N75-21035

SET PROCESSING IN A NETWORK ENVIRONMENT

W. T. Hardgrave

Langley Research Center
Hampton, Virginia

March 1975

ICASE REPORT

SET PROCESSING IN A NETWORK ENVIRONMENT

W. T. Hardgrave

Report Number 75-7

March 31, 1975

INSTITUTE FOR COMPUTER APPLICATIONS

IN SCIENCE AND ENGINEERING

Operated by the

UNIVERSITIES SPACE RESEARCH ASSOCIATION

at

NASA'S LANGLEY RESEARCH CENTER

Hampton, Virginia

SET PROCESSING IN A NETWORK ENVIRONMENT

W. T. Hardgrave

ABSTRACT

The combination of a local network, a mass storage system (MSS), and an autonomous set processor serving as a data/storage management machine provides an environment that offers potential advantages to both the user community and the central computing facility of a multi-mainframe computing installation. Potential advantages for the users include:

1. Content-accessible data bases usable from all connected devices.

2. Efficient storage/access of large data bases.

3. Simple and direct programming with data manipulation and storage management handled by the set processor.

4. Simple data base design and entry from source representation to set processor representation with no predefinition necessary.

5. Capability available for user sort/order specification.

Potential advantages for the central computing facility include:

1. Significant reduction in tape/disk pack storage and mounts.

2. Flexible environment that allows upgrading hardware/software configuration without causing major interruptions in service.

3. Minimal traffic on data communications network.

4. Improved central memory usage on large processors.

This paper was a result of work performed under NASA Grants NGR 47-102-001 and NSG 1668 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, Virginia 23665.

1.0 Introduction

Combining the concepts of a local data network, a mass storage system (MSS), an autonomous data base computer and set processing into an integrated system provides an environment that offers potential advantages to both the user population and the central computing facility of a multi-mainframe computing installation. The local network provides for communication among all modules connected to the network. This insures a large measure of flexibility for growth and modification of the configuration. The mass storage system (e.g. Ampex Terabit Memory System [29, 39] or IBM 3850 Honeycomb Store [32], provides a medium for the on-line storage of (multiple) large data bases (in excess of 10^{12} bits) accessible to the network. The autonomous data base computer serves as an interface between the network and the mass storage system. Furthermore, this machine could be cognizant of logical data relationships and would be capable of managing the users' data bases and the mass storage more effectively. Set processing provides a sound theoretical basis for constructing a data base/storage management computer. Set theory is comprehensive in its ability to describe logical data relationships and storage configurations making it a natural choice as a basis for a data base computer. Furthermore, a relatively small collection of operations may be defined providing the user with an effective interface with which to manipulate his data items and data aggregates. Potential advantages for the users include:

1. Content-accessible data bases usable from all connected devices.

2. Efficient storage/access of large data bases.

3. Simple and direct programming with data manipulation and storage management handled by the set processor.

4. Simple data base design and entry from source representation to set processor representation with no predefinition necessary.

5. Capability available for user sort/order specification.

Potential advantages for the central computing facility include:

1. Significant reduction in tape/disk pack storage and mounts.

2. Flexible environment that allows upgrading hardware/software configuration without causing major interruptions in service.

3. Minimal traffic on data communications network.

4. Improved central memory usage on large processors.

The concepts of data networks [1] and mass storage systems [29, 39] are relatively well known and detailed discussions are beyond the scope of this paper. However, the concepts of set processing (see Childs [9,10,11] and Schwartz [41,42]), and specialized data base computers (see Canaday et. al. [7] and Winter [52]), are relatively new and the investigation of their role as tools in data management is the primary goal of this research. A review of set processing and a comparison of potential advantages with those of other generalized data base management approaches are given in section 2. A secondary goal was to provide the means by which a mass storage system could be connected to a local

network so that it was conveniently accessible by the user community. A further consideration was that the addition of the mass store did not saturate the network with data base transfers. The addition of an autonomous set processor seems to be a suitable solution. The user community may easily communicate with the set processor by means of a collection of operations callable from either interactive terminals or programs executing on connected processors. The network traffic remains at a minimum because in most cases the data bases are not transferred but are manipulated at the set processor/mass storage system (SP/MSS) node. Only descriptive messages and data to be displayed or used in calculations is actually transferred. This will be demonstrated by example in section 4.0.

A recent paper by Canaday et. al [7] describes an approach to data base management using a "back-end" computer as an autonomous data base machine. The proposals set forth in our paper differ from the Canaday proposals in two important respects. First, we emphasize the autonomous data base computer as a node in a network serving all connected parties. The Canaday emphasis is on the data base computer as a back-end machine serving one or more hosts. Secondly, in our proposal, the techniques for implementing the data base machine are based on set theory (plus extensions) while the Canaday implementation is based on the CODASYL DBTG [17] specifications. These differences and their implications are discussed in detail in Sections 2 and 3.

2.0 Background

Set processing is developing in two areas of computer science: programming languages and information systems (i.e. data base systems). The programming language development (see Earley [21,22,23]) effort has been pioneered by the development of SETL [42]. However, this paper is primarily concerned with the information system aspects of set processing. A recent paper by Whitney [51] details the evolution of data management development culminating in the fourth generation information systems. In particular, Whitney notes that "Concepts from set theory and relation theory will become more widely used as the advantages of a sound theoretical basis for information systems become more widely appreciated." Another study, Hardgrave [26], provides more incentives for exploring set theory as a foundation for the analysis of mass storage structures. This study demonstrated that (1) some queries using Boolean connectors when applied to tree structures are open to multiple interpretations and (2) using set theory as a tool not only clarifies the problems, but provides intrinsic solutions.

Another important result is the development by Childs [11] of a new approach, called "extended set theory", that underlies the classical notion of set theory and solves or, more accurately, circumvents a number of outstanding theoretical problems. The major problem, the general n-tuple definition, is mentioned in the text, Berztiss [3] and discussed in detail in Skolem [45]. This is a previously ill-defined

area of mathematics that is of the utmost importance for data structures and information processing. To a large extent, the anomaly of the n-tuple definition, specifically the ordered pair definition, was the impetus for developing extended set theory. This new theory provides a convenient vehicle for the definition of n-tuples as well as sets in a simple and direct manner. Most of the existing body of mathematical development based on classical set theory is unaffected by the introduction of extended set theory at the lowest level. From a computer science viewpoint extended set theory lends itself to the implementation of sets, n-tuples, and nestings thereof more readily than classical set theory (see [11]). Research is still required in the area of set processor implementations, but Childs' theory provides a solid foundation upon which to build.

Over a period of several years, Childs and the Set-Theoretic Information Systems (STIS) Corporation have developed a software package called Set-Theoretic Data System (STDS) [43]. This package is available at the computing centers at the University of Michigan and Wayne State University and it is commercially available to IBM 360/370 installations through STIS Corporation. Besides STDS, the STIS Corporation proposes to develop a Set-Theoretic Storage Management System (ST/SMS) that will serve as an autonomous set processor as well as drive and manage mass storage devices at the controller level. The concepts behind STDS and ST/SMS represent different approaches to solving the data handling problem although both are based on extended set theory. Below we briefly describe these systems giving some points for comparison.

