leqslant m$ means the prepared state and who the lock is prepared for. Where order 1 to m is treated as a priority ordering.

It is assumed that the controllers are reliable in the first instance, but the discussion is extended to unreliable controllers later.

A machine, process or lock is defined as being *local* to a particular controller if that machine or process is under the control of that controller. The following defines the actions performed in each of the three states by a controller.

Available State (Lock = 0)

1. Accept a request from another controller to update some data entity, change the lock or that data entity to the corresponding prepare state, and transmit to other nodes having a copy of the data to be updated a request to change to the prepare state (locally).
2. Accept a request from another controller to enter the prepare state and acknowledge the execution of the request.
3. Ignore other messages.

Prepare State (Lock $1 \leqslant j \leqslant m$)

1. If the current prepare state is for a machine of lower priority, accept a local

request to update, assign the prepare state to the local machine, and transmit to the other controllers having a copy of the data to be updated a request to change the lock to the prepare state.

2. Accept a request from another controller to enter prepare state. If the current prepare state is for a machine of lower priority than the requesting one, assign the prepare state to the higher priority request, and acknowledge. Otherwise, send a negative acknowledge to the requesting controller.
3. Accept acknowledgement of own request to others to enter prepare state. If acknowledgement is received from all requested controllers, and the local lock is still prepared for local machine, change the local lock to update state (lock = m + 1) and transmit a request to change to the update state to other controllers.
4. Accept a request from another controller to enter update state, change to update state, and acknowledge to the requesting machine.
5. Accept from another controller a negative acknowledge of a request to prepare, signal the local requesting process to wait and try again. If the local lock is prepared for the local machine, change to *available* state.
6. Ignore other messages.

Update State (Lock = m + 1)

1. Accept an acknowledgement of own request to others to enter update state. If acknowledgement is received from all controllers involved in the update, then transmit update requests to these controllers.
2. Accept an actual update request, execute it and acknowledge.
3. Accept acknowledgements of execution of update from all controllers to which it was transmitted. If all are received, change to available state, and transmit a request to change to available state to all these controllers.
4. Accept a request to change to available state and make the change.
5. Accept a negative acknowledge of a request to prepare from another machine. If the local process has not already been signalled to wait and try again, do so.
6. Ignore other messages.

Fig.3 gives a sample sequence for an update, where 7 = m + 1 is the lock value. A transmitted message is indicated by t (m, message) where m indicates the controller transmitted to, and message may be one of:–

p —request to prepare
ap—acknowledge prepare
np—negative acknowledge prepare
u —request to update
au—acknowledge change to update state
a —request to change to available state.

A message received is indicated by r (m, message), where m indicates the controller received from and 'message' is the message received.

In the case of unreliable controllers, the system relies on the consistency and recovery mechanisms supported on the controllers; and these are discussed in more detail in chapter 1 of this book.

DISTRIBUTION OF DATA STRUCTURE

The most centralized system would have one data structure with strict subsetting (e.g. tree structures). As one frees these restrictions, with mapping onto a distributed network, then we are faced with problems of communication between systems which describe the same data in different ways or support different data structures. There are a range of papers in this area:– Housel, Lum

and Shu (1974), Liu and Heller (1974), Shu, Lum and Housel (1976), Su and Lam (1974) and Yamaguchi and Merten (1974). These papers are concerned with the problem of translating data from one logical structure to another using various techniques. For example Shu, Lum and Housel present an approach which views data migration from one data system to another as a three-step process:–

1. A conversion interface using source system facilities transforms the source data into a normal form in files.
2. These files are processed by a translator/converter where the restructuring takes place. The result is a set of target files.
3. A conversion interface using the *target* system facilities loads the target files into the target system resulting in the final target form.

To achieve data conversion in this way, two languages have been developed:– a data description language, DEFINE and a conversion specification language, CONVERT. Both languages are high-level and non-procedural. Earlier systems (e.g. Anderson et al, 1971; and the DSCL system of Schneider, 1975), support low level reformatting of data entities not the restructuring of entities at the logical level.

In a system in which a unified network-wide logical (structural) data model is supported based on non-homogeneous host data systems, then the implementation of the DBMS will rely strongly on techniques such as those described, requiring the restructuring and reformatting as data migrate. For example, crime reports gathered and maintained in various cities inherently constitute a database of more than local interest. It is natural to think of these related data files at different sites as elements of a single unified data collection. However, more often than not, the local data files are managed by different host data management systems and cluttered with non-essential data of local interest or from a different applications point of view. To treat them as elements of a single unified data collection would require at least the capabilities to decompose the local files, extract the relevant information, map extracted data into a structure suitable for the application in hand, and combine the data from multiple sources into one collection. Thus data conversion and restructuring are essential components in a unified but distributed database system.

DISTRIBUTION OF SELECTION MECHANISMS

Selection mechanisms and algorithms in the distributed environment have received scant attention, and I can find no references that directly address the subject, apart from the specialization of function described earlier. Consider a database consisting of local data/host systems for general practitioners with intercommunications between them. Also consider that there are higher level systems (Hospital and Regional medical database) involving centralized host systems with access to the data on the local host systems. These latter will have sets of patient records with orderings in terms of local keys (e.g. NEXT in terms of checkup time, disease, family etc.). The higher level system will support different orderings, keys, data domains etc., based on different requirements. Now, if search, selection and update are to be carried out by the central system via the local systems, then there is the problem of knowing the search mechanisms of the local systems in order to access the data. One local system may support only sequential files of patient records whereas another may have a multi-key inverted file system, again for patient records. The central system must be able to drive the local selection mechanisms appropriately in order to collect patient records.

This is an interesting problem involving the mapping of selection *procedures* as well as the data *structure* mappings discussed earlier.

AN EXAMPLE OF A DISTRIBUTED DATABASE

Change (1976) describes the software design problems of a distributed medical database to be implemented on a homogeneous computer loop network of PDP11/45 machines using the UNIX operating system (Ritchie and Thompson, 1974). The system is a medical information system in which the producers of information are distributed geographically, e.g. physicians, laboratories, pharmacies, emergency rooms. The types of data produced by the nodes vary widely. Even among physicians in a common category the details and formats of data recorded vary from physician to physician. The data in such an environment are difficult to manage in a centralized computer system and the normal solution of using a common simplified data model is a severe restraint on users. Each user needs a capability for defining data, suited to his problem area.

Fig.4 illustrates the database organization under consideration.

Also, such systems, partly because of their distributed nature, tend to exhibit strong locality of geographic reference, in that *most* of the data generated at a source is referenced locally – as in a typical physicians office.

Therefore, where the requirement for user-definition of user-owned information is high, and data sharing is very important, data files should be owned locally by the data creator, but be accessible to others through a communications network.

Furthermore, such data files should possess a characteristic that all users be able to view them in ways most suitable to their needs. Thus, Chang has used a data (format) translation scheme, in which all files are self describing, such descriptions giving physical and logical definitions to enable inter-node translation to take place.

At each node, there is implemented a virtual database machine (DBM). A user of the DBM is said to be in database mode, otherwise local model is in effect.

The DBM is itself built from other virtual machine (described later). These are implemented as concurrent processes, through the timesharing capabilities of UNIX. Each DBM has access to a single set of database files, known as the database file collection of the node. Users in database mode can only *retrieve* information from the network. Updates, insertions and deletions must take place locally and are restricted to the owner of the database file.

Also a restriction on the system is that network retrieve requests are treated as batched transactions.

The DBM virtual machine has three parts:–

1. A variable set of user machines (process) UM.
2. A file machine (FM), which accesses the database file collection at a node.
3. A Network Access Machine (NAM) – connecting DBM to the communications network.

The user machine can initiate a query from one of the following types:–

1. Network related queries (e.g. topology information).
2. A request for file-names of the database collection of a node (data directory).
3. A request for the logical maps pertaining to a database file.
4. Set up a search vector consisting of a Boolean combination of field-names and values.
5. Send off a query to the 'Net'.
6. Examine request log to find status of requests.

7. Examine returned data as a result of a request.

A set of utilities is provided to:

1. Sort
2. Print from record n to record m
3. Print records satisfying simple conditionals

The current implementation of the system only supports retrieval from the net with local update.

REFERENCES

Anderson, G.A. and Jensen, E.D.'Computer Interconnection Structures: Taxonomy Characteristics and Examples.' ACM Computing Surveys, vol.7, No.4, Dec.1975, pp.197-213.

Anderson, R.H., Harshem, E.F., Heafner, J.F., Cerf, V.G., Madden, J., Metcalfe, R.M., Shoshami, A., White, J., and Wood, D.C. 'The Data Reconfiguration Service – An experiment in Adaptable Process to Process Communication', Proceedings of Symposium on Problems in the Optimization of Data Communication Systems, October 1971, Palo Alto, pp.1-9.

Casey, R.G. 'Design of tree networks for distributed data'. AFIPS proceedings, vol.42, NCC 1973, pp.251-257.

Canaday, R.H., Harrison, R.D., Ivie, E.L., Ryder, J.L., and Wehr, L.A. 'A back-end Computer for Data Base Management'. Communications of ACM, vol.17, No.10, Oct. 1974, pp.575-582.

Chang, Shi-Kuo, 'Data Base Decomposition in a Hierarchical Computer System'. IBM Research Report, RC5345, March 1975.

Chang, E.J.H. 'A Distributed Medical Data Base: Network Software Design'. Computer Networks, vol.1, No.1, June 1976, pp.33-38.

Chu, W.W. 'Optimal File Allocation in a Multi-Computer Information System' IEEE Transactions on Computers, vol.C-18, No.10, October 1969, pp.885-889.

Eswaran, K.P. 'Placement of Records in a File and File Allocation in a Computer Network'. IFIP Conf. proc. August 1974, pp.304-307.

Gentile, R.B. and Lucas, J.R. 'The TABLON Mass Storage Network'. AFIPS proceedings, SJCC 1971, pp.345-356.

Housel, B.C., Lum, V.Y., and Shu Nan. 'Architecture of an Interactive Migration System (AIMS)'. ACM SIGMOD Worshop on Data Description, Access and Control, Rustin, R. (Ed.), May 1974.

Levin, K.D. and Morgan, H.L. 'Optimizing Distributed Data Bases – A Framework for Research'. 1975 National Computer Conference, AFIPS proceedings, vol.44, pp.473-478.

Liu, S. and Heller, J. 'A Record Oriented, Grammar Driven Data Translation Model'. ACM SIGMOD Workshop on Data Description, Access and Control, Rustin, R. (Ed), May 1974.

Mahmoud, S. and Riordan, J.S. 'Optimal Allocation of Resources in Distributed Information Networks'. ACM Transactions on Database Systems, vol.1, no.1, March 1976, pp.66-78.

Marrill, T. and Stern, D. 'The Datacomputer – A Network Data Utility'. AFIPS proceedings, vol.44, pp.389-395, NCC 1975.

Mullery, A.P. 'The Distributed Control of Multiple Copies of Data'. IBM Research Report RC 5782, December 1975.

Ritchie, D.M. and Thompson, K. 'The UNIX time-sharing system'. Communications of ACM, vol.17, No.7, 1974, pp.365-375.

Schneider, G.M. 'DSCL – A Data Specification and Conversion Language for Networks'. ACM SIGMOD International Conference on Management of Data, King, W.F. (Ed), May 1975, pp.139-148.

Shu, Nan C., Lum, V.Y., and Housel, B.C. 'An Approach to Data Migration in Computer Networks'. IBM Research Report, RJ 1703, January 1976.

Su, S.Y.W. and Lam, H. 'A Semi-Automatic Data Base Translation System for Achieving Data Sharing in a Network Environment'. ACM SIGMOD Workshop on Data Description, Access and Control, Rustin, R. (Ed), May, 1974.

Whitney, V.K.M. 'A Study of Optimal File Assignment and Communication Network Configuration in Remote-Access Computer Message Processing and Communications

Systems'. SEL Technical Report No. 48, U. of Michigan, Ann Arbor, Mich., Sept. 1970.
Yamaguchi, K. and Marten, A.G. 'Methodology for Transferring Programs and Data'. ACM SIGMOD Workshop on Data Description, Access and Control, Rustin, R. (Ed), May 1974.

8 Linked Data Base in a Geographic Application

B.G. Cook

SUMMARY

A data base design and implementation for a geographic application are described. Two data bases were used, one to store positional data and relationships and the other to handle attribute data and relationships. The data bases are logically linked, and communicate by files passed between them.

THE APPLICATION

The South Coast Project was a multidisciplinary land-use study of a 6000 km² area on the south coast of N.S.W. Its overall aim was to develop and demonstrate a methodology for land-use planning at a regional scale. An early stage in the project was a biological, physical and socio-economic inventory of the area. This was to provide the basic data for the application of explicit criteria which were to be designed to define the suitability of areas of land for particular uses.

The project area was initially subdivided into approximately 4000 regions on the basis of uniform biological and physical pattern and socio-economic status, the region boundaries being delineated on transparent compilation sheets, overlays to 1:25 000 or 1:31 680 scale topographic maps. Descriptive data about each region, such as shire, land tenure, present use, recreation possibilities, roads, altitude, etc., were compiled or collected from available maps, air photographs, etc.

In general, the regions defined were not homogeneous in landform, but contained patterns of landform components or *facets* such as crests, slopes, plains, streams, etc., too complex and numerous to be mapped and individually described for the whole area. Fig. 1 illustrates how one region maps into nine recurring facets. Soil associations and vegetation communities were assumed to correspond well to the observed landform facets. The facets occurring in each region were identified (but not mapped) by interpretation of contour maps and airphotos and a stratified field sampling program was conducted to establish typical landform, vegetation and soil descriptions which sufficiently represented the landforms, vegetation communities and soil associations observed. Each of about 22 000 facets was assigned to the most appropriate descriptive class for each of landform, vegetation and soil.

These attribute data (the above gives only an idea of the approximately 150 data items involved) were to form the raw material upon which land-use suitability algorithms would operate. These algorithms would be one product of a series of land-use studies of food production, forestry, recreation, urban activities and conservation.

If the body of data were to be addressed with reasonable efficiency, it was evident that some form of structured data base organization would be needed.

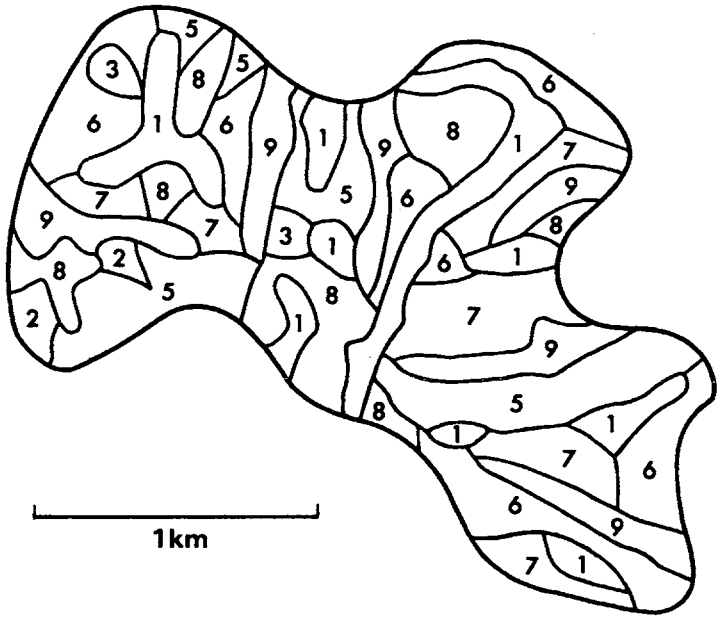


Fig. 1 A region mapped into facets

The data base would be essentially a research tool to experiment with and use the data gathered, data which, for the duration of the project, would be regarded as essentially static. Once established, the data base would not be subject to continual update.

DESIGN DECISIONS

A problem was that data base design must proceed without any clear specification of how the data would be used. One thing was reasonably certain however, a common method of display would be by thematic maps of the whole or a part of the area. This implied that particular attention should be paid to the organization of the map data defining the region boundaries.

There is, of course, no *necessity* for data specifying region boundaries (or other geographic location data) to be treated differently from other data. The coordinates defining the lines of a boundary can be regarded as attributes of the bounded region similar to other descriptive attributes such as soil properties or land tenure class. But the elements of a map, the points, lines and regions, exhibit strong relationships with one another (point to line, line to line, line to region, region to region, etc.), and the network of elements and relationships can be very dense. Furthermore, it is common in map applications for *all* the information within a given rectangle (say) to be needed, requiring that many relationships be followed and a large number of map elements accessed. Thus, there may be advantages in giving special attention to the structuring and storage of map data to allow efficient access to all data within any specified area.

The decision was made to use two linked data bases for the project — one to store the map data and the other to hold the descriptive attribute data. (This decision was strongly influenced by the fact that two apparently suitable data base systems were under development and expected to be available soon: a map data base system being developed in the CSIRO Division of Land Research

(now Land Use Research), and an implementation of the Codasyl Data Base Task Group (DBTG) recommendations in the Division of Computing Research.

ATTRIBUTE DATA BASE

The attribute data base for the project was established within the FORDATA system (Smith and Mackenzie 1976), an implementation of the Codasyl DBTG Report (Codasyl 1971) under Fortran for Control Data Cyber computers. Concepts of the Codasyl data base structure have been discussed by Taylor and Frank (1976) and by Mackenzie in the previous paper. Using the schematic conventions adopted by the DBTG, and used by Mackenzie, features of the structure used are discussed below.

