THE RELATIONAL APPROACH TO THE MANAGEMENT OF DATA BASES

MAC Technical Memorandum 23

Alois J. Strnad

April 1971

Work reported herein was supported in part by Project MAC, an M.I.T. research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract N00014-69-A-0276-0002.

PROJECT MAC

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

THE RELATIONAL APPROACH TO THE MANAGEMENT OF DATA BASES

MAC Technical Memorandum 23

Alois J. Stirmad

April 1971

PROJECT MAC

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

I. Introduction.

The ultimate goal of Project MacAIMS (MAC Advanced Interactive Management System) is to build a computer facility which will be able to support non-trivial decision making processes. (See reference 4). In the early stages of our experiments we discovered that traditional approaches to the management of data bases do not satisfy our needs. We have determined the following requirements for the management of Large Data Bases (LDB) in a dynamically varying environment such as an interactive Management Information System:

- high degree of flexibility
- data independence (i.e. programs are unaffected by changes of data representation or by addition of new data types)
- ability to operate on different data structures
- access path independent of data structure
- acces control below the file level
- uniform retrieval time and (possibly) update time

The relational approach to management of LDB offers a solution which satisfies most of the above requirements.

This paper is concerned primarily with our implementation of this approach. The theoretical basis for the relational approach to management of LDB can be found in recently published papers (1,2).

We are gradually implementing our system on MULTICS (Multiplexed Information and Computing Service). MULTICS is a major achievment in the area of multi-programming, multi-processing and time-sharing computer systems. It is a prototype of the computer utility, where "utility" is used in the same sense as the telephone or power utility. It can run seven days a week, twenty four hours a day, and it is capable of meeting a wide range of service demands.

Two aspects of MULTICS are very important for our implementation -- memory organization and access control. In MULTICS, the traditional hierarchy of storage -- ranging from core through disks, tapes etc.-- appears to the user as a single level, two-dimensional memory organization. Each user process has available up to 256,000 identical memories called segments, and each segment may contain up to 64,000 36-bit words. The entire range of this space (2^{34}) is directly addressable.

The other important aspect of MULTICS for our purposes is the facility for access control. Again, a

two-dimensional approach has been taken. Each segment has associated with it an access control list which identifies all the users authorized to use that segment and which specifies the mode of access for each user. The second dimension of access control is provided by concentric protection rings. Any attempt to access a segment from an insufficiently privileged ring (i.e. an outer ring) will cause a trap, and a routine supplied by the owner of the segment will be called to decide whether or not to permit the access.

Let me also mention that our system is entirely programmed in PL/1, the primary language on MULTICS and the language in which MULTICS system itself is written.

II. General Information.

We take the view that information we might store in our data base consists of sets of Data Elements (DE) and sets of relations among them. The basic set theoretic primitive operations are used for manipulating the Relation Data Sets (RDS).

Given Data Element Setss (DES) S1, S2,...Sn, corresponding Relation Data Sets consist of n-tuples (tuples

of degree n), each of which has its first element from set S1, its second element from S2, and so on. The Relation Descriptor (RD) is, in our terminology, the n-tuple composed of the names of the sets S1, S2, ...Sn. Suppose, for example, that there are Data Element Sets for persons' names, for addresses and for telephone numbers. We might construct an RDS which will represent the relations among members of these sets. The Relation Descriptor for this RDS will be the 3-tuple <person-name,address,telephone-number>. All other tuples will express the relation among the members of these sets. In our implementation, the relations are stored exclusively in terms of Reference Numbers.

The whole system is logically divided into two major parts. In the first, Data Element Sets are stored, Reference Numbers are assigned to the Data Elements, and operations are performed on DES's. In the second part of the system, Relation Data Sets are created and stored and basic set theoretic primitive operations are performed on them.