STDS is a software system designed as a research tool and implemented on top of a general purpose operating system on third generation computing equipment (e.g. IBM 360/67). The ST/SMS will be a stand-alone system of minicomputers connected on one side to a auxiliary storage system. On the other side, the ST/SMS may be attached to a single minicomputer, a single large scale mainframe, a local network of mainframes or a geographically distributed network such as ARPANET [1]. The auxiliary storage system should be direct access and could range from a single disk to several trillion bit storage devices. This paper concentrates on a more detailed study of the set processor (e.g. ST/SMS) in a local network environment. STDS is currently operational. ST/SMS is still in the development stage and no prototype exists. The hardware design, a proprietary item belonging to STIS, is complete in the preliminary design and the minicomputers most suited to the task have been tentatively selected. However, the system must be custom tailored (i.e. microprogrammed) to the mass storage device and the network or communications protocol. Because of rapid changes in hardware technology and pricing structures these choices must be constantly re-evaluated.

The user may find STDS similar in many ways to the notion of a relational data base system as described by Codd [12,13,14,15,16]. However, STDS was developed independently of and is conceptually different from the relational systems that are based on the Codd model. The names given to operations are different, but users of

both types of systems will notice distinct similarities. In particular, the primary structure available in both STDS and the relational systems is the n-ary relation. Formally, the n-ary relation is a set of tuples and each tuple in the set has exactly n elements. The system would support a large number of relations simultaneously and n may vary from relation to relation. From the outset, a major concern expressed by Codd [12] has been that data base structures be easy to understand for the non-programming user as well as the programmer. Relations are very powerful in this respect in that they may be viewed in tabular form and with only a few restrictions they may be manipulated as tables of data. There are a number of papers (see e.g. [16] or [19]) available that serve as excellent tutorial material in this area.

In contrast, ST/SMS supports not only n-ary relations but the entire spectrum of structures that may be defined using sets, n-tuples and arbitrarily deep nestings thereof. We refer to these as "set-based structures" rather than "data structures" because the latter term often is intertwined with linked structures and pointer structures in the minds of computer specialists. The "set-based structures" do not imply an implementation technique; they serve as a formalism for expressing the various relationships between items. This capability to support a broad range of structures gives the ST/SMS sufficient power to serve as a generalized storage manager. That is, the current design philosophies of generalized data base management systems rely on a fixed storage structure scheme that forces all data management

requirements, no matter how diverse or dynamic, to be supported by a pre-structured implementation. The ST/SMS is a low level support system that replaces fixed-storage implementation philosophy with a dynamic and efficient means for optimizing the storage media characteristics in co-operation with the data management requirements. A storage management system (e.g. ST/SMS) based on extended set theory allows dynamic creation, restructuring and accessing of diverse structure types (e.g. sequential, inverted, multiply-linked rings, DBTG "sets", etc.).

While STDS is an existing system with a more limited capability, ST/SMS is a conceptual system that promises to provide a number of important advantages for multi-mainframe and network systems. The software for STDS is to a large extent written in FORTRAN and could be installed on a third generation operating system (e.g., KRONOS) in three months. Although ST/SMS is still in the design stage, it seems that a prototype could be built in one or two calendar years.

In the following paragraphs, we compare the set processor approach with other generalized data base management approaches. In recent years, there have been three distinct theoretical approaches to solving the generalized data base management problem, the TDMS approach, the CODASYL DBTG approach and the set processor (SP) approach. The TDMS approach originated in the Time-Shared Data Management System [4,5]. The philosophy involves the use of partitioned tree structures, inverted files, and a Boolean query and access language. An advanced commercial system using the TDMS approach is SYSTEM 2000 [49]. The CODASYL DBTG

approach has been defined and revised over a period of years by the CODASYL Data Base Task Group [17]. The philosophy employs networks of "owner" and "member" records in order to describe and manipulate the logical data relationships required by users. The design provides mechanisms for accessing data within the network relative to the current position. An advanced commercial system using the DBTG specifications is the Integrated Data Management System (IDMS[30,31]). The set processor approach has been developed primarily by D. L. Childs and the STIS Corporation. The only commercial system available using the set processor approach is STDS [43]. Our purpose here is not to compare implementations but approaches (i.e. philosophies). That is, we are concerned with the potential capabilities and limitations characteristic of the three approaches. Particular implementations may have limitations not imposed by the corresponding philosophy.

One other philosophy, the relational approach [12], should be explained at this point. For our comparison here, the relational approach will be considered to be included in the set processor approach. This is due to the fact that relations are well-defined set-based structures and as such can be easily supported by a set processor. The inclusion of the relational work under set processing is not meant to detract from its importance to information system design. On the contrary, relations are valuable to users because they are convenient to define and manipulate even for the non-computer-oriented person. Furthermore, the theory surrounding relational data bases has been well

developed by Codd [12], Date [19] Heath [28] and others. Figure 2-1 gives a list of characteristics of the three approaches. We make no claims that this list is complete, but hopefully it is representative of the concerns of the potential user of an information system. We will comment briefly on each of the characteristics.

Possibly the most important long term criteria for an information system is its theoretical foundation. The logical data relationships, and the operations provided to manipulate those relationships, must be carefully and consistently defined at the theoretical level in order to insure that anomalies cannot occur in retrieval or update processes. These considerations are important since the primary purpose of an information system is to provide reliable answers to queries. In order to answer queries, the information system must incorporate a query language as a user interface. Natural language (e.g. English) has not been (to date) a feasible interface because of its inherent ambiguities and idiosyncrasies necessitating extensive iteration in order to obtain reliable answers. An acceptable query language can be defined using elementary concepts from predicate calculus and symbolic logic. This language can be designed to be functionally similar to natural language. This has already been achieved and is available in systems using the TDMS approach (see [49]). The class of query languages based on the predicate calculus make use of Boolean connectors (e.g. AND, OR, NOT) and as such have a duality with set theory restricted to a Boolean algebra (e.g. intersection, union, universal relative complement). If one

INFORMATION SYSTEM APPROACHES

POTENTIAL CHARACTERISTICS	TDMS	DBTG	SP
STABLE MATHEMATICAL BASIS	No	No	YES
DYNAMIC DEFINITION	No	No	YES
ACCESSIBILITY BY CONTENT	YES	No	YES
AGGREGATE PROCESSING	No	No	YES
STORAGE REDUCTION	No	No	YES
PROGRAM ACCESS	YES	YES	YES
BOOLEAN ACCESS	YES	No	YES
INTERACTIVE ACCESS	YES	YES	YES
BOOLEAN ACCESS	YES	No	YES
LARGE DATA BASE SUPPORT	No	No	YES
MSS SUPPORT	No	No	YES
DATA INDEPENDENCE	YES	YES	YES
SHARED DATA BASES	YES	YES	YES
DATA BASE PROTECTION	YES	YES	YES

FIGURE 2-1

expects responses from information systems to be consistently reliable, it is mandatory that the information system be designed on a consistent foundation. The mathematics of set theory provide such a foundation. Using this foundation, with particular emphasis on the membership condition, query languages based on the predicate calculus may be implemented and responses will be consistent and reliable.