The *region* and *facet* defined above being clearly the key entities in the descriptive scheme adopted for the project, the skeletal structure of Fig. 2 was adopted. It represents a hierarchy of the entire *project* area, subdivided into *regions*, each *region* being further subdivided into *facets*.

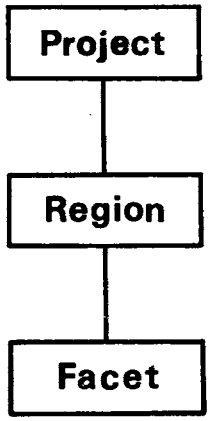


Fig. 2 The project, region, facet hierarchy

Some groups of data, descriptive of the regions, defined simple region classes which, if explicitly represented, would provide an efficient means of accessing important sub-sets of the regions. For example, land tenure, defined by class and sub-class, could facilitate access to just those regions having a particular land tenure. Data such as these were incorporated as in Fig. 3, forming

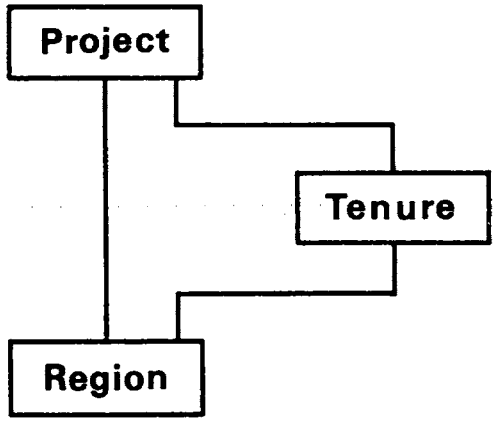


Fig. 3 Structure for region classes

hierarchies from *project* to *region*. Representation of the classification of regions by the topographic map sheets on which they occurred demanded a more complex structural solution. Here, rather than the one-to-many relationship exhibited by the *tenure* and *region* records, there is a many-to-many relationship, requiring an additional link or relational record type as in Fig. 4 to represent

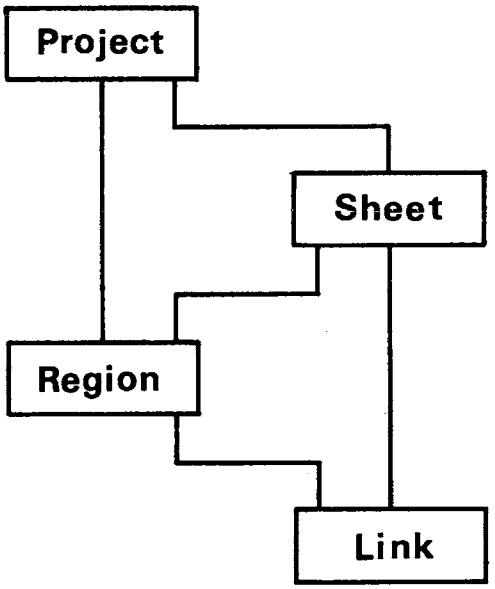


Fig. 4 Structure for the map sheet: region relationship

those parts of a region sub-divided by map sheet borders. A simpler sheet to region relationship was retained for those regions wholly within a map sheet.

Other groups of region attributes, which did not define a usefully-small number of classes, were related to the region as in Fig. 5. An example is present

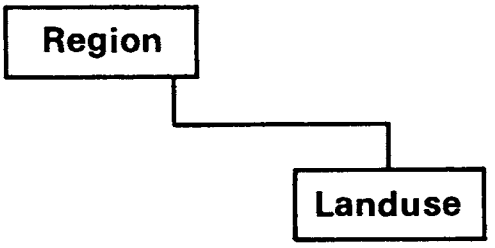


Fig. 5 Structure for region properties

land use, one record being used for each land use occurring within a region. Some other records structured into the data base in this way are logically merely extensions of the *region* record, in that one and only one is related to each *region* record.

The soil, landform and vegetation descriptions, to which individual facets had been assigned, were represented in the data base as in Fig. 6. Here the *landform* record, for example, provides an intermediate level in a *project* to *facet* hierarchy.

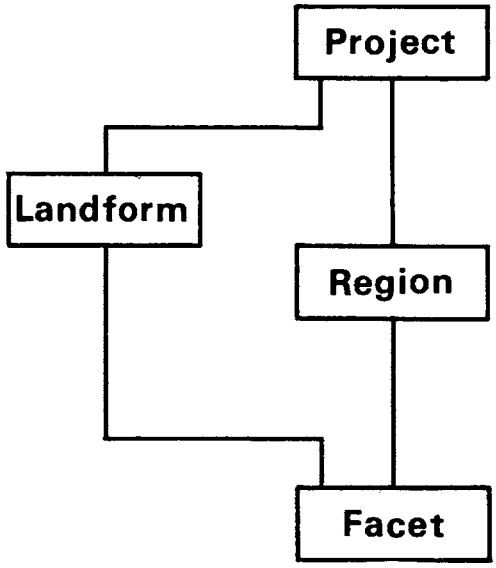


Fig. 6 Structure for facet description classes

One further structural feature remains to be explained. It was thought that, for some land-use suitability algorithms, the properties not only of the region considered but also of its adjacent regions would need to be taken into account. *Identification* of region adjacencies would be a function of the map data base system; however it was felt that efficiency would require that adjacency be represented in the attribute data base as well. This is another many-to-many relationship, but between occurrences of the one record type, that is *region* to *region*. Fig. 7 illustrates the structure used, the relational record representing an adjacent pair of regions, two different set types being required (since the map-

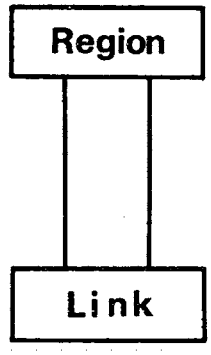


Fig. 7 Structure for region adjacency

ping from set member to set owner must be unique). The choice of which region occurrence of an adjacent pair is linked via which set was an arbitrary one; to find the regions adjacent to a given region the relational records (and thence the adjacent *region* records) in both sets must be accessed.

The entire structure of the attribute data base is indicated in Fig. 8, which

also gives the number of occurrences of each record type. The simple structural forms described have been combined into a network of 28 record types using 35 set types.

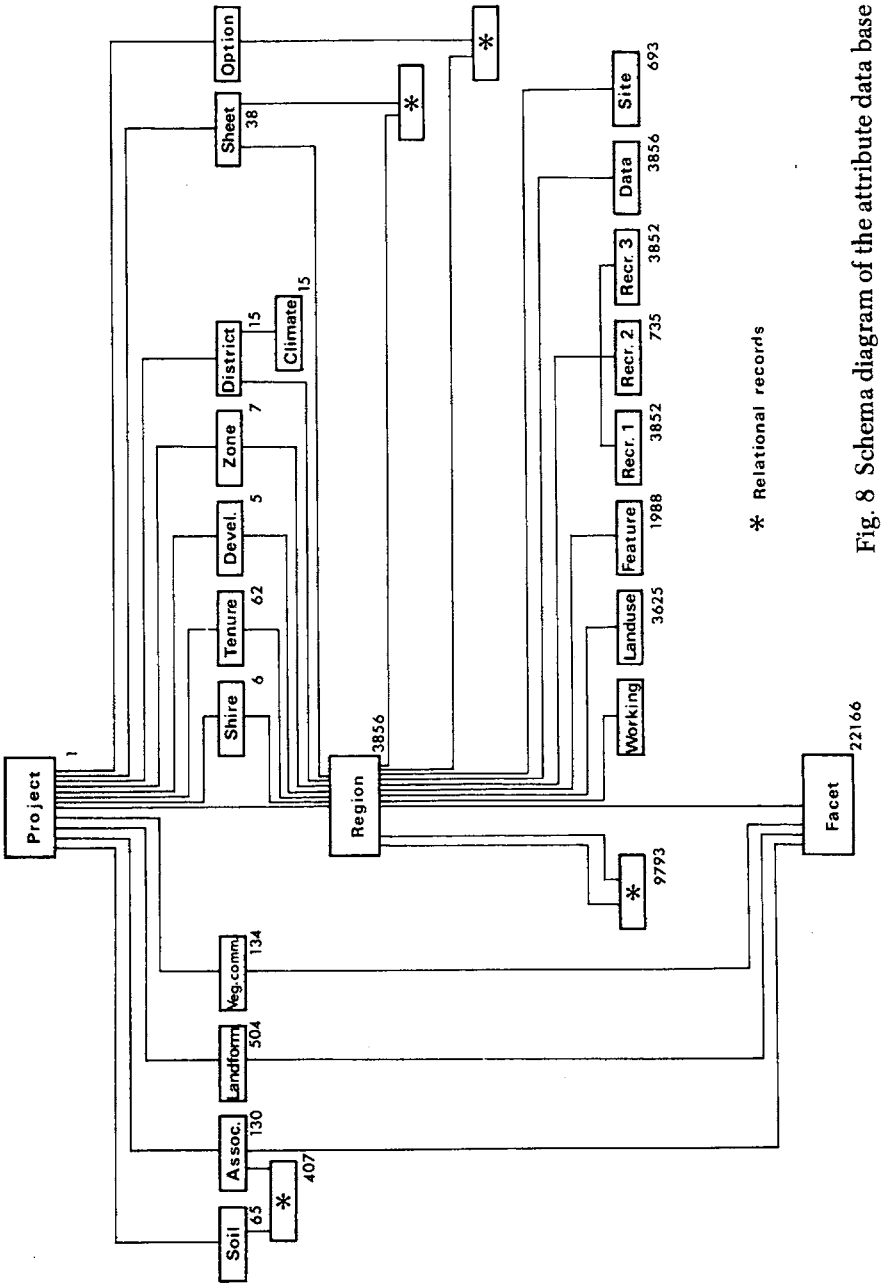


Fig. 8 Schema diagram of the attribute data base

MAP DATA BASE

The map data base, which stores the map of region boundaries for the project, makes use of a system designed and implemented in the CSIRO Division of Land Use Research, and intended for use in similar application areas. The system has some basic capabilities beyond those used in the application described, although some of these are under-developed, not having been used in an application.

The data is structured at two levels: firstly into a mosaic of square *tiles* and secondly, within each tile, as a network of graphic elements.

Each tile of the map corresponds to a single fixed length page in the storage structure. The density of tiling varies with the density of map data; a tile is automatically subdivided into four when the corresponding page is at the point of overflow. Fig. 9 shows the mosaic generated by the map data for this project.

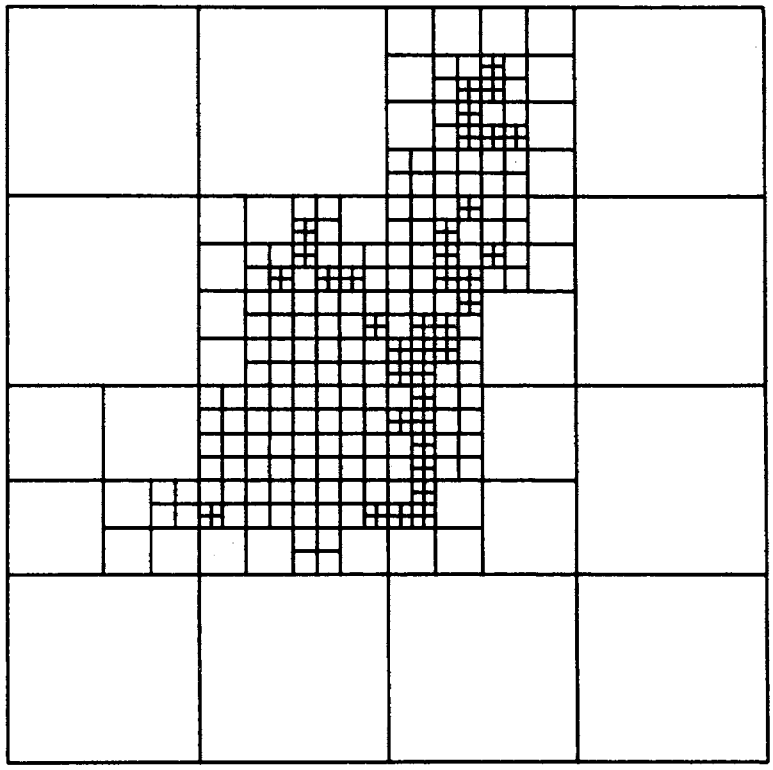


Fig. 9 Tile mosaic of the map data base

Within each tile, the map is represented by a network of structural components as follows. *Junctions* (J) are linked by chains of *points* (P) containing packed incremental coordinates relative to absolute coordinates held in *junctions* (Fig. 10). A *line* (L) points to the extreme elements of the corresponding inter-junction *point* chain, and to the adjacent *lines* clockwise about the regions on either side (Fig. 11). *Attributes* (A) can be associated with *junctions* and *lines*, and explicitly represent regions. Each *junction* points to one of the *lines* radiating from it. Each *line* points to the *attributes* representing the regions on either side (Fig. 12).

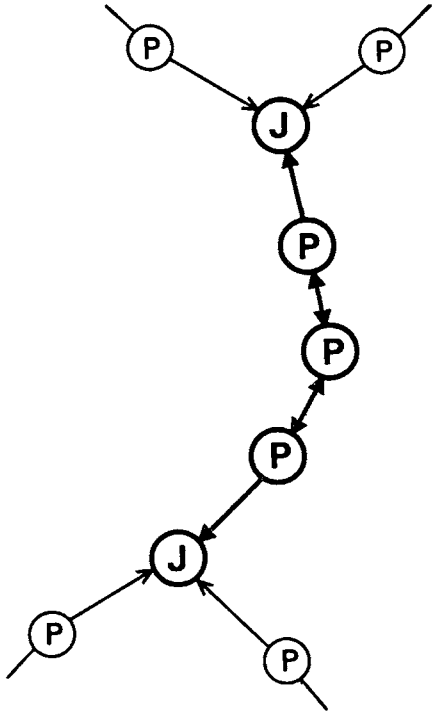


Fig. 10 Map structure—junctions and point chains

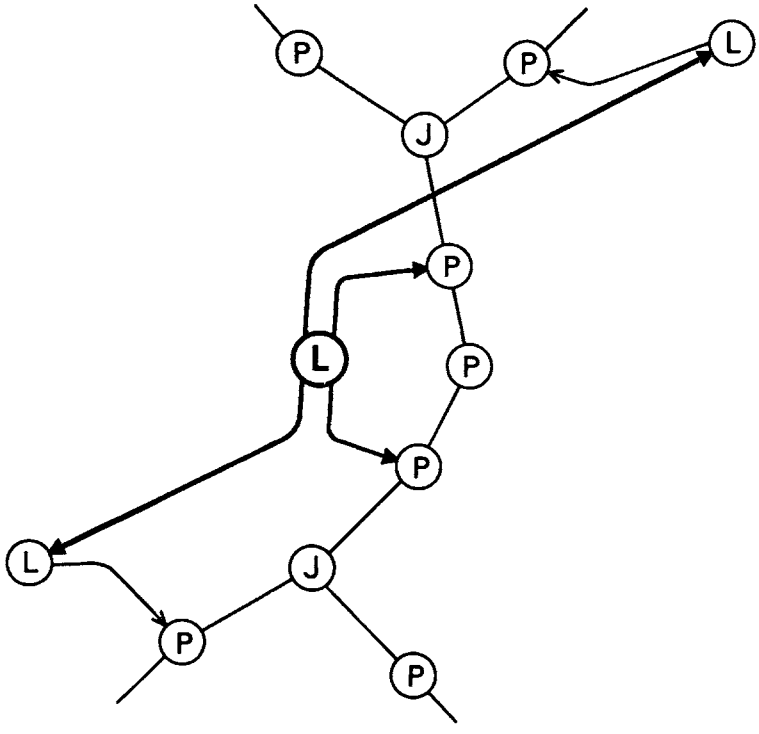


Fig. 11 Map structure—lines

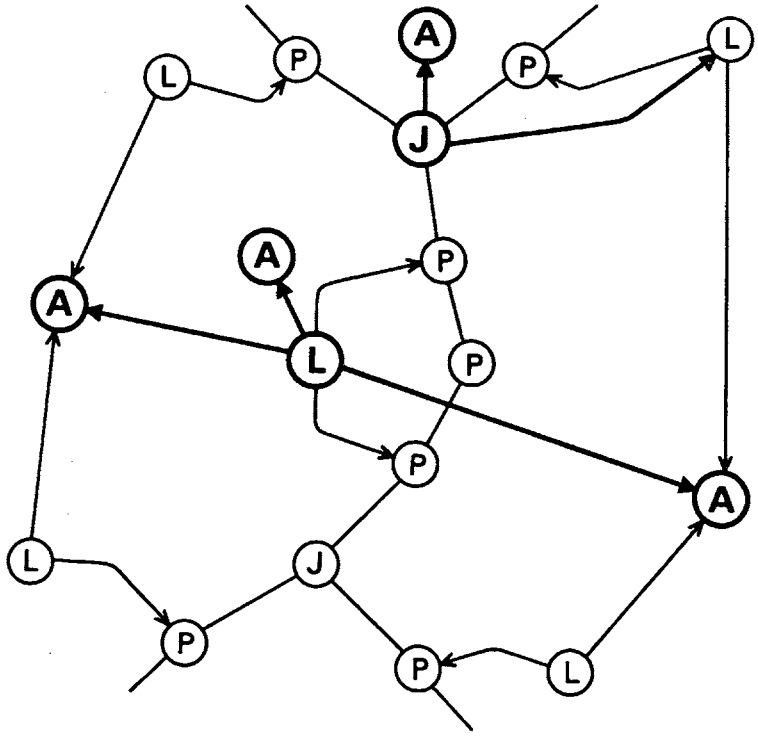


Fig. 12 Map structure—attributes

Additional *lines*, having no corresponding graphic in the map, are used as tile border lines and as virtual lines which connect inclusions to an enclosing boundary (Fig. 13).

