

AD-A116 593

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
INFOSAM: A SAMPLE DATABASE MANAGEMENT SYSTEM.(U)

DEC 81 B BLUMBERG

N00039-81-C-0663

UNCLASSIFIED

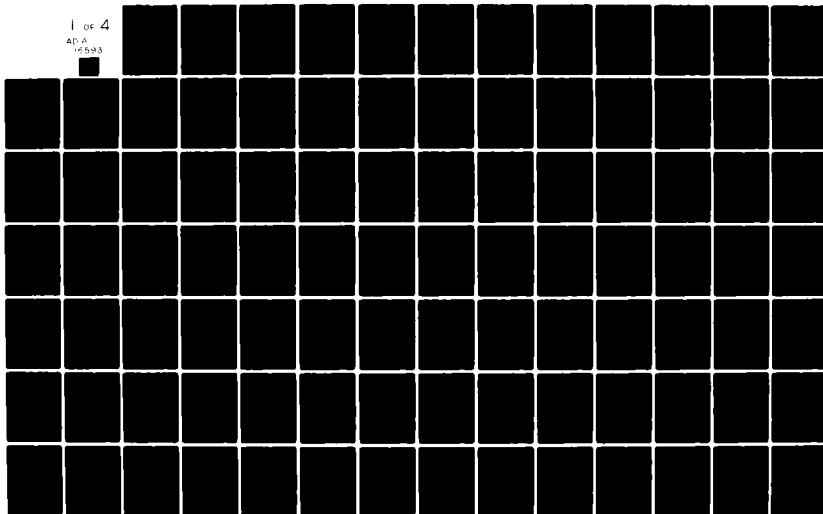
CISR-M010-8112-07

NL

1 OF 4

AD-A

16593

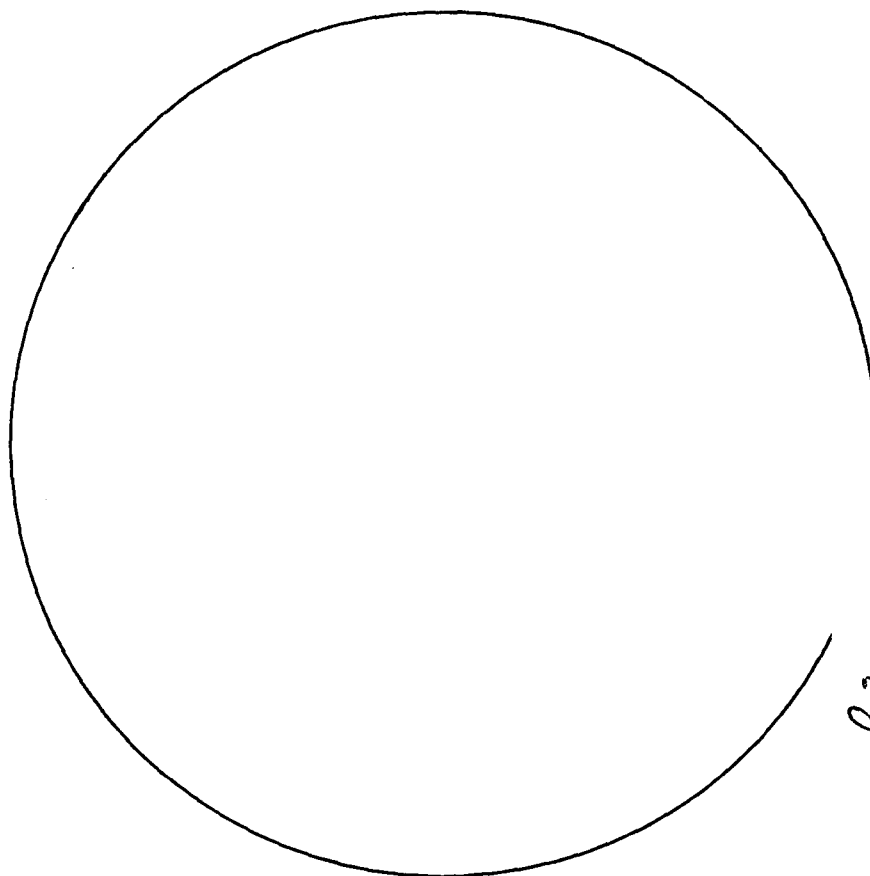


AD A116593

DTIC FILE COPY



12



DTIC
ELECTE

JUL 7 1982

Handwritten initials

DISTRIBUTION STATEMENT #

Approved for public release;
Distribution Unlimited

Center for Information Systems Research

Massachusetts Institute of Technology
Sloan School of Management
77 Massachusetts Avenue
Cambridge, Massachusetts, 02139

82 07 06 270

Contract Number N00039-81-C-0663 (MIT # 91445)
Internal Report Number M010-8112-07
Deliverable Number 2

12

INFOSAM:
A SAMPLE DATABASE MANAGEMENT
SYSTEM

Technical Report #7

By
Bruce Blumberg

December 1981

SECRET
JUL 7 1982
H

Principal Investigator;
Professor Stuart E. Madnick

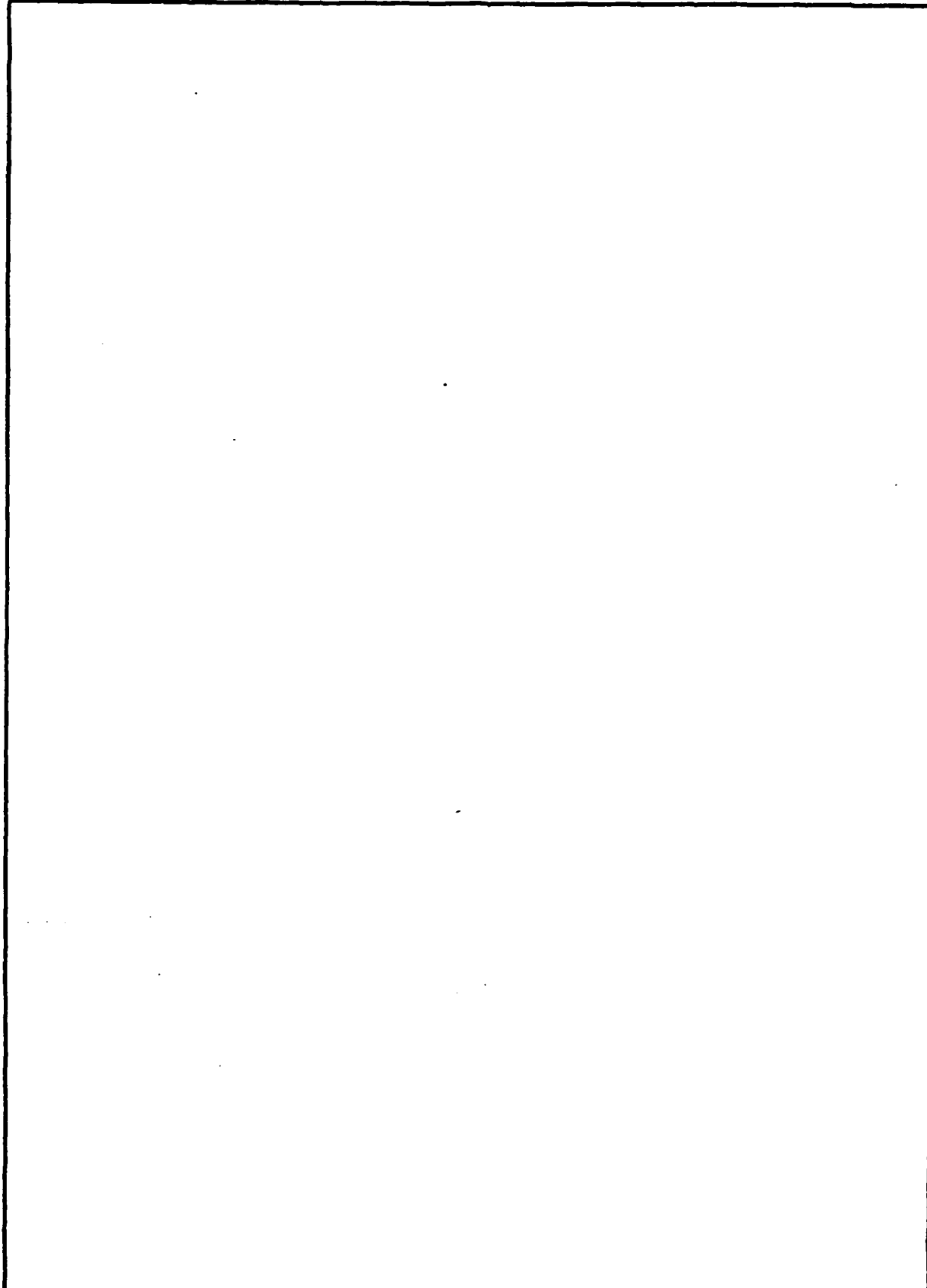
Prepared for:
Naval Electronics Systems Command
Washington, D.C.

DISTRIBUTION STATEMENT A

Approved for public release:
Distribution Statement A

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report #7	2. GOVT ACCESSION NO. AD-A115 593	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) INFOSAM - A Sample Database Management System		5. TYPE OF REPORT & PERIOD COVERED
7. AUTHOR(s) Bruce Blumberg		6. PERFORMING ORG. REPORT NUMBER M010-8112-07
9. PERFORMING ORGANIZATION NAME AND ADDRESS Sloan School of Management Massachusetts Institute of Technology Cambridge, MA 02139		8. CONTRACT OR GRANT NUMBER(s) N0039-81-C-0663
11. CONTROLLING OFFICE NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE December 1981
		13. NUMBER OF PAGES 331
		15. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) database computer, database management system, Software Test Vehicle, hierarchical system		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes the design and implementation of a relational database management system called INFOSAM. Its objective is to provide the INFOPLEX database computer project with a software test vehicle which could be used to gain insights into the Functional Hierarchy of the INFOPLEX database computer. Its design is largely based on Madnick's proposal for a hierarchically decomposed database management system. This implementation incorporates 3 levels which are primarily distinguished by their view of data.		

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)



SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

V

1

DTIC
COPY
INSPECTED
2

comprehensible, yet it illustrates many of the key features of a full scale DBMS. The second objective was to provide the INFOPLEX project with a software test vehicle which could be used to gain insights into the Functional Hierarchy of the INFOPLEX DBMS. Hence, where possible its design reflected the proposed design for the Functional Hierarchy.

This thesis is organized as follows: Chapter 1 provides an introduction to the area of Database Management Systems, and sets the stage for the remaining chapters. Chapter 2 describes the relationship of INFOSAM to the INFOPLEX concept. Chapter 3 is a detailed overview of the design and implementation of INFOSAM. Chapter 4 summarizes the preliminary implications of INFOSAM for the INFOPLEX design. A sample terminal session is included in Appendix 1 and the complete listings for the system are included in Appendix 2.

I would also like to extend my thanks to my father, who through his inspiration and support made Sloan a reality.

Finally, I would like to thank my wife, Janie, for her unfailing support and cheerfulness during this last year. I count myself very lucky to have such a wonderful wife.

TABLE OF CONTENTS

ABSTRACT	2
ACKNOWLEDGEMENTS	4
Chapter	page
1. INTRODUCTION	9
2. INFOSAM AND THE INFOPLEX PROJECT	16
The Role of INFOSAM as a Software Test Vehicle	17
An Introduction to INFOPLEX	20
Implications of INFOPLEX for the INFOSAM Design	25
Major Design Differences between INFOSAM and INFOPLEX	27
Concluding Remarks	30
3. LOGICAL OVERVIEW OF INFOSAM	32
Design Overview of INFOSAM	33
The Internal Level	37
The Primitive layer	39
Databases of the Primitive Layer	48
BEU	48
PSET_CAT	50
INDEX	56
Temporary Databases Built by Primitive Layer	57
The Modules of the Primitive Layer	58
DEFINEP	58
CREATEP	60
SEARCH	61
FETCH	64
CREATE_E	65
CREATE_I	65
HASH	66
INIT_P	66
The Binary Layer	67
Databases of the Bset Layer	75

BSET_CAT	76
Modules of the Binary Layer	78
DEFINEB	78
CREATEB	80
SELECTF	82
Concluding Remarks on Internal Level	83
THE NSET LEVEL	85
Overview of the Nset Level	86
Databases of the Nset Level	96
NSET_CAT	96
Inter-level Communication Databases	99
DEF_ARG	100
DV_ARG	101
INSERT_ARG	101
RET_ARG	102
DOM_RET	105
The FV_ARG database	107
Modules of the Nset Level	108
DEFINEN	108
DEFINEV	109
INSERTN	110
FETCHT	111
FETCHV	118
BUILD_C	119
NINIT	119
Summary of the Nset Level	120
THE EXTERNAL LEVEL	121
Logical Overview of External Level	122
Databases of the External Level	127
VTABLE	128
RTABLE	131
VTABLE	131
The Tl_ARG Database	131
Communication Databases	133
The Modules of the External Level	135
SAM	136
DEFINE	137
DEFDOM	137
DEFREL	138
DEFVIEW	139
QUERY	140
GETVIEW	141
SHVIEW	142
SHREL	142
SELECT	143
PROJECT	144
JOIN	145
PRINT	147

LOAD	148
LEXICAL ANALYSIS ROUTINES	150
Concluding Remarks on the External Level	151
4. THE LESSONS OF INFOSAM	153
INFOSAM and INFOPLEX - What Have We Learned	153
Potential Areas for Enhancement.	156
Changes in Design	157
Additions to the System	159
Concluding Remarks	162
Appendix	page
A. SAMPLE TERMINAL SESSION	164
B. LISTINGS AND DOCUMENTATION FOR THE INFOSAM SYSTEM	179
BIBLIOGRAPHY	330

Chapter 1

INTRODUCTION

Increasingly, information is being viewed as an important resource. It has been argued that the ability of a firm to effectively manage its information resource may provide it with an important competitive advantage in the coming years, particularly as other resources become increasingly more scarce. However, information is unlike any other resource available to the firm, in that the problem is not a scarcity of information but rather that there is often too much information. The problem has become one of organizing, and managing vast amounts of information, while at the same time shielding users from the complexity of managing the information, and allowing them to access information quickly and easily. This has necessitated the development of software and hardware tools with which to address this problem.

One such type of tool is the database management system (DBMS). A DBMS is specialized software system which is designed to shield the user from the complexity of the physical management of the information by acting as an interface

between the user and the data. While the DBMS is responsible for the actual physical management of the data, it usually provides the user with a capability to express actions on the database in terms of a conceptual data model, which may bear no resemblance to the manner in which the data is actually stored. For example, the DBMS may allow the user to view the database as a collection of tables. Any operation the user issues is expressed in terms of operations on these tables. It is the responsibility of the DBMS to translate operations expressed in such a manner into the corresponding operations on the database as it is physically implemented. The user never needs to know how the database is actually implemented. Hence, the user's view of the database need not change, even if the physical implementation changes.

There are other aspects of a DBMS worth noting, which focus more on the data management aspects. For example, the DBMS may provide security control, whereby only certain users may have access to particular types of data. On the otherhand, the DBMS may allow the sharing of data between different users. The DBMS may increase the reliability of the information. This results in part because the DBMS centralizes control over the data. This in turn may reduce the

redundancy of data, and hence the opportunity for inconsistent data values. In addition, the DBMS minimizes the redundancy of function among application programs, and this, in itself, increases reliability. A 100 different application programs can share the services provided by 1 search routine. Once that search routine has been debugged, there is no need to worry that application programs won't work because of an incorrect search routine.

There are currently many available Database Management Systems, and much work is being done in this area. For a description of some currently available systems see <Date> or <Cardenas> Key areas of research include, Concurrency control <Badal>, <Bernstein>; Multiple External views <Klug>; Security <Hsiao>; Database Machines <Madnick75, Madnick79, Madnick80>, <Hsaio77>; and Relational DBMS <Codd>, <Astrahan>, <Hall>. From the standpoint of a manager, a DBMS is probably the single most important systems program with which he need be familiar. Hence, it will become increasingly important for managers to understand database management systems.

This thesis describes the implementation of a sample database management system called INFOSAM. INFOSAM is a

small scale relational database management system, designed and implemented as part of a Master's thesis at MIT's Sloan School of Management. The design of INFOSAM is based on Madnick's design for a DBMS called INFOPLEX (see <Madnick75>, <Madnick78>, <Hsu>). This approach incorporates the concept of a hierarchical decomposition of a DBMS, into levels or collections of modules which share a common function or view of the data. The levels are hierarchically related such that a given level is largely implemented via services provided by the next lower level.

The objective of this thesis is twofold. First, it is intended to be a sample database management system that could be used as a teaching tool in a course at Sloan on database management systems. The goal was to implement a system which was small enough to make it comprehensible as a case study, yet be sophisticated enough to illustrate the key features of a DBMS. For example, it illustrates how the Ansi/Sparc <Ansi/Sparc> concept of shielding the user's view of the database from the internal implementation of the database through the use of a conceptual level, might actually be implemented. It illustrates the different data models of the different levels and how the mapping between levels is achieved. At the same time, it illustrates a

range of things that are somewhat mundane, but nonetheless important. For example, how a character string key may be hashed into a location, how a scatter table and overflow chaining may be implemented, or how a command line can be lexically analyzed.

Second, it is intended to be a software test vehicle for the INFOPLEX project. INFOPLEX is a proposed design for a database machine which incorporates the concept of hierarchical decomposition. <Madnick75, Madnick78, Hsu>. As part of the design process, a software prototype was needed in order to further explore design issues. Since, the INFOPLEX design incorporates much of the current thinking regarding the desirable design of a DBMS, it was felt that the objective of developing a sample database management system could be met by implementing a small scale software prototype of the INFOPLEX functional hierarchy. This would allow it to be used as a software test vehicle for the INFOPLEX project as well as for a teaching tool.

Why is INFOSAM important? At the outset it should be noted that INFOSAM was primarily an engineering effort, rather than a research effort. The conceptual design of INFOSAM was largely derived from the previous work of Mad-

nick <Madnick79>, Hsu <Hsu>, Astrahan <Astrahan> and Senko <Senko>. Nonetheless, there are several aspects of INFOSAM which are worthy of mention.

1. It is, if you will, a version 0 of the INFOPLEX functional hierarchy, and as such it provides an initial confirmation that the design of the functional hierarchy is basically sound.
2. A related point is that it clearly illustrates how the concept of heirarchical decomposition can be applied, in practice, to the design and implementation of a Database Management System.
3. The mapping technique used by the external level to process relational operations is potentially interesting and worth pursuing. While the concept is probably not unique to INFOSAM, it is probably one of the few actual implementations of such a technique.

The last point is that it has the potential to be a useful teaching tool. While it is somewhat complex, it could serve as a useful case study because it illustrates many key features of a full scale DBMS.

This thesis is organized as follows: In chapter 2 we discuss the context within which INFOSAM was developed so that the reader has some understanding why the system looks as it does. In particular, we will discuss the relationship of INFOSAM to the INFOPLEX project, its role in the project, how its role effected its design, and the major differences between INFOSAM and INFOPLEX. In chapter 3 we essentially walk the reader through the logical structure of INFOSAM. Here we discuss each level in INFOSAM in terms of its function, data model, databases and modules. The objective of this chapter is to not only give the reader an understanding of the logical structure of the system, but also illustrate how mapping between levels can be implemented and how the concept of a functional hierarchy has been implemented in the system. The last chapter summarizes what we have learned from the implementation of INFOSAM, both regarding the proposed design for INFOPLEX and for the current implementation of INFOSAM. In addition, there are 2 appendices. Appendix 1 illustrates a sample terminal session, and Appendix 2 contains the complete listings for the system.

Chapter 2

INFOSAM AND THE INFOPLEX PROJECT

While INFOSAM is a relational database management system in its own right, one of its primary objectives was to serve as a software test vehicle for the INFOPLEX project. That is, as a software prototype of a proposed hardware configuration. Hence, much of INFOSAM's design is taken from the proposed INFOPLEX design for a DBMS. In this chapter we will provide the reader with an understanding of the context within which INFOSAM was designed and implemented so that the reader has some understanding why the system looks as it does. We will begin by introducing the concept of a software test vehicle, what it is, and why it might be used. We will then provide a very quick overview of the INFOPLEX concept, and its relationship to INFOSAM. In the next section, we will outline design considerations that result from INFOSAM's relationship with INFOPLEX. Finally, we will highlight a few areas in which INFOSAM is different from INFOPLEX and why. This last section is aimed primarily at those who are familiar with INFOPLEX and want a quick summary of the major differences between the two systems.

2.1 THE ROLE OF INFOSAM AS A SOFTWARE TEST VEHICLE

A software test vehicle is best viewed as a software prototype of a system that may eventually be implemented all or in part via hardware. The software test vehicle (STV) is a collection of procedures which are organized in conceptually the same manner, and perform the same functions, as their hardware counterparts. Hence, the logical relationship and function of the modules is the same as in the proposed hardware configuration. In addition, the algorithms used by the software modules are as close as possible to those proposed for the hardware system.

The objective of a STV is to provide the system designer with a better understanding of the proposed system, and to allow him to test out various approaches before committing the system to hardware. Clearly, it is desirable to thoroughly understand a system prior to committing it to hardware, since mistakes at that stage can be very costly. A designer must understand not only the relationship of functional modules to each other but also the optimal internal structure of the functional modules. The STV concept aids the designer in both of these areas, by forcing him to implement a working software version of the system. Valuable lessons can be gained not only through the implementation

process, but also through a detailed performance analysis of the resulting system. The implementation process forces the designer to come to grips with how a particular function might actually be implemented. Often, the important but subtle implications of a design decision only come to light during the implementation process. In a similar fashion a performance analysis of the resulting system may bring to light issues that weren't readily apparent until the system was actually tested. Through the use of diagnostic code within the modules, detailed statistics may be collected and analyzed. Such measures may range from the number of times a particular module is called to service a request, to the amount of CPU time spent performing certain tasks. The designer can use these measures to identify better configurations, or more efficient algorithms.

However, an STV is not a static creation, and that is one of its great strengths. It provides the designer with a relatively flexible means of testing different configurations and different algorithms. Through the use of strictly defined uni-function modules, the designer can alter the configuration or modify the implementation of a particular module relatively quickly and easily. Thus, many different configurations can be tested and evaluated. This in turn

means that the system designer has little excuse for locking himself into a particular design without good reason.

Note, however, that the STV is just one stage in the design process. It is not a replacement for other stages of the process. Techniques such as Systematic Design Methodology (SDM) (see <Andreu>, <Huff>) are still essential for the preliminary design. Indeed, the STV should be based on the design suggested by SDM. Simulation techniques are also important tools for the designer, particularly when used in conjunction with STV. An STV can not be used to predict response times since the speed of the STV may bear no relation to the speed of the hardware configuration. For much the same reason a STV may not highlight bottlenecks in the system to the same degree that is possible through simulation techniques. Ideally, a simulation model would be used both before and after the use of an STV. Initially, it can be used in the preliminary design process, to give a rough idea of the system performance. During and after the STV stage the simulation model can be run again to give a much more accurate picture of the final system performance.

As the reader will see in the next section, INFOPLEX is a highly complex system, still very much in the design phase.

Ultimately, the system will be implemented via microprocessors. However, for many of the reasons discussed above there was a strong desire to build a software test vehicle to aid the design process. INFOSAM, was implemented, in part, to answer that need. Hence, much of the key design aspects of INFOSAM reflect the proposed design of the INFOPLEX system. In the next section, we will review the important aspects of the INFOPLEX design.

2.2 AN INTRODUCTION TO INFOPLEX

INFOPLEX was initially proposed by Madnick <Madnick 75> as a design for a database computer which incorporates the concept of a hierarchical decomposition of both function and hardware(<Madnick75>,<Madnick79>, <Hsu>). Hierarchical decomposition is a design and implementation strategy whereby a complex system is broken down into simpler subsystems, which are tightly defined, and hierarchically related to each other. That is, a given level, or subsystem is implemented making use of the services provided by the next lower level in the hierarchy. This approach has been shown to be effective in several software systems. (see <Madnick74> <Andreu77>,<Madnick79>) Indeed, a formal design process, called Systematic Design Methodol-

ogy(<Andreu77>,<Huff>) has been developed around this concept.

INFOPLEX is composed of 2 hierarchically related subsystems, each of which, in turn, is hierarchically decomposed into levels. This is shown in table 1. The storage hierarchy is responsible for all storage and device management (See <Madnick75> and <Madnick80> for a detailed discussion of the storage hierarchy). It is composed of a physical hierarchy of levels, where each level is composed of a type of storage device with a particular cost/performance tradeoff. Faster, but more expensive devices are at the top of the hierarchy, whereas cheaper, but slower, devices are at the bottom of the hierarchy. Information is moved automatically from level to level via algorithms implemented through microprocessors associated with each level. This movement of data is transparent to the functional hierarchy which sits on top of the storage hierarchy. As far as the functional hierarchy is concerned, memory consists of a huge virtual address space.

The Functional Hierarchy is responsible for all DBMS functions other than device management. It relies on the concept of a hierarchical decomposition based on both func-

TABLE 1
OVERVIEW OF INFOPLEX

FUNCTIONAL HIERARCHY	
*	Responsible for all DBMS functions except device and storage management.
*	Functional hierarchy of modules
STORAGE HIERARCHY	
*	Responsible for storage and device management.
*	Physical hierarchy of storage devices.

tion and view of the data (see <Madnick79> and <Hsu> for detailed discussions of the Functional Hierarchy). Levels in the hierarchy are identified either by a specific function, such as data validity checking, or by a particular view of the data, such as the stored form of the data, versus the external or user's view of the data. This approach is similar to other attempts to stratify the design of a DBMS based on either function or view of the data (see <Ansi/Sparc>, <Astrahan>, and <Senko>). However, Madnick's approach is unique in that each level in the hierarchy is implemented via 1 or more microprocessors, which in turn rely on services provided provided by modules in the next lower level.

The decomposition of a DBMS as proposed by Madnick makes a great deal of sense for a number of reasons. For one thing, most transactions which are processed by a DBMS call for a common sequence of tasks. Hsu writes in this regard:

For example, it may first be checked by a security control module; then it is passed to a name-mapping module which determines the records to be accessed; and then it is given to a search module which determines the address of the records; finally a storage module is invoked to obtain the record from memory. These stages strongly suggest a database system structure that reflects their sequence. Moreover, the modules which support the earlier stages of processing (e.g., security control and name mapping) also require the services provided by those modules that support the later stages of processing (e.g., searching and accessing. <Hsu>

Another motivation for a hierarchical decomposition based on data model is that it allows the user's view of the data to be independent of the actual implementation by the lower levels. This means a user's view of the data need not change if the implementation changes. In addition, multiple external views of the database (i.e. relational, hierarchical or network) can be supported if the conceptual data model is flexible enough to support these multiple views. The conceptual data model of INFOPLEX is that of the binary network. A binary network is composed of Entities and Attributes, where an Entity is an object which is described by a set of attri-

butes, which may be either atomic values, or entities themselves. Madnick has proposed that such a structure is capable, given semantic information, of supporting multiple views of the data.<Madnick79>

Finally, another motivation for such a structure is that by its very design a functional hierarchy tends to minimize the redundancy of functions within the system. This in turn may increase reliability since functions are isolated within specific modules and can be more easily tested and debugged <Parnas>. Further, if an error occurs the problem can be isolated to a single module rather than requiring changes to multiple modules all of which employ the same logic.

In summary, INFOPLEX is a proposed hardware configuration for a database computer. The Functional Hierarchy, is the subsystem responsible for database functions. It views itself as sitting on a huge virtual address space, and is not concerned with device management. The Functional Hierarchy is designed around the concept of a functional hierarchical decomposition of a DBMS. The system is decomposed into levels which are collections of modules which either perform a common, but level specific function, or share a common data model. In addition, a level maintains its own

level-specific database used to perform its function. Each level is implemented via a cluster of microprocessors. The data models chosen were chosen on the basis of supporting multiple external views of the data and a flexible physical data structure.

INFOSAM was developed, in part, to be a software test vehicle for the functional hierarchy. In the next section, we will discuss the implications of this for the design of INFOSAM.

2.3 IMPLICATIONS OF INFOPLEX FOR THE INFOSAM DESIGN

The potential role of INFOSAM as a software test vehicle for the functional hierarchy subsystem of INFOPLEX had certain implications for its logical design as well as for its actual implementation. These implications are outlined below.

The logical design of INFOSAM was to incorporate the concept of a functional hierarchy. On at least a conceptual basis, the modules of the system were to be combined into levels. Levels were to be identified by either a specific function or by a particular data model and level specific databases. Modules in one level, could only call modules in

the level immediately beneath them. Where possible the levels would correspond to the levels identified by Hsu, and would incorporate the same data models as proposed by Hsu.<Hsu>

Since the functional hierarchy's view of storage was that of a virtual address space, and it was not concerned with device management, it was decided that INFOSAM could be implemented using in-core storage only. That is, INFOSAM was not required to access information from external devices.

Where possible communication between modules was to be done via bit string messages. This reflects the potential development of a message handling facility which would oversee and monitor the communication between modules of different levels. In addition, where possible, no pointers would be passed between levels. The idea being that only the lowest levels should be aware of the physical location of the data.

In order to facilitate the use of the Software Test Vehicle, as well as to be consistent with the concept of a functional hierarchy, modules were to be strictly defined and unifunctional. The idea was that it should be easy to modify

or completely change a module and 'plug' it into the system. This also meant that a premium was to be placed on documentation standards.

2.4 MAJOR DESIGN DIFFERENCES BETWEEN INFOSAM AND INFOPLEX

There are several major design differences between INFOSAM and INFOPLEX. In most cases, the differences reflect limitations present in INFOSAM that would not be present in a full implementation of INFOPLEX. A few of the most important differences are described below. As mentioned earlier, this section is addressed toward those who are fairly familiar with the proposed design of the Functional Hierarchy.

One significant difference is in the number and definition of levels. As will be discussed in the next chapter, INFOSAM is composed of 3 levels, the Internal level, the Nset or Conceptual level, and the External level. The Internal level represents a union of Hsu's proposed Unary and Binary levels. The rationale for combining the levels was that in this implementation the Binary level was aware of the stored structure of the data, i.e. the BEU, and it required access to the equivalent of the unary set catalogue. As a result, it wasn't clear that the levels should be

separated. The Nset level is equivalent to Hsu's N-ary level, though on a reduced scale. The External level combines the View translation level, the View Enforcement level, and the Validity/Integrity level. The rationale for doing so was that only very primitive view enforcement and validity functions were implemented, so really our External level represents the View translation level with some view enforcement and data validity functions embedded in it.

A second major difference is in the type of data models supported by the system, in particular, by the External and Nset levels. Only a single external view is supported, that being a relational view. This is in contrast to the INFOPLEX concept of multiple external views. A relational view was chosen for several reasons. From a user's standpoint its conceptually easy to understand, it supports ad-hoc queries, queries can be expressed in a simple yet powerful query language, and it is somewhat non-procedural. In addition, it has the advantage that the user does not need to know how his database is actually implemented. From a system standpoint, the relational data model is the easiest of the 3 traditional external views to map to the conceptual data model of the Nset level.

The data model of the Nset level is a very restricted form of the binary network proposed by Hsu. In Hsu's proposed data model, an entity could have another entity as one of its attributes, and this would be implemented as a binary association between instances of the entities. <Hsu> This allows her data model to be 'rich in semantics', that is, to be able to support a variety of external data models. However, the implementation of such a data model appears to be very complex. A modified binary network was implemented in INFOSAM, in which entities can share attributes, but an entity can not explicitly (i.e. be linked via a binary association) have another entity as an attribute. In addition, a given define or insert request may only reference 1 entity set, although a retrieval request may involve several entity sets. This restriction allows the Nset level to be much less complex. Since, this model is very close to the relational model, it is fully capable of supporting a relational external model. While the define and insert logic would have to be enhanced a great deal to support Hsu's model, the retrieval logic will probably require relatively few changes.

The points discussed above represent the significant conceptual differences between INFOSAM and INFOPLEX. There are

a few less significant differences which are worth noting. For one thing, the distinction between levels is totally conceptual. There is nothing in the system which prevents a module from calling any other module in the system. However, as currently implemented, modules in a given level only call modules in the level logically beneath it. Another difference is the manner in which binary sets are implemented. INFOPLEX supports 3 possible methods, pointer chaining, physical duplication, and physical embedding. INFOSAM only supports pointer chaining. This restriction was made primarily to simplify the implementation. INFOPLEX also makes the distinction between associative pointers and set pointers, and allocates different parts of the basic storage unit to store each type of pointer. In INFOSAM they share a common area. Finally, INFOPLEX allows an element in a unary set, to be in more than 1 primary set. Since, INFOSAM does not make a distinction between unary sets and primary sets, an element can only be in 1 primary set, though it can be in several subsets.

2.5 CONCLUDING REMARKS

While, INFOSAM can be viewed as a prototype of a DBMS without regard to the INFOPLEX project, an understanding of

the context in which it was designed is important to understand why certain design choices were made. This chapter, hopefully, has given the reader a sense of the context within which INFOSAM was designed and implemented. With this understanding in hand, we can now proceed to a detailed overview of the design and implementation of the INFOSAM system.

Chapter 3

LOGICAL OVERVIEW OF INFOSAM

The purpose of this chapter is to provide the reader with an understanding of the logical structure of INFOSAM. As discussed in the previous chapter INFOSAM can be thought of as having three distinct levels. Each level has it's own data model, databases and collection of modules which are responsible for the definition, updating and retrieval of data items within the context of the level's view of the data. Our discussion of the logical structure of INFOSAM will center around a discussion of the separate levels. Thus for each level we will discuss, the conceptual data model used by the level, how the data model is implemented, the major databases used and maintained by the level, and finally, a brief overview of the major modules which comprise the level. In general, we will skim over the implementation of the modules unless the implementation provides a useful insight into the logical structure of the system. The reader who is interested in a detailed discussion of the implementation is urged to consult the documentation and listing at the end of this report.

3.1 DESIGN OVERVIEW OF INFOSAM

While the design of INFOSAM is largely based on design proposed by the INFOPLEX project, it should be noted that it also makes use of concepts proposed by <Ansi/Sparc>, <Senko>, and <Astrahan>. In particular, the choice of 3 levels distinguished by data model, while consistent with the INFOPLEX design, is also based on the work cited above. In addition, the conceptual design and use of a basic storage element to build complex data structures via the elements and links between elements is similar to the strategy proposed by <Senko>

INFOSAM incorporates 3 levels, an External level which supports a relational view of the database, an Internal level which is responsible for actually storing and accessing the data, and a Conceptual level which acts to insulate the External level from the Internal level. Each level has it's own data model or view of the data. The External data model is that of the Relation. This means that it not only supports relational operations issued by a user but it views its own databases as relations and performs relational operations to manipulate the data in its databases. The External level is independent of the logical or stored structure of the data, and need not be changed if the Internal structure

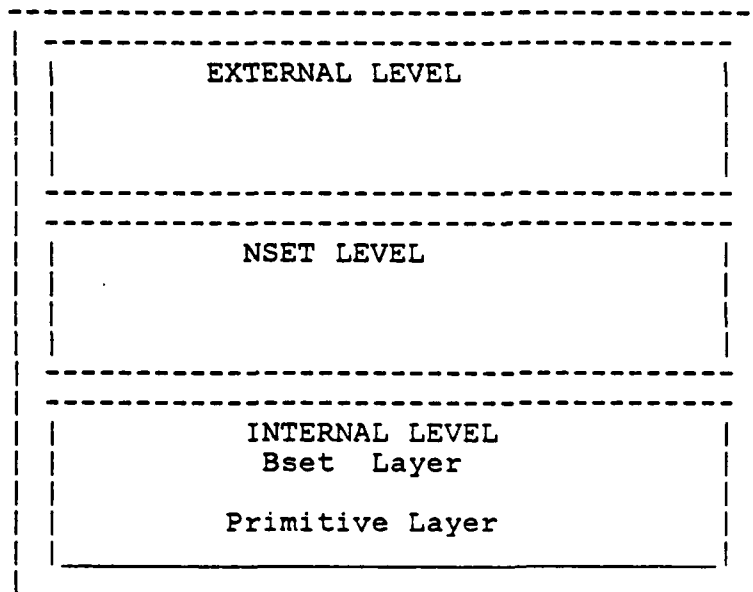
of the data changes. The Conceptual level's data model is that of the Entity set. An Entity set is composed of a collection of objects or entities which are described by a common set of attributes. The Conceptual Level's view of the database is as it is logically organized. That is, it is aware of the information contained in the database, and the logical relationship of among Entities within the database. However, it is not aware of the physical organization of the data, nor of how the External level views the data. As with the External level, the Conceptual level views it's databases within the context of its data model, i.e. as being Entity sets. The Internal level's data model consists of Primitive sets and Binary sets. A Primitive set is a collection of elements which share some common property. A Binary set is a collection of binary associations between elements within two Primitive sets. The Internal level's view of the database is as the database is actually stored and represented. That is, it is aware of how elements with Primitive sets are physically stored, and how the associations among elements within a Primitive set or a Binary set are physically represented. However, it is not aware of how higher levels view the data.

Both the External and the Conceptual levels implement their data models via calls to the level immediately beneath them. These calls reflect the conceptual data model of the level being called, but do not depend on how that data model is actually implemented. Hence, a given level need only be aware of the data model of the level immediately beneath it and be able to map its conceptual data model to it. This provides a high degree of data independence within the system. The Internal level is the only level which must be aware of how its conceptual data model is physically represented.

Table 2 provides a schematic view of INFOSAM. As can be seen from the table, INFOSAM is composed of 3 levels: the External level, the Nset level, and the Internal level. The Internal level is further decomposed into 2 layers: the Primitive or Pset layer and the Binary or Bset level. A level is distinguished by a) a level specific data model, b) level specific databases, and c) modules necessary to maintain its data bases and implement its data model (albeit via calls to modules in the level beneath it). The Internal level is broken down into 2 layers because while the layers share common databases and routines they use different data models. It should be noted that this decomposition is tran-

sparent to the Nset level, which only sees a unified Internal level. In keeping with the concept of the functional hierarchy modules in one level may only call modules in it's own level or the level immediately beneath it.

TABLE 2
Schematic Overview of Infosam



In the following sections we will discuss each level in term's of its data model and it's major responsibilities. Once we have presented this overview we will go back and discuss the implementation in greater detail.

3.2 THE INTERNAL LEVEL

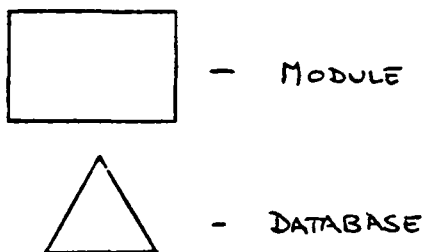
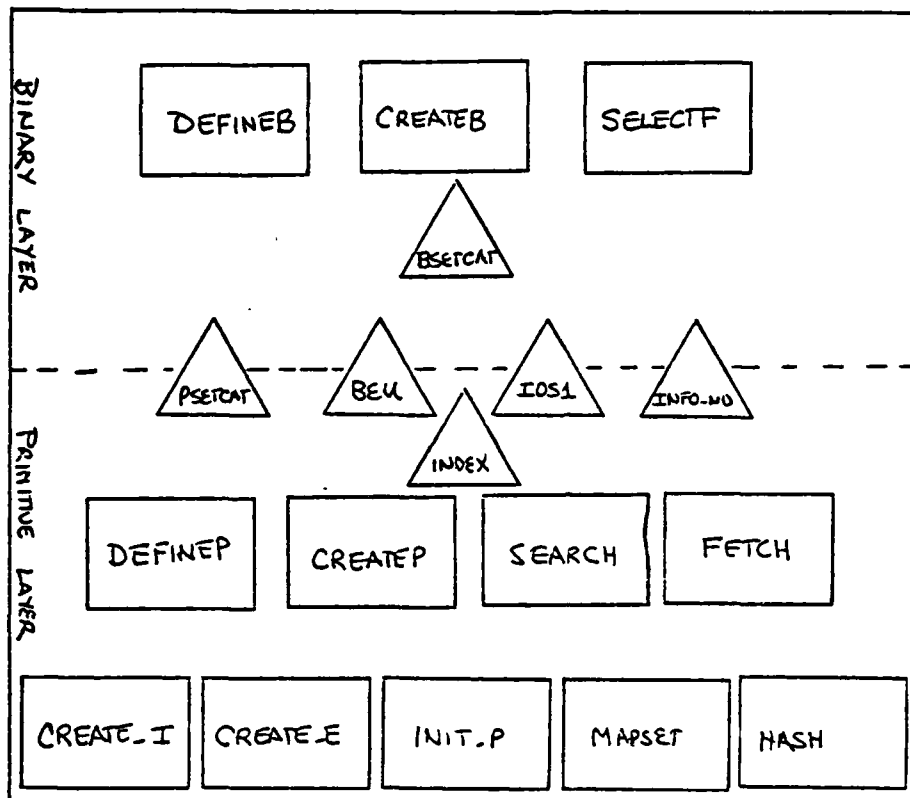
The Internal level is the lowest level in the system. It is concerned with the actual storage, and retrieval of data items, as well as maintaining associations among data elements which are either logically similar or logically related, and being able to retrieve associated data elements given a definition of the association. The Internal level shields the Conceptual level from needing to know how data items are stored or linked, and thus makes the Conceptual level somewhat independent of the actual implementation of the Internal level. At the same time the Internal level provides the Conceptual level with a data model which allows the Conceptual level to be able to express reasonably complicated data structures.

Table 3 provides a schematic view of the modules and databases of the Internal level.

The Internal level is composed of 2 layers which are hierarchically related, the Primitive or Pset layer and the Binary or Bset layer. The Primitive layer is unaware of the Binary level, whereas the Binary level relies heavily on the data model of the Primitive layer. While the layers have different data models, they share common databases and hence

TABLE 3

Modules and Databases of the Internal Level



do not warrant being distinguished as separate levels. Note, this is in contrast to the approach taken by Hsu <Hsu>, in which the Unary and binary levels are viewed as separate levels. As implemented here, both the Primitive and Binary layers require knowledge of the stored form of the data since linkage information is stored with the data element. In addition, they both share the same linkage information area in the basic storage unit. Hence, its not clear how the Binary layer could be isolated from the stored form of the data. However, in order to simplify our discussion of the Internal level we will discuss the layers separately.

3.2.1 The Primitive layer

The conceptual data model of the Primitive layer is the Primitive set. A Primitive set is a collection of elements which share some common property. For example, a Primitive set could be composed of a collection of supplier's names, the common property being that the data elements represent the names of your suppliers. Note, that the interpretation of a Primitive set is external to the set. That is, what you view as a set of supplier's names may be viewed by someone else as a Primitive set comprised of 8-byte character strings. A Primitive set can be either a Primary set, or a

subset. By a subset we mean that it is a subset of another Primitive set. For example, a set containing the names of all students at MIT might be considered a Primitive set as well as a Primary set. That Primitive set, in turn, contains a subset which represents the names of all students at the MIT Sloan School of Management. That subset is also a Primitive set in that it's elements share a common property. A Primary set is a Primitive set which is not a subset of another Primitive set. Conceptually, there is no reason why a subset could not also contain subsets. However, in this implementation, subsets can not contain subsets.

The purpose of the Primitive set layer is to accept calls to perform actions on its conceptual data model (i.e. define pset, insert pset, retrieve pset), map those requests to actions on the physical representation of its data model, and actually perform the required actions. Unlike the other levels, it is also responsible for implementing the physical representation of its data model. In the following section we will briefly discuss the logic of the Primitive layer and then elaborate on the layer's databases and routines.

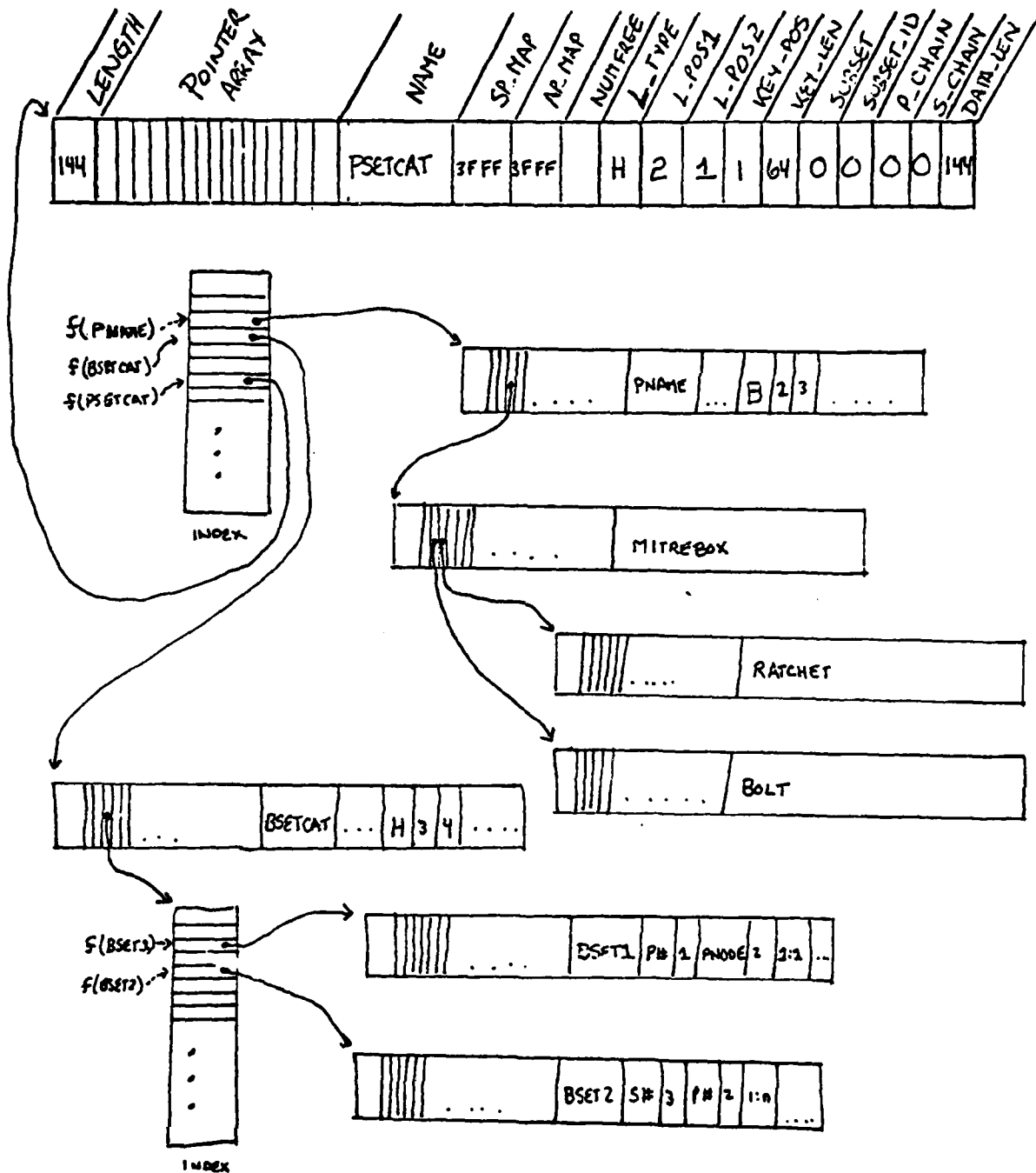
All stored information in the system is stored in based structures called Basic Encoding Units (BEU). A BEU consists

of a pointer array, which is used to hold linkage information, and a bit string data area which is used to hold the actual data. A Primitive set is implemented as a set of BEUs which hold the actual elements of the Pset and an additional BEU which contains information concerning the organization of the pset (i.e. how elements are linked, location of key field, etc.). The BEU which describes the Pset's organization is also part of a Pset called PSET_CAT which is a Primitive set consisting of all the BEUs which describe Psets in the system. Hence, that BEU is referred to as the PSET_CAT catalogue entry for the Pset. This is illustrated in Table 4. Any action on a Pset first requires the retrieval of the PSET_CAT catalogue entry in order to know how the Primitive set is implemented. The only exception to this is the define action. In this case the major task of the define module is to create the PSET_CAT entry for the pset.

Linkage among elements in a Pset is accomplished in one of three ways: (1) hashing via a scatter table and overflow chaining, (2) as a B-tree, or (3) via simple linear chaining. Each of these methods are illustrated in tables 5 - 7. If simple linear chaining is used, a pointer slot in the p_array of the Pset's BEUs is reserved to be used for Pset chaining. In addition, a pointer slot in the P_array of the

TABLE 4

Organization of the PSET_CAT Pset



catalogue BEU is reserved to point to the last element inserted into the Pset. When an element is inserted into the Pset the pointer to the last element in the Pset is placed in the reserved pointer slot of the BEU which contains the newest insertion to the Pset, and the pointer slot in the catalogue entry is updated to point to the new element. On retrieval, the catalogue entry is fetched and the pointer chain is followed until the desired element is found or until the pointer slot contains a null value.

B-tree linkage is very similar except that 2 pointer slots are reserved to be used for Pset chaining, and the pointer slot in the PSET_CAT entry for the Pset points to the first element in the set. When an element is to be inserted the existing B-tree for the Pset must be searched to find the node (i.e. BEU) to which the new BEU should be chained. Chaining is accomplished by setting the appropriate pointer slot in that node so that it points to the new element.