In the case of the TDMS approach, the application of Boolean logic to partitioned tree structures may cause several anomalies to occur. As mentioned previously, these are detailed in Hardgrave [26]. A parallel study by Ray [38] predicts that even more alarming results will occur if Boolean logic is applied to general network structures similar to those found in the CODASYL DBTG [17] specifications (see Parsons et. al. [36,37]). Furthermore, the DBTG [17] specifications rely on questionable theoretical foundations. For instance, the DBTG concept of a "SET" is not precisely defined. This structure is characterized in terms of "owner records" and "member records" and if we take A to be an owner record and $M_1, M_2, \ldots, M_N$ to be member records, then the DBTG "SET" concept is graphically described as shown in Figure 2-2. However, this is not an adequate substitute for a precise mathematical definition. In order to illustrate this point, we will direct attention to several pitfalls that are encountered if the basic structures of an information system are not precisely defined. We do this by suggesting several possible candidates

DBTG "SET" REPRESENTATION

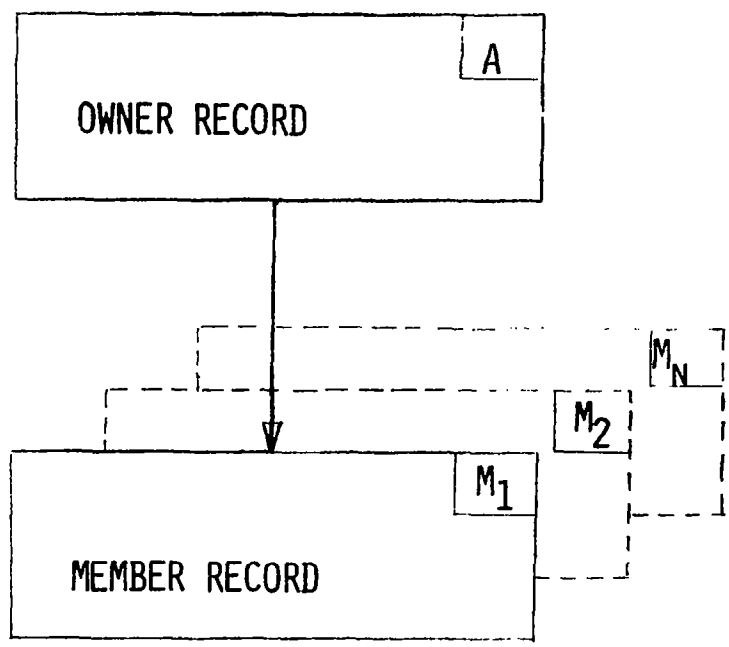

FIGURE 2-2

for the DBTG "SET" defined in set-theoretic terms and written in set-based structure notation.

(a) $\{A, M_1, M_2, \ldots, M_n\}$

(b) $\langle A, M_1, M_2, \ldots, M_n \rangle$

(c) $\langle A, \{M_1, M_2, \ldots, M_n\} \rangle$

(d) $\langle A, \langle M_1, M_2, \ldots M_N \rangle \rangle$

(e) $\{A, \{M_1, M_2, \ldots, M_N\}\}$

(f) $\{A, \langle M_1, M_2, \ldots, M_N \rangle\}$

The notation above uses { , } to delimit mathematical sets and < , > to delimit sequences (i.e. tuples). One definition (a) relies strictly on the notion of a mathematical set and another (b) relies strictly on the notion of a mathematical sequence and the others (c-f) are variations using sets and sequences. Numerous possibilities exist using other nestings of sets and sequences. The absence of an official set-theoretic definition of the DBTG "SET" invites an important question. Is $M_1, M_2, \ldots, M_N$ meant to represent a mathematical set or a mathematical sequence? Consider (a) as the definition for the DBTG "SET". The DBTG specifications provide a "next" operator. If the current position is established in a DBTG "SET", then the "next" operator may be invoked to reposition to the next logical record in the "SET". This is in conflict with the mathematical notion of a set in which there is no intrinsic order and as such could have no "next" operator. Therefore definition (a) is not a viable definition for the DBTG notion "SET". Consider (b) as the definition for "SET". This

definition relies on the notion of a sequence, and mathematical set operations (e.g. union, intersection) do not produce meaningful results when applied to these sequences. (We assume here that nesting of the tuples into ordered pairs and the Kuratowski ordered pair definition are imposed (see [45])). Therefore (b) is not a viable definition either. Similarly the other definitions (c-f) can be shown to be inadequate. Furthermore, if a logical structure is treated as a mathematical set at one instant and as a tuple at another instant, anomalies will certainly occur. In short, the CODASYL DBTG [17] specifications, a proposed standard, has provided a data structure for implementation without providing a theoretical foundation for logical data relationships and high level queries. If the set operations do not produce meaningful results in all cases, then queries expressed in the predicate calculus based language cannot produce reliable and consistent responses in all cases. An implementation of such a language will be subject to anomalies as severe, if not worse, than those already encountered in TDMS approach implementations. The query capability and associated operations can only be as reliable and consistent as the lowest level definitions allow. The key is precisely defined low level primitives and this can be accomplished using precisely defined membership conditions.

Next, there is the question of a definition for a data base before it is entered into a generalized information system. It has become fashionable to have "data definition languages" and force the user (or data adminis-

trator) to predefine the logical relationships before actually entering the data. This has some advantages; however, the predefined schemata is a constraining factor. Data bases grow not only in quantity but often in arrangement. New logical relationships are necessary and must be added. In a predefined environment when this occurs, existing data bases must be "restructured" or completely dumped out in source form and reloaded using an updated definition. Both the DBTG approach and the TDMS approach require predefinition. The set processing requires no definition. Definitions imposed by the user are dynamic. Logical relationships may be created, destroyed or updated after data has been entered. The definition of the data base is a user view. If the user wishes to change this view, he may do so dynamically by redefining his sets and tuples. In fact, the overhead involved in "restructuring" a set-theoretical data base is minimal because the definition was not frozen in advance.

Another important aspect in the analysis of information systems and data management systems is accessibility by content. This implies that the user of such systems may access data items and aggregates by their attributes and values rather than by their mass storage addresses, either absolute or relative. The DBTG specifications provide a very limited capability to access records by identifier. In addition, it is possible to access a record through a calculated (e.g. hash-coded) identifier if the key was pre-defined as a calculated key. However, there is no possibility to access a record by specifying a combination

of conditions as there is in the TDMS approach and the set processor approach. The TDMS approach provides accessibility by content through the use of a Boolean expression following the reserved word WHERE. An example is given below:

PRINT JOB WHERE CP TIME LE 1.0 AND FIELD LENGTH LE 75000;

The general form of the access commands is:

< ACTION> < X > WHERE < Q >;

The < ACTION> may be PRINT, UPDATE, etc., < X> is a list of named items and < Q> is a Boolean qualification clause. The set processor approach provides in its most rudimentary form a capability that is syntactically less elegant but semantically more elegant than the TDMS access clause. First, with a parser/translator the TDMS access clause can be syntactically translated into the set-theoretic functions because of the duality between the predicate calculus and set theory. Secondly, the set processing approach provides the user with the capability to isolate and save intermediate sets and perform operations on those sets as "sub-data-bases". This leads us to a discussion of aggregates. Inherent in mathematical set theory is the notion that names may be assigned to arbitrarily large aggregates. This concept is carried over to the extent possible (i.e. large finite aggregates) in the set processing approach to information systems. Although all three approaches allow naming of aggregates ("sets" in DBTG, "repeating groups" in TDMS, "sets" or "tuples" in SP) only the SP approach provides operators that take aggregates as operands and produce aggregates as results. In DBTG

and TDMS in order to perform operations on data the items must be retrieved and operations performed on an item by item basis.