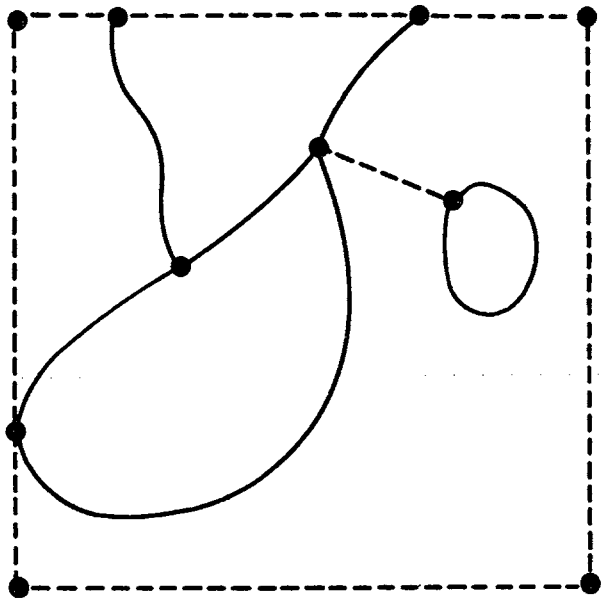


Fig. 13 Map structure—tile border lines and virtual lines

Attribute elements can hold a limited amount of descriptive data or, alternatively, pointers to descriptive data stored elsewhere.

Entry to a map data base is by the cartesian coordinates of a point which determine the tile to which the point belongs; that tile is then searched to determine the region within which the point lies. The most common retrieval method is to impose a polygonal (usually rectangular) window on the map by actually incorporating its boundary into the map base, and systematically traversing the map within the polygonal frame to recover whatever information is required. The imposed frame is progressively deleted during the course of the traversal to leave the data base virtually undisturbed at its completion.

LINKING THE DATA BASE

The region is the only entity common to both data bases, being represented by the *region* record in the attribute data base and by *attribute* elements in the map data base. Linkage between the data bases was therefore established at the region level. To provide a link from the map data base to the attribute data base, the data base key of the *region* record was planted in the *attribute* element(s) representing the corresponding region in the map base; this provided a direct access path from the map into the attribute data base. To link in the other direction, the cartesian coordinates of a point within the map region were included as data items within the *region* record (the point used was one computed to be a suitable location for placing a label within the region boundary).

The two data bases were constructed independently, the links being established in a subsequent phase. The procedures which effected the linking also passed the area of each region to the attribute data base, established the region adjacency pairs, and grouped the regions by map sheet.

In the absence of an operating system facility for communication between running jobs, the data base systems communicate by passing files between them. The total data base system is represented in Fig. 14.

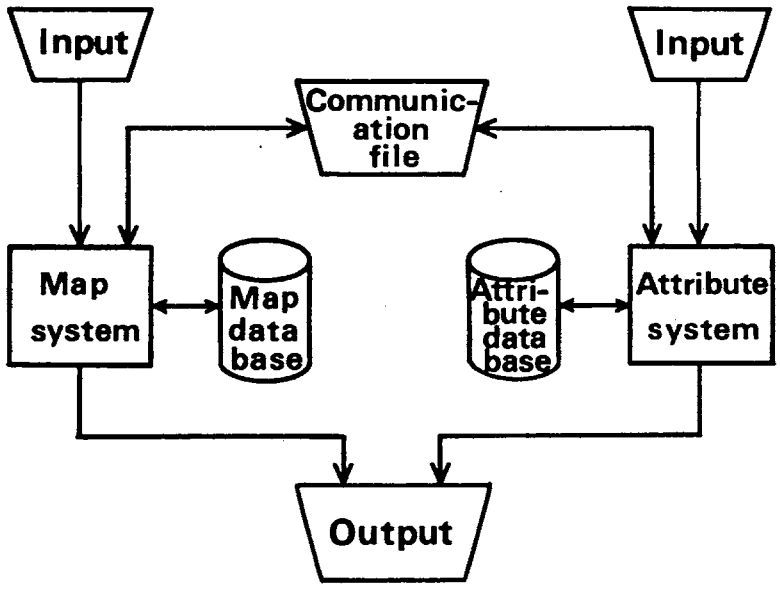


Fig. 14 Linked data base system

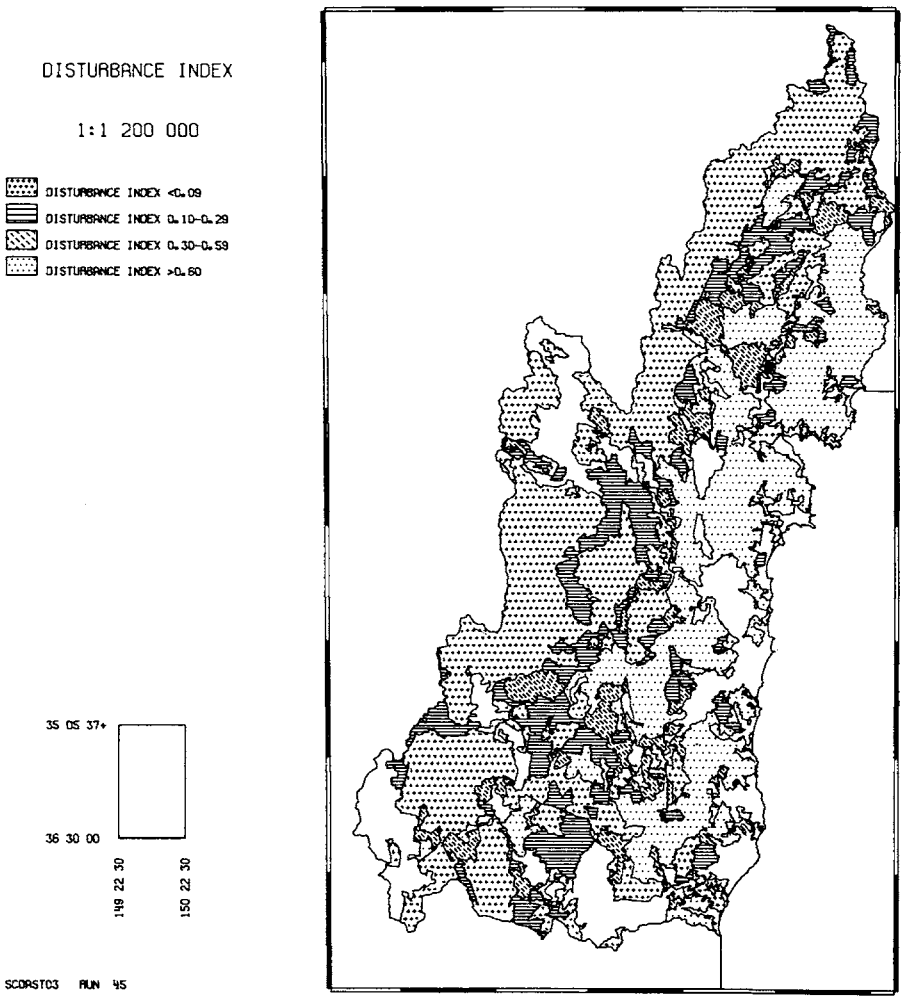


Fig. 15 Thematic map output from the system

USING THE DATA BASES

Entry to the data bases is either via the *project* record of the attribute data base to a sub-set of *region* or *facet* records, or by definition of a window of interest in the map data base and thence by direct access to the *region* records of the attribute data base which correspond to the regions enclosed. The latter method is usual when a thematic map is to be produced; a typical sequence of procedures is as follows.

1. In the map data base, input a polygon defining the area of interest and traverse the area, accumulating a file of attribute data base keys corresponding to the regions wholly or partially in the area, and computing the coordinates of a suitable labelling location for the partial region.

2. In the attribute data base, access directly each *region* record whose data base key is in the file and evaluate for each a specified function of its related data items, assigning to it a label determined by the function value. Update the

file with the assigned lable for each region and insert the labelling coordinates for the regions wholly within the area.

3. In the map data base, reinput the windowing polygon and traverse the map within it, plotting the region boundaries. At each labelling location in the file, plot the corresponding label.

The results of such a retrieval are shown in Fig. 15. Here, however, the whole of the project area is included, unnecessary lines of the map have been edited from the map data base, and the resulting regions hatched rather than labelled. (The map displays an index of disturbance of the natural vegetation as a basis for assessment of capability for conservation.)

The data bases as described have been in use for about two years, during which few defects have become apparent, although more generous provision of working areas in the attribute data base structure would have been useful.

Retrievals from the map data base are specified by parameter. The programming of retrievals from the attribute data base has used data manipulation language (DML), necessitating the employment of skilled programmers. More recently, however, the FORDATA sub-schema facility has proved its value and some limited work has been done on tailoring simple high level languages for specific sub-schemas.

REFERENCES

CODASYL Data Base Task Group Report (1971). ACM: New York.

MACKENZIE, H.G. (1977). These proceedings.

SMITH, J.L., and MACKENZIE, H.G. (1976). FORDATA: A Database Management Package under Fortran on CDC Cyber Computers. CSIRO Division of Computing Research.

TAYLOR, R.W., and FRANK, R.L. (1976). CODASYL data-base management systems. *Computing Surveys* 8, 1, 67–103.

9 The Place of Data Dictionaries in Large Scale Data Management

D.R. Erbacher

ABSTRACT

The paper outlines the problems involved in large scale data management as opposed to data base management. The capabilities of Data Base Management Software (DBMS and Self Contained Systems) are discussed in relation to the problems of data management. The conclusions drawn indicate the need to supplement this software by a Data Dictionary System. The properties of Data Dictionaries currently available are discussed and the need to expand their capabilities in relation to particular organisational needs is discussed. A distinction is drawn between data dictionary and data directory the former containing basically information about data and the latter information about operations on data. A preprocessor is suggested as an appropriate mechanism for controlling the storage and use of information in the data directory.

INTRODUCTION

During the last five years data base management tools (data base management systems, query languages and report writers) have experienced an increasing popularity and recognition as tools to assist the storage, maintenance and dissemination of data. The purpose of this paper is to attempt to demonstrate the partial achievement of users' data management requirements (particularly large scale data management requirements) by these tools, to indicate the appropriateness of these tools for solving some of the problems of data management and to suggest some possible ways of supplementing these tools in order to more nearly achieve the aims of users' in relation to their data management requirements.

Being generalized software these tools aim to satisfy a number of users' requirements in the one implementation. However no matter how generalized, (and extensive generalization incurs unacceptable development and operational costs), there always remains some requirements that need to be implemented by customers, specifically to meet their own data management requirements.

Jeremy Firth in his paper on the Infological approach has outlined and end-user top-down approach to data management and system definition. To the extent that he has proposed a philosophy of system development I can say I feel a degree of sympathy for the ideas he expresses. However in terms of software to support such an approach one must ask whether it is implementable. It is certainly not procurable in the market phase. Give the tools available and keeping the organizations data management requirements in mind, installation management should be able to create an environment into which the most appropriate of the tools can be placed in such a way as to satisfy the bulk of the requirements.

WHAT IS DATA MANAGEMENT?

Any organization requires data about its operation and environment in order to

function. It requires this data in order to make decisions so that it can operate successfully. Consequently a major task which must be performed in the organization is the collection, processing and availability of this data so that it meets the needs of the decision makers in the organization. To enable each member of the organization to perform his task, whether it be to extract data or to assist and organize for dissemination of data, he must know what data is available, where and how it is stored and how to get at it.

In most organizations where large systems are being implemented, managers are recognizing data to be an institutional or corporate resource. That recognition represents a shift in understanding and emphasis about where data originates, what processes it undergoes, and how it is accessed and released (both internally and externally to the organization). Until data is regarded as a resource which belongs to the organization instead of a division or department, there was only minimal awareness that data (as a resource) requires management, planning and control similar to that required by other resources. The real shock to managers, however, does not come so much from the discovery that data is an institutional resource, but that the data as it presently exists in various parts of the organization cannot be managed let along controlled.

For example, many organizations find their data to be conflicting, inaccurate, superfluous, or wasteful. In addition, the documentation and coordination is scanty and so poor that proper management and control are difficult, if not impossible. The knowledge about data and where it exists is spread about computer files, manual files, documentation, scraps of paper and the minds of people.

The aims of any large scale coordinated data management environment should be to provide the following:

1. to make available data in a suitable form to users who have a legitimate need to access and manipulate it;
2. to store data in a manner accessible to legitimate users;
3. to provide a range of tools to enable users to store, update, manipulate and extract data in a form suitable to his purpose and in a responsive manner;
4. to facilitate the above in as efficient manner as possible taking into account both machine efficiency and ease of use by all people concerned;
5. and finally to maintain this service in an environment of ever expanding scope (in terms of size and nature) and propensity to change (in terms of data requirements and software and hardware advancements).

As an organization expands its service and increases in size it moves towards an increase in functional specialization. During this process the information systems became more selective in its dissemination of information, supplying to each area only what is necessary to support it. Two problems are created by these changes. Users lose sight of the overall function of the organization and tend to become unaware of the significance of their part in it and fail to appreciate the affect they have on it. Also, it becomes difficult to ensure that the information disseminated is the best for the purpose and is as complete as possible.

Information systems can be classified by levels of complexity as follows.

1. small systems where the control over flow of information is entirely informal;
2. simple systems with simple formal controls;
3. systems large enough to be departmentalized with highly formalized in-

formation flows;

4. large systems where the information flow is such that a control function must be established to manage it properly;
5. large systems with a complex structure requiring automation of many of the information systems' control functions to deal with the variety of information management tasks now tackled;
6. very large systems which are so complex that additional aids are required to predict and optimise the information systems' performance.

As an organization's information systems develop in size and complexity, through the above levels, standards alone fail to ensure an optimum use of the data resource. This is due to the fact that as the system develops, its data becomes available to more and more functional areas which may not inform each other fully about the uses to which they put the data. In fact the amount of information about the data itself becomes large and unwieldy. This creates a need for formalized methods of handling the information and a need to establish a data administration function to control them. This paper asserts that the formalized methods are embodied in the Data Dictionary System which provides the means for effectively recording and handling this new information.

The Data Dictionary System is particularly applicable at levels (3) to (6) above. At levels (3) and (4) it may be possible to maintain control with a manual Data Dictionary System. At level (5) the automation of control functions will require an automated approach. Level (6) requires a full battery of automated facilities including impact analysis and simulation.

Information systems are rarely static in form and may change for a variety of reasons. Normal development and growth of the organization leads to a progressive development of the information systems through levels of complexity such as those described above. Changes in organization structure brought about by reorganization, merger or change of government may produce abrupt change in the form and complexity of the information system. Changes in the software environment by the introduction of a DBMS may alter the form and complexity of the information system.

The effectiveness of the information system will be impaired if it is not possible to incorporate the results of these changes quickly and smoothly.

Another facet complicating large scale data management is the level of user sophistication in terms of both an understanding of the organization's data and computing expertise. Users vary from the casual, uninformed, ad hoc user who has a problem to be solved and needs to understand the availability of data to help formulate his problem to the expert data manipulator and disseminator who has a thorough understanding of the tools available and can manipulate them efficiently to achieve his purposes. The casual user needs a large amount of information about the data available, where it is, how to get at it and the limitation of the data in relation to his purpose. At the other extreme the expert while he may be proficient in the use of the tools can't possibly remember the state, placement and limitations of the data when the volumes of data being handled increase.

Other users requiring different information include, data managers, data base administrators, systems programmers, application development and maintenance programmers and management at a number of levels.

The number of users with differing access data and information requirements in a large scale data management environment necessitate a systematic approach to security, integrity and accountability if they are to co-

exist while making demands for scarce resources from an integrated information system. One might expect the expert user (with some controls on and boundaries limiting his operations) to be an efficient user in terms of the machine and his own time, but at all levels the user needs information about data and facility availability rules, guidelines and standard practices. Most importantly he needs help as he formulates his requests for information. If this guidance can be automated using the information about data in some form of 'browsing' system then all users will become proficient in terms of being quickly guided to an appropriate use of resources. The three main attributes for users are a user profile indicating access privileges to data and facilities, a browsing system to assist him automatically, where possible, to solve his problem and a central storage of information about data and its operations on it.

THE TOOLS AVAILABLE

The market today offers a range of Data Base Management Systems (DBMS), query languages and report writers operating on data bases which I will label self-contained Systems (SCS). How do these systems perform in relation to the data management problem described above. At best they meet only part of the problem even if you grant that they meet that part well. I don't propose to go into detail about the capabilities of packages. Suffice it to say that DBMS's offer a range of structuring techniques to handle complexity in data structuring and accessing. They provide levels of security, integrity and recovery, definition languages as well as manipulation languages, varying degrees of data independence and a range of utilities for maintenance purposes including restructuring and reorganizing facilities. SCS's provide a means for non DP professionals to interrogate and produce reports by using an english-like, high level, non procedural language.

However the use of these software tools requires the establishment of a central control and management function called the Data Base Administration (DBA) which is concerned wholly or in part with at least the following:

1. data analysis;
2. maintenance of storage of data;
3. documentation;
4. data base design;
5. restructuring and reorganization of data;
6. storage structures;
7. data independence;
8. security and integrity;
9. backup and recovery;
10. data standards.

The information the DBA needs at his disposal in order to control these information flows is very large in large scale data management and such a DBA must attempt to use its best tool, the computer, to store this information and to automate the utility function performed against this information as far as possible.