If hashing is to be used, a based structure is created at Pset definition time to act as a scatter table for the Pset, and a pointer slot in the catalogue BEU for the Pset is updated to point to the scatter table. In addition, a poin-

TABLE 5

Example of Linear Chaining

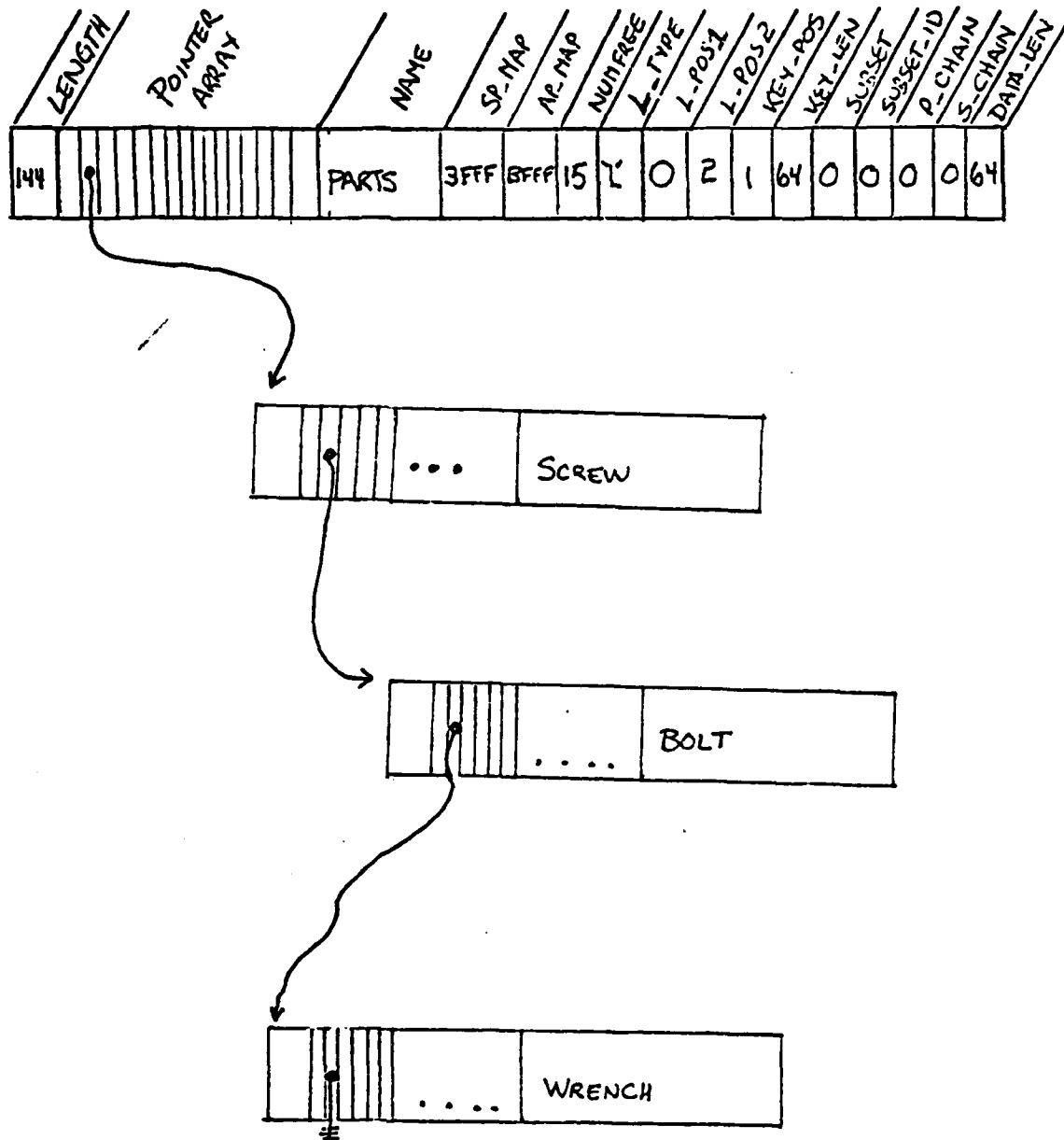


TABLE 6

Example of B_tree linkage

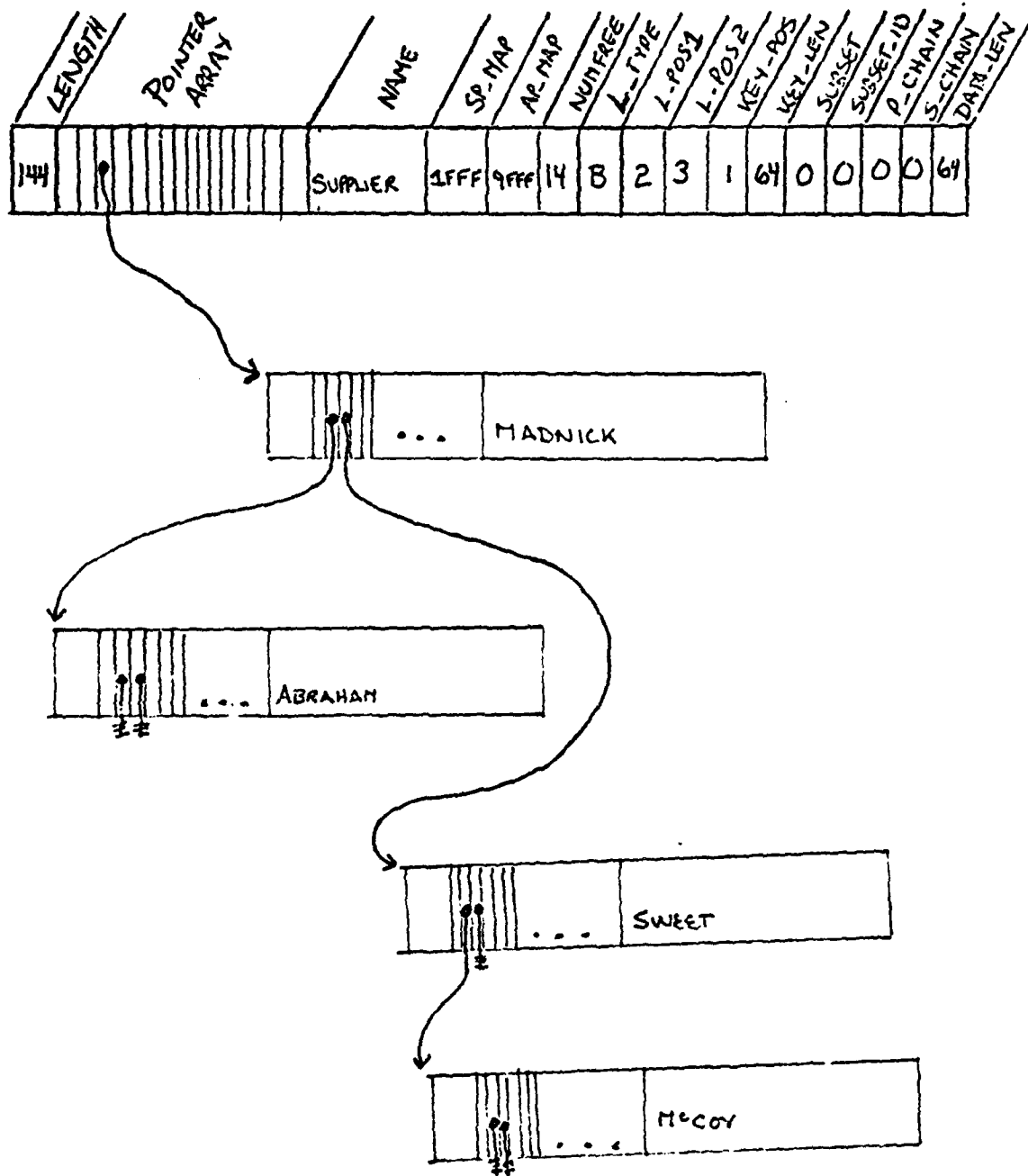
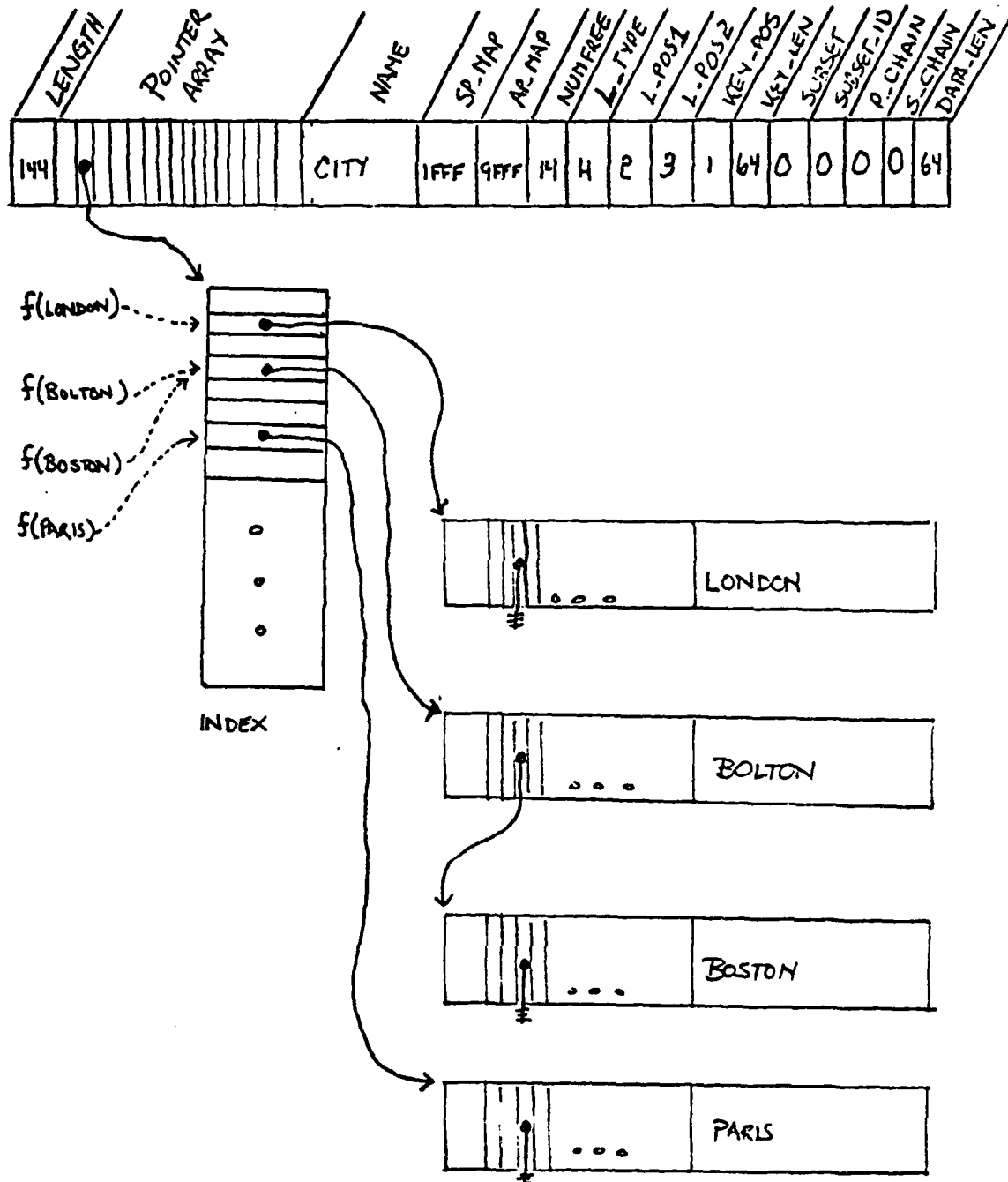


TABLE 7

Example of Hashing via Scatter Table and Overflow Chaining



ter slot is reserved in the Pset's BEUs to be used for overflow chaining. When an element is to be inserted the catalogue entry for the Pset is first fetched and the location of the scatter table is found. The key in the element to be inserted is then hashed via a system hash function and the corresponding element in the scatter table is checked to see whether it is null. If it isn't, then the element in the scatter table is updated to point to the new BEU in the Pset. Otherwise, the pointer slot in the new BEU which was reserved for overflow chaining is updated to point to the element specified in the scatter table, and the element in the scatter table is updated to point to the new BEU. On retrieval, the catalogue entry is fetched and the location of the scatter table is found. The key is then hashed and the corresponding location in the scatter table is checked. If it isn't null, then the contents of the BEU pointed to by the entry in the scatter table is checked to see if it matches the key. If it doesn't then the overflow chain is followed until either the desired element is found or a null value is found in the pointer slot reserved for overflow chaining.

The implementation of the Pset layer involves 3 major databases and 4 major modules. In the next section we will

briefly outline the structure of the databases and the logic of the modules.

3.2.2 Databases of the Primitive Layer

The Primitive layer makes use of 3 databases: (1) the BEU, (2) the PSET_CAT, and (3) the Index databases. These will be outlined below.

3.2.2.1 BEU

All stored information in the system is stored in based structures called Basic Encoding Units. A BEU contains the data value as well as linkage information. Formally, a BEU is declared as follows:

```
1 BEU BASED(ID),  
  2 LENGTH    FIXED BIN(15),  
  2 P_ARRAY(16) POINTER,  
  2 INFO BIT(320);
```

where the elements are interpreted as follows:

ID - Each allocation of a BEU has a unique ID, by which it can be referenced. This ID corresponds to the pointer value on which the structure is based at the time the structure is allocated. A BEU can be accessed by overlaying a BEU

structure on a location in memory pointed to by the ID. Note, however, that the ID is not stored in the BEU, but serves as an External reference to it.

LENGTH - The length field is used to specify the length of actual data in INFO. Since Info is a fixed length string, it may contain a data element whose length is less than that of Info. Hence, the need to specify the length. Note, in a later implementation the length of Info would probably be variable and LENGTH in that case would be the length of the Info field.

P_ARRAY - This is a pointer array which is used to contain linkage information. Both the Pset and Bset layers use the pointer slots in P_ARRAY to implement their respective linkages. Typically, at set definition time (be it a Pset or a Bset) the definition module reserves a pointer slot to be used to implement the linkage being defined. By reserve, we mean that a flag is set in the PSET_CAT entry for the Pset indicating that that pointer slot in BEUs containing elements of the Pset is reserved for a specific purpose. In addition, the PSET_CAT or BSET_CAT entry will contain information on how the contents of the pointer slot are to be interpreted. The P_ARRAY is of fixed size to simplify

implementation. However, this does limit the number of link-ages that can be implemented.

INFO - This is a fixed length bit string which holds the data to be stored. A bit string representation was chosen to make the BEU generalizable and to allow common routines to be used regardless of the type of information contained in the BEU. It is fixed length to simplify implementation.

The BEU is a powerful structure that allows a single data element to take on more than one meaning through it's link-ages. In addition, it supports the creation of complex and varied data structures, and once again a single instance of a data element may be a part of different data structures. It is not without it's drawbacks, however. In particular, a single BEU, as implemented in this system, requires over 100 bytes regardless of the amount of information it contains. If, however, the size of the pointer array and the data area was determined dynamically, then this problem would be somewhat reduced.

3.2.2.2 PSET_CAT

Every Primitive set defined by the system contains a BEU which describes the implementation of the Pset, i.e. how the

BEUs which contain the elements of the Pset are linked and how they are to be interpreted. This BEU is in turn a member of a Primitive set called PSET_CAT which is the set of all BEUs used to describe the implementation of Psets in the system. That set is in fact a catalogue of all the Primitive sets in the system. The entire BEU is used as the catalogue entry by overlaying the following structure on any BEU which contains a catalog entry.

```
1 CAT_ENTRY BASED(P),
  2 LENGTH FIXED BIN(15),
  2 P_ARRAY(16) POINTER,
  2 DATA
    3 NAME BIT(64),
    (3 SP_MAP,
      3 AP_MAP) BIT(16),
      3 NUMFREE BIT(8),
      3 SEARCH_INFO
        (4 L_TYPE,
          4 L_POS1,
          4 L_POS2,
          4 KEY_POS,
          4 KEY_LEN ) BIT(8),
      3 SET_TYPE
        (4 SUBSET,
```

```
4 SUBSET_ID,  
4 P_CHAIN,  
4 S_CHAIN) BIT(8);
```

where the elements of the catalogue are interpreted as follows (Note, the reader may find it useful to refer to tables 5 - 7 during this discussion for examples of how the fields of the catalogue are in fact used.):

LENGTH - Used to specify the length, in bits, of the catalogue entry.

P_ARRAY - Contains linkage information used to link the catalogue entry to the elements in the pset (either directly or via a pointer to a scatter table), or to link the catalogue entry to other related catalogue entries (for example, all catalogue entries for a Primary set and any subsets of it are chained together). The contents of the pointer slots within P_ARRAY are interpreted depending on the contents of Search_Info.

NAME - The name of the Pset being described.

SP_MAP & AP_MAP - These bit strings are used as maps to indicate the status of the pointer slots in the P_ARRAYs of the BEUs within the Pset. Each map has 16 bits correspond-

ing to the 16 pointer slots in a P_array. A '1'b indicates that the pointer slot is available for use, whereas a '0'b indicates that the pointer slot has been reserved. When the Pset definition module needs to reserve a pointer slot in the BEUs it calls a routine which finds the first '1'b in the SP_MAP, sets it and the corresponding bit in AP_MAP to '0'b and returns the position of the bit to the define routine. This represents the position of the pointer slot being reserved. A similar procedure is used to reserve pointer slots in order to implement linkages in BSETs, except that the AP_MAP is searched first. The reason 2 maps are used rather than 1 is that certain pointer slots are not available for Pset linkage, but are available for Bset linkages. For example, pointer slot 1 is used for overflow chaining in the PSET_CAT Pset and is unavailable for other Pset linkages, but it is available for Bset linkages. This non-symmetry results from the fact that the position of the pointer slots linking the PSET_CAT entries to their elements is the same as the position of the pointer slots linking the elements to each other. Since pointer slot 1 is already allocated to overflow chaining within the PSET_CAT Pset, it can not be used to chain catalogue entries to their elements.

NUMFREE - number of available pointer slots left in BEUs of Pset.

L_TYPE - Specifies the access method used to locate elements within the Pset. This can be either Hashed, B_tree or linear chaining.

L_POS1 - Specifies pointer slot used to implement the Pset linkage among BEUs which contain elements of the Pset. Exact meaning depends on the type of linkage implemented. If hashed, then refers to pointer slot in PSET_CAT entry which points to the scatter table. If B_tree, then refers to the pointer slot in the BEUs of the set used to chain right descendants of the B_tree. If linear used, L_POS1 not used.

L_POS2 - also used to specify pointer slots used for Pset linkages. Exact meaning also depends on type of linkage implemented. If Hashed, then it is used to specify pointer slot used for overflow chaining. If B_tree, then it is used for a dual purpose. First, it specifies the pointer slot in the PSET_CAT entry used to point to the first element in the Pset. Second, the same pointer slot is used to chain left descendants among BEUs of the Pset. Finally, if Linear, then it specifies the pointer slot used for chaining the PSET_CAT entry to the last BEU in the set as well as for chaining BEUs of the Pset together.

KEY_POS - specifies starting location of the key within the INFO field of the BEUs containing the data elements.

KEY_LEN - specifies length of the key within the INFO field.

SUBSET - flag to indicate if this Pset is a Primary set or a subset.

SUBSET_ID - If Pset is a subset, then this element indicates the pointer slot used for chaining elements of the subset together. As currently implemented, subset linkages are all implemented via linear chaining. Note, if subsets are exclusive, then subsets can share a common subset_id.

P_CHAIN - If Pset is a subset, then this element specifies the pointer slot in the PSET_CAT entry used to point to the Primary Pset PSET_CAT entry. This information is required to implement insertions and deletions correctly within both the Primary set and it's subsets.

S_CHAIN - If Pset is a subset, then this element is used to specify the pointer slot in the PSET_CAT used to chain catalogue entries for all subsets within a given Primary set. Simple linear chaining is used.

3.2.2.3 INDEX

As discussed earlier, hashing is implemented via a scatter table and overflow chaining. When the link type is specified as hashed, the Pset definition module allocates the following structure to act as a scatter table for the Pset:

```
1  INDEX BASED(INDEX_PTR),  
   2  NAME_ENTRY BIT(64),  
   2  TEST_LEN FIXED BIN(15),  
   2  PTR_TO_ENTRY(50) POINTER;
```

Where the elements are interpreted as follows:

INDEX_PTR - specifies the pointer value on which this structure is based. Once allocated the value of INDEX_PTR is placed into the L_POS1 pointer slot in the catalogue entry for the Pset.

NAME_ENTRY - indicates the name of the Pset for which this is a scatter table. Note, this field serves no real purpose and could be eliminated in a later implementation.

TEST_LEN - specifies the length in bits of the key field in the BEUs of the pset to be used in determining the hash value.

PTR_TO_ENTRY - This is a pointer array which acts as a scatter table for the Pset. An element of this pointer array is null or points either to the first element whose key hashed to that location, or to the beginning of an overflow chain of BEUs which hashed to that same location. For efficiency reasons, elements are inserted at the beginning of the overflow chain rather than at the end. Hence, the beginning of the overflow chain is, in fact, the last element to be added to that overflow chain.

3.2.2.4 Temporary Databases Built by Primitive Layer

There are two other databases of significance which the Primitive Layer builds in order to return values to higher layers or levels. These databases are temporary stacks built by the Primitive layer and destroyed by the upper levels once they have examined the contents of stack. The first of these databases is called IDS1, and it is used to return the IDs of elements found by the SEARCH module. It is declared as follows:

IDS1 POINTER EXTERNAL CONTROLLED;

The second of these databases is called INFO_ND, and it is used to return the data portion of BEUs within a Pset. It is created by the FETCH module. Its formal declaration is as follows:

INFO_ND BIT(320) EXTERNAL CONTROLLED;

A stack structure was chosen because of the dynamic nature of the number items that might be returned, and because implementation via the controlled attribute makes management of the databases fairly simple.

3.2.3 The Modules of the Primitive Layer

The Primitive layer consists of 4 major modules, and 5 support modules. These modules are responsible for the physical implementation of the Pset data model. In this section we will briefly outline the purpose of each module, and where appropriate discuss the logical structure of the modules. The reader is urged to consult the documentation in Appendix 1 for a detailed discussion of the Internal structure of the modules.

3.2.3.1 DEFINEP

This module is responsible for creating a PSET_CAT catalog entry for each Primary set and Subset defined in the system. In addition, it is responsible for reserving pointer slots to be used for Pset linkages, as well as creating support structures if necessary (i.e. INDEX if the Pset is to be hashed). In order to define a Pset, DEFINEP requires par-

ameters which specify the name of the Pset, the desired access or linkage method, the position and length of the key, and if it is a subset of a Primary set it needs to know the name of the Primary set and the pointer slot to be used for subset linkage. The logic of DEFINEP is fairly simple. If this is the first Pset to have been defined, the INIT_P module is called which defines the PSET_CAT Pset and inserts a catalogue entry into the Pset which describes the organization of the PSET_CAT. Otherwise, a temporary structure is allocated to serve as a template for the new catalogue entry. If the Pset to be defined is a Primary set, then this template is simply a copy of the PSET_CAT entry describing the PSET_CAT Pset. Otherwise, the template is a copy of the PSET_CAT entry for the Primary set for which this is a subset. In this manner the Subset organization is made to reflect the organization of the Primary set. The module then proceeds to build the catalogue entry to reflect the parameters passed to it. If pointer slots need be reserved, then the MAPSET module is called to analyze the appropriate map in the catalogue entry, find the first available pointer slot, and return that value as well as update the maps. Once the catalogue entry is built, the Data portion of the catalogue entry is passed to CREATEP which is responsible

for actually creating the BEU which will contain the catalogue entry, and for inserting that BEU into the PSET_CAT Pset. The final task is to update the pointer slots of the newly created BEU. If the link type is hashed, then the CREATE_I module is called to create a scatter table for the Pset, and a pointer to that scatter table is placed into the appropriate pointer slot in the catalogue entry. If the Pset is a subset, then it is also necessary to chain the subset definition to both the Primary set definition and any other subset definitions for that Primary set. In addition, the SP_MAP and AP_MAPs of the Primary set must be updated to reflect any additional pointer slots which are no longer available for use.

3.2.3.2 CREATEP

This module is responsible for inserting an element into a previously defined Pset, given the name of the Pset and a bit string representation of the data. In order to accomplish this, this module must perform several tasks. First, it must retrieve the PSET_CAT entry for the Pset. This is done via a call to the SEARCH module, passing it PSET_CAT as the Pset to search and the name of the Pset as the key to search on. The next task is to create a BEU which contains

the bit string representation of the data. This is accomplished via a call to the CREATE_E module which actually creates a BEU and inserts the data into the newly created BEU. The final task is to insert the BEU into the Pset in accordance with the link type specified in the PSET_CAT entry for the Pset. This is done via a call to the CHAIN module which inserts the BEU into the Pset, taking into account the organization of the Pset, and the contents of the BEU to be inserted. If the Pset is a subset, then CHAIN is called a second time to insert the BEU into the Primary set.

3.2.3.3 SEARCH

This module is responsible for retrieving the IDs of one or more elements in a Pset given the name of the Pset, the retrieval mode and a key to search on if necessary. The SEARCH module supports 3 modes, 1) First element in set which matches key, 2) All elements in set which match key, and 3) All elements in set. It returns a stack which contains the IDs found. The SEARCH module is, in fact, composed of 6 specialized Internal search routines which are called depending on the Pset organization and the retrieval mode. The logic of the SEARCH module is as follows. The first task

is to fetch the PSET_CAT entry for the Pset. This is done by first retrieving the PSET_CAT entry for the PSET_CAT Pset, and then by calling the appropriate search routine given the organization of the PSET_CAT specified in its catalogue entry and the name of the Pset. If the mode is either (1) or (2) then the appropriate search routine is called (i.e. L_SEARCH, B_SEARCH, or H_SEARCH). L_SEARCH is a simple linear search which returns the ID of the first element which contains a match with the key. It, like all of the search modules, relies on the PSET_CAT entry for the Pset for information concerning which pointer slots are used for chaining, how the contents of those pointer slots are to be interpreted, and the position and length of the key. B_SEARCH performs a Binary search of the B_tree pointed to by the PSET_CAT entry for the Pset, and returns the ID of the first element in the Pset which contains a match with the key, or a null value if not found. H_SEARCH performs a hash search, using the scatter table pointed to by the PSET_CAT entry for the Pset, and the pointer slot designated in the catalogue entry to be used for overflow chaining. It uses the system HASH function to hash the key into the scatter table. If the corresponding entry in the table is not null, then it checks the contents of the BEU pointed to by

that entry to the key. If they are equal it returns the ID of that BEU. Otherwise, it performs a linear search of the overflow chain until it either finds a match or a null pointer slot.

If a match is found and the retrieval mode is (1) then the ID found is placed on the top of the stack and SEARCH returns. If a match is found but the mode is (2) then a linear search is used to retrieve any additional elements. If the link type is Hashed, then only the remainder of the overflow chain need be searched. If the link type is a B_tree, then only the right descendents need be searched until a match isn't found. This is because elements having the same key are linked via the right descendent pointer slot and are, hence, chained together. However, if the link type is Linear then an exhaustive search of the remainder of the Pset is necessary.

If the retrieval mode is (3), that is, that all elements in the set be fetched then one of 3 routines is called. If the link type is Linear then LINEAR_L is called which retrieves all the IDs in the set by simply following the pointer chain pointed to by the PSET_CAT entry for the Pset. If the link type is B_tree then LINEAR_B is called which per-

forms an inorder traversal of the B-tree pointed to by the PSET_CAT entry. Finally, if the link type is Hashed then LINEAR_H is called. This routine goes through the scatter table pointed to by the PSET_CAT entry for the Pset and returns all the IDs which are found either in the scatter table or in the associated overflow chains. In any event, the IDs found are placed on a stack and SEARCH returns.

3.2.3.4 FETCH

This module is responsible for fetching the contents of one or more elements within a Pset, given the name of the Pset, the retrieval mode, and a key if needed. The retrieval modes are identical to those of SEARCH. The FETCH module returns a temporary database called INFO_ND which contains the data portions of the elements fetched (See description of INFO_ND in database section). Conceptually FETCH is very simple, because it relies on the SEARCH module to retrieve the IDs of the desired elements. The first thing FETCH does is call SEARCH, passing it the name of the Pset, retrieval mode, and key value. SEARCH returns a stack of pointers (i.e. in the temporary database IDS¹¹) which point to the BEUs containing the desired elements. FETCH then takes the top of the stack, overlays a BEU template on the memory

location specified by the ID on the top of the stack, extracts the data portion of the BEU, and places it on the top of the INFO_ND stack. It then pops the IDS1 stack and continues until the stack is empty.

3.2.3.5 CREATE_E

This module is responsible for creating a BEU given a bit string representation of a data value and the length of that string. It returns a pointer to the newly created BEU which serves as the BEU's ID. Basically, the module allocates a BEU, initializes the P_array to contain null values, inserts the bit string into the data portion of the BEU, and sets the length field accordingly. It then returns the pointer which points to this allocation of the BEU.

3.2.3.6 CREATE_I

This module is responsible for creating an allocation of the INDEX database to be used as a scatter table for a given Pset. It returns the pointer to this allocation of INDEX. It requires the name of the Pset and the length of the key to be hashed. The procedure is straightforward. It allocates a copy of INDEX, initializes the pointer array to null, inserts the name of the Pset and the length accordingly, and returns the pointer to this allocation of INDEX.

3.2.3.7 HASH

This function is used to hash key values, specified as bit strings, into the scatter tables used to link the Psets. It requires the key value and the length, in bytes, of the key. The hashing function is as follows: It takes the key, 2 bytes at a time, treats it as an unsigned integer and adds it to a running total for the key. Once the key has been converted in this fashion, it is divided by the size of the scatter table + 1, and the remainder represents the hashed value. Since HASH is implemented as a function, it takes on the value of the hashed value.

3.2.3.8 INIT_P

This module is responsible for initializing the PSET_CAT Pset. Hence, it must be called before any other Psets are defined. The module contains a temporary structure which is initialized to contain the desired Pset organization for the PSET_CAT Pset. It calls CREATE_E, passing it the information contained in the structure. The BEU that is created represents the PSET_CAT catalogue entry for the PSET_CAT Pset. CREATE_I is then called to create a scatter table for the Pset, and the p_array of the BEU is updated accordingly. The HASH function is then called, passing it PSET_CAT as the

key, and the corresponding entry in the scatter table is updated to point to the BEU containing the catalog entry. Finally, the ID of the catalogue entry is saved in a static External variable called PCATPTR. The resulting structure for the PSET_CAT Pset was shown in table 4 .

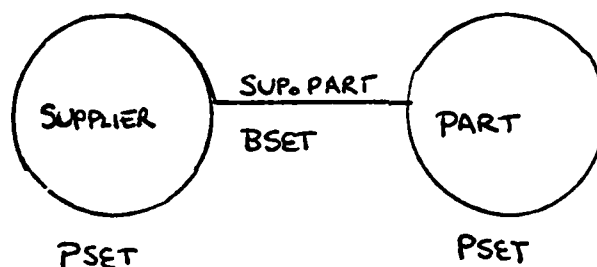
3.2.4 The Binary Layer

The conceptual data model of the Binary layer is the BINARY SET (Bset). A Bset is defined to be a set of relationships or associations, possessing some common meaning, between elements of 2 Primitive sets. For example, if one Primitive set consists of supplier's names, and another Primitive set consists of parts, then one could define a Binary set linking suppliers to the parts that they supply. It is a Binary relationship in the sense that only 2 sets of elements are involved, however, an instance of a Binary set in no way need be 1 to 1. For example, an instance of this Bset might consist of all the parts supplied by supplier A. In this case the relationship might be 1 to n, or even m to n if more than 1 supplier supplies the same part. A Binary association is simply an instance of a Binary set. A Binary set is considered to be uni-directional, i.e. supplier-parts is one Bset, parts -supplier is another Bset.

However, the existence of a Bset implies the existence of it's reciprocal.

Thus, a Binary set is represented by a collection of links between elements in two Primitive sets. It is typically diagrammed as shown below, where the circles or nodes represent the Primitive sets and the arc represents the Binary set linking the two Primitive sets.

Diagram of a Binary Set



The purpose of the Binary layer is to map requests on its conceptual data model (the Bset) to it's physical representation, i.e. as links among elements of 2 Primitive sets. The Bset is implemented via the pointer slots of the BEUs which contain the associated elements in the Psets. This means that the Binary level must be aware of the structure of BEUs, be able to modify the contents of the F_arrays of

the BEUs, and be aware of the organization of the Psets (i.e. it must have access to the PSET_CAT). In addition, the Binary layer must maintain a catalogue of all the Bsets defined in the system. This catalogue would contain for each Bset, the name of the Bset, the names of the Psets involved, specification of the type of Binary association found in the Bset, and implementation information.

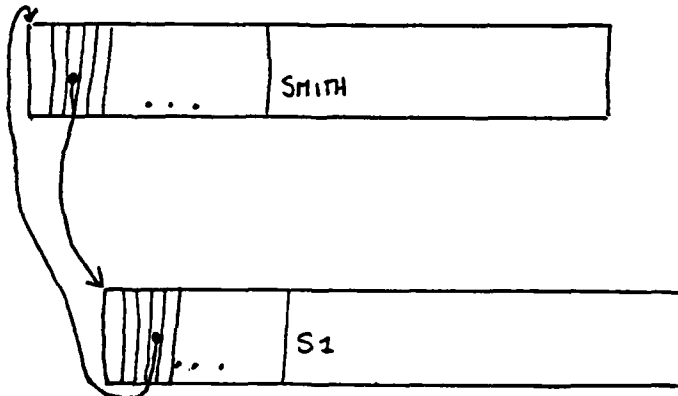
The type of Binary association in the Bset determines the method by which the Bset is implemented. Table 8 illustrates the implementation of a 1 to 1 Binary association. If the type of Binary association is 1 to 1, then the Bset define module uses the information contained in the PSET_CAT entries for the 2 Psets to locate and reserve a free pointer slot in the BEUs of each Pset to be used for Bset linkage. All Binary associations in the Bset will use the same pointer slots to implement the Binary linkage. Note also that the pointer slot need not be the same in the two Psets. A Binary association is created by setting the contents of the appropriate pointer slot in the BEU containing the desired instance of the first Pset, so that it points to the BEU containing the desired instance of the second Pset, and vice-versa.

TABLE 8

Implementation of a 1 to 1 Binary Association

LENGTH	POINTER ARRAY	NAME	DOIN NAME(S)	AC-NO(S)	DOIN NAME(S)	AC-NO(S)	TYPE	SUBSET-ID	FIN-NAME	UNUSED
	...	BSET1	SNAME	2	S#	1	1:1	0	-	

BSET.CAT ELEMENT DESCRIBING BSET1

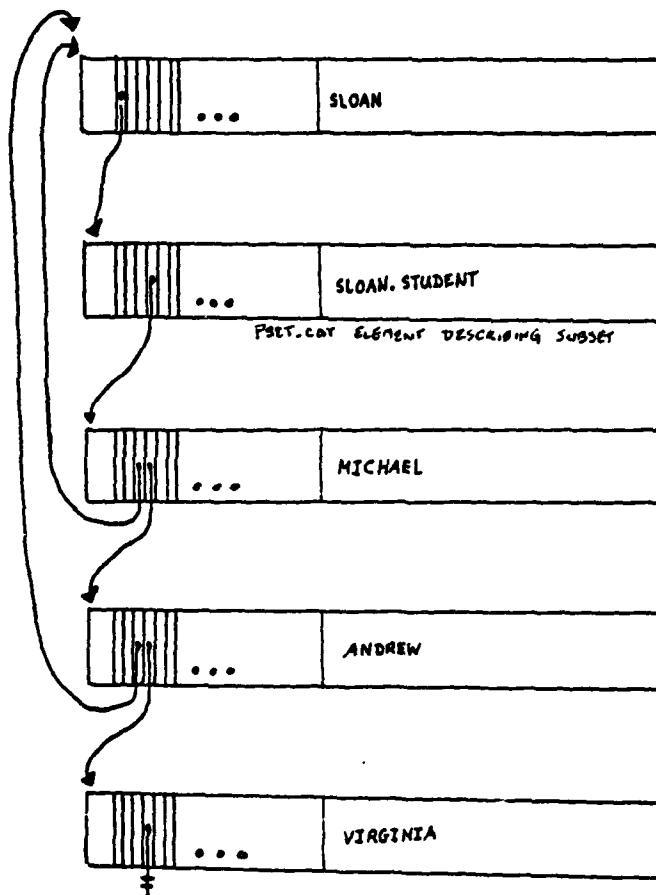
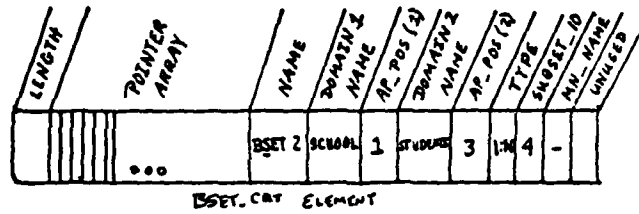


If the type of Binary association is 1 to n, then it becomes more complicated, as shown in table 9 . The Bset define module still locates and reserves a free pointer slot in the BEUs of each Pset. However, the pointer slots are put to different uses. The pointer slot in an instance of domain 1 of the Binary set does not point directly to the associated instances within domain 2, but rather it points to a PSET_CAT entry for a subset of domain 2. The subset is defined as being those elements in domain 2 which are related via the Binary association to that instance of domain 1. Hence, the Binary layer must request that the Pset layer create a PSET_CAT entry for the subset. In order to retrieve an instance of the associated elements in domain2, it is necessary to follow the pointer in the instance of domain 1 to the subset catalogue entry, and then follow the pointer chain specified in the subset catalogue. The pointer slot in the associated instances of domain 2 points to the associated instance of domain 1.

N to 1 Binary associations are implemented in an analogous fashion, except that the pointer slot in an instance of domain 1 points to the associated instance in domain 2, whereas the pointer slot in an instance of domain 2 points to a PSET_CAT entry corresponding to a subset catalogue entry for domain 1.

TABLE 9

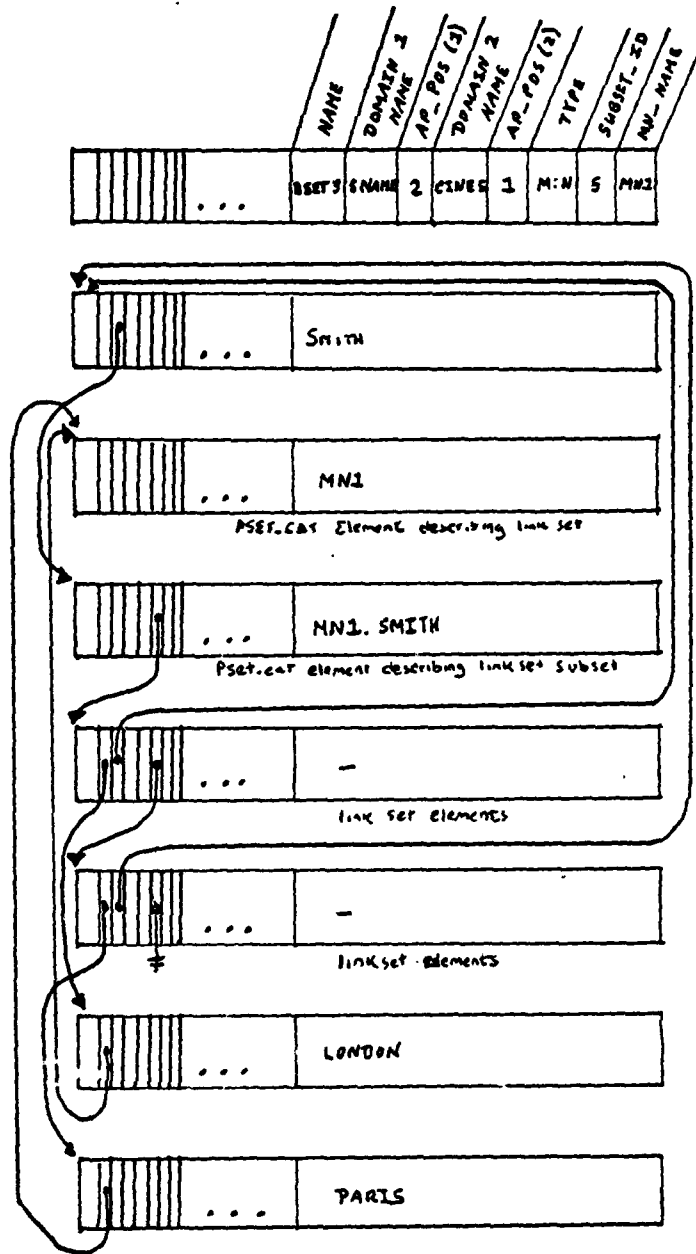
Implementation of a 1 to N Binary Association



M to N Binary associations require the creation of a Primitive set which acts as a link set between instances in domain 1 and domain 2. This is illustrated in table 10. Each element in the link set links one instance of domain 1 to one instance of domain 2. Once again, the Bset define module locates and reserves a pointer slot in the BEUs of each Pset, however, they too are used for different purposes. The Bset definition module also calls the Primitive Layer to define a Pset to be used as a link set. The Binary association is implemented so that the pointer slot of an instance of domain 1 points to a PSET_CAT entry corresponding to the subset of elements in the link set which are used to link that instance of domain 1 to its associated elements in domain 2. By contrast, the specified pointer slot in an instance of domain 2 points to the PSET_CAT entry for the Primary set definition of the link set. Thus to retrieve an instance of a m to n Binary association, it is first necessary to establish the instance of domain 1. Then one must follow the pointer in the specified pointer slot of the BEU containing the instance of domain 1 to establish the subset definition within the link set. Then for each instance of the subset, the pointer contained in the appropriate pointer slot must be followed to establish the associated instances of domain 2.

TABLE 10

Implementation of a M to N Binary Association



Two things should be noted here. The first is that the discussion so far presumes that the instance of domain 2 exists. This need not be the case. A Pset corresponding to domain 2 must be defined prior to the definition of a Bset which involves that Pset. However, if an instance of domain 2 does not exist when a Binary association is being created, the Binary layer will call on the Primitive layer to create the required instance of domain 2. In addition, if the type is M to N the Bset layer will call on the Primitive layer to create elements within the link set. The second point is that actions on a Bset are always specified in terms of an instance of domain 1. That is, given a Bset definition, and an instance of domain 1 actions are performed on the associated instances of domain 2.

3.2.5 Databases of the Bset Layer

The Bset layer implements it's data model via the databases and modules of the Primitive layer together with 1 layer specific database, BSET_CAT, and 3 layer specific modules, DEFINEB, CREATEB, SELECT. In the following sections we will the databases and modules unique to the Binary level.

3.2.5.1 BSET_CAT

Every Binary set defined in the system has an entry in the BSET_CAT. The BSET_CAT entry for a Binary set provides information on the Binary set and how it is implemented. Specifically, an entry in the BSET_CAT is declared as follows:

```
1 BSET_CAT DEFINED(BASE)
  2 SET_NAME    BIT(64),
  2 DOMAIN_INFO(2),
    3 NAME      BIT(64),
    3 AP_POS     BIT(8),
  2 TYPE        BIT(8),
  2 SUB_ID      BIT(8),
  2 MN_NAME     BIT(64),
BASE BIT(320);
```

Where each element is interpreted as follows:

SET_NAME - This is the name of the BSET.

DOMAIN_INFO(2) - For each Pset (i.e. domain) in the Bset it is necessary to know the name of the Pset, which is specified by NAME(i), and the position of the pointer slot reserved in the Pset for implementing the Binary association, which is specified by AP_POS(i). Note, the names spe-

cified are those of the Primary sets involved and not those of the subsets which may be defined during creation of a Binary association.

TYPE - This element specifies the type of Binary association found in the Binary set. The type may be either 1 to 1, 1 to n, n to 1, or m to n. A bit string code is used to represent the type of Binary association contained in the Bset.

SUB_ID - If the type is anything other than 1 to 1, then some form of subset chaining is involved as discussed in an earlier section. This field specifies the pointer slot in the Pset which is to be used for subset chaining.

MN_NAME - If the type is m to n, then this field is used to specify the name of the link set used to implement the Bset.

BASE - This is used to overlay the BSET_CAT entry on the data portion of the BEU in which it is contained. The BSET_CAT is implemented as a Pset, where each element in the Pset corresponds to a BSET_CAT entry. Via a string overlay, the contents of a BEU containing an entry of the BSET_CAT, can be interpreted as such. This means of implementing the

BSET_CAT allows the Bset layer to use the functions provided by the Primitive layer to help it manage its catalogue. For example, the CREATEP module can be used to insert new entries into the BSET_CAT, and the SEARCH and FETCH modules can be used to retrieve elements in the BSET_CAT. This is a clear example of how the concept of a functional hierarchy can reduce redundancy of function.

3.2.6 Modules of the Binary Layer

3.2.6.1 DEFINED

This module is responsible for defining the physical representation of a Bset, given the name of the Bset, the names of the 2 Primary sets involved, and the type of Binary association. To accomplish this task it must: 1) check for the existence of the 2 Psets, 2) locate and reserve a free pointer slot in each Pset, 3) If the link type is anything other than 1 to 1 it must reserve an additional pointer slot in one of the Psets to be used for subset chaining, 4) If the set type is m to n it must define a Pset to act as a link set, and e) it must build the BSET_CAT entry for the Bset and have it inserted into the BSET_CAT Pset.

Tasks 1), 2) and 3) require that the DEFINED module have access to the PSET_CAT entries for the Psets involved. It

calls the SEARCH module to retrieve the IDs of the needed PSET_CAT entries, and overlays a copy of PSET_CAT on the BEUs found, hence allowing it to interpret the contents of the BEUs as entries in the PSET_CAT. It locates and reserves pointer slots in the same manner as the Primary layer, i.e. via calls to the MAPSET module, except that the relevant map in the catalogue entries is the AP_MAP. Note, since the DEFINE module is working on the the actual PSET_CAT entries for the Psets, they are automatically updated to reflect any changes made by the DEFINE module. DEFINE also has an equivalent mode whereby if a Bset being defined is the reciprocal of an existing Bset, then the same pointer slots are used to implement the 2 Bsets.

Task 4) is accomplished via a call to DEFINEP, passing it the name of the link set to be defined and it's characteristics. In a similar manner, to perform task e) CREATEP is called, passing it BSET_CAT as the name of the Pset and a bit string representation of the new BSET_CAT entry as the data value.

3.2.6.2 CREATEB

This module is responsible for implementing an instance of a Bset, given the name of the Bset, a means of establishing the instance of the first domain, and a data value to establish the instance of the second domain. In order to implement the Binary association the CREATEB module must accomplish several tasks: 1) It must first fetch the BSET_CAT entry for the Bset, then 2) it must establish the desired instances within domain 1 and domain 2, finally 3) it must implement the linkage in accordance with the Binary association type specified in BSET_CAT entry for the Bset.

Task 1) is accomplished via a call to the FETCH module, passing it BSET_CAT as the Pset name and the name of the Bset as the key. FETCH returns a bit string which corresponds to the data portion of the BEU which contains the catalogue entry. By setting BASE equal to this bit string, the BSET_CAT structure is, in effect, overlaid on the bit string, and the contents of the bit string can be interpreted accordingly.

Task 2) is also straightforward. When CREATEB is called, the desired instance of domain 1 must already exist. However, it is not necessary for the instance of domain 2 to

exist. The desired instance of domain 1 can be identified in the call either by its ID or by a key value. If a key is specified, then the SEARCH module is invoked to return the ID of the element in domain 1 which matches the key. If no element is found the module returns. Since, the desired instance of domain 2 is specified via a key, it is necessary to call SEARCH to establish if the desired instance of domain 2 exists. If it doesn't, then CREATEP is called to create an instance of the element in domain 2.

Task 3) is fairly complex because of the possible need to define a subset catalogue entry. If the Binary association is 1 to 1 the linkage is straightforward. If the link type is 1 to n then the appropriate pointer slot in the BEU containing the instance of domain 1 is examined to see if it is null. If it is, then DEFINEP is called to create a PSET_CAT definition for the subset within domain 2, and CREATEB places the ID of the catalogue entry in the pointer slot of the BEU in domain 1. In either event, the instance of domain 2 is inserted into the subset specified by the subset catalogue entry. The pointer slot in the instance of domain 2 is then updated to point to the instance of domain 1. If the set type is n to 1, similar logic is employed. If the set type is m to n, then if the pointer slot in the BEU contain-

ing the instance of domain 1 is null, then DEFINEP is called to create a subset definition for a subset within the link set, and the pointer slot in the BEU is updated to point to this catalogue entry. In any event, CREATEP is called to insert a new link element into the subset, and the appropriate pointer slots in the link element are updated to point to the associated instances of domain 1 and domain 2. Finally, the pointer slot in the instance of domain 2 is updated to point to the Primary set catalogue entry for the link set.

3.2.6.3 SELECTF

This module is responsible for retrieving the associated instances of domain 2, given the name of the Bset and a means of identifying an instance of domain 1. It returns the associated instances of domain 2 in the temporary database INFO_ND. The logic is straightforward. It first calls FETCH to retrieve the BSET_CAT entry for the Bset. If the instance of domain 1 is identified by a key value, then SEARCH is called to return the ID of the BEU containing the desired instance of domain 1. The associated elements in domain 2 are also fetched via the FETCH module. If the set type is 1 to 1 or N to 1, then FETCH is called, passing it the con-

tents of the specified pointer slot in the BEU containing the instance of domain 1 which represents the ID of the associated element in domain 2. If the type is 1 to N, then FETCH is called, passing it the name of the PSET_CAT catalogue entry pointed to by the pointer slot in domain 1 as the name of the Pset to fetch, and specifying a retrieval mode of all. This has the effect of fetching all of the elements contained in the subset. If the type is m to n, a similar approach is used to retrieve the subset of link elements which link the instance of domain 1 to instances of domain 2. FETCH is then called for each element in the subset of the link set, passing it the contents of the pointer slot used in the link set to point to instances of domain 2.

3.2.7 Concluding Remarks on Internal Level

This concludes our discussion of the Internal level. The conceptual data model it presents to the Nset or Conceptual layer is that of Primitive sets and Binary sets. On the one hand this is a very simple data model, yet by building up aggregates of Primitive sets and Binary sets it is possible to support complex data structures. As we will see later on, the Conceptual level's data model is nothing more than the aggregation of Primitive sets and Binary sets. The power of

this approach is that while the Conceptual level makes great use of Binary and Primitive sets, it is not necessary for it to be concerned with the actual implementation of them. It simply issues calls to the Internal level to either define Primitive and Binary sets, or to insert elements into them, or to retrieve elements from within a Primitive set or that are associated through a Binary set. It is the Internal level's responsibility to translate the requests which are in term's of its conceptual data model into operations on the data model as actually implemented by the Internal level.

The hierarchical relationship of the Binary layer to the Primitive layer should have been apparent from this discussion. We have seen that the BSET_CAT is implemented as a Pset. As a result, it is possible to make use of the functions provided by the Primitive layer to perform many of the requisite catalogue management tasks. Rather than write a specialized BSET_CAT search routine, it is possible to use the Primitive layer's SEARCH routine. In a similar fashion, there is no need for a specialized BSET_CAT insert routine, because the Primitive layer's CREATEP routine can be used. Even the Binary layer routines such as CREATEB rely on the Primitive layers routines. For example, if an instance of

domain 2 need be created, the CREATEB routine calls CREATEP to create the new element.

The power of this approach is that it reduces redundancy of function to a minimum, which is one of the prime objectives of the functional hierarchy concept <Madnick79>. This serves a dual purpose. On the one hand, it greatly decreases development time because once a particular functional module has been tested and debugged it can be used by any other module which needs to perform that same function. On the otherhand, it increases the reliability of the system not only because it reduces the complexity of the system, but also because it isolates functionality. That is, if a particular function is not being performed correctly, the problem can be isolated to a particular module, rather than having to change multiple modules all of which perform a similar function.

3.3 THE NSET LEVEL

The Nset or N-ary level sits on top of the Internal level, and beneath the External level. The role of the Nset level is to provide an interface between the External and Internal levels, and in effect shield them from each other. It allows the External level to express actions on the data-

base in terms of the Nset level's conceptual data model. The Nset level then translates these actions into the appropriate calls to the Internal level. In this manner, the External level is made completely independent of the implementation of the Internal level.

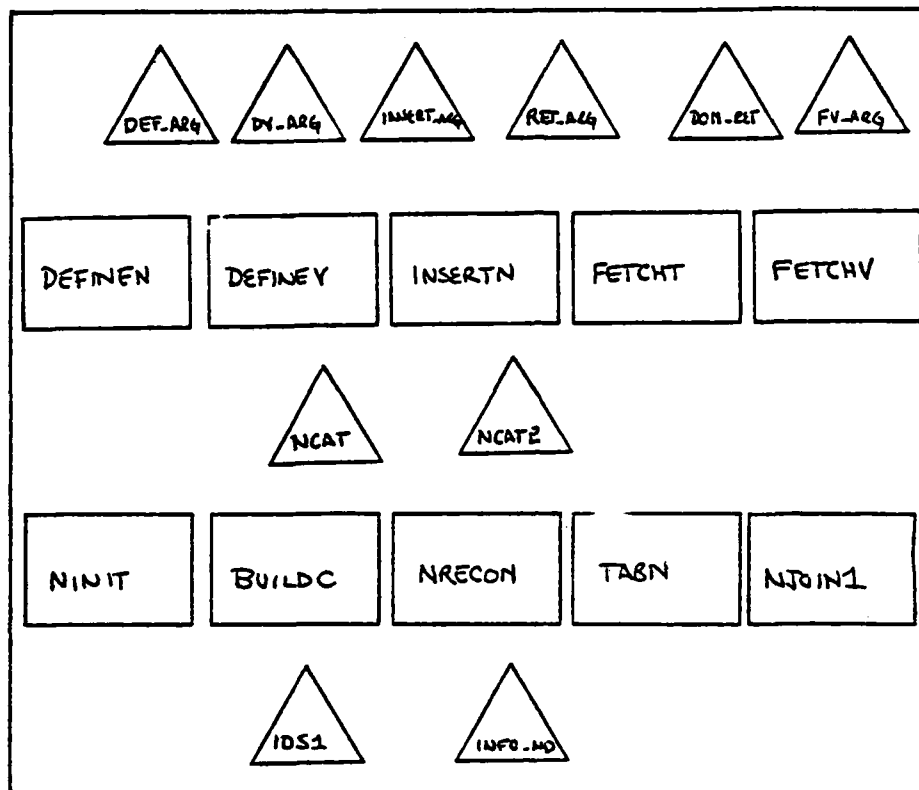
3.3.1 Overview of the Nset Level

This section will describe the conceptual data model of the Nset level, and outline the logical structure of the level. Table 11 illustrates the modules and databases of the Nset level.

The data model chosen for this implementation is a modified Binary network. The basic structure is referred to as an Entity set. Conceptually, an Entity set consists of one or more attributes which collectively describe an object or entity. An instance of an Entity set is represented by the collection of instances of its attributes. Table 12 illustrates a way of diagramming an Entity set using the concept of nodes and arcs. There are 2 types of nodes in an Entity set, value nodes, which correspond to attributes, and entity nodes which act to join the attributes. An entity node derives its value from the values of the attached value nodes. The arcs connecting the nodes represent sets of

TABLE 11

The Modules and Databases of the Nset Level



- MODULE



- DATABASE

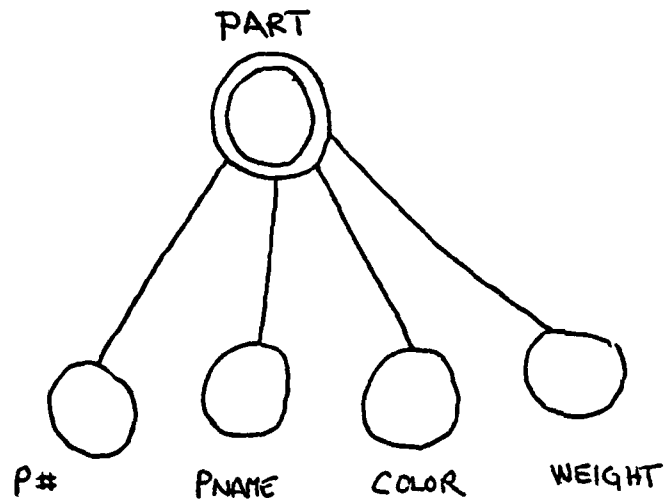
Binary associations (i.e. Binary set) connecting the attributes to the entity node.

As the reader has probably deduced an Entity set can be viewed as a collection of Primitive sets and Binary sets which have some collective meaning. The nodes represent Primitive sets and the arcs, Binary sets. More specifically, an Entity set consists of n Primitive sets which contain actual data values, linked via $2n$ Bsets to a Primitive set which contains no data other than information needed to link the instances of the attributes. $2n$ Bsets are required to implement essentially bi-directional links between the attributes and the entity node. The Binary association between an attribute and the entity node may be 1 to 1, 1 to n , n to 1 or m to n .

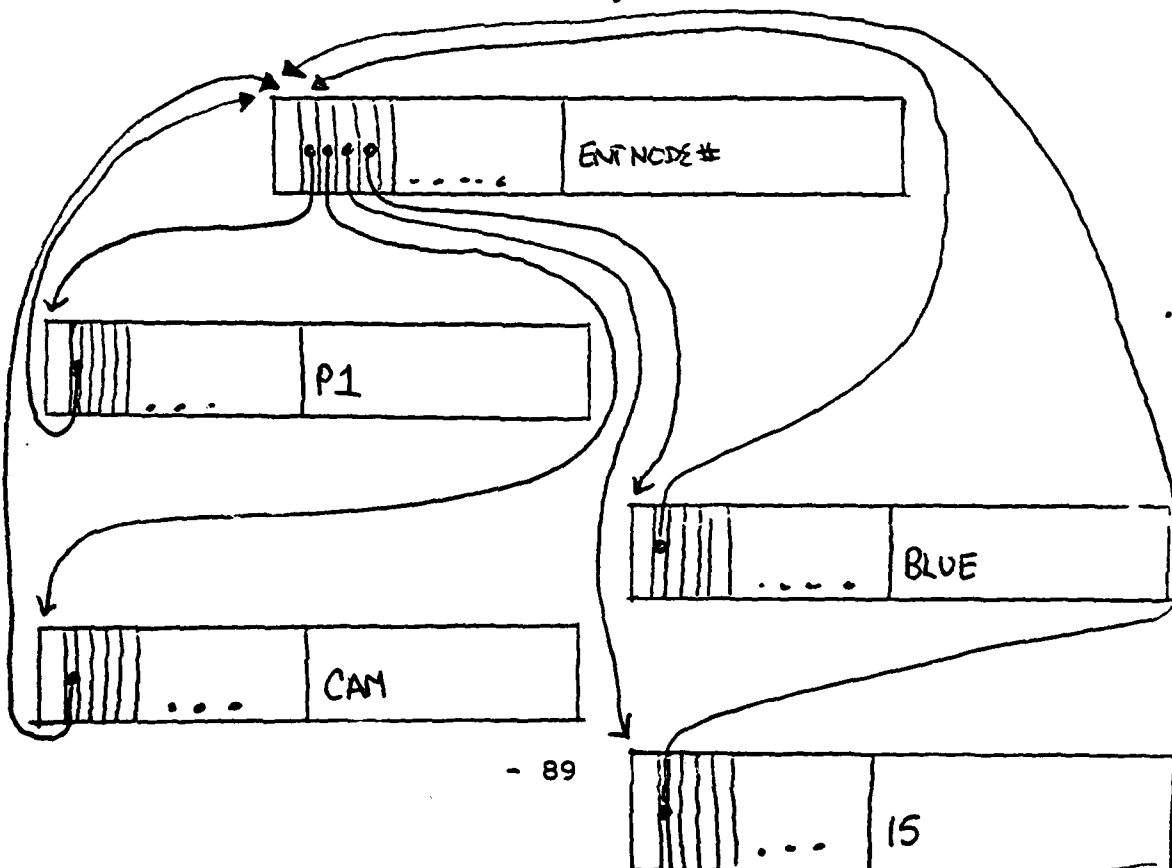
Several points are worth noting here. First, it should be clear that mapping the conceptual data model of the Entity set to the conceptual data model of the Internal level is straightforward. Second, entity sets provide a very simple mapping for the External data model of the relation. Conceptually a relation can be viewed as a special type of Entity set. The domains correspond to attributes, and a tuple corresponds to an instance of the Entity set. In addition, the

TABLE 12

Example of an Entity Set



IMPLEMENTATION OF AN INSTANCE :



Bset connecting an attribute to the entity node, is restricted to 1 to 1 (if the attribute is a unique key) or n to 1 (if a value of the attribute can be in more than one tuple) if the Entity set is used to implement a relation. A third point to note is that this structure is conceptually identical to a fully inverted file. Once an instance of a particular attribute is found, it becomes a simple matter to find all the instances of 1 or more Entity sets which have that instance of the attribute. Finally, there is at most 1 occurrence of a given data item within an attribute, no matter how many Entity sets containing that attribute. Thus the problems of data redundancy largely disappear.