Set processing provides a potential storage reduction capability not offered to any extent by either the DBTG or the TDMS approach. The TDMS approach can in some very special cases reduce the overall storage requirements if the data base has a large amount of redundancy and the redundant values consist of long character strings (e.g. 10 characters or more). In general totally inverted TDMS data bases may be expanded by a factor of seven over the source character representation (without redundant blanks). Partially inverted data bases with relatively few inverted keys may only expand by a factor between one and two. DBTG data bases offer little hope for storage reduction over the source character representation. The overhead in pointers, etc. will generally expand the data base by a factor between one and two. Set processing offers a larger potential for storage reduction for data bases because it capitalizes on the use of the membership condition. The membership condition may either be given explicitly (i.e. by enumeration of members) or implicitly (i.e. by providing a statement describing the circumstances defining membership). An implicit membership condition may reference existing sets previously defined or system defined sets (e.g. the integers or the reals representable by a given machine). An example of an explicit definition of the set of integers between (and including) ten and twenty is:

$$A = \{10,11,12,13,14,15,16,17,18,19,20\}$$

An example of an implicit definition for the same set is:

$A = \{ i \mid i \text{ is an integer and } 10 \leq i \leq 20 \}$

Sets and tuples that may be expressed by an implicit definition need not be stored explicitly. The potential for storage reduction of redundant data through the use of implicit membership conditions is one of the most attractive features of the set processing approach. Set theory plus the extensions defined by Childs [10,11] provides a semantic capability for implementing such an approach. Furthermore, the set processor serving as a storage manager may go a step further. The set processor may alter the stored version of a data base by "factoring out" redundant items but enabling the user to retain his view that includes redundancies. Removing a domain that has a high degree of redundancy from a relation - thereby creating a set-based structure that is not a relation - is an example of this. But the user continues to view his data as a relation.

Another important aspect is that of program/interactive access. A well-designed information system will provide both program and interactive access on a compatible basis. That is, data bases will be accessible from either executing programs or interactive terminals through an interface that is similar in both modes. Currently, the DBTG specifications provide only program access. Some suggestions have been made to provide an interactive mode, but we know of no implementation using Boolean connectors. In any case, it will be difficult to provide an adequate query capability because of the theoretical

problems with Boolean operators mentioned previously. In the TDMS approach, the interactive mode is the natural mode and usually the first implemented. However, a program mode can be implemented in a conveniently usable form by using a pre-processor for the host language (see [49]). In the set processor approach, the interface is via a collection of set-theoretic operators and those operators may be conveniently called from either an interactive terminal or an executing program (see[43] or [25]).

Next, there are the related issues of support for large data bases (i.e. greater than 10^{12} bits) and support for mass storage systems (MSS) (i.e. capacity greater than 10^{12} bits). Since very few large data bases exist and none of these are maintained under the three generalized approaches mentioned here, there is little concrete evidence to support our conjectures. The same is true for the MSS's since their appearance in computing centers is not yet widespread. However, the following logical arguments can be presented. First, there are two characteristics of all proposed MSS systems that are germane to this discussion: (1) the access times for the MSS systems will be slow relative to disks (1-15 seconds) (2) the transfer rates will be comparable to disks (5-10 million bits per second). These figures suggest that approaches that rely heavily on pointers and linked structures will not perform well in conjunction with the MSS systems. In general, each link or pointer that must be followed requires a disk access. The DBTG approach and the TDMS approach both rely on linked structures. In contrast, sets

and tuples may be stored and retrieved sequentially (for example, see Hardgrave [27]). Thus, set operations may be implemented to take advantage of the characteristics of the MSS systems.

Finally, there are the three issues of data independence, data sharing, and data base protection. These issues have been considered in numerous papers (see Stonebraker[46] and Canaday et. al. [7]) and will not be discussed in detail here. Data independence and data sharing are provided in all three approaches through the use of named items and named groupings. Mechanisms for protection from unauthorized use (security) can be implemented in conjunction with any of the three approaches. Mechanisms for protection from inadvertent damage (integrity) due to human or machine error can also be implemented using recovery and rollback techniques with any of the three approaches. However, protection from error propagation and detection of hardware errors continues to be an area demanding more research.

3.0 A Sample Network Configuration

In this section, a sample network configuration, shown in Figure 3-1, is described briefly on an item by item basis. This configuration consists of (1) a communications network, (2) a number of large scale processors (LSP's), (3) a number of time-sharing/remote job entry (TS/RJE) terminals, (4) a peripheral driver, and (5) set processor/mass storage

system (SP/MSS). Other components may be added as required; however, those enumerated are necessary to provide batch, remote batch, and time-sharing services in a consistent manner with universal access to data bases. The configuration as described would exist as a local network with all equipment except the TS/RJE terminals geographically concentrated in a single building. However, the network approach using a set processor can be readily extended to geographically distributed configurations.

In general, each large scale processor (LSP) is a large expensive computing system designed to run complex numerical codes. These codes often require large amounts of data input and produce large amounts of data output. Each LSP would normally have only scratch disks as peripherals.

Recently, the operating systems for these third generation machines have grown increasingly more cumbersome. Most support some level of multiprogramming. Some drive a host of peripherals such as tapes, card readers and printers. Many have built-in permanent file systems and time-sharing or interactive systems. The introduction of an interactive service and permanent file handlers dictates that much of the CP will be devoted to file manipulation, character handling, text editing, and job overhead; tasks for which these machines were not designed. These tasks can be handled very effectively and inexpensively by modern mini-computers. Furthermore, the development of such a complex operating system for a new hardware design is a multi-manyear undertaking for the vendor and these systems are seldom

SAMPLE NETWORK CONFIGURATION

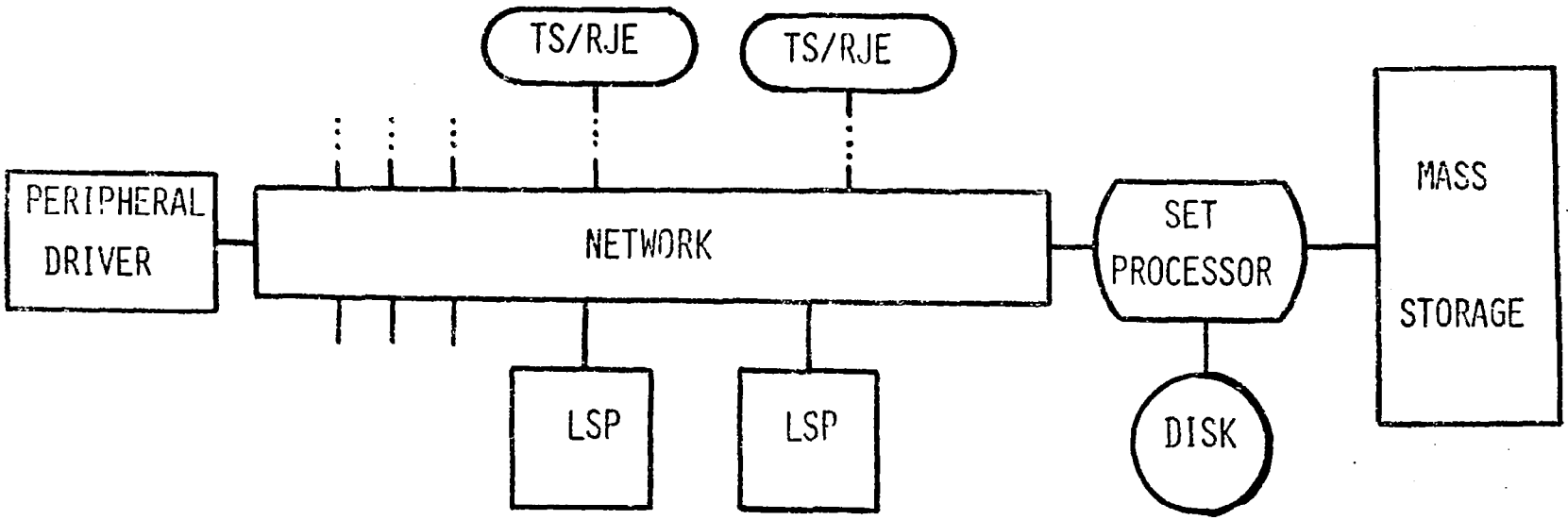

FIGURE 3-1

reliable for at least two calendar years after their introduction.