No matter how good the DBMS acquired is, it will be deficient in relation to the organization requirements in areas such as data independence, ease of use, security, integrity, recovery, restructuring, reorganizing, statistics gathering and manipulation function. It will certainly not link well to existing software tools, either procured or privately built, to form a total compatible mix of software tools to meet the organizations total data processing requirements.

SCS's provide a limited casual user access without any significant browsing

capability and not linked to a central storage of information to assist users. The SCS languages attain varying levels of non procedurability and may be closely tuned to the individual organization's requirements or not.

Kemmis (1976) says that rarely is there to be found a thorough discussion on the relevance of the DBMS's major aspects to a specific organization. He suggests that data base implementation is fraught with fear and that fear of understanding and managing this huge task causes us to draw back, to limit our objectives. It seems that some of the failures of DBMS implementations is due to not understanding the part to be played by data management software in relation to data management objectives.

It comes as no surprise that the advent of the data base management software tools has greatly accelerated the already existing trend to Data Dictionaries. Let us look in general at the capabilities of Data Dictionaries currently available and see how far they supplement the capability of data base management tools in relation to our large scale data management problems.

Data Dictionaries on the market today have two major components, viz, a storage of information and facilities to update information, and utility functions which operate on the information. Data Dictionaries store a subset of the following information:

items, fields;
records, groups, segments;
files, data sets;
programs;
modules;
systems;
data bases;
transactions.

Data Dictionaries store information to enable a subset of the following functions to be performed:

generation of DBMS descriptions;
generation of input/output modules;
generation of report programs;
generation of input validation programs;
generation of COBOL data divisions;
generation of a range of output documentations;
generation of test data.

While this development is desirable and will continue much still is possible to be done. The above list of capabilities would be of use to any organization with a data management problem and I suggest that each organization has special requirements in addition to those above and needs to automate the requirements by the enhancement of the Data Dictionary capability outlined above.

Let us now look conceptually at Data Dictionary contents.

CONCEPTUAL VIEW OF DATA DICTIONARY CONTENTS

The general objectives of a Data Dictionary are to

1. improve the collection and dissemination of information regarding availability, definition, location and use of data, programs, systems and reports;
2. improve control, coordination and integration of data resources, particularly in the data base environment;
3. increase the efficiency of users, programmers and administrators by

providing facilities to simplify or automate the more routine programming and clerical tasks;

4. reduce application system and user request development, implementation and modification times and costs.

Although these objectives are similar to the often cited objectives of a DBMS, to a certain degree these can be achieved even outside of a DBMS environment by means of a Data Dictionary. However, the combination of a Data Dictionary and DBMS can achieve these objectives to a much higher degree than either by itself. A conceptual definition of a Data Dictionary might be: a tool which contains all the attributes about the description and use of data by both people and computer programs in such a manner that more flexible and dynamic interactives are possible and that management and control of data are facilitated.

When one surveys the definitions of Data Dictionaries one immediately identifies various dichotomies with a common thread. The British Computer Society Data Dictionary System Working Party (BCS/DDSWP) suggests; 'A Data Dictionary System is a tool for recording and processing information about the STRUCTURE and USAGE of data'.

Martin states that the Data Dictionary can be used in two types of ways. It can be used by PEOPLE and it can be used by data base and other SOFTWARE. The distinction could be regarded as one of SOURCE definitions and OBJECT definitions. Plagman and Altshuler divide the contents of Data Dictionaries into NON-VARIABLE META-DATA and VARIABLE META-DATA. Non-variable meta-data refers to attributes which cannot change from one use of the element to another while meta-data can change from one use to another particularly over time. The representation binary string as opposed to character string would be classified as variable while the name of the element as non-variable. Uhrowczik states that a Data Dictionary is a central repository of information about data descriptions. It is the basic tool in the data management environment that assists company managers, systems analysts, applications programmers and data base administrations in effectively planning, controlling and evaluating the collection, storage and use of data as a resource. This particular usage of the Data Dictionary is termed the MANAGEMENT USE MODE. The process of application development and maintenance can be further improved by extending the use of Data Dictionaries to programming systems — compilers, DBMS and so forth. This usage of the Data Dictionary is called the COMPUTER USE MODE.

All these seemingly different views are of the one dichotomy and are useful constructs in helping to understand the nature of Data Dictionaries. I prefer personally to use Data Dictionary/Directory System (DD/DS) to highlight these conceptual differences. In practice however the DD/DS forms a unified whole, an integrated set of data (bases) organized to meet predetermined needs. The dichotomy is artificial but useful to understanding the concepts of DD/DS. In practice the descriptions of data need to be related to the usage of data to meet the data management objectives. The contents of the Dictionary part change only as the organization and its environment change. The Directory part changes in order to continuously support the efficient data management function in terms of people and machines.

HISTORY OF DD/DS

Since computerized data processing began there has been a tendency for programmers to isolate their procedure logic from data variables, and data

descriptions. Input/output modules called from the main stream logic, preset variables at the front of programs, auxiliary files and even COBOL's data divisions, COPY library and working storage have separated the relatively volatile elements of his program from his procedural logic. Even some procedure logic such as edits which change sometimes have been isolated by use of decision tables and storage in ancillary files. The current trends in data base management tools (including DD/DS) are an extension of this notion. In this sense DD/DS are not new although the centralized notion of DD/DS has come to the fore by virtue of DBMS. However, King, in 1969 proposed a Data Dictionary as a central tool for use in formalizing techniques for systems design and analysis. If DD/DS would have come to the fore independently of DBMS then at least DBMS showed the way as regards centralization and speeded up the process.

Recently two reports from eminent bodies have extended the understanding of the DD/DS concept. I refer to the ANSI/X3/SPARC Study Group on Data Base Management Systems — Interim Report, and the British Computer Society, Data Dictionary Systems Working Party, Draft Interim Report. The SPARC committee's chartered purpose is to investigate the potential for standardization in the area of data base management systems. The architecture they describe depicts processing functions and human functions and isolates a number of interfaces between these functions that are the subject of possible standardization. In addition the SPARC committee isolates three models:

1. conceptual model — a collection of objects that represents the entities in an enterprise independently of computer storage;
2. internal model — a collection of objects containing the stored data that represents the external and conceptual models concerned with the most economical use of the computing facility consistent with processing requirements;
3. external model — a collection of objects of interest to a specific application or family of applications.

Each of these models is associated with a schema and schema processor in the architecture and a DD/DS facility is included to store such information as is needed for the control of this architecture and to allow automation of the mapping between these schema processors.

The importance of this report to the advancement of DD/DS is that if future implementations follow this model, DD/DS will disappear as a separate piece of software and become integrated within the DBMS.

The BCS/DDSWP's prime aim is to produce a report on the use of and the facilities which should be provided by, Data Dictionary Systems and related data base design and operational aids.

The BCS/DDSWP outline a conceptual data model, an implementation data structure and a mapping between the two. It outlines the uses of DD/DS in relation to the users. The concept portrayed is of a DD/DS in a central place in an organization's data management environment. In this sense it is a guideline to users in terms of establishing an environment now. On the other hand the ANSI SPARC concept of total integration of all data management tools is the ultimate way to go.

DATA STRUCTURE OF DD/DS

One must remember that if the data of an organization is to be integrated then the data about the data is more heavily so integrated. When one looks at the structures of DD/DS one finds a multitude of links between different entities — a much higher linkage rate than in the data they describe. Many organizations

will require many entities and attributes to adequately cope with their data management problems. There are two main reasons for an organization to want to expand the DD/DS it might purchase. The first is to overcome the deficiencies of the DBMS and SCS in terms of missing function, incomplete function and ease of use. The second is to support utility functions peculiar to the organizations data management requirements.

There are some functions that should rightly be in the operating system, some in the DBMS and other software. The DD/DS, however, provides the possibility to expand software and interface software elements into a total data processing environment by the storage of relevant data and the application of additional software to access that data. Also while it is desirable for purposes of relating data that it all be centralized a reasonable compromise is for the DD/DS to access data stored in operating systems files and elsewhere.

THE USERS OF DD/DS

As suggested by the capabilities of a DD/DS, a variety of users can interface with the DD/DS. The design aim of a DD/DS in fact is to make availability to all data users in an organization whatever their function, the necessary information about the data they wish to access.

The trend of data processing personnel in large scale data management environments is to provide tools for users to solve their problems and as such to provide more and more tools that he can use without specialist data processing staff being heavily involved. The use of the DD/DS is likely to grow in importance as data processing systems come within the reach of large numbers of users at terminals. The casual user requires the facility to 'browse' through the available data and facilities available in order to formulate a data processing strategy to solve his user problem whereas the same DD/DS is essential for maintenance, operations and management. A terminal dialogue, question and answer software heavily based on data in the DD/DS will help a user in most cases to converge quickly to a practical solution to his problem without the need for specialists' assistance.

THE POSITION IN THE ENVIRONMENT

The central position of DD/DS in the ANSI SPARC architecture and the integrated nature of its role in the DBMS model undoubtedly constitute the ideal position of DD/DS in data management environment.

The ANSI SPARC architecture is unfortunately very theoretical, highly detailed and in an evolutionary state. One looks expectantly to further clarification of their ideas and finally commercial implementations.

The DD/DS is defined and established by use of a data definition language. This definition would by virtue of the translator produce a DD/DS data base and generate the appropriate DD/DS software to drive it. Subsequently data bases are defined by use of data base description language. The translator stores the object description for use by the DBMS proper thereby removing the duplication of information in the internal directions of the DBMS and the DD/DS.

Applications containing source code and DBMS directed commands are compiled and link edited into executable code. The DD/DS provides all parameters needed about the data being used. The DD/DS also receives information about the functions of the program regarding the data being referenced.

The DBMS translates the source programs into I/0 requests or processing I/0 and perform requests. The DD/DS provides directory information on the physical environment for last minute bridging between it and logical requests.

In this view the DD/DS is defined and established by use of the data base definition facility of the DBMS. The data base is described to the DD/DS by use of an update program through the DBMS. The DD/DS receives the definition parameters for the content type data base. Information is extracted from the DD/DS through the DBMS by a program which converts the DD/DS parameters about the content type data base into the definition file that the DBMS depends on to process that data base.

High level source code programs with DBMS directed commands are compiled by a preprocessor which converts the DBMS commands into 'standard' source code which reflects the intentions of the program to the DBMS Interface Module. The DD/DS provides all parameters needed about the data being used. The DD/DS receives information about the functions of the program regarding the data being referenced.

The preprocessor source code is compiled by 'standard' COBOL or other compiler into standard object code. The standard object code is link edited by a 'standard' linkage edited with object code requested by the preprocessor and to link the compiled program to the DBMS Interface Module at execution time.

The executable program is connected at load time to the latest version of the DBMS Interface Module which serves as a translator from the logical processing requests of the program to the physical processing requests that need to be made of the present version of the DBMS. The DD/DS serves as a source of information to assist this translation.

The DBMS uses its 'built in' data base definition file and the physically specified access requests from the DBMS Interface Module to service the program request from the data base.

The objective is utilization of the directory portion of the DD/DS. It should be pointed out that although this environment appears to be very complex, in reality it involves the creation of only three or four modules of computer code. The environment makes use of the one substantial tool the DBA has at its disposal-the DBMS — to help build and join the elements together. The DBMS is same module used for several different purposes. The other standard modules are available as routine software products. The modules which will need to be provided by the user organization are the preprocessor, DBMS Interface Module, DD/DS Update program and linkage routines. However there are good alternatives available in addition to doing 'inhouse' design and coding. The DD/DS update program is very similar to the update program used for a content type data base and can be written by applications programmers or alternatively a packaged DD/DS can be procured which includes a data manipulation translator language. The Preprocessor is already used in a similar fashion by some DBMS packages and can be either modified or extended to interface with the DD/DS in the proper fashion and made to put out logically instead of physically translated requests. The linkage routines are small and can be written in a few hours by a systems programmer. The DBMS interface module is usually very similar to code that exists in the DBMS vendor's preprocessor. This code can be consolidated and restructured. The workloads involved here are only to set up the environment. One must remember that each additional element of functionality will require extra development of code and DD/DS expansion. The more ambitious the data management aim the more expensive will be the solution. What is described here is essentially a set of environmental 'shells' communicating with each other. Of course the easiest alternative is to examine the shops of other users (of the same DBMS) and buy the necessary modules to

build the desired environment. This latter choice is becoming more and more feasible as more organizations are discovering the value of the DD/DS applied in this manner, a percentage of which choose to develop their solution and add it to the market place.

Listed in DB Systems: Data Independence from Infotech State of the Art 1975, is a number of companies including Rowntree MacKintosh in the UK and SKF in Sweden who have implemented preprocessors. The results of a survey in the USA indicated that 4 companies of 18 surveyed had or were planning preprocessors. Rowntree MacKintosh listed as benefits of their preprocessor: reduction of impact of change to a superior DPMS; ease of use for programmer; security enhancement and statistics gathering. SKF listed benefits: implementation of features not yet supported by DBMS; ability to change DBMS without impacting applications; greater data independence; inclusion of extra security and integrity features and ease of physical implementation. Other companies that have produced a DD/DS to work to their own requirements are British Rail, Eastern Airlines, Anderson Clayton Foods and Wells Fargo. These systems are available in general on a non guarantee basis for nominal cost.

SOME ADDITIONAL FUNCTIONS

The implementation of an environment similar to that described above allows the management of an organization to easily expand the DD/DS data structure to provide extra data management facilities as follows:

1. the DD/DS could be expanded to record data about multiple versions of programs and the different data editions that each version works or worked against. Data about differences in programs at different sites and different status of programs at various stages in its testing, design and online phases can be added;
2. information about recovery, security and integrity procedures can be recorded to help ease the DBA workload by automation and enhancement of some currently manual functions. The automation of restructuring and reorganizing can be achieved to some extent;
3. the DBA should be able to define a utility to indicate the impact of proposed data base structure changes and for simpler changes may be able to automatically recompile and repreprocess programs which do not need further modification;
4. a data base load facility could be provided if one does not already exist. Job Control language can be generated as a means of easing the depth of technical understanding of the casual user and for building systems out of smaller building blocks;
5. as the DD/DS can store information about the data processing requirements of programs it could provide facilities to minimize manual operation work involved in system operations and automatically assist with scheduling of batch jobs;
6. the range of statistics needed to manage a large scale data management environment could be accumulated;
7. file control in terms of file generations, backup and physical placement can be further automated;
8. the casual user can be supported by a browsing capability linked to the DD/DS data with terminal dialogue and presearch statistics and costing facilities. Higher level languages for the casual user can be supported by DD/DS information used in mapping between different levels of data independence, path tracing and transparency of derived data;

9. improved interfaces can be implemented making DBMS easier to use, multi DBMS support transparent and perhaps coordinating data management on a number of machines.

DD/DS EXPANSION

The volume, diversity and complexity of large scale data management combined with the need to develop as yet undefined functions, make it necessary that the development of the 'complete' DD/DS will be a long range project. The evolutionary nature of DBMS technology combined with new and varied user requirements further accentuates the fact that the development of the DD/DS will itself be evolutionary. This means that any acquired DD/DS and other facilities available should facilitate the expansion in both depth (degree of detail) and scope (area of data about data) of both the DD/DS structure and supporting utilities. The computer use of the DD/DS as opposed to the human use is in its infancy in today's implementations and the ability of the organization to expand the DD/DS in this area is of vital importance.

If one implements the shell of the DD/DS — DBMS environment to begin with, then additional functions are added by enhancements to the Preprocessor and DD/DS update program, the data structure of the DD/DS being expanded by use of the restructuring capability of the data base definition facility of the DBMS. New versions of the DBMS or even different DBMS are allowed by rewrites or modifications of the DBMS Interface Module.

The environment allows for easy expansion of functions in terms of human input to the DD/DS. Once the Preprocessor and DBMS Interface Module are automated they have the capability to not only use the DD/DS but to record information in it. These modules allow the very use of the environment to automate data capture for use by later developed functions.

CONCLUSIONS

Although a DD/DS oriented exclusively towards the human use mode can be developed by individual users, it is obvious that if the DD/DS is to be helpful in the computer use mode, the DD/DS and the proper interfaces should be defined, implemented and supported by the developers of compilers and DBMS's. The current indications are that the ANSI SPARC report will lend some direction to this goal.

This paper has explored an approach to improving the development and maintenance of information systems in the context of a controlled and managed data environment. When it is recognized by an organization that data is a major organizational resource and hence should be managed as such, a solution emerges. Proper management of the data resource can be achieved via a central data administering function whose responsibility is controlling the specifications and uses of data.

Central control is made possible by the DD/DS which provides a variety of services to a number of different users. In addition, the use of a central DD/DS can be extended in the future to compilers and DBMS for further productivity improvements.

Although the benefits of a DD/DS apply to any data management environment, the combination of a DD/DS and a DBMS makes for a more effective data management environment. While it is possible for a DD/DS to exist by itself without a DBMS, it is beginning to be recognized that a DD/DS is a prerequitite for a successful DBMS installation, a necessity in the case of large scale data management.

ACKNOWLEDGEMENT

The author is indebted to Mr. E.W.W. Miller, Mr. J. Palmer and Mr. R. Johnston for their participation in discussion of ideas presented in this paper and to Mr. N. McQueen for the maintenance of comprehensive material which was heavily used in researching for this paper.

REFERENCES

BRITISH COMPUTER SOCIETY (1976): Data Dictionary Systems Working Party-Draft Interim Report.

CAHILL, J.J. (1970): 'A Dictionary/Directory Method for Building a Common MIS Data Base', Journal of Systems Management, November, 1970.