Reference Numbers (identification numbers) play an important role in our implementation. Whenever a new DE enters the system, it is immediately assigned a Reference Number. The RN is used for all subsequent operations on

that DE. The method used for storing and assigning RN's to DE's guarantees that the particular DE is stored only one time within the system. This fact is a significant contribution to our goal of storage and operational efficiency. In our particular implementation, RN's are all 36-bit quantities, which is the word length of the GE 645 computer on which MULTICS runs. Because the most frequent operations are performed on the Relation Data Sets and because all relations are expressed in terms of RN's, the operational efficiency of using the fixed length numbers is obvious. We also save storage space since the average length of a Data Element is almost always greater than one machine word.

Both parts of the system are described in detail in the following sections of this paper.

III. Data Element Sets.

In our system each category of data, such as name, address, telephone number, pointer, tuple, names of sets, etc. might be stored in one Data Element Set (DES). Note that tuples, pointers, etc. are treated in the same fashion as other categories. For example, persons' names can be

stored in one DES, and the name of this set is simply "person's name".

Moreover, the name of a set is just another data category. Therefore there is also one DES for names of Data Element Sets, with its own name being "set name". The names of all DES's are stored in this DES. One reason for this provision is that the system is self-descriptive -- i.e. all information about the data is stored within the system itself. Futhermore this descriptive data is treated exactly the same way as any other data. This is one significant advantage to our approach.

The total number of Data Element Sets depends on the particular application. For example, a Library Information System probably will have a DES for titles.

The reader might be somewhat confused at this point, but figure 1 and the rest of the paper will give a clearer explanation of the above concepts.

A Data Element Module (DEM) is a procedure which operates on a Data Element Set. Each DES may have its "own" DEM. We have defined the following functions of a DEM:

read - read the input stream and produce the Standard Form (SF). For example, the input to the DEM for dates might be "8, Febr., 1969". The SF returned by this

entry is "19690208". Data Elements can be stored in the DES in SF only.

write - write the given SF in natural, human-oriented form. Taking the Standard Form from the previous example, the output would be "February 8, 1969".

get reference number - the Reference Number corresponding to the given SF will be returned.

get data element - the SF corresponding to the given RN will be returned.

insert - the given SF will be inserted into the specified DES and RN assigned.

delete - the specified SF is deleted from the DES.

Corresponding functions of all DEM's exept "read" and "write" perform the same processing. Therefore a common procedure called a Data Strategy Module (DSM) may be invoked as an intermidiary between the DEM and the DES. However the mechanism for assigning the RN to the DE (actually to the SF) is different for each data type. Therefore there are different DSM's for each data type. The most freguently occuring data types are integers, character strings, and real numbers. For example, data categories such as salary or date have SF's which fall into the data type integer.

The Standard Forms of Data Elements are physically

stored in the DES in the form of binary trees. The main distinction between them is that for integers the SF and RN are identical.

Figures 2 and 3 show the mechanism for storing the SF's and assigning the RN's to them for both types.

As the Data Element Sets grow, the binary trees may become unbalanced, resulting in less efficient operation of the system. Programs which run in background mode handle this situation by rebalancing the trees.

We have also implemented two "special" Reference Numbers which are very useful. One is used to indicate a null, or absent DE. The other one acts as a "wild card"--that is, upon comparison with other Reference Numbers, it will match any RN regardless of value. In the following examples the null RN will be represented by "0" and the "wild card" by "*".

The left part of figure 1 illustrates the interaction between procedures and Data Element Sets. Notice that the DEM's provide the direct interface between the outside world and the part of the system described above.

IV. Relation Data Sets.

In section II the notion of Relation Data Set (RDS), Relation Descriptor (RD), relation, and tuple were introduced. In section III, the mechanism for keeping track of Data Elements (DE) and assigning Reference Numbers (RN) to them was presented.