The task of the Nset level is to accept actions against its conceptual data model and map them to the necessary calls to the Internal level which would actually perform the required actions on the physical representation of the database. For example, to define an Nset, given a name, the names of the attributes and information concerning the relationship of the attributes to the entity, the Nset level must perform several tasks. First, it must have the Internal level define a Pset to act as the entity node. Then, for each attribute it must call the Internal level and have it define a Bset linking the Pset acting as the entity node to

the Pset acting as the attribute node. This presumes that the Internal level was called earlier to define the Psets referenced as attributes. Finally, it must create a catalogue entry describing the Nset, and have the Internal level insert the physical representation of the catalogue entry into the the physical representation of the Nset catalogue.

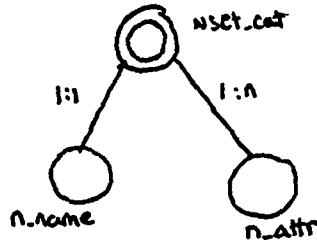
The Nset catalogue is implemented as an Nset called NSET_CAT as shown in t . It consists of 3 Psets (N_NAME, which contains the name of the Nset; N_ATTR, which contains the attribute descriptions; and, NSET_CAT which acts as the entity node) and 2 Bsets connecting the 2 attribute Psets with the entity node Pset. Since the NSET_CAT is nothing more than an aggregation of Psets and Bset, its possible to use the Internal level's routines to help retrieve and manage the catalogue. For example, to retrieve an instance of the NSET_CAT, the FETCH routine is called to establish the desired instance of the name attribute. The SELECTF routine is then called to establish the associated instance of the entity node. It is then called again to retrieve the associated elements of the attribute description Pset.

To insert an instance of an entity into a Nset, given the name of the Nset and values for the attributes, the first

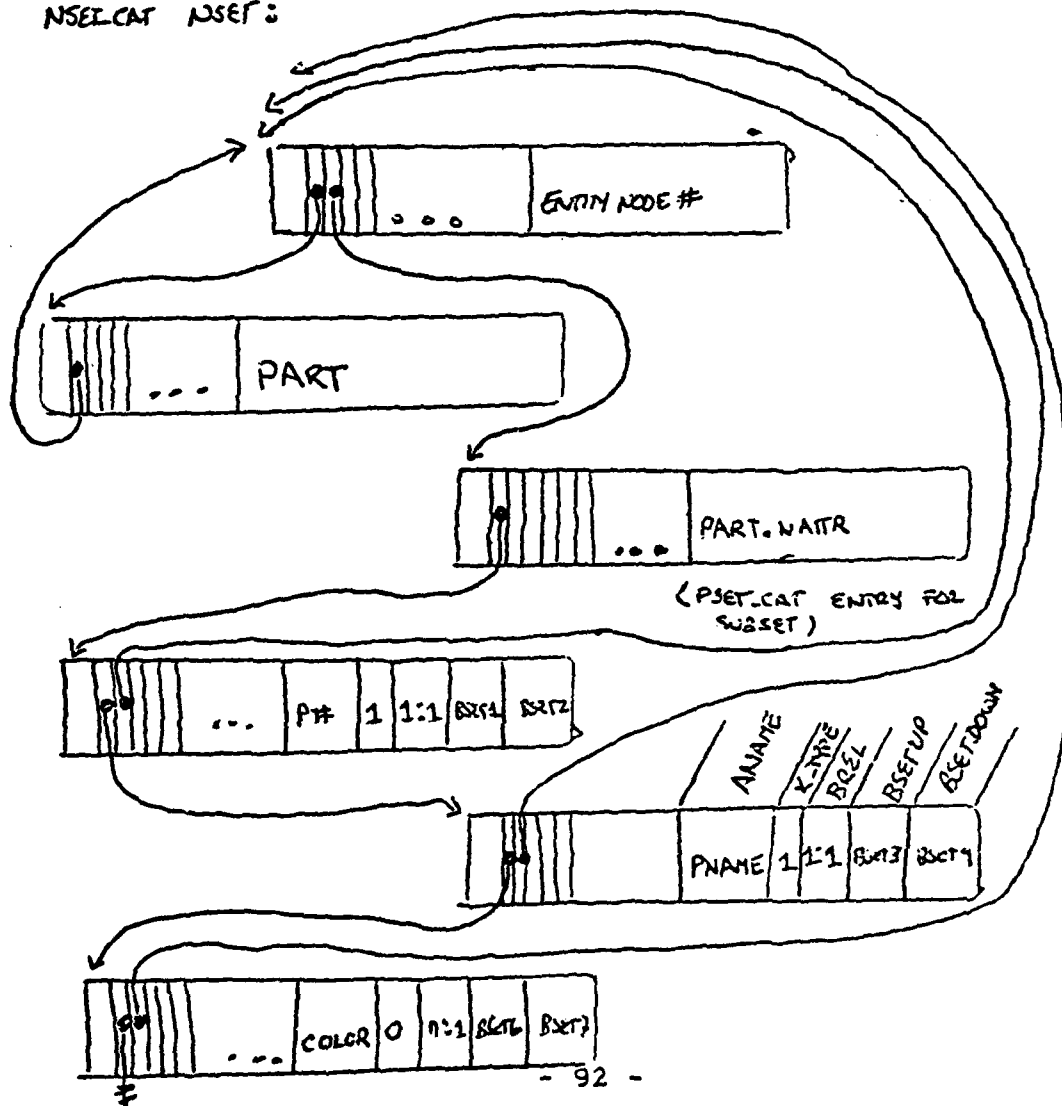
TABLE 13

Implementation of the NSET_CAT

SCHEMATIC VIEW OF
NSET_CAT NSET:



INSTANCE OF
NSET_CAT NSET:



task is to have the Internal level fetch the Nset catalogue entry for the Nset. The level must then check to insure that insertion of the instance of the Nset would not violate the definition of the Nset (i.e. is a unique key value already existed in the Nset). Once again, it would be the Internal level which would be called to perform the searching. Given, that it was a valid request, the Nset level would then call the Internal level to create an instance of the Pset acting as the entity node. Finally, for each attribute it would call the Internal level to create a Binary association between the instance of the entity node Primitive set and an instance of the attribute Pset containing the appropriate value.

Retrieval is somewhat more complex, in that a request may specify restrictions on certain attributes (i.e. $p = 298$), and may encompass more than 1 Nset which share common attributes (i.e. if a join is being requested). The External level specifies a retrieval request via a temporary structure called RET_ARG. RET_ARG contains: the names of the Nsets to be fetched, and any restrictions on the values of attributes including join restrictions. The first task is to build a temporary database which combines the information contained in the request, together with the NSET catalogue

entries for each Nset specified. This is accomplished in part via multiple calls to the Internal level. Restrictions other than join restrictions are then taken into account. For any attribute on which a restriction has been placed, the Internal level is called, first to locate the instance of the attribute which meets the restriction and then to locate the associated instances of the entity node for that Nset. These entity nodes are then compared to other entity nodes for the Nset which satisfied previous restrictions. The intersection of those nodes represents the entity nodes which satisfy all of the non-join restrictions specified for attributes of that Nset. Once this is done for all of the Nset's specified in the request, the join restrictions are followed through. This is done by going through the set of restricted entity nodes for one of the Nsets and for each node establishing the entity nodes in the other restricted sets of entity nodes which both satisfy the join criteria for this Nset as well as with other Nsets specified in the request. The complicating factor here is that a retrieval request can specify joins on more than 2 Nsets, the reason for which will be discussed in reference to the External level. The result of the join operation is a collection of entity nodes which satisfy all of th restrictions specified

AD-A116 593

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/6 9/2
INFOSAM: A SAMPLE DATABASE MANAGEMENT SYSTEM. (U)

DEC 81 B BLUMBERG

N00039-81-C-0663

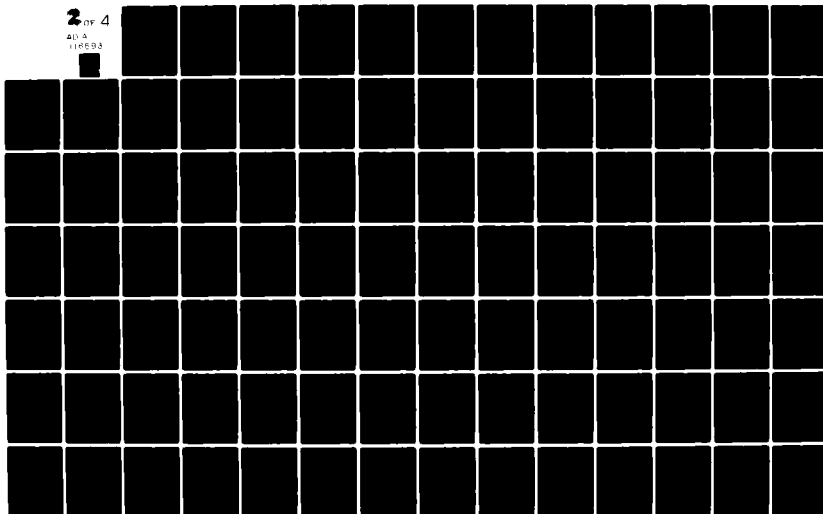
UNCLASSIFIED

CISR-M010-8112-07

NL

2 OF 4

20-A
116593



in the request. The entity nodes are then organized to form a table, where the rows correspond to instances of joined Nsets and the columns correspond to the different Nsets involved. This structure acts as a proxy for the relation satisfying the join and select restrictions specified in the retrieval request. The final task is to fetch the associated instances of the attributes. The retrieved values for the attributes are placed in a temporary database which is returned to the External level.

The reader should note that joins are conceptual rather than physical. When a join is specified by the External level, it is done so within the context of a retrieval request, rather than within the context of a define request. That is, a new Nset is not created, at the Nset level, as a result of a join action. However, the user sees an implicit Nset which is consistent with the join action. This means that every time a user wants to see the contents of the Nset 'created' by the join action, the Nset level must go through the join logic described above. This may be inefficient if it is referenced a great deal.

3.3.2 Databases of the Nset Level

The Nset level has 3 classes of databases which it uses to implement its data model and communicate with the other layers. The first type is the NSET_CAT which the Nset level maintains to implement it's data model. It provides information on all of the Nsets defined in the system. The second type of Databases are those which are used by the Nset and External levels to communicate with each other. INSERT_ARG, RET_ARG, DV_ARG, DEFINE_ARG are all used by the External level to provide information to the Nset level necessary to implement requests made by the External level. DOM_RET is used by the Nset level to return data values, resulting from a retrieval request, back to the External level. The third type are databases used to communicate with the Internal level, i.e. INFO_ND and IDS1, both of which were discussed with regard to the Internal level and so will not be discussed here. In the following section we will outline the structure of the databases mentioned above.

3.3.2.1 NSET_CAT

This database is used to manage the implementation of the Nset level's data model. Every Nset defined in the system has an entry in the NSET_CAT which describes the Nset and

provides information on its implementation. The NSET_CAT is viewed by the Nset level as being an Nset and is implemented as shown in Table 13. Since, an entry of the NSET_CAT is ultimately stored as a collection of BEUs, all of the fields are specified as bit strings. Whenever an entry of the NSET_CAT is required, the BUILD module maps the NSET_CAT entry into the following data structure:

```
1 NCAT,  
  2 NNAME BIT(64),  
  2 NATTR BIT(8),  
  2 ATTR(20),  
    3 ANAME BIT(64),  
    ( 3 K_TYPE,  
      3 BREL ) BIT(8),  
    ( 3 BSETUP,  
      3 BSETDOWN ) BIT(64);
```

Where each element is interpreted as follows:

NNAME - A unique name identifying the Nset.

NATTR - Number of Attributes in this Nset.

ATTR(20) - For each attribute in the Nset:

ANAME - Name of a previously defined Primary set.

K_TYPE - Used to specify if instances of this attribute uniquely define instances of the Nset, i.e. if this is a key or candidate key.

BREL - The type of Binary association between instances of the entity node and instances of the attribute. In this implementation the type is limited to 1 to 1 (if instances of the attribute uniquely define an instance of the Nset), or n to 1.

BSETUP - Name of the Bset used to implement the attribute-entity Binary association.

BSETDOWN - Name of the Bset used to implement the entity-attribute Binary association.

There is an associated database called NCAT2 which is created during the retrieval process. It is implemented as an array where each element corresponds to the NSET_CAT entry for an Nset and retrieval information for that Nset derived from the RET_ARG database. There is an element for each Nset specified in the retrieval request. This database is declared as follows:

- 1 NCAT2(5),
- 2 NCAT_INFO LIKE NCAT,
- 3 RET_INFO,

```
(4 FETCH,  
  4 SAME ) BIT(8),  
  4 VALUE BIT(160);
```

Where the RET_INFO fields are interpreted as follows:

FETCH - Used to specify if values of the attribute are to be fetched. This allows the retrieval request to essentially project the Nset on desired attributes.

SAME - Used to specify if the attribute is the same as an attribute in another Nset specified in the request. The first 4 bits of SAME specify the index of the Nset in NCAT2, the last 4 bits specify the index of the attribute within the Nset. (SAME is described in greater detail in a later section)

VALUE - Used to specify any restrictions placed on the attribute in the retrieval request. As currently implemented, only equality restrictions are supported.

3.3.2.2 Inter-level Communication Databases

The databases described in this section are temporary databases used by the External and Nset levels to communicate with each other. They are temporary in the sense that they are not stored as Psets and Bsets, and, in fact, exist only while certain routines are active.

3.3.2.3 DEF_ARG

Used in conjunction with a DEFINEN request (i.e. a request by the External level to define an Nset). Specifies the conceptual structure of the Nset to be defined. It is declared as follows:

```
1 DEF_ARG,  
  2 NNAME BIT(64),  
  2 NATTR BIT(8),  
  2 ATTR(20),  
    3 ANAME BIT(64),  
    3 K_TYPE BIT(8);
```

Where each element has the following meaning:

NNAME - Unique name identifying Nset to be defined.
Entity node Pset takes on this name.

NATTR - The number of attributes in this Nset.

ATTR(20) - The following information must be specified for each attribute:

ANAME - Name of attribute. Must uniquely identify a previously defined Pset.

K_TYPE - Specifies if values for this attribute uniquely define the instance of the Nset.

3.3.2.4 DV_ARG

Used by the External level in conjunction with a DEFINEV call to specify the structure of a value node(i.e. a Primitive set) to be defined by the Nset level. It is declared as follows:

```
1 DV_ARG
  2 NAME BIT(64),
  2 KEY_LEN BIT(8);
```

Where each element is defined as follows:

NAME - Unique name of the Value node to be defined. The Primitive set defined by the Nset level a result of this request takes on this name.

KEY_LEN - Length of key field within value node/Pset. Specified in terms of bit length.

3.3.2.5 INSERT_ARG

Used by External level in conjunction with an INSERTN call to specify a tuple to be inserted by the Nset level into a previously defined Nset. It is declared as follows:

```
1 INSERT_ARG,
  2 NNAME BIT(64),
  2 NATTR BIT(8),
  2 ATTR(20),
```

3 NAME BIT(64),

3 VALUE BIT(320);

Where the elements are interpreted as follows:

NNAME - Name of previously defined Nset into which this instance is to be inserted.

NATTR - Number of attribute-values specified in this request.

ATTR(20) - The following information must be supplied for each instance of an attribute specified in the request:

NAME - Name of attribute. Note, the order in which attributes are specified is not important.

VALUE - Bit string representation of instance of attribute.

3.3.2.6 RET_ARG

Used by External level to specify arguments of a retrieval request to the Nset level. It allows the External level to specify the equivalent of a join on up to 5 Nsets, place restrictions on the values of attributes, and specify which attributes are to be fetched. It is declared as follows:

1 RET_ARG,

```

2 NUMN BIT(8),
2 NSET(5) BIT(64),
2 ARGS(20),
  3 N_INDEX BIT(8),
  3 NAME BIT(64),
  3 RET_INFO,
    (4 FETCH,
      4 SAME ) BIT(8),
      4 VALUE BIT(160);

```

Where each field is defined as follows (Note, Table 14 illustrates an example of how RET_ARG may be used to express a retrieval request, and the reader may want to study that table in conjunction with the following discussion.)

NUMN - Number of Nsets specified in the request.

NSET(5) - An array used to specify the names of the Nsets involved in the request.

ARGS(20) - Every attribute of an Nset specified in NSET must have an entry in the ARG array.

N_INDEX - Used to specify for which Nset in NSET this is an attribute description.

NAME - the name of the attribute, must correspond to an attribute in the NSET(N_INDEX) Nset.

RET_INFO - Used to specify retrieval information for the attribute. It contains the following elements:

FETCH - Flag to indicate if instances of this attribute are to be fetched, i.e. returned to the External level, if they are contained in instances of the Nset which satisfy all restrict and join criteria.

SAME - Used to specify if this attribute is the same as a previously specified attribute. Used essentially to specify joins. First 4 bits represent the index of the Nset, the last 4 bits specify the attribute within that Nset on which instances of this attribute are to be joined. For example, in Table 14 the Nset SP is joined on S with SUPPLIER. The SAME field for the S attribute of SP is 14 (hex) which indicates that it is to be joined with the first Nset specified in RET_ARG, on the 4th attribute specified for that Nset. Note, this approach limits the number of Nsetsspecified in a request to 16, with a maximum of 16 attributes per Nset.

VALUE - Used to specify a value on which this attribute is to be restricted.

3.3.2.7 DOM_RET

This database is used by the Nset level to return data elements to the External level retrieved as a result of a retrieval request. The database is implemented as a stack of bit strings, where each element corresponds to an instance of an attribute which met the retrieval criteria specified in RET_ARG. Retrieved elements are placed on the stack such that instances of tuples are represented by consecutive data elements. The database is declared as follows:

```
1 DOM_RET EXTERNAL CONTROLLED,  
  2 D_ID BIT(8),  
  2 VALUE BIT(320);
```

Where each field is interpreted as follows:

D_ID - Identifies the Nset and Attribute to which this data element belongs. Uses the same convention as used in RET_ARG.SAME, ie. the first 4 bits specify the index of the Nset in RET_ARG.NSET, and the last 4 bits specify the attribute within that Nset.

Example use of RET_ARG

```

    graph TD
      PNAME((PNAME)) --- CITY((CITY))
      QTY((QTY)) --- CITY
      SNAME((SNAME)) --- CITY
      PNAME --- PNAME_attr((PNAME))
      PNAME --- WEIGHT((WEIGHT))
      PNAME --- COLOR((COLOR))
      QTY --- QTY_attr((QTY))
      QTY --- STATUS1((STATUS))
      SNAME --- SNAME_attr((SNAME))
      SNAME --- STATUS2((STATUS))
  
```

```
SELECT SUPPLIER ON CITY=LONDON, S# = S1 GIVING T1
JOIN T1 AND SP ON S# GIVING T2
PROJECT T2 ON SNAME, P#, QTY GIVING T3
PRINT T3
```

NUMN	NSET	NINDEX	NAME	FETCH	SATE	VALUE
2	SUPPLIER	1	SNAME	'1'8	'00'H	-
	SP	1	STATUS	'0'9	'00'H	-
	-	1	CITY	'0'8	'00'H	'LONDON'
	-	1	S#	'0'B	'00'H	'S1'
	-	2	S#	'0'8	'14'H	-
		2	P#	'1'8	'00'H	-
		2	QTY	'1'8	'00'H	-
		:	:	:	:	:

VALUE - Bit string representation of the value of this instance of the attribute.

3.3.2.8 The FV_ARG database

This database is used by the External level to specify the retrieval of an instance of a Pset. It is also used by the Nset level to return either the data item or a flag indicating that no data value was found matching the key. It is used in conjunction with a call to the FETCHV module. It is declared as follows:

```
1 FV_ARG,  
  2 D_NAME BIT(64),  
  2 KEY_VAL BIT(160),  
  2 FOUND BIT(1),  
  2 DATA BIT(320);
```

Where the elements of the database are defined as follows:

D_NAME - The name of the domain/Pset in which to search.

KEY_VAL - Specifies a key value on which to search.

FOUND - Specifies if an instance of the Pset was found which matched the key.

DATA - The data value found.

3.3.3 Modules of the Nset Level

The Nset level is implemented via 5 major modules and 5 support modules. The 5 major modules correspond to the 5 types of requests that the External level can make of the Nset level. The purpose of these modules is to translate those requests into requests to the Internal level necessary to physically implement them. In addition, they are responsible for translating the responses of the Internal level into appropriate responses to the External level. Finally, the modules are responsible for Nset catalogue maintenance where appropriate. The support modules are not callable by the External level and are used solely by the Nset modules. This section will briefly describe the modules of the Nset level.

3.3.3.1 DEFINEN

This procedure accepts requests to define Nsets, issues the calls necessary to implement the Nset at the Internal level, and creates a NSET_CAT entry describing the Nset and its implementation. It relies on information provided in DEF_ARG to define the Nset. It begins by defining a Pset to act as the entity node for the Nset. This is done via a call to DEFINEP passing it the name of the Nset as the name of

the Pset defined. Then, for each attribute defined in DEF_ARG it defines 2 Bsets linking the attribute to the entity node. The Bsets correspond to the entity-attribute link and the attribute-entity link. The type of Bset implemented depends on the key_type specified for the attribute in DEF_ARG. The Bsets are defined via 2 calls to DEFINEB, one of which specifies the equiv option so that the Bsets share the same pointer slots. Finally, the DEFINEN module creates an NSET_CAT entry for the Nset being defined and has it inserted into the NSET_CAT Nset. This is done via a call to INSERTN, passing it a copy of INSERT_ARG built by DEFINEN so that it contains an entry of NSET_CAT in a form that can be inserted into the NSET_CAT Nset. Note, DEFINEN presumes that the names of the attributes reference previously defined Psets. It is the responsibility of the External level to insure that this is the case.

3.3.3.2 DEFINEV

This procedure accepts requests to define value nodes, issued by the External level, and issues the call necessary to create a Pset definition for the value node. DV_ARG provides it with the name and the key length of the value set to be defined. DEFINEV is very simple. All it does is issue

a call to DEFINEP, passing it the information provided in DV_ARG together with system default characteristics for the structure of Psets. The system default information specifies things like, link type (Hashed), maximum length (40 char), position of the key field, etc... . The rationale for DEFINEV acting as an intermediary between the External and the Internal level is to entirely shield the External level from any knowledge of the Internal level.

3.3.3.3 INSERTN

This module accepts requests to insert instances of an Nset into a previously defined Nset, validates the request, and issues the appropriate commands to the Internal level in order to physically implement the request. It relies on INSERT_ARG to specify the instance of the Nset to be inserted, and on the NSET_CAT entry for the Nset to provide it with the necessary information to implement the request. The logic of the module is as follows. The module first calls BUILDN, passing it the name of the Nset, which returns a copy of the NSET_CAT entry for the Nset. It then validates the request by checking to see that no duplicate values are specified for attributes which are defined as key attributes. Once the request has been validated, a unique tag for

the entity node is generated via a call to NAMEGEN, and CREATEP is called passing it the name of the entity node as the Pset, and the tag value as the bit string. This creates an instance of the entity Pset to be used to chain the instances of the attributes. Then, for each attribute specified, CREATEB is called to create a Binary association between the instance of the entity Pset and the attribute. CREATEB is passed BSETDOWN (from the NSET_CAT entry) as the BSET, the Tag value as the desired instance of the entity Pset, and the attribute value specified in INSERT_ARG as the desired instance of the attribute Pset. It is the responsibility of the CREATEB module to create an instance of the attribute Pset, if necessary.

3.3.3.4 FETCHT

This module accepts requests to retrieve instances of 1 or more Nsets from the database, takes into account any restrictions placed on instances of the attributes, issues the appropriate commands to the Internal level to retrieve the data values, and formats the retrieved instances into the data model of the External level. The retrieval request as specified in RET_ARG may involve several Nsets, restrictions on all attributes, and involve joins between the vari-

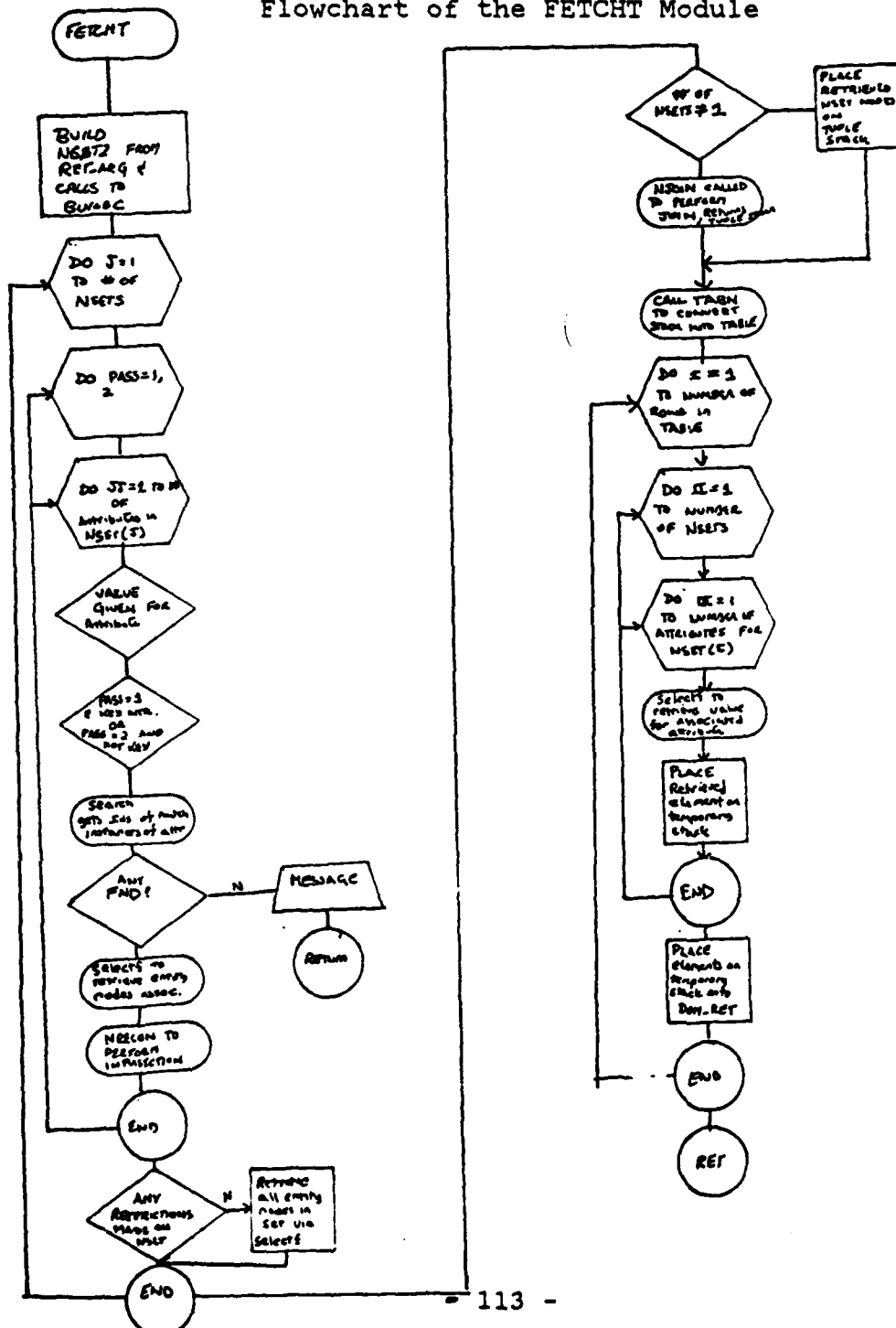
ous Nsets. The resulting complexity means that FETCHT and its associated support routines NJOIN1, NRECON, and TABN represent, perhaps, the most complicated part of the system. As a result, we have provided flowcharts in tables 15 and 16 which outline the logical structure of the 2 most complicated modules, namely FETCHT and NJOIN1. The reader is urged to study those flowcharts as well as the documentation provided in appendix 2. In the discussion that follows we will discuss only the broad outlines of the modules.

FETCHT is the central module in the retrieval system. Its Primary tasks are to 1) create a database which combines the information contained in RET_ARG and the NSET_CAT entries for the Nsets specified in the request, 2) Oversee the restriction phase of the retrieval, 3) call Njoin1 if joins are specified in the request, 4) call TABN to convert the retrieved instances of the Nsets into tabular form, and e) build and fill the database used to return retrieved elements to the External level.

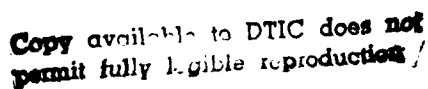
Task 1) is accomplished via multiple calls to the BUILD module. Each Nset specified in the request requires a call to BUILD to retrieve its NSET_CAT entry. The information contained in the NSET_CAT entry is combined with the

TABLE 14

Flowchart of the FETCHT Module



Flowchart of NJOIN1 module



RET_INFO information specified in RET_ARG to form a temporary database called NCAT2. The objective here is to create one database which contains both the retrieval information (i.e. restrictions on the attributes, join information etc...) and information on how the Nset is implemented.

The objective of task 2) is to create a set of entity nodes for each Nset which satisfy any non-join restrictions placed on their attributes. This is done in two passes. On the first pass the Nset is restricted on any attributes for which a value has been specified, and which has been defined as being a key attribute. On the The rationale for doing this restriction first is twofold. First, it is the most limiting restriction placed on the Nsets (i.e. if it is key attribute, any given value of that attribute can be associated with only one instance of the Nset). This then greatly reduces the universe of entity nodes that need be considered in the second pass or during the join process. Second, since the key attribute is likely to be hashed, access is very quick. On the second pass, any non-join restrictions not taken into account on pass1, are taken into account. A temporary database called NLIST is maintained which contains only those entity nodes which satisfy all non-join restrictions placed on them. The NRECON module is responsible for

maintaining that database given candidate entity nodes retrieved by FETCHT. At the end of this process NLIST contains only those instances of the Nset nodes which satisfied all restrictions placed on them.

Task 3) is performed largely by NJOIN1. The task of NJOIN1 is to create an ordered stack which corresponds to instances of the entity nodes which meet both join and all other attribute restrictions. It uses the entity nodes contained in NLIST as its universe of possible candidates. The logic employed is shown in the flowchart for NJOIN1. The complicating factor in NJOIN1 is that more than 2 Nsets may be involved. For example, suppose three Nsets are specified in the request, and Nset2 is joined on Nset1 and Nset3 is joined on Nset2. Hence, Nset3 is implicitly joined on Nset1 as well. To establish an instance of the joined Nsets, given an instance of Nset3, it is first necessary to establish the instances of Nset2 which satisfy the join restrictions placed on Nset3. Given those instances of Nset2 it is then necessary to restrict them to only those which satisfy join restrictions on Nset1. At the conclusion of this process a given instance of Nset3 may be joined with X instances of Nset2 which in turn may be joined with Y instances of Nset1. This process must be repeated for each instance of the last

Nset. Since this is a recursive process, NJOIN1 was implemented as a recursive procedure. It requires that all joins be specified as joins on previously specified Nsets in NCAT2. The rationale for performing the join logic after the other restrict logic was to limit the number of entity nodes to be examined. At the end of this process the stack created by NJOIN1 would contain only those entity nodes which met all join and other restrictions specified in the request.

Task 4) is to convert the stack into a rectangular table format, where each entry in the table corresponds to an instance of an entity node. This is done essentially to convert the format of the data from the Nset data model to that of the External data model. For example in our earlier example, 1 instance of Nset 3 was joined on X instances of Nset 2, and through those instances of Nset 2 on Y instances of Nset 1. The task of TABN is to convert those entries on the stack into X*Y tuples. It does this as follows, it fills the first column in a temporary table called TAB with the elements from the stack until the tuple id is no longer 1. It then fills column 2 with the next element on the stack until column 2 contains as many elements as column 1. It then gets the next element from the stack. If its tuple id indicates that it corresponds to a 3rd Nset then the same

process as given for column2 is repeated. Otherwise, the procedure returns to column 1 and repeats the process again, starting where it left off.

The final task is to convert the table created by TABN into instances of the associated attributes, and place them in a database called DOM_RET which is passed back to the External level. This is done by processing the table created by TABN, a row at a time, and for each entry fetching the associated attributes. Only those attributes for which the FETCH flag in RET_INFO is turned on and which are not the same as previously specified attributes are fetched.

3.3.3.5 FETCHV

This module allows the External level to retrieve a single instance of a value set, i.e. a Primitive set. The External level specifies the name of the value set/Pset, as well as a key value on which to search in the FV_ARG database. FETCHV simply calls the FETCH module, passing it the name of the Pset and the key. When the FETCH module returns, FETCHV updates the FV_ARG database accordingly and returns. The rationale for the FETCHV module is identical to that as for the DEFINEV module.

3.3.3.6 BUILD

This is a support module which is called by the other modules in order to retrieve an entry contained in the NSET_CAT Nset and formats it into the form of the NCAT structure described earlier. It is passed the name of the NSET_CAT entry to be fetched and a copy of NCAT. It returns the copy of NCAT containing the information contained in the NSET_CAT entry for the Nset. The logic is as follows: It first fetches the instance of the Nset name from the Pset NSETNAME via a call to FETCH. This retrieves a bit string which contains the Nset's name and number of attributes. This instance is used in a call to SELECTF to establish the associated instance of the entity node of the NSET_CAT entry. Once the instance of the entity node is fetched, it in turn is used in a call to SELECTF to retrieve the associated instances of the attribute descriptions contained in N_ATTR. The attribute descriptions are placed into the NCAT structure via string overlays.

3.3.3.7 NINIT

This procedure is required to initialize the NSET_CAT and must be called prior to any Nset definitions. It begins by issuing 3 DEFINEP calls to set up the Pset's used to imple-

ment the catalogue, NSETNAME which contains instances of the Nset names, NSETCAT which acts as the entity Pset, and N_ATTR which is used to hold attribute descriptions. It then calls DEFINEB to set up the Bsets linking the 2 value Psets with the entity Pset. Finally it inserts the first NSET_CAT entry into the NSET_CAT, namely the NSET_CAT entry describing the NSET_CAT NSET.

3.4 SUMMARY OF THE NSET LEVEL

This concludes our discussion of the Nset or Conceptual level of INFOSAM. Several points are worth noting in summary. First, the Nset level effectively shields the External level from the Internal level, while at the same time making extensive use of the functions provided at the Internal level to implement the physical representation of the External data model. Second, the functionality of the Nset level is such that the External level, which implements a relational data model essentially becomes an interface between the user and the Conceptual level. As we will see in our discussion of the External level, much of the work of the External level is to create the required communication databases based on requests from the user. Third, we chose to implement a very simple form of the Nset data model. This

implementation highlights 2 points associated with that. The first is that despite it's simplicity it provides a powerful data model for the External level. The second is that despite its simplicity, its implementation is far from trivial, particularly in the area of retrieval. An Nset level as envisioned by Hsu<Hsu>, which is far more general than the one implemented here, would be a challenging task indeed.

3.5 THE EXTERNAL LEVEL

The External level is the highest level in INFOSAM, and as such, sits between the user and the Nset level. This is somewhat different from the structure envisioned by Madnick <Madnick79>. In his original design there were several levels between the External level and the Nset or N-ary level, for example, a data validity level and a virtual information level. Conceptually, INFOSAM could incorporate those levels, however, in this implementation we did not implement them. The External level is designed to provide the user with a simple interactive Relational interface to the system. It allows the user to define his database in terms of the relational data model, and it supports relational operations or queries against the database so defined. The External level accomplishes this by mapping the user's view, i.e. a rela-

tional view, to the conceptual data model of the Nset level, and then by issuing the calls to the Nset level necessary to implement his view at the Nset level. As a result, the user is effectively shielded from both the conceptual and physical implementation of his database.

3.5.1 Logical Overview of External Level

The conceptual data model of the External level and, hence the data model visible to the user, is that of domains, relations, and views. Since, the concept of domains and relations are probably familiar to the reader, they will not be discussed here. The concept of a view, however, may not be so familiar. A view is defined here as the collection of relations that a given user sees or to which he has access. <Date, Hsu > A user may only issue relational operators (i.e. join, project, select, load and print) against relations which are within, or derived from his view. Each view has a unique ID which the user must specify before he can access relations within that view. Note, a relation can appear in more than 1 view. Thus, the concept of a view provides a measure of security control.

In order to implement actions expressed in terms of the relational data model, the External level must map those

actions to equivalent actions on the Nset representation of the relation, and call the appropriate Nset level routines. This mapping process is very straightforward, because as mentioned earlier, a relation can be viewed as a restricted form of an Nset. The domains of the relation correspond to value nodes, the relation to the Entity set, and a tuple to an instance of an Nset. Thus, a relation can be easily implemented as an Nset. When a user wishes to define a relation, the External level translates that request into a call to DEFINEN, passing it the name of the relation as the name of the Nset being defined, and the names of the domains as the associated attributes. An insertion into an existing relation is translated into an INSERTN request, passing it the name of the corresponding Nset, and the values for the attributes.

Relational operations such as joins, projects and selects are also easily mapped onto operations on the Nset model. Conceptually, a SELECT can be viewed as a retrieval operation on an Nset where instances of the Nset are restricted on the value of some attribute. A PROJECT is a retrieval request where all instances of an Nset are to be fetched but only the values for certain attributes are to be displayed. Finally, a JOIN operation is a retrieval request where the

request spans 2 or more Nsets which share 1 or more common attributes. Here, it is also necessary to normalize the returned instances to reflect the rectangular structure of the relational model. For example, at the Nset level 1 instance of Nset2 may be joined with 3 instances of Nset1, at the External level this represents 3 tuples of the relation created via the join. Conceptually then, each relational operation could be implemented by issuing a call to FETCHT, and passing it a copy of RET_ARG which contained the necessary retrieval information.

In fact, the implementation of RET_ARG and FETCHT is such that it allows the External level to implement the concept of virtual commands with regard to relational operations (see<Astrahan76> for a discussion of a similar approach). When a user specifies the view he wishes to use, a temporary data structure is created which is essentially an array where each element in the array is equivalent to a copy of RET_ARG. For each relation in his view, an element in the temporary structure is set up so that it is identical to the copy of RET_ARG required to fetch the Nset representation of the relation. When a user issues a relational operation on a relation in his view, the appropriate entry in the temporary structure is updated to reflect the implications of his

request. If a user defines a temporary relation as the result of relational operations on other relations in his view, a new entry is created in the structure, such that if it were passed to FETCHT, FETCHT would retrieve the appropriate elements given the restrictions specified. The Nset level isn't called on to retrieve data until the user issues a PRINT command, hence, the notion of virtual commands. When a user specifies a relational operation, it is as if the operation was really performed, but in fact only a virtual version of the operation is performed.

The rationale for this approach is severalfold. From a user's viewpoint it gives the appearance of a faster response time since the actual consequences of a relational operation can be put off until the last possible moment. In fact, the real response time may be faster, since the sequence of operations may be such that relatively few data items ultimately need to be fetched. From a system standpoint it is also advantageous since it reduces the number of calls to the Nset level, it eliminates the need to manage the data returned as a result of an operation which is part of a sequence of operations, and the need for modules at the External level to handle relational operations on the data returned as the result of prior relational operations. If

the system were multi-threaded, there would be an additional advantage. If data were retrieved after each operation, the Nset's involved would have to be locked at the time the user accessed his view, since they might be modified between one request and the next. In this approach the Nset's would only have to be locked while the FETCHT module was being called on to retrieve the actual data items.

Since the External level acts as an interface to the User, a certain degree of attention was paid to the quality of the interface. In particular, the user interface was designed to be 'user friendly' yet able to adapt to the increased sophistication of the user as he becomes more familiar with the system. Thus, the routines were set up so that a user can specify all or part of a command and be prompted for the rest. For example, a new user may need to be prompted for each item in a command, while a sophisticated user doesn't need the prompts, and in fact would probably find them a hinderance. If a user makes an error, ranging from a syntax error to a data error (i.e. enters a character where a number is required), the routines are set up so that only the offending element need be re-entered. The routines are also responsible for checking the validity of requests. For example, that relations are only defined

over previously defined domains, or that views contain only previously defined relations, or that the user doesn't issue a request to define a domain, relation or view which is previously defined. As mentioned earlier, it also performs simple data validity checks. When a domain is defined, its characteristics (type, length, minimum and maximum value if a number) must also be specified. During data entry, data being entered is checked against the characteristics specified for the domain. If it doesn't agree, an error message is printed out and the data value must be re-entered. A sample terminal session is provided in appendix I.

3.5.2 Databases of the External Level

The External level makes use of 3 types of databases. The first type of database represents the catalogues used by the External level to store information regarding domains, relations and views defined by the user. These catalogues are called DTABLE, RTABLE and VTABLE respectively. Just as the Internal and Nset levels' catalogues were viewed as being implemented within the context of the level's data model, so too the External level views its catalogues as relations. DTABLE is viewed as a relation containing 2 domains; DNAME, which contains the names of all domains defined by the user

and, DATTR which contains the attributes of the domains. RTABLE is also viewed as a relation containing 2 domains; RNAME, which contains the names of all the relations defined by the user, and DNAME which is the same as DNAME in DTABLE. VTABLE also contains 2 domains, one of which is also shared with RTABLE; RNAME as defined before, and V_ID which contains the IDs of all views identified in the system. These relations are implemented as Nsets via a calls to the Nset level. This is illustrated in Table 17 The External level maintains and manipulates these relations using the same operators it uses to implement user defined relations. For example, when a user issues a GETVIEW command (which retrieves the names of all relations associated with a given view) the External level SELECTs VTABLE on V_ID = ID as specified by user, JOINS the resulting relation with RTABLE on the RNAME domain, and projects the resulting relation on RNAME, and V_ID. Note that this is all done via 1 call to the Nset level.

3.5.2.1 VTABLE

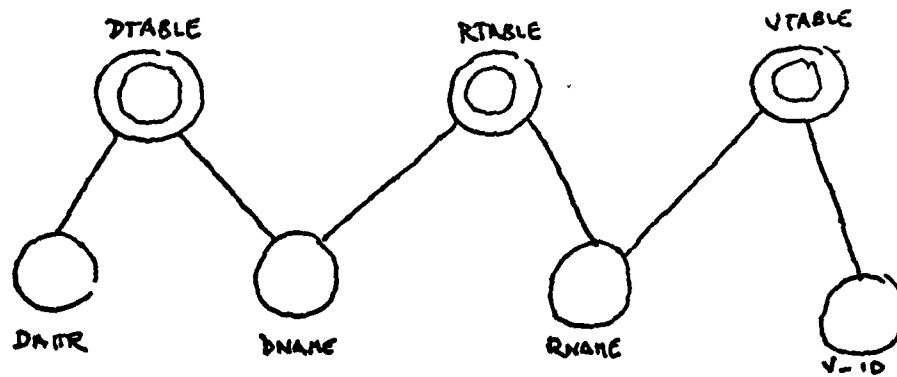
A tuple within DTABLE is declared as follows:

- 1 DTABLE,
- 2 DNAME bit(64),
- 2 DATTR,

TABLE 17

Mapping of External Level Catalogues to Nset Level

DTABLE		RTABLE		VTABLE	
DNAME	DATR	RNAME	.DNAME	RNAME	V-ID
S#	C 2 - -	PART	P#	SUPPLIER	V1
SNAME	C 6 - -	PART	PNAME	PART	V1
STATUS	N 2 0 50	PART	COLOR	SP	V2
CITY	C 6 - -	PART	WEIGHT	SALARIES	V2
P#	C 2	PART	CITY	SPECIFICATIONS	V2
PNAME	C 8	SUPPLIER	S#	.	.
COLOR	C 8	SUPPLIER	SNAME	.	.
WEIGHT	N 2 10 20	SUPPLIER	STATUS		
QTY	N 3 0 500	SUPPLIER	CITY		
.	.	.	.		
.	.	.	.		
.	.	.	.		



3 TYPE BIT(8),
3 TLEN BIT(16),
3 TMIN BIT(16),
3 TMAX BIT(16);

Where the fields are defined as follows:

DNAME - The name of the Domain

DATTR - Contains a bit string which contains the following attributes for the domain:

TYPE - Specifies if the data type is numeric or character.

TLEN - Specifies the maximum length for a data element in this domain.

TMIN - If the type is numeric this field contains the minimum permissible value for an element of this domain.

TMAX - If the type is numeric, then this element contains the maximum permissible value. DTABLE is implemented as an Nset consisting of 2 attributes, DNAME and DATTR.

3.5.2.2 RTABLE

A tuple within RTABLE is declared as follows:

```
1 RTABLE,  
  2 DNAME BIT(64),  
  2 RNAME BIT(64);
```

The meaning of the fields should be clear. Note, however, that a relation of n domains requires n tuples in RTABLE. RTABLE is implemented as an Nset consisting of 2 attributes, DNAME and RNAME.

3.5.2.3 VTABLE

A tuple in VTABLE is declared as follows:

```
1 VTABLE,  
  2 RNAME BIT(64),  
  2 V_ID BIT(64);
```

The meaning of the fields should also be clear. Note, too, that a view which contains n relations requires n tuples in VTABLE. VTABLE is implemented as an Nset consisting of 2 attributes, RNAME and V_ID.

3.5.2.4 The T1_ARG Database

T1_ARG represents the second type of database used by the External level. It is the temporary database through which

the External level implements its virtual relational operations. It is declared as follows:

```
1 T1_ARG(20) EXTERNAL,  
  2 N1 BIT(8),  
  2 C1(5) BIT(64),  
  2 T2(20),  
  3 N2 BIT(8),  
  3 C2 BIT(64),  
  3 T3,  
    4 N3 BIT(8),  
    4 N4 BIT(8),  
    4 C3 BIT(160);
```

Which is functionally equivalent to :

```
1 T1_ARG(20) EXTERNAL,  
  2 VIRT_REL LIKE RET_ARG;
```

When the GETVIEW command is issued, the GETVIEW module via the logic described earlier, fetches the RTABLE tuples for the relations specified in the user's view and loads T1_ARG. At that point T1_ARG will contain 1 entry per relation, and the entry can be used in the FETCHT call to retrieve a relation. As a user issues relational operations, the modules

responsible for implementing them update the appropriate entries in Tl_ARG to reflect the operations. If the user defines a temporary relation as the result of a relational operation on 1 or more previously defined relations in Tl_ARG, a new entry is created in Tl_ARG which reflects both the implications of the relational operation, as well as any previous operations on the relations. When the PRINT command is issued, the PRINT module finds the appropriate entry for the relation in Tl_ARG, and calls FETCHT passing it a copy of the entry. Tl_ARG is overwritten whenever a new GETVIEW command is issued, and is lost in any event at the end of the session. Note, unlike relations which are defined using the DEFREL command, there is no Nset definition for temporary relations. table 18 illustrates a how Tl_Arg is used to map a sequence of commands.

3.5.2.5 Communication Databases

The final type of databases used by the External Level are databases used to communicate with the Nset level. These include INSERT_ARG, DEF_ARG, RET_ARG, DV_ARG, FV_ARG, and DOM_RET. Since these databases were described previously in our discussion of the Nset level, they will not be discussed here.

TABLE 18

Example Use of T1_ARG

THIS EXAMPLE USES THE SAME DATABASE &
SEQUENCE OF COMMANDS AS SHOWN IN TABLE 14

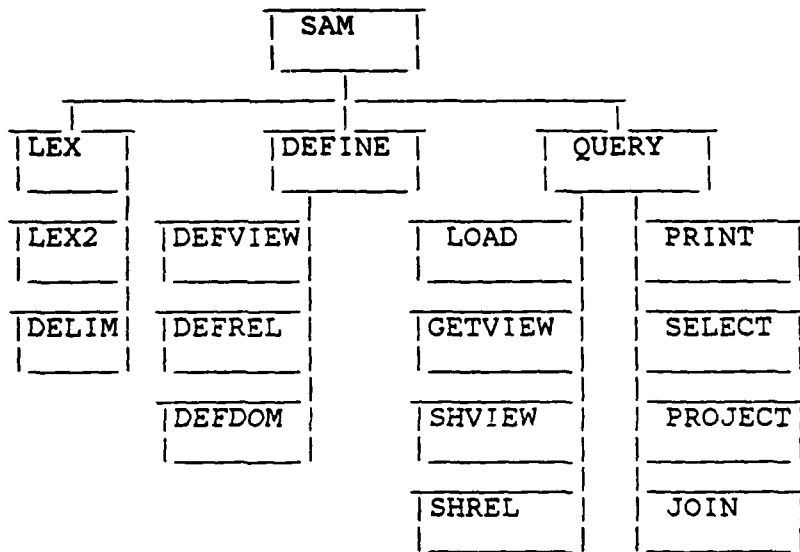
RELNAME	NUM	NSET	N-INDEX	NAME	FETCH	SAME	VALUE
SUPPLIER	1	SUPPLIER	1	S#	'1'B	'00'H	-
FILLED			1	SNAME	'1'B	'00'H	-
			1	STATUS	'1'B	'00'H	-
			1	CITY	'1'B	'00'H	-
			1	CITY	'1'B	'00'H	-
PART	1	PART	1	P#	'1'B	'00'H	-
AS A RESULT OF			1	PNAME	'1'B	'00'H	-
			1	COLOR	'1'B	'00'H	-
			1	WEIGHT	'1'B	'00'H	-
			1	CITY	'1'B	'00'H	-
SP	1	SP	1	S#	'1'B	'00'H	-
GETVIEW COMMAND			1	P#	'1'B	'00'H	-
			1	QTY	'1'B	'00'H	-
T1	1	SUPPLIER	1	S#	'1'B	'00'H	S1
(RESULT OF SELECT COMMAND)			1	SNAME	'1'B	'00'H	-
			1	STATUS	'1'B	'00'H	-
			1	CITY	'1'B	'00'H	-
			1	CITY	'1'B	'00'H	LONDON
T2	2	SUPPLIER	1	S#	'1'B	'00'H	S1
(RESULT OF JOIN COMMAND)			1	SNAME	'1'B	'00'H	-
			1	STATUS	'1'B	'00'H	-
			1	CITY	'1'B	'00'H	LONDON
			2	S#	'1'B	'11'H	-
			2	P#	'1'B	'00'H	-
			2	QTY	'1'B	'00'H	-
			2	QTY	'1'B	'00'H	-
T3	2	SUPPLIER	1	S#	'0'B	'00'H	S1
(RESULT OF PROJECT COMMAND)			1	SNAME	'1'B	'00'H	-
			1	STATUS	'0'B	'00'H	-
			1	CITY	'0'B	'00'H	LONDON
			2	S#	'0'B	'11'H	-
			2	P#	'1'B	'00'H	-
			2	QTY	'1'B	'00'H	-
			2	QTY	'1'B	'00'H	-

3.5.3 The Modules of the External Level

The External level is composed of 16 modules, as illustrated in Table 19. With the exception of LEX and LEX2 each module corresponds to a command issued by the user and is responsible for implementing that command. When the user enters the system he is under the control of the SAM module, and is in the IFS environment. In this environment he can issue 2 commands, DEFINE and QUERY. These commands call the DEFINE and QUERY modules respectively. The DEFINE module sets up the DEFINE environment, in which the user may define domains, relations and views via calls to DEFDOM, DEFREL and DEFVIEW respectively. The QUERY module sets up the QUERY environment which, in fact, has 2 environments, the LOAD environment and the QUERY environment. The LOAD environment allows a user to enter data into a specified relation, whereas the QUERY environment allows a user to examine his database and issue relational operations against it. To move from one environment to another, it is necessary to return to the level which contains that environment, and then issue the appropriate command to invoke the environment. A null line will always return the user to the next higher level.

In this section we will briefly outline the purpose of each module and discuss its structure.

TABLE 19
SCHEMATIC VIEW OF EXTERNAL LEVEL



3.5.3.1 SAM

This is the entry point for the INFOSAM system and represents the outermost environment of the system. It has 2 roles. The first is to call NINIT which is required to initialize the Nset level. The second is to prompt the user to enter either the QUERY or DEFINE environments. If the user enters a null line at this level the procedure ends.

3.5.3.2 DEFINE

This module implements the DEFINE environment in which a user may define his database. The DEFINE module has 2 major tasks. The first task is to define the External level databases, RTABLE, DTABLE, and VTABLE if they have not already been defined. This is done by multiple calls to DEFINEV in order to define the domains of the relations (DNAME,DATTR,RNAME, and ID), and then 3 calls to DEFINEN to define the Nset representations of the relations. The second task is to prompt the user for a valid DEFINE command and dispatch the appropriate routine. The support routine LEX is called to parse the user's response, and the routines are dispatched based on the key word starting the input string. If a user returns a null line, DEFINE returns to the IFS level. When the define modules (DEFDOM,DEFREL or DEFVIEW) return control to the DEFINE module, the DEFINE module prompts the user for a new command.

3.5.3.3 DEFDOM

This module processes the DOMAIN command which is used to define a domain. The form of the DOMAIN command is as follows:

DOMAIN <dname> <d_attr> where

<d_attr> ::= C <Len> | N <Len> <Min value> <max value>

Basically this requires it to perform 4 tasks. 1) If the user has not specified all the information necessary to process the request, the DEFDOM module must prompt the user for the additional information. 2) It must verify that no other domain has been defined with the same name. This is done via a call to FETCHV and requesting that it search the DNAME domain for an occurrence of the name. If it is found, an error message is printed, and no further action is taken. Otherwise, the definition continues. 3) The next task is to format and insert the appropriate entry into DTABLE. This requires it to format the DATTR entry to reflect the domain characteristics, the DNAME entry to contain the name of the domain, and then set up the necessary call to INSERTN. 4) The final task is to call DEFINEV, passing it the name of the domain, so that the Nset level can issue the call necessary to define a Pset corresponding to that domain.