CANNING, R.G. (1974): 'The Data Dictionary/Directory Function', EDP Analyser, Vol. 12, No. 11, November, 1974.

COLLARD, A.F. (1974): 'A Data Dictionary/Directory', Journal of Systems Management, June, 1974.

CURTICE, R.M. (1974): 'Some Tools for Data Base Development', Datamation, July, 1974.

GOOS, G. and HARTMANIS, J. (eds)(1976): 'Lecture Notes in Computer Science, No. 39, Data Base Systems', Proceedings, 5th Informatik Symposium, Bad Hamburg, Germany, September 24–26, 1975.

INFOTECH STATE OF THE ART — DATA BASE SYSTEMS (1975): 'Data Base Systems: Data Independence'.

KEMMIS, P.F. (1976): 'Data Bases: The Promise of Yester-Year', Proceedings, 7th Australian Computer Conference, Perth, 31 August – 3 September, 1976.

KING, P.J.H. (1969): 'Systems Analysis Documentation: Computer-Aided Data Dictionary Definition', Computer Journal, February 1969.

MARTIN, G.N. (1973): 'Data Dictionary/Directory System', Journal of Systems Management, December 1973.

MARTIN, J. (1976): 'Principles of Data Base Management', Prentice Hall Inc., New Jersey, 1976.

MILLER, E.W.W.M. and ERBACHER, D.R. (1976): 'A Computing Environment for large Scale Integrated Data Management', Proceedings, 7th Australian Computer Conference, Perth, 31 August – 3 Septemebr 1976.

PALMER, I. (1975): 'Data Base Systems — A Practical Reference', Q.E.D. Information Science Inc., Massachusetts 1975.

PLAGMAN, B.K. and ALTSHULER, G.P. (1972): 'A Data Dictionary/Directory System Within the Context of an Integrated Corporate Data Base', AFIPS, Fall Joint Computer Conference, 1972.

UHROWCZIK, P.P. (1973): 'Data Dictionary/Directories', IBM Systems Journal, No. 4, 1973.

10 Information Retrieval Problems in the National Library of Australia

R.A. Simmons

INTRODUCTION

Information is the framework on which civilization is built. The continued functioning and progress of modern society depend on the ready availability of information.

The transfer of information from source material to end user has evolved into a two-stage process. The first stage involves building and controlling a collection of primary material (books, journals, reports, theses, etc.), and providing a mechanism for identifying, locating, and retrieving a specific document from the collection. The second stage involves the creation of a secondary source of information by indexing and abstracting articles in journals, books, etc., and providing a means by which someone can formulate an expression of his information needs and derive a set of references to primary material which may contain the desired information.

Libraries have always fulfilled the first function through traditional operations of selection, acquisition, cataloguing, circulation, and reference services. Collection control is a costly and largely labour-intensive operation, and considerable effort has been expended on applying computer and other advanced technologies to library operations to make them more capital-intensive. This should result in computer-based systems better able to cope with the ever-increasing flood of published material.

While libraries have participated in the second stage in the past, there is a growing tendency for secondary sources to be handled by specialist information centres: that is, places of subject expertise that analyze and synthesize the literature in a particular field. These centres do not necessarily build or manage collections of source material; their primary or only function is to create document surrogates and selectively distribute them to subscribers. More or less as a byproduct to the preparation of printed indexes, many of the organizations generate periodic (usually weekly or monthly) files of bibliographic data which they send out of magnetic tape.

Partly due to the quick growth of specialist indexing services, and partly because the machine-readable data is of secondary importance in automating the printing process, there is considerable overlap in coverage, variation in the depth and quality of indexing, and diversity in the formatting and encoding of the data. However, these files can be acquired and used as input to a data base of document references which users can interrogate.

Although both stages in the information-transfer process can be regarded as information-retrieval applications, there are important differences between them. In the library system, the emphasis is on uniquely identifying a document in the collection. The cataloguing data is extensive, involves a complex structure for the bibliographic record, and usually applies to an entire document

rather than a part (such as an article in a journal). The subject matter is briefly indicated by coarse indicators such as a classification code or one or two broad subject headings. There is normally some indication of precisely where the source document is located.

As already noted, an information centre may or may not maintain a document collection. The document surrogate is heavily biased towards designation of subject content. Particularly in science and technology, the surrogate applies to individual papers or journal articles. It contains just enough detail for the source material to be identified, but gives no indication of the material's location. Speed of dissemination and extensiveness of coverage generally take precedence over bibliographic quality.

For the purposes of this paper, the discussion of information-retrieval systems will be in the context of secondary sources of information; the problems libraries face in matching document requests retrieved through such systems against collections of primary source material are of a quite different nature and will not be addressed.

THE INFORMATION-RETRIEVAL SYSTEM

The CODASYL Development Committee (CODASYL, 1962) defined an information system as one that ' ... deals with objects and events in the real world that are of interest. These real objects and events, called entities, are represented in the system by data. Information about a particular entity is in the form of 'values' which describe quantitatively and/or qualitatively a set of attributes that have significance in the system.'

Summit (1974) produced a more specific definition of an information-retrieval system as ' ... a system capable of retrieving information relating to documents, usually a representation or surrogate of a document (e.g. title or abstract) in response to a question, and the detection of relevant or near-relevant documents (or references to documents) that either answer the questions in themselves or that can be further scanned to determine the answer.'

The terms 'document surrogate', 'bibliographic record', 'bibliographic reference', or 'citation', are used more or less interchangeably to refer to a collection of data elements used to characterize a source document. It normally includes a unique identifier, names of personal or corporate authors, title, publishing source, and information to indicate the subject content.

A diagram showing the data flow and activities performed in an information-retrieval system (IRS) is given in Figure 1. The major components will now be discussed in more detail.

The User/System Interface

The user is the most important part of an IRS, for it is his needs the system is designed to serve. Failure to pay attention to human factors in systems engineering, as is well known, can all too easily turn a technical masterpiece into an ineffective, largely unusable product.

The problem is well described by Ivie (1966):

> 'In many cases a user who comes to an information system cannot state precisely what he wants. He has a very real need for information, but he cannot define exactly what that need is verbally. In other cases a user can accurately specify his interests but changes his mind as to what he wants when he finds that too many or too few articles which satisfy the request.'

The information needs of users fall into two main categories, one active, the other passive. A user may wish to find out all the material published on a particular topic, or to be kept informed as to what is being published in his field of

interest.

The former calls for a 'demand' search across a file of bibliographic data covering an appropriate span of time, usually amounting to several years. The search is usually ad hoc and quite specific.

The latter need is met by a 'current-awareness', or selective dissemination of information (SDI), service. Current publications are scanned and indexed, and matched against a statement of the user's interests known as a 'subject profile'. The user is notified regularly and automatically of new material that meets his profile criteria, from a far wider selection of source documents than he could hope to cover himself.

There are significant differences between the two types of service that affect the design of computer-based systems to support them. However, the two are complementary. A person entering a new subject field needs to be brought up to date on the literature covering that field, and then kept up to date by being regularly alerted to the latest publications that may be relevant.

Retrieval characteristics are quite different. An SDI service might typically match 500 profiles against the latest batch of, say, 5000 references, whereas a demand search matches a single query against a retrospective data base of perhaps a million references.

SDI profiles are relatively stable, mostly requiring deletions and additions to accommodate changes in user participation, and profile 'tuning' to cope with drifts both in user interests and in terminology in the literature. With each SDI run, the batch of bibliographic records is completely different from the previous one. This is the reverse of the demand search, where the query is completely different each time while the data base differs only by the addition of a small amount of new material.

Finally, the user of an SDI service has no way of knowing in advance how much material his profile may retrieve. An upper limit is usually set so that he won't be inundated with references, but he will not know if he has missed anything significant. Demand search, on the other hand, can be tailored through dialogue with the system to retrieve as many citations as the user chooses.

This dialogue is a most important part of the user/system interface. The user must be free to present his problem in stages, refining his query heuristically with the system providing data to help him. People are largely helpless when asked to cope with large, amorphous concepts, and the system must provide a means by which a user can progressively decompose a large, complex concept into a hierarchy of smaller, simpler notions which can be expressed as a query in terms the system can understand (Dijkstra, 1972; Newell and Simon, 1972).

Lefkovitz (1969) described this man-machine dialogue as an interaction '... wherein precise and highly formatted information is provided to the automated system by the user in relatively small amounts, and the system, in turn, responds with statistics relevant to an impending search, or with certain tables or tutorials that will assist the user in the further formulation of his query.'

The user should be allowed to examine pre-retrieval statistics for efficient searching. Most of the cost of an IRS is incurred in the selection and presentation of bibliographic records, so the amount of material retrieved must be kept as small as possible, and it should also be as relevant as possible.

The system should supply data that will enable the user to assess the two most commonly used measures of information retrieval effectiveness: recall, and precision. Recall is the proportion of relevant material actually retrieved;

precision is the proportion of retrieved material actually relevant. One would like to retrieve much of what is relevant while rejecting much of what is extraneous, i.e. high recall *and* high precision. Unfortunately, the two tend to be inversely related.

As well as allowing the user to judge recall and precision, the system should also let the user use these measures to modify his search formulation quickly and easily. If he is to judge the effectiveness of his query, the user should be able to inspect a sample of records retrieved. A speedy response can be obtained if the interaction is conducted through a visual-display unit. Ideally, it should have upper/lower-case capability and programmable function keys to invoke the most commonly selected operations such as 'browse' or 'print'. An attached matrix printer would permit the user to get hard-copy output of particular items displayed on the screen.

When satisfied that his latest prototype is the optimum formulation, the user needs to be able to have the system store the query for possible future use. He should also be able to have the full search conducted either in real time or in batch mode, with retrieved references directed to some local or remote device capable of producing hard-copy output.

This interaction with the system should be easy to learn, and have user aids such as a 'help' function that can be invoked at any stage to provide tutorial material for a user that has got into difficulties.

Query Processing

Many IR systems provide quite extensive software that allows the user to input his query in a restricted form of natural language, and translates it into some normalized form suitable for machine searching.

The first step, query recognition, involves syntactic analysis and semantic interpretation of the query string. The system can initiate a conversation with the user to identify, clarify, or amend parts of the input string.

The next step is to remove unnecessary information punctuation, non-substantive words, etc., leaving the query in the form of author names, words in the title, descriptors, source language, publication date, etc. together with logical and relational connectives, and perhaps a required order of processing.

The reduced query is then normalized by extension and expansion using a thesaurus or other form of dictionary. The search expression is enriched by synonyms, more-general terms, and terms of greater specificity.

At this stage the query has been converted to a standard form that closely matches the characteristics of the data base, and can be input to the search routine.

Data-Base Organizations

The effectiveness and efficiency of the search are tied very closely to the choice of data-base organization. IR systems are currently built around five main types: serial, chained, inverted, scatter-storage, and clustered. The main factors governing the choice are search time and maintenance overhead, and these tend to be inversely related.

At one end of the spectrum is the serial file, in which bibliographic records are stored in no particular order and a complete scan of the file is required for searching. This is the most efficient in terms of storage requirements as only the data is stored. File maintenance is reduced to concatenation of new records. All data items can be examined with equal ease. It is well suited to an SDI service, where the set of user profiles can be matched against each record in turn, and the record placed in an output queue for those profiles whose criteria it meets.

However, the time taken for a complete scan renders this method largely unsuitable for real-time systems, particularly when there are a large number of records or the records are very long.

With a chained organization the data elements with the same attribute are linked by pointers, and a directory used to gain access to the first item. Storage requirements are higher than for serial organization, but maintenance is still relatively simple because new items can be added to the data base and the chains extended to incorporate them. For large data bases the search time can become unacceptable because of the length of the chains and the high number of disk accesses needed to inspect them. A number of variants have been tried (cellular multilist, ring structures, etc.) but have proven suitable only for limited amounts of data.

Following Reiter (1972), an inverted file can be defined as expressing associative structure by grouping records which contain the same value for a particular attribute. This grouping is typically represented by storing, for each attribute value, a list of pointers to the records containing that value. The attribute values and their associated lists are usually ordered alphabetically into an index for each attribute. Such indexes can greatly speed up retrieval when the inverted attributes are explicitly specified in the normalized query. On the other hand, inverting on every attribute value can impose a prohibitive overhead in storage space and data-base maintenance.

Scatter-storage organization is implemented by partitioning the data into groups or records whose keys are related in some mathematical way. The available storage space is divided into lots of smaller units known as 'buckets'. A mathematical computation known as a 'hashing' function transforms the key into the address of the bucket in which the record is stored, and records in the bucket are then searched serially. Special overflow procedures are needed for those occasions when a record is assigned to a bucket that is already full. A good hashing function will distribute records evenly across the buckets.

This method gives quick search times, but can only provide access through a single key. It is particularly suited to storing bibliographic records to be accessed through a unique indentifier selected by the search of indexes in response to a query. Its chief drawback is the difficulty in determining a hashing algorithm that can be applied to a variable-length text string to provide a uniform mapping on to the available storage space.

A clustered organization involves grouping together bibliographic records which have similar contents. Each group, or cluster, is identified by a representative, 'cluster profile' consisting of a list of terms occurring in the cluster, weighted to show the relative importance of the terms.

Searching is first conducted against the set of profiles, and a short-list produced of profiles with a sufficiently high degree of similarity with the query expression. Each chosen profile is examined and the records in the corresponding clusters are ranked in decreasing order of similarity. Bibliographic records are then selected from the appropriate clusters as required. Salton (1972) claimed that this approach is more economical and flexible than the inverted organization.

Summarizing the applicability of the various type of data-base organization, Salton (1975) says, 'For practical purposes the serial file organization cannot be used for many library processes because of the slow response time; the same is true of the chained organizations when controls are not present to limit the chain lengths. Many of the computer-access file arrangements present problems

because of the uneven usage characteristics of the various storage positions. This leaves the inverted and clustered organizations as the main ones capable of furnishing rapid responses for large query and document files.'

In practice, IR systems employ combinations of the various file organizations. The widely-used systems that offer demand searching generally have inverted indexes to a bibliographic-reference file. Search times are kept as short as possible by conducting the bulk of the search against the indexes alone. A number of systems permit serial access to the bibliographic records for terms used so infrequently as to not warrant the creation and maintenance of a separate index. The bibliographic records are usually organized serially or scattered with direct access by unique record identifier.

Search

The function of the search is to present to the user bibliographic references with characteristics that match the query. It is usually conducted in two stages: the normalized query formulation is first matched against indexes, directories, cluster profiles, etc., to retrieve a set of record identifiers; these identifiers are then used as keys to select references from the bibliographic data base.

Set operations are applied to the indexes to provide the pre-retrieval statistics to be displayed to the user. The set of required identifiers is derived from the lists associated with specific attribute values by forming inter-selections, unions, and complements as appropriate to the search formulation. Relational operations are handled by associating a Boolean variable with each operation (e.g. Publication-date. GT.1972 is either *true* or *false*) and assigning the required value as the expression is evaluated.

Data-Base Source

As noted in the introduction, bibliographic data is generated widely and usually with no attempt to standardize format or content. An institution running an IR system rarely has control over the form of data supplied, and does not always create data itself to be used in the system. Herbert Landau (1972) observed that the library community cannot afford the generation of a plethora of machine-readable files in the vague hope that users might eventually fit them into their operations. He pointed out that little had been done to combine data bases from various sources into a unified data base searchable through a single system.

In fact, most operators of large-scale IR systems subscribe to a number of externally-supplied data bases and convert them on receipt to meet the requirements of the system. However, the different codes of practice adopted and varying levels of data resolution create considerable problems.

The problem of creating the source data base of references starts with the descriptive cataloguing that identifies the source document, but becomes acute with the subject cataloguing, or indexing. This consists of analyzing the document and deciding how best to characterize the subject matter when confronted with the fact that the nature of possible enquiries cannot be known in advance.

Indexing has rightly been described as the costliest bottleneck in information retrieval, and considerable experimentation has been done in the application of computers to this task.

A document can be indexed in two different ways. One way is to pick out words or phrases appearing in the text that are judged to be significant, and assign them as descriptors. The other way is to construct descriptors that may or may not appear in the text, using a dictionary, thesaurus, or some other form of controlled vocabulary. If this approach is adopted, it is vital that the dictionary be kept up-to-date by incorporating new terms as they appear in the literature,

and linking them to existing, related terms where appropriate. The significance of a descriptor can be indicated by assigning a weight to it.

The computer can be used with great effect to construct document surrogates. Although a trained indexer can assign descriptors through an intellectual comprehension of the text, reasonable results have been obtained by indexers who do not try to understand the document but select words or phrases whose importance the author has already highlighted by using them in the title, paragraph headings, the abstract, etc. This activity can readily be programmed. Statistical analysis of word frequency, reference to a stop-word list (a 'negative dictionary') to eliminate commonly-occurring but insignificant words, and the use of a machine-readable dictionary or thesaurus, all contribute to the quality of automatic document-indexing.

A simple but very effective indexing technique is citation analysis, a method pioneered by Garfield (1964). When indexing a document, the material cited by the author is recorded. A citation index is formed by listing all the cited references alphabetically by author, each reference being followed by a list of those documents that cited it. Two papers that cite the same article are considered to be bibliographically coupled, and can reasonably be considered to deal with the same subject matter; the more references shared, the greater the degree of coupling and the higher the probability of them being about the same subject.

An extremely effective IR system can be built around citation indexing. A user needs only one reference known to be relevant to his information needs to enter the system and retrieve all the material bibliographically coupled to that reference. This can be used as the basis for both demand searching and SDI.