The exact structure of the RDS's is irrelevant to the theory underlying the relational approach to data management. However, we believe that data (tuples in the case of RDS) should be stored using the representation most efficient for the particular application. Therefore the procedure, called a Relation Strategy Module (RSM), which operates on a particular RDS is designed for a particular data structure. Thus there is an RSM for lists, an RSM for trees, an RSM for arrays, etc. Note the difference between DEM's and RSM's: each DEM operates on Data Element Sets and is designed for a particular category of DE's; each RSM operates on Relational Data Sets, and is designed for a particular data structure.

The basic set primitive operations and several non-set theoretic operations are defined. These operations are performed by an RSM on a particular RDS. Before we describe

these operations in detail, we present examples of two RDS's. For the purposes of the following explanations, it is sufficient to represent RDS's as arrays, and the array elements as DE's. Keep in mind, however, that other representations are possible and that, in any case, the DE's are actually represented by RN's.

RDS (A)

name	city	prof.	sal.
Wels	Cambr	eng.	18000
Owens	Boston	prog	16000
Niles	Waban	prog	15000
Jones	Lynn	dir.	24000
Strand	Lexin.	clerk	12000

RDS (B)

name	phone	age
Jones	6205	30
Smith	5432	20
Strand	5857	34
Wyly	6669	42

The above RDS's have the following properties:

1. Each row represents an n-tuple.
2. All rows are distinct.
3. The ordering of rows and columns is immaterial.

In our implementation, all of the set primitive operations require two RDS's as input and produce a third

RDS which is the result of the operation. The other operations do not necessarily follow this same scheme.

In order to have the capability of creating RDS's and filling them with data we have had to define two "non-set theoretic" operations:

create set

Input: RD Output: RDS

Example: RD = <name,city,phone>

Result: RDS(C)

name	city	phone

The result is an RDS with one tuple -- the RD.

insert tuple

Input: RDS,tuple Output: RDS

Example: RDS(C), tuple = <Golub,Boston,888542>

Result: RDS(C)

name	city	phone
Golub	Boston	888542

The following operations are basic set primitives:

Intersection

Input: RDS,RDS Output: RDS

Example: RDS(A),RDS(E)

Suppose we created RDS(E)

name	city	prof.	sal.
*	*	prog	*

Result: RDS(F)

name	city	prof.	sal.
Owens	Boston	prog	16000
Niles	Waban	prog	15000

Note the use of the "wild card".

difference

Input: RDS,RDS Output: RDS

Example: RDS(A),RDS(F)

Result: RDS(G)

name	city	prof.	sal.
Wels	Camb	eng.	18000
Jones	Lynn	dir.	24000
Strand	Lexin	clerk	12000

projection

Input: RDS,RDS Output: RDS

Example: RDS(A),RDS(H)

Suppose we have RDS(H)

name	sal.

Result: RDS(I)

name	sal.
Wels	18000
Owens	16000
Niles	15000
Jones	24000
Strand	12000

join

Input: RDS,RDS Output: RDS

Example: RDS(A),RDS(B)

Result: RDS(J)

name	city	prof.	sal.	phone	age
Jones	Lynn	dir.	24000	6205	30
Strand	Lexin	clerk	12000	5857	34

composition

Input: RDS,RDS Output: RDS

Example: RDS(A),RDS(K)

Suppose, we have the RDS(K)

sal.	seniority
8500	2
32000	45
16000	10
12000	14
24000	16

Result: RDS(M)

name	city	prof.	seniority
Owens	Boston	prog	10
Jones	Lynn	dir.	16
Strand	Lexin	clerk	14

cartesian product

Input:RDS,RDS Output:RDS

Example: RDS(B),RDS(N)

Suppose we have the RDS(N)

affiliat.	bulding
MIT	0
MAC	E-58

Result: RDS(O)

name	phone	age	affilit.	bldg.
Jones	6205	30	MIT	0
Jones	6205	30	MAC	E-58
Smith	5432	20	MIT	0
Smith	5432	20	MAC	E-58
Strand	5857	34	MIT	0
Strand	5857	34	MAC	E-58
Wyly	6669	42	MIT	0
Wyly	6669	42	MAC	E-58

union

Input: RDS,RDS Output: RDS

Example: RDS(B),RDS(P)