Hopefully, the network approach to computing systems will alleviate a number of these problems while providing the flexibility to add more equipment of any type without major upheavals in the users' operating procedures. The set processor will remove most of the file handling and data management tasks from the LSP operating systems and the TS/RJE terminals will remove the time-sharing, character handling and text editing burdens. Thus, LSP operating systems may be simpler, containing at most multiprogramming overhead and local file handlers. CP efficiencies on these expensive machines should increase and development time for operating systems on newly designed processors should decrease dramatically. It is also important to realize that memory connected to LSP's has a higher cost per bit at a given memory speed than memory associated with a minicomputer. Therfore, buffers for auxiliary storage devices (e.g. disks or MSS's) should reside in minicomputer memory and free LSP memory to perform the functions for which it was designed.

The peripheral driver is the computer on the network to which most, if not all, of the input/output peripherals belonging to the central computing facility are attached. This would include high speed printers, card readers, tape drives, plotting equipment and the like. It might also include some specialized equipment such as graphics devices or data aquisition machines. The computer itself might be of the same class as an LSP (e.g. CDC 6400); however, it may be more economic to

use powerful minicomputers. This will depend to a large extent on the choice of peripherals.

The time-sharing and remote job entry terminals (TS/RJE) are the users' primary contact with the network. Each TS/RJE terminal would be a rather large minicomputer to which both keyboard terminals and peripherals would be attached. The memory size for a TS/RJE should be about 128K bytes and floating point hardware (or firmware) should be included. These computers must be extremely reliable with a mean time between failure measured in months. Furthermore, the environmental conditions necessary for their operation must not be more stringent than normal office conditions.

Peripherals on the TS/RJE terminals should include minimally several cartridge disk units plus slow speed card reader, a tape drive, and a printer. Higher performance card readers, printers and tape drives may be added as the need arises. It is impossible to over-emphasize the importance of the disk storage on the TS/RJE terminals. This should consist of a minimum of two disk units each with at least one removeable platter. The total capacity should exceed 10 million bytes for each TS/RJE terminal. This storage is partially used for the disk operating system residence and overhead; however, the bulk of the storage will be used for storage of "small" user files programs, and small data sets. Large data files and data bases will reside on the mass storage system.

The software necessary to produce a reasonable TS/RJE terminal is already available to a large extent from minicomputer manufacturers. This consists of a multi-user disk operating system that can be bootstrapped in from a disk (i.e. deadstarted) with only a few manual operations. The disk operating system (DOS) should also include a file system that provides a capability to the user community to store small files on the local cartridge disks, and modify and delete them as desired in an interactive mode. A good context text editor and an ANSI FORTRAN compiler are absolutely essential. Other software may be included as enhancements.

Each TS/RJE should accomodate 30 or more users with about 10 terminals on-line simultaneously. Larger groups may be accommodated if more memory and disk space is added. This approach has several distinct advantages over traditional approaches. First the user has immediate control over his files and his tapes since they reside with him at the TS/RJE. Secondly, should the central network be unavailable to the TS/RJE user for any reason, he may still perform a large number of functions at the TS/RJE. He may edit programs and small data sets. He may compile programs, correct syntax errors, execute small test cases, and prepare larger jobs to be sent to the central facility when it is available. Because of the reliability of the TS/RJE, there is no reason why it cannot be available at all times. The operation is simple and the user may bring up the operating system himself. Another major advantage is that the central facility will only be responsible for the

larger data sets and data bases belonging to the user community and these will reside on the mass storage system. This will dramatically decrease the need for tapes and thus alleviate the tape storage problem, a major headache of many central computing sites today. Many of the tapes that will remain will be held by the user, not by the central computing center, and will be read at the TS/RJE terminal and transferred over the network when necessary.

One other significant advantage is that terminals connected directly to TS/RJE terminals can be driven at high data rates (e.g. 9600 bps for storage tubes and display terminals). This is a significant increase over many time-sharing systems running today on large scale processors and the higher data rates make interaction immeasurably more pleasant for users, particularly for text editing or graphics.

The hardware, software, and peripherals necessary to obtain suitable TS/RJE terminals are already available from a number of minicomputer manufacturers. At present, each terminal will cost $50,000 to $100,000 depending on the amounts and types of memory, disks, and other peripherals chosen; however, these costs are steadily decreasing as technology makes the hardware less expensive.

The set processor/mass storage system (SP/MSS) provides an on-line storage managed by a processor capable of handling a broad spectrum of logical data relationships and physical storage techniques. Furthermore, the processor can react to user requests ranging from a

single data item to a large aggregate and transfer only the required data. The storage environment could range in capacity from a few disks to several trillion bit devices. For our discussions in this paper, we will assume that the central computing facility will have at least one MSS. Examples of these systems currently on the market are the Ampex Terabit Memory [29,39] and the IBM 3850 Honeycomb Store [32]. The Ampex systems range in cost from a basic system of approximately 10^{11} bits at $600,000 to a maximum capacity of 3×10^{12} bits at about $4,000,000. Access times for these systems will be in the range 1-15 seconds, significantly slower than disks. However, transfer rates will be more comparable with disk rates at 6-8 million bits per second. The set processor itself might consist of several minicomputers connected via a high speed bus. In addition, some disk and/or drum space is necessary for spooling and processing information. A more detailed hardware description is beyond the scope of this paper, but may be obtained from the STIS Corporation. However, the functional description of the set processor will be illustrated in the next section.

4.0 Using the Set Processor

This section contains examples of user interaction with the set processor in the network environment shown in Figure 3-1. This consists of an example of an interactive session using a test data base plus an example of a FORTRAN program accessing the same data base. The

possibilities for a user interface to the set processor are abundant and equally diverse. In order to demonstrate the following examples with a minimum of preliminary discussion, we will use a test data base that is in STDS format. It is important to recognize that constraining the set based structures to n-ary relations is not a restriction imposed by the set processor. However, relations are very versatile and easy to understand and manipulate. Furthermore, a large class of data bases may be represented in relational form.

In the following paragraphs, we will attempt to describe the set processor in functional terms by giving examples of its usage. A more complete description of the functions that may be available in a typical set processor is given in Appendix A. The communication between the set processor and the network (i.e. any machine or human connected to the network) will be in terms of set-theoretic functions. At any point in time the set processor will consist of a collection of functions that it will recognize and execute. Furthermore, for each user identifier, it will maintain a collection of names of sets (i.e. a universe of discourse) that it will recognize and manipulate using the designated function. The functions that can be made available in the set processor are also abundant and diverse. For the sake of simplicity, we will use, in this report, only the functions defined in Appendix A. A more complete and somewhat different collection of functions for manipulation of relations is given in the STDS/OS Users' Guide [43]. Thus, the communication between the network and the set processor is in the form of references

to functions and parameters. The parameters often reference sets, relations or n-tuples to be manipulated. For example, the operation

IN (A,B,C)

would intersect the existing sets A and B defining a new set C. Figure 4-1 gives a few entries from a hypothetical data base that logs jobs that were run at a scientific computing center. A similar data base with actual data run at the NASA Langley Research Center computing facility may be found in [25]. The sample data base would contain about ten million characters for each year logged. The DAY, MONTH and YEAR domains (i.e. columns) give the date the job was introduced into the system. The JOB domain gives an identifier for the job. The CP domain gives the central processor time in minutes used by the job, the FL gives the field length in words (i.e. memory requirements) used by the job, the OS domain gives the operating system calls (i.e. supervisor calls) made by the job, and the MT domain gives the number of tape mounts required by the job.