Citation analysis is being used increasingly for purposes other than information retrieval. For example, following the chain of references backwards from any current article in a subject field can lead a researcher to a fundamental paper in that field. Articles can be counted and ranked in decreasing order of frequency of citation, and there are published lists of some of the world's most-cited papers. Citation frequency can be taken to indicate the relative importance of a paper, and is being used by university administrators to determine cases for promotion and tenure (Wade, 1975).

Summary

Roger Summit (1973) concluded that interactive IR systems could prove cost-effective; that one of the most important features is the report between the user and the data base through interaction; that the ability to consult an online thesaurus and consider pre-retrieval statistics during the search could make all the difference between success and failure; and that the system must let the user structure his search incrementally, starting with simple concepts and proceeding heuristically to more complex formulations.

NATIONAL LIBRARY IR SYSTEMS

The National Library of Australia has been involved in the operation of computer-based IR systems for the past seven years. Imposed restraints on staffing and expenditure have restricted growth in services offered, but some progress has been made. The Library provides demand searching and SDI services in biomedicine, and current-awareness services in the fields of educational research, general science and technology, and the social sciences. Three major systems are in current use, and a fourth has been the subject of a recent feasibility study.

MEDLARS

MEDLARS (Medical Literature Analysis and Retrieval System) is an information service for people involved in health care: doctors, nurses, pharmacists, dentists, veterinarians, etc. It is a computer-based system that analyzes, stores, and retrieves citations of articles from more than 3000 bio-medical journals covering the past ten years. The system originated with the U.S. National Library of Medicine but is now an international biomedical information network with agencies in Australia, Drazil, France, Germany, Japan, Sweden, and the U.K.

The Australian MEDLARS centre is at the National Library, using the computing facilities of the Department of Health. Online, interactive searches can be conducted against the most recent portion of the data base through terminals in the National Library and in large biomedical libraries throughout the country.

At the moment, only 400M bytes of direct-access storage are available for online searching of about 170000 references representing some 6–7 months of the total data base. Back files are searching in batch mode. Additional disk space will be made available next year which will allow three years' data, amounting to about 700000 references to be searched interactively.

The system allows the user to build his query in stages, testing it on inverted indexes and modifying if after examination of pre-retrieval statistics. A controlled vocabulary of medical subject headings (MeSH), supplied and maintained in machine-readable form by the National Library of Medicine, can be inspected online and used to expand and extend the search formulation. The user can have retrieved citations printed either at his terminal or offline.

Requests for articles cited in MEDLARS are handled by the user's local librarian, or through the lending services of the National Library, which guarantees document back-up for all MEDLARS enquiries.

Aus/SDI

In 1973, the National Science Library of Canada (now the Canadian Institute for Scientific and Technological Information) offered free of charge the programs in its CAN/SDI system. The main components of this system are: a program to build and maintain a file of user profiles; a set of programs to convert bibliographic data from a variety of sources to a common format; a program to match the profiles against the data base; and a program to print out retrieved references to be distributed to users.

The National Library of Australia designed and implemented a new program to print references in a more flexible manner and more readable form, and augmented the programs with new data entry and control procedures to permit operation by lay users (Wilson, 1974).

The Australian variant, named Aus/SDI, went into production use on the Biological Abstracts data base in 1974, and the Science Citation Index and Social Science Citation Index data bases in 1975.

The high cost of running the system on a commercial service bureau, together with staffing restrictions, have curtailed growth in the services. Instead of running 3000 SDI profiles under Aus/SDI, the Library can service less than 1000 at the moment.

TEXT-PAC

Text-Pac is a natural-language text retrieval system marketed by IBM and designed for batch-mode operation. It includes programs to build and maintain the data base and generate up to six prescribed indexes; to formulate user

queries; to match the queries against the data base; to print retrieved references; and to provide statistical data by which retrieval effectiveness may be gauged.

The National Library undertook a joint study with IBM during 1971–1973 which established the feasibility of using TEXT-PAC to provide an SDI service to Australian educationalists from the ERIC (Educational Resources Information Centre) data base (McCallum, 1975). An ongoing SDI service has been in operation for the past two years.

STAIRS

During the first half of this year, the Library conducted a second joint feasibility study with IBM which demonstrated the effectiveness of a demand-search service from a retrospective file of ERIC data. The data base, covering 2–3 years of data, was accessed by users through visual-display terminals in Canberra, Melbourne, and Sydney using IBM's software package STAIRS.

STAIRS (Storage and Information Retrieval System) is a general-purpose system for storing and retrieving unformatted text and/or formatted data. It allows searches to be made for combinations of words occurring in particular sentences or paragraphs of text. The 'text' can consist of the actual text of the documents, or can be a document surrogate. Word stems can be looked for, and retrieved records can be ranked according to relevance (IBM, 1974).

DIALOG

After a certain amount of experimentation, the National Library announced last month that it would conduct searches on behalf of Australian users of the 37 or so data bases available under the Lockheed DIALOG service. The data bases cover science, technology, education, social science, business, economics, and finance.

The DIALOG system features commands to inspect a term dictionary, combine terms using Boolean operators to formulate the query, select records from the main file, and display or print selected records.

DIALOG can be accessed in Lockheed's Palo Alto Research Laboratory through the TYMSHARE network. The National Library connects its dial-up typewriter terminals to TYMSHARE using International Subscriber Dialling (ISD) facilities provided by Telecom.

PROBLEM AREAS

The pressing need for improved information services in Australia was spelled out in the STISEC Report (1973). The committee strongly recommended that a national authority be established which, among other things, would ' ... give maximum consideration to the introduction of those services and information handling methods which utilise modern computer and telecommunication techniques.'

Two years later, it was pointed out that the committee members of STISEC had made representations to the Government and the National Library ' ... deploring the slow progress in implementing the STISEC Report, particularly in relation to computer-based services' (National Library of Australia, 1975). This was despite the fact that some progress had been achieved through the initiatives of CSIRO, the Australian Atomic Energy Commission, the Department of Industry and Commerce and the National Library.

Why this lack of progress? The brief answer is a lack of resources. Not just computing power, but all sorts of resources. The best IR system still only provides literature citations, which are of no use to anyone without ready access to the source documents. Researching for a paper on 'Document Retrieval

Systems and Techniques', Summit (1974) commented, ' ... extensive use was made of the Lockheed DIALOG online retrieval system. Although it was relatively easy to identify over 1200 references from searches of the data bases available under this system, several subsequent problems arose in obtaining corresponding source material.' When it began to offer information services from imported data bases, the National Library guaranteed that it would subscribe to all the documents covered by the indexing agency, and would be able to supply a copy of any cited article within 24 hours.

To pursue this policy means that whenever a data base is added to an IR system for use by people in Australia, document backup must be acquired, added to the document collection, and catalogued for ready access. This requires money, people, space, equipment, and time. The Library's resources have been subject to severe constraint during the past few years, and this is why so little progress has been made.

As well as regarding IR systems as only part of the total information-transfer operation, the National Library has limited computer capacity at its disposal and insufficient staff to consider designing and implementing a system tailored to meet its requirements. The big IR systems represent enormous investments in development effort. MEDLARS is reputed to have taken 150–200 man years and several million dollars to get it to its current level of operational excellence. It is highly probable that Lockheed's DIALOG system took about the same amount of effort to implement.

Put simply, only a handful of organizations have the resources to invest in building yet another IR system. For the vast majority of institutions it is probably far more cost-effective to make use of an existing system.

Even if a sophisticated DBMS were available that could permit really fast access to a data base of complex, bibliographic data of the order of 10^{10} or 10^{11} characters, there still remains a great deal of applications programming to be done in the user/system interface. The shortage of skilled programmers in this area, and the general lack of priority that has been accorded library processing until recently, has meant that there are insufficient resources around to undertake the necessary work.

It is extremely doubtful if the National Library will ever assign a high priority to building a tailor-made IR system of its own. It has managed to provide the benefits of computer-based information retrieval to the Australian community by acquiring software developed by others better equipped, and either adapting them to meet local requirements or accepting their shortcomings.

Some people have questioned the wisdom of relying on systems from overseas sources, and believe Australia should try and keep abreast of, if not ahead of, other countries developing IR systems. It is hard to find justification for this attitude when in fact the bulk of published material is produced overseas, and Australia contributes very little in terms of machine-readable bibliographic data to the total world resource. As the costs of data communications continues to fall, and the cost of software development continues to rise, it makes more and more sense for the people of Australia to make use of the big IR systems with wide varieties of data available overseas at very reasonable rates.

CONCLUSION

In summary, the problems the National Library faces are not so much technical as political and economic. A few major overseas centres offer excellent IR services at attractive prices, and the technology is readily available for users to gain

online access to them.

The emergence of flexible DBMSs capable of handling complex data efficiently may well assist those organizations with the need and resources to develop bigger and better IR systems. It will not by itself solve the National Library's data-processing problems.

REFERENCES

CODASYL DEVELOPMENT COMMITTEE. 'An Information Algebra, Phase 1 Report of the Language Structure Group'. CACM, 5, 4 (April 1962): 190–204.

DIJKSTRA, E.W. 'The Humble Programmer'. CACM, 15, 10 (October 1972): 859–866.

GARFIELD, E. 'Science Citation Index — A new Dimension in Indexing'. Science, 144, 3619 (May 1964): 649–654.

IBM. STAIRS/VS. General Information. GH12–5114–2. New York, IBM World Trade, 1974.

IVIE, E.L. Search Procedures Based on Measures of Relatedness Between Documents. Ph.D. Dissertation. Mass., M.I.T., 1966.

LANDAU, H.B. 'The Proliferation of Machine-Readable Data Bases: Current Problems'. Drexel Library Quarterly, 8, 1 (January 1972): 63–69.

LEFKOVITZ, D. File Structures for Online Systems. New York, Spartan Books, 1969.

McCALLUM, I.S. 'Educational Information Service'. International Library Review, 7, 2 (April 1975): 265–269.

NATIONAL LIBRARY OF AUSTRALIA. Development of Resource Sharing Networks. Canberra, National Library of Australia, 1975.

NEWELL, A. and SIMON, H.R. Human Problem Solving. Englewood Cliffs, Prentice-Hall, 1972.

REITER, A. and others. 'Representation and Execution of Searches Over Large Tree-Structured Data Bases'. In: Information Processing 71. Amsterdam, North-Holland, 1972: 460–472.

SALTON, G. 'Dynamic Documents Processing'. CACM, 15, 7 (July 1972): 658–668.

SALTON, G. Dynamic Information and Library Processing. Englewood Cliffs, Prentice-Hall, 1975.

STISEC. The STISEC Report: Report to the Council of the National Library of Australia by the Scientific and Technological Information Services Enquiry Committee. Volume 1, Canberra, National Library of Australia, 1973.

SUMMIT, R.K. 'Evolution and Future of Computer-Assisted Information Retrieval'. In: Forum on Interactive Bibliographic Systems. Edited by Charles T. Meadow. Oak Ridge, Tenn., USAEC, 1973.

SUMMIT, R.K. and FIRSCHEIN, O. 'Document Retrieval Systems and Techniques'. In: Annual Review of Information Science and Technology, Vol. 9. Edited by Carlos A. Cuadra. Washington, ASIS, 1974.

WADE, N. 'Citation Analysis: A New Tool for Science Administrators'. Science, 188 (May, 1975): 429–432.

WILSON, A.J. The CAN/SDI System: Its Development and its Implementation in Australia'. Aust. Special Lib. News, 7 (July 1974): 82–86.

11 Australian Mutual Provident Society—Case Study

K. Unsworth

HISTORY OF AMP

Australian Mutual Provident Society was registered under the Friendly Societies Act in 1848 and later incorporated by private act of Parliament in New South Wales in 1857. The Society opened for business early in 1849 but it was two years before the 100th policy was issued.

Until 1886 all applicants were required to appear personally before the Board, and great care was exercised in selecting and rating risks. The full-time agent was appointed in 1860.

By 1884 Branch offices had been opened in all Australian States and in New Zealand. In 1908 the United Kingdom Office was opened. The Society's New South Wales Branch also controls operations in Papua New Guinea and Fiji. A current organization chart is shown in the following pages.

BUSINESS DEVELOPMENTS

In 1905 the first policies were issued for which a collection service in the home was provided. They were then known as Industrial policies - now Collector policies.

In recent years, the trustees of many superannuation schemes have specified the type of investment to which the premiums for their members should be directed. A separate fund has been established to account for this class of business.

There has also been an increasing demand for individual superannuation policies from persons who wish to take advantage of higher bonus rates declared for policies in respect of which investment income is free of tax. A significant number of individual policies are now issued in this was but, in terms of the statutory provisions, the policyholders' opportunities to deal with these policies are restricted — for example, loans on policies may only be granted in certain circumstances.

It is interesting to note that the first two contracts issued by the Society in 1849 were for temporary insurance. This special class of business has been in demand ever since life insurance was first known several hundreds of years ago — despite what some media writers would have us believe. About 20% of the Society's new Ordinary business last year was one or another form of temporary insurance.

In 1940 the Society's policyholders authorized the Principal Board of Directors to include 'equities' in the Society's investment portfolio; until this time the Society's investment pattern had been limited to a type of 'trustee' security. Because of World War II, action to develop an equities portfolio did not receive great impetus until the 1950's.

Currently it is usual practice to limit a shareholding to not more than 10% of a company's issued capital. Exceptions occur in the Society's subsidiary com-

panies which are either wholly or jointly owned. These companies include interests in the fields of:

General insurance
Short term money markets (official and unofficial)
Pastoral/beef cattle
Information sciences

Significant current features (approximate figures): –

Life Insurance
Annual Income $900 million
Number of policies in force over 3 million
Sums insured in force over $25,000 million
New sums insured 1975 over $5,500 million

Assets

Public Sector —		
government, local authorities	$1200 million	30%
Loans to individuals —		
house, farm, loans on policies	$ 480 million	12%
Loans to companies —		
commercial, debentures, notes	$ 720 million	18%
Equities —		
shares, property	$1450 million	36%
All other	$ 150 million	4%
ASSETS — TOTAL	$4000 million	100%

DATA BASE EXPERIENCE OVERSEAS

Although the first commercially available Data Base Management System, Honeywell's IDS, was introduced more than 10 years ago and the 'CODASYL' proposals are now more than 5 years old, the installation and usage of DBMS is still not as widespread as the popular computing press would have its readers believe. File Management Systems have been in use since the mid 1960's and are often confused with Data Base Management Systems but do not provide the facilities to support complex data structures and relationships available in true DBMS packages.

Statistics released in 1975 indicated that less than 2000 Data Base Management Systems were in operation world-wide, with perhaps an equivalent number under development. Of these 1500+ were in the USA and nearly 200 in the U.K. Currently there are very few operational DBMS in Australia, but several companies are seriously considering, or are currently installing EDP systems involving the use of DBMS.

Why should a company decide to install a DBMS?

Some of the benefits of DBMS are claimed to be:

Program-data independence
The elimination of redundant data
Increased programmer productivity
Improved data integrity
Improved data security
The facility for concurrent access and update of data

These are all very desirable features, but not always achievable.

SELECTION CRITERIA

With the benefits described above a DBMS looks very attractive but it must be remembered that it is simply another piece of System Software to be evaluated

according to:

requirements
costs
benefits

Just as any other addition to the range of computing facilities offered to Users. It is also possible that the use of more than one DBMS may be appropriate within an organization to serve differing requirements.

METHOD OF IMPLEMENTATION — SUCCESS OR FAILURE?

If reasonable rules of selection have been followed the DBMS chosen will be appropriate to requirements, most importantly it will provide the ability to express the data relationships and structures in existence in the 'real world' situation.

It is important that an organization does not fall into the trap of 'inventing' data structures or relationships that do not exist in the real world.

It has been said that it is expensive to install a DBMS. Many organizations have found that it is even more expensive to 'take out' or tune a badly installed DBMS. The most common reason for failure as with many EDP systems, is that the original design concepts are too 'grand'. Instead of installing a simple system which can be refined, there is a tendency to produce a system which is over-designed and to install it 'all in one go' rather than in easily manageable parts. The User is often consulted initially to gather his requirements but soon finds himself out of his depth and hands over to the DBMS 'expert' who builds in features which are not required, through lack of understanding of the initial, and often fairly simple, User requirements.

Organizations that have succeeded with DBMS, as with EDP projects in general, have taken the 'simple' approach and installed their system in easily manageable parts, always ensuring an 'escape route' if the latest addition does not function adequately. It is important that the data and data relationships are defined by the User. The setting up of a data dictionary in the early stages of a development giving a clear definition of each data element, its format, ranges, usage and relationship with other data should be a task shared by User and Systems Analyst. Set theory, hierarchical structures, and even Codd's Relational Model would not then be held in awe by the User but easily understood just as the 'real world' relationships he already knows because he uses them in his every day business. In AMP there probably *is* a relationship between policies in force and the obligatory public sector investments in terms of the 30/20 legislation, but is it a meaningful relationship?

EDP IN AMP

AMPNET Network

AMPNET was originally conceived in 1972 to replace outdated second generation equipment and systems. The intention was to build a system which would serve the Society through the next 10 years at least.

In AMP data is an important corporate resource, because an Insurance Society, in common with financial organizations and some others, is not a producer of goods but of services. The provision of these services relies heavily on access to accurate records, some of which have a very long life and to which many amendments may be applied during their currency. Some of the amendments are as a result of statutory changes which, if not applied correctly and promptly, may result in legal action.