Suppose, we created RDS(P)

name	phone	age
Sames	5555	52
Strand	5857	34
Buck	8765	64

Result: RDS(T)

name	phone	age
Jones	6205	30
Smith	5432	20
Strand	5857	34
Wyly	6669	42
Sames	5555	52
Buck	8765	64

In order to improve efficiency and to make the usage of the system simpler we have defined several redundant operations i.e. they can be performed by several calls of primitive operations.

sort

get successor

replace tuple

As has been previously mentioned, a Relation Strategy Module is designed to operate on a particular data structure. However, one of our requirements for management of data bases is to have the capability to combine and operate on data which have different representations. To accomplish this, each RSM has the capability to process not only the particular structure for which it is designed, but

also to operate on the cannonical form of an RDS. A RSM is able to reproduce its "own" type of RDS in the canonical form, which in our current implementation is a list.

V. Final Remarks.

We believe that our implementation of a relational system will give a solution for the management of Large Data Bases which will be able to support our overal research objectives. For experimental purposes, we have developed a simple user interface, and we are just now starting to move data into the system. We expect that during a relatively short period of time we will know enough about the behavior and efficiency of the system that we will be able to make substantial improvements. We should be able to report the experimental results of this preliminary system in late February 1971.

VI. List of Abbreviatios.

DE - Data Element

DES - Data Element Set

DEM - Data Element Module

DSM - Data Strategy Module

RDS - Relation Data Set

RD - Relation Descriptor

RDS - Relation Data Set

RSM - Relation Strategy Module

MULTICS - Multiplexed Information and Computing Service.

VII. References.

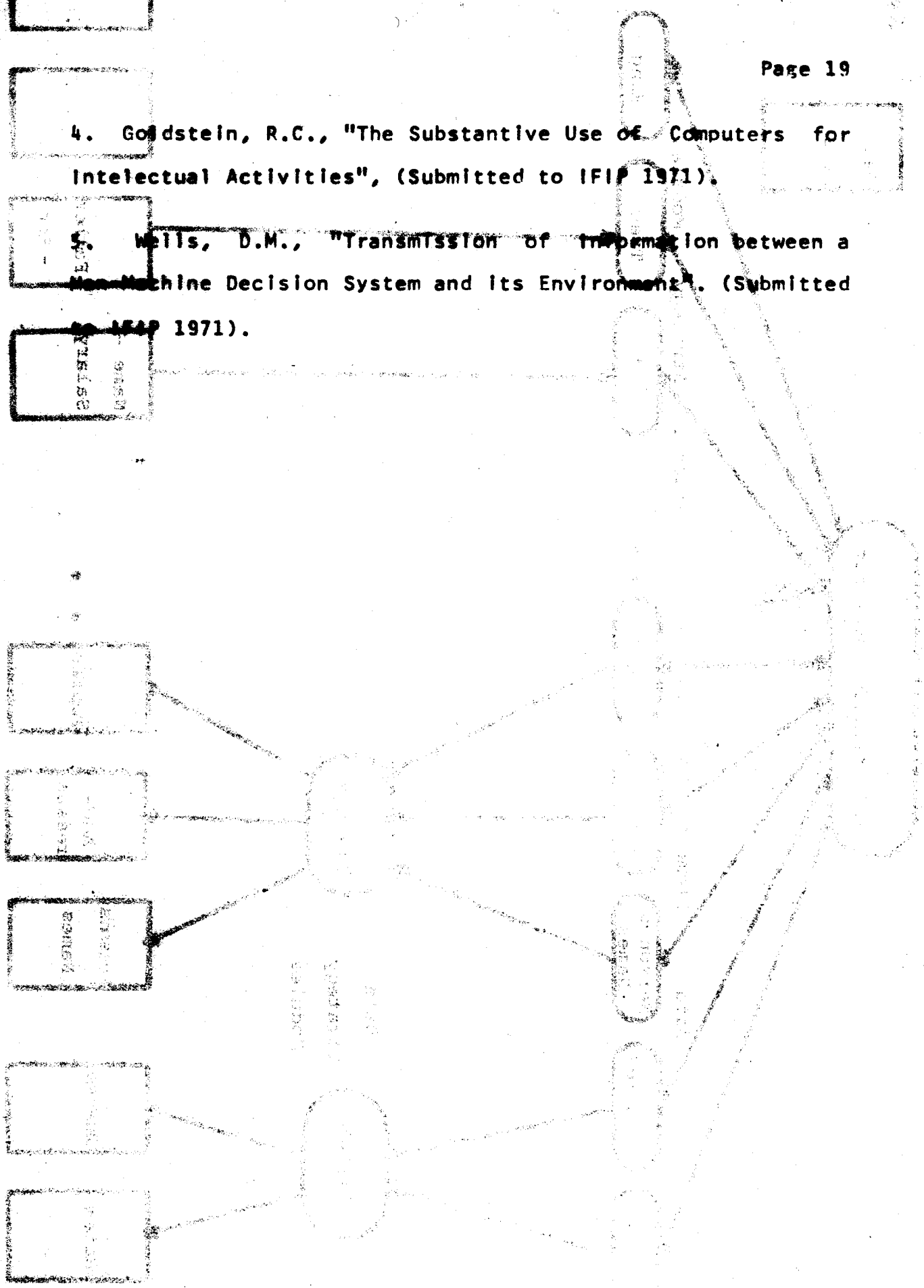
1. Codd, E.F., "A Relational Model for Large Shared Data Banks", Comm. ACM13,6 (June 1970),377-387.