Assume an interrogator logs on from a TS/RJE terminal and he wishes to access the sample data base. Figure 4-2 describes an interactive session in which the interrogator sends queries to the set processor and receives replies answering his questions. A "?" precedes the lines entered by the interrogator; and all other lines are output from the set processor via the network and the TS/RJE. We will describe the session on a line by line basis. At the same time we will note the flow through the network and the effect on its performance.

SAMPLE DATA BASE - COMPUTER JOB LOG

DAY	MONTH	YEAR	JOB	CP	FL	OS	NT
01	JAN	74	AA13216	.01	52000	66	0
01	JAN	74	AA23764	1.37	137000	4813	1
01	JAN	74	AB37152	11.64	120000	1287	0
01	JAN	74	AB43261	1.06	145000	1576	0
01	JAN	74	AB71351	.68	55000	916	0
01	JAN	74	AC13754	.17	55000	323	0
:	:	:	:	:	:	:	:

FIGURE 4-1

INTERACTIVE USAGE

```
                    ⋮
? OPEN (D,JOB-LOG-1974,ID=WTH,PW=NOW)
     #DOM = 8          CARD = 1017341

? DOMAINS (D)
     DAY, MONTH, YEAR, JOB, CP, FL, OS, NT

? RS (D,NT=0,NEW=NOTAPES)
     #DOM = 8          CARD = 657231

? AVG (NOTAPES,DOMAIN=CP)
     AVG CP = 2.83

? RS (D,CP≤1.0,NEW=SMALLCP)
     #DOM = 8          CARD = 601317

? IN (NOTAPES,SMALLCP,NEW=SMALLJOB)
     #DOM = 8          CARD = 450911

? SAVE (SMALLJOB,SMALLJOB-LOG-1974)
                    ⋮
```

FIGURE 4-2

PROGRAM USAGE

```
(1)        CALL OPEN (D,JOB-LOG-1974,ID=WTH,PW=NOW)
(2)        CALL RS(D,MONTH=JULY,NEW=TARGET)
(3)        CALL RS(TARGET,NT≥1,NEW=TARGET)
(4)        CALL RS(TARGET,CP≤1.0,NEW=TARGET)
(5)        CALL RS(TARGET,FL≤60000,NEW=TARGET)
(6)        CALL RS(TARGET,OS≤2000,NEW=TARGET)
(7)        TOTAL=0.0
(8)        K=KARD(TARGET)
(9)        IF(K.EQ.0)GO TO 500
(10)       DØ 200 I=1,K
(11)       CALL SOME(TARGET,N=1,NEW=ONE)
(12)       CALL EXT(ONE,DOMAIN=CP,V=CP)
(13)       CALL EXT(ONE,DOMAIN=OS,V=OS)
(14)       CALL EXT(ONE,DOMAIN=FL,V=FL)
(15)       CP = CP + OS/2000
(16)       COST = (5*CP*FL)/60000
(17)       TOTAL = TOTAL + COST
(18)       CALL RC(TARGET,ONE,NEW=TARGET)
(19) 200   CONTINUE
(20) 500   CONTINUE
(21)       AVG = TOTAL/K
(22)       PRINT 90, AVG
(23)  90   FORMAT (1X,'AVERAGE COST =',F7.2)
```


FIGURE 4-3

The data base "JOB-LOG-1974" is opened for access with the OPEN statement and the set processor responds by returning the number of domains and the cardinality (i.e. number of elements) of the set. At this point the set processor moves the most frequently accessed part of the data base from the slow access trillion bit store to one of its scratch disks. Note also that the only traffic passed through the network is less than 50 characters that comprise the OPEN command and the return reponse of less than 30 characters (probably one packet in each direction).

The next command given is the DOMAINS command. This command asks the set processor to return the names of the domains of D. The response is the domain headings that may then be used in other commands. The next command, RS (i.e. restrict), asks the set processor to form a new set called NOTAPES from the elements of D where the value in domain NT equals zero. That is, NOTAPES is the subset of D such that NT equals zero. Again the set processor returns the number of domains and the cardinality. At this point the interrogator, having determined that roughly 65% of the jobs are no-tape jobs, would like to know the average CP time used by these jobs. Since the no-tape jobs have been isolated in NOTAPES, this can be easily accomplished by issuing the AVG command on the set NOTAPES for the domain NOTAPES. The set processor responds with the average of 2.83 CP minutes. Next, the interrogator isolates the jobs that used one minute of CP time or less in the set SMALLCP. The set processor responds with the number of domains and a cardinality of 601317. Thus, about 60% of the jobs require less than one minute of CP time. In the next command, the intersection of the

two sets, NOTAPES and SMALLCP is formed and called SMALLJOB. This is the set of jobs using no tapes and one minute of CP time or less. About 45% of the jobs are of this class. Finally, the set SMALLJOB is saved by the interrogator as another data base for future use. It is important to emphasize again that the traffic on the network activated by this session was less than a few hundred characters. Using a third generation data management system, it would have been necessary to transfer the entire data base to the disk storage associated with an LSP, and then interact with the LSP. The traffic in that case would be measured in millions of characters. Using the set processor approach, the user may manipulate his data base at the SP/MSS node with only messages traveling on the network.

Figure 4-3 describes the use of set processor commands in a FORTRAN program environment. Some liberties have been taken in the areas of FORTRAN interface and calling sequences (e.g. keyword parameters) for the sake of readability. The problems in using ANSI FORTRAN is an analogous manner would not be insurmountable; however, it would be considerably more cumbersome than the code shown here. These shortcomings may be attributed to FORTRAN rather than the set processor. Again we will describe the program on a line by line basis as well as pointing out areas that may affect overall network performance. The lines in Figure 4-3 are numbered on the left for easy reference.

We assume that the purpose of the program is to calculate average cost of small tape jobs run in July 1974. We arbitrarily define a small

tape job as one using one minute or less of CP time, having one tape or more, using a field length of 60000 words or less and making 2000 OS calls or less. We also assume that the cost function, in dollars, is

$$C = (5*FL*(CP + \frac{OS}{2000})/60000$$

This implies that the cost in dollars C is five dollars per CP minute at the nominal field length of 60000 words. We assume that this function is not built into the set processor (although it could be) and thus it is necessary to write a program to evaluate the average cost per job.

The data base is opened as a local set D in line 1 in a manner similar to the interactive session. Lines 2 through 6 restrict the set to be considered to the set of small jobs as previously defined. This produces the set "TARGET" as the set to be processed on an item by item basis in the remainder of the program. This is a very important aspect because it is not necessary to transfer the entire data base to the large scale processor (as would be the case with a standard file system) in order to answer a question involving only a portion of the data base. For a typical data center only about 10% of the jobs would be in the class of small tape jobs and, furthermore, we will only consider the month of July. It would be unwise to transfer data through the network that is not needed in the calculation process.

At line 7, we initialize the total cost used later to calculate average cost. The function call at line 8 obtains the cardinality of the

set, TARGET, and stores that in K. If the cardinality is zero, we transfer control to the exit condition. Otherwise, we enter a loop to traverse the set element by element. The function, SOME, referenced at line 11 gives the programmer the capability to access one or more elements at random from one set and form a new set of these elements. In this particular case, we pick a single element from TARGET and put into a new set named ONE. Lines 12 through 14 extract domains CP, OS and FL from the element of ONE and store them in the local variables of the same names. Lines 15 through 17 calculate and total the cost function. Line 18 references the relative complement function, RC, which in this case deletes the element of ONE from TARGET. This loop is then continued until TARGET is exhausted. Lines 20 through 23 process the termination condition. The average cost is calculated and printed.