Although computers have been in use in the Society for many years, the facilities offered by technological developments in the early 70's appeared very attractive. These included:

The extension of the On-line enquiry facilities to Branches (some limited facility was available in current systems).
Collection and editing of data at source.
The provision of a transaction driven system (more in line with business systems than the second generation batch systems).
The rationalization of stored data through the use of a DBMS.

The network currently consists of:

A dual processor U1110 computer in Sydney
—64K primary storage
—512K extended storage
— Drums
—4000 million characters of mass storage
PDP 11/40 computers at all branches (two at large branches, NSW, Victoria, NZ).
VDUs in user departments (currently 200+)
Communications via leased or dial lines.

Selection Criteria for AMPNET DBMS

Late in 1972 tenders from all major hardware manufacturers were evaluated. The availability of a DBMS for the Central Site computer was a requirement, as the Society considered that the benefits traditionally claimed for a DBMS were both desirable and achievable. The final choice of the UNIVAC 1110 was, however, based on many other considerations. Having made the decision to purchase the U1110 it was decided that the manufacturer supplied DBMS, DMS110 which closely follows the CODASYL recommendations, met the AMPNET requirements. At the same time it was decided that the UNIVAC transaction processing package was not suitable, and the Society commissioned the building of a special purpose TCS. It was originally intended to use CUDN, but when this proved non viable, the Society also had to commission the building of a communications system.

Original Approach to DBMS Implementation in AMPNET

In 1974 it was apparent that there were problems with the AMPNET DBMS. After an evaluation the initial reaction was 'why use a DBMS at all?' — a good file management system would have met the Society's requirements adequately. Additionally the Society had fallen into the 3 common traps previously mentioned:

poor requirements definition
too little User involvement
attempts at the 'grand design'

Fortunately the development was not too far advanced because the 'Data Base Experts' had spent most of their time, quite profitably, understanding the package DMS 1100 and the modifications required to interface it with the 'home built' TCS; however little progress towards system implementation had been made because the schemas proposed were 'over designed'.

New Approach to DBMS in AMP

The Society was more fortunate than some organizations in that the situation was caught just in time, the design and implementation of the AMPNET Database was not too far advanced, and a change of direction was possible with very little cost.

CURRENT STATUS OF DBMS IMPLEMENTATION IN AMP

New Philosophy

The new philosophy towards the development and implementation of DBMS

within AMPNET can be summarized in one word; simplicity.

This philosophy is reflected not only in the structural design of AMPNET Data Bases but in the phased implementation of those Data Bases and the associated Application Systems. In the case of the Ordinary Life Application, this phased implementation is proposed to be carried out at the Branch level.

AMPNET will thus become a multiple data base installation operating under a single DBMS. Combined with the phased implementation, this approach is seen to offer the following advantages:

Experience gained in the development, implementation and operation of one application can be reflected in the design of future Application Systems.

Problems associated with recovery, re-organization and restructuring a Data Base associated with corruption, destruction or redesign can be limited to a single Application; it will not effect the normal processing of other applications.

An important factor which makes this philosophy possible is that within the AMP business functions, there is very little interaction of data across Application boundaries. For example there is no need for data within the Property Application to be available to the Ordinary Life Application and vice versa.

It should be noted that this current philosophy does not preclude the future combination of all AMPNET Data Bases into a single integrated Data Base should it be considered a desirable requirement on a technical or business grounds.

Size/Scope of AMPNET DBMS

The latest release of the DMS 1100 Software Package used in AMPNET, has a primary memory requirement of 48K words of which 12K words are taken up by the systems buffer available to all concurrent executing DML programs.

Table 1 sets out the size and scope of the existing and proposed AMPNET Data Bases.

For the purposes of this table, a 'word' may be considered capable of storing 4 alphanumeric characters or 10 numeric digits.

The figures contained in Table 1 are:

Actual for production Data Bases;

Estimated following investigation for those in development;

Anticipated for proposed development.

Examples of Schemas (Complex/Simple)

What attributes make a schema complex as opposed to simple?

A complex schema is one in which the facilities provided by the DBMS are combined to an extent that the processing associated with the storage and deletion of records is excessive and the ancillary processes such as recovery and reorganization become not only inefficient but difficult to control. It is normally associated with:

A single record type being specified as participating in a large number of relationships.

Relationships between records which are physically stored in different. Data Base areas being specified as maintained by DBMS linkages.

An excessive (?) number of levels of hierarchy.

Every effort is made within AMPNET to reduce the complexity of the structure of the Data Bases. The Application processing requirements are analyzed reviewed and discussed in an effort to be able to meet them using the simplest structure possible.

TABLE 1
SIZE AND SCOPE OF AMPNET DATA BASES

Application Project	No. of Data Base Areas	No. of Record Types	No. of DMS Set Linkages	Storage in words '000,000's'
In Production				
General Ledger	4	13	6	7
BOES—Victoria Branch	4	3	—	45
DMS Test Data Base	6	10	12	3
In Development				
Ordinary Life—Tasmanian Branch°	16	85	20	14
Property—(Expenditure)	4	6	2	10
Deposit Administration—Stg. 1	5	20	14	3
BOES—N.Z. & N.S.W. Branches	—	—	—	200
Proposed Development				
Ordinary Life—Total	—	—	—	700
Property Income	—	—	—	2
Others (Not yet specified)	—	—	—	400

° A pilot scheme later to be extended to all branches

The above statement does not mean that complex processing requirements will not result in a complex Data Base structure; rather that it need not necessarily do so. Perhaps the opening question should be re-phrased to read, 'What makes a schema unnecessarily complex?' with the response:

'An unnecessarily complex schema is one which makes use of more facilities of DBMS than is necessary to meet the 'real world' requirements which it addresses'.

ROLE OF DATA ADMINISTRATION

The role of the Data Administrator has been the topic of many papers over the past few years. These papers paint him as everything from a janitor, to a czar, to a joke (something out of a comic book). His place within the organization is shown in a range from a Vice-President with a great deal of authority, to a clerk in charge of a (albeit expensive) filing system.

Intelligent appreciation of such papers will show that the decision as to what the role of the Data Administrator should be, and his position within the organization structure, is something which must be based upon the requirements and circumstances of each individual organization.

Within AMPNET:

Currently the role is mainly one of consultancy; to applications under development and in production. Organizationally the DBA is a member of the Corporate Advisory Group.

An outline of the Terms of Reference for the AMPNET Data Administrator are shown below.

Although functions itemized below are traditionally associated with Data Administration it should be noted that the role is one of co-ordination and control. A single individual can accept responsibility for such functions only when given sufficient authority, and access to resources as and when required.

The functions considered to be within the terms of reference of the AMPNET Data Administrator are:

Data Base Design

Advise on Schema design, the use of the Data Description Language (DDL) for Schema Definition, and the use of the Data Manipulation Language (DML) for access and update of the Data Base. Consultancy on the use of Recovery and Integrity features — the use of CALC versus VIA SET etc.

.Consulting to Application

Assistance in the early stages of feasibility and design and throughout development and implementation.

.Data Base Standards

The issue of codes of good practice and guidelines. Mandatory standards — methods and usage of DMS1100 options, to ensure that a consistent basic approach exists AMPNET wide to the use of DMS1100.

.Data Documentation (Data Dictionary)

Currently available data dictionary and data directory systems are being evaluated; hopefully a system will not have to be produced specifically for AMPNET.

.Data Base Documentation (Design Notes, etc.)

Notes on usage, programmers notes, design notes, tips etc.

.Data Base Tuning

A performance analyzer is already in use, statistics gained will provide assistance in more efficient use of mass storage, reduction in accesses, optimum distribution of data etc.

.Data Base Utilities/Audits

Currently available packages will be evaluated and recommendations made with regard to use — if required specific utilities will be produced.

.DMS Education

Provision of educative material from 'overview' feature descriptions, to detailed programmer tutorials, in written and oral form as appropriate.

.DMS Software Evaluation

Evaluation of new features of DMS1100 current features and utilities.

.Data Base Recovery

Standards and guidelines will be provided to ensure that the required level of integrity is provided for the AMPNET Data Base in a cost effective manner.

.Data Base Re-orgqnization

Assistance in planning Data Base re-organization in an efficient manner which will hopefull be transparent to users — following tuning or perhaps the introduction of a new application.

To fulfil his role, the Data Administrator must have a high level of interaction with all other areas within the AMPNET organization.

In addition, a close association must be established with other UNIVAC (and

especially DMS) sites, both within Australia and overseas.

It should be noted that the use of Data Base Management System 'techniques' should not be considered the province of 'DMS Experts'. The design of Schemas should not be beyond the ability of any good systems designer; DML and DDL should be a skill acquired by any high level language programmer. It is recognized, however, that within AMPNET there may be a learning curve especially for newly introduced staff, and certainly the efficient use of DMS techniques will grow with experience.

TOOLS FOR DBMS SUPPORT

It appears to be a common belief that, given a DBMS, the Data Administrator should be able to perform all the activities normally associated with his role, without involving the installation in the expenditure of resources in the development of 'Tools' to assist him to meet his responsibilities. Unless an installation selects a DBMS with manufacturer or vendor supplied tools, these have to be developed. The latter is usually the case.

The nature of these 'tools' which are in fact utility type programs, are as follows:

System Simulation
Performance Evaluation
Audit Routines
Statistics Gathering
Integrity Controls
Re-organization Routines
Restructure Routines
Data Base Patch
Data Base Dump
Data Base Print
Initial Load Procedures
Audit Train (Log) Analysis
Relationship Verification
Data Base Compaction

12 Vendor's Forum

A transcription of the panel discussion held during the seminar.

Panelists

Name	Affiliation	Abbreviation
Vic Davies	ICL	VD
Bill Fulton	DEA	BF
Clark Gason	Honeywell	CG
Harold Korte	Auditor-General's office	HK
Peter Kemmis	IBM	PK
Ann Leach	AMP Society	AL
Larry Langman	CINCOM	LL
Gene Ritzman	UNIVAC	GR

Chairman

E.W.W. Miller — Australian Bureau of Statistics

Q: To what extent do manufacturers include phonetic name searching techniques in their DBMS products?

PK: Do have an alpha-search capability linked in with IMS. Apart from this, I feel that manufacturers, have been rigid in their products, whereas the users have required more flexible techniques, such as phonetic searching.

CG: A technique that is used is the *synonym* search.

Q: But synonyms are not the same as phonetic searches.

<there was no more discussion on this question>

Q: The papers during day have been rather academic and unrelated to the needs of the DBMS user. This is illustrated by an anecdote. A man went into a tailor's for a suit, and the tailor put one on him. The tailor said that it was a good fit except for the need for a little 'taking-in' under the arms, a bit of lengthening in the legs and so on. As the man was walking out of the shop, another man passed him. He said 'Incredible. If this tailor can fit a deformed chap like that, he's for me'.

In a similar vein, the user today is being forced to fit the suit provided by manufacturers. The question is, what is being done by manufacturers today to make their systems *usable* and easy to use?

AL: One of the most important things when you are looking at DBMS packages is 'what are your requirements?'.

Too many people think that, like programming language systems in the late fifties and operating systems in the sixties, data base management systems will be the answer to all their problems. However, they won't. They are just another piece of systems software that someone is trying to sell *today*. If this software

solves your problems — well good. But if you have to 'deform yourself' to fit the product, then *don't* use it.

Now, looking at the products of the vendors here today, although they ostensibly support the same type of data base system, they are in fact all different. We are happy with DMS1100, but it is not a *true* CODASYL implementation, it hasn't got all the features of the CODASYL proposals, but it is adequate for our requirements. In summary, be specific about your requirements and do not twist them round to fit available software.

HK: I would agree with the previous speaker. From an audit point of view, data base management systems lack a lot of features that the auditor would like to see, but I wouldn't necessary blame manufacturers/software houses for not including these features, because I believe that the auditing/accounting professions have been slack in specifying what they require from data base management systems. I expect this is because they believe that ADP is just a passing phase and will go away anyway!

VD: I think that the speaker is in a bit of a hurry. We have been developing data base management systems for 10-15 years at the most, and during that period we have gradually raised the level of interface between the software and user, and I think that process will continue. However, it will be some time before we actually get to the level implied by the question.

CG: There is a lot of confusion between user facilities and models. For example, consider Codd's relational model. It is often assumed that a system based on that model demands that the user level facilities must also conform to the model. Also, the CODASYL reports referred to languages and metalanguages that would develop from the CODASYL model. Languages have been evolved for the relational model with the kind of syntax that we saw in the paper on the relational model, and there is this tendency to confuse the *model* and the *user language*.

PK: I would tend to disagree with Vic Davis in his optimism in the progress towards satisfying the needs of users. Certainly we do see the growth of query languages, but I think that the point that the questioner raised is quite valid. The users have not been seeing the sort of results from computing that they had been told, 10-15 years ago, would occur. 5 years ago, they were told that database would herald a new era. I am not really satisfied that we have made significant steps in making things very much easier for the user. In fact this is supported by the tremendous growth in the minicomputer market. Users are buying minis since, in this way, they can control their own computing.

BF: There is quite a significant trend towards the distributed database approach, and integration of the worlds of communication and of database. The application program is not aware of the system which actually holds the data required by the application. This is very much in its infancy, but there are indications that this approach is what many users may be wanting. The smaller machines are going to play an ever growing part in data base management systems in the coming years.

<end of discussion on this question>

Q: How many of the vendors produce their products as a kit of parts. Like many system generation procedures, the user can build a system specifically suited for his needs.

PK: For some reason the gentleman doesn't want a large generalized system!

VD: What we can offer is some software available on a range of systems, with a subset that is runable on the whole range of machines, being compatible all the

way up.

PK: An analogy would be the dials on an aeroplane, and the technology that makes them possible. However these are understandable only to experts. Now a data management system that does not have a complicated generation procedure may, in fact, be a better product. As to inclusion and exclusion of certain facilities, the strong trend towards the virtual memory environment, may reduce the need for such trimming and generation complexities, in that if you never use the feature, it never comes into memory anyway.

LL: Being the only representative of an independent software house, I can say in answer, that the approach taken by the house has been that of the 'one database system' philosophy, translated across every machine type available. Currently fourteen different machines are supported. Modularity is essential in such an approach. The system produced by us ranges from about 8k–20k bytes. The features under user control are checkpointing, logging, statistics and so on, which are modular and can be added or omitted as needed. Apart from these options, it would be very difficult to subset DDL and DML options.

<end of discussion of the question>

Q: I would like comments on the overlap of functions between operating systems and data base management systems particularly in the case of IBM situation. And the views of the panelists on the use of a dedicated processor for DBMS.

PK: Yes there is overlap. It is overlap for historical reasons. It is the same overlap you see when a user buys hardware to handle a mainframe site and hardware to handle a network; its the same kind of overlap you find in systems design in double or treble queuing; I put it down to insufficient far sightedness, to an insufficient understanding of directions in which developments will occur, sheer business pressures that do exist where there is a market for one particular thing and not such a market for an overall concept.

And now to the Data Base Machine idea. I think this is only a technical issue, and an awful lot of research into such an idea would be required. A purely personal feeling is that we could well see something like it in the near future. However, I think that much greater problems are those hinted at in papers given earlier today. And those are the problems of understanding what we are trying to solve. Technical problems pale in comparison.

VD: A comment on the dedicated processor. In the 2900 architecture, the data base system exists as a separate virtual machine, which corresponds to the concept in the question. The only difference is that it avoids all the network problems created by the use of a separate processor. In essence it is treated like a separate database machine, but is running under the same processor as the other virtual machines on the system.

Q: Don't you get unnecessary overheads?

VD: What do you mean by unnecessary overheads?

Q: Well, with one processor servicing all the virtual machines, it is in fact *not* an independent process as far as resource utilization is concerned.

PK: There are existing systems today constructed from cooperating processors, and work is farmed out to them by a central control processor. But you can't treat a database management system as an overhead, since it is providing a service that you would not otherwise have had.

GR: One big problem we see at UNIVAC is one of price. When you consider the cost in implementing such a machine it may not be worth it. We put well over a hundred man years into the development of DMS1100, and if you add

the development costs of a back end machine system, it becomes very expensive, plus — what would be the market for it? We don't see that much of a market for it currently. Perhaps in the future, 5-10 years, there may be such a market.

VD: If your database management system takes 10-20% of your processing power, then it is cheaper to add to the power of the processor than to go to a separate one.

<end of discussion>

Q: During the introduction today by Dr. Smith, we heard a comment that there is a need to increase the speed of operational database systems, and that he saw this coming from some form of parallel processing. I wonder if the concepts of associative processors and content addressible memories are also relevant here?

VD: Yes. We have a thing called CAS which is a hardware development.

AL: There is one on the market, used mainly for military systems in the United States, called STARAN by Goodyear. That's the only one I know of except that ICL is working on something.

<end of discussion>

Q: Do systems that require data to be of many varying types and lengths, lend themselves to data base management systems as provided by manufacturers.

PK: Your current MEDIBANK record is a variable length record; but I think that what you are getting at is the issue of — not knowing at the beginning how large records are going to be, or not knowing the characteristics of the data. Is that what you mean?

Q: I'm talking of data bases in which records may be of the order of thousands of bytes, and there may be billions of such records.

PK: Sheer volume is no real problem except, of course, in the resources needed to support it. Searching such a database is a real problem.

VD: There are two aspects to the problem of variability in data content of records. If one takes the CODASYL philosophy, you have two options open to you. One, is the specification of a sufficiently large data group that may repeat a large number of times. The other method is the SET mechanism, enabling you to encompass a high degree of variability. If you are talking about variability at a character level, then you need the former technique. With both mechanisms available there is no reason at all why such a database as you describe cannot be supported.