3.5.3.4 DEFREL

This module is responsible for processing the RELATION command which allows a user to define a relation. The format of this command is as follows:

```
RELATION <rname> <dname 1>....<dname i>
```

This requires DEFREL to accomplish 3 tasks, in addition to the task of prompting the user for any information required but not entered with the command. The first task is to validate the request. This takes the form of 2 checks. The first check is to verify that the relation name has not already been used. This is done via a call to FETCHV and having it search the RNAME domain. The second check is to insure that all domains specified are indeed defined in the system. This is accomplished via multiple calls to FETCHV, this time having it search within DNAME. If a domain name is not found in DNAME, the user is prompted to enter a new name. Once the request has been validated, the next task is to insert the relation definition into RTABLE. This is accomplished by multiple calls to INSERTN, passing it, via INSERT_ARG the relation name and the domain name. Each domain in the relation will necessitate a call to INSERTN. The final task is to format DEF_ARG to correspond to the relation being defined, and then to call DEFINEN in order to create the Nset definition for the relation.

3.5.3.5 DEFVIEW

This module is responsible processing the VIEW command which allows a user to define a view. The command form is:

```
VIEW <View Id> <rname1> ...<rname i>
```

The tasks of DEFVIEW are very similar to those of DEFREL, except that no Nset definition is required. It must first validate the request. This consists of checking for the existence of a previously defined view with the same ID, once again accomplished via FETCHV except that it is requested to search in the ID domain. In addition, DEFVIEW must check for the existence of the relations specified in the request. This is analogous to the checking for domains in DEFREL, except that the RNAME domain is checked. If the request is validated, then the view definition is inserted into VTABLE via multiple calls to INSERTN, where each call inserts a tuple with a different relation name.

3.5.3.6 QUERY

This module is responsible for implementing the QUERY environment. It is called by issuing the QUERY command when at the IFS level. Once in the query environment, the user remains in that environment until he enters a null line while interacting with the QUERY module, which returns him to the IFS environment. The task of the QUERY module is to prompt the user for a valid QUERY command, call LEX to parse the response, and then if valid, dispatch the appropriate query routine. The dispatching is done by comparing the key

word in the user's response to an operator table, and calling the routine that matches. The logic is such that the user may also enter a 2 letter synonym for the command.

3.5.3.7 GETVIEW

This module processes the GETVIEW or GV command which allows a user to access relations within a previously defined view. The command form is:

GV <View Id>

It has 3 basic tasks. The first task, as always, is request validation. In this case it must check for the existence of the view id specified in the request. This is done via a call to FETCHV to search the ID domain. If the ID is in fact found, the second task is to fetch the relation definitions (i.e. the RTABLE tuples) of relations within that view. This is done via a call to FETCHT, passing it a copy of RET_ARG which in effect specifies that VTABLE is to be selected on the view ID, joined with RTABLE on RNAME, and the result projected over Dname and Rname. The third task is to build T1_ARG based on the elements returned by FETCHT. As the reader will remember, each entry in T1_ARG corresponds to a copy of RET_ARG which if passed to FETCHT would retrieve the contents of the relation. Hence, GETVIEW must fill an entry

of T1_ARG for each relation name returned as a result of the retrieval request. In addition, GETVIEW creates a table which contains the names of the relations contained in the view.

3.5.3.8 SHVIEW

This module is responsible for processing the SHVIEW or SV command. This command allows a user to display the names of the relations contained in his current view. This module simply takes the table created by GETVIEW and prints it out. The command form is:

SV

3.5.3.9 SHREL

This module processes the the SHREL or SR command which allows a user to see the domains and domain attributes of a relation defined in the current view. The syntax is:

SR <rname>

The module works in the following manner. It first verifies the request by checking the relation name specified against the names of the relations in the current view. This is done via the table created by GETVIEW. If the relation name is found then the corresponding entry in T1_ARG is located. For

each domain in the relation, FETCHT is called, passing it a copy of RET_ARG which is set up to select DTABLE on DNAME = the domain name, and project the result on DATTR. This returns the attributes of the domain to SHREL which formats the attributes and prints them out along with the domain name.

3.5.3.10 SELECT

This procedure is responsible for processing the SELECT command. The form of the SELECT command is :

```
SELECT < Rname1> ON <dname1> = <value1>,...  
      <dname i> = <value i> GIVING <rname2>
```

As described earlier this command is translated into a virtual operation on T1_ARG, and no data retrieval is done until a PRINT command is issued. The objective of the SELECT module is to create a new entry in T1_ARG which contains the information contained in the T1_ARG entry for Rname1 updated to reflect the select restriction on dname. Its first task is to verify that Rname1 exists in the current view. This is done by searching the REL table created by GETVIEW when the current view was loaded. If the relation is found, then it's corresponding entry in T1_ARG is found, and copied into the

next available entry in T1_ARG. Then, for each restriction specified in the request, the module locates the corresponding domain description in the T1_ARG entry for the new relation, and checks to see if it has an existing restriction placed on its value. If so, an error message is printed out. If not, the value specified in the request is inserted into the appropriate element in the entry, i.e. into T1_ARG(i).RET_INFO.VALUE(j,k), where i = the index of the rname2 entry in T1_ARG, j= Nset which contains domain, and k= index of domain within the Nset. If the new domain is not found an error message is printed out. The final task is to create a new entry in REL which contains the name of the new relation, rname2, and its index in the T1_ARG database. If rname2=rname1 then REL is not updated.

3.5.3.11 PROJECT

This module is responsible for the virtual processing of the PROJECT command. The form of the command is :

PROJECT <rname1> ON <dname1>, ..., <dname i> GIVING <rname2>

The logic of the project module is virtually identical to that of the SELECT routine except that instead of placing the restrict value into the appropriate element in the T1_ARG entry for the new relation, the module sets the

appropriate elements in T1_ARG to indicate that they are not to be retrieved. That is, the T1_ARG(i).RET_INFO.FETCH(j,k) fields of domains not specified in the command are set to indicate that those domains within that relation are not to be fetched.

3.5.3.12 JOIN

This module is responsible for the virtual implementation of the JOIN command. The format of this command is as follows:

```
JOIN <rname1> AND <rname2> ON <dname1>,...<dname i> GIVING <rname3>
```

The logic of this module is also very similar to that of PROJECT and SELECT except that it is somewhat more complex. In particular, it must check that the domain(s) on which the relations are to be joined are logically in the relations. For example, if dname1 is physically in the Nset from which rname1 is derived, but as the result of a previous PROJECT command, is not logically in rname1, the join on dname1 is meaningless. Hence, the JOIN module checks for that condition and prints an error message if any of the domains specified in the command are not physically or logically in rname1 or rname2. The second difference is that the fields which are updated in the T1_ARG entry for rname3 are the

T1_ARG(i).RET_INFO.SAME(j,k) fields, where j and k specify the Nset and attribute pairs which correspond to the domains in rname2 on which the join is to be made. These fields must be updated to contain the index to the corresponding domains in rname1. In addition, any RET_INFO.SAME fields in the T1_ENTRY for rname3 must be updated to reflect the new position of the Nset definitions. For example, rname2 may have been created as a result of a join on 2 other relations. To implement that join, the SAME fields for the common domains in the second relation would have been updated to specify the matching domains in the first relation (i.e. the SAME fields would appear as '0001xxxx'b where xxxx corresponds to the domain's index in the first Nset). However, when rname1 and rname2 are joined, the T1_ARG entry for the result will contain arguments for the Nset(s) which make up rname1 and for the 2 Nsets which make up rname2. Hence, the SAME fields for the domains which originally implemented the join in the definition of rname2 must be changed to reflect that the 1st Nset of the rname2 component of rname3 is now the 2 Nset specified in rname3, i.e. the SAME fields must be changed to '0010xxxx'b.

3.5.3.13 PRINT

This module is responsible for processing the PRINT command which allows the user to display the contents of any relation defined in his view, permanent or temporary. The format of the Print command is as follows:

PRINT <rname>

The PRINT module has 3 basic tasks. The first task is to retrieve formatting information for each domain specified in the T1_ARG entry for the relation for which data is to be fetched. This is done by going through the domains specified in the T1_ARG entry for the relation, checking the FETCH field, and assuming it indicates retrieval, calling FETCHT to select the DTABLE on the domain name and to return the DNAME - DATTR tuple. The PRINT module then extracts the maximum length for the domain and places it in an array which is used to format the output line. The 2nd task is to retrieve the data elements contained in the relation as logically defined in the T1_ARG entry for the relation. This is accomplished by calling the FETCHT module and passing it the T1_ARG entry for the relation. FETCHT interprets T1_ARG and returns a stack of fixed length bit strings which correspond to the data elements found which met the restrictions specified in the T1_ARG entry. The final task is to print out

the retrieved data elements. This is done a tuple at a time. For example, if the relation contains n domains, the PRINT module takes the first n elements on the stack, uses the formatting information retrieved earlier to extract the relevant portion of each element, concatenates the resulting strings and prints them out. This continues until the stack is empty.

3.5.3.14 LOAD

This module processes the LOAD command which allows a user to enter tuples into a previously defined permanent relation. The syntax is as follows:

LOAD <Rname>

The LOAD command puts the user into a sub-environment of the QUERY environment, namely the LOAD environment. In this environment a user may continuously enter tuples into a relation, without having to enter LOAD each time. The LOAD module has 3 basic tasks to accomplish. The first task is to validate the user's request. This task is done in conjunction with the second task, namely the retrieval of the domain attributes for the relation specified. This is accomplished by calling FETCHT and passing it a copy of RET_ARG which specifies that VTABLE is to be selected on the View

ID, joined with RTABLE on RNAME, the result selected on RNAME equal to the relation name specified in the request, the result joined with DTABLE on DNAME, and the final relation projected over DNAME and DATTR. If no tuples are returned then LOAD prints an error message. Otherwise, the tuples returned are put into a temporary copy of DTABLE, and a copy of INSERT_ARG is initialized to contain the relation name as the Nset name and the domain names as the attribute names. Next, a header is printed out to specify the order in which values for the domains must be entered. The final task is to prompt the user to enter the tuples to be inserted. For each tuple entered, LOAD checks the values for each domain against the domain specifications in DTABLE. In particular it checks the data type, length, and if numeric whether the number is within the boundary values specified. If a value entered does not meet the criteria, an error message is printed and the user is given a chance to re-enter that data value. Once the tuple has been validated, the domain values are inserted into INSERT_ARG and INSERTN is called to insert the tuple specified in INSERT_ARG into the Nset representation of the relation.

3.5.3.15 LEXICAL ANALYSIS ROUTINES

There are 3 routines which are responsible the lexical analysis of command lines entered by the user. These are LEX, LEX2 and DELIM. LEX is the Primary lexical analysis routine. It performs 2 functions. The first function is to parse the command line into a token array, where each token is a character string of length 8. It recognizes commas, blanks and the equals sign as valid break characters, although the equals sign is also an operator. In addition, it allows a user to embed blanks in an argument if the argument is enclosed in quotes. This first task is accomplished by separating the input string into non-blank character strings separated by blanks. The non-blank character strings are put into the token array. The second task, which is done while the token array is being built, is to keep track of the position of key words in the command line. For example, AND, ON, GIVING, and =. The rationale for this is to simplify the interpretation task performed by the DELIM module.

LEX2 is a specialized routine which is only used by the LOAD module to parse the data values entered by the user. In this case a comma is required as a break character. LEX2 is very similar to LEX except that along with creating the tokens, it specifies the length of the character string con-

tained in the token. However, it is not concerned with the position of key words, since there are none.

DELIM is responsible for performing syntax checking of command lines issued within the QUERY environment. It is called by the QUERY module after LEX has lexically analyzed the input string, and after QUERY has identified the type of command. It is passed the type of command and the positions of the key words. The DELIM module then checks the position of the key words to see if they are positions that clearly imply an error in the command line. If so, the module makes an educated guess at the error, prints a diagnostic error message, and returns a flag to the QUERY module to indicate that the command line is to be ignored.

3.6 CONCLUDING REMARKS ON THE EXTERNAL LEVEL

This concludes our discussion of the External level. Several points are worth noting. First, it should be clear from this discussion that the bulk of the External level's function is very much that of an interface. As such its major functions are, verifying the validity of user' commands, creating the necessary communication databases, and issuing calls to the Nset level to act on the contents of the communication databases. While only very rudimentary

data validity and security control functions were implemented, they are illustrative of a possible approach. Second, the implementation of the relational operators as virtual operations on essentially a mapping table is an interesting concept from a number of perspectives and probably warrants further thought. Third, the ability of the user to enter all or part of a command and be prompted for the rest, may seem trivial but it does add to the perceived flexibility and friendliness of the system.

Chapter 4

THE LESSONS OF INFOSAM

One of the primary objectives of the INFOSAM project was to develop a software test vehicle for the INFOPLEX functional hierarchy. The purpose of this software test vehicle was to gain further insight into the design of the functional hierarchy. While INFOSAM awaits a detailed performance analysis, the implementation itself, has several interesting implications for the design of the functional hierarchy. In this chapter we will discuss the implications of INFOSAM for the INFOPLEX project. In addition, we will suggest areas where the current implementation might be improved.

4.1 INFOSAM AND INFOPLEX - WHAT HAVE WE LEARNED

As a result of the process of implementing INFOSAM we gained some useful insights into the design of the INFOPLEX DBMS. In this section, we will address some of the key issues raised by this implementation of INFOSAM for the INFOPLEX DBMS concept.

INFOSAM suggests that the basic concept of the INFOPLEX DBMS, as proposed by Hsu, works. While it may be argued that INFOSAM is a pale version of the proposed system it does incorporate many of the key design aspects of the system. It is implemented as a functional hierarchy, it makes use of similar data models, and it performs the full translation from External view to the Primitive layer view. We have seen that the mapping process between levels is both possible and powerful.

From an implementation standpoint there is no doubt in my mind that the concept of a functional hierarchy greatly simplifies the implementation process. By building the system a level at a time, starting with the lowest level and working up, debugging was made much simpler since it could be done for each level individually. In addition, since the levels rely heavily on the functions provided by the levels beneath them, the levels became progressively quicker to implement and debug as much of the upper level's tasks were nothing more than calls to the lower level modules. The strategy of implementing a level's catalogues in terms of the level's data model also simplified the debugging phase. Since, the levels employed the same logic in managing their catalogues as for implementing user requests, once a level managed its

catalogues correctly , chances were that it could handle user requests correctly as well.

However, the implementation also raised some disquieting issues as well. For one thing, it was not until we started to implement the Nset level that it became apparent that a full implementation of the Nset level as proposed by Hsu would be extremely complex, if not nearly impossible. The generality of the proposed structure makes operations, even like insertion, very complex. This is particularly true if the operation involves 2 or more Nsets. This complexity wasn't readily apparent until we were faced with the problem of actually implementing it. Our response was to sidestep the issue and only implement a restricted form of the binary network. Hence, the Nset level as implemented here lacks much of the semantic richness that Hsu's design incorporates. However, Hsu's design would probably be a master's thesis in itself to implement.

A second point is that by the very nature of a functional hierarchy, modules in lower level's tend to be called on a great deal to support upper level functions. For example, the SEARCH module may be called upwards of 20 times just to support a given insert tuple command. If this were a multi-

threaded system we suspect that you would see a good deal of contention for key modules in the INTERNAL and NSET levels. In this implementation every attempt was made to avoid a redundancy of of function between the levels. In a later implementation it may make sense to examine this objective and see whether there may be functions which, in the interest of efficiency, are best duplicated across levels. One area where this might be true is in regard to the handling of system databases. The approach taken in INFOSAM is conceptually clean and appealing, but it may not necessarily be the most efficient.

4.2 POTENTIAL AREAS FOR ENHANCEMENT.

In this section we will briefly outline a few areas in which the current implementation of INFOSAM could be enhanced or improved. In some cases these areas represent changes in the current design, and in other cases these are areas which were not addressed in this initial implementation.

4.2.1 Changes in Design

As currently implemented, the BEU is a fixed length structure, consisting of a fixed length pointer array and a fixed length data area. This is highly inefficient in terms of storage utilization, and represents an area which probably should be changed. The change to a variable length pointer array and data area could be easily implemented via the REFER option in PL/1.

The current system implements chaining among elements of a subset via linear chaining. This may represent an area where the system should be changed, or at least examined. However, there may be tradeoffs here depending on the relative frequency that single elements of the subset are retrieved versus retrieval of all elements.

The join process of the Nset level is also an area for potential improvement. Intuitively, we feel that the join process is basically analogous to searching a tree structure, and there is probably a more efficient way of doing this than the one that is implemented.

The External level modules currently have security and data validation functions embedded in them. It would probably make sense to create separate modules to handle both

functions. Since the validation and security requirements of the various modules are fairly similar, it would make sense to consolidate the functions into separate uni-function modules.

The current implementation of the External level treats the relations defined by the user somewhat differently than the system defined relations. In particular, information concerning user's relations is temporarily stored in T1_ARG and all user issued retrieval requests and relational operations involve T1_ARG. The system relations should probably be treated the same way. Two approaches might be taken. One might be to maintain a system copy of T1_ARG, and all requests or relational operations on system relations would reference this table. Another approach would be as follows. When a user's copy of T1_ARG is loaded, the system would additionally load the upper entries in T1_ARG with the information required to retrieve the tuples in the system relations relevant to the user.

4.3 ADDITIONS TO THE SYSTEM

Perhaps the single most important addition to the system would be to allow it to read and write to disk. Currently, the system does not have this capability, which means that a user must redefine and load his database each time he uses the system. A possible intermediate solution would be to create an initialization file which contained both the commands and data necessary to create the database. By redefining SYSIN in the external level so that it references the initialization file, and by adding an ON ENDFILE block which redefines SYSIN to be the terminal when end of file is encountered, a permanent database could be simulated. Every time the system was executed it would initially read from the file until it reached the end of the file, at which point it would begin reading from the terminal. Hence, it would be as if a permanent copy of the database was stored.

Ultimately, however, the system should have the capability to read and write to disk. This could be done with relatively few changes to the existing code through the use of AREAS and OFFSETs. One approach would be to declare an AREA in which all storage allocations were to be made. In addition, all pointer declarations would be changed to OFFSET declarations referencing this area. Otherwise, the code

would not have to be changed. When the system was executed it would begin by reading in this AREA into storage. Prior to terminating a session it would be written back to disk. In this manner, INFOSAM could be modified without great difficulty to support a permanent database.

Another area of potential interest might be to implement the Nset level as originally envisioned by Hsu. As mentioned earlier, it's our sense that this project might be of the same magnitude as the whole of the current INFOSAM project. Our sense is that the insert and definition modules will have to be made far more sophisticated, while the retrieval logic probably won't have to be changed very much. It may also be that the binary level will have to be enhanced to treat a binary association between 2 entities somewhat differently than the binary association between an entity and an attribute. That is, in an CREATEB request, it may be desirable to be able to specify the instance of one of the domains by specifying an instance of a previously defined binary association which contains the element. Another area of interest here would be to determine an approach toward handling Nsets in which not all of the attribute values were given when it was created. Hsu's design supports this concept, but it isn't clear how it might be implemented.

Another area of potential interest would be to implement a deletion capability. Here there are two type of deletions with which to be concerned. The first, and easiest would be to support the deletion of binary associations. This would not be very difficult. If the association was 1 to 1, all that would have to be done is to update the appropriate pointer slots to null. Otherwise, if the binary association involved a subset, then it would be necessary to unchain the element from the subset, and set the appropriate pointer slot to null. By modifying the SEARCH module slightly, it could be set up to return a pointer to the BEU in the subset chain which points to the element to be deleted. This would allow you to unchain the element from the subset. The deletion of elements from primary sets is a more complicated issue since 1 element may be linked to a variety of other elements. One approach might be to set a flag in the BEU indicating that it has been logically deleted, and modifying the SEARCH routine so that it recognises that the element has been deleted. In addition, a list would be kept of all elements which had been deleted during a session. At the end of the session, a module would then go through to physically implement the deletion and its associated ramifications. The rationale for such an approach is that the deletion process

may be sufficiently involved that it may not be desirable, from the view of response time, to physically delete elements when the command is issued.

The provision of an update capability is another area which warrants work. It is also linked fairly closely with the deletion capability since much of the logic would be the same. Once again, the biggest problem would be the updating of an element which was in more than one binary association.

4.4 CONCLUDING REMARKS

At this point the reader should have a clear picture of INFOSAM, its design, its relationship to the INFOPLEX project, and its strengths and weaknesses. We have seen that it is a relational DBMS, which is designed around the complementary concepts of a functional hierarchy and the ANSI/SPARC proposed design. We discussed its role in the INFOPLEX project as a software test vehicle whereby additional insights could be gained into the design of the INFOPLEX functional hierarchy through the implementation, performance analysis and future modification of the INFOSAM system. We then presented an overview of the design. Here we stressed the hierarchical decomposition of the system into levels, where a level was distinguished by a unique

conceptual view of the data, level specific databases, and modules which accepted calls in terms of the level's data model and translated them into requests to the next lower level, expressed in terms of that level's data model. We then took the reader through each level in INFOSAM, and described its data model, databases and modules. In particular, we showed how the level's mapped their data model onto that of the next lower level. Finally, we discussed the lessons we have learned so far from INFOSAM, both for the INFOPLEX project and for future implementations of INFOSAM.

It is the author's sincere hope that the INFOSAM system will prove as useful to others, in particular the INFOPLEX project, as the process of its implementation proved to me.

Appendix A
SAMPLE TERMINAL SESSION

R;

GO

EXECUTION BEGINS...

-- INFOSAM --

[

IFS:

DEFINE

```
*****
* This next section illustrates the definition of the      *
* sample database. The database consists of 9              *
* domains, 3 relations and 1 view. The relations are      *
*      SUPPLIER                      PART                  *
* SS SNAME STATUS CITY      PP PNAME COLOR WEIGHT CITY *
*
*      SP                      *
* SS PP QTY                  *
*
* and the domains are as shown above. The example below *
* illustrates how these domains, relations and view      *
```

* are defined

*

D:

DOMAIN

DOMAIN NAME:

SS

DATA TYPE:

C

MAXIMUM LENGTH:

2

D:

DOMAIN SNAME C 6

D:

D STATUS

DATA TYPE:

N

MAXIMUM LENGTH:

2

MINIMUM VALUE:

0

MAXIMUM VALUE:

50

D:

D CITY C 6

D:

D QTY N 3 0 500

D:

D PP C 2

D:

D PNAME C 6

D:

D COLOR C 6

D:

D WEIGHT N 2 0 50

D:

D CITY C 6

DOMAIN ALREADY DEFINED.

[

D:

RELATION

RELATION NAME:

SUPPLIER

DOMAIN NAMES:

SS SNAME STATUS CITY

UNIQUE DOMAIN INDEXES:

1 2

D:

RELATION PART PP PNAME COLOR WEIGHT CITY

UNIQUE DOMAIN INDEXES:

1

D:

R SP SS PP QTY

UNIQUE DOMAIN INDEXES:

D:

VIEW

VIEW ID:

VIEW1

RELATION NAMES:

SUPPLIER PART SP

D: <CR>

IFS:

QUERY

-- READY FOR QUERIES --

* The following section illustrates the Query environment*

* commands: GETVIEW, SHVIEW, SHREL *

Q:

GV VIEW1

VIEW LOADED.

Q:

SV

SP

PART

SUPPLIER

Q:

SR SP

DOMAIN	TYPE	LEN	MIN	MAX
=====	=====	=====	=====	=====
QTY	NUM	3	0	00
PP	CHAR	2	--	--
SS	CHAR	2	--	--

Q:

SR SUPPLIER

DOMAIN	TYPE	LEN	MIN	MAX

=====	=====	=====	=====	=====
CITY	CHAR	6	--	--
STATUS	NUM	2	0	50
SNAME	CHAR	6	--	--
SS	CHAR	2	--	--

Q:

SR PART

DOMAIN	TYPE	LEN	MIN	MAX
=====	=====	=====	=====	=====
CITY	CHAR	6	--	--
WEIGHT	NUM	2	0	50
COLOR	CHAR	6	--	--
PNAME	CHAR	6	--	--

PP

CHAR

2

--

--

* This next section illustrates the LOAD environment of *
* the system. After each relation is loaded, the contents*
* of the relations are printed out. *

Q:

LOAD SP

|QTY |PP |SS |

L:

300,P1,S1

L:

200,P2,S1

L:

100,P3,S1

L:

140,P4,S1

L:

501,P5,S1

DATA FOR DOMAIN QTY

ABOVE MAX:

51

L:

490,P6,S1

L:

300,P1,S2

L:

400,P2,S2

L:

200,P2,S3

L:

100,P2,S4

L:

340,P4,S4

L:

400,P5,S4

L: <CR>

Q:

PRINT SP

|QTY |PP |SS |

|340|P4|S4|

|51 |P5|S1|

|100|P3|S1|

|400|P2|S2|

|140|P4|S1|

|490|P6|S1|

|400|P5|S4|

|300|P1|S2|

|100|P2|S4|

|300|P1|S1|

|200|P2|S3|

|200|P2|S1|

Q:

LOAD SUPPLIER

<u>SNAME</u>	<u>CITY</u>	<u>SS</u>	<u>STATUS</u>

L:

SMITH, LONDON, S1, 10

L:

JONES, PARIS, S2, 13

L:

BLACK, PARIS, S3, 14

L:

CLARK, LONDON, S3, 27

REQUEST IGNORED, DUPLICATE KEY FOUND IN RELATION

L:

CLARK, LONDON, S4, 27

L:

ADAMS, ATHENS, S5, 24

L: <CR>

Q:

PRINT SUPPLIER

CITY	STATUS	SNAME	SS	

ATHENS 24	ADAMS	S5

LONDON 27	CLARK	S4

LONDON 10	SMITH	S1

PARIS 14	BLACK	'S3

|PARIS |13|JONES |S2|

Q:

LOAD PART

|PNAME |PP |CITY |COLOR |WEIGHT |

L:

NUT, P1, LONDON, RED, 23

L:

BOLT, P2, PARIS, GREEN, 32

L:

SCREW, P3, ROME, BLUE, 14

L:

SCREW, P4, LONDON, RED, 24

L:

CAM, P5, PARIS, BLUE, 37

L:

COG, P6, LONDON, RED, 18

L: <CR>

Q:

PRINT PART

|CITY |WEIGHT |COLOR |PNAME |PP |

|ROME |14|BLUE |SCREW |P3|

|PARIS |32|GREEN |BOLT |P2|

|PARIS |37|BLUE |CAM |P5|

|LONDON|23|RED |NUT |P1|

|LONDON|24|RED |SCREW |P4|

|LONDON|18|RED |COG |P6|

* This next section illustrates the use of the relational*
* operators, PROJECT, JOIN, and SELECT. In this first *
* sequence we show how the SELECT command can be used *
* to retrieve tuples (rows) which, meet certain *
* restrictions from a relation. We also illustrate the *
* use of the PROJECT command to retrieve only relevant *
* columns. *

Q:

SELECT PART ON COLOR=BLUE GIVING T1

Q:

PRINT T1

CITY	WEIGHT	COLOR	PNAME	PP

PARIS	37	BLUE	CAM	P5

ROME	14	BLUE	SCREW	P3

Q:

SELECT PART ON CITY=PARIS,COLOR=BLUE GIVING T2

Q:

PRINT T2

CITY	WEIGHT	COLOR	PNAME	PP

PARIS	37	BLUE	CAM	P5

Q:

PROJECT PART ON PNAME,COLOR GIVING T3

Q:

PRINT T3

COLOR	PNAME

BLUE	SCREW

GREEN	BOLT

|BLUE |CAM |

|RED |NUT |

|RED |SCREW |

|RED |COG |

* The following sequence of commands illustrates the use *
* of the JOIN command. The objective of the sequence is *
* to retrieve the PARTS information for all parts *
* supplied by CLARK and available in LONDON. *

Q:

SELECT SUPPLIER ON SNAME=CLARK GIVING T11

Q:

JOIN SP AND T11 ON SS GIVING T12

Q:

PRINT T12

|QTY |PP |SS |CITY |STATUS |SNAME |

|400|P5|S4|LONDON|27|CLARK |

|340|P4|S4|LONDON|27|CLARK |

|100|P2|S4|LONDON|27|CLARK |

Q:

JOIN T12 AND PART ON PP,CITY GIVING T13

Q:

PRINT T13

QTY	PP	SS	CITY	STATUS	SNAME	WEIGHT	COLOR

PNAME	

340 P4 S4 LONDON 27 CLARK	24 RED	SCREW

Q:

IFS:

-- END OF SESSION --

R;

Appendix B

LISTINGS AND DOCUMENTATION FOR THE INFOSAM SYSTEM

```

1 0 SAM: PROCEDURE;
/*
MODULE DESCRIPTION
*/
PURPOSE: PRINT GREETING MESSAGE, ENTER DEFINE OR QUERY MODE
ACCORDING TO USER INPUT, AND PRINT LOGOFF MESSAGE.
METHOD: NOT SIGNIFICANT
INPUT: NONE
OUTPUT: NONE
CALLS: NINIT, DEFINE, QUERY
/*
/* COMMAND LISTS */
DCL OP(2) CHAR(6) INITIAL('DEFINE','QUERY'),
AOP(2) CHAR(6) INITIAL('D','Q');
/* SUBROUTINES */
DCL DEFINE ENTRY;
DCL QUERY ENTRY;
/* INCLUDE ENINIT: *****
/* NSET LEVEL INITIALIZING MODULE */
DCL NINIT ENTRY;
*****
/* MISCELLANEOUS */
DCL NAME CHAR(6),
PCATPTR POINTER STATIC EXTERNAL INIT(NULL()),

```

```

SAM00010
DOC00020
DOC00030
DOC00040
DOC00050
DOC00060
DOC00070
DOC00080
DOC00090
DOC00100
DOC00110
DOC00120
DOC00130
DOC00140
DOC00150
DOC00160
DOC00170
DOC00180
DOC00190
DOC00200
DOC00210
DOC00220
DOC00230
DOC00240
DOC00250
DOC00260
DOC00270
DOC00280
DOC00290
SAM00010
SAM00020
SAM00030
SAM00040
SAM00050
SAM00060
SAM00070
SAM00080
SAM00090
SAM00100
DEC00080
DEC00090
SAM00100
SAM00110
SAM00120
SAM00130
SAM00140

```

SAM00150
 SAM00160
 SAM00170
 SAM00180
 SAM00190
 SAM00200
 SAM00210
 SAM00220
 SAM00230
 SAM00240
 SAM00250
 SAM00260
 SAM00270
 SAM00280
 SAM00290
 SAM00300
 SAM00310
 SAM00320
 SAM00330
 SAM00340
 SAM00350
 SAM00360
 SAM00370
 SAM00380
 SAM00390
 SAM00400
 SAM00410
 SAM00420
 SAM00430
 SAM00440
 SAM00450

```

      TRUE BIT(1) INIT('1'8);

      /* BEGIN SESSION */
      CALL NINIT;
      PUT SKIP(2) LIST ('--- INFOSAM ---');

      /* GET COMMAND */
      DO WHILE(TRUE);
        PUT SKIP(2) LIST ('IFS:');
        GET EDIT (NAME) (A(6));
        IF NAME = (6) ' ' THEN LEAVE;

        /* SELECT COMMAND */
        SELECT(NAME);

        /* DATA DEFINITION */
        WHEN(OP(1),AOP(1)) CALL DEFINE;

        /* QUERIES */
        WHEN(OP(2),AOP(2)) CALL QUERY;

        /* INVALID OPERATORS */
        OTHERWISE
          PUT SKIP(0) EDIT (NAME,' IS AN INVALID COMMAND') (A,A);
        END;
      END;

      /* END SESSION */
      PUT SKIP(2) LIST ('--- END OF SESSION ---');
      END SAM;
  
```

```

XINCLUDE DEFINE:*****D 00010
/***** FOR00010
* FOR00020
* FOR00030
* FOR00040
* FOR00050
* FOR00060
* FOR00070
* FOR00080
* FOR00090
* FOR00100
* FOR00110
* FOR00120
* FOR00130
* FOR00140
* FOR00150
* FOR00160
* FOR00170
* FOR00180
* FOR00190
* FOR00200
* FOR00210
* FOR00220
* FOR00230
* FOR00240
* FOR00250
* FOR00260
* FOR00270
* FOR00280
* FOR00290
* FOR00300
* FOR00310
* FOR00320
* FOR00330
* FOR00340
* FOR00350
D 00010
D 00020
D 00030
D 00040
D 00050
D 00060
D 00070
D 00080
D 00090
DCL00010

*****
MODULE DESCRIPTION
*****
/
DEFINE: PROCEDURE:
/
PURPOSE:
*****
THIS MODULE IS RESPONSIBLE FOR ESTABLISHING THE DEFINE
ENVIRONMENT. IT IS CALLED BY SAM WHEN THE USER ENTERS
'DEFINE'. IT PROMPTS THE USER TO ENTER A DEFINE COMMAND,
CALLS LEX TO ANALYZE THE INPUT STRING, AND THEN IT CALLS
THE APPROPRIATE DEFINE MODULE.
*****
METHOD:
*****
STRAIGHTFORWARD. NOTES:
A) IS RESPONSIBLE FOR INITIALIZING EXTERNAL LEVEL
CATALOGUES, DTABLE, RTABLE, AND VTABLE. THIS IS
DONE VIA CALLS TO DEFINEV TO DEFINE THE DOMAINS
OF THE CATALOGUES (DNAME,DATTR,RNAME,TD), AND
CALLS TO DEFINEIN TO SET UP THE NSET REPRESENTATIONS
OF THE EXTERNAL LEVEL CATALOGUES.
B) LEX IS RESPONSIBLE FOR PARSING INPUT STRING.
*****
INPUT PARAMETERS:
*****
NONE,BUT PROMPTS USER FOR A COMMAND AND
CALLS LEX TO PARSE COMMAND.
*****
OUTPUT PARAMETERS:
*****
NONE
*****
CALLS PROCEDURES:
*****
DEFVIEW,DEFREL,DEFDOM,LEX,DEFINEV,DEFINEN
*****
/
/* COMMAND LISTS */
DCL OP(3) CHAR(8) INITIAL('VIEW','RELATION',
'DOMAIN'),
ADP(3) CHAR(8) INITIAL('V','R','D');
/* DEFINEN TABLE */
XINCLUDE DEFARG:*****
DCL 1 DEF_ARG, /* USED TO DEFINE AN NSET */
DCL00010

```

IE;

```

2 NNAME BIT(64),          /* NAME OF NSET */          DCL00020
2 NATTR BIT(8),           /* NUMBER OF ATTRIBUTES */ DCL00030
2 ATTR(20),               /* FOR EACH ATTRIBUTE */   DCL00040
    3 ANAME BIT(64),       /* ATTRIBUTE NAME */                 DCL00050
    3 K_TYPE BIT(8);       /* UNIQUE KEY OR NOT */                DCL00060
                                                                    DCL00070
*****
    /* DEFINEV TABLE */                                         D 00090
%INCLUDE DVARG;*****D 00100
    /* STRUCTURE USED TO PASS NAME OF VALUE SET TO BE DEFINED */ D 00110
4 1 0    DCL 1 DV_ARG,                                           DVA00010
    2 NAME BIT(64), /* NAME OF VALUE SET */                     DVA00020
    2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */                 DVA00030
*****DVA00040
                                                                    D 00110
                                                                    D 00120
                                                                    D 00130
    /* SUBROUTINES */                                           D 00140
%INCLUDE EDEFVW;*****DEC00020
    /* DEFINE VIEW MODULE */                                     DEC00030
5 1 0    DCL DEFVIEW ENTRY(FIXED BIN(15),(*) CHAR(8));          D 00140
*****D 00150
%INCLUDE EDEFREL;*****DEC00050
    /* DEFINE RELATION MODULE */                                 DEC00060
6 1 0    DCL DEFREL ENTRY(FIXED BIN(15),(*) CHAR(8));           D 00150
*****D 00160
%INCLUDE EDEFDOM;*****DEC00080
    /* DEFINE RELATION MODULE */                                 DEC00090
7 1 0    DCL DEFDOM ENTRY(CHAR(8),CHAR(8),CHAR(8),CHAR(8),CHAR(8)); D 00160
*****D 00170
%INCLUDE ELEX;*****DEC00110
    /* LEXICAL ANALYZER */                                     DEC00120
8 1 0    DCL LEX ENTRY(FIXED BIN(15),(*) CHAR(8));              D 00170
*****D 00180
%INCLUDE EDEFINV;*****DEC00140
    /* DEFINE ATTRIBUTE MODULE */                               DEC00150
9 1 0    DCL DEFINEV ENTRY(1, 2 BIT(64), 2 BIT(8));             D 00180
*****D 00190
%INCLUDE EDEFINN;*****DEC00170
    /* DEFINE NSET MODULE */                                    DEC00180
10 1 0    DCL DEFINEN ENTRY(1, 2 BIT(64), 2 BIT(8),              DEC00190
    2 (20), 3 BIT(64), 3 BIT(8));                                DEC00200
*****D 00190
                                                                    D 00200
                                                                    D 00210
                                                                    D 00220
                                                                    D 00230
                                                                    D 00240
                                                                    D 00250
                                                                    D 00260
11 1 0    /* MISCELLANEOUS */
    DCL VNAME(20) CHAR(8),
    N_TOK FIXED BIN(15),
    TRUE BIT(1) INIT('1'B),
    FLAG BIT(1) STATIC INIT('1'B);

```

```

12 1 0 /* DEFINE DICTIONARIES */
      IF FLAG
      THEN DO;
      /* DEFINE PSETS */
13 1 1 KEY_LEN = '00111000'B; NAME = UNSPEC('DATTR ');
15 1 1 CALL DEFINEV(DV_ARG);
16 1 1 KEY_LEN = '01000000'B; NAME = UNSPEC('DNAME ');
18 1 1 CALL DEFINEV(DV_ARG);
19 1 1 KEY_LEN = '01000000'B; NAME = UNSPEC('RNAME ');
21 1 1 CALL DEFINEV(DV_ARG);
22 1 1 KEY_LEN = '01000000'B; NAME = UNSPEC('ID ');
24 1 1 CALL DEFINEV(DV_ARG);

      /* DEFINE NSETS */
25 1 1 NATR = '00000010'B;
26 1 1 NNAME = UNSPEC('VTABLE '); ANAME(1) = UNSPEC('ID ');
28 1 1 ANAME(2) = UNSPEC('RNAME ');
29 1 1 K_TYPE(1),K_TYPE(2) = '00000000'B;
30 1 1 CALL DEFINEN(DEF_ARG);
31 1 1 NNAME = UNSPEC('RTABLE '); ANAME(1) = ANAME(2);
33 1 1 ANAME(2) = UNSPEC('DNAME ');
34 1 1 CALL DEFINEN(DEF_ARG);
35 1 1 NNAME = UNSPEC('DTABLE '); ANAME(1) = ANAME(2);
37 1 1 ANAME(2) = UNSPEC('DATTR '); K_TYPE(1) = '00000001'B;
39 1 1 CALL DEFINEN(DEF_ARG);
40 1 1 FLAG = '0'B;
41 1 1 END;

/* START SUBROUTINE */
42 1 0 DO WHILE(TRUE);
43 1 1 PUT SKIP LIST ('D:');
44 1 1 CALL LEX(N_TOK,VNAME);
45 1 1 IF N_TOK = 0 THEN RETURN;

      /* SELECT COMMAND */
46 1 1 SEL_COM: SELECT(VNAME(1));

      /* DEFINE VIEW */
47 1 2 WHEN(OP(1),AOP(1)) CALL DEFVIEW(N_TOK,VNAME);

      /* DEFINE RELATION */
48 1 2 WHEN(OP(2),AOP(2)) CALL DEFREL(N_TOK,VNAME);

      /* DEFINE DOMAIN */
49 1 2 WHEN(OP(3),AOP(3))
      CALL DEFDO(M(VNAME(2),VNAME(3),VNAME(4),VNAME(5),VNAME(6)));

      /* INVALID OPERATORS */

```

```

D 00270
D 00280
D 00290
D 00300
D 00310
D 00320
D 00330
D 00340
D 00350
D 00360
D 00370
D 00380
D 00390
D 00400
D 00410
D 00420
D 00430
D 00440
D 00450
D 00460
D 00470
D 00480
D 00490
D 00500
D 00510
D 00520
D 00530
D 00540
D 00550
D 00560
D 00570
D 00580
D 00590
D 00600
D 00610
D 00620
D 00630
D 00640
D 00650
D 00660
D 00670
D 00680
D 00690
D 00700
D 00710
D 00720
D 00730
D 00740
D 00750

```


IE;

50	1	2	OTHERWISE	D	00760
			DO;	D	00770
51	1	3	PUT SKIP(0) EDIT (VNAME(1), ' IS AN INVALID COMMAND')	D	00780
			(A,A);	D	00790
52	1	3	PUT SKIP LIST ('RETYPE COMMAND:');	D	00800
53	1	3	GET EDIT (VNAME(1)) (A(8));	D	00810
54	1	3	IF VNAME(1) ^= (8) ' THEN GO TO SEL_COM;	D	00820
55	1	3	END;	D	00830
56	1	2	END;	D	00840
57	1	1	END;	D	00850
				D	00860
58	1	0	END DEFINE;	D	00870

```

P
1;

%INCLUDE DEFDOM;*****DEF00010
/*****DEF00010
*DEF00020
*DEF00030
*DEF00040
*DEF00050
*DEF00060
*DEF00070
*DEF00080
*DEF00090
*DEF00100
*DEF00110
*DEF00120
*DEF00130
*DEF00140
*DEF00150
*DEF00160
*DEF00170
*DEF00180
*DEF00190
*DEF00200
*DEF00210
*DEF00220
*DEF00230
*DEF00240
*DEF00250
*DEF00260
*DEF00270
*DEF00280
*DEF00290
*DEF00300
*DEF00310
*DEF00320
*DEF00330
*DEF00340
*DEF00350
*DEF00360
*DEF00370
*DEF00380
*DEF00390
*DEF00400
*DEF00410
*DEF00420
*DEF00430
*DEF00440
*DEF00450

MODULE DESCRIPTION
*****/
0 DEFDOM: PROCEDURE
    ( TNAME, /* CHAR(8) */
      CTYPE, /* CHAR(8) */
      CLEN, /* CHAR(8) */
      CMIN, /* CHAR(8) */
      CMAX, /* CHAR(8) */
      );

PURPOSE:
THE PURPOSE OF THIS PROCEDURE IS TO ACCEPT REQUESTS
TO DEFINE DOMAINS, VALIDATE THE REQUESTS, THEN ISSUE
A CALL TO DEFINEV TO DEFINE A VALUE SET CORRESPONDING TO
THE DOMAIN, AND FINALLY INSERT THE DOMAIN NAME AND
DESCRIPTION INTO THE DTABLE RELATION.

METHOD:
LOGIC IS STRAIGHTFORWARD. SEVERAL NOTES, HOWEVER, 1)
USES A PRE-SPECIFIED COPY OF INSERT_ARG TO INSERT THE
DOMAIN NAME AND DESCRIPTION INTO THE DTABLE RELATION.
2) FETCHV IS CALLED, USING A PRE-SPECIFIED COPY OF
FV_ARG TO SEARCH DNAME FOR DOMAIN NAME.
3) ARGUMENTS TO DEFDOM ARE CREATED BY LEX WHEN IT
SCANS AN INPUT LINE. BLANKS ARE PASSED, IF ARGUMENTS
NOT SPECIFIED BY USER.
4) THE STRUCTURE 'RECORD' IS USED TO BUILD THE DATTR
STRING.

INPUT PARAMETERS:
ARGUMENTS CREATED BY THE LEX ROUTINE DURING ITS SCAN OF
AN INPUT LINE. IF ARGUMENTS MISSING, BLANKS ARE PASSED
AND DEFDOM PROMPTS THE USER FOR THE NECESSARY INFORMATION.
ARGUMENTS ARE AS FOLLOWS:
TNAME - NAME OF DOMAIN TO BE DEFINED
CTYPE - C OR N, INDICATING TYPE OF INFORMATION CONTAINED IN
DOMAIN.
CLEN - MAXIMUM PERMISSABLE LENGTH FOR ELEMENTS WITHIN THE
DOMAIN
CMIN - IF NUMERIC DATA, THE MINIMUM PERMISSABLE VALUE.
CMAX - IF NUMERIC DATA, THE MAXIMUM PERMISSABLE VALUE.

```

-187-

5 1 0

```

        /* INSERT NSET MODULE */
6 1 0      DCL INSERTN ENTRY(1, 2 BIT(64), 2 BIT(8), 2 (20),
                                3 BIT(64), 3 BIT(320));
        *****
        %INCLUDE EDEFINV;*****
        /* DEFINE ATTRIBUTE MODULE */
7 1 0      DCL DEFINEV ENTRY(1, 2 BIT(64), 2 BIT(8));
        *****
        %INCLUDE EFETCHV;*****
        /* USED TO FETCH INSTANCES OF DOMAINS */
8 1 0      DCL FETCHV ENTRY(1, 2 BIT(64), 2 BIT(160),
                                2 BIT(1), 2 BIT(320));
        *****

        /* ON CONDITIONS */
9 1 0      DCL ONSOURCE BUILTIN;
10 1 0     ON CONVERSION
        BEGIN;
11 2 0         PUT SKIP(0) LIST ('NUMERIC DATA REQUIRED. ');
12 2 0         ONSOURCE = '0';
13 2 0     END;

        /* SUBROUTINE STARTS */

        /* GET DOMAIN NAME */
14 1 0     IF TNAME = (8) ' '
        THEN DO;
15 1 1         PUT SKIP LIST ('DOMAIN NAME: ');
16 1 1         GET EDIT (TNAME) (A(8));
17 1 1         IF TNAME = (8) ' ' THEN RETURN;
18 1 1     END;

        /* VALIDATE DOMAIN NAME */
19 1 0     KEY_VAL = UNSPEC(TNAME);
20 1 0     CALL FETCHV(FV_ARG);
21 1 0     IF FOUND
        THEN DO;
22 1 1         PUT SKIP LIST ('DOMAIN ALREADY DEFINED. ');
23 1 1         RETURN;
24 1 1     END;

        /* GET DATA TYPE */
25 1 0     BTYPE = CTYPE;
26 1 0     DO WHILE(BTYPE ^= 'C' & BTYPE ^= 'N');
27 1 1         PUT SKIP LIST ('DATA TYPE: ');
28 1 1         GET EDIT (BTYPE) (A(1));
29 1 1         IF BTYPE = ' ' THEN RETURN;
30 1 1     END;

```

```

DEC00070
DEC00080
DEC00090
DEF00380
DEF00390
DEF00140
DEC00150
DEF00390
DEF00400
DEF00010
DEF00020
DEF00030
DEF00040
DEF00400
DEF00410
DEF00420
DEF00430
DEF00440
DEF00450
DEF00460
DEF00470
DEF00480
DEF00490
DEF00500
DEF00510
DEF00520
DEF00530
DEF00540
DEF00550
DEF00560
DEF00570
DEF00580
DEF00590
DEF00600
DEF00610
DEF00620
DEF00630
DEF00640
DEF00650
DEF00660
DEF00670
DEF00680
DEF00690
DEF00700
DEF00710
DEF00720
DEF00730
DEF00740
DEF00750

```

M;

31	1	0	/* GET MAXIMUM LENGTH */	DEF00760
			IF VERIFY(CLEN, '0123456789 ') = 0	DEF00770
			THEN IF CLEN = (8) ' '	DEF00780
			THEN BLEN = 0;	DEF00790
			ELSE BLEN = BIN(CLEN);	DEF00800
32	1	0	ELSE BLEN = 0;	DEF00810
33	1	0	DO WHILE(BLEN < 1 BLEN > 40);	DEF00820
34	1	0	PUT SKIP LIST ('MAXIMUM LENGTH: ');	DEF00830
35	1	1	GET LIST (BLEN);	DEF00840
36	1	1	IF BLEN = 0 THEN RETURN;	DEF00850
37	1	1	END;	DEF00860
38	1	1	/* CHECK IF NUMERAL DATA TYPE */	DEF00870
			IF BTYPE = 'N'	DEF00880
39	1	0	THEN DO;	DEF00890
			/* GET MINIMUM AND MAXIMUM VALUES */	DEF00900
			IF VERIFY(CMIN, '0123456789 ') = 0 & CMIN != (8) ' '	DEF00910
40	1	1	THEN BMIN = BIN(CMIN);	DEF00920
			ELSE DO;	DEF00930
41	1	1	PUT SKIP LIST ('MINIMUM VALUE: ');	DEF00940
42	1	2	GET LIST (BMIN);	DEF00950
43	1	2	IF BMIN = -1 THEN RETURN;	DEF00960
44	1	2	END;	DEF00970
45	1	2	IF VERIFY(CMAX, '0123456789 ') = 0	DEF00980
46	1	1	THEN IF CMAX = (8) ' '	DEF00990
			THEN BMAX = BMIN - 1;	DEF01000
			ELSE BMAX = BIN(CMAX);	DEF01010
47	1	1	ELSE BMAX = BMIN - 1;	DEF01020
48	1	1	DO WHILE(BMAX < BMIN);	DEF01030
49	1	1	PUT SKIP LIST ('MAXIMUM VALUE: ');	DEF01040
50	1	2	GET LIST (BMAX);	DEF01050
51	1	2	IF BMAX = -1 THEN RETURN;	DEF01060
52	1	2	END;	DEF01070
53	1	2	END;	DEF01080
54	1	1	/* CALL INSERTN TO ADD NEW DOMAIN TO TABLE */	DEF01090
			TTYPE = UNSPEC(BTYPE); TLEN = BLEN; TMIN = BMIN; TMAX = BMAX;	DEF01100
55	1	0	VALUE(1) = UNSPEC(TNAME);	DEF01110
59	1	0	VALUE(2) = DATTR;	DEF01120
60	1	0	CALL INSERTN(INSERT_ARG);	DEF01130
61	1	0	/* DETERMINE KEY LENGTH */	DEF01140
62	1	0	KLEN = MIN(TLEN*8, 64);	DEF01150
63	1	0	KEY_LEN = BIN(KLEN);	DEF01160
			/* CALL DEFINEV TO CREATE A PSET */	DEF01170
64	1	0	VNAME = UNSPEC(TNAME); KEY_LEN = '01000000'B;	DEF01180
66	1	0	CALL DEFINEV(DV_ARG);	DEF01190
				DEF01200
				DEF01210
				DEF01220
				DEF01230
				DEF01240

67 1 0

END DEFDOM;

DEF01250

AD-A116 593

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/6 9/2
INFOSAM: A SAMPLE DATABASE MANAGEMENT SYSTEM.(U)

DEC 81 B BLUMBERG

N00039-81-C-0663

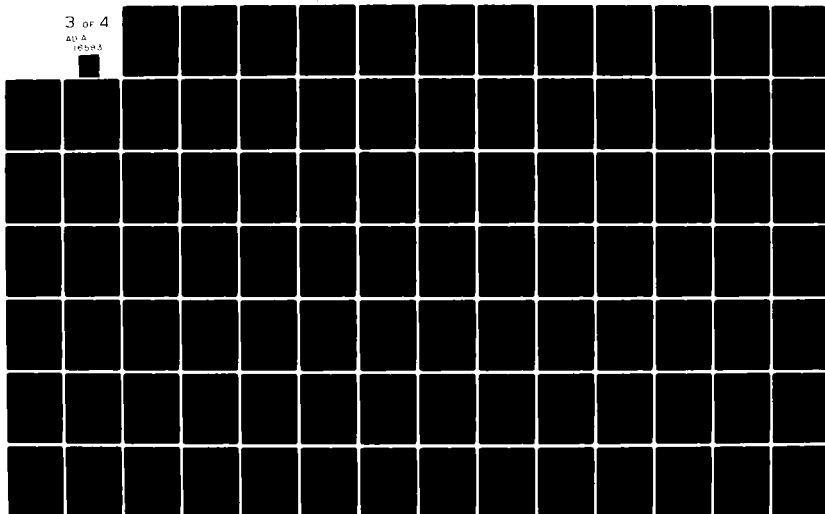
UNCLASSIFIED

CISR-M010-8112-07

NL

3 of 4

AD A
16883



```

XINCLUDE DEFREL;*****DEF00010
/*****DR 00010
*DR 00020
*      MODULE      DESCRIPTION      *DR 00030
*****DR 00040
1 0 DEFREL: PROCEDURE (N, /* FIXED BIN(15) */DR 00050
      TNAME /* (20) CHAR(8) */);DR 00060
/*****DR 00070
*****DR 00080
*****PURPOSE:DR 00090
*****THIS MODULE IS RESPONSIBLE FOR PROCESSING REQUESTS TODR 00100
*****DEFINE RELATIONS. IT VALIDATES THE REQUEST ( I.E CHECKS DR 00110
*****THAT ALL DOMAINS EXIST AND THAT RELATION NOT PREVIOUSLY DR 00120
*****DEFINED), ISSUES A CALL TO DEFINEN TO DEFINE THE NSET DR 00130
*****REPRESENTATION OF THE RELATION, AND THEN ISSUES A CALL DR 00140
*****TO INSERTN TO INSERT RELATION DEFINITION INTO RTABLE. DR 00150
*****DR 00160
*****DR 00170
*****METHOD:DR 00180
*****LOGIC IS SIMILAR TO THAT OF DEFOOM.NOTES:DR 00190
*****A) PRE-SPECIFIED COPY OF INSERT_ARG USED TO INSERT TUPLES DR 00200
*****INTO RTABLE. INSERTN MUST BE CALLED ONCE FOR EACH DOMAIN DR 00210
*****IN THE RELATION. DR 00220
*****DR 00230
*****DR 00240
*****INPUT PARAMETERS:DR 00250
*****ARGUMENTS CREATED BY LEX DURING ITS SCAN OF THE COMMAND DR 00260
*****ISSUED BY USER. ARGUMENTS NOT GIVEN BY USER ARE PASSED DR 00270
*****AS BLANKS. DR 00280
*****TNAME- ESSENTIALLY TOKEN CHAIN CREATED BY LEX. CONTAINS DR 00290
*****NAME OF RELATION AND NAMES OF DOMAINS WITHIN RELATION. DR 00300
*****N - NUMBER OF TOKENS IN CHAIN. DR 00310
*****DR 00320
*****DR 00330
*****OUTPUT PARAMETERS:DR 00340
*****NONE, BUT VIA ITS CALL TO INSERTN, IT INSERTS THE RELATIONDR 00350
*****DEFINITION INTO RTABLE, AND VIA ITS CALL TO DEFINEN IT DR 00360
*****CREATES AN NSET DEFINITION FOR THE NSET REPRESENTATION OF DR 00370
*****THE RELATION. DR 00380
*****DR 00390
*****DR 00400
*****CALLS PROCEDURES:DR 00410
*****DEFINEN,INSERTN,FETCHV,LEX DR 00420
*****DR 00430
*****DEF00010
*****DEF00020