First, the reader will notice that no storage allocation or buffers are required in the memory of the large scale computer. This means that many programs that require large blocks of memory can be reduced to minimal memory because the set processor manages the data. On the other hand, many large scientific codes need ready access to large blocks of data in order to use the central processor efficiently. For these programs, it will still be necessary to request that large blocks be moved to central memory from the SP/MSS. In these cases the task is made easier in that the programmer may request exactly the data that he needs in exactly the ordering that he needs to perform the calculations. This will be of the utmost interest to those working with vector machines.

Secondly, it is important to note that programs written to call the set processor are completely independent of the size of the data base.

Many times large scientific codes have been produced to process an incore case. When the time came that the problem size exceeded the core limitations, it caused major problems in converting the code to use auxiliary storage. We submit that the proper approach to producing large scientific codes is to use the following procedure. Initially write the code in a straightforward, structured and readable manner. All data management functions should be delegated to the set processor and only required information should be requested.

Upon execution of the code, if it is found that the performance of the program is not satisfactory because wait times for data are excessively high, then it is necessary to do the following. For sections of code where the data requirements are high (e.g. tight loops accessing matrix elements), buffers (i.e. working storage) should be allocated, and requests for data on an item by item basis should be replaced by requests on a block basis. The programmer may additionally request that the data be ordered in a fashion that is most suitable for him; a feature not generally available using conventional auxiliary storage techniques.

5.0 Concluding Remarks

In this section, we will enumerate some of the more salient advantages of the set processing approach implied if not explicitly stated in previous sections. Many advantages may be attributed to the mass storage system, some to the concept of a autonomous data base/storage management computer, others to the communications network, and still others

to set processing. However, it is important to consider the overall configuration and examine the inherent advantages. Some of the benefits are realized by the user community while others are realized by the central computing facility. There is a large degree of overlap.

The user community will find itself with a more responsive system than it had with stand-alone mainframes. The system is easy to use and simpler to program. All data bases will be accessible by content in either an interactive mode or a program mode from the LSP's, the TS/RJE's or other equipment attached to the network. The user may reference his data and manipulate his data bases on an aggregate basis. This results in more efficient processing as well as user convenience. Multiple users may access a single data base simultaneously, and single users may access multiple data bases. A comprehensive sorting capability is provided that permits the user to obtain data in an ordering most convenient for his use. Programs should be easier to read and write because the numerous burdens of data management will be handled by the SP/MSS. It will not be necessary for the user to allocate large blocks of storage, manage tables, or set up buffers. He is functionally separated from the storage media and may ask for the items needed for computation or display. As a result, these programs will be conceptually simpler, more structured and easier to prove correct. Furthermore, the program should more closely reflect the theoretical formulation of the problem. Other important advantages of the set processor approach to networking include a stable mathematical foundation for information system design, a dynamic data base definition and restructuring capability, a potential for reduction of data base storage requirements over source character representations, and a convenient and easy to use procedure for initially entering data bases into the SP/MSS mode.

The central computing center or facility stands to realize substantial gains as well. A most urgent problem, the storage of large numbers of tapes (many of which are only partially used) should disappear. Large data bases will be maintained by the set processor/mass storage system. Small files (e.g. routines under development) will either reside on the modest file systems at the TS/RJE terminals or, possibly, on tapes held by the user. The central facility will no longer be responsible for the small files. The most traumatic times during the histories of large computing facilities occur when new hardware or software is introduced that changes the overall configuration of the facility. Existing programs are often difficult to convert and each programmer must learn about a new system. Many of these problems will be circumvented in a network environment. New hardware may be added by interfacing it to the network. Immediately the new hardware has access to existing data bases. Users that prefer hardware of different vendors or different operating systems on the same hardware may easily coexist (even using the same data base) in the network environment. In short, this approach provides for a maximum of flexibility because it is modular; and modules may be added, deleted, modified or tuned for better performance without disrupting the entire system. Another important advantage is the more efficient use of central memory on the large scale processors. In many cases the level of multiprogramming may be raised substantially because applications programs will require less memory for buffers and working storage. As a result each program will require less total memory and thus more programs will fit in memory at one time. As noted previously, in some cases, it will be advantageous to maintain large buffers or working storage for the sake of computational efficiency; however,

APPENDIX A

Set Processor Operations

Brief descriptions are given here of a few operations that would likely appear in a typical set processor. Positional parameters and mandatory keyword parameter are listed in the calling sequence in small letters. Optional keyword parameters are listed in the parameters section. All positional parameters are mandatory.

Name: OPEN(s,g)

Parameters:

s: local set name

g: global data base name

ID: user identification

PW: user pass word

Function:

Opens the global data base, g for use with local set name, s.

Name: CLOSE(s)

Parameters:

e: local set name

Function:

Close the set with local name, s.

Name: SAVE(s,g)

Parameters:

s: local set name

g: global data base name

Function:

Save the local set as a global permanent data base with global name, g.

Name: DOMAINS(s)

Parameters:

s: local set name

Function:

If s is a relational set, list the domain names.

Name: EXT(s,DOMAIN=d,V=v)

Parameters:

s: local set name

d: domain name

v: variable name

Function:

If s is a relational set, extract the elements of domain d and store them in a vector starting with local variable V. Note that this function is callable only from programs.

Name: ENTER(s,m,u,f)

Parameters:

s: local set name

m: number of domains

u: logical input/output unit

f: FORTRAN format specification

Function:

Create a new relation, s, by reading data from unit u by format f until an end-of-file is encountered. The format f should specify m items to be entered. Each record that is read represents one entry (tuple) in the relation.

Name: RS(s,e,NEW=n)

Parameters:

s: local set name

e: expression of the form <domain> <op><value.>
<domain> must be a domain
<op> must be one of $\neq$ = > < $\geq$ $\leq$
<value>must be a numeric value

n: local set name

Functions:

If s is a relational set, restrict s by the expression e and form the new relational set n. That is, n will contain the elements of s for which e is true.

Name: DMR(s,DOMAIN$_1$=d_1,DOMAIN$_2$=d_2,...,DOMAIN$_k$=d_k,NEW=n)

Parameters:

s: local set name

d_1,...d_k: domain names

n: local set name

Function: If s is a relational set, form a new relational set, n, with only the domains d_1,...d_k.

Name: KARD(s)

Parameters:

s: local set name

Function:

Returns the cardinality of the local set s. Note that KARD is callable only from programs. In interactive mode the cardinality of any set created or altered is automatically returned.

Name: UN(s,m,NEW=n)

Parameters:

s,m,n: local set names

Function: Perform the union of the two sets s and m and form the new set n.

ORIGINAL PAGE IS
OF POOR QUALITY

Name: RC(s,m,NEW=n)

Parameters:

s,m,n: local set names

Function: Perform the relative complement of m with respect to s forming the new set, n.

Name: SD(s,m,NEW-n)

Parameters: local set names

Function: Perform the symmetric difference of the two sets, s and m, forming the new set, n.

Name: SOME(s,N=k,NEW=n)

Parameters:

s: local set name

k: positive integer

n: local set name

Function: Form the new set n by selecting k elements from s at random. The set s is unaltered.

Name: AVG(s,DOMAIN=d)

Parameters:

s: local set name

d: domain name

Function: If s is a relation, calculate and return the average of all elements of domain, d.