2. Fillat, A.I., and Kraning, A.J., "Generalised Organisation of Large Data Bases; A Set Theoretic Approach to Relations", M.I.T. Electrical Engineering Dept., Master Thesis, June 1970.

3. Goldstein, R.C., and Strnad, A.J. "The MacAIMS Management System", Proceedings ACM - SIGFIDET Workshop 1970.

4. Goldstein, R.C., "The Substantive Use of Computers for Intelectual Activities", (Submitted to IFIP 1971).

5. Wells, D.M., "Transmission of Information between a Man-Machine Decision System and its Environment". (Submitted to IFIP 1971).



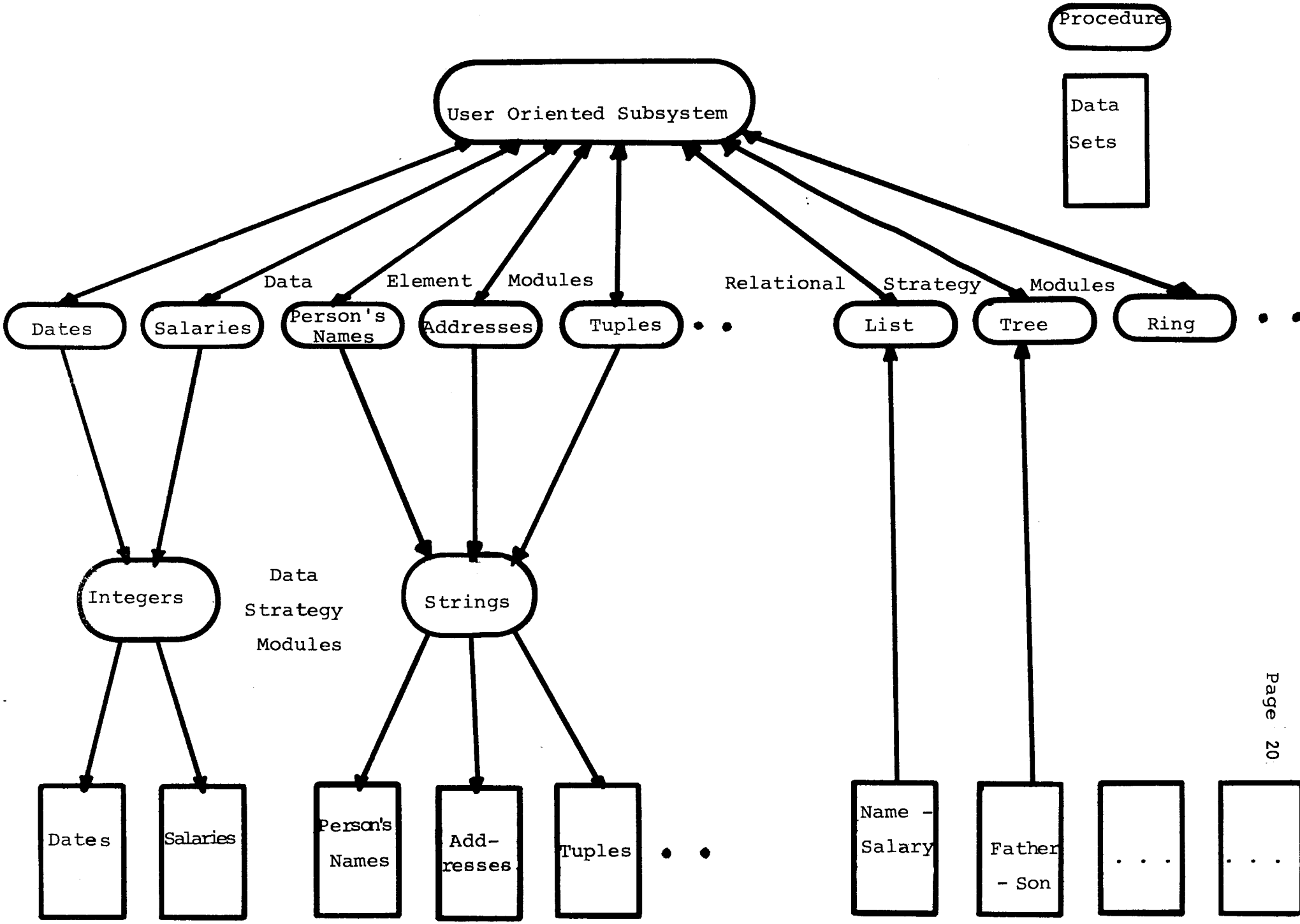
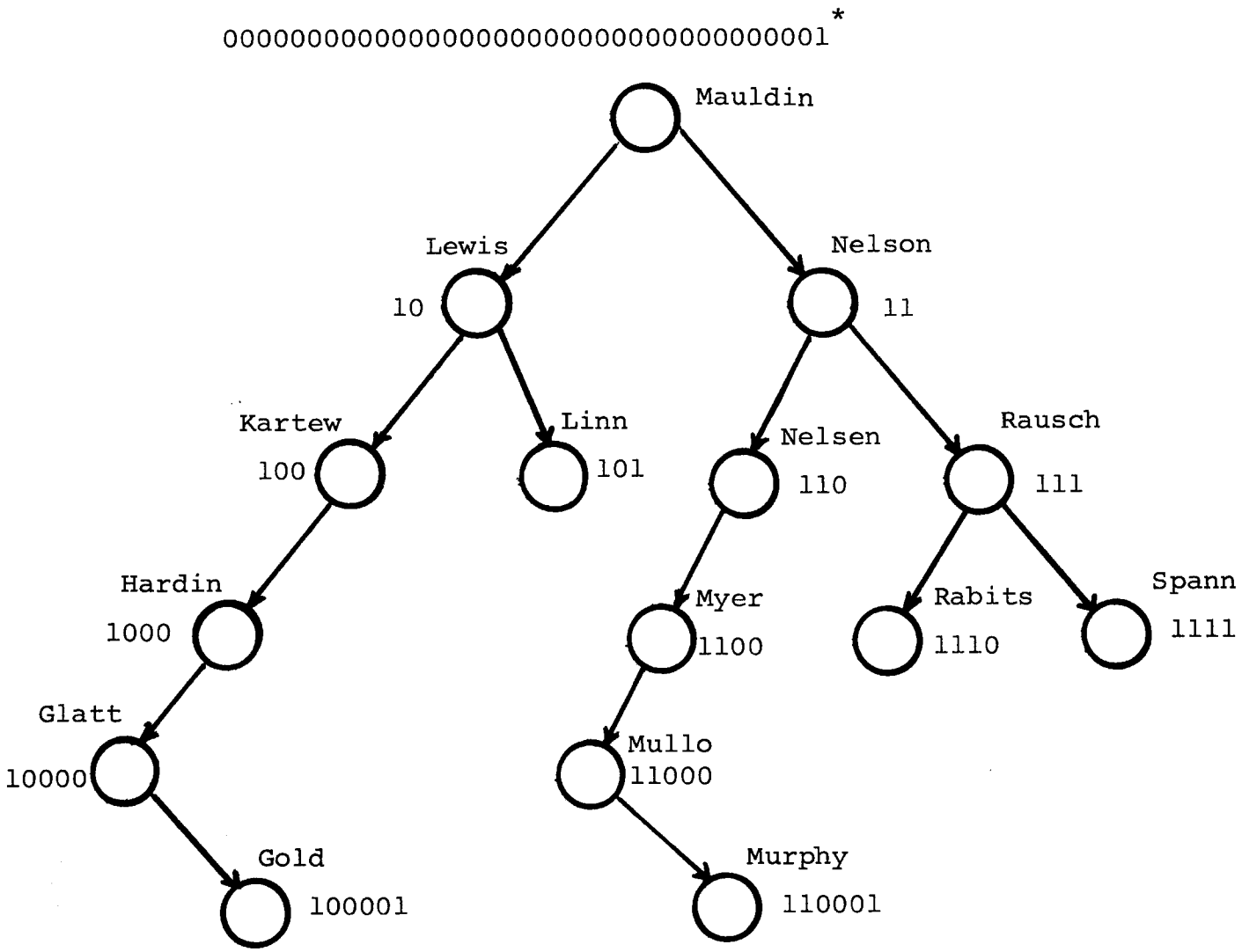