```



```

/* DEFINEN TABLE */
XINCLUDE DEFARG;.....DEF00030
2 1 0 DCL 1 DEF_ARG, /* USED TO DEFINE AN NSET */ DEF00040
      2 NNAME BIT(64), /* NAME OF NSET */ DCL00010
      2 NATTR BIT(8), /* NUMBER OF ATTRIBUTES */ DCL00020
      2 ATTR(20), /* FOR EACH ATTRIBUTE */ DCL00030
          3 ANAME BIT(64), /* ATTRIBUTE NAME */ DCL00040
          3 K_TYPE BIT(8); /* UNIQUE KEY OR NOT */ DCL00050
      ..... DCL00060
      ..... DCL00070
      ..... DEF00040
      ..... DEF00050
      ..... DEF00060
      ..... DEF00070
      ..... DEF00080
      ..... DEF00090
      ..... DEF00100
      ..... DEF00110
      ..... DEF00120
      ..... DEF00130
      ..... DEF00140
      ..... DEF00150
/* INSERTN TABLE */
3 1 0 DCL 1 INSERT_ARG, DEF00160
      2 NNAME BIT(64) INIT(UNSPEC('RTABLE ')), DEF00010
      2 NATTR BIT(8) INIT('00000010'B), DEF00020
      2 ATTR(20), DEF00030
          3 ANAME BIT(64) INIT(UNSPEC('RNAME ')), DEF00040
          UNSPEC('DNAME '), DEF00050
      3 VALUE BIT(320); DEF00060
      ..... DEF00070
      ..... DEF00080
      ..... DEF00090
      ..... DEF00100
      ..... DEF00110
      ..... DEF00120
      ..... DEF00130
      ..... DEF00140
      ..... DEF00150
/* FETCHV TABLE */
XINCLUDE FVARG;.....DEF00160
/* FETCHV TABLE -USED TO RETRIEVE INSTANCES OF A DOMAIN */ DEF00010
4 1 0 DCL 1 FV_ARG, DEF00020
      2 D_NAME BIT(64), /* NAME OF DOMAIN */ DEF00030
      2 KEY_VAL BIT(160), /* KEY TO SEARCH ON */ DEF00040
      2 FOUND BIT(1), /* IF FOUND, '1'B, OTHERWISE '0'B */ DEF00050
      2 DATA BIT(320); /* RETRIEVED ELEMENT */ DEF00060
      ..... DEF00070
      ..... DEF00080
      ..... DEF00090
      ..... DEF00100
      ..... DEF00110
      ..... DEF00120
      ..... DEF00130
      ..... DEF00140
      ..... DEF00150
/* SUBROUTINES */
XINCLUDE EDEFINN;.....DEF00160
/* DEFINE NSET MODULE */ DEF00010
5 1 0 DCL DEFINEN ENTRY(1, 2 BIT(64), 2 BIT(8), DEF00020
      2 (20), 3 BIT(64), 3 BIT(8)); DEC00170
      ..... DEC00180
      ..... DEC00190
      ..... DEC00200
      ..... DEF00190
XINCLUDE EINERN;.....DEF00200
/* INSERT NSET MODULE */ DEF00010
6 1 0 DCL INSERTN ENTRY(1, 2 BIT(64), 2 BIT(8), 2 (20), DEC00070
      3 BIT(64), 3 BIT(320)); DEC00080
      ..... DEC00090
      ..... DEF00200
XINCLUDE EFETCHV;.....DEF00210
/* USED TO FETCH INSTANCES OF DOMAINS */ DEF00010
7 1 0 DCL FETCHV ENTRY(1, 2 BIT(64), 2 BIT(160), DEF00020
      2 BIT(1), 2 BIT(320)); DEF00030
      ..... DEF00040
      ..... DEF00050
      ..... DEF00060
      ..... DEF00070
      ..... DEF00080
      ..... DEF00090
      ..... DEF00100
      ..... DEF00110
      ..... DEF00120
      ..... DEF00130
      ..... DEF00140
      ..... DEF00150
      ..... DEF00160
      ..... DEF00170
      ..... DEF00180
      ..... DEF00190
      ..... DEF00200
      ..... DEF00210

```

		%INCLUDE ELEX;*****	DEF00220	
		/* LEXICAL ANALYZER */	DEC00110	
8	1	0	DCL LEX ENTRY(FIXED BIN(15),(*) CHAR(8));	DEC00120
		*****	DEF00220	
			DEF00230	
		/* MISCELLANEOUS */	DEF00240	
9	1	0	DCL TNAME(20) CHAR(8),	DEF00250
		TREL CHAR(8),	DEF00260	
		IND(10) FIXED BIN(15),	DEF00270	
		TNUM FIXED BIN(8),	DEF00280	
		(1,I,M2) FIXED BIN(15),	DEF00290	
		TRUE BIT(1) INIT('1'B);	DEF00300	
			DEF00310	
		/* START SUBROUTINE */	DEF00320	
10	1	0	TREL = TNAME(2);	DEF00330
			DEF00340	
		/* GET RELATION NAME */	DEF00350	
11	1	0	IF N = 1	DEF00360
		THEN DO;	DEF00370	
12	1	1	PUT SKIP LIST('RELATION NAME:');	DEF00380
13	1	1	SET EDIT (TREL) (A(8));	DEF00390
14	1	1	IF TREL = (8)' ' THEN RETURN;	DEF00400
15	1	1	END;	DEF00410
			DEF00420	
		/* VALIDATE RELATION NAME */	DEF00430	
16	1	0	D_NAME = UNSPEC('RNAME '); KEY_VAL = UNSPEC(TREL);	DEF00440
18	1	0	CALL FETCHV(FV_ARG);	DEF00450
19	1	0	IF FOUND	DEF00460
		THEN DO;	DEF00470	
20	1	1	PUT SKIP LIST ('RELATION ALREADY DEFINED.');	DEF00480
21	1	1	RETURN;	DEF00490
22	1	1	END;	DEF00500
			DEF00510	
		/* MOVE DOMAIN NAMES DOWN IN ARRAY */	DEF00520	
23	1	0	DO I = 3 TO 12;	DEF00530
24	1	1	TNAME(I-2) = TNAME(I);	DEF00540
25	1	1	END;	DEF00550
26	1	0	N = N - 2;	DEF00560
			DEF00570	
		/* CHECK FOR CORRECT NUMBER OF DOMAINS */	DEF00580	
27	1	0	IF N > 10	DEF00590
		THEN PUT SKIP(0) LIST ('TOO MANY DOMAINS SPECIFIED.');	DEF00600	
28	1	0	IF N > 10 ; N < 1	DEF00610
		THEN DO WHILE(TRUE);	DEF00620	
29	1	1	PUT SKIP LIST ('DOMAIN NAMES:');	DEF00630
30	1	1	CALL LEX(N,TNAME);	DEF00640
31	1	1	IF N = 0 THEN RETURN;	DEF00650
32	1	1	IF N < 11 THEN LEAVE;	DEF00660
33	1	1	PUT SKIP(0) LIST ('TOO MANY DOMAINS SPECIFIED.');	DEF00670

34	1	1	END;	DEF00680
			/* VALIDATE DOMAIN NAMES */	DEF00690
35	1	0	D_NAME = UNSPEC('DNAME ');	DEF00700
36	1	0	DO I = 1 TO N;	DEF00710
37	1	1	DO WHILE(TRUE);	DEF00720
38	1	2	KEY_VAL = UNSPEC(TNAME(I));	DEF00730
39	1	2	CALL FETCHV(FV_ARG);	DEF00740
40	1	2	IF FOUND THEN LEAVE;	DEF00750
41	1	2	PUT SKIP EDIT ('DOMAIN ',TNAME(I),' UNDEFINED:')	DEF00760
			(A,A,A);	DEF00770
42	1	2	GET EDIT (TNAME(I)) (A(8));	DEF00780
43	1	2	IF TNAME(I) = (8) ' ' THEN RETURN;	DEF00790
44	1	2	END;	DEF00800
45	1	1	END;	DEF00810
			/* CALL INSERTN TO ADD NEW RELATION TO TABLE */	DEF00820
46	1	0	VALUE(1) = UNSPEC(TREL);	DEF00830
47	1	0	DO I = 1 TO N;	DEF00840
48	1	1	VALUE(2) = UNSPEC(TNAME(I));	DEF00850
49	1	1	CALL INSERTN(INSERT_ARG);	DEF00860
50	1	1	END;	DEF00870
			/* CALL DEFINEN TO CREATE A NEW NSET */	DEF00880
51	1	0	DEF_ARG.NNAME = UNSPEC(TREL); TNUM = N; DEF_ARG.NATTR = TNUM;	DEF00890
54	1	0	DO I = 1 TO N;	DEF00900
55	1	1	DEF_ARG.ANAME(I) = UNSPEC(TNAME(I));	DEF00910
56	1	1	END;	DEF00920
57	1	0	K_TYPE = '00000000'B;	DEF00930
58	1	0	PUT SKIP LIST ('UNIQUE DOMAIN INDEXES:');	DEF00940
59	1	0	CALL LE (N2,TNAME);	DEF00950
60	1	0	DO I = 1 TO N2;	DEF00960
61	1	1	IF VERIFY(TNAME(I),' 123456789') = 0 &	DEF00970
			BIN(TNAME(I)) >= 1 & BIN(TNAME(I)) <= N2	DEF00980
			THEN K_TYPE(BIN(TNAME(I))) = '00000001'B;	DEF00990
62	1	1	END;	DEF01000
63	1	0	CALL DEFINEN(DEF_ARG);	DEF01010
64	1	0	END DEFREL;	DEF01020
				DEF01030
				DEF01040
				DEF01050
				DEF01060

```

%INCLUDE DEFVIEW;*****DEF00010
/*****FOR00010
*FOR00020
*MODULE DESCRIPTION*FOR00030
*****/FOR00040
1 0 DEFVIEW: PROCEDURE
      (N, /* FIXED BIN(15) */
       TNAME /* (20) CHAR(8) */);
/*****FOR00050
*****PURPOSE:FOR00060
*****THIS MODULE IS RESPONSIBLE FOR PROCESSING REQUESTS TOFOR00070
*****DEFINE VIEWS, WHERE A VIEW IS A COLLECTION OF PREVIOUSLYFOR00080
*****DEFINED RELATIONS TO WHICH A USER MAY HAVE ACCESS. EVERYFOR00090
*****VIEW IS IDENTIFIED BY A UNIQUE ID. THIS MODULE VALIDATESFOR00100
*****THE REQUEST (CHECKS THAT THE RELATIONS EXIST, AND THATFOR00110
*****THE VIEW I.D. IS UNIQUE), AND THEN INSERTS THE VIEWFOR00120
*****DESCRIPTION INTO VTABLE VIA A CALL TO INSERTN.FOR00130
*****FOR00140
*****FOR00150
*****FOR00160
*****FOR00170
*****METHOD:FOR00180
*****LOGIC IS VIRTUALLY THE SAME AS DEFREL AND DEFDOM.FOR00190
*****FOR00200
*****FOR00210
*****INPUT PARAMETERS:FOR00220
*****ARGUMENTS PASSED ARE CREATED BY LEX DURING ITS SCAN OF THEFOR00230
*****USERS COMMAND LINE.FOR00240
*****N - THE NUMBER OF TOKENS FOUND IN THE COMMAND LINE.FOR00250
*****TNAME - THE TOKEN CHAIN CREATED BY LEX CONTAINING THEFOR00260
*****VIEW ID AND THE NAME OF THE RELATIONS CONTAINEDFOR00270
*****IN THE VIEW.FOR00280
*****FOR00290
*****FOR00300
*****OUTPUT PARAMETERS:FOR00310
*****NONE, HOWEVER, IT DOES INSERT ENTRIES INTO THE VTABLE ASFOR00320
*****A RESULT OF ITS CALLS TO INSERTN. THESE ENTRIES CORRESPONDFOR00330
*****TO THE VIEW DEFINITIONS.FOR00340
*****FOR00350
*****FOR00360
*****CALLS PROCEDURES:FOR00370
*****INSERTN, FETCHV, LEXFOR00380
*****/FOR00390
*****DEF00010
*****DEF00020
*****DEF00030
*****DEF00040
*****DEF00050
*****DEF00060

/* INSERTN TABLE */
DCL 1 INSERT_ARG,
      2 NNAME BIT(64) INIT(UNSPEC('VTABLE ')),
      2 NATTR BIT(8) INIT('00000010'B),

```

```

2 ATTR(20),
3 ANAME BIT(64) INIT(UNSPEC('ID
                                UNSPEC('RNAME
                                '))).
3 VALUE BIT(320);
DEF00070
DEF00080
DEF00090
DEF00100
DEF00110
DEF00120
DEF00130
DEF00010
DEF00020
DEF00030
DEF00040
DEF00050
DEF00060
DEF00070
DEF00130
DEF00140
DEF00150
DEF00160
DEC00070
DEC00080
DEC00090
DEF00160
DEF00170
DEF00010
DEF00020
DEF00030
DEF00040
DEF00170
DEF00180
DEC00110
DEC00120
DEF00180
DEF00190
DEF00200
DEF00210
DEF00220
DEF00230
DEF00240
DEF00250
DEF00260
DEF00270
DEF00280
DEF00290
DEF00300
DEF00310
DEF00320
DEF00330
DEF00340
DEF00350

/* FETCHV TABLE */
%INCLUDE FVARG;*****
/* FETCHV TABLE -USED TO RETRIEVE INSTANCES OF A DOMAIN */
3 1 0 DCL 1 FV_ARG,
      2 D_NAME BIT(64), /* NAME OF DOMAIN */
      2 KEY_VAL BIT(160), /* KEY TO SEARCH ON */
      2 FOUND BIT(1), /* IF FOUND, '1'B, OTHERWISE '0'B */
      2 DATA BIT(320); /* RETRIEVED ELEMENT */
*****
/* SUBROUTINES */
%INCLUDE EINSERT;*****
/* INSERT NSET MODULE */
4 1 0 DCL INSERTN ENTRY(1, 2 BIT(64), 2 BIT(8), 2 (20),
                        3 BIT(64), 3 BIT(320));
*****
%INCLUDE EFETCHV;*****
/* USED TO FETCH INSTANCES OF DOMAINS */
5 1 0 DCL FETCHV ENTRY(1, 2 BIT(64), 2 BIT(160),
                        2 BIT(1), 2 BIT(320));
*****
%INCLUDE ELEX;*****
/* LEXICAL ANALYZER */
6 1 0 DCL LEX ENTRY(FIXED BIN(15), (*) CHAR(8));
*****
/* MISCELLANEOUS */
7 1 0 DCL TNAME(20) CHAR(8),
      TVIEW CHAR(8),
      (I,N) FIXED BIN(15),
      TRUE BIT(1) INITIAL('1'B);
/* START SUBROUTINE */
8 1 0 TVIEW = TNAME(2);
/* GET VIEW ID */
9 1 0 IF N = 1
      THEN DO;
10 1 1 PUT SKIP LIST('VIEW ID:');
11 1 1 GET EDIT (TVIEW) (A(8));
12 1 1 IF TVIEW = (8)' ' THEN RETURN;
13 1 1 END;

```

14 1 0
16 1 0
17 1 0

18 1 1
19 1 1
20 1 1

21 1 0
22 1 1
23 1 1
24 1 0

25 1 0

26 1 0

27 1 1
28 1 1
29 1 1
30 1 1
31 1 1
32 1 1

33 1 0
34 1 0
35 1 1
36 1 2
37 1 2
38 1 2
39 1 2

40 1 2
41 1 2
42 1 2
43 1 1

44 1 0
45 1 0
46 1 1
47 1 1
48 1 1

```

/* VALIDATE VIEW ID */
D_NAME = UNSPEC('ID '); KEY_VAL = UNSPEC(TVIEW);
CALL FETCHV(FV_ARG);
IF FOUND
THEN DO;
    PUT SKIP LIST ('ID ALREADY DEFINED:');
    RETURN;
END;

/* MOVE DOWN RELATION NAMES IN ARRAY */
DO I = 3 TO 12;
    TNAME(I-2) = TNAME(I);
END;
N = N - 2;

/* CHECK FOR CORRECT NUMBER OF RELATIONS */
IF N > 10
THEN PUT SKIP(0) LIST ('TOO MANY RELATIONS SPECIFIED. ');
IF N > 10 ; N < 1
THEN DO WHILE(TRUE);
    PUT SKIP LIST ('RELATION NAMES: ');
    CALL LEX(N,TNAME);
    IF N = 0 THEN RETURN;
    IF N < 11 THEN LEAVE;
    PUT SKIP(0) LIST ('TOO MANY RELATIONS SPECIFIED. ');
END;

/* VALIDATE RELATION NAMES */
D_NAME = UNSPEC('RNAME ');
DO I = 1 TO N;
    DO WHILE(TRUE);
        KEY_VAL = UNSPEC(TNAME(I));
        CALL FETCHV(FV_ARG);
        IF FOUND THEN LEAVE;
        PUT SKIP EDIT ('RELATION ',TNAME(I),' UNDEFINED: ')
            (A,A,A);
        GET EDIT (TNAME(I)) (A(8));
        IF TNAME(I) = (8) ' ' THEN RETURN;
    END;
END;

/* CALL INSERTN TO ADD NEW VIEW ID TO TABLE */
VALUE(1) = UNSPEC(TVIEW);
DO I = 1 TO N;
    VALUE(2) = UNSPEC(TNAME(I));
    CALL INSERTN(INSERT_ARG);
END;

```

DEF00360
DEF00370
DEF00380
DEF00390
DEF00400
DEF00410
DEF00420
DEF00430
DEF00440
DEF00450
DEF00460
DEF00470
DEF00480
DEF00490
DEF00500
DEF00510
DEF00520
DEF00530
DEF00540
DEF00550
DEF00560
DEF00570
DEF00580
DEF00590
DEF00600
DEF00610
DEF00620
DEF00630
DEF00640
DEF00650
DEF00660
DEF00670
DEF00680
DEF00690
DEF00700
DEF00710
DEF00720
DEF00730
DEF00740
DEF00750
DEF00760
DEF00770
DEF00780
DEF00790
DEF00800
DEF00810
DEF00820
DEF00830
DEF00840

49 1 0

END DEFVIEW;

DEF00850

```

%INCLUDE QUERY;*****QUE00010
/*****DOC00310
*DOC00320
*MODULE DESCRIPTIONDOC00330
*****/DOC00340
1 0 QUERY: PROCEDURE;DOC00350
/*****DOC00360
*****DOC00370
*****PURPOSE: PROMPTS FOR DISPLAY, INSERT, RELATIONAL OPERATOR, ANDDOC00380
*****PRINT COMMANDS.DOC00390
*****DOC00400
*****DOC00410
*****DOC00420
*****METHOD: NOT SIGNIFICANTDOC00430
*****DOC00440
*****DOC00450
*****INPUT PARAMETERS:DOC00460
*****NONEDOC00470
*****DOC00480
*****DOC00490
*****OUTPUT PARAMETERS:DOC00500
*****DOC00510
*****OUTPUT: NONEDOC00520
*****DOC00530
*****CALLS PROCEDURES:DOC00540
*****DOC00550
*****GETVIEW, VIEWCAT, SHVIEW, SHREL, JOIN, SELECT,DOC00560
*****DOC00570
*****/QUE00010
*****QUE00020
/* CURRENT VIEW ID */QUE00030
DCL C_ID CHAR(8) EXTERNAL,QUE00040
GET BIT(1) STATIC EXTERNAL INIT('1'B);QUE00050
QUE00060
/* COMMAND LISTS */QUE00070
DCL OP(10) CHAR(8) INITIAL('GET_VIEW','CUR_VIEW',QUE00080
'VIEW_CAT','SH_VIEW','SH_REL','JOIN',QUE00090
'SELECT','PROJECT','PRINT','LOAD'),QUE00100
AOP(10) CHAR(8) INITIAL('GV','CV','VC','SV','SR',QUE00110
'J','S','PJ','PR','L');QUE00120
QUE00130
/* SUBROUTINES */QUE00140
%INCLUDE EGETVIEW;*****QUE00150
DCL GETVIEW ENTRY(CHAR(8));QUE00020
*****QUE00150
%INCLUDE ESHVIEW;*****QUE00160

```



```

5 1 0      DCL SHVIEW ENTRY;                                QUE00060
      .....                                                QUE00160
%INCLUDE ESHREL;.....QUE00170
6 1 0      DCL SHREL ENTRY(CHAR(8));                        QUE00080
      .....                                                QUE00170
%INCLUDE EJOIN;.....QUE00180
7 1 0      DCL JOIN ENTRY((*) CHAR(8));                     QUE00100
      .....                                                QUE00180
%INCLUDE ESELECT;.....QUE00190
8 1 0      DCL SELECT ENTRY((*) CHAR(8));                   QUE00120
      .....                                                QUE00190
%INCLUDE EPROJET;.....QUE00200
9 1 0      DCL PROJECT ENTRY((*) CHAR(8));                  QUE00140
      .....                                                QUE00200
%INCLUDE EPRINT;.....QUE00210
10 1 0     DCL PRINT ENTRY(CHAR(8));                         QUE00160
      .....                                                QUE00210
%INCLUDE ELOAD;.....QUE00220
11 1 0     DCL LOAD ENTRY(CHAR(8));                          QUE00180
      .....                                                QUE00220
%INCLUDE ELEX;.....QUE00230
      /* LEXICAL ANALYZER */                                DEC00110
12 1 0     DCL LEX ENTRY(FIXED BIN(15),(*) CHAR(8));        DEC00120
      .....                                                QUE00230
%INCLUDE EDELIM;.....QUE00240
13 1 0     DCL DELIM ENTRY(BIT(1)) RETURNS(BIT(1));        QUE00200
      .....                                                QUE00210
      .....                                                QUE00240
      .....                                                QUE00250
      .....                                                QUE00260
14 1 0     /* MISCELLANEOUS */                               QUE00270
      DCL NAME(20) CHAR(8),                                QUE00280
      N_TOK FIXED BIN(15),                                QUE00290
      TRUE BIT(1) INIT('1'B);                             QUE00300
      .....                                                QUE00310
15 1 0     /* START SUBROUTINE */                             QUE00320
      PUT SKIP(2) LIST ('-- READY FOR QUERIES --');        QUE00330
      .....                                                QUE00340
      /* GET QUERIES */                                       QUE00350
16 1 0     DO WHILE(TRUE);                                   QUE00360
17 1 1         PUT SKIP(2) LIST ('Q:');                     QUE00370
18 1 1         CALL LEX(N_TOK,NAME);                        QUE00380
19 1 1         IF N_TOK = 0 THEN RETURN;                     QUE00390
      .....                                                QUE00400
20 1 1     /* SELECT COMMAND */                               QUE00410
      SEL_COM: SELECT(NAME(1));                             QUE00420
      .....                                                QUE00430
21 1 2     /* GET VIEW */                                    QUE00440
      WHEN(OP(1),AOP(1)) CALL GETVIEW(NAME(2));            QUE00450

```

Y;

22	1	2	/* CURRENT VIEW */	QUE00460
			WHEN(OP(2),AOP(2))	QUE00470
			IF GET	QUE00480
23	1	2	THEN PUT SKIP LIST ('NO VIEW LOADED YET.');	QUE00490
			ELSE PUT SKIP EDIT (C_ID) (X(10),A);	QUE00500
				QUE00510
24	1	2	/* VIEW CATALOGUE */	QUE00520
			WHEN(OP(3),AOP(3)) CALL VIEWCAT;	QUE00530
				QUE00540
			/* SHOW VIEW */	QUE00550
25	1	2	WHEN(OP(4),AOP(4))	QUE00560
			IF GET	QUE00570
			THEN PUT SKIP LIST ('NO VIEW LOADED YET.');	QUE00580
26	1	2	ELSE CALL SHVIEW;	QUE00590
				QUE00600
			/* SHOW RELATION */	QUE00610
27	1	2	WHEN(OP(5),AOP(5))	QUE00620
			IF GET	QUE00630
			THEN PUT SKIP LIST ('NO VIEW LOADED YET.');	QUE00640
28	1	2	ELSE CALL SHREL(NAME(2));	QUE00650
				QUE00660
			/* JOIN */	QUE00670
29	1	2	WHEN(OP(6),AOP(6))	QUE00680
			IF DELIM('0'B) = '0'B THEN CALL JOIN(NAME);	QUE00690
				QUE00700
			/* SELECT */	QUE00710
30	1	2	WHEN(OP(7),AOP(7))	QUE00720
			IF DELIM('0'B) = '0'B THEN CALL SELECT(NAME);	QUE00730
				QUE00740
			/* PROJECT */	QUE00750
31	1	2	WHEN(OP(8),AOP(8))	QUE00760
			IF DELIM('0'B) = '0'B THEN CALL PROJECT(NAME);	QUE00770
				QUE00780
			/* PRINT */	QUE00790
32	1	2	WHEN(OP(9),AOP(9)) CALL PRINT(NAME(2));	QUE00800
				QUE00810
			/* LOAD */	QUE00820
33	1	2	WHEN(OP(10),AOP(10))	QUE00830
			IF GET	QUE00840
			THEN PUT SKIP LIST ('NO VIEW LOADED YET.');	QUE00850
34	1	2	ELSE CALL LOAD(NAME(2));	QUE00860
				QUE00870
			/* INVALID OPERATORS */	QUE00880
35	1	2	OTHERWISE	QUE00890
			DO;	QUE00900
36	1	3	PUT SKIP EDIT (NAME(1),' IS AN INVALID COMMAND') (A,A);	QUE00910
37	1	3	PUT SKIP LIST ('RETYPE COMMAND:');	QUE00920
38	1	3	GET EDIT (NAME(1)) (A(8));	QUE00930
39	1	3	IF NAME(1) ^= (8) ' ' THEN GO TO SEL_COM;	QUE00940

Y;

40	1	3	
41	1	2	END;
42	1	1	END;
43	1	0	END QUERY;

QUE00950
QUE00960
QUE00970
QUE00980
QUE00990

```

%INCLUDE GETVIEW;*****GET00010
/***** DOC00590
* DOC00600
* DOC00610
* MODULE DESCRIPTION * DOC00620
*****/ DOC00630
1 0 GETVIEW: PROCEDURE(TNAME /* CHAR(8) */); DOC00640
/***** DOC00650
***** PURPOSE: DOC00660
***** LOAD A VIEW FOR QUERIES BY CREATING RELATION TABLES
***** ALONG WITH THEIR DOMAINS FROM ALL RELATIONS IN THE
***** PARTICULAR VIEW. DOC00670
***** DOC00680
***** DOC00690
***** DOC00700
***** METHOD: DOC00710
***** NOT SIGNIFICANT DOC00720
***** DOC00730
***** DOC00740
***** INPUT PARAMETERS: DOC00750
***** 1) TNAME - NAME OF VIEW DOC00760
***** DOC00770
***** DOC00780
***** OUTPUT PARAMETERS: DOC00790
***** NONE DOC00800
***** DOC00810
***** DOC00820
***** CALLS PROCEDURES: DOC00830
***** FETCHT, FETCHV DOC00840
***** DOC00850
***** DOC00860
*****/ DOC00860
***** GET00010
***** GET00020
***** /* CURRENT VIEW INDEX */ GET00030
***** DCL C_ID CHAR(8) EXTERNAL, GET00040
***** GET BIT(1) STATIC EXTERNAL; GET00050
***** GET00060
***** GET00070
***** /* FETCHT TABLE */ GET00080
%INCLUDE RETARG;*****GET00080
3 1 0 DCL 1 RET_ARG, /* USED TO RETRIEVE NSETS */ DCL00090
2 NUMN BIT(8), /* NUMBER OF NSETS */ DCL00100
2 NSET(5) BIT(64), /* NAMES OF NSETS TO BE FETCHED*/DCL00110
2 ARGS(20), /* INFO FOR EACH ATTRIBUTE */ DCL00120
3 N_INDEX BIT(8), /* WHICH NSET IS THIS IN */ DCL00130
3 NAME BIT(64), /* NAME OF ATTRIBUTE */ DCL00140
3 RET_INFO, /* RETRIEVE INFORMATION */ DCL00150
(4 FETCH, /* IS IT TO BE FETCHED */ DCL00160
4 SAME ) BIT(8), /* SAME AS PREVIOUSLY DCL00170

```

```

4 VALUE BIT(160); /* VALUE TO SEARCH ON OR
                     NONE */
*****
/* FETCHT OUTPUT */
%INCLUDE DOMRET;*****
/* FETCHT OUTPUT */
4 1 0 DCL 1 DOM_RET CONTROLLED EXTERNAL, /* STACK OF DATA VALUES */
      2 D_ID FIXED BIN(15), /* NSET AND ATTRIBUTE ID */
      2 VALUE BIT(320); /* DATA VALUE */
*****
/* FETCHV TABLE */
5 1 0 DCL 1 FV_ARG,
      2 D_NAME BIT(64),
      2 KEY_VAL BIT(160),
      2 FOUND BIT(1),
      2 DUMMY BIT(320);
/* RELATION TABLES */
6 1 0 DCL 1 T1_ARG(20) EXTERNAL,
      2 N1 BIT(8),
      2 C1(5) BIT(64),
      2 T2(20),
      3 N2 BIT(8),
      3 C2 BIT(64),
      3 T3,
      4 N3 BIT(8),
      4 N4 BIT(8),
      4 C3 BIT(160),
      R_IND FIXED BIN(15) EXTERNAL,
      REL(20) CHAR(8) EXTERNAL,
      NDOM(0:20) FIXED BIN(15) EXTERNAL;
/* SUBROUTINES */
%INCLUDE EFETCHT;*****
/* FETCH NSET MODULE */
7 1 0 DCL FETCHT ENTRY(1, 2 BIT(8), 2 (5) BIT(64), 2 (20), 3 BIT(8),
                     3 BIT(64), 3, 4 BIT(8), 4 BIT(8), 4 BIT(160));
*****
%INCLUDE EFETCHV;*****
/* USED TO FETCH INSTANCES OF DOMAINS */
8 1 0 DCL FETCHV ENTRY(1, 2 BIT(64), 2 BIT(160),
                     2 BIT(1), 2 BIT(320));
*****
/* MISCELLANEOUS */

```

```

DCL00180
DCL00190
DCL00200
GET00080
GET00090
GET00100
GET00110
DEC00020
DEC00030
DEC00040
DEC00050
GET00110
GET00120
GET00130
GET00140
GET00150
GET00160
GET00170
GET00180
GET00190
GET00200
GET00210
GET00220
GET00230
GET00240
GET00250
GET00260
GET00270
GET00280
GET00290
GET00300
GET00310
GET00320
GET00330
GET00340
GET00350
GET00360
DEC00110
DEC00120
DEC00130
GET00360
GET00370
DEF00010
DEF00020
DEF00030
DEF00040
GET00370
GET00380
GET00390

```

9	1	0	DCL TNAME CHAR(8), (BUFF,TEMP) BIT(64), (I,D_IND) FIXED BIN(15);	GET00400
			/* START SUBROUTINE */	GET00410
10	1	0	/* GET VIEW ID */	GET00420
			IF TNAME = (8) ' '	GET00430
			THEN DO;	GET00440
11	1	1	PU SKIP LIST ('VIEW ID:');	GET00450
12	1	1	GE EDIT (TNAME) (A(8));	GET00460
13	1	1	IF TNAME = (8) ' ' THEN RETURN;	GET00470
14	1	1	END;	GET00480
			/* VALIDATE VIEW ID */	GET00490
15	1	0	D_NAME = UNSPEC('ID '); KEY_VAL = UNSPEC(TNAME);	GET00500
17	1	0	CALL FETCHV(FV_ARG);	GET00510
18	1	0	IF ^FOUND	GET00520
			THEN DO;	GET00530
19	1	1	PUT SKIP EDIT ('ID ',TNAME,' UNDEFINED.') (A,A,A);	GET00540
20	1	1	RETURN;	GET00550
21	1	1	END;	GET00560
22	1	0	C_ID = TNAME; GET = '0'B;	GET00570
			/* CALL FETCH TO LOAD VIEW */	GET00580
24	1	0	NUMN = '00000010'B; NSET(1) = UNSPEC('RTABLE ');	GET00590
26	1	0	NSET(2) = UNSPEC('VTABLE ');	GET00600
27	1	0	N_INDEX(1),N_INDEX(2) = '00000001'B;	GET00610
28	1	0	N_INDEX(3),N_INDEX(4) = '00000010'B;	GET00620
29	1	0	NAME(2) = UNSPEC('DNAME ');	GET00630
30	1	0	NAME(1),NAME(3) = UNSPEC('RNAME ');	GET00640
31	1	0	NAME(4) = UNSPEC('ID ');	GET00650
32	1	0	FETCH(4) = '00000000'B;	GET00660
33	1	0	FETCH(2),FETCH(3),FETCH(1) = '10000000'B;	GET00670
34	1	0	SAME(1),SAME(2),SAME(4) = '00000000'B;	GET00680
35	1	0	SAME(3) = '00010001'B;	GET00690
36	1	0	RET_ARG.VALUE = '01010101'B; RET_ARG.VALUE(4) = UNSPEC(TNAME);	GET00700
38	1	0	CALL FETCHT(RET_ARG);	GET00710
			/* CREATE RET_ARGS FOR RELATIONS */	GET00720
39	1	0	N1 = '00000001'B; N2 = '00000001'B; N3 = '10000000'B;	GET00730
42	1	0	N4 = '00000000'B; C3 = '01010101'B;	GET00740
44	1	0	TEMP = UNSPEC((8) ' ');	GET00750
45	1	0	R_IND,D_IND = 0;	GET00760
46	1	0	DO I = 1 TO ALLOCATION(DOM_RET)/2;	GET00770
47	1	1	BUFF = DOM_RET.VALUE;	GET00780
48	1	1	IF BUFF ^= TEMP	GET00790
			THEN DO;	GET00800
49	1	2	NDOM(R_IND) = D_IND;	GET00810
				GET00820
				GET00830
				GET00840
				GET00850
				GET00860
				GET00870
				GET00880

50	1	2	R_IND = R_IND + 1;	GET00890
51	1	2	C1(R_IND,1) = BUFF;	GET00900
52	1	2	D_IND = 0;	GET00910
53	1	2	UNSPEC(REL(R_IND)) = BUFF;	GET00920
54	1	2	TEMP = BUFF;	GET00930
55	1	2	END;	GET00940
56	1	1	FREE DOM_RET;	GET00950
57	1	1	D_IND = D_IND + 1;	GET00960
58	1	1	C2(R_IND,D_IND) =DOM_RET.VALUE;	GET00970
59	1	1	FREE DOM_RET;	GET00980
60	1	1	END;	GET00990
61	1	0	NDOM(R_IND) = D_IND;	GET01000
62	1	0	PUT SKIP EDIT ('VIEW LOADED.') (X(10),A);	GET01010
63	1	0	END GETVIEW;	GET01020
				GET01030


```

%INCLUDE SHREL;*****SHR00010
/*****DOC01160
*DOC01170
*MODULE DESCRIPTION*DOC01180
*****/DOC01190
1 0 SHREL: PROCEDURE(TNAME /* CHAR(8) */);DOC01200
/*****DOC01210
*****PURPOSEDOC01220
*****DISPLAY ALL DOMAINS WITH THEIR ATTRIBUTES OF A GIVE DOC01230
*****RELATION.DOC01240
*****DOC01250
*****METHOD:DOC01260
*****NOT SIGNIFICANTDOC01270
*****DOC01280
*****DOC01290
*****DOC01300
*****INPUT PARAMETERS:DOC01310
*****1) TNAME - NAME OF RELATIONDOC01320
*****DOC01330
*****DOC01340
*****OUTPUT PARAMETERS:DOC01350
*****NONEDOC01360
*****DOC01370
*****DOC01380
*****CALLS PROCEDURES:DOC01390
*****DOC01400
*****FETCHTDOC01410
*****/DOC01420
*****SHR00010
*****SHR00020
/* FETCHT TABLE */SHR00030
DCL 1 RET_ARG,SHR00040
2 NUMN BIT(8) INIT('00000001'B),SHR00050
2 NSET(5) BIT(64) INIT(UNSPEC('DTABLE ')),SHR00060
2 ATTR(20),SHR00070
3 N_INDEX BIT(8) INIT((2)('00000001'B)),SHR00080
3 NAME BIT(64) INIT(UNSPEC('DNAME '),SHR00090
UNSPEC('DATTR ')),SHR00100
3 RET_INFO,SHR00110
4 FETCH BIT(8) INIT('00000000'B,'10000000'B),SHR00120
4 SAME BIT(8) INIT((2)('00000000'B)),SHR00130
4 FVALUE BIT(160);SHR00140
SHR00150
/* FETCHT OUTPUT */SHR00160
%INCLUDE DOMRET;*****SHR00170
/* FETCHT OUTPUT */DEC00020

```

```

3 1 0      DCL 1 DOM_RET CONTROLLED EXTERNAL, /* STACK OF DATA VALUES */ DEC00030
            2 D_ID FIXED BIN(15), /* NSET AND ATTRIBUTE ID */ DEC00040
            2 VALUE BIT(320); /* DATA VALUE */ DEC00050
            ..... SHR00170
            SHR00180
            /* RELATION TABLES */ SHR00190
4 1 0      DCL 1 T1_ARG(20) EXTERNAL, SHR00200
            2 N1 BIT(8), SHR00210
            2 C1(5) BIT(64), SHR00220
            2 T2(20), SHR00230
            3 N2 BIT(8), SHR00240
            3 C2 BIT(64), SHR00250
            3 T3, SHR00260
            4 N3 BIT(8), SHR00270
            4 N4 BIT(8), SHR00280
            4 C3 BIT(160), SHR00290
            R_IND FIXED BIN(15) EXTERNAL, SHR00300
            REL(20) CHAR(8) EXTERNAL, SHR00310
            NDOM(0:20) FIXED BIN(15) EXTERNAL; SHR00320
            SHR00330
            /* DOMAIN ATTRIBUTES MASK */ SHR00340
5 1 0      DCL 1 RECORD, SHR00350
            2 TTYPE BIT(8), SHR00360
            2 TLEN BIT(16), SHR00370
            2 TMIN BIT(16), SHR00380
            2 TMAX BIT(16), SHR00390
            DATTR BIT(56) DEFINED RECORD, SHR00400
            TYPE CHAR(1), SHR00410
            (BLEN,BMIN,BMAX) FIXED BIN(16); SHR00420
            SHR00430
            SHR00440
            /* SUBROUTINES */ SHR00450
6 1 0      DCL FETCH ENTRY(1, 2 BIT(8), 2 (5) BIT(64), 2 (20), 3 BIT(8), SHR00460
            3 BIT(64), 3, 4 BIT(8), 4 BIT(8), 4 BIT(160)); SHR00470
            SHR00480
            /* MISCELLANEOUS */ SHR00490
7 1 0      DCL TNAME CHAR(8), SHR00500
            (I,J) FIXED BIN(15); SHR00510
            SHR00520
            /* VALIDATE RELATION NAME */ SHR00530
8 1 0      DO I = 1 TO R_IND; SHR00540
9 1 1      IF TNAME = REL(I) THEN LEAVE; SHR00550
10 1 1      END; SHR00560
11 1 0      IF I = R_IND + 1 SHR00570
            THEN DO; SHR00580
            PUT SKIP LIST ('RELATION NOT IN VIEW. '); SHR00590
            RETURN; SHR00600
            END; SHR00610
            SHR00620
            /* PRINT OUT HEADER */

```

L;

15	1	0	PUT SKIP EDIT ('DOMAIN','TYPE','LEN','MIN','MAX')	SHR00630
			(X(5),A,X(14),A,X(8),A,X(8),A,X(8),A);	SHR00640
16	1	0	PUT SKIP EDIT ('=====','=====','=====','=====','=====')	SHR00650
			(X(5),A,X(14),A,X(8),A,X(8),A,X(8),A);	SHR00660
17	1	0	FVALUE(2) = '01010101'B;	SHR00670
				SHR00680
			/* PRINT OUT DOMAINS AND ATTRIBUTES */	SHR00690
18	1	0	DO J = 1 TO NDOM(I);	SHR00700
19	1	1	IF NJ(I,J) = '10000000'B & N4(I,J) = '00000000'B	SHR00710
			THE I DO;	SHR00720
20	1	2	FVALUE(1) = C2(I,J);	SHR00730
21	1	2	CALL FETCHT(RET_ARG);	SHR00740
22	1	2	DATTR = VALUE;	SHR00750
23	1	2	UNSPEC(TYPE) = TTYPE; BLEN = TLEN;	SHR00760
25	1	2	BMIN = TMIN; BMAX = TMAX;	SHR00770
27	1	2	FREE DOM_RET;	SHR00780
28	1	2	UNSPEC(TNAME) = C2(I,J);	SHR00790
29	1	2	IF TYPE = 'C'	SHR00800
			THEN PUT SKIP EDIT (TNAME,'CHAR',BLEN,'--','--')	SHR00810
			(X(5),A,X(12),A,X(9),F(2),X(9),A,X(9),A);	SHR00820
30	1	2	ELSE PUT SKIP EDIT (TNAME,'NUM',BLEN,BMIN,BMAX)	SHR00830
			(X(5),A,X(13),A,X(9),F(2),X(9),F(2),X(9),F(2));	SHR00840
31	1	2	END;	SHR00850
32	1	1	END;	SHR00860
33	1	0	END SHREL;	SHR00870
				SHR00880

	%INCLUDE LOAD;*****	LOA00010
	/*****	DQC04520
	*	DQC04530
	MODULE DESCRIPTION	DQC04540
	*****	DQC04550
1	0 LOAD: PROCEDURE(TNAME /* CHAR(8) */);	DQC04560
	/*****	DQC04570
	PURPOSE:	DQC04580
	TO INSERT TUPLES OF DATA INTO A GIVEN RELATION.	DQC04590
	*****	DQC04600
	METHOD:	DQC04610
	NOT SIGNIFICANT	DQC04620
	*****	DQC04630
	INPUT PARAMETERS:	DQC04640
	1) TNAME - NAME OF RELATION	DQC04650
	*****	DQC04660
	OUTPUT PARAMETERS:	DQC04670
	NONE	DQC04680
	*****	DQC04690
	CALLS PROCEDURES:	DQC04700
	FETCHT, INSERTN, LEX2	DQC04710
	*****	DQC04720
	/*****	DQC04730
	/* INSERTN TABLE */	DQC04740
2	1 0 DCL 1 INSERT_ARG,	DQC04750
	2 NNAME BIT(64),	DQC04760
	2 NATTR BIT(8),	DQC04770
	2 ARG(20),	LOA00010
	3 INAME BIT(64),	LOA00020
	3 IVALUE BIT(320);	LOA00030
	/* FETCHT TABLE */	LOA00040
3	1 0 DCL 1 RET_ARG,	LOA00050
	2 NUMN BIT(8),	LOA00060
	2 NSET(5) BIT(64),	LOA00070
	2 ATTR(20),	LOA00080
	3 N_INDEX BIT(8),	LOA00090
	3 NAME BIT(64),	LOA00100
	3 RET_INFO,	LOA00110
	4 FETCH BIT(8),	LOA00120
		LOA00130
		LOA00140
		LOA00150
		LOA00160
		LOA00170
		LOA00180
		LOA00190

);

```

                                4 SAME BIT(8),
                                4 FVALUE BIT(160);
                                4 1 0
/* FETCHT OUTPUT */
DCL 1 DOM_RET CONTROLLED EXTERNAL,
    2 D_ID FIXED BIN(15),
    2 VALUE BIT(320);

/* SUBROUTINES */
%INCLUDE EINSERT;*****
/* INSERT NSET MODULE */
5 1 0 DCL INSERTN ENTRY(1, 2 BIT(64), 2 BIT(8), 2 (20),
    3 BIT(64), 3 BIT(320));
*****
%INCLUDE EFETCHT;*****
/* FETCH NSET MODULE */
6 1 0 DCL FETCHT ENTRY(1, 2 BIT(8), 2 (5) BIT(64), 2 (20), 3 BIT(8),
    3 BIT(64), 3, 4 BIT(8), 4 BIT(8), 4 BIT(160));
*****
%INCLUDE ELEX2;*****
/* LEXICAL ANALYZER */
7 1 0 DCL LEX2 ENTRY(FIXED BIN(15), (*) CHAR(40), (*) FIXED BIN(15));
*****
/* DOMAIN ATTRIBUTES TABLE */
8 1 0 DCL 1 RECORD,
    2 TTYPE BIT(8),
    2 TLEN BIT(16),
    2 TMIN BIT(16),
    2 TMAX BIT(16),
    DAT1R BIT(56) DEFINED RECORD,
    TYPE(10) CHAR(1),
    BLEN(10) FIXED BIN(16),
    BMIN(10) FIXED BIN(16),
    BMAX(10) FIXED BIN(16);

/* CURRENT VIEW ID */
9 1 0 DCL C_ID CHAR(8) EXTERNAL;

/* MISCELLANEOUS */
10 1 0 DCL TNAME CHAR(8),
    DUM(20) CHAR(8),
    DATA(20) CHAR(40) INIT( (20)((40)' ')),
    L(20) FIXED BIN(15),
    TEMP(20) CHAR(40) INIT( (20)((40)' ')),
    TL(20) FIXED BIN(15),
    STR CHAR(80) VARYING,
    I FIXED BIN(8),
    (J,N,N2) FIXED BIN(15),

```

LOA00200
 LOA00210
 LOA00220
 LOA00230
 LOA00240
 LOA00250
 LOA00260
 LOA00270
 LOA00280
 LOA00290
 DEC00070
 DEC00080
 DEC00090
 LOA00290
 LOA00300
 DEC00110
 DEC00120
 DEC00130
 LOA00300
 LOA00310
 DEC00150
 DEC00160
 LOA00310
 LOA00320
 LOA00330
 LOA00340
 LOA00350
 LOA00360
 LOA00370
 LOA00380
 LOA00390
 LOA00400
 LOA00410
 LOA00420
 LOA00430
 LOA00440
 LOA00450
 LOA00460
 LOA00470
 LOA00480
 LOA00490
 LOA00500
 LOA00510
 LOA00520
 LOA00530
 LOA00540
 LOA00550
 LOA00560
 LOA00570

		TRUE BIT(1) INIT('1'B);	LOA00580
		/* START LOAD */	LOA00590
		/* GET RELATION NAME */	LOA00600
11	1	0	LOA00610
		IF TNAME = (8) ' '	LOA00620
		THEN DO;	LOA00630
12	1	1	LOA00640
		PUT SKIP LIST ('RELATION NAME:');	LOA00650
13	1	1	LOA00660
		GET EDIT (TNAME) (A(8));	LOA00670
14	1	1	LOA00680
		IF TNAME = (8) ' ' THEN RETURN;	LOA00690
15	1	1	LOA00700
		END;	LOA00710
		/* VALIDATE RELATION NAME */	LOA00720
16	1	0	LOA00730
		NUMN = '00000001'B; NSET(1) = UNSPEC('VTABLE ');	LOA00740
18	1	0	LOA00750
		N_INDEX(1),N_INDEX(2) = '00000001'B;	LOA00760
19	1	0	LOA00770
		NAME(1) = UNSPEC('ID '); NAME(2) = UNSPEC('RNAME ');	LOA00780
21	1	0	LOA00790
		FETCH(1) = '00000000'B; FETCH(2) = '10000000'B;	LOA00800
23	1	0	LOA00810
		SAME(1),SAME(2) = '00000000'B;	LOA00820
24	1	0	LOA00830
		FVALUE(1) = UNSPEC(C_ID); FVALUE(2) = UNSPEC(TNAME);	LOA00840
26	1	0	LOA00850
		CALL FETCHT(RET_ARG);	LOA00860
27	1	0	LOA00870
		IF ALLOCATION(DOM_RET) = 0	LOA00880
		THEN DO;	LOA00890
28	1	1	LOA00900
		PUT SKIP LIST ('RELATION NOT IN VIEW.');	LOA00910
29	1	1	LOA00920
		RETURN;	LOA00930
30	1	1	LOA00940
		END;	LOA00950
31	1	0	LOA00960
		FREE DOM_RET;	LOA00970
		/* GET DOMAINS AND ATTRIBUTES OF RELATION */	LOA00980
32	1	0	LOA00990
		NUMN = '00000010'B; NSET(1) = UNSPEC('RTABLE ');	LOA01000
34	1	0	LOA01010
		NSET(2) = UNSPEC('DTABLE ');	LOA01020
35	1	0	LOA01030
		N_INDEX(1),N_INDEX(2) = '00000001'B;	LOA01040
36	1	0	LOA01050
		N_INDEX(3),N_INDEX(4) = '00000010'B;	LOA01060
37	1	0	
		NAME(1) = UNSPEC('RNAME '); NAME(4) = UNSPEC('DATTR ');	
39	1	0	
		NAME(2),NAME(3) = UNSPEC('DNAME ');	
40	1	0	
		FETCH(1) = '00000000'B;	
41	1	0	
		FETCH(2),FETCH(3),FETCH(4) = '10000000'B;	
42	1	0	
		SAME(1),SAME(2),SAME(4) = '00000000'B; SAME(3) = '00010010'B;	
44	1	0	
		FVALUE(1) = UNSPEC(TNAME);	
45	1	0	
		FVALUE(2),FVALUE(3),FVALUE(4) = '01010101'B;	
46	1	0	
		CALL FETCHT(RET_ARG);	
47	1	0	
		DO I = 1 TO ALLOCATION(DOM_RET)/2;	
48	1	1	
		UNSPEC(DOM(I)) = VALUE;	
49	1	1	
		FREE DOM_RET;	
50	1	1	
		DATTR = VALUE;	
51	1	1	
		UNSPEC(TYPE(I)) = TTYPE; BLEN(I) = TLEN;	
53	1	1	
		BMIN(I) = TMIN; BMAX(I) = TMAX;	
55	1	1	
		FREE DOM_RET;	
56	1	1	
		END;	
57	1	0	
		I = I - 1	

58	1	0	/* PRINT OUT DOMAINS */	LOAO1070
59	1	0	STR = ';;';	LOAO1080
60	1	1	DO J = 1 TO I;	LOAO1090
61	1	1	STR = STR DOM(J) ';;';	LOAO1100
62	1	0	END;	LOAO1110
			IF I > 8	LOAO1120
			THEN DO;	LOAO1130
63	1	1	PUT SKIP LIST (SUBSTR(STR,1,73));	LOAO1140
64	1	1	PUT SKIP LIST (SUBSTR(STR,73));	LOAO1150
65	1	1	END;	LOAO1160
66	1	0	ELSE PUT SKIP LIST (STR);	LOAO1170
				LOAO1180
				LOAO1190
67	1	0	/* INITIALIZE INSERTN VARIABLES */	LOAO1200
69	1	0	NNAME = UNSPEC(TNAME); NATR = I;	LOAO1210
			INAME = UNSPEC(DOM);	LOAO1220
				LOAO1230
			/* GET DATA */	LOAO1240
70	1	0	DO WHILE(TRUE);	LOAO1250
71	1	1	PUT SKIP LIST ('L:');	LOAO1260
72	1	1	CALL LEX2(N,DATA,L);	LOAO1270
73	1	1	IF N = 0 THEN RETURN;	LOAO1280
74	1	1	IF N ^= I	LOAO1290
			THEN PUT SKIP LIST ('INCORRECT NUMBER OF DATA ITEMS.');	LOAO1300
75	1	1	ELSE DO;	LOAO1310
76	1	2	DO J = 1 TO I;	LOAO1320
77	1	3	DO WHILE(L(J) > BLEN(J));	LOAO1330
78	1	4	PUT SKIP EDIT ('DATA FOR DOMAIN ',DOM(J),	LOAO1340
			' TOO LONG:') (A,A,A);	LOAO1350
79	1	4	CALL LEX2(N2,TEMP,TL);	LOAO1360
80	1	4	IF N2 = 0 THEN RETURN;	LOAO1370
81	1	4	DATA(J) = TEMP(1);	LOAO1380
82	1	4	L(J) = TL(1);	LOAO1390
83	1	4	END;	LOAO1400
84	1	3	IF TYPE(J) = 'N'	LOAO1410
			THEN DO;	LOAO1420
85	1	4	DO WHILE(VERIFY(DATA(J),'-0123456789 ') ^= 0);	LOAO1430
86	1	5	PUT SKIP EDIT ('DATA FOR DOMAIN ',DOM(J),	LOAO1440
			' MUST BE NUMERIC:')((3)A);	LOAO1450
87	1	5	CALL LEX2(N2,TEMP,TL);	LOAO1460
88	1	5	IF N2 = 0 THEN RETURN;	LOAO1470
89	1	5	DATA(J) = TEMP(1);	LOAO1480
90	1	5	END;	LOAO1490
91	1	4	DO WHILE(BIN(DATA(J)) < BMIN(J));	LOAO1500
92	1	5	PUT SKIP EDIT ('DATA FOR DOMAIN ',DOM(J),	LOAO1510
			' BELOW MIN:') (A,A,A);	LOAO1520
93	1	5	CALL LEX2(N2,TEMP,TL);	LOAO1530
94	1	5	IF N2 = 0 THEN RETURN;	LOAO1540
95	1	5	DATA(J) = TEMP(1);	LOAO1550

```

96 1 5
97 1 4
98 1 5

99 1 5
100 1 5
101 1 5
102 1 5
103 1 4
104 1 3

105 1 2
106 1 2
107 1 2
108 1 1
109 1 0

```

```

END;
DO WHILE(BIN(DATA(J)) > BMAX(J));
  PUT SKIP EDIT ('DATA FOR DOMAIN ',DOM(J),
    ' ABOVE MAX:') (A,A,A);
  CALL LEX2(N2,TEMP,TL);
  IF N2 = 0 THEN RETURN;
  DATA(J) = TEMP(1);
END;
END;

END;

/* FINALLY, CAN CALL INSERTN */
IVALUE = UNSPEC(DATA);
CALL INSERTN(INSERT_ARG);
END;

END;

END LOAD;

```

```

LOA01560
LOA01570
LOA01580
LOA01590
LOA01600
LOA01610
LOA01620
LOA01630
LOA01640
LOA01650
LOA01660
LOA01670
LOA01680
LOA01690
LOA01700
LOA01710
LOA01720
LOA01730