CG: If I understand the questioner correctly, the problem is mainly one of information retrieval, based on today's data base technology. My feeling is that the technology is not up to the job, and that the information retrieval and data base technologies are different. They will not merge into one for some time.

PK: We are coming to the time when we will see implementations of the conceptual schema given in the ANSI-SPARC proposals, and the problem that is faced there is that of bringing it down to an implementation level.

VD: We have been mainly talking about the CODASYL approach. We musn't forget the full text information retrieval system which is another weapon in the armoury. In many applications the optimum choice of system is a balance between both types of approach. For example, earlier we heard a paper on a land-usage database, and we found that the CODASYL type of model was useful only when combined with another tool uniquely designed for the problem in hand.

GR: UNIVAC DMS1100 allows variable length records and sets including variable length records. But our thrust has been to give the basic structures to

enable applications to be developed, using those structures, but also providing tools for special applications. For example in text file searching, UNIVAC has a product (UNIDAS) which is an application based on DMS1100.

<end of discussion on question>

Q: Being is the Public Service Board I would like to ask — 'what are the most important trends in DBMS?' I would first of all like to make a comment. I believe that for the first time we are in a position, through the new technology, where we can locate separately, the location of decision making and the location of information processing. This is going to lead to fundamental changes in management information systems; consequently the future is with distributed information systems. I wonder to what extent the manufacturers whose equipment will make this possible, dispute this.

VD: What you are really saying to us is that you think that distributed data systems are going to be a thing of the future. There are a lot of problems that are yet to be solved in the distributed database environment, as we heard earlier in the paper on distributed databases. One of the big problems in how you control the concurrency of operations in distributed systems. It is going to be a long time before we can provide off the shelf software in this area. The first approach is to come up with an internationally agreed standard, by which you control data bases. Such a standard would provide a basis for growth into such areas as distributed databases. Until there is such a standard, until more research is put into distributed databases, it is going to be virtually impossible to provide the software and support for such systems.

LL: We have, so far, installed four distributed database systems. Each was on a one-off basis. Each had unique requirements.

The sort of problems that were met indicated that there is a long way to go in this area.

In fact there are indications that what is general across distributed database systems is far less than what is unique to the system under consideration. For example, in the United States, a system can be based on packet switching and be charged by the packet, thus being on line 24-hours, whereas in Australia, systems cannot be based on packet switching since there is no support for such communication links. These differences strongly determine the kind of distributed system that can be implemented. In distributed data base systems, you have database, coexisting with a data communications problem, and I do not believe that such combined systems will be available as viable products for a long time.

Current Systems can be used to emulate distributed systems, and the problem is to decide when it becomes cost effective to replace emulation with a new system. The interfaces between software, hardware and communications devices are all non-standard, adding significantly to the problems in this area.

Q: Perhaps the type of environment I was considering was rather more simple than so far discussed. Perhaps 'linked' databases which the user sees as a complete entity is a more accurate description of the type of system I have in mind. The database as a unified entity,mapped onto a communications system. The system in mind wouldn't hit the concurrent update problems discussed earlier, or the need for large amounts of data traffic, especially whole files.

AL: I think it is a very interesting question and I would like to get at the manufacturers a little bit and say that, they are to blame for the way things are at the moment.

At the end of the sixties and early seventies, *integrated* databases were the

thing. The analogy given to me was 'You wouldn't want to run your business with regional offices all over the place, would you?' — things are much easier when you have got everybody in one building.

However, the business I am in, *is* regionalized to a high degree, and putting everything on a centralized integrated database presents horrendous problems. Manufacturers have sold one computer and focussed their database system round it. Although AMPNET is an integrated system, it is not as integrated as the manufacturers first thought. Also the business is very non centralized. We have branches in Brisbane, and in Wellington and the data in one has no effect on the data in the other. And yet they have to send it all to Sydney, and then its all sent back again. Its a waste of money.

The pressure has got to come from the users, and manufacturers must respond.

<end of discussion on the question>

Q: What are you really doing for the user?

CG: We now have a much more developed contact with the user than we used to have, and this brings me onto my own hobby horse. I think that if you can start to discuss problems with the user and keep him understanding what it is you're talking about, then we can come out of the dark caves in which we have been operating for years. My own people are trained so that they must continue to operate in such a way that the user can understand what they are talking about.

In my view we don't design databases, we recognize what is required by the user, draw them and model them for the user.

LL: The message I am getting is that, currently, users are having a hard time. What do you mean by a 'hard-time'?

Q: I am more concerned with support for the casual user, the general population. In the very near future we'll see data base systems as information system utilities demanding a casual user interface. We are approaching the problem from the hardware point of view with the evolution of intelligent terminals, but not from the software point of view as fast as we should be.

There are a number of query languages available but what is happening in manufacturers 'workrooms' that can be thought of as progress towards support for the general or casual user, especially in terms of English-language queries.

VD: We are getting things out of perspective. The idea of a casual user is valid, but he constitutes a very small part of the total class of users. In the vast majority of data base environments, of the order of 80% of contact with the system is *not* at the casual level. Most users are not casual, but interact with the system in more formal and standardized ways. When you talk about the casual user you are mainly talking about the full text information storage and retrieval systems. They exist today and you can talk to them in English-like languages. It may be slightly stylised, but in essence it is English.

Styles can be changed to suit different kinds of casual user (e.g. a lawyer or an M.P.). In fact the user can style and structure the language to suit his needs and to look even more like english if required.

As a problem it is recognized by the manufacturers. They have done somethings towards its solution, and they will continue to do more.

I cannot accept that users are really hard done by.

GR: A point I would like to make is — the reason why we make software. We make software to sell machines. The software that we make, we make for a market. The user, you, form the market, so its really the user that determines

what should be implemented. In fact that is why we have manufactured many parts of our data base management system, and why it was started originally. The reason was that it was thought the user would buy systems because of such software, and this has turned out to be the case.

It is up to the user to tell manufacturers what they need, and this is done by buying software and buying machines.

PK: I rather support that view. I think that users are as naive as the suppliers of hardware and software. We should be seeing a more critical approach by users about what they expect from data processing. I would expect that first from the data processing professionals amongst users. I would expect them to take a change of stance from what they adopted during the sixties. Which was to go along to, say, clerical people and say 'we can do such and such for you', but you tell us what you really want.

I think it is time that we saw a much greater sophistication, whereby we went out and said — 'We can present such and such for you; if it is anticipated that this could be required then we can do it'. Then we can enter into the details of the form in which it is provided. I would, in answering your question, toss a lot of the responsibility back to the data processing professionals themselves. They are as guilty as the suppliers.

LL: The poor, blighted user, isn't so poor and isn't so blighted. Who are we really talking about? If you are talking about the one-off report, then it isn't the data professional that needs it, its other levels of management. Why haven't these other levels of management not availed themselves of the opportunities to learn.

Or, given that data professionals have realized that the management requirement for reportage is going to come, why haven't they made sure that management be educated and informed.

It is very rare that manufacturers or suppliers actually get to the real end user. There is a region in between, occupied by the data processing professionals. They are professional for one reason. They know D.P. The reason they are part of an organization is that they are supposed to know the organization's requirements. Somewhere along the line, that whole communication is breaking down. There are possibly many reasons. However, the nett result is that manufacturers and suppliers cannot accurately find out the real needs of the users of their software.

Nearly always, the point of contact is the D.P. professional and not the actual user. The user is remaining ignorant because D.P. professionals are failing to play their part in the education and communication process. It is a problem and needs repair. If the D.P. profession doesn't solve it, the other levels of management will.

<end of discussion on this question>

Q: Would any of the vendors care to comment on facilities required and provided for simulation and emulation, for improvement and measurement of performance of existing data base management systems and for the design of such systems, entailing the prediction of performance.

GR: On DMS1100 utilities are given to the user. It is probably more applicable to let the user decide which utilities are the most useful for his applications. For example, there is a utility to search the database and print out statistics on the records that are on the data base, or the number of accesses to those records, and statistical routines to give information on the functioning of the on line file handler, giving overhead measurements. The operating system also provides

statistics of this kind. Most users of DMS1100 have used these utilities to a great extent. They do not report on response times. But there are facilities for getting response times.

CG: The only method we have found successful in the design process has been the use of a pilot study, and numerical simulation.

VD: We have a simulator that has just been released so that we have very limited experience of it.

Basically it attempts to predict the distribution of records and the flow of data. It models the activity of the data base. The basis for the use of it is: – if the distribution of data falls within the range you require, then the corresponding simulated performance will be of the order of what would occur in practice. Some of the results that have been obtained have been surprising, and we have had to modify schema designs on the basis of its predictions.

LL: It is very difficult to predict what is going to be the situation in any real production environment. We have no tools that allow us to predict what will happen when parameters are changed in systems. What can be done, is to look at the problem, and assuming that its a simple one, then an approximation can be found by calculation. I am impressed that there is a simulation tool available.

VD: The simulation is specific to the system produced by ICL. It is not a general purpose tool.

Q: We are very much at the stage where would welcome any tool for the simulation of data base systems.

VD: One of the major problems in a CODASYL system, is that when you design a schema, you look at it statically. What you cannot see are the different kinds of problems that can arise at run time due to different types and mixes of data.

The classic example of this is that which arises when you are *loading* a system as opposed to *running* a system. Often two different schemas are required. There is a great disparity between relational requirements during the initial load and those for normal daily running. You do in fact need a simulator to aid you in schema design.

CG: The most effective simulation is a pilot study.

VD: Yes, but you are putting a lot of money into a pilot study. A simulation may only take twenty minutes.

CG: It is certainly desirable to simulate as a preliminary step, but the sheer complexity of a DBMS and the running data base, must make it desirable to set up a pilot study.

AL: It certainly is possible to simulate but using the simulations based on the available systems means that you have to simulate for DMS1100, IMS and so on. This can become very expensive. We should be given a system simulator for use before systems are developed or bought and as a tuner for existing systems. This is one of the things we hope to develop at AMP. Manufacturers have been lax in providing such tools.

<end of discussion on this question>

Q: Value judgements have been made on the use of DMS1100 in AMPNET, but would AMPNET be possible at all without it? Do the manufacturers know, without mentioning names, of disastrous applications of their software?

AL: We certainly could have produced a system without a manufacturer's data base management system. We are using DMS1100 and it does the job for us, but there is no doubt that we could have produced our own system, without such overheads. On the other hand, DMS1100 has given us some good features, and at the moment we are so far down the track that we cannot retreat. Given

our time over again, we would look into systems more carefully and perhaps have chosen a file management system rather than a data base management system.

Commenting on the second point: — There has been an awful lot of failures. In England, I was chairman of one of the users' societies working groups for four years. This was a self help group with well over a hundred members. There were many involved in implementing data base. Some had tried, or had given up, and the numbers were many.

If you think it is expensive to put a data base management system in, just think how expensive it is to take it out. It is very, very expensive.

PK: What we see is a whole series of failures to one degree or another, or if you like — successes to one degree or another.

I think that the reasons for the large difference between what was hoped for and what was actually achieved can be put down to a number of factors: One of the reasons can be very simply that people have not understood enough of the complexity of a particular piece of software.

Another reason might be because of inadequacies in the design of the software. Or the failure of code in the software.

The more important reason, is that with the introduction of a data base system into an installation, there is the presupposition that the way applications are going to run is altered significantly.

Any user who takes on a database system on the assumption that it is going to come up, work perfectly, and all applications will run perfectly, as absolutely naive. He has got to put himself in a position, whereby, whatever happens he can recover. That whatever happens, he has a controlled environment.

I think that if you could dissect these failures (or unsuccesses), you would often find that they are associated with an uncontrolled and rather naive data processing installation.

I am trying to be as objective as possible and not in any way absolving myself or the organization for which I work, from blame in these failures, but what I am saying is that any organization that is going to move to a far more complex data base system, and an associated communications network, say, has to have a much better control over the way it does things, than it has generally had up to that stage.

Often, before, it had batched processing where each application was run alone, and people have been able to manage that kind of system. If a failure occurs then you can go back to the father or grandfather. This is not possible in a data base environment.

As suppliers we haven't understood the magnitude of that sort of problem.

VD: Have users considered the problems of TP (Transaction Processing) systems from the point of view of success and failure. Today, people move into a TP environment with little difficulty. Looking back seven years, TP was fraught with failures just as data base management systems are today. There was a lot of inexpertise — but gradually lessons were learned, and now people tend not to fail in their implementation of TP systems.

In database systems we are entering the phase where people should not fail, since we have quite a long portion of the learning curve behind us. The days of process oriented file design are over. Let us forget them. Data bases are the only natural way to go.

LL: I would agree strongly with what has just been said. The database management software today tends to be well designed and to work.

The errors seem to lie with professionals not realizing what the software package that he has can or cannot do, and what it does most effectively, what are its weaknesses and so on. They do not *understand* the systems which they purchase. In recognition of this, they must build into their systems controls, to ensure recoverability and data protection. The data base administrator is responsible for these, since he effectively controls the security and integrity of the data resources of the organization that employs him. Very few people realize the full scope of this responsibility. Hence you get data base put in, no one quite knows what it will do, and they get hit by problems they don't understand. They continue development based on the system without proper controls or proper documentation, but they get more committed and dependent on it. There is no turning back, so that when things do go wrong, the failure becomes a *disaster*.

Q: The last speaker but one said that 'data base is the way to go', but the last speaker said that we must look at database very closely and in many applications it is not the way to go.

Can the various members of the panel come up with a formula when you should go database. Is it when you've got three files? or what?

VD: I do not see any value in implementing any systems today on the basis of process oriented files. We have completely inverted the design process. We used to design a system on the principle of 'what do you want out?', finding out what had to go in, and hanging a series of files about the process that handled the data.

Today we are saying that data is very expensive. Today we also have devices that can store it with a reasonable degree of efficiency. We treat data as a primary resource, and therefore all designs are made on the basis of the data. It is irrelevant how big the data system is, it is the data that is always important. It has the greatest degree of continued existence on the system. Processes come and go. Needs for information change from month to month or year to year and new processes are required, but the data remains reasonably static, and therefore you should concentrate your design at the data level.

AL: We were talking, however about data base management systems, not the data. I agree that the data is all important. But AMP don't only have one data management system. They have two to suit two different requirements. One is for a simple system and one is for a complex system, and I thought the question was about choice of system and not the data itself.

VD: I would like to make a distinction to clear up my point. What I was talking about was data base design, not to be confused with the complex file handler. We make a lot of play on the software. However, the software is only the bottom level of the design process. The process of data base design goes well beyond this level.

One of the primary reasons for failures is that people have not recognized this.

CG: When we develop a system today we produce a data structure diagram. We may not implement this in a data management system, but this would be the exception.

The data structure diagram comes first in our design process.

Q: Two things seem quite clear. Manufacturers often sell machinery on the basis of their data base management software and this software is very similar from manufacturer to manufacturer apart from minor differences.

How does the user, therefore, decide between them in the absence of

simulators.

LL: I disagree with the questioner.

Because the user has a data base requirement, he does not have to link the software and data model with the machine on which it runs. They should be independent. My company is in business to ensure that such a separation exists.

DBMS systems are not the same. There are great and significant differences between them.

There are differences in the range of files, security and backup capabilities and so on.

BF: The considerations that are important are not really those of performance, but there are other factors such as functionality and reliability, and also the utilities that come with the package.

It requires a comprehensive and lengthly study to come up with the best solution.

CG: If it were my money, I would buy a company's track record and its people.

PK: I would not buy data base software purely on the basis of performance.

You should sort out how you are visualizing your data, how you want to organize it, and try and get hold of a piece of software that is not going to force you to deviate significantly from the way you want to go.

Chairman *In summary:*

During Jeremy Firth's paper this morning I was reminded that there are three views of software packages: —

One is what the user thinks he wants from it; another is what he needs and the third one is what he actually gets. They are all different. There are three groups of people deeply involved in this DBMS situation; the vendors being one, the data processing professionals being another, and the real users.

I am very concerned, and have been during the symposium, to see for myself if users really are indeed the people who can stand up and give learned dissertations on relational data models, distributed databases or whatever. Or are they simple people who want something straight forward for their organization and are being befuddled by the hypothesis peddlars in the data processing world.

I am never too sure whether vendors' representatives are smart operators who nurture the megalomania of data processing professionals or whether they are 'good guys'.

I will finish by picking up something that Ann Leach said, that is 'users ought to get together and try to tell vendors, what they really want'. — Perhaps they should be called 'Data Base Anonymous'. Perhaps it really is time for the data processing professionals and users organizations to first of all find out what their own users really want and then try and translate these into terms that the vendors will understand and can respond to. In this way we can get products we really want, instead of having to make do with products that only do half the job or don't do the job at all.

AUTHOR AFFILIATIONS

Dr. B.G. Cook	Division of Land Use Research, CSIRO, Canberra, ACT
Mr. P.N. Creasy	Department of Computer Science, ANU, Canberra, ACT
Mr. D.R. Erbacher	Australian Bureau of Statitics, Cameron Offices, Belconnen, ACT
Mr. J. Firth	Informantion Sciences, Canberra College of Advanced Education
Dr. I.T. Hawryszkiewcz	Information Sciences, Canberra College of Advanced Education
Mr. H. McKenzie	Division of Computer Research, CSIRO, Canberra, ACT
Mr. R.A. Simmons	National Library of Australia, Parkes, ACT
Dr. J.L. Smith	Division of Computer Research, CSIRO Canberra, ACT
Mr. K. Unsworth	Computer Sciences of Australia, St. Leonards, N.S.W.
Dr. G.L. Wolfendale	Computer Centre, Australian National University, Canberra, ACT