REFERENCES

1. Abramson, N., and Kuo, F. F., Computer Communications Networks, Prentice-Hall, Englewood Cliffs, N.J., 1973.

2. A buyer's guide to data base management systems. Technical report number 703-010-61a, Datapro Research Corporation, Delran, N. J., 1974.

3. Berztiss, A. T., Data Structures: Theory and Practice, Academic Press, New York, 1971.

4. Bleier, R. E., Treating hierarchical data structures in the SDC Time-shared Data Management System (TDMS). Proc. ACM 22nd National Conference, MDI Publications, Wayne, PA, 1967, 41-49.

5. Bleier, R. E., Vorhaus, A. H., File organization in the SDS Time-Shared Data Management System (TDMS). IFIP Congress 1968, North Holland, 1968, 1245-1252.

6. Byrom, S. T., and Hardgrave, W. T., Representation of sets on mass storage devices for information retrieval systems. AFIPS Conf. Proc. Vol. 42, AFIPS Press, Montvale, N. J., 1973, 245-250.

7. Canaday, R. H., Harrison, R. D., Ivie, E. L., Ryder, J. L., Wehr, L. A., A back-end computer for data base management, Comm. ACM, 17, 10, (October 1974), 575-582.

8. Canning, R. G., The debate on data base management. EDP Analyzer, 10, 3 (March 1972).

9. Childs, D. L., Description of a set-theoretic data structure. AFIPS Conf. Proc., Vol. 33, Part 1, AFIPS Press, Montvale, N.J. 1968, 557-564.

10. Childs, D. L., Feasibility of a set-theoretic data structure: a general structure based on a reconstructed definition of relation. IFIP Congress 1968, North-Holland, Amsterdam, 1968, 420-430.

11. Childs, D. L., Extended set theory: a formalism for the design, implementation, and operation of information systems. STIS Corp., Ann Arbor, Michigan, 1973.

12. Codd, E. F., A relational model of data for large shared data banks. Comm. ACM, 13, (June 1970), 377-387.

13. Codd, E. F., Further normalization of the data base relational model. Courant Computer Science Symposia 6: Data Base Systems, Englewood Cliffs, N.J., 1971.

14. Codd, E. F., A data base sublanguage founded on the relational calculus. ACM-SIGFIDET Workshop on Data Description, Access and Control, ACM, New York, 1971.

15. Codd, E. F., Relational completeness of data base sublanguage. Courant Computer Science Symposia 6: Data Base Systems, Englewood, Cliffs, N. Y., 1971.

16. Codd, E. F., Normalized data base structure: a brief tutorial. Proc. 1971, ACM-SIGFIDET Workshop on Data Description, Access and Control, ACM, New York 1971.

17. Conference of Data Systems Languages (CODASYL) Data Base Task Group Report, ACM, New York, Oct 1969 and April 1971.

18. Conference of Data Systems Languages (CODASYL) Systems Committee, Feature analysis of generalized data management systems, ACM, New York, May 1971.

19. Date, C. J., Relational database systems: a tutorial. Information Systems, COINS IV, Plenum, New York, 1974.

20. Dodd, G. E., Elements of data management systems. Computing Surveys, 1 (June 1969), 117-133.

21. Earley, J., On the semantics of data structures. Courant Computer Science Symposia 6: Data Base Systems, Prentice-Hall, Englewood Cliffs, N.J. 1971.

22. Earley, J., Relational level data structures for programming languages. Acta Informatica, 2 (Feb. 1973), 293-309.

23. Earley, J., Towards an understanding of data structures. Comm. ACM, 14, (October 1971), pp. 617-627.

24. Goldstein, R. C., and Strnad, A. L., The MacAims data management system. ACM-SIGFIDET Workshop on Data Description and Access, ACM, New York, 1970.

25. Hardgrave, W. T., Accessing technical data bases using STDS: a collection of scenarios. ICASE Report 75-8, Hampton, VA 1975.

26. Hardgrave, W. T., BOLTS: A retrieval language for tree-structured data base systems. Information Systems, COINS IV, Plenum, New York, 1974.

27. Hardgrave, W. T., The prospects for large capacity set support systems imbedded within generalized data management systems. International Computing Symposium - 1973, North-Holland, Amsterdam, 1974.

28. Heath, I. J., Unacceptable file operations in a relational data base. ACM-SIGFIDET Workshop on Data Description, Access and Control, ACM, New York, 1971.

29. Howie, H. R., Salbu, E., Mass storage implementation approaches: a spectrum. Ampex Corporation, Sunnyvale, California, 1974.

30. IDMS Data Definition Langluage Mnaual. Cullinane Corporation, Boston, 1974.

31. IDMS Data Manipulation Language Manual, Cullinane Corporation, Boston, 1974.

32. Introduction to IBM 3850 mass storage system (MSS). Technical report number GS32-0028-1, IBM Corporation, Boulder, Colorado, 1974.

33. Krageloh, K. D., and Lockemann, P. C., Retrieval in a set-theoretically structured data base: concepts and practical considerations. International Computing Symposium - 1973, North-Holland, Amsterdam, 1974.

34. Kellogg, C.H., Designing artificial languages for information storage and retrieval. Automated Language Processing, Wiley, New York, 1967.

35. McIntosh, E., Large storage systems. DD/73/8, CERN, Geneva, March 1973.

36. Parsons, R. G., Dale, A. G., Yurkanan, C. V., A structure processing sub-language for data base management. TSN-28, Computation Center, The University of Texas at Austin, August 1972.

37. Parsons, R. G., Date, A. G., Yurkanan, C. V., Data manipulation requirements for data management systems. The Computer Journal, 17 (May 1974), 99-103.

38. Ray, F. B., Directed graph structures for data base management: theory, storage structures, and algorithms. Ph.D. Diss., University of Texas, Austin, 1972.

39. Salbu, E., Current status and outlook of MSS; the IBM Memory System. Ampex Corporation, Sunnyvale, California, 1974.

40. Schubert, F. F., Basic concepts in data base management systems. Datamation, 18 (July 1972) 42-47.

41. Schwartz, J. T., Abstract and concrete problems in the theory of files. Courant Computer Symposia 6: Data Base Systems, Prentice-Hall, Englewood Cliffs, N. J., 1971.

42. Schwartz, J. T., On Programming: An Interim Report on the SETL Proje Courant Institute of Mathematical Sciences, New York University, New York, 1973.

43. Set Theoretic Data System User's Guide. Set Theoretic Information Systems Corporation, Ann Arbor, Michigan, 1974.

44. Sibley, E. H., Taylor, R. W., Data definition and mapping language. Comm. ACM, 16 (December 1973) 750-759.

45. Skolem, T., Two remarks on set theory. Math. Scand. 5, 40-46.

46. Stonebraker, M., A functional view of data independence. ACM-SIGFIDENT Workshop on Data Description, Access and Control, ACM, New York, 1974.

47. Su, S. Y., Copeland, G. P., Lipovski, G. J., Retrieval Operations and data representations in contexted addressed disc system, SIGIR Forum, 9, (Jan. 1975), 144-160.

48. Suppes, P., Axiomatic Set Theory, Van Nostrand, New York, 1960.

49. SYSTEM 2000 Data Management System, Reference Manual. MRI Systems Corp., Austin, Texas, July 1973.

50. Whitney, V. K. M., A relational data management system. Informations Systems, COINS-IV, Plenum, New York, 1974.

51. Whitney, K. M., Fourth generation data management systems. AFIP Conf. Proc. 42, AFIPS Press, Montvale, N. J., 1973, 239-244.

52. Winter, R., Hill, J., Greiff, W., Further data language design concepts. Working Paper No. 8, Computer Corporation of America, Cambridge, Mass., 1973.