```



```

%INCLUDE SELECT;*****SEL00010
/***** DOC01720
* DOC01730
* DOC01740
* DOC01750
* DOC01760
* DOC01770
* DOC01780
* DOC01790
* DOC01800
* DOC01810
* DOC01820
* DOC01830
* DOC01840
* DOC01850
* DOC01860
* DOC01870
* DOC01880
* DOC01890
* DOC01900
* DOC01910
* DOC01920
* DOC01930
* DOC01940
* DOC01950
* DOC01960
* DOC01970
* DOC01980
* DOC01990
* SEL00010
* SEL00020
* SEL00030
* SEL00040
* SEL00050
* SEL00060
* SEL00070
* SEL00080
* SEL00090
* SEL00100
* SEL00110
* SEL00120
* SEL00130
* SEL00140
* SEL00150
* SEL00160
* SEL00170
* SEL00180

1 0 SELECT: PROCEDURE(TNAME /* (20) CHAR(8) */);
PURPOS:
    RESTRICT GIVEN DOMAINS OF A RELATION TO CERTAIN
    VALUES.
METHOD:
    REFER TO NJOIN1 FOR RESTRICT METHOD
INPUT PARAMETERS:
    1) TNAME - ALL THE TOKENS FOUND IN THE SELECT COMMAND
OUTPUT PARAMETERS:
    NONE
CALLS PROCEDURES:
    NONE

2 1 0 /* RELATION TABLES */
DCL 1 T1_ARG(20) EXTERNAL,
      2 N1 BIT(8),
      2 C1(5) BIT(64),
      2 T2(20),
      3 N2 BIT(8),
      3 C2 BIT(64),
      3 T3,
      4 N3 BIT(8),
      4 N4 BIT(8),
      4 C3 BIT(160),
      R_IND FIXED BIN(15) EXTERNAL,
      REL(20) CHAR(8) EXTERNAL,
      NDOM(0:20) FIXED BIN(15) EXTERNAL;

/* MISCELLANEOUS */

```

P

```

3 1 0      DCL TNAME(20) CHAR(8),                      SEL00190
              LVALUE(20) CHAR(40) EXTERNAL,              SEL00200
              ANS CHAR(1),                                SEL00210
              GIVING FIXED BIN(15) EXTERNAL,              SEL00220
              (1,J,J1) FIXED BIN(15),                    SEL00230
              FLAG BIT(1);                                SEL00240
                                                    SEL00250
/* START SUBROUTINE */                                  SEL00260
                                                    SEL00270
              /* VAL DATE RELATION NAME */               SEL00280
              DO I = 1 TO R_IND;                          SEL00290
2 1 1      IF TNAME(2) = REL(I) THEN LEAVE;              SEL00300
3 1 1      END;                                           SEL00310
4 1 1      IF I = R_IND + 1                               SEL00320
5 1 0      THEN DO;                                       SEL00330
6 1 1      PUT SKIP LIST ('RELATION NOT FOUND. ');       SEL00340
7 1 1      RETURN;                                        SEL00350
8 1 1      END;                                           SEL00360
                                                    SEL00370
              /* COPY OLD RELATION ONTO NEW RELATION */   SEL00380
11 1 0      T1_ARG(R_IND+1) = T1_ARG(I);                 SEL00390
                                                    SEL00400
              /* DO SELECTION */                          SEL00410
              DO J = 4 TO GIVING - 1;                     SEL00420
12 1 0      FLAG = '1'B;                                  SEL00430
13 1 1      DO J1 = 1 TO NDOM(I);                          SEL00440
14 1 1      IF UNSPEC(TNAME(J)) = C2(I,J1) & N3(I,J1) = '10000000'B SEL00450
15 1 2      THEN DO;                                       SEL00460
16 1 3      FLAG = '0'B;                                   SEL00470
17 1 3      IF C3(I,J1) = '01010101'B                   SEL00480
18 1 3      THEN C3(R_IND+1,J1) = UNSPEC(LVALUE(J));     SEL00490
19 1 4      ELSE DO;                                       SEL00500
20 1 4      PUT SKIP EDIT('DOMAIN ',TNAME(J),            SEL00510
21 1 4      ' ALREADY SELECTED ON.')((3)A);              SEL00520
22 1 5      ANS = ' ';                                     SEL00530
23 1 5      DO WHILE(ANS ^= 'Y');                          SEL00540
24 1 5      PUT SKIP LIST ('IGNORE(Y OR N):');           SEL00550
25 1 6      GET EDIT (ANS) (A(1));                         SEL00560
26 1 6      IF ANS = ' ' ; ANS = 'N'                      SEL00570
27 1 6      THEN DO;                                       SEL00580
28 1 5      PUT SKIP LIST                                SEL00590
29 1 4      ('NO NEW RELATION CREATED. ');               SEL00600
30 1 4      RETURN;                                        SEL00610
31 1 3      END;                                           SEL00620
32 1 2      END;                                           SEL00630
              GO TO NEXT;                                  SEL00640
              END;                                           SEL00650
              END;                                           SEL00660
              ENL;                                           SEL00670

```

33	1	1	NEXT:	IF FLAG	SEL00680
				THEN DO;	SEL00690
34	1	2		PUT SKIP EDIT ('DOMAIN ',TNAME(J),' NOT IN RELATION.')	SEL00700
				(A,A,A);	SEL00710
35	1	2		ANS = ' ';	SEL00720
36	1	2		DO WHILE(ANS ^= 'Y');	SEL00730
37	1	3		PUT SKIP LIST ('IGNORE(Y OR N):');	SEL00740
38	1	3		GET EDIT (ANS) (A(1));	SEL00750
39	1	3		IF ANS = ' ' ANS = 'N' THEN RETURN;	SEL00760
40	1	3		END;	SEL00770
41	1	2		END;	SEL00780
42	1	1	END;		SEL00790
				/* CREATE NEW RELATION */	SEL00800
43	1	0		DO J = 1 TO R_IND;	SEL00810
44	1	1		IF TNAME(GIVING+1) = REL(J) THEN LEAVE;	SEL00820
45	1	1	END;		SEL00830
46	1	0		IF J = R_IND + 1	SEL00840
			THEN DO;		SEL00850
47	1	1		R_IND = R_IND + 1;	SEL00860
48	1	1		NDOM(R_IND) = NDOM(I);	SEL00870
49	1	1		REL(R_IND) = TNAME(GIVING+1);	SEL00880
50	1	1		END;	SEL00890
51	1	0		ELSE T1_ARG(J) = T1_ARG(R_IND+1);	SEL00900
52	1	0	END SELECT;		SEL00910
					SEL00920
					SEL00930


```

ANS CHAR(1),
GIVING FIXED BIN(15) EXTERNAL,
(I,J,J1) FIXED BIN(15),
FLAG BIT(1);

/* START SUBROUTINE */

/* VALIDATE RELATION NAME */
DO I = 1 TO R_IND;
  IF TNAME(2) = REL(I) THEN LEAVE;
END;
IF I = R_IND + 1
THEN DO;
  PUT SKIP LIST ('RELATION NOT FOUND. ');
  RETURN;
END;

/* COPY OLD RELATION ONTO NEW RELATION */
T1_ARG(R_IND+1) = T1_ARG(I);
N3(R_IND+1,*) = '00000000'B;

/* DO PROJECTION */
DO J = 4 TO GIVING - 1;
  FLAG = '1'B;
  DO J1 = 1 TO NDOM(I);
    IF UNSPEC(TNAME(J)) = C2(I,J1) & N3(I,J1) = '10000000'B
    THEN DO;
      N3(R_IND+1,J1) = '10000000'B;
      FLAG = '0'B;
    END;
  END;
  IF FLAG
  THEN DO;
    PUT SKIP EDIT ('DOMAIN ',TNAME(J),' NOT IN RELATION.')
      (A,A,A);
    ANS = ' ';
    DO WHILE(ANS ^= 'Y');
      PUT SKIP LIST ('IGNORE(Y OR N): ');
      GET EDIT (ANS) (A(1));
      IF ANS = ' ' | ANS = 'N' THEN RETURN;
    END;
  END;

END;

/* CREATE NEW RELATION */
DO J = 1 TO R_IND;
  IF TNAME(GIVING+1) = REL(J) THEN LEAVE;
END;
IF J = P_IND + 1

```

PRO00200
 PRO00210
 PRO00220
 PRO00230
 PRO00240
 PRO00250
 PRO00260
 PRO00270
 PRO00280
 PRO00290
 PRO00300
 PRO00310
 PRO00320
 PRO00330
 PRO00340
 PRO00350
 PRO00360
 PRO00370
 PRO00380
 PRO00390
 PRO00400
 PRO00410
 PRO00420
 PRO00430
 PRO00440
 PRO00450
 PRO00460
 PRO00470
 PRO00480
 PRO00490
 PRO00500
 PRO00510
 PRO00520
 PRO00530
 PRO00540
 PRO00550
 PRO00560
 PRO00570
 PRO00580
 PRO00590
 PRO00600
 PRO00610
 PRO00620
 PRO00630
 PRO00640
 PRO00650
 PRO00660
 PRO00670
 PRO00680

35	1	1	THEN DO;		PRO00690
36	1	1	R_IND = R_IND + 1;		PRO00700
37	1	1	NDOM(R_IND) = NDOM(I);		PRO00710
38	1	1	REL(R_IND) = TNAME(GIVING+1);		PRO00720
39	1	0	END;		PRO00730
			ELSE T1_ARG(J) = T1_ARG(R_IND+1);		PRO00740
40	1	0	END PROJECT;		PRO00750
					PRO00760

3 1 0

```
DCL TNAME(20) CHAR(8),
      TEMP BIT(64),
      DIFF BIT(8),
      TRUE BIT(1) INIT('1'B),
      GIVING FIXED BIN(15) EXTERNAL,
      (I,J,K,L,P1,P2) FIXED BIN(15),
      (A,B,C) FIXED BIN(8);
```

/* START SUBROUTINE */

```
4 1 0      /* VALIDATE RELATION 1 */
5 1 1      DO WHILE(TRUE);
6 1 2      DO I = 1 TO R_IND;
7 1 2      IF TNAME(2) = REL(I) THEN LEAVE;
8 1 1      END;
9 1 1      IF I = R_IND + 1 THEN LEAVE;
10 1 1     PUT SKIP LIST ('RELATION 1 UNDEFINED:');
11 1 1     GET EDIT (TNAME(2)) (A(8));
12 1 1     IF TNAME(2) = (8) THEN RETURN;
13 1 0     END;
          P1 = I;
```

```
14 1 0     /* VALIDATE RELATION 2 */
15 1 1     DO WHILE(TRUE);
16 1 2     DO I = 1 TO R_IND;
17 1 2     IF TNAME(4) = REL(I) THEN LEAVE;
18 1 1     END;
19 1 1     IF I = R_IND + 1 THEN LEAVE;
20 1 1     PUT SKIP LIST ('RELATION 2 UNDEFINED:');
21 1 1     GET EDIT (TNAME(4)) (A(8));
22 1 1     IF TNAME(4) = (8) THEN RETURN;
23 1 0     END;
          P2 = I;
```

```
24 1 0     /* COPY RELATION 1 INTO NEW RELATION */
          T1_ARG(R_IND+1) = T1_ARG(P1);
```

```
25 1 0     /* CORRECT NUMN */
28 1 0     A = N1(P1); B = N1(P2); C = A + B;
          N1(R_IND+1) = C;
```

```
29 1 0     /* ADD NSET NAMES */
30 1 1     DO I = 1 TO B;
31 1 1     C1(R_IND+1,A+I) = C1(P2,I);
          END;
```

```
32 1 0     /* ADD N_INDEXES, ATTRIBUTE NAMES, AND RET_INFO */
33 1 1     DO I = 1 TO NDOM(P2);
          B = N2(P2,I); C = A + B;
```

JOI00190
JOI00200
JOI00210
JOI00220
JOI00230
JOI00240
JOI00250
JOI00260
JOI00270
JOI00280
JOI00290
JOI00300
JOI00310
JOI00320
JOI00330
JOI00340
JOI00350
JOI00360
JOI00370
JOI00380
JOI00390
JOI00400
JOI00410
JOI00420
JOI00430
JOI00440
JOI00450
JOI00460
JOI00470
JOI00480
JOI00490
JOI00500
JOI00510
JOI00520
JOI00530
JOI00540
JOI00550
JOI00560
JOI00570
JOI00580
JOI00590
JOI00600
JOI00610
JOI00620
JOI00630
JOI00640
JOI00650
JOI00660
JOI00670

35	1	1	N2(R_IND+1,I+NDOM(P1)) = C;	J0100680
36	1	1	C2(R_IND+1,I+NDOM(P1)) = C2(P2,I);	J0100690
37	1	1	T3(R_IND+1,I+NDOM(P1)) = T3(P2,I);	J0100700
				J0100710
			/* FIX UP SAMES */	J0100720
38	1	1	IF N4(P2,I) ^= '00000000'B	J0100730
			THEN DO;	J0100740
39	1	2	B = N4(P2,I);	J0100750
40	1	2	C = 16*A + B;	J0100760
41	1	2	N4(R_IND+1,I+NDOM(P1)) = C;	J0100770
42	1	2	END;	J0100780
43	1	1	END;	J0100790
				J0100800
			/* JOIN THE 2 RELATIONS */	J0100810
44	1	0	DO I = 6 TO GIVING - 1;	J0100820
45	1	1	TEMP = UNSPEC(TNAME(I));	J0100830
46	1	1	DO J = NDOM(P1)+1 TO NDOM(P1)+NDOM(P2);	J0100840
47	1	2	IF TEMP = C2(R_IND+1,J) & N3(R_IND+1,J) = '10000000'B	J0100850
			THEN DO;	J0100860
48	1	3	BUFF = '00000000'B;	J0100870
49	1	3	DO K = 1 TO NDOM(P1);	J0100880
50	1	4	IF N2(R_IND+1,K) = BUFF	J0100890
			THEN L = L + 1;	J0100900
51	1	4	ELSE DO;	J0100910
52	1	5	L = 1;	J0100920
53	1	5	BUFF = N2(R_IND+1,K);	J0100930
54	1	5	END;	J0100940
55	1	4	IF TEMP = C2(R_IND+1,K) &	J0100950
			N3(R_IND+1,K) = '10000000'B	J0100960
			THEN DO;	J0100970
56	1	5	B = N2(R_IND+1,K);	J0100980
57	1	5	C = 16*B + L;	J0100990
58	1	5	N4(R_IND+1,J) = C;	J0101000
59	1	5	GO TO NEXT;	J0101010
60	1	5	END;	J0101020
61	1	4	END;	J0101030
62	1	3	PUT SKIP EDIT ('DOMAIN ',TNAME(I),	J0101040
			' NOT IN RELATION 1.') (A,A,A);	J0101050
63	1	3	RETURN;	J0101060
64	1	3	END;	J0101070
65	1	2	END;	J0101080
66	1	1	PUT SKIP EDIT ('DOMAIN ',TNAME(I),' NOT IN RELATION 2.') (A,A,A);	J0101090
			RETURN;	J0101100
67	1	1	END;	J0101110
68	1	1	NEXT: END;	J0101120
				J0101130
			/* CREATE NEW RELATION */	J0101140
69	1	0	DO I = 1 TO R_IND;	J0101150
70	1	1	IF TNAME(GIVING+1) = REL(I) THEN LEAVE;	J0101160

71	1	1	END;		JOI01170
72	1	0	IF I = R_IND+1		JOI01180
			THEN DO;		JOI01190
73	1	1	R_IND = R_IND + 1;		JOI01200
74	1	1	REL(R_IND) = TNAME(GIVING+1);		JOI01210
75	1	1	NDOM(R_IND) = NDOM(P1) + NDOM(P2);		JOI01220
76	1	1	END;		JOI01230
77	1	0	ELSE DO;		JOI01240
78	1	1	T1_ARG(I) = T1_ARG(R_IND+1);		JOI01250
79	1	1	NDOM(I) = NDOM(P1) + NDOM(P2);		JOI01260
80	1	1	END;		JOI01270
81	1	0	END JOIN;		JOI01280
					JOI01290

```

XINCLUDE PRINT;*****PRI00010
/*****DOC04250
*DOC04260
*MODULE DESCRIPTIONDOC04270
*****DOC04280
1 0 PRINT: PROCEDURE(TNAME /* CHAR(8) */);DOC04290
/*****DOC04300
*****PURPOSE:DOC04310
*****PRINT ALL THE TUPLES OF A GIVEN RELATION.DOC04320
*****DOC04330
*****DOC04340
*****METHOD:DOC04350
*****NOT SIGNIFICANTDOC04360
*****DOC04370
*****DOC04380
*****INPUT PARAMETERS:DOC04390
*****1) TNAME - NAME OF RELATIONDOC04400
*****DOC04410
*****DOC04420
*****OUTPUT PARAMETERS:DOC04430
*****NONEDOC04440
*****DOC04450
*****DOC04460
*****CALLS PROCEDURES:DOC04470
*****FETCHTDOC04480
*****DOC04490
*****DOC04500
*****DOC04500
*****PRI00010
*****PRI00020
*****PRI00030
/* FETCHT TABLE */PRI00040
XINCLUDE RETARG;*****PRI00040
2 1 0 DCL 1 RET_ARG, /* USED TO RETRIEVE NSETS */DCL00090
2 NUMN BIT(8), /* NUMBER OF NSETS */DCL00100
2 NSET(5) BIT(64), /* NAMES OF NSETS TO BE FETCHED */DCL00110
2 ARGS(20), /* INFO FOR EACH ATTRIBUTE */DCL00120
3 N_INDEX BIT(8), /* WHICH NSET IS THIS IN */DCL00130
3 NAME BIT(64), /* NAME OF ATTRIBUTE */DCL00140
3 RET_INFO, /* RETRIEVE INFORMATION */DCL00150
(4 FETCH, /* IS IT TO BE FETCHED */DCL00160
4 SAME ) BIT(8), /* SAME AS PREVIOUSLYDCL00170
DEFINED DOMAIN */DCL00180
4 VALUE BIT(160); /*VALUE TO SEARCH ON ORDCL00190
NONE */DCL00200
*****PRI00040
/* FETCHT OUTPUT */PRI00050
XINCLUDE DOMRET *****PRI00060

```

```

/* FETCH OUTPUT */
3 1 0 DCL 1 DOM_RET CONTROLLED EXTERNAL, /* STACK OF DATA VALUES */
      2 D_ID FIXED BIN(15), /* NSET AND ATTRIBUTE ID */
      2 VALUE BIT(320); /* DATA VALUE */
*****

/* RELATION TABLES */
4 1 0 DCL 1 T1_ARG(20) EXTERNAL,
      2 N1 BIT(8),
      2 C1(5) BIT(64),
      2 T2(20),
      3 N2 BIT(8),
      3 C2 BIT(64),
      3 T3,
      4 N3 BIT(8),
      4 N4 BIT(8),
      4 C3 BIT(160),
      R_IND FIXED BIN(15) EXTERNAL,
      REL(20) CHAR(8) EXTERNAL,
      NDOM(0:20) FIXED BIN(15) EXTERNAL;

/* DOMAIN ATTRIBUTES TABLE */
5 1 0 DCL 1 RECORD,
      2 MASK1 BIT(8),
      2 TLEN BIT(16),
      2 MASK2 BIT(32),
      DATTR BIT(56) DEFINED RECORD,
      LEN(20) FIXED BIN(16);

/* SUBROUTINES */
XINCLUDE EFETCH;*****
/* FETCH NSET MODULE */
6 1 0 DCL FECHT ENTRY(1, 2 BIT(8), 2 (5) BIT(64), 2 (20), 3 BIT(8),
      3 BIT(64), 3, 4 BIT(8), 4 BIT(8), 4 BIT(160));
*****

/* MISCELLANEOUS */
7 1 0 DCL (TNAME,TEMP) CHAR(8),
      STR CHAR(100) VARYING,
      DATA CHAR(40),
      COUNT(20) FIXED BIN(15),
      (I,J,J1,J2,K,L,M,SUM) FIXED BIN(15);

/* START SUBROUTINE */

/* GET RELATION NAME */
8 1 0 IF TNAME = (8) ' '
      THEN DO;
9 1 1 PUT SKIP LIST ('RELATION NAME:');

```

DEC00020
 DEC00030
 DEC00040
 DEC00050
 PRI00060
 PRI00070
 PRI00080
 PRI00090
 PRI00100
 PRI00110
 PRI00120
 PRI00130
 PRI00140
 PRI00150
 PRI00160
 PRI00170
 PRI00180
 PRI00190
 PRI00200
 PRI00210
 PRI00220
 PRI00230
 PRI00240
 PRI00250
 PRI00260
 PRI00270
 PRI00280
 PRI00290
 PRI00300
 PRI00310
 PRI00320
 DEC00110
 DEC00120
 DEC00130
 PRI00320
 PRI00330
 PRI00340
 PRI00350
 PRI00360
 PRI00370
 PRI00380
 PRI00390
 PRI00400
 PRI00410
 PRI00420
 PRI00430
 PRI00440
 PRI00450
 PRI00460

10	1	1	GET EDIT (TNAME) (A(B));	PRI00470
11	1	1	IF TNAME = (B)' ' THEN RETURN;	PRI00480
12	1	1	END;	PRI00490
				PRI00500
			/* VALIDATE RELATION NAME */	PRI00510
13	1	0	DO I = 1 TO R_IND;	PRI00520
14	1	1	IF TNAME = REL(I) THEN LEAVE;	PRI00530
15	1	1	END;	PRI00540
16	1	0	IF I = R_IND + 1	PRI00550
			THEN DO;	PRI00560
17	1	1	PU SKIP LIST ('RELATION NOT FOUND.');	PRI00570
18	1	1	RETURN;	PRI00580
19	1	1	END;	PRI00590
				PRI00600
			/* GET DOMAINS TO BE PRINTED */	PRI00610
20	1	0	NUMN = '00000001'B; NSET(1) = UNSPEC('DTABLE ');	PRI00620
22	1	0	N_INDEX(1),N_INDEX(2) = '00000001'B;	PRI00630
23	1	0	NAME(1) = UNSPEC('DNAME '); NAME(2) = UNSPEC('DATTR ');	PRI00640
25	1	0	FETCH(1) = '00000000'B; FETCH(2) = '10000000'B;	PRI00650
27	1	0	SAME(1),SAME(2) = '00000000'B; RET_ARG.VALUE(2) = '01010101'B;	PRI00660
29	1	0	STR = ' ';	PRI00670
30	1	0	K = 0;	PRI00680
31	1	0	DO J = 1 TO NDOM(1);	PRI00690
32	1	1	IF N3(I,J) = '10000000'B & N4(I,J) = '00000000'B	PRI00700
			THEN DO;	PRI00710
33	1	2	UNSPEC(TEMP) = C2(I,J);	PRI00720
34	1	2	STR = STR TEMP ' ';	PRI00730
35	1	2	RET_ARG.VALUE(1) = C2(I,J);	PRI00740
36	1	2	CALL FETCH(RET_ARG);	PRI00750
37	1	2	DATTR =DOM_RET.VALUE;	PRI00760
38	1	2	K = K + 1;	PRI00770
39	1	2	LEN(K) = TLEN;	PRI00780
40	1	2	FREE DOM_RET;	PRI00790
41	1	2	END;	PRI00800
42	1	1	END;	PRI00810
				PRI00820
			/* CREATE PRINTING FORMAT */	PRI00830
43	1	0	SUM,L = 1;	PRI00840
44	1	0	COUNT = 0;	PRI00850
45	1	0	DO J = 1 TO K;	PRI00860
46	1	1	SUM = SUM + LEN(J) + 1;	PRI00870
47	1	1	COUNT(L) = COUNT(L) + 1;	PRI00880
48	1	1	IF SUM > 80	PRI00390
			THEN DO;	PRI00900
49	1	2	COUNT(L) = COUNT(L) - 1;	PRI00910
50	1	2	L = L + 1;	PRI00920
51	1	2	COUNT(L) = 1;	PRI00930
52	1	2	SUM = LEN(J) + 2;	PRI00940
53	1	2	END;	PRI00950

54	1	1	END;	PRI00960
			/* PRINT OUT DOMAIN HEADER */	PRI00970
55	1	0	IF K > 8	PRI00980
			THEN DO;	PRI00990
56	1	1	PUT SKIP LIST (SUBSTR(STR,1,73));	PRI01000
57	1	1	PUT SKIP LIST (SUBSTR(STR,73));	PRI01010
58	1	1	END;	PRI01020
59	1	0	ELSE PUT SKIP LIST (STR);	PRI01030
			/* GET THE DATA */	PRI01040
60	1	0	RET_ARG = T1_ARG(1);	PRI01050
61	1	0	CALL FETCHT(RET_ARG);	PRI01060
			/* PRINT OUT THE DATA */	PRI01070
62	1	0	DO J = 1 TO ALLOCATION(DOM_RET)/K;	PRI01080
63	1	1	M = 0;	PRI01090
64	1	1	DO J1 = 1 TO L;	PRI01100
65	1	2	STR = '';	PRI01110
66	1	2	DO J2 = 1 TO COUNT(L);	PRI01120
67	1	3	M = M + 1;	PRI01130
68	1	3	UNSPEC(DATA) = DOM_RET.VALUE;	PRI01140
69	1	3	STR = STR SUBSTR(DATA,1,LEN(M)) ' ';	PRI01150
70	1	3	FREE DOM_RET;	PRI01160
71	1	3	END;	PRI01170
72	1	2	PUT SKIP LIST (STR);	PRI01180
73	1	2	END;	PRI01190
74	1	1	END;	PRI01200
			END PRINT;	PRI01210
75	1	0		PRI01220
				PRI01230
				PRI01240
				PRI01250

4;

/* START SUBROUTINE */

```

5 1 0      /* GET INPUT STRING */
6 1 0 LOOP: STR = '';
7 1 0      GET EDIT (LINE) (A(80));
8 1 0      POS = INDEX(LINE, '/');
          IF POS ^= 0
          THEN DO;
9 1 1        STR = STR || SUBSTR(LINE, 1, POS-1);
10 1 1       GO TO LOOP;
11 1 1       END;
12 1 0      STR = STR || LINE || ' ';

          /* LEX STARTS */
13 1 0      STR = TRANSLATE(STR, ' ', ',');
14 1 0      TOKEN = (8) ' '; LVALUE = (40) ' ';
16 1 0      I, AND, ON, GIVING = 0;

          /* EXTRACT TOKENS */
17 1 0      START = VERIFY(STR, ' ');
18 1 0      DO WHILE(START ^= 0);
19 1 1        STR = SUBSTR(STR, START);
20 1 1        END = INDEX(STR, ' ');
21 1 1        TEMP = SUBSTR(STR, 1, END-1);
22 1 1        I = I + 1;
23 1 1        EQ = INDEX(TEMP, '=');
24 1 1        IF EQ = 0
          THEN TOKEN(I) = TEMP;
          ELSE DO;
25 1 1          TOKEN(I) = SUBSTR(TEMP, 1, EQ-1);
26 1 2          IF SUBSTR(TEMP, EQ+1, 1) = ' '
27 1 2          THEN DO;
28 1 3            STR = SUBSTR(STR, EQ+2);
29 1 3            END = INDEX(STR, ' ');
30 1 3            LVALUE(I) = SUBSTR(STR, 1, END-1);
31 1 3            END;
32 1 2          ELSE LVALUE(I) = SUBSTR(TEMP, EQ+1);
33 1 2          END;
34 1 1          SELECT(TOKEN(I));
35 1 2            WHEN('AND ' ' ') AND = I;
36 1 2            WHEN('ON ' ' ') ON = I;
37 1 2            WHEN('GIVING ' ' ') GIVING = I;
38 1 2            OTHERWISE;
39 1 2          END;
40 1 1          STR = SUBSTR(STR, END+1);
41 1 1          START = VERIFY(STR, ' ');
42 1 1        END;
43 1 0      LAST = I.

```

LEX00180
 LEX00190
 LEX00200
 LEX00210
 LEX00220
 LEX00230
 LEX00240
 LEX00250
 LEX00260
 LEX00270
 LEX00280
 LEX00290
 LEX00300
 LEX00310
 LEX00320
 LEX00330
 LEX00340
 LEX00350
 LEX00360
 LEX00370
 LEX00380
 LEX00390
 LEX00400
 LEX00410
 LEX00420
 LEX00430
 LEX00440
 LEX00450
 LEX00460
 LEX00470
 LEX00480
 LEX00490
 LEX00500
 LEX00510
 LEX00520
 LEX00530
 LEX00540
 LEX00550
 LEX00560
 LEX00570
 LEX00580
 LEX00590
 LEX00600
 LEX00610
 LEX00620
 LEX00630
 LEX00640
 LEX00650
 LEX00660

xi

44 1 0

END LEX;

LEX00670

```

%INCLUDE LEX2:*****LEX00010
/*****
*
*          MODULE      DESCRIPTION
*          *****/
1  0  LEX2:  PROCEDURE(I /* FIXED BIN(15) */,
                DATA /* (10) CHAR(40) */,
                L /* (10) FIXED BIN(15) */);
/*****
*      PURPOSE:
*      LEXICAL ANALYZER FOR LOAD DATA LINES.
*      *****/
*      METHOD:
*      NOT SIGNIFICANT
*      *****/
*      INPUT PARAMETERS:
*      NONE
*      *****/
*      OUTPUT PARAMETERS:
*      1) I - NUMBER OF DATA ITEMS FOUND
*      2) DATA - DATA ARRAY
*      3) L - ARRAY OF LENGTHS OF EACH DATA ITEM
*      *****/
*      CALLS PROCEDURES:
*      NONE
*      *****/
/*****
2  1  0      /* LEXICAL ANALYZER VARIABLES */
                DCL DATA(10) CHAR(40),
                L(10) FIXED BIN(15),
                (I, LAST) FIXED BIN(15);
                LEX00010
                LEX00020
                LEX00030
                LEX00040
                LEX00050
                LEX00060
                LEX00070
                LEX00080
3  1  0      /* INPUT STRING VARIABLES */
                DCL STR CHAR(240) VARYING,
                LINE CHAR(80),
                POS FIXED BIN(15);
                LEX00090
                LEX00100
                LEX00110
                LEX00120
                LEX00130
                LEX00140
                LEX00150
                /* START SUBROUTINE */
                /* GET INPUT STRING */

```

2;

4 1 0
5 1 0
6 1 0
7 1 0
8 1 1
9 1 1
10 1 1
11 1 0
12 1 0
13 1 1
14 1 0
15 1 0
16 1 0
17 1 1
18 1 1
19 1 1
20 1 1
21 1 1
22 1 1
23 1 0

LOOP:

```
STR = '';
GET EDIT (LINE) (A(80));
POS = INDEX(LINE, '/');
IF POS ^= 0
THEN DO;
    STR = STR || SUBSTR(LINE, 1, POS-1);
    GO TO LOOP;
END;
STR = STR || LINE;

/* FIX UP LINE */
DO I = LENGTH(STR) TO 1 BY -1 WHILE(SUBSTR(STR, I, 1) = ' ');
END;
STR = SUBSTR(STR, 1, I) || '.,.';

/* LEX STARTS */
I = 0;

/* EXTRACT TOKENS */
DO WHILE(VERIFY(STR, ',') ^= 0);
    LAST = INDEX(STR, ',') - 1;
    I = I + 1;
    DATA(I) = SUBSTR(STR, 1, LAST);
    L(I) = LAST;
    STR = SUBSTR(STR, LAST+2);
END;
END LEX2;
```

LEX00160
LEX00170
LEX00180
LEX00190
LEX00200
LEX00210
LEX00220
LEX00230
LEX00240
LEX00250
LEX00260
LEX00270
LEX00280
LEX00290
LEX00300
LEX00310
LEX00320
LEX00330
LEX00340
LEX00350
LEX00360
LEX00370
LEX00380
LEX00390
LEX00400
LEX00410
LEX00420
LEX00430

```

%INCLUDE DELIM;*****DEL00010
/*****FOR00010
*
*          MODULE      DESCRIPTION          *
*          *****/
1  0  DELIM: PROCEDURE (TYPE /* BIT(1) */ )
                      RETURNS(BIT(1));
/*****FOR00050
***** PURPOSE:
*****THIS MODULE IS RESPONSIBLE FOR THE SYNTAX CHECKING OF
*****RELATIONAL OPERATIONS (SELECT, PROJECT AND JOIN ). IT IS
*****CALLED BY THE QUERY MODULE, AFTER LEX HAS PROCESSED THE
*****COMMAND LINE. DELIM MAKES USE OF THE POSITIONS OF DELIMITERS
*****TO DETERMINE IF A SYNTAX ERROR HAS OCCURRED. IF AN ERROR HAS
*****OCCURRED IT PRINTS A MESSAGE AND RETURNS A CODE TO QUERY.
*****FOR00060
***** METHOD:
*****   DELIM RELIES ON THE POSITION OF DELIMITERS TO INDICATE
*****   POSSIBLE PROBLEMS. WHEN LEX PROCESSES THE COMMAND LINE
*****   IT KEEPS TRACK OF THE POSITION OF DELIMITERS SUCH AS
*****   (AND,ON,GIVING) AND PUTS THEIR POSITIONS IN EXTERNAL
*****   VARIABLES OF THE SAME NAME.
*****FOR00070
*****
***** INPUT PARAMETERS:
*****   TYPE - FLAG TO INDICATE IF OPERATION IS A JOIN:
*****         '1'B - JOIN
*****         '0'B - NO
*****   AS MENTIONED ABOVE IT ALSO USES THE FOLLOWING EXTERNAL
*****   VARIABLES:(AND,ON,GIVING, LAST) FIXED BIN(15) EXT;
*****FOR00080
*****
***** OUTPUT PARAMETERS:
*****   RETURNS A '1','0' VALUE TO INDICATE IF ANY ERRORS WERE
*****   DISCOVERED. '0'B INDICATES NO ERRORS WERE DETECTED.
*****FOR00090
*****
***** CALLS PROCEDURES:
*****   NONE
*****FOR00100
*****
*****
*****FOR00110
*****
*****FOR00120
*****
*****FOR00130
*****
*****FOR00140
*****
*****FOR00150
*****
*****FOR00160
*****
*****FOR00170
*****
*****FOR00180
*****
*****FOR00190
*****
*****FOR00200
*****
*****FOR00210
*****
*****FOR00220
*****
*****FOR00230
*****
*****FOR00240
*****
*****FOR00250
*****
*****FOR00260
*****
*****FOR00270
*****
*****FOR00280
*****
*****FOR00290
*****
*****FOR00300
*****
*****FOR00310
*****
*****FOR00320
*****
*****FOR00330
*****
*****FOR00340
*****
*****FOR00350
*****
*****FOR00360
*****
*****FOR00370
*****
*****FOR00380
*****
*****FOR00390
*****
*****DEL00010
*****DEL00020
*****DEL00030
*****DEL00040
*****DEL00050
*****DEL00060

/* RELATIONAL OPERATOR INDEXES */
DCL (AND,ON,GIVING, LAST) FIXED BIN(15) EXTERNAL;

/* MISCELLANEOUS */

```

2 1 0

3	1	0	DCL TYPE BIT(1);	DEL00070
			/* START SUBROUTINE */	DEL00080
			/* CHECK MISSING DELIMITERS */	DEL00090
4	1	0	IF TYPE = '1'B	DEL00100
			THEN IF AND = 0	DEL00110
			THEN DO;	DEL00120
5	1	1	PUT SKIP LIST('AND KEYWORD MISSING.');	DEL00130
6	1	1	RETURN('1'B);	DEL00140
7	1	1	END;	DEL00150
8	1	0	IF ON = 0	DEL00160
			THEN DO;	DEL00170
9	1	1	PUT SKIP LIST('ON KEYWORD MISSING.');	DEL00180
10	1	1	RETURN('1'B);	DEL00190
11	1	1	END;	DEL00200
12	1	0	IF GIVING = 0	DEL00210
			THEN DO;	DEL00220
13	1	1	PUT SKIP LIST('GIVING KEYWORD MISSING.');	DEL00230
14	1	1	RETURN('1'B);	DEL00240
15	1	1	END;	DEL00250
			/* CHECK MISSING RELATION AND DOMAIN NAMES */	DEL00260
16	1	0	IF TYPE = '1'B	DEL00270
			THEN DO;	DEL00280
17	1	1	IF AND = 2	DEL00290
			THEN DO;	DEL00300
18	1	2	PUT SKIP LIST('FIRST RELATION NAME MISSING.');	DEL00310
19	1	2	RETURN('1'B);	DEL00320
20	1	2	END;	DEL00330
21	1	1	IF AND > 3	DEL00340
			THEN DO;	DEL00350
22	1	2	PUT SKIP LIST('TOO MANY NAMES FOR FIRST RELATION.');	DEL00360
23	1	2	RETURN('1'B);	DEL00370
24	1	2	END;	DEL00380
25	1	1	IF ON - AND < 2	DEL00390
			THEN DO;	DEL00400
26	1	2	PUT SKIP LIST('SECOND RELATION MISSING.');	DEL00410
27	1	2	RETURN('1'B);	DEL00420
28	1	2	END;	DEL00430
29	1	1	IF ON - AND > 2	DEL00440
			THEN DO;	DEL00450
30	1	2	PUT SKIP LIST('TOO MANY NAMES FOR SECOND RELATION.');	DEL00460
31	1	2	RETURN('1'B);	DEL00470
32	1	2	END;	DEL00480
33	1	1	END;	DEL00490
34	1	0	ELSE DO;	DEL00500
35	1	1	IF LN = 2	DEL00510
			THEN DO;	DEL00520
				DEL00530
				DEL00540
				DEL00550

36	1	2	PUT SKIP LIST('FIRST RELATION MISSING.');	DEL00560
37	1	2	RETURN('1'B);	DEL00570
38	1	2	END;	DEL00580
39	1	1	IF ON > 3	DEL00590
			THEN DO;	DEL00600
40	1	2	PUT SKIP LIST('TOO MANY NAMES FOR FIRST RELATION.');	DEL00610
41	1	2	RETURN('1'B);	DEL00620
42	1	2	END;	DEL00630
43	1	1	EN';	DEL00640
44	1	0	IF GIVING - ON < 2	DEL00650
			THEN DO;	DEL00660
45	1	1	PUT SKIP LIST('DOMAINS ARE MISSING.');	DEL00670
46	1	1	RETURN('1'B);	DEL00680
47	1	1	END;	DEL00690
48	1	0	IF LAST - GIVING < 1	DEL00700
			THEN DO;	DEL00710
49	1	1	PUT SKIP LIST('NEW RELATION NAME MISSING.');	DEL00720
50	1	1	RETURN('1'B);	DEL00730
51	1	1	END;	DEL00740
52	1	0	IF LAST - GIVING > 1	DEL00750
			THEN DO;	DEL00760
53	1	1	PUT SKIP LIST('TOO MANY NAMES FOR NEW RELATION.');	DEL00770
54	1	1	RETURN('1'B);	DEL00780
55	1	1	END;	DEL00790
			/* HERE, NO SYNTAX ERROR WAS FOUND */	DEL00800
56	1	0	RETURN('0'B);	DEL00810
57	1	0	END DELTM;	DEL00820
				DEL00830

```

%INCLUDE DEFINEN;*****DEF00010
/*****
*          MODULE DESCRIPTION          *
*****/DEF00010
1 0  DEFINEN:  PROCEDURE          DEF00020
                (DEF_ARG      /* 1,  DEF00030
                                DEF00040
                                2 BIT(64),  DEF00050
                                2 BIT(8),    DEF00060
                                2 (20),      DEF00070
                                3 BIT(64),   DEF00080
                                3 BIT(8) */ );  DEF00090
/*****DEF00100
*****PURPOSE:DEF00110
*****THIS PROCEDURE CREATES AN ENTRY IN THE NSET CATALOGUEDEF00120
*****CORRESPONDING TO THE NSET TO BE DEFINED. IN ADDITION ITDEF00130
*****IS RESPONSIBLE FOR DEFINING THE PSET WHICH ACTS AS THEDEF00140
*****ENTITY NODE OF THE NSET. IT IS ALSO RESPONSIBLE FORDEF00150
*****DEFINING THE BINARY ASSOCIATIONS (I.E. BSETS) BETWEENDEF00160
*****THE ENTITY NODE AND ITS ATTRIBUTES.DEF00170
*****DEF00180
*****DEF00190
*****DEF00200
*****METHOD:DEF00210
*****THE PROCEDURE BEGINS BY FILLING THE NCAT STRUCTURE WITHDEF00220
*****THE CORRESPONDING ARGUMENTS PASSED TO IT BY DEF_ARG. THISDEF00230
*****INCLUDES: THE NSET NAME, THE NUMBER OF ATTRIBUTES,DEF00240
*****THE NAME OF EACH ATTRIBUTE, AND WHETHER THE ATTRIBUTEDEF00250
*****HAS A 1 TO 1 RELATIONSHIP TO THE ENTITY NODE. IT THENDEF00260
*****CALLS DEFINER TO DEFINE A PSET WHICH CORRESPONDS TODEF00270
*****TO THE ENTITY NODE. THE ENTITY NODE IS GIVEN THE NAMEDEF00280
*****OF THE NSET, IS LINKED VIA HASHING, AND HAS A 32 BITDEF00290
*****KEY. THE NEXT STEP IS TO CREATE BSET DEFINITIONS BETWEENDEF00300
*****THE ENTITY NODE AND ITS ATTRIBUTES. FOR EACH ATTRIBUTEDEF00310
*****IT DETERMINES IF A 1 TO 1 OR A N TO 1 LINK IS TO BEDEF00320
*****DEFINED BETWEEN THE ENTITY NODE AND ATTRIBUTE. IT THENDEF00330
*****CALLS NAMEGEN TO CREATE A NAME FOR THE BSET, AND THENDEF00340
*****CALLS DEFINER TO CREATE THE BSET DEFINITION. IN ADDITION,DEF00350
*****IT CALLS DEFINER USING THE EQUIVALENT OPTION TO DEFINEDEF00360
*****THE RECIPROCAL LINK (I.E. BETWEEN THE ATTRIBUTE AND THEDEF00370
*****ENTITY NODE).ONCE THIS IS DONE IT UPDATES NCAT TO REFLECTDEF00380
*****THE BSETS CREATED. FINALLY IT USES THE INFORMATION INDEF00390
*****NCAT TO FILL INSERT_ARG, AND CALLS INSERTN, PASSING ITDEF00400
*****INSERT_ARG, TO CREATE AN ENTRY IN THE NSET CATALOGUE,DEF00410
*****CORRESPONDING TO THE NSET DEFINITION CREATED BY DEFINEN.DEF00420
*****DEF00430
*****DEF00440
*****DEF00450
*****INPUT PARAMETERS:

```

```

*****      1 DEF_ARG   CONTAINS INFO ON NSET TO BE DEFINED      DEF00460
*****      2 NNAME     NAME OF NSET TO BE CREATED              DEF00470
*****      2 NATTR     NUMBER OF ATTRIBUTES                    DEF00480
*****      2 ATTR(20)  UP TO 20 ATTRIBUTES MAY BE IN AN NSET   DEF00490
*****      3 ANAME     NAME OF ATTRIBUTE, CORRESPONDING TO A    DEF00500
*****      3 K_TYPE    '00000001'B IF UNIQUE, '00000000'B      DEF00510
*****                                     PREVIOUSLY DEFINED PSET. DEF00520
*****      3 K_TYPE    '00000001'B IF UNIQUE, '00000000'B      DEF00530
*****                                     OTHERWISE.              DEF00540
*****                                     DEF00550
*****      *****
*****      OUTPUT PARAMETERS:                                     DEF00560
*****      NCNE RETURNED.                                         DEF00570
*****                                                             DEF00580
*****      ***** DEF00590
*****      PROCEDURES INVOKED:                                     DEF00600
*****      DEFINEB,DEFINEP, NAMEGEN,INSERTN                       DEF00610
*****                                                             DEF00620
*****      *****/ DEF00630
*****      ***** DEF00010
*****      ***** DEF00020
*****      ***** DEF00030
2 1 0 %INCLUDE NCAT;*****
      DCL 1 NCAT, /* NSET CATALOGUE ENTRY */ DCL00010
      2 NNAME BIT(64), /* NAME OF NSET */ DCL00020
      2 NATTR BIT(8), /* NUMBER OF ATTRIBUTES */ DCL00030
      2 ATTR(20), /* UP TO 20 ATTRIBUTES */ DCL00040
      3 ANAME BIT(64), /* ATTRIBUTE NAME */ DCL00050
      3 K_TYPE, /* UNIQUE KEY OR NOT */ DCL00060
      3 BREL BIT(8), /* TYPE OF BSET */ DCL00070
      3 BSETUP, /* BSET(ATTR->N_NODE) */ DCL00080
      3 BSETDOWN BIT(64); /* BSET(N_NODE->ATTR) */ DCL00090
      ***** DEF00030
      ***** DEF00040
3 1 0 %INCLUDE DEFARG;***** DEF00050
      DCL 1 DEF_ARG, /* USED TO DEFINE AN NSET */ DCL00010
      2 NNAME BIT(64), /* NAME OF NSET */ DCL00020
      2 NATTR BIT(8), /* NUMBER OF ATTRIBUTES */ DCL00030
      2 ATTR(20), /* FOR EACH ATTRIBUTE */ DCL00040
      3 ANAME BIT(64), /* ATTRIBUTE NAME */ DCL00050
      3 K_TYPE BIT(8); /* UNIQUE KEY OR NOT */ DCL00060
      ***** DCL00070
      ***** DEF00050
      ***** DEF00060
4 1 0 %INCLUDE INSERTA;***** DEF00070
      DCL 1 INSERT_ARG, /* USED TO INSERT INTO AN NSET */ INS00010
      2 NNAME BIT(64), /* NAME OF NSET */ INS00020
      2 NATTR BIT(8), /* NUMBER OF ATTRIBUTES */ INS00030
      2 ATTR(20), /* FOR EACH ATTRIBUTE */ INS00040
      3 NAME BIT(64), /* NAME OF ATTRIBUTE */ INS00050
      3 VALUE BIT(320); /* VALUE TO BE INSERTED */ INS00060

```



```

*****
/* USED TO CREATE ATTRIBUTE STRING FOR INSERT */
5 1 0 DCL 1 ATTR_TEMP DEFINED ATTR_STR,
      2 ANAME BIT(64),
      2 K_TYPE,
      2 BREL ) BIT(8),
      2 BSETUP,
      2 BSETDOWN ) BIT(64),
      ATTR_STR BIT(208);
*****
%INCLUDE BSETSYM;*****
/* BSET LINK TYPES */
6 1 0 DCL A1_TO_1 BIT(8) INIT('00000001'B),
      A1_TO_N BIT(8) INIT('00000010'B),
      N_TO_1 BIT(8) INIT('00000100'B),
      M_TO_N BIT(8) INIT('00001000'B);
*****
7 1 0 DCL (N_NAME, NSETCAT,N_ATTR,BNSET1,BNSET2) BIT(64) STATIC
      EXTERNAL,
      UNIQUE BIT(8) STATIC EXTERNAL;
8 1 0 DCL TEMP BIT(64), ID POINTER,TEMP1 CHAR(8),L FIXED BIN(8),
      RECIP_BREL BIT(8);
      /* PROCEDURES CALLED */
%INCLUDE EDEFINP;*****
/* DEFINE PSET MODULE */
9 1 0 DCL DEF_NEP ENTRY((BIT(64),BIT(8),BIT(8),BIT(8),BIT(8),
      BIT(8),BIT(8),POINTER) ;
*****
%INCLUDE EDEFINB;*****
/* DEFINE BSET MODULE */
10 1 0 DCL DEFINEB ENTRY(BIT(64),BIT(64),BIT(64),BIT(8),BIT(1));
*****
%INCLUDE EINSERT;*****
/* INSERT NSET MODULE */
11 1 0 DCL INSERTN ENTRY(1, 2 BIT(64), 2 BIT(8), 2 (20),
      3 BIT(64), 3 BIT(320));
*****
%INCLUDE ENAMEGN;*****
/* RANDOM NAME GENERATOR */
12 1 0 DCL NAMEGEN ENTRY(FIXED BIN(15)) RETURNS(BIT(64));
*****
/* START OF NSET DEFINITION */
/* FILL NCAT ENTRY WITH INFORMATION PASSED IN CALL */
13 1 0 NCAT.NNAME=DEF_ARG.NNAME;

```

```

DEF00070
DEF00080
DEF00090
DEF00100
DEF00110
DEF00120
DEF00130
DEF00140
DEF00150
DEF00160
DEF00170
DEF00180
BCA00480
BCA00490
BCA00500
BCA00510
BCA00520
DEF00180
DEF00190
DEF00200
DEF00210
DEF00220
DEF00230
DEF00240
DEF00250
DEF00260
BCA00590
BCA00600
BCA00610
DEF00260
DEF00270
DEC00020
DEC00030
DEF00270
DEF00280
DEC00070
DEC00080
DEC00090
DEF00280
DEF00290
ECR00080
ECR00090
DEF00290
DEF00300
DEF00310
DEF00320
DEF00330
DEF00340
DEF00350

```

14	1	0	NCAT.NATTR=DEF_ARG.NATTR;	DEF00360
15	1	0	DO K = 1 TO NCAT.NATTR;	DEF00370
16	1	1	NCAT.ANAME(K)=DEF_ARG.ANAME(K);	DEF00380
17	1	1	NCAT.K_TYPE(K)=DEF_ARG(K).K_TYPE;	DEF00390
18	1	1	END;	DEF00400
				DEF00410
			/* CREATE ENTITY NODE PSET DEFINITION */	DEF00420
19	1	0	CALL DEFINEP(NCAT.NNAME,'00000001'B,'00000001'B,'00010000'B,	DEF00430
			'00010000'B,'0'B,'0'B,ID);	DEF00440
				DEF00450
			/* SET UP BSET DEFINITIONS FOR EACH ATTRIBUTE - ENTITY LINK */	DEF00460
20	1	0	DO J = 1 TO NCAT.NATTR;	DEF00470
			/* DETERMINE BINARY ASSOCIATION TYPE */	DEF00480
21	1	1	IF NCAT.K_TYPE(J) = UNIQUE	DEF00490
			THEN DO;	DEF00500
22	1	2	NCAT.BREL(J)= A1_TO_1;	DEF00510
23	1	2	RECIP_BREL=A1_TO_1;	DEF00520
24	1	2	END;	DEF00530
25	1	1	ELSE DO;	DEF00540
26	1	2	NCAT.BREL(J) = N_TO_1;	DEF00550
27	1	2	RECIP_BREL= A1_TO_N;	DEF00560
28	1	2	END;	DEF00570
				DEF00580
			/* GENERATE NAME FOR BSETDOWN */	DEF00590
29	1	1	NCAT(J).BSETDOWN = NAMEGEN(8);	DEF00600
30	1	1	TEMP = NCAT(J).BSETDOWN;	DEF00610
				DEF00620
			/* DEFINE BSETDOWN */	DEF00630
31	1	1	CALL DEFINEB(TEMP,NCAT.NNAME,NCAT.ATTR(J).ANAME,	DEF00640
			NCAT.BREL(J),'0'B);	DEF00650
				DEF00660
			/* DEFINE BSETUP AS RECIPROCAL OF BSETDOWN */	DEF00670
32	1	1	CALL DEFINEB(TEMP,NCAT.ATTR(J).ANAME,NCAT.NNAME,	DEF00680
			RECIP_BREL,'1'B);	DEF00690
33	1	1	NCAT(J).BSETUP = TEMP;	DEF00700
34	1	1	END;	DEF00710
				DEF00720
			/* SET UP INSERT_ARG FOR CALL TO INSERT NCAT INTO NSET_CAT */	DEF00730
35	1	0	INSERT_ARG.NNAME = NSETCAT;	DEF00740
36	1	0	L= 1 + NCAT.NATTR;	DEF00750
37	1	0	INSERT_ARG.NATTR=BIN(L);	DEF00760
38	1	0	INSERT_ARG.NAME(1)=N_NAME;	DEF00770
39	1	0	INSERT_ARG.VALUE(1)=NCAT.NNAME;	DEF00780
				DEF00790
			/* CREATE BIT STRING REP OF ATTRIBUTE DESCRIPTION */	DEF00800
40	1	0	DO J =1 TO NCAT.NATTR;	DEF00810
41	1	1	INSERT_ARG(J+1).NAME=N_ATTR;	DEF00820
42	1	1	ATTR_TEMP= NCAT.ATTR(J);	DEF00830
43	1	1	INSERT_ARG.VALUE(J+1)= ATTR_STR;	DEF00840

```
44 1 1      END;
      /* INSERT INTO NSET_CAT */
45 1 0      CALL INSERTN(INSERT_ARG);
46 1 0      RETURN;
47 1 0  END DEFINEN;
```

```
DEF00850
DEF00860
DEF00870
DEF00880
DEF00890
DEF00900
DEF00910
```


6	1	0	DCL INFO_ND BIT(320) EXTERNAL CONTROLLED;	BCA00460
				INS00110
			/* PROCEDURES CALLED */	INS00120
			%INCLUDE ECREATP;.....	INS00130
			/* CREATE PSET MODULE */	BCA00630
7	1	0	DCL CREATEP ENTRY(BIT(64),BIT(320),POINTER);	BCA00640
				INS00130
			%INCLUDE ECREATP;.....	INS00140
			/* CREATE BSET MODULE */	DEC00050
8	1	0	DCL CREATEB ENTRY(BIT(64),POINTER,BIT(320),BIT(320),POINTER);	DEC00060
				INS00140
			%INCLUDE ESELECTF;.....	INS00150
			/* BSET RETRIEVAL MODULE */	DEC00050
9	1	0	DCL SELECTF ENTRY(BIT(2),POINTER,BIT(64),BIT(320),POINTER);	DEC00060
				INS00150
			%INCLUDE EBUILD;.....	INS00160
			/* NSET CATALOGUE ENTRY RETRIEVAL MODULE */	DEC00110
10	1	0	DCL BUILD;.....	DEC00120
			3 BIT(64), 3 BIT(8), 3 BIT(8), 3 BIT(64),	DEC00130
			3 BIT(64));	DEC00140
				INS00160
			%INCLUDE ENAMEGN;.....	INS00170
			/* RANDOM NAME GENERATOR */	ECR00080
11	1	0	DCL NAMEGN ENTRY(FIXED BIN(15)) RETURNS(BIT(64));	ECR00090
				INS00170
			/* FILL NCAT WITH NSET DEFINITION */	INS00180
12	1	0	CALL BUILD;.....	INS00190
			/* CHECK FOR DUPLICATE KEY VALUES */	INS00200
			DO J= 1 TO NCAT.NATTR;	INS00210
13	1	0	IF NCAT.K_TYPE(J)=UNIQUE	INS00220
14	1	1	THEN DO;	INS00230
			ID2 = NULL();	INS00240
15	1	2	/* GET CORRESPONDING ENTRY IN INSERT_ARG */	INS00250
			DO L= 1 TO INSERT_ARG.NATTR;	INS00260
16	1	2	IF INSERT_ARG.NAME(L)=NCAT.ANAME(J)	INS00270
17	1	3	THEN LEAVE;	INS00280
18	1	3	END;	INS00290
			/* SEE IF INSTANCE ALREADY EXISTS */	INS00300
19	1	2	CALL SELECTF('01'B,ID2,BSETUP(J),VALUE(L),ID);	INS00310
20	1	2	K=ALLOCATION(INFO_ND);	INS00320
21	1	2	IF K ^=0	INS00330
			THEN DO;	INS00340
22	1	3	PUT SKIP EDIT('REQUEST IGNORED, DUPLICATE '	INS00350
			','KEY FOUND IN RELATION')(A);	INS00360
23	1	3	FREE INFO_ND;	INS00370
				INS00380
				INS00390
				INS00400
				INS00410

24	1	3		RETURN;	INS00420
25	1	3		END;	INS00430
26	1	2	END;		INS00440
27	1	1	END;		INS00450
			/* INSERTION ROUTINE */		INS00460
			/* GENERATE TAG FOR ENTITY NODE INSTANCE */		INS00470
28	1	0	NNN=NAMEGEN(8); NSET_ID=NNN;		INS00480
			/* CREATE INSTANCE OF ENTITY NODE */		INS00490
30	1	0	CALL CREATEP(NCAT.NNAME,NSET_ID,N_ID);		INS00500
			/* BUILD VALUE NODES */		INS00510
31	1	0	DO K=1 TO INSERT_ARG.NATTR;		INS00520
			/* FIND CORRESPONDING ENTRY IN INSERT_ARG */		INS00530
32	1	1	DO J=1 TO NCAT.NATTR;		INS00540
33	1	2	IF NCAT.ANAME(J)=INSERT_ARG.NAME(K)		INS00550
			THEN LEAVE;		INS00560
34	1	2	END;		INS00570
			/* LINK INSTANCE OF ATTRIBUTE TO INSTANCE OF ENTITY */		INS00580
35	1	1	CALL CREATEB(BSETDOWN(J),N_ID,NSET_ID,INSERT_ARG(K).VALUE,		INS00590
			ID2);		INS00600
36	1	1	END;		INS00610
37	1	0	RETURN;		INS00620
38	1	0	END INSERTN;		INS00630
					INS00640
					INS00650
					INS00660
					INS00670
					INS00680


```

***** LINK. SELECTF RETURNS THE ENTITY NODE WHICH LINKS FET00460
***** ALL THE ATTRIBUTE-VALUES FOR THAT TUPLE. THE VALUE OF FET00470
***** THE NSET NODE IS PLACED IN ID_LIST, WHICH IS A TEMPORARY FET00480
***** STRUCTURE, AND NRECON IS CALLED WHICH COMPARES THE VALUE FET00490
***** IN ID_LIST AND NLIST(I) (NOTE: THERE IS A NLIST FOR EACH FET00500
***** NSET SPECIFIED IN RET_ARG) AND CREATES A NEW NLIST(I) FET00510
***** WHICH CONTAINS ONLY THOSE NSET VALUES WHICH WERE IN FET00520
***** BOTH IDLIST AND NLIST(I). THE 2ND PASS IS SIMILAR EX- FET00530
***** THAT IT ONLY LOOKS AT NON-UNIQUE ATTRIBUTES FOR WHICH FET00540
***** VALUES WERE SPECIFIED. AT THE END OF THIS PROCESS IF FET00550
***** NO RESTRICTIONS WERE SPECIFIED FOR THE NSET ALL THE FET00560
***** VALUES FOR THE NSET NODE ARE FETCHED AND PLACED IN FET00570
***** NLIST(I). THIS PROCESS IS REPEATED FOR ALL OF THE NSET FET00580
***** SPECIFIED IN RET_ARG. AT THE CONCLUSION OF ALL THIS FET00590
***** NLIST WILL CONTAIN ONLY THOSE NODES WHICH MEET THE FET00600
***** SELECT CRITERIA IN RET_ARG. FET00610
4) THE NEXT TASK IS TO PERFORM ANY JOINS WHICH HAVE BEEN FET00620
***** SPECIFIED. THIS IS ACCOMPLISHED BY CALLING NJOIN1 WHICH FET00630
***** IS RESPONSIBLE FOR HANDLING JOINS. NJOIN1 IS PASSED FET00640
***** THE NUMBER OF NSETS INVOLVED, AND BECAUSE NLIST AND NCAT FET00650
***** ARE DEFINED TO BE EXTERNAL NJOIN1 HAS ACCESS TO THEM AS FET00660
***** WELL. NJOIN1 RETURNS A STACK(CTL) CALLED TUPLE WHICH FET00670
***** CORRESPONDS TO THE NSET NODES WHICH SATISFY THE JOIN. FET00680
***** IF ONLY ONE NSET HAS BEEN SPECIFIED, FETCH DOES NOT FET00690
***** CALL NJOIN1 BUT RATHER BUILDS TUPLE ITSELF. FET00700
5) THE NEXT STEP IS TO RECTANGULARIZE THE NSET NODES SO FET00710
***** A TABLE CAN BE CREATED. THIS TASK IS PERFORMED BY TABN FET00720
***** WHICH USES THE TUPLE STACK (ALSO DEFINED AS EXTERNAL) FET00730
***** TO CREATE A TABLE CALLED TAB, WHERE EACH ENTRY IN TAB FET00740
***** AN NSET NODE. ONLY THOSE NODES WHICH MET BOTH THE SELECT FET00750
***** AND THE JOIN RESTRICTIONS ARE CONTAINED IN TAB. FET00760
6) THE LAST STEP IS TO FETCH THE ACTUAL ATTRIBUTE VALUES FET00770
***** WHICH ARE LINKED TO THE NODES IN TAB AND WHICH ARE FET00780
***** REQUESTED IN RET_ARG (I.E. FOR WHICH RET_INFO.FETCH = FET00790
***** '1'B). THIS IS ACCOMPLISHED BY GOING THROUGH TAB A ROW FET00800
***** AT A TIME AND USING SELECTF TO FETCH THE ATTRIBUTE FET00810
***** ASSOCIATED WITH THE ENTITY NODES CONTAINED IN TAB IF T FET00820
***** ATTRIBUTE VALUE IS TO BE FETCHED. THE FETCHED VALUES FET00830
***** ARE PLACED ON A STACK CALLED DOM_RET (CTL EXT). EACH FET00840
***** ENTRY IN DOM_RET CONTAINS AN ID WHICH IDENTIFYS THE FET00850
***** NSET AND DOMAIN WITH WHICH THIS VALUE IS TO BE FET00860
***** ASSOCIATED AS WELL AS THE ACTUAL DATA ITEM. FET00870
***** FET00880
***** FET00890
***** FET00900
***** FET00910
***** FET00920
***** FET00930
***** FET00940
*****
***** INPUT PARAMETERS:
***** 1 RET_ARG
***** 2 NUMN NUMBER OF NSETS INVOLVED (UP TO 5)
***** 2 NSET(5) NAME OF EACH NSET (MUST BE PREVIOUSLY DEFINED)

```


3	1	0	DCL 1 RET_ARG, /* USED TO RETRIEVE NSETS */	DCL00090
			2 NUMN BIT(8), /* NUMBER OF NSETS */	DCL00100
			2 NSET(5) BIT(64), /* NAMES OF NSETS TO BE FETCHED */	DCL00110
			2 ARGS(20), /* INFO FOR EACH ATTRIBUTE */	DCL00120
			3 N_INDEX BIT(8), /* WHICH NSET IS THIS IN */	DCL00130
			3 NAME BIT(64), /* NAME OF ATTRIBUTE */	DCL00140
			3 RET_INFO, /* RETRIEVE INFORMATION */	DCL00150
			(4 FETCH, /* IS IT TO BE FETCHED */	DCL00160
			4 SAME) BIT(8), /* SAME AS PREVIOUSLY	DCL00170
			DEFINED DOMAIN */	DCL00180
			4 VALUE BIT(160); /*VALUE TO SEARCH ON OR	DCL00190
			NONE */	DCL00200
			*****	FET00060
			/* MISC DCL */	FET00070
4	1	0	DCL NOT_GIVEN BIT(8) INIT('01010101'B),	FET00080
			(ID, ID2, IDPOS) POINTER,	FET00090
			N_TAG BIT(32),	FET00100
			I FIXED BIN(8),	FET00110
			TEMP1 CHAR(8),	FET00120
			TEMP2 BIT(32);	FET00130
				FET00140
				FET00150
			/* STRUCTURE HOLDS NSET_CAT DEFINITIONS AND RET_INFO FOR EACH NSET */	FET00160
5	1	0	DCL 1 NCAT2(5) EXTERNAL,	FET00170
			2 NNAME BIT(64),	FET00180
			2 NATTR BIT(8),	FET00190
			2 ATTR(20),	FET00200
			3 ANAME BIT(64),	FET00210
			3 K_TYPE BIT(8),	FET00220
			3 BREL BIT(8),	FET00230
			3 BSETUP BIT(64),	FET00240
			3 BSETDOWN BIT(64),	FET00250
			3 RET_INFO,	FET00260
			4 FETCH BIT(8),	FET00270
			4 SAME BIT(8),	FET00280
			4 VALUE BIT(160);	FET00290
				FET00300
			/* HOLDS ENTITY NODE TAGS WHICH MEET SELECT RESTRICTIONS */	FET00310
6	1	0	DCL 1 NLIST(5) STATIC EXT,	FET00320
			2 NUM FIXED BIN(15),	FET00330
			2 NODE(50) BIT(32),	FET00340
				FET00350
			/* RECTANGULARIZED VERSION OF TUPLE */	FET00360
			1 TAB(5) STATIC EXT,	FET00370
			2 ROW_NUM FIXED BIN(15),	FET00380
			2 ROW(50) BIT(32),	FET00390
				FET00400
			/* HOLDS ENTITY NODE TAGS WHICH MEET SELECT AND JOIN CRIT */	FET00410
			1 TUPLE CTL EXT,	FET00420

```

                2 D_ID FIXED BIN(15),
                2 NODE BIT(32),
                /* DOMAIN VALUES RETURNED TO EXTERNAL LEVEL */
                1 DOM_RET CTL EXT,
                2 D_ID FIXED BIN(15),
                2 VALUE BIT(320),
                /* TEMPORARY STACK TO HOLD DOMAIN VALUES */
                1 D_TEMP CTL EXT,
                2 D_ID FIXED BIN(15),
                2 VALUE BIT(320),
                /* TEMPORARY COPY OF NLIST */
                1 IDLIST STATIC EXTERNAL,
                2 NUM FIXED BIN(15) INIT(0),
                2 NODE(50) BIT(32);

%INCLUDE IDS1;*****
                /* POINTER STACK RETURNED BY SEARCH */
7   1   0   DCL IDS1 PTR EXTERNAL CONTROLLED;
                *****
%INCLUDE INFOIND;*****
8   1   0   /* DATA STACK RETURNED BY FETCH */
                DCL INFO_IND BIT(320) EXTERNAL CONTROLLED;
                *****
9   1   0   DCL IDP1 POINTER EXT CTL,
                UNIQUE BIT(8) STATIC EXT, IDXX PTR EXT;

                /* PROCEDURES CALLED */
%INCLUDE ENJOIN1;*****
10  1   0   /* JOIN MODULE */
                DCL NJOIN1 ENTRY RETURNS(FIXED BIN(15));
                *****
%INCLUDE ESELECTF;*****
11  1   0   /* BSET RETRIEVAL MODULE */
                DCL SELECTF ENTRY(BIT(2),POINTER,BIT(64),BIT(320),POINTER);
                *****
%INCLUDE EFETCH;*****
12  1   0   /* FETCH PSET MODULE */
                DCL FETCH ENTRY(BIT(2),POINTER,BIT(64),BIT(64),BIT(1));
                *****
%INCLUDE ESEARCH;*****
13  1   0   /* SEARCH MODULE */
                DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER);
                *****
%INCLUDE EBUILD;*****
                /* PSET CATALOGUE ENTRY RETRIEVAL MODULE */

```

```

FET00430
FET00440
FET00450
FET00460
FET00470
FET00480
FET00490
FET00500
FET00510
FET00520
FET00530
FET00540
FET00550
FET00560
FET00570
FET00580
FET00590
FET00600
FET00610
BCA00420
BCA00430
FET00610
FET00620
BCA00450
BCA00460
FET00620
FET00630
FET00640
FET00650
FET00660
FET00670
DEC00020
DEC00030
FET00670
FET00680
DEC00050
DEC00060
FET00680
FET00690
EFE00010
EFE00020
FET00690
FET00700
BCA00700
BCA00710
BCA00720
FET00700
FET00710
DEC00110

```

```

14 1 0      DCL BUILD ENTRY(BIT(64), 1, 2 BIT(64), 2 BIT(8), 2 (20),
              3 BIT(64), 3 BIT(8), 3 BIT(8), 3 BIT(64),
              3 BIT(64));
              *****
              %INCLUDE ETABN;*****
              /* RECTANGULARIZATION MODULE */
15 1 0      DCL TABN ENTRY;
              *****
              %INCLUDE ENRECO;*****
              /* ENTITY NODE RESTRICTION MODULE */
16 1 0      DCL NRECON ENTRY(BIT(1), 1, 2 FIXED BIN(15), 2 (*) BIT(32),
              1, 2 FIXED BIN(15), 2 (50) BIT(32));
              *****

              /* BUILD NSET)CAT */
17 1 0      K=1;
18 1 0      DO I=1 TO RET_ARG.NUMN;
19 1 1      IK=BIN(I);
              /* GET NSET_CAT ENTRY FOR NSET */
20 1 1      CALL BUILD(RET_ARG.NSET(I), NCAT);
21 1 1      NCAT2(I).NNAME=NCAT.NNAME;
22 1 1      NCAT2(I).NATTR=NCAT.NATTR;
23 1 1      JJ=0;
              /* FOR EACH ATTRIBUTE IN NSET */
24 1 1      DO III=K TO 20 WHILE(N_INDEX(III)=IK);
25 1 2      JJ=JJ+1;

              /* FIND RET_ARG ENTRY FOR ATTRIBUTE */
26 1 2      DO II=1 TO NCAT2(I).NATTR;
27 1 3      IF RET_ARG.NAME(III)=NCAT.ANAME(II)
              THEN LEAVE;
28 1 3      END;

              /* ENTER REST OF INFO FOR ATTRIBUTE */
29 1 2      NCAT2(I).ANAME(JJ)=NCAT.ANAME(II);
30 1 2      NCAT2(I).K_TYPE(JJ)=NCAT.K_TYPE(II);
31 1 2      NCAT2(I).BREL(JJ)=NCAT.BREL(II);
32 1 2      NCAT2(I).BSETUP(JJ)=NCAT.BSETUP(II);
33 1 2      NCAT2(I).BSETDOWN(JJ)=NCAT.BSETDOWN(II);
34 1 2      NCAT2(I).ATTR(JJ).RET_INFO=RET_ARG(III).RET_INFO;
35 1 2      END;
36 1 1      K=III;
37 1 1      END;

              /* END OF BUILD CAT SEQUENCE */

              /* INITIALIZE NLTEMP */
38 1 0      DO J=1 TO 5;
39 1 1      NLIST(J).NUM=0;

```

```

DEC00120
DEC00130
DEC00140
FET00710
FET00720
DEC00160
DEC00170
FET00720
FET00730
DEC00190
DEC00200
DEC00210
FET00730
FET00740
FET00750
FET00760
FET00770
FET00780
FET00790
FET00800
FET00810
FET00820
FET00830
FET00840
FET00850
FET00860
FET00870
FET00880
FET00890
FET00900
FET00910
FET00920
FET00930
FET00940
FET00950
FET00960
FET00970
FET00980
FET00990
FET01000
FET01010
FET01020
FET01030
FET01040
FET01050
FET01060
FET01070
FET01080
FET01090

```

```

40 1 1      DO JJ=1 TO 50;                                FET01100
41 1 2      NLIST(J).NODE(JJ)=0;                          FET01110
42 1 2      END;                                           FET01120
43 1 1      END;                                           FET01130
                                                    FET01140
                                                    FET01150
                                                    FET01160
                                                    FET01170
                                                    FET01180
                                                    FET01190
                                                    FET01200
                                                    FET01210
                                                    FET01220
                                                    FET01230
                                                    FET01240
                                                    FET01250
                                                    FET01260
                                                    FET01270
                                                    FET01280
                                                    FET01290
                                                    FET01300
                                                    FET01310
                                                    FET01320
                                                    FET01330
                                                    FET01340
                                                    FET01350
                                                    FET01360
                                                    FET01370
                                                    FET01380
                                                    FET01390
                                                    FET01400
                                                    FET01410
                                                    FET01420
                                                    FET01430
                                                    FET01440
                                                    FET01450
                                                    FET01460
                                                    FET01470
                                                    FET01480
                                                    FET01490
                                                    FET01500
                                                    FET01510
                                                    FET01520
                                                    FET01530
                                                    FET01540
                                                    FET01550
                                                    FET01560
                                                    FET01570
                                                    FET01580

/* RESTRICTION PHASE */
/* FOR EACH NSET IN NCAT2 */
DO J=1 TO RET_ARG.NUMN;
  RESTRICT='0'B;
  DO JJ=1 TO 2;

    /* PASS1 - RESTRICTS ON KEY, PASS 2 OTHER RESTRICTS */
    DO K = 1 TO NCAT2(J).NATTR;
      IF NCAT2(J).VALUE(K) ^= NOT_GIVEN
        THEN IF ((NCAT2(J).K_TYPE(K) ^= UNIQUE &
                  JJ=2) | (NCAT2(J).K_TYPE(K) = UNIQUE &
                           JJ=1))
          THEN DO;

            /* GET ID OF ATTRIBUTE INSTANCE IN NSET */
            CALL SEARCH('01'B, NCAT2(J).ANAME(K),
                       NCAT2(J).VALUE(K), IDXX, IDPOS);

            IF ALLOCATION(IDS1) = 0
              THEN DO;
                PUT SKIP EDIT('NO TUPLE EXISTS')(A);
                RETURN;
              END;

            /* SET ID TO POINT TO INSTANCE */
            ID=IDS1;
            FREE IDS1;

            /* GET ASSOCIATED NSET NODES */
            CALL SELECTF('11'B, ID, NCAT2(J).BSETUP(K),
                       '0'B, ID2);

            /* PLACE RETURNED ENTITY NODES IN IDLIST */
            IDLIST.NUM=ALLOCATION(INFO_ND);
            DO JK = 1 TO IDLIST.NUM;
              IDLIST.NODE(JK)=INFO_ND;
              FREE INFO_ND;
            END;

            /* CALL NRECON TO GET INTERSECTION WITH NLTEMP */
            CALL NRECON(RESTRICT, NLIST(J), IDLIST);
            RESTRICT='1'B;
          END;
        END;
      END;
    END;
  END;
END;

```

67	1	1	/* IF NO RESTRICTIONS ON NSET GET ALL ENTITY NODES */	FET01590
			IF RESTRICT='0'B	FET01600
			THEN DO;	FET01610
68	1	2	ID=NULL();	FET01620
69	1	2	CALL FETCH('11'B, ID, NCAT2(J).NNAME, '0'B, '1'B);	FET01630
70	1	2	NLIST(J).NUM=ALLOCATION(INFO_ND);	FET01640
71	1	2	DO I= 1 TO NLIST(J).NUM;	FET01650
72	1	3	NLIST(J).NODE(I)=INFO_ND;	FET01660
73	1	3	FREE INFO_ND;	FET01670
74	1	3	END;	FET01680
75	1	2	END;	FET01690
76	1	1	END;	FET01700
			/* BEGIN JOIN LOGIC */	FET01710
77	1	0	IF NUMN>1 THEN	FET01720
			DO;	FET01730
78	1	1	L=NUMN;	FET01740
			/* CALL NJOIN1 TO HANDLE JOIN LOGIC */	FET01750
79	1	1	NUM_NODES=NJOIN1(L);	FET01760
80	1	1	END;	FET01770
			/* IF NO JOINS BUILD TUPLE FOR NSET 1 */	FET01780
81	1	0	ELSE DO;	FET01790
82	1	1	DO K=1 TO NLIST.NUM(1);	FET01800
83	1	2	ALLOCATE TUPLE;	FET01810
84	1	2	TUPLE.D_ID=1;	FET01820
85	1	2	TUPLE.NODE=NLIST(1).NODE(K);	FET01830
86	1	2	END;	FET01840
87	1	1	END;	FET01850
			/* RECTANGULARIZE CONTENTS OF TUPLE FOR RELATION */	FET01860
88	1	0	CALL TABN;	FET01870
			/* CONSTRUCT DOMAIN STACK */	FET01880
89	1	0	DO I=1 TO TAB(1).ROW_NUM;	FET01890
90	1	1	DO II=1 TO NUMN;	FET01900
91	1	2	DO III=1 TO NCAT2(II).NATTR;	FET01910
			/* IF TO BE FETCHED AND NOT FETCHED ALREADY */	FET01920
92	1	3	IF NCAT2(II).FETCH(III)='1'B &	FET01930
			NCAT2(II).SAME(III)='00000000'B	FET01940
			THEN DO;	FET01950
			/* FETCH DATA VALUE */	FET01960
93	1	4	ID=NULL();	FET01970
94	1	4	CALL SELECTF('01'B, ID, NCAT2(II).BSETDOWN	FET01980
			(III), TAB(II).ROW(I), ID2);	FET01990
				FET02000
				FET02010
				FET02020
				FET02030
				FET02040
				FET02050
				FET02060
				FET02070

95	1	4		/* PLACE ON TEMPORARY STACK */	FET02080
96	1	4		ALLOCATE D_TEMP;	FET02090
97	1	4		D_TEMP.D_ID=11+16+111;	FET02100
98	1	4		D_TEMP.VALUE=INFO_NO;	FET02110
99	1	4		FREE INFO_NO;	FET02120
100	1	3		END;	FET02130
101	1	2		END;	FET02140
				/* REVERSE ORDER OF ELEMENTS ON STACK */	FET02150
102	1	1		DO WHILE(ALLOCATION(D_TEMP)~=0);	FET02160
103	1	2		ALLOCATE DOM_RET;	FET02170
104	1	2		DOM_RET = D_TEMP;	FET02180
105	1	2		FREE D_TEMP;	FET02190
106	1	2		END;	FET02200
107	1	1		END;	FET02210
108	1	0		RETURN;	FET02220
109	1	0		END FETCHT;	FET02230
					FET02240
					FET02250
					FET02260


```

%INCLUDE NJJOIN1;*****NJ000010
/*****FOR00010
*      MODULE DESCRIPTION      *FOR00020
*****/*****FOR00030
1 0 NJJOIN1:      PROCEDURE      FOR00040
      (L /*FIXED BIN(15) */)      FOR00050
      RETURNS(FIXED BIN(15))      FOR00060
      RECURSIVE;      FOR00070
/*****FOR00080
*****FOR00090
*****PURPOSE:      FOR00100
*****      THIS MODULE IS RESPONSIBLE FOR HANDLING JOINS BETWEEN      FOR00110
*****      ONE OR MORE NSETS. WHEN ORIGINALLY CALLED BY FETCH IT      FOR00120
*****      MAKES USE OF THE INFORMATION CONTAINED IN NCAT2 TO      FOR00130
*****      DETERMINE WHAT DOMAINS ARE TO BE JOINED, AND IT USES      FOR00140
*****      THE NSET NODES CONTAINED IN NLIST AS ITS UNIVERSE OF      FOR00150
*****      POSSIBLE TUPLES. AS CURRENTLY IMPLEMENTED IT WILL ONLY      FOR00160
*****      HANDLE JOINS ON ELEMENTS HAVING EQUAL VALUES.      FOR00170
*****      ALSO, IT REQUIRES THAT ALL JOINS BE EXPRESSED IN TERMS      FOR00180
*****      PREVIOUS NSETS. THAT IS, WHEN AN ATTRIBUTE IN NSET L IS      FOR00190
*****      DEFINED TO BE THE SAME AS A DOMAIN IN ANOTHER NSET J,      FOR00200
*****      J MUST BE LESS THAN L. THIS RESTRICTION DOES NOT COMPROMIS      FOR00210
*****      THE GENERALITY OF THE JOIN LOGIC. IT DOES ,HOWEVER, RESTRI      FOR00220
*****      THE MANNER IN WHICH THE JOIN IS EXPRESSED.      FOR00230
*****      FOR00240
*****      FOR00250
*****METHOD:      FOR00260
*****      THIS IS A RECURSIVE PROCEDURE WHICH BUILDS TUPLES A TUPLE      FOR00270
*****      AT A TIME, AND CREATES A STACK OF NSET NODES WHICH COR-      FOR00280
*****      RESPOND TO THE JOINED TUPLES. THE STRATEGY EMPLOYED IS      FOR00290
*****      AS FOLLOWS:      FOR00300
*****      1)IT BEGINS BY CREATING A TEMPORARY COPY OF THE RELEVANT      FOR00310
*****      NSET NODES IN A CONTROLLED STRUCTURE CALLED NLTEMP      FOR00320
*****      IF NO PREVIOUS ALLOCATIONS OF NLTEMP EXIST THEN A COPY      FOR00330
*****      OF NLIST IS MADE. OTHERWISE, A COPY OF THE MOST RE-      FOR00340
*****      CENT ALLOCATION OF NLTEMP IS MADE.      FOR00350
*****      2)IT THEN GOES THROUGH THE NODES CONTAINED IN NLTEMP(L)      FOR00360
*****      WHICH CORRESPONDS TO THE NSET NODES OF THE LAST NSET      FOR00370
*****      TO BE JOINED. FOR EACH NODE IT GOES THROUGH THE FOL-      FOR00380
*****      LOWING LOOP:      FOR00390
*****      A)IT PLACES A COPY OF THE NODE ON THE TOP OF A STACK      FOR00400
*****      CALLED TUPLE.      FOR00410
*****      B)IT THEN GOES THROUGH ALL THE ATTRIBUTES OF THAT      FOR00420
*****      NSET NODE TO SEE IF ANY ATTRIBUTE IS DEFINED IN      FOR00430
*****      NCAT2 TO BE THE SAME AS AN ATTRIBUTE OF ANOTHER      FOR00440
*****      NSET. IF SO:      FOR00450
*****      1) IT DETERMINES THE NSET AND DOMAIN WITHIN THAT

```

```

..... NSET BY DECOMPOSING NCAT2.SAME FOR THE ATTRIBUTE. FOR00460
..... THIS IS REFERRED TO AS J_NSET AND J_DOMAIN FOR00470
..... RESPECTIVELY. FOR00480
2) IT THEN FINDS THE OCCURENCE OF THE ATTRIBUTE FOR00490
   WHICH IS LINKED TO THE NSET NODE ON THE TUPLE STACK FOR00500
   THIS IS ACCOMPLISHED VIA A CALL TO SELECTF. FOR00510
3) IT THEN RETRIEVES ALL OF THE NSET NODES FOR00520
   IN J_NSET WHICH ARE ASSOCIATED WITH THAT OCCURENCE FOR00530
   OF J_DOMAIN. THIS IS ACCOMPLISHED THROUGH A CALL FOR00540
   TO SELECTF, PASSING IT THE VALUE OF THE ATTRIBUTE FOR00550
   NODE AND THE NAME OF THE BSET WHICH LINKS THE FOR00560
   J_DOMAIN ATTRIBUTE IN J_NSET TO THE ENTITY NODE FOR00570
   IN J_NSET. FOR00580
4) IF THE NUMBER OF NSET NODES FOUND = 0 FOR00590
   THEN NO JOIN IS POSSIBLE WITH THE NSET NODE FOR00600
   CURRENTLY ON THE TOP OF THE STACK AND SO YOU FOR00610
   POP THE STACK AND DROP OUT OF THE LOOP. OTHERWISE FOR00620
   THE NSET NODES FOUND ARE COMPARED WITH THE FOR00630
   NSET NODES CONTAINED IN NLTEMP(J_NSET) AND A FOR00640
   A TEMPORARY STRUCTURE IS CREATED WHICH CONTAINS FOR00650
   THE INTERSECTION OF NLTEMP(J_NSET) AND THE NSET FOR00660
   NODES FOUND. IF THE INTERSECTION IS EMPTY, NO FOR00670
   JOIN IS POSSIBLE, THE STACK IS POPPED AND YOU FOR00680
   DROP OUT OF THE LOOP. OTHERWISE, NLTEMP(J_NSET) FOR00690
   EQUAL TO THE TEMPORARY STRUCTURE. FOR00700
5) THIS LOOP IS CONTINUED FOR ALL OF THE ATTRIBUTES FOR00710
   ASSOCIATED WITH THE NSET NODE ON THE TOP OF THE FOR00720
   STACK. AT THE END OF THE LOOP NLTEMP WILL CONTAIN FOR00730
   ONLY THOSE NSET NODES WHICH ARE CONSISTENT WITH FOR00740
   JOINS WITH THE LTH NSET AND FOR THE PARTICULAR FOR00750
   OCCURENCE OF THE NSET NODE. FOR00760
C) IT THEN CHECKS THE CURRENT VALUE OF L, IF L > FOR00770
   2 THEN NJOIN1 CALLS ITSELF, PASSING ITSELF FOR00780
   L-1. THE EFFECT OF THIS IS TO PERFORM THE JOIN FOR00790
   LOGIC ON THE NODES IN NLTEMP FOR NSETS (L=1 FOR00800
   TO L-1). WHEN NJOIN1 RETURNS, IT RETURNS THE FOR00810
   NUMBER OF NSET NODES WHICH IT ADDED TO THE TUPLE FOR00820
   STACK. IF NO NODES WERE ADDED THEN NO JOIN IS POS- FOR00830
   SIBLE AND THE STACK IS POPPED, OTHERWISE THE FOR00840
   STACK CONTAINS 1 OR MORE COMPLETE TUPLES ASSOCIATED FOR00850
   WITH THE ORGINAL NODE ON THE TOP OF THE STACK. IT FOR00860
   THEN PROCEEDS TO THE NEXT NSET NODE ON NLTEMP(L). FOR00870
D) IF L= 2 THEN THE CONTENTS OF NLTEMP(1) ARE PLACED FOR00880
   ON THE TOP OF THE STACK. IF NLTEMP IS EMPTY THEN FOR00890
   NO JOIN IS POSSIBLE AND THE STACK IS POPPED. IN FOR00900
   ANY EVENT IT THEN PROCEEDS TO THE NEXT NSET NODE FOR00910
   ON NLTEMP(L). FOR00920
3) AFTER GOING THROUGH ALL THE NSET NODES IN NLTEMP(L) FOR00930
   IT FREES THE CURRENT ALLOCATION OF NLTEMP AND RETURNS FOR00940

```

```
%INCLUDE NJJOIN1;
```

```

*****
THE NUMBER OF NODES ADDED TO THE TUPLE STACK.
*****
INPUT PARAMETERS:
L - THE HIGHEST RELEVANT NSET (SEE PURPOSE)
NOTE: THIS PROCEDURE MAKES USE OF SEVERAL EXTERNAL
      ARGUMENTS, INCLUDING NLIST (CREATED BY FETCHT)
      NCAT2 (ALSO CREATED BY FETCHT)
*****
OUTPUT PARAMETERS:
1 TUPLE_CTL_EXT
2 D_ID BIT(8) SPECIFIES WHICH NSET THIS IS
                FOR WHICH THIS IS AN NSET NODE
2 NODE_BIT(32) CONTAINS THE NSET_NODE_TAG WHICH
                UNIQUELY IDENTIFIES AN OCCURENCE
                OF AN NSET.
*****
PROCEDURES INVOKED:
SELECTF, NJOIN1
*****

```

FOR00950
FOR00960
FOR00970
FOR00980
FOR00990
FOR01000
FOR01010
FOR01020
FOR01030
FOR01040
FOR01050
FOR01060
FOR01070
FOR01080
FOR01090
FOR01100
FOR01110
FOR01120
FOR01130
FOR01140
FOR01150
FOR01160
FOR01170
NJ000010
NJ000020
NJ000030
NJ000040
NJ000050
NJ000060
NJ000070
NJ000080
NJ000090
NJ000100
NJ000110
NJ000120
NJ000130
NJ000140
NJ000150
NJ000160
NJ000170
NJ000180
NJ000190
NJ000200
NJ000210
NJ000220
NJ000230
NJ000240
NJ000250
NJ000260

-258-

2 1 0

/* RETRIEVAL AND NSET ORGANIZATION INFO */

```

DCL 1 NCAT2(5) EXTERNAL,
    2 NNAME BIT(64),
    2 NATTR BIT(8),
    2 ATTR(20),
        3 ANAME BIT(64),
        3 K_TYPE BIT(8),
        3 BREL BIT(8),
        3 SSETUP BIT(64),
        3 BSFTDOWN BIT(64),
        3 RET_INFO,
        4 FETCH BIT(8),
        4 SAME BIT(8),
        4 VALUE BIT(160);

```

3 1 0

```

/* HOLDS ENTITY NODES THAT SATISFIED RESTRICTIONS */

```

```

DCL 1 NLIST(5) STATIC EXTERNAL,
    2 NUM FIXED BIN(15),
    2 NOCE(50) BIT(32);

```

4 1 0

```
/* TEMPORARY STRUCTURE TO HOLD NODES MEETING A GIVEN
JOIN RESTRICTION */
```

DCL 1 NLTEMP(5) CTL EXTERNAL,
2 NUM FIXED BIN(15).

```

                2 NODE(50)BIT(32);                                NJ000270
                                                                NJ000280
/* STACK OF ENTITY NODES MEETING JOIN AND SELECT RESTRICTIONS */ NJ000290
5   1   0   DCL 1 TUPLE CTL EXT,                                NJ000300
                2 D_ID FIXED BIN(15),                          NJ000310
                2 NODE BIT(32);                                NJ000320
6   1   0   DCL 1 TEMP(5),                                    NJ000330
                2 NUM FIXED BIN(15),                            NJ000340
                2 NODE(50)BIT(32);                             NJ000350
                                                                NJ000360
7   1   0   DCL 1 T_LIST,                                    NJ000370
                2 NUM FIXED BIN(15),                            NJ000380
                2 NODE(50) BIT(32);                             NJ000390
                                                                NJ000400
%INCLUDE IDS1;*****NJ000400
/* POINTER STACK RETURNED BY SEARCH */                          BCA00420
8   1   0   DCL IDS1 PTR EXTERNAL CONTROLLED;                  BCA00430
                                                                NJ000400
%INCLUDE INFOIND;*****NJ000410
/* DATA STACK RETURNED BY FETCH */                              BCA00450
9   1   0   DCL INFO_NO BIT(320) EXTERNAL CONTROLLED;          BCA00460
                                                                NJ000410
*****NJ000410
/* MISC DECLARATIONS */                                         NJ000420
10  1   0   DCL (ID,ID2,IDPOS) POINTER INIT( NULL()),N_TAG CHAR(8); NJ000430
11  1   0   DCL IDAX POINTER CONTROLLED;                       NJ000440
12  1   0   DCL (OK,FND,STATUS) BIT(1) INIT('1'B),            NJ000450
                TEMP1 BIT(320);                                NJ000460
                /* PROCEDURES CALLED */                         NJ000470
                                                                NJ000480
%INCLUDE ENJOIN1;*****NJ000490
/* JOIN MODULE */                                              DEC00020
13  1   0   DCL NJCINT ENTRY RETURNS(FIXED BIN(15));          DEC00030
                                                                NJ000490
*****NJ000490
%INCLUDE ESELECF;*****NJ000500
/* BSET RETRIEVAL MODULE */                                    DEC00050
14  1   0   DCL SELECTF ENTRY(BIT(2),POINTER,BIT(64),BIT(320),POINTER); DEC00060
                                                                NJ000500
*****NJ000500
%INCLUDE EFETCH;*****NJ000510
/* FETCH PSET MODULE */                                        EFE00010
15  1   0   DCL FETCH ENTRY(BIT(2),POINTER,BIT(64),BIT(64),BIT(1)); EFE00020
                                                                NJ000510
*****NJ000510
%INCLUDE ESEARCH;*****NJ000520
/* SEARCH MODULE */                                           BCA00700
16  1   0   DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER); BCA00710
                                                                BCA00720
*****NJ000520
                                                                NJ000530
/* IF FIRST CALL, THEN COPY NLIST INTO NLTEMP */              NJ000540
17  1   0   IF ALLOCATION(NLTEMP) = 0                           NJ000550
                THEN DO;                                       NJ000560

```

18	1	1	TEMP=NLIST;	NJ000570
19	1	1	ALLOCATE NLTEMP;	NJ000580
20	1	1	NLTEMP=TEMP;	NJ000590
21	1	1	END;	NJ000600
				NJ000610
			/* FOR EACH ELEMENT IN THE HIGHEST ORDER NLIST */	NJ000620
22	1	0	DO I=1 TO NLTEMP(L).NUM;	NJ000630
23	1	1	IF ALLOCATION(NLTEMP)=0	NJ000640
			THEN DO;	NJ000650
24	1	2	ALLOCATE NLTEMP;	NJ000660
25	1	2	NLTEMP=NLIST;	NJ000670
26	1	2	END;	NJ000680
				NJ000690
			/* PLACE ELEMENT ON TOP OF TUPLE STACK */	NJ000700
27	1	1	ALLOCATE TUPLE;	NJ000710
28	1	1	NODES_ADDED=1;	NJ000720
29	1	1	TUPLE.D_ID=L;	NJ000730
30	1	1	TUPLE.NODE=NLTEMP(L).NODE(I);	NJ000740
				NJ000750
			/* INPREPARATION FOR JOIN CHECKING GET FRESH COPY OF NLTEMP */	NJ000760
31	1	1	TEMP=NLTEMP;	NJ000770
32	1	1	ALLOCATE NLTEMP;	NJ000780
33	1	1	NLTEMP=TEMP;	NJ000790
				NJ000800
			/* FOR EACH ATTRIBUTE OF THE LTH NSET */	NJ000810
34	1	1	DO II = 1 TO NCAT2(L).NATTR;	NJ000820
			/* IF ATTRIBUTE THE SAME AS A PREVIOUS ATTRIBUTE */	NJ000830
35	1	2	IF NCAT2(L).SAME(II)='00000000'B	NJ000840
			THEN DO;	NJ000850
			/* CALCULATE WHICH NSET AND DOMAIN */	NJ000860
36	1	3	J_NSET=(BOOL(NCAT2(L).SAME(II),'11110000'B,	NJ000870
			'0001'B))/16.;	NJ000880
37	1	3	J_DOMAIN=BOOL(NCAT2(L).SAME(II),'00001111'B,	NJ000890
			'0001'B);	NJ000900
			/* GET INSTANCE OF ATTRIBUTE IN THE LTH NSET	NJ000910
			WHICH IS ASSOCIATED WITH THE I TH ENTITY	NJ000920
			NODE */	NJ000930
38	1	3	ID=NULL();	NJ000940
39	1	3	CALL SELECTF('11'B,ID,NCAT2(L).BSETDOWN(II),	NJ000950
			NLTEMP(L).NODE(I),ID2);	NJ000960
40	1	3	TEMP1=INFO_NO;	NJ000970
41	1	3	FREE INFO_NO;	NJ000980
				NJ000990
			/* GET ASSOCIATED INSTANCES OF ENTITY NODES IN	NJ001000
			THE J_NSET TH WHICH SHARE THE COMMON	NJ001010
			ATTRIBUTE VALUE */	NJ001020
42	1	3	ID=NULL();	NJ001030
43	1	3	CALL SELECTF('11'B,ID,NCAT2(J_NSET).BSETUP	NJ001040
			(J_DOMAIN),TEMP1,ID2);	NJ001050

44	1	3	NUM_FND=ALLOCATION(INFO_ND);	NJ001060
				NJ001070
45	1	3	IF NUM_FND= 0	NJ001080
			/* IF NONE FOUND, THIS ENTITY NODE CAN NOT BE	NJ001090
			JOINED, HENCE REMOVE FROM TUPLE */	NJ001100
			THEN DO;	NJ001110
46	1	4	FREE TUPLE;	NJ001120
47	1	4	NODES_ADDED=0;	NJ001130
48	1	4	END;	NJ001140
49	1	3	ELSE DO;	NJ001150
			/* OTHERWISE GET INTERSECTION OF NODES	NJ001160
			FOUND AND NODES IN CURRENT COPY OF	NJ001170
			NLTEMP FOR THAT NSET, PUT UNION IN	NJ001180
			T_LIST */	NJ001190
50	1	4	T_LIST.NUM=0;	NJ001200
51	1	4	DO J=1 TO NUM_FND;	NJ001210
52	1	5	FND='0'B;	NJ001220
53	1	5	DO LL=1 TO NLTEMP(J_NSET).NUM WHILE(~FND);	NJ001230
54	1	6	IF INFO_ND=NLTEMP(J_NSET).NODE(LL)	NJ001240
			THEN FND= '1'B;	NJ001250
55	1	6	END;	NJ001260
56	1	5	IF FND	NJ001270
			THEN DO;	NJ001280
57	1	6	T_LIST.NUM=T_LIST.NUM+1;	NJ001290
58	1	6	T_LIST.NODE(T_LIST.NUM)=INFO_ND;	NJ001300
59	1	6	END;	NJ001310
60	1	5	FREE INFO_ND;	NJ001320
61	1	5	END;	NJ001330
			/* IF INTERSECTION IS EMPTY, NO JOIN */	NJ001340
62	1	4	IF T_LIST.NUM=0	NJ001350
			THEN DO;	NJ001360
63	1	5	FREE TUPLE;	NJ001370
64	1	5	NODES_ADDED=0;	NJ001380
65	1	5	END;	NJ001390
66	1	4	ELSE NLTEMP(J_NSET)=T_LIST;	NJ001400
67	1	4	END;	NJ001410
68	1	3	END;	NJ001420
			/* IF JOIN ATTEMPT UNSUCCESSFUL FOR THIS NODE	NJ001430
			GO ON TO NEXT NODE */	NJ001440
69	1	2	IF NODES_ADDED=0	NJ001450
			THEN LEAVE;	NJ001460
70	1	2	END;	NJ001470
			/* OTHERWISE PERFORM JOIN LOGIC ON L-1TH NSET */	NJ001480
71	1	1	IF NODES_ADDED=0	NJ001490
			THEN DO;	NJ001500
			/* IF MORE THAN 1 NSET REMAINS CALL NJJOIN1*/	NJ001510
72	1	2	IF L>2	NJ001520
			THEN ADDED=NJOIN1(L-1);	NJ001530
				NJ001540

73	1	2	ELSE DO;	NJ001550
			/* OTHERWISE NLTEMP(1) WILL CONTAIN	NJ001560
			ONLY THOSE ENTITY NODES FOR NSET 1	NJ001570
			THAT SATISFY THE JOIN LOGIC, HENCE	NJ001580
			THEY ARE PLACED ON THE TUPLE STACK*/	NJ001590
74	1	3	DO K=1 TO NLTEMP(1).NUM;	NJ001600
75	1	4	ALLOCATE TUPLE;	NJ001610
76	1	4	TUPLE.D_ID=1;	NJ001620
77	1	4	TUPLE.NODE=NLTEMP(1).NODE(K);	NJ001630
78	1	4	END;	NJ001640
79	1	3	ADDED=NLTEMP(1).NUM;	NJ001650
80	1	3	END;	NJ001660
			/* IF NONE ADDED THEN JOIN WAS NOT	NJ001670
			SATISFIED NODE SHOULD BE REMOVED FROM	NJ001680
			TUPLE STACK */	NJ001690
81	1	2	IF ADDED=0	NJ001700
			THEN DO;	NJ001710
82	1	3	FREE TUPLE;	NJ001720
83	1	3	NODES_ADDED=0;	NJ001730
84	1	3	END;	NJ001740
			/* UPDATE NODE_ADDED */	NJ001750
85	1	2	ELSE NODES_ADDED=NODES_ADDED+ADDED;	NJ001760
86	1	2	END;	NJ001770
			/* FREE CURRENT ALLOCATION OF NLTEMP WHICH WAS	NJ001780
			USED FOR THIS INSTANCE OF THE ENTITY NODE */	NJ001790
87	1	1	FREE NLTEMP;	NJ001800
88	1	1	END;	NJ001810
89	1	0	FREE NLTEMP;	NJ001820
90	1	0	RETURN(NODES_ADDED);	NJ001830
91	1	0	END NJJOIN1;	NJ001840
				NJ001850
				NJ001860
				NJ001870

			2 ROW(50) BIT(32);	TAB00050
			/* STACK CONTAINING ENTITY NODE TAGS */	TAB00060
			1 TUPLE CTL EXT,	TAB00070
			2 D_ID FIXED BIN(15),	TAB00080
			2 NODE BIT(32);	TAB00090
3	1	0	DO WHILE(ALLOCATION(TUPLE)~=0);	TAB00100
			/* ADD ENTITY NODE TAGS FOR NSET 1 TO COL 1 */	TAB00110
4	1	1	DO WHILE(ALLOCATION(TUPLE)~=0&TUPLE.D_ID=1);	TAB00120
5	1	2	ROW_NUM(1)=ROW_NUM(1)+1;	TAB00130
6	1	2	TAB(1).ROW(ROW_NUM(1))=TUPLE.NODE;	TAB00140
7	1	2	FREE TUPLE;	TAB00150
8	1	2	END;	TAB00160
9	1	1	LAST_COL=1;	TAB00170
			/* FILL ROWS FOR REMAINING NSETS */	TAB00180
10	1	1	DO WHILE(ALLOCATION(TUPLE)~=0&TUPLE.D_ID~=1);	TAB00190
			/* FILL SO NUMBER OF ENTRIES SAME AS PREVIOUS	TAB00200
			COLUMN */	TAB00210
11	1	2	DO K=1 TO (ROW_NUM(LAST_COL)-ROW_NUM(TUPLE.D_ID));	TAB00220
12	1	3	ROW_NUM(TUPLE.D_ID)=ROW_NUM(TUPLE.D_ID)+1;	TAB00230
13	1	3	TAB(TUPLE.D_ID).ROW(ROW_NUM(TUPLE.D_ID))=	TAB00240
			TUPLE.NODE;	TAB00250
14	1	3	END;	TAB00260
15	1	2	LAST_COL=TUPLE.D_ID;	TAB00270
			/* POP TOP OF TUPLE STACK */	TAB00280
16	1	2	FREE TUPLE;	TAB00290
17	1	2	END;	TAB00300
18	1	1	END;	TAB00310
19	1	0	RETURN;	TAB00320
20	1	0	END TABN;	TAB00330
				TAB00340
				TAB00350
				TAB00360
				TAB00370

```

%INCLUDE NRECON;*****NRE00010
/*****FOR00010
*FOR00020
*FOR00030
*FOR00040
*FOR00050
*FOR00060
*FOR00070
*FOR00080
*FOR00090
*FOR00100
*FOR00110
*FOR00120
*FOR00130
*FOR00140
*FOR00150
*FOR00160
*FOR00170
*FOR00180
*FOR00190
*FOR00200
*FOR00210
*FOR00220
*FOR00230
*FOR00240
*FOR00250
*FOR00260
*FOR00270
*FOR00280
*FOR00290
*FOR00300
*FOR00310
*FOR00320
*FOR00330
*FOR00340
*FOR00350
*FOR00360
*FOR00370
*FOR00380
*FOR00390
*FOR00400
*FOR00410
*FOR00420
*FOR00430
*FOR00440
*FOR00450

1 0 NRECON: PROCEDURE
    (MODE, /* BIT(1) */
    ARG_NODE, /* 1,
                2 FIXED BIN(15),
                2 (50) BIT(32) */
    TEMP /* 1,
            2 FIXED BIN(15),
            2 (50) BIT(32) */ );

/*****PURPOSE:
*****THIS IS A SUPPORT MODULE FOR THE FETCH MODULE, AND IS
*****RESPONSIBLE FOR DETERMINING THE INTERSECTION OF THE NODES
*****CONTAINED IN ARG_NODE AND TEMP, AND FOR RETURNING THE
*****INTERSECTION IN ARG_NODE. THIS IS USED EXCLUSIVELY FOR
*****IMPLEMENTING RESTRICTIONS.
*****
*****METHOD:
*****BASICALLY THE MODULE COMPARES THE CONTENTS OF ARG_NODE
*****AND TEMP, AND MAINTAINS A TEMPORARY LIST ALL THE NODES
*****WHICH WERE IN BOTH ARG_NODE AND TEMP. AT THE END OF THE
*****ROUTINE, ARG_NODE IS UPDATED SO THAT IT ONLY CONTAINS
*****THE NODES THAT WERE IN BOTH. THE ONLY EXCEPTION TO THIS
*****IS IF MODE='0'B, WHICH INDICATES THAT THIS IS THE FIRST
*****RESTRICTION ON THE NSET, AND SO TEMP IS TO BE COPIED
*****DIRECTLY INTO ARG_NODE.
*****
*****INPUT PARAMETERS:
*****MODE - FLAG TO INDICATE IF THIS IS THE FIRST RESTRICTION.
*****        '0'B - YES
*****        '1'B - NO
*****ARG_NODE AND TEMP ARE EQUIVALENT TO ELEMENTS OF NLIST.
*****
*****OUTPUT PARAMETERS:
*****ARG_NODE (SEE ABOVE )
*****
*****CALLS PROCEDURES:
*****NONE
*****

```

```

*****
2 1 0      DCL 1 ARG_NODE,
              2 NUM FIXED BIN(15),
              2 NODES(*) BIT(32),

              1 WORK_NODE(2),
              2 NUM FIXED BIN(15) INIT((2) 0),
              2 NODES(50) BIT(32);

3 1 0      DCL 1 TEMP,
              2 NUM FIXED BIN(15),
              2 NODES(50) BIT(32);

4 1 0      DCL MODE BIT(1), FND BIT(1);

5 1 0      WORK_NODE(1)=TEMP;
6 1 0      IF ARG_NODE.NUM=0 & MODE='0'B
              THEN DO;
7 1 1          DO J =1 TO WORK_NODE(1).NUM;
8 1 2              ARG_NODE.NODES(J)=WORK_NODE(1).NODES(J);

9 1 2              END;
10 1 1          ARG_NODE.NUM=WORK_NODE(1).NUM;
11 1 1          RETURN;
12 1 1      END;
13 1 0      ELSE DO;
14 1 1          DO J=1 TO ARG_NODE.NUM;
15 1 2              FND='0'B;
16 1 2              DO JJ =1 TO WORK_NODE(1).NUM WHILE(~FND);
17 1 3                  IF ARG_NODE(J).NODES=WORK_NODE(1).NODES(JJ)
                      THEN FND='1'B;
18 1 3              END;
19 1 2              IF FND
                      THEN DO;
20 1 3                  WORK_NODE(2).NUM=WORK_NODE(2).NUM+1;
21 1 3                  WORK_NODE(2).NODES(WORK_NODE(2).NUM)=
                      ARG_NODE.NODES(J);
22 1 3                  END;
23 1 2              END;
24 1 1              ARG_NODE.NUM=WORK_NODE(2).NUM;
25 1 1              DO I= 1 TO ARG_NODE.NUM;
26 1 2                  ARG_NODE(I).NODES=WORK_NODE(2).NODES(I);
27 1 2              END;
28 1 1          END;
29 1 0          RETURN;
30 1 0      END NRECON;

```

```

NRE00010
NRE00020
NRE00030
NRE00040
NRE00050
NRE00060
NRE00070
NRE00080
NRE00090
NRE00100
NRE00110
NRE00120
NRE00130
NRE00140
NRE00150
NRE00160
NRE00170
NRE00180
NRE00190
NRE00200
NRE00210
NRE00220
NRE00230
NRE00240
NRE00250
NRE00260
NRE00270
NRE00280
NRE00290
NRE00300
NRE00310
NRE00320
NRE00330
NRE00340
NRE00350
NRE00360
NRE00370
NRE00380
NRE00390
NRE00400
NRE00410
NRE00420
NRE00430
NRE00440
NRE00450
NRE00460
NRE00470

```

```

%INCLUDE BUILD;*****BUI00010
/*****FOR00010
*FOR00020
*      MODULE      DESCRIPTION      *FOR00030
******/FOR00040
0 BUILD: PROCEDURE      FOR00050
      (NAME1,  /* BIT(64) */      FOR00060
      NCAT,    /* 1,              FOR00070
                  2 BIT(64),      FOR00080
                  2 BIT(8),       FOR00090
                  2 (20),         FOR00100
                  3 BIT(64),      FOR00110
                  3 BIT(8),       FOR00120
                  3 BIT(8),       FOR00130
                  3 BIT(64),      FOR00140
                  3 BIT(86) */ );  FOR00150
/*****FOR00160
*****PURPOSE:      FOR00170
*****THIS MODULE IS RESPONSIBLE FOR FETCHING THE NSET_CAT ENTRYFOR00180
*****FOR NSET NAME1, AND FOR RETURNING THE INFORMATION IN THE  FOR00190
*****NCAT STRUCTURE.  FOR00200
*****FOR00210
*****METHOD:      FOR00220
*****PLEASE SEE COMMENTS IN THE CODE. ITS FAIRLY      FOR00230
*****STRAIGHTFORWARD EXCEPT FOR THE USE OF ATTR_TEMP. SINCE  FOR00240
*****THE ATTR DESCRIPTION FOR AN ATTRIBUTE IS STORED AS A BIT  FOR00250
*****STRING, IT IS NECESSARY TO USE A STRING OVERLAY TO MAP   FOR00260
*****THE CONTENTS OF THE BIT STRING TO NCAT.ATTR, AND THAT IS  FOR00270
*****THE PURPOSE OF ATTR_TEMP.  FOR00280
*****FOR00290
*****FOR00300
*****INPUT PARAMETERS:  FOR00310
*****NAME1 - NAME OF A PREVIOUSLY DEFINED NSET.  FOR00320
*****FOR00330
*****OUTPUT PARAMETERS:  FOR00340
*****NCAT - SEE OTHER DESCRIPTIONS OF NCAT.  FOR00350
*****FOR00360
*****CALLS PROCEDURES:  FOR00370
*****SELECTF,SEARCH  FOR00380
******/FOR00390
*****BUI00010
*****BUI00020
%INCLUDE NCAT;*****BUI00030
2 1 0 DCL 1 NCAT, /* NSET CATALOGUE ENTRY */ DCL00010
      2 NNAME BIT(64), /* NAME OF NSET */ DCL00020
      2 NATTR BIT(8), /* NUMBER OF ATTRIBUTES */DCL00030

```

```

                2 ATTR(20),                /* UP TO 20 ATTRIBUTES */ DCL00040
                3 ANAME BIT(64),           /* ATTRIBUTE NAME */ DCL00050
                3 K_TYPE,                   /* UNIQUE KEY OR NOT */ DCL00060
                3 BREL) BIT(8),            /* TYPE OF BSET */ DCL00070
                3 BSETUP,                   /* BSET(ATTR->N_NODE) */ DCL00080
                3 BSETDOWN) BIT(64);       /* BSET(N_NODE->ATTR) */ DCL00090
*****
                /* OVERLAID ON INFO_ND TO EXTRACT ATTRIBUTE DESCRIPTION */
3 1 0 DCL 1 A TR_TEMP DEFINED ATTR_STR,
                2 ANAME BIT(64),
                (2 K_TYPE,
                2 BREL) BIT(8),
                (2 BSETUP,
                2 BSETDOWN) BIT(64),
                ATTR_STR BIT(208);
                %INCLUDE IDS1;*****
                /* POINTER STACK RETURNED BY SEARCH */
4 1 0 DCL IDS1 PTR EXTERNAL CONTROLLED;
                *****
                %INCLUDE INFO_ND;*****
                /* DATA STACK RETURNED BY FETCH */
5 1 0 DCL INFO_ND BIT(320) EXTERNAL CONTROLLED;
                *****
                /* MISC DECLARATIONS */
6 1 0 DCL (IDXX,ID,ID1) POINTER INIT(NULL()),
                NAME1 BIT(64);
7 1 0 DCL IDN_DE BIT(160);
8 1 0 DCL (N_NAME, NSETCAT,N_ATTR,NSETB1,NSETB2) BIT(64) STATIC
                EXTERNAL, L FIXED BIN(8);
                /* PROCEDURES CALLED */
                %INCLUDE ESELECT;*****
                /* BSET RETRIEVAL MODULE */
9 1 0 DCL SELECTF ENTRY(BIT(2),POINTER,BIT(64),BIT(320),POINTER);
                *****
                %INCLUDE ESEARCH;*****
                /* SEARCH MODULE */
10 1 0 DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER);
                *****
                /* ESTABLISH INSTANCE OF THE NSET NAME IN ATTRIBUTE N_NAME */
11 1 0 CALL SEARCH('01'B,N_NAME,NAME1,IDXX,ID);
12 1 0 ID1=IDS1;
13 1 0 ID=NULL();
14 1 0 FREE ID1;

```

BUI00030
 BUI00040
 BUI00050
 BUI00060
 BUI00070
 BUI00080
 BUI00090
 BUI00100
 BUI00110
 BUI00120
 BUI00130
 BUI00140
 BCA00420
 BCA00430
 BUI00140
 BUI00150
 BCA00450
 BCA00460
 BUI00150
 BUI00160
 BUI00170
 BUI00180
 BUI00190
 BUI00200
 BUI00210
 BUI00220
 BUI00230
 BUI00240
 DEC00050
 DEC00060
 BUI00240
 BUI00250
 BCA00700
 BCA00710
 BCA00720
 BUI00250
 BUI00260
 BUI00270
 BUI00280
 BUI00290
 BUI00300
 BUI00310
 BUI00320

			/* GET THE ASSOCIATED ENTITY NODE */	BUI00330
15	1	0	CALL SELECTF('11'B,ID1,NSETB1,'0'B,IDXX);	BUI00340
16	1	0	IDNODE=INFO_ND;	BUI00350
17	1	0	FREE INFO_ND;	BUI00360
			/* GET THE ASSOCIATED INSTANCES OF ITS ATTRIBUTE DESCRIPTIONS*/	BUI00370
18	1	0	CALL SELECTF('11'B,ID,NSETB2,IDNODE, ID1);	BUI00380
19	1	0	L=ALLOCATION(INFO_ND);	BUI00390
			/* BUILD NCAT */	BUI00400
20	1	0	NCAT.NATTR= BIN(L);	BUI00410
21	1	0	NCAT.NNAME=NAME1;	BUI00420
22	1	0	DO J=1 TO NCAT.NATTR;	BUI00430
			/* MAP ATTR_STR ONTO INFO_ND */	BUI00440
23	1	1	ATTR_STR=INFO_ND;	BUI00450
24	1	1	NCAT(J).ATTR=ATTR_TEMP;	BUI00460
25	1	1	FREE INFO_ND;	BUI00470
26	1	1	END;	BUI00480
27	1	0	RETURN;	BUI00490
28	1	0	END BUILD;	BUI00500
				BUI00510
				BUI00520
				BUI00530


```

%INCLUDE EDEFINB;*****NIN00140
/* DEFINE BSET MODULE */ DEC00020
4 1 0 DCL DEFINES ENTRY(BIT(64),BIT(64),BIT(64),BIT(8),BIT(1)); DEC00030
***** NIN00140
%INCLUDE ECREATB;*****NIN00150
/* CREATE BSET MODULE */ DEC00050
5 1 0 DCL CREATEB ENTRY(BIT(64),POINTER,BIT(320),BIT(320),POINTER); DEC00060
***** NIN00150
%INCLUDE ECREATP;*****NIN00160
/* CREATE PSET MODULE */ BCA00630
6 1 0 DCL CREATEP ENTRY(BIT(64),BIT(320),POINTER); BCA00640
***** NIN00160
NIN00170
/* START OF PROCEDURE - INITIALIZE VARIABLES */ NIN00180
7 1 0 UNIQUE='00000001'B; NIN00190
8 1 0 N_NAME=UNSPEC('NSETNAME'); NIN00200
9 1 0 NSETCAT=UNSPEC('NSETCAT '); NIN00210
10 1 0 N_ATTR=UNSPEC('NSETATTR'); NIN00220
11 1 0 NSETB1=UNSPEC('NSETB1 '); NIN00230
12 1 0 NSETB2=UNSPEC('NSETB2 '); NIN00240
NIN00250
/* DEFINE THE NSETNAME PSET FOR THE NSETCAT */ NIN00260
13 1 0 CALL DEFINEP(N_NAME,'00000001'B,'00000001'B,'00100000'B, NIN00270
'000000000010100'B,'0'B,'0'B, PTR); NIN00280
NIN00290
/* DEFINE THE ENTITY NODE PSET FOR THE NSET_CAT */ NIN00300
14 1 0 CALL DEFINEP(NSETCAT,'00000001'B,'00000001'B,'00100000'B, NIN00310
'0000000000100000'B,'0'B,'0'B, PTR); NIN00320
NIN00330
/* DEFINE THE ATTRIBUTE DESCRIPTION PSET FOR THE NSET_CAT */ NIN00340
15 1 0 CALL DEFINEP(N_ATTR,'00000001'B,'00000001'B,'00100000'B, NIN00350
'0000000001101000'B,'0'B,'0'B, PTR); NIN00360
NIN00370
/* DEFINE THE NSETNAME-NSETCAT ENTITY NODE BSET */ NIN00380
16 1 0 CALL DEFINEB(NSETB1,N_NAME,NSETCAT,'00000001'B,'0'B); NIN00390
NIN00400
/* NOW DEFINE ITS RECIPROCAL */ NIN00410
17 1 0 TEMP=NSETB1; NIN00420
18 1 0 CALL DEFINEB(TEMP,NSETCAT,N_NAME,'00000001'B,'1'B); NIN00430
/* DEFINE THE ENTITY NODE - ATTR DESCRIPTION BSET */ NIN00440
19 1 0 CALL DEFINEB(NSETB2,NSETCAT,N_ATTR,'00000010'B,'0'B); NIN00450
NIN00460
/* NOW INSERT THE NSET_CAT CAT ENTRY FOR ITSELF INTO THE NIN00470
NSET_CAT, FIRST CREATE AN INSTANCE OF THE NSET NAME */ NIN00480
20 1 0 CALL CREATEP(N_NAME,NSETCAT,PTR); NIN00490
NIN00500
/* CREATE AN ASSOCIATED INSTANCE OF THE ENTITY NODE */ NIN00510
21 1 0 CALL CREATEB(NSETB1,PTR,'0'B,'0'B,PTR2); NIN00520
NIN00530

```


22	1	0	/* CREATE ATTR DESCRIPTION FOR N_NAME */	NIN00540
			ANAME=N_NAME;	NIN00550
23	1	0	K_TYPE='00000001'B;	NIN00560
24	1	0	BREL='00000001'B;	NIN00570
25	1	0	BSETUP=NSETB1;	NIN00580
26	1	0	BSETDOWN=TEMP;	NIN00590
			/* CREATE AN ASSOCIATED INSTANCE OF THE ATTR DESCRIPTION*/	NIN00600
27	1	0	CALL CREATEB(NSETB2, PTR2, '0'B, ATTR_STR, PTR);	NIN00610
				NIN00620
			/* CREATE ATTR DESCRIPTION FOR N_ATTR */	NIN00630
28	1	0	ANAME=N_ATTR;	NIN00640
29	1	0	K_TYPE='00000000'B;	NIN00650
30	1	0	BREL='00000010'B;	NIN00660
31	1	0	BSETDOWN=NSETB2;	NIN00670
			/* CREATE AN ASSOCIATED INSTANCE OF THE N_ATTR DESCRIPT */	NIN00680
32	1	0	CALL CREATEB(NSETB2, PTR2, '0'B, ATTR_STR, PTR);	NIN00690
				NIN00700
33	1	0	RETURN;	NIN00710
34	1	0	END NINIT;	NIN00720

```

%INCLUDE DEFINEV;*****DV 00010
/*****
*
*          MODULE      DESCRIPTION
*          *****/
1  0  DEFINEV: PROCEDURE
      ( DV_ARG      /* 1,
          2 BIT(64),
          2 BIT(8)   */ );
/*****
*****  PURPOSE:
*****  THIS MODULE ACTS AS THE NSET INTERFACE TO THE PSET MODULE
*****  DEFINEV. IT IS USED PRINCIPALLY TO DEFINE THE UNDERLYING
*****  PSETS FOR USER DEFINED DOMAINS. NOTE THE EXTERNAL LEVEL
*****  IS RESPONSIBLE FOR CHECKING WHETHER A DUPLICATE DOMAIN IS
*****  BEING DEFINED.
*****
*****
*****  METHOD:
*****  THE MODULE IS RATHER TRIVIAL. IT IS PASSED VIA DV_ARG
*****  THE NAME OF THE PSET TO BE CREATED AND THE LENGTH OF
*****  THE KEY. THE MODULE THEN USES DEFAULT PARAMETERS CON-
*****  TAINED IN SYS_DEFAULT TO SET UP THE CALL TO DEFINEV
*****  WHICH IS THE MODULE RESPONSIBLE FOR DEFINING THE UNDER-
*****  LYING PSET. AT THE CURRENT TIME THE SYSTEM DEFAULTS ARE
*****  LINK TYPE - HASHED
*****  LENGTH - 320 BITS
*****  KEY POSITION - STARTING ON FIRST BIT OF DATA AREA
*****  SUBSET - NO
*****  S_ID - 0
*****  ID - NOT RELEVANT
*****
*****
*****  INPUT PARAMETERS:
*****  1 DV_ARG      STRUCTURE TO DEFINE A DOMAIN
*****  2 NAME        NAME OF DOMAIN/PSET
*****  2 KEY_LEN     LENGTH OF KEY (MAXIMUM 32 CHAR)
*****
*****
*****  OUTPUT PARAMETERS:
*****  NONE
*****
*****
*****  CALLS PROCEDURES:
*****  DEFINEV
*****

```

FOR00010
 FOR00020
 FOR00030
 FOR00040
 FOR00050
 FOR00060
 FOR00070
 FOR00080
 FOR00090
 FOR00100
 FOR00110
 FOR00120
 FOR00130
 FOR00140
 FOR00150
 FOR00160
 FOR00170
 FOR00180
 FOR00190
 FOR00200
 FOR00210
 FOR00220
 FOR00230
 FOR00240
 FOR00250
 FOR00260
 FOR00270
 FOR00280
 FOR00290
 FOR00300
 FOR00310
 FOR00320
 FOR00330
 FOR00340
 FOR00350
 FOR00360
 FOR00370
 FOR00380
 FOR00390
 FOR00400
 FOR00410
 FOR00420
 FOR00430
 FOR00440
 FOR00450

```

.....*/ FOR00460
.....DV 00010
%INCLUDE DVARG;.....DV 00020
/* STRUCTURE USED TO PASS NAME OF VALUE SET TO BE DEFINED */DVA00010
2 1 0 DCL 1 DV_ARG, DVA00020
      2 NAME BIT(64), /* NAME OF VALUE SET */DVA00030
      2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */DVA00040
.....DV 00020
/* DEFAULTS USED TO DEFINE PSETS */DV 00030
3 1 0 DCL 1 SYS_DEFAULT, DV 00040
      2 LINK BIT(8) INIT('00000001'B), DV 00050
      2 LEN BIT(16) INIT('0000000010000000'B), DV 00060
      2 KEY_POS BIT(8) INIT('00000001'B), DV 00070
      2 SUBSET BIT(8) INIT('00000000'B), DV 00080
      2 S_ID BIT(8) INIT('00000000'B), DV 00090
      2 ID PTR; DV 00100
DV 00110
      /* PROCEDURE CALLED */DV 00120
%INCLUDE EDEFINP;.....DV 00130
      /* DEFINE PSET MODULE */BCA00590
4 1 0 DCL DEFINEP ENTRY(BIT(64),BIT(8),BIT(8),BIT(8),BIT(8),BCA00600
      BIT(8),BIT(8),POINTER) ; BCA00610
.....DV 00130
      /* CALL DEFINEP MODULE */DV 00140
5 1 0 CALL DEFINEP (NAME, LINK, KEY_POS, KEY_LEN, LEN, SUBSET, S_ID, DV 00150
      ID); DV 00160
6 1 0 RETURN; DV 00170
7 1 0 END DEFINEV; DV 00180

```

```

%INCLUDE FETCHV;*****FET00010
/*****
*
*          MODULE      DESCRIPTION
*          *****/
1  0  FETCHV: PROCEDURE
      (FV_ARG      /* 1,
                        2 BIT(64),
                        2 BIT(160),
                        2 FOUND BIT(1),
                        2 DATA BIT(320) */ );
/*****
*****  PURPOSE:
*****  THIS MODULE IS RESPONSIBLE FOR RETRIEVING A SINGLE INSTANCE
*****  OF A DOMAIN. IT IS USED PRIMARILY BY THE EXTERNAL LEVEL TO
*****  CHECK IF AN ELEMENT EXISTS. SHIELDS EXTERNAL LEVEL FROM THE
*****  INTERNAL LEVEL.
*****
*****  METHOD:
*****  VERY SIMPLE, IT CALLS FETCH, PASSING IT THE ARGUMENTS REQUIRED
*****  AND RETURNS THE ELEMENT FOUND, IF IT INDEED EXISTS.
*****
*****  INPUT PARAMETERS:
*****  1 FV_ARG,
*****  2 D_NAME - THE NAME OF THE DOMAIN TO BE SEARCHED
*****  2 KEY_VAL - VALUE ON WHICH TO SEARCH
*****  2 FOUND - FLAG TO INDICATE IF ELEMENT WAS FOUND, NOT USED
*****              ON INPUT.
*****  2 DATA - BIT STRING REPRESENTATION OF ELEMENT, NOT USED
*****              INPUT.
*****
*****  OUTPUT PARAMETERS:
*****  SEE ABOVE
*****
*****  CALLS PROCEDURES:
*****  FETCH
*****
*****
*****
%INCLUDE FVARG;*****FET00020
/* FETCHV TABLE -USED TO RETRIEVE INSTANCES OF A DOMAIN */
DCL 1 FV_ARG,
      2 D_NAME BIT(64), /* NAME OF DOMAIN */
      2 KEY_VAL BIT(160), /* KEY TO SEARCH ON */
      2 FOUND BIT(1), /* IF FOUND, '1'B, OTHERWISE '0'B */

```

```

FOR00010
FOR00020
FOR00030
FOR00040
FOR00050
FOR00060
FOR00070
FOR00080
FOR00090
FOR00100
FOR00110
FOR00120
FOR00130
FOR00140
FOR00150
FOR00160
FOR00170
FOR00180
FOR00190
FOR00200
FOR00210
FOR00220
FOR00230
FOR00240
FOR00250
FOR00260
FOR00270
FOR00280
FOR00290
FOR00300
FOR00310
FOR00320
FOR00330
FOR00340
FOR00350
FOR00360
FOR00370
FOR00380
FET00010
FET00020
DEF00010
DEF00020
DEF00030
DEF00040
DEF00050

```

2 1 0

		2 DATA BIT(320);	/* RETRIEVED ELEMENT */	DEF00060
		*****		DEF00070
		/* MISC DECLARATIONS */		FET00020
3	1	0	DCL ID PTR INIT(NULL());	FET00030
			FND BIT(1) INIT('0');	FET00040
		%INCLUDE INFOIND;*****		FET00050
		/* DATA STACK RETURNED BY FETCH */		FET00060
4	1	0	DCL INFO_ND BIT(320) EXTERNAL CONTROLLED;	BCA00450
		*****		BCA00460
		/* PROCEDURES CALLED */		FET00060
		%INCLUDE EFETCH;*****		FET00070
		/* FETCH PSET MODULE */		FET00080
5	1	0	DCL FETCH ENTRY(BIT(2),POINTER,BIT(64),BIT(64),BIT(1));	EFE00010
		*****		EFE00020
				FET00080
		/* CALL FETCH TO RETRIEVE ELEMENT */		FET00090
6	1	0	CALL FETCH('01'B,ID,D_NAME,KEY_VAL,FND);	FET00100
7	1	0	IF ALLOCATION(INFO_ND)>0	FET00110
			THEN DO;	FET00120
8	1	1	DATA = INFO_ND;	FET00130
9	1	1	FREE INFO_ND;	FET00140
10	1	1	FOUND = '1'B;	FET00150
11	1	1	END;	FET00160
12	1	0	ELSE FOUND = '0'B;	FET00170
13	1	0	RETURN;	FET00180
14	1	0	END FETCHV;	FET00190
				FET00200

```

%INCLUDE DEFINE;*****DB 00010
/*****
*          MODULE DESCRIPTION          *
*****/
1 0 DEFINE:  PROCEDURE
              ( SET_NAME1, /* BIT(64) */
                DOMAIN1,   /* BIT(64) */
                DOMAIN2,   /* BIT(64) */
                TYPE1,     /* BIT(8) */
                EQUIV      /* BIT(1) */);

/*****
*****  PURPOSE:
*****  THIS MODULE IS RESPONSIBLE FOR DEFINING A BINARY
*****  ASSOCIATION BETWEEN DOMAIN1 AND DOMAIN2. THIS MEANS
*****  THAT IT IS RESPONSIBLE FOR ASSIGNING POINTER SLOTS
*****  WITHIN A PSET'S POINTER ARRAY TO PARTICULAR BINARY
*****  ASSOCIATIONS. IN ADDITION, IF SUBSETS ARE REQUIRED
*****  IT ALLOCATES THE NECESSARY POINTER SLOTS. FINALLY IT
*****  IS RESPONSIBLE FOR UPDATING A PSET'S P_CAT ENTRY TO
*****  REFLECT ANY CHANGES MADE, AS WELL AS CREATE A BSET_CAT
*****  ENTRY WHICH CONTAINS ALL INFORMATION NECESSARY TO
*****  CONSTRUCT A BINARY ASSOCIATION BETWEEN THE 2 DOMAINS.
*****  IT IS CURRENTLY CAPABLE OF IMPLEMENTING 1-1, 1-N, N-1, AND
*****  M-N RELATIONSHIPS. IT IS ALSO CURRENTLY CAPABLE OF
*****  DEFINING RECIPROCAL RELATIONSHIPS (I.E. IF A 1-N
*****  RELATIONSHIP HAS BEEN DEFINED BETWEEN DOMAIN1 AND DOMAIN2
*****  A N-1 RELATIONSHIP CAN BE DEFINED BETWEEN DOMAIN2 AND
*****  DOMAIN1 WITHOUT ALLOCATING ANY ADDITIONAL POINTER SLOTS.
*****
*****  METHOD:
*****  1) IF THIS IS THE FIRST BSET TO HAVE BEEN DEFINED IT
*****     IS NECESSARY TO CREATE BSET_CAT WHICH IS A CATALOGUE
*****     CONTAINING THE NAME OF EVERY BSET DEFINED AND
*****     IMPLEMENTATION INFORMATION. BSET_CAT IS ITSELF IMPLEMENTED
*****     AS A PSET. HENCE, IT IS NECESSARY TO CALL DEFINE IN
*****     ORDER TO DEFINE THE PSET.
*****  2) IF EQUIV='1' IT MEANS THAT THE BSET TO BE DEFINED IS
*****     THE RECIPROCAL OF A PREVIOUSLY DEFINED BSET, AND HENCE
*****     NO NEW POINTER SLOTS NEED BE ALLOCATED. THE CATALOGUE
*****     ENTRY CORRESPONDING TO THE PREVIOUSLY DEFINED BSET
*****     (SETNAME1) IS FETCHED FROM BSET_CAT VIA A CALL TO FETCH.
*****     THE INFORMATION CONTAINED IN BSET_CAT IS USED TO CREATE
*****     A RECIPROCAL BSET. NAMEGEN IS CALLED TO CREATE A NAME
*****     FOR THE NEW BSET, AND THE NEW BSET IS INSERTED INTO THE
*****     BSET_CAT PSET VIA A CALL TO CREATEP. IT THEN RETURNS TO

```



```

4 L_POS2,      /* ADDITIONAL PTR SLOT FOR LINK */BCA00250
4 KEY_POS,     /* STARTING POSITION OF KEY */    BCA00260
4 KEY_LEN ) BIT(8), /* LENGTH OF KEY */        BCA00270
3 SET_TYPE,    /* SET TYPE INFO */            BCA00280
( 4 SUBSET,    /* IF PRIMARY OR SUBSET */      BCA00290
  4 SUBSET_ID, /* PTR SLOT FOR SUBSET LINK */  BCA00300
  4 P_CHAIN,   /* PTR SLOT PTS TO PRIMARY DCL*/BCA00310
  4 S_CHAIN ) BIT(8), /* SUBSET DCL CHAIN */    BCA00320
3 DATA_LEN BIT(15); /* LENGTH OF ELEMENTS */  BCA00330
*****
DB 00100
DB 00110
5 1 0      DCL (P_CAT,B_CAT) BIT(64) STATIC EXTERNAL; DB 00120
6 1 0      DCL (A1_TO_1,A1_TO_N,N_TO_1,M_TO_N) BIT(8) STATIC EXT; DB 00130
*****
DB 00140
%INCLUDE IDS1;*****DB 00150
/* POINTER STACK RETURNED BY SEARCH */          BCA00420
7 1 0      DCL IDS1 PTR EXTERNAL CONTROLLED;      BCA00430
*****
DB 00150
DB 00160
%INCLUDE INFO_ND;*****DB 00170
/* DATA STACK RETURNED BY FETCH */              BCA00450
8 1 0      DCL INFO_ND BIT(320) EXTERNAL CONTROLLED; BCA00460
*****
DB 00170
DB 00180
/* PROCEDURES CALLED */                          DB 00190
%INCLUDE EDEFINP;*****DB 00200
/* DEFINE PSET MODULE */                          BCA00590
9 1 0      DCL DEFINEP ENTRY(BIT(64),BIT(8),BIT(8),BIT(8), BCA00600
              BIT(8),BIT(8),POINTER) ;            BCA00610
*****
DB 00200
%INCLUDE ECREATEP;*****DB 00210
/* CREATE PSET MODULE */                          BCA00630
10 1 0     DCL CREATEP ENTRY(BIT(64),BIT(320),POINTER); BCA00640
*****
DB 00210
%INCLUDE EFETCH;*****DB 00220
/* FETCH PSET MODULE */                          EFC00010
11 1 0     DCL FETCH ENTRY(BIT(2),POINTER,BIT(64),BIT(64),BIT(1)); EFL00020
*****
DB 00220
%INCLUDE ESEARCH;*****DB 00230
/* SEARCH MODULE */                              BCA00700
12 1 0     DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER); BCA00710
*****
DB 00230
DB 00240
%INCLUDE EMAPSET;*****DB 00240
/* MAP MAINTENANCE MODULE */                    ECR00050
13 1 0     DCL MAPSET ENTRY(BIT(1),FIXED BIN(8),BIT(16),BIT(16),BIT(8)); ECR00060
*****
DB 00240
%INCLUDE ENAMEGN;*****DB 00250
/* RANDOM NAME GENERATOR */                    ECR00080

```

14	1	0	DCL NAMEGEN ENTRY(FIXED BIN(15)) RETURNS(BIT(64));	ECR00090
			*****	DB 00250
			/* MISC DECLARATIONS */	DB 00260
15	1	0	DCL (ID1,ID2,ID_CAT(2)) POINTER INIT(NULL()),	DB 00270
			IDX PTR CTL,	DB 00280
			/* TEMPORARY STRUCTURE FOR EQUIVALENCE OPERATION */	DB 00290
			1 TEMP_INFO,	DB 00300
			2 NAME BIT(64),	DB 00310
			2 AP_POS1 BIT(8);	DB 00320
				DB 00330
				DB 00340
				DB 00350
			/* IF THIS IS THE FIRST CALL TO DEFINEB, DEFINE THE BSETCAT	DB 00360
			PSET */	DB 00370
16	1	0	IF ~BSET_FLAG THEN	DB 00380
			DO;	DB 00390
17	1	1	B_CAT=UNSPEC('BSET_CAT');	DB 00400
18	1	1	CALL DEFINEP(B_CAT,'00000001'B,'00000001'B,	DB 00410
			'01000000'B,'0000000100010111'B,	DB 00420
			'000000000'B,'00000000'B,ID1);	DB 00430
19	1	1	BSET_FLAG='1'B;	DB 00440
20	1	1	A1_TO_1='00000001'B;	DB 00450
21	1	1	A1_TO_N='00000010'B;	DB 00460
22	1	1	N_TO_1='00000100'B;	DB 00470
23	1	1	M_TO_N='00001000'B;	DB 00480
24	1	1	END;	DB 00490
				DB 00500
			/* IF EQUIV FLAG SET, MEANS BSET TO BE DEFINED IS A RECIPROCAL	DB 00510
			OF THE BSET SET_NAME1 */	DB 00520
25	1	0	IF EQUIV	DB 00530
			THEN DO;	DB 00540
			/* GET BSETCAT ENTRY FOR SET_NAME1 TO USE AS A	DB 00550
			TEMPLATE */	DB 00560
26	1	1	CALL FETCH('01'B,ID1,B_CAT,SET_NAME1,'0'B);	DB 00570
27	1	1	BASE=INFO_ND;	DB 00580
28	1	1	FREE INFO_ND;	DB 00590
				DB 00600
			/* MODIFY DOMAIN_INFO TO REFLECT RECIPROCAL */	DB 00610
29	1	1	TEMP_INFO=DOMAIN_INFO(1);	DB 00620
30	1	1	DOMAIN_INFO(1)=DOMAIN_INFO(2);	DB 00630
31	1	1	DOMAIN_INFO(2)=TEMP_INFO;	DB 00640
				DB 00650
			/* GENERATE NAME FOR BSET */	DB 00660
32	1	1	SET_NAME1=NAMEGEN(8);	DB 00670
33	1	1	SET_NAME=SET_NAME1;	DB 00680
34	1	1	TYPE=TYPE1;	DB 00690
				DB 00700
			/* INSERT NEW CAT ENTRY INTO BSETCAT PSET */	DB 00710
35	1	1	CALL CREATEP(B_CAT,BASE,ID1);	DB 00720

36	1	1	RETURN;	DB 00730
37	1	1	END;	DB 00740
			/* OTHERWISE */	DB 00750
			/* BUILD CAT ENTRY FOR NEW BSET */	DB 00760
38	1	0	SET_NAME=SET_NAME1;	DB 00770
39	1	0	DOMAIN_INFO.NAME(1)=DOMAIN1;	DB 00780
40	1	0	DOMAIN_INFO.NAME(2)=DOMAIN2;	DB 00790
41	1	0	TYPE=TYPE1;	DB 00800
			/* VERIFY EXISTENCE OF EACH DOMAIN, AND ALLOCATE PTR SLOT */	DB 00810
42	1	0	DO J=1 TO 2;	DB 00820
43	1	1	CALL SEARCH('01'B,P_CAT,DOMAIN_INFO.NAME(J),IDX,ID1);	DB 00830
44	1	1	IF ALLOCATION(IDS1)='0'B THEN	DB 00840
			DO;	DB 00850
45	1	2	PUT SKIP EDIT('DOMAIN',J,'DOESNT EXIST ,ERROR')	DB 00860
			(A,F(7),A);	DB 00870
46	1	2	SUCCESS='0'B;	DB 00880
47	1	2	RETURN;	DB 00890
48	1	2	END;	DB 00900
			/* ID_CAT(J) PTS TO PSET CAT ENTRY FOR DOMAIN */	DB 00910
49	1	1	ID_CAT(J)=IDS1;	DB 00920
50	1	1	FREE IDS1;	DB 00930
			/* ALLOCATE A FREE POINTER SLOT TO BE USED FOR	DB 00940
			IMPLEMENTING BSET, UPDATE PSETCAT TO REFLECT SLOT	DB 00950
			ALLOCATED. IF NONE AVAILABLE, PRINT ERROR */	DB 00960
51	1	1	CALL MAPSET('1'B,2,ID_CAT(J)->SP_MAP,ID_CAT(J)->AP_MAP,	DB 00970
			FREE_SLOT);	DB 00980
52	1	1	IF FREE_SLOT='00000000'B THEN	DB 00990
			DO;	DB 01000
53	1	2	PUT SKIP EDIT('NO MORE FREE SLOTS IN DOMAIN',J)	DB 01010
			(A,F(7));	DB 01020
54	1	2	SUCCESS=FREE_SLOT;	DB 01030
55	1	2	RETURN;	DB 01040
56	1	2	END;	DB 01050
			/* PLACE POSITION OF ALLOCATED POINTER SLOT IN BSETCAT	DB 01060
			ENTRY FOR BSET */	DB 01070
57	1	1	AP_POS(J)=FREE_SLOT;	DB 01080
58	1	1	END;	DB 01090
59	1	0	IF TYPE=M_TO_N THEN	DB 01100
			DO;	DB 01110
			/* IF M TO N, NECESSARY TO DEFINE A LINK SET TO BE	DB 01120
			USED TO IMPLEMENT BSET */	DB 01130
60	1	1	MN_NAME=NAMEGEN(8);	DB 01140
61	1	1	CALL DEFINEP(MN_NAME,'00000100'B,'00000001'B,	DB 01150
			'00010000'B,'00010000'B,'0'B,'0'B,ID_CAT(2));	DB 01160
				DB 01170
				DB 01180
				DB 01190
				DB 01200
				DB 01210

			/* MODIFY ALLOCATION OF PTR SLOTS IN LINK SET SO THAT	DB 01220
			SAME PTR SLOTS CAN BE USED AS SPECIFIED FOR EACH	DB 01230
			OF THE DOMAINS */	DB 01240
62	1	1	ID_CAT(2)->L_POS2='00001111'B;	DB 01250
63	1	1	SUBSTR(ID_CAT(2)->SP_MAP,AP_POS(1),1)='0'B;	DB 01260
64	1	1	SUBSTR(ID_CAT(2)->SP_MAP,AP_POS(2),1)='0'B;	DB 01270
65	1	1	ID_CAT(2)->AP_MAP=ID_CAT(2)->SP_MAP;	DB 01280
66	1	1	CALL MAPSET('1'B,1,ID_CAT(2)->SP_MAP,ID_CAT(2)->	DB 01290
			AP_MAP,SUB_ID);	DB 01300
67	1	1	END;	DB 01310
68	1	0	ELSE IF TYPE=A1_TO_N;TYPE=N_TO_1 THEN	DB 01320
			DO;	DB 01330
			/* IF EITHER 1 TO N OR N TO 1, IT IS NECESSARY TO	DB 01340
			ALLOCATE A POINTER SLOT TO BE USED TO CHAIN	DB 01350
			ELEMENTS IN A SUBSET TOGETHER */	DB 01360
69	1	1	IF TYPE=A1_TO_N THEN	DB 01370
			K=2;	DB 01380
70	1	1	ELSE K=1;	DB 01390
			/* FINDS FREE PTR SLOT AND RESERVES IT BY UPDATING	DB 01400
			PSETCAT ENTRY FOR DOMAIN TO REFLECT ALLOCATION */	DB 01410
71	1	1	CALL MAPSET('1'B,1,ID_CAT(K)->SP_MAP,ID_CAT(K)->	DB 01420
			AP_MAP,SUB_ID);	DB 01430
72	1	1	IF SUB_ID='0'B THEN	DB 01440
			DO;	DB 01450
73	1	2	PUT SKIP EDIT('NO MORE AVAILABLE SLOTS IN ',K)	DB 01460
			(A,F(7));	DB 01470
74	1	2	SUCCESS='0'B;	DB 01480
75	1	2	RETURN;	DB 01490
76	1	2	END;	DB 01500
77	1	1	END;	DB 01510
			/* INSERT ENTRY NTO THE PSETCAT PSET */	DB 01520
78	1	0	CALL CREATEP(B_CAT,BASE,101);	DB 01530
79	1	0	RETURN;	DB 01540
80	1	0	END DEFINE;	DB 01550
				DB 01560
				DB 01570

```

XINCLUDE CREATEB;*****CB 00010
/*****FOR00010
*      MODULE DESCRIPTION      *FOR00020
*****/FOR00030
1 0 CREATEB:      PROCEDURE      FOR00040
      (B_SET, /* BIT(64) */      FOR00050
       ID1,  /* POINTER */      FOR00060
       DATA1, /* BIT(320)*/      FOR00070
       DATA2, /* BIT(320)*/      FOR00080
       ID2   /* POINTER */);      FOR00090
/*****FOR00100
*****PURPOSE:      FOR00110
*****THIS MODULE IS RESPONSIBLE FOR CREATING A BINARY      FOR00120
*****ASSOCIATION BETWEEN DATA1 AND DATA2 BASED ON IMPLEMENTATION      FOR00130
*****INFORMATION CONTAINED IN THE BSET CATALOGUE FOR THE      FOR00140
*****BINARY SET IDENTIFIED BY B_SET. THE BINARY SET MUST      FOR00150
*****HAVE BEEN PREVIOUSLY DEFINED USING THE DEFINEB PROCEDURE.      FOR00160
*****AS CURRENTLY IMPLEMENTED AN OCCURENCE OF DATA1 MUST      FOR00170
*****ALREADY EXIST IN DOMAIN1. HOWEVER, IF DATA2 DOES NOT EXIST      FOR00180
*****IT WILL BE CREATED.      FOR00190
*****      FOR00200
*****      FOR00210
*****      FOR00220
*****METHOD:      FOR00230
*****1) THE BSET_CAT ENTRY FOR B_SET IS FIRST RETRIEVED FROM      FOR00240
*****   BSET_CAT USING THE FETCH PROCEDURE. THIS IN EFFECT      FOR00250
*****   PLACES THE BSET_CAT INFORMATION FOR THIS BSET INTO      FOR00260
*****   THE BSET_CAT STRUCTURE WITHIN THE PROCEDURE.      FOR00270
*****2) THE USER HAS THE CHOICE OF SPECIFYING EITHER THE ACTUAL      FOR00280
*****   DATA VALUE IN THE FIRST DOMAIN OR BY SPECIFYING A POINT      FOR00290
*****   FORMER, THEN THE SEARCH PROCEDURE IS INVOKED TO RETURN      FOR00300
*****   A POINTER TO THE OCCURENCE OF DATA1 IN DOMAIN1.      FOR00310
*****   IF NO VALUE IS FOUND WHICH MATCHES DATA1 THEN AN ERROR      FOR00320
*****   MESSAGE IS PRINTED AND THE PROCEDURE RETURNS.      FOR00330
*****3) IF THE BINARY ASSOCIATION IS 1 TO 1 THEN AN INTERNAL      FOR00340
*****   PROCEDURE CALLED GET_DOMAIN2 IS CALLED WHICH RETURNS      FOR00350
*****   A POINTER WHICH POINTS TO AN OCCURENCE OF DATA2 IN      FOR00360
*****   DOMAIN2. GET_DOMAIN2 EITHER FOUND AN EXISTING OCCURENCE      FOR00370
*****   OF DATA2 WHICH WAS ELIGIBLE (I.E. ITS AP_POS(2) WAS      FOR00380
*****   NULL) OR IT CREATED A NEW ENTRY VIA THE CREATEP ROUTINE      FOR00390
*****   IN EITHER EVENT, POINTER SLOT (AP_POS(1)) IN THE BEU      FOR00400
*****   CONTAINING DATA1 IS UPDATED TO POINT TO DATA2, AND THE      FOR00410
*****   POINTER SLOT(AP_POS(2)) IN THE BEU CONTAINING DATA2 IS      FOR00420
*****   UPDATED TO POINT TO DATA1.      FOR00430
*****4) IF THE BSET IS 1-N THEN AP_POS(1) SHOULD POINT TO THE      FOR00440
*****   PSET CATALOGUE DEFINITION FOR THAT SUBSET. THE PROCEDURE      FOR00450

```

```

***** FIRST CHECKS TO SEE IF THAT POINTER SLOT IS NULL OR FOR00460
***** NOT. IF SO, THIS IS THE FIRST ENTRY FOR THIS SUBSET FOR00470
***** AND IT IS NECESSARY TO CREATE THE SUBSET DEFINITION FOR00480
***** WITHIN THE PSET CATALOGUE. THIS IS DONE VIA A CALL TO FOR00490
***** THE DEFINEP PROCEDURE, PASSING IT THE NAME OF THE FOR00500
***** PSET CONTAINING THE SUBSET ( I.E. DOMAIN2), AS WELL AS FOR00510
***** THE POINTER SLOT ALLOCATED BY THE DEFINED PROCEDURE FOR00520
***** TO BE USED FOR CHAINING ELEMENTS OF THE SUBSET TOGETHERFOR00530
***** THE DEFINEP PROCEDURE RETURNS A POINTER TO THE CATALOG FOR00540
***** ENTRY CREATED AND THIS POINTER IS PLACED IN AP_POS FOR00550
***** (1) OF THE BEU CONTAINING DATA1. IF THE POINTER SLOT FOR00560
***** WAS ORIGINALLY NON-NULL THEN IT ALREADY POINTED TO THE FOR00570
***** CATALOGUE ENTRY. THE NEXT STEP IS TO CHAIN DATA2 TO FOR00580
***** DATA1. THIS CONSISTS OF SEVERAL TASKS. FIRST GET_DOMAIN2FOR00590
***** IS CALLED AND IT EITHER FINDS AN ELIGIBLE OCCURENCE FOR00600
***** OF DATA2 IN DOMAIN2 AND CHAINS IT INTO THE SUBSET, OR FOR00610
***** IT CREATES AN OCCURENCE OF DATA2 WITHIN THE SUBSET THISFOR00620
***** ALSO HAS THE EFFECT OF INSERTING IT INTO DOMAIN2. IN FOR00630
***** EITHER EVENT IT RETURNS A POINTER TO THE BEU WHICH FOR00640
***** CONTAINS DATA2, AND CREATED UPDATES THE AP_POS(2) FOR00650
***** POINTER SLOT POINTED TO BY THAT POINTER TO POINT TO THEFOR00660
***** OCCURENCE OF DATA1 IN DOMAIN1. FOR00670
***** 5) IF THE BSET IS N-1 THE LOGIC IS VERY SIMILAR BUT IN FOR00680
***** REVERSE. IT FIRST CHECKS TO SEE IF AN OCCURENCE OF DATAFOR00690
***** EXISTS, AND IF NOT IT CREATES AN OCCURENCE OF DATA2. IT FOR00700
***** THEN CHECKS TO SEE IF THE AP_POS(2) POINTER SLOT IN THEFOR00710
***** BEU CONTAINING DATA2 IS NULL OR NOT. IF SO, IT IS FOR00720
***** NECESSARY TO CREATE A CATALOGUE ENTRY FOR THE SUBSET FOR00730
***** TO BE CREATED IN DOMAIN1, AND THIS IS ACCOMPLISHED VIA FOR00740
***** A CALL TO DEFINEP. IF IT WAS NECESSARY TO CREATE THE FOR00750
***** CATALOGUE ENTRY THEN THE AP_POS(2) POINTER SLOT OF DATAFOR00760
***** IS UPDATED TO POINT TO IT. THE PROCEDURE THEN CHAINS FOR00770
***** THE OCCURENCE OF DATA1 INTO THE SUBSET, AND UPDATES FOR00780
***** THE AP_POS(1) POINTER SLOT WITHIN THE BEU CONTAINING FOR00790
***** DATA1 SO THAT IT POINTS TO THE OCCURENCE OF DATA2. FOR00800
***** 6) IF THE BSET IS M-N THE FOLLOWING STRATEGY IS FOLLOWED: FOR00810
***** A) IT FIRST CHECKS TO SEE IF THE AP_POS(1) POINTER SLOT FOR00820
***** IN DATA1, IS NULL OR NOT. IF IT IS THEN IT CALLS FOR00830
***** DEFINEP TO CREATE A SUBSET WITHIN THE PSET IDENTIFIEDFOR00840
***** BY MN_NAME AND WHICH ACTS AS A LINK SET BETWEEN FOR00850
***** DOMAIN1 AND DOMAIN2. IT THEN UPDATES THE AP_POS(1) FOR00860
***** POINTER SLOT SO THAT IT POINTS TO THE SUBSET FOR00870
***** CATALOGUE DEFINITION. FOR00880
***** B) IT THEN CREATES AN ENTRY IN THE SUBSET OF THE LINK FOR00890
***** SET VIA A CALL TO CREATEP (NOTE THIS ALSO CREATES FOR00900
***** AN ENTRY IN THE PRIMARY LINK SET PSET. FOR00910
***** C) IT THEN CALLS GET_DOMAIN2 WHICH EITHER FINDS AN FOR00920
***** ELIGIBLE OCCURENCE OF DATA2 IN DOMAIN2 OR IT CREATES FOR00930
***** A NEW OCCURENCE, AND IN ANY EVENT RETURNS A POINTER FOR00940

```


AD-A116 593

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/6 9/2
INFOSAM: A SAMPLE DATABASE MANAGEMENT SYSTEM.(U)
DEC 81 B BLUMBERG

N00039-81-C-0663

UNCLASSIFIED

CISR-M010-8112-07

NL

4 OF 4

AD A
9536



END
DATE
FILMED
08-82
DTIC


```

2 P_ARRAY(16) POINTER,      /* PTR ARRAY FOR LINKING */      BCA00160
2 DATA,                    /* INFO ON PSET ORGANIZATION */    BCA00170
  3 NAME BIT(64),           /* NAME OF PSET */          BCA00180
  ( 3 SP_MAP,               /* MAP OF POINTER ARRAY, */      BCA00190
    3 AP_MAP ) BIT(16),    /* GIVING STATUS OF P_SLOTS */    BCA00200
  3 NUMFREE BIT(8),        /* NOT USED */          BCA00210
  3 SEARCH_INFO,          /* LINKAGE INFORMATION */      BCA00220
    ( 4 L_TYPE,            /* TYPE OF LINK (HASHED ETC..) */  BCA00230
      4 L_POS1,           /* PTR SLOT USED FOR CHAINING */    BCA00240
      4 L_POS2,           /* ADDITIONAL PTR SLOT FOR LINK */  BCA00250
      4 KEY_POS,          /* STARTING POSITION OF KEY */      BCA00260
      4 KEY_LEN ) BIT(8), /* LENGTH OF KEY */          BCA00270
  3 SET_TYPE,             /* SET TYPE INFO */          BCA00280
    ( 4 SUBSET,           /* IF PRIMARY OR SUBSET */      BCA00290
      4 SUBSET_ID,        /* PTR SLOT FOR SUBSET LINK */    BCA00300
      4 P_CHAIN,          /* PTR SLOT PTS TO PRIMARY DCL */  BCA00310
      4 S_CHAIN ) BIT(8), /* SUBSET DCL CHAIN */          BCA00320
  3 DATA_LEN BIT(15);     /* LENGTH OF ELEMENTS */      BCA00330
*****
/* BEU TEMPLATE */      CB 00040
4 1 0 DCL 1 T_ELEMENT BASED(1D),      CB 00050
      2 LENGTH FIXED 8IN(15),      CB 00060
      2 P_ARRAY(16) POINTER,      CB 00070
      2 INFO,                      CB 00080
      3 DATA BIT(320);            CB 00090
5 1 0 DCL (P_CAT,B_CAT) BIT(64) STATIC EXTERNAL;      CB 00100
%INCLUDE BSETSYM;*****CB 00110
/* BSET LINK TYPES */      CB 00120
6 1 0 DCL AI_TO_1 BIT(8) INIT('00000001'B),      BCA00480
      AI_TO_N BIT(8) INIT('00000010'B),      BCA00490
      N_TO_1 BIT(8) INIT('00000100'B),      BCA00500
      M_TO_N BIT(8) INIT('00001000'B);      BCA00510
*****CB 00520
%INCLUDE IDS1;*****CB 00130
/* POINTER STACK RETURNED BY SEARCH */      BCA00420
7 1 0 DCL IDS1 PTR EXTERNAL CONTROLLED;      BCA00430
*****CB 00130
%INCLUDE INFOIND;*****CB 00140
/* DATA STACK RETURNED BY FETCH */      BCA00450
8 1 0 DCL INFO_ND BIT(320) EXTERNAL CONTROLLED;      BCA00460
*****CB 00140
/* PROCEDURES CALLED */      CB 00150
%INCLUDE EDEFINP;*****CB 00160
/* DEFINE PSET MODULE */      BCA00590
9 1 0 DCL DEFINEP ENTRY(BIT(64),BIT(8),BIT(8),BIT(8),BIT(8),
      BIT(8),BIT(8),POINTER);      BCA00600
*****CB 00160
%INCLUDE ECREATEF;*****CB 00170
      * CREATE PSET MODULE */      BCA00630

```

```

10 1 0      DCL CREATEP ENTRY(BIT(64),BIT(320),POINTER);          BCA00640
*****                                           CB 00170
%INCLUDE EFETCH;*****                                           CB 00180
/* FETCH PSET MODULE */                                           EFE00010
11 1 0      DCL FETCH ENTRY(BIT(2),POINTER,BIT(64),BIT(64),BIT(1)); EFE00020
*****                                           CB 00180
%INCLUDE ESEARCH;*****                                           CB 00190
/* SEARCH MODULE */                                           BCA00700
12 1 0      DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER); BCA00710
*****                                           BCA00720
/* MISC PTR VARIABLES */                                           CB 00190
13 1 0      DCL IDXX PTR CTL;                                       CB 00200
14 1 0      DCL (ID,ID_POS,ID2A,ID1,ID2,ID_CAT(2)) POINTER ;      CB 00210
15 1 0      DCL IDINIT POINTER INIT(NULL()), (DATA1,DATA2) BIT(*), CB 00220
      B_SET BIT(64), J FIXED BIN(15),JB BIT(16);                  CB 00230
      CB 00240
      CB 00250
/* GET BSET_CAT ENTRY FOR BSET */                                   CB 00260
16 1 0      CALL FETCH('01'B,IDINIT,B_CAT,B_SET,'1'B);           CB 00270
17 1 0      BASE=INFO_ND;                                           CB 00280
18 1 0      FREE INFO_ND;                                           CB 00290
      CB 00300
/* GET ID OF INSTANCE OF DOMAIN1 IF NOT SUPPLIED */               CB 00310
19 1 0      IF ID1=NULL()                                           CB 00320
      THEN DO;                                                       CB 00330
20 1 1      CALL SEARCH('01'B,BSET_CAT.NAME(1),DATA1,IDXX,ID_POS); CB 00340
21 1 1      IF ALLOCATION(IDS1)=0                                     CB 00350
      THEN DO;                                                       CB 00360
22 1 2      PUT SKIP EDIT('DATA1 NOT FOUND IN DOMAIN1')           CB 00370
      (A);                                                           CB 00380
23 1 2      RETURN;                                                 CB 00390
24 1 2      END;                                                     CB 00400
25 1 1      ID1=IDS1;                                               CB 00410
26 1 1      FREE IDS1;                                              CB 00420
27 1 1      END;                                                    CB 00430
      CB 00440
/* SELECT ON BSET TYPE */                                           CB 00450
28 1 0      SELECT(BSET_CAT.TYPE);                                  CB 00460
      CB 00470
29 1 1      WHEN(A1_TO_1)                                           CB 00480
      DO;                                                            CB 00490
/* ESTABLISH INSTANCE OF DOMAIN2 */                                  CB 00500
30 1 2      ID2=GET_DOMAIN2(BSET_CAT.NAME(2),BSET_CAT.NAME(2),DATA2, CB 00510
      AP_POS(2));                                                  CB 00520
/* CHAIN ACCORDINGLY */                                           CB 00530
31 1 2      ID1->T_ELEMENT.P_ARRAY(AP_POS(1))=ID2;                CB 00540
32 1 2      ID2->T_ELEMENT.P_ARRAY(AP_POS(2))=ID1;                CB 00550
33 1 2      END;                                                    CB 00560
      CB 00570

```

34	1	1	WHEN(A1_TO_N)	CB 00580
			DO;	CB 00590
			/* IF FIRST ELEMENT IN SUBSET, DEFINE SUBSET, AND CHAIN TO	CB 00600
			INSTANCE OF DOMAIN 1 */	CB 00610
35	1	2	IF ID1->T_ELEMENT.P_ARRAY(AP_POS(1))=NULL() THEN	CB 00620
			DO;	CB 00630
36	1	3	CALL DEFINEP(BSET_CAT.NAME(2),'00000100'B,'0'B,	CB 00640
			'0'B,'0'B,'1'B,SUB_ID,ID_CAT(2));	CB 00650
37	1	3	ID1->T_ELEMENT.P_ARRAY(AP_POS(1))=ID_CAT(2);	CB 00660
38	1	3	END;	CB 00670
			/* OTHERWISE SET ID_CAT(2) TO PT TO SUBSET CAT ENTRY */	CB 00680
39	1	2	ELSE ID_CAT(2)=ID1->T_ELEMENT.P_ARRAY(AP_POS(1));	CB 00690
			/* ESTABLISH INSTANCE OF DOMAIN 2 */	CB 00700
40	1	2	ID2=GET_DOMAIN2(BSET_CAT.NAME(2),ID_CAT(2)->CAT_ENTRY.NAME,	CB 00710
			DATA2,AP_POS(2));	CB 00720
			/* CHAIN ACCORDINGLY */	CB 00730
41	1	2	ID2->T_ELEMENT.P_ARRAY(AP_POS(2))=ID1;	CB 00740
42	1	2	END;	CB 00750
43	1	1	WHEN(N_TO_1)	CB 00760
			DO;	CB 00770
			/* ESTABLISH INSTANCE OF DOMAIN 2, CREATING ELEMENT IF	CB 00780
			NECESSARY, ID2 IS SET TO PT TO INSTANCE OF DOMAIN2 */	CB 00790
44	1	2	CALL SEARCH('01'B,BSET_CAT.NAME(2),DATA2,IDX,ID_POS);	CB 00800
45	1	2	IF ALLOCATION(IDS1)= 0	CB 00810
			THEN CALL CREATEP(BSET_CAT.NAME(2),DATA2,ID2);	CB 00820
46	1	2	ELSE DO;	CB 00830
47	1	3	ID2=IDS1;	CB 00840
48	1	3	FREE IDS1;	CB 00850
49	1	3	END;	CB 00860
			/* IF FIRST SUBSET ENTRY, DEFINE SUBSET AND CHAIN TO INSTANCE	CB 00870
			OF DOMAIN 2 */	CB 00880
50	1	2	IF ID2->T_ELEMENT.P_ARRAY(AP_POS(2))=NULL() THEN	CB 00890
			DO;	CB 00900
51	1	3	CALL DEFINEP(DOMAIN_INFO.NAME(1),'00000100'B,'0'B,'0'B,	CB 00910
			'0'B,'1'B,SUB_ID,ID_CAT(1));	CB 00920
52	1	3	ID2->T_ELEMENT.P_ARRAY(AP_POS(2))=ID_CAT(1);	CB 00930
53	1	3	END;	CB 00940
			/* CHAIN INSTANCE OF DOMAIN1 INTO THE SUBSET POINTED TO	CB 00950
			BY THE INSTNACE OF DOMAIN 2 */	CB 00960
54	1	2	ID1->T_ELEMENT.P_ARRAY(SUB_ID)=ID2->T_ELEMENT.P_ARRAY(	CB 00970
			AP_POS(2))->T_ELEMENT.P_ARRAY(SUB_ID);	CB 00980
55	1	2	ID2->T_ELEMENT.P_ARRAY(AP_POS(2))->T_ELEMENT.P_ARRAY(	CB 00990
				CB 01000
				CB 01010
				CB 01020
				CB 01030
				CB 01040
				CB 01050
				CB 01060

```

SUB_ID)=ID1;
/* CHAIN INSTANCE OF DOMAIN2 TO DOMAIN 1 */
ID1->T_ELEMENT.P_ARRAY(AP_POS(1))=ID2;
END;
WHEN(M_TO_N)
DO;
/* IF FIRST ELEMENT IN DOMAIN 2 TO BE LINKED TO INSTANCE OF
DOMAIN 1, THEN DEFINE LINK SET SUBSET AND CHAIN TO INSTANCE
OF DOMAIN 1, SET ID_CAT(2) TO PT TO CREATED CAT ENTRY */
IF ID1->T_ELEMENT.P_ARRAY(AP_POS(1))=NULL() THEN
DO;
CALL DEFINEP(MN_NAME,'00000100'B,'0'B,'0'B,'0'B,'1'B,
SUB_ID,ID_CAT(2));
ID1->T_ELEMENT.P_ARRAY(AP_POS(1))=ID_CAT(2);
END;
/* OTHERWISE SET ID_CAT(2) TO PT TO EXISTING SUBSET CAT ENTRY */
ELSE ID_CAT(2)=ID1->T_ELEMENT.P_ARRAY(AP_POS(1));
/* CREATE INSTANCE OF LINK SET ELEMENT */
J=J+1;
CALL CREATEP(ID_CAT(2)->CAT_ENTRY.NAME,'0'B,ID2A);
/* SET PTR SLOT IN LINK SET ELEMENT TO PT TO INSTANCE OF
OF DOMAIN 1 */
ID2A