Figure 1 - System Overview

Binary Tree for Character Strings



*In succeeding reference numbers, all bits to the left of the flag have been omitted for clarity. First left-most "one" bit is used as a flag to indicate the start of significance.

Figure 2

Binary Tree for Integers

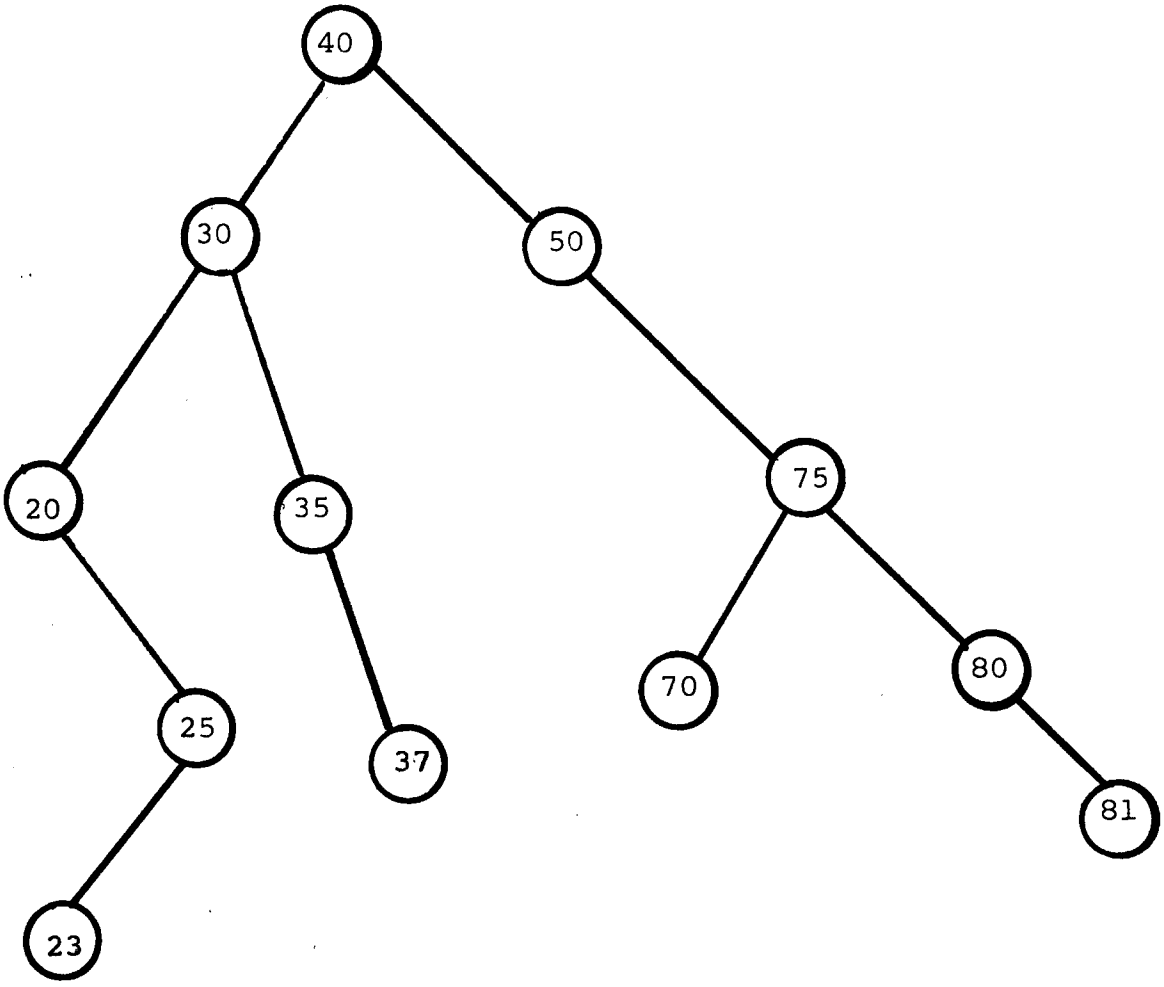


Figure 3

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Massachusetts Institute of Technology
Project MAC

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
None

3. REPORT TITLE

The Relational Approach to the Management of Data Bases

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Technical Memorandum

5. AUTHOR(S) *(Last name, first name, initial)*

Strnad, Alois J.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
April 1971	26	5

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
N00014-69-A-0276-0002	MAC TM-23
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency 3D-200 Pentagon Washington, D.C. 20301

13. ABSTRACT

This paper is concerned with the design and implementation of a relational system for management of Large Data Bases (LDB) at M.I.T., Project MAC.

We have determined the following six major requirements for the management of LDB in a dynamically varying environment, such as an Interactive Management System: high degree of flexibility; data independence; ability to operate on different data structures; access path independent of data structure; access control below the file level; uniform retrieval time.

We take the view that information we might store in our LDB consists of sets of data elements and sets of relations among data elements. The basic set theoretic operations are used for manipulating and operating upon these sets.

14. KEY WORDS

Data Management System	Data Manipulation	Management Information System
Information Retrieval	Access Control	Data Base Management
Data Structure	Data Organization	Set-Theoretic Data Structures

DD FORM 1 NOV 68 1473 (M.I.T.)

UNCLASSIFIED
Security Classification

MIT/LCS/TM-23

THE RELATIONAL APPROACH TO THE MANAGEMENT OF DATA BASES

Alois J. Strnad

April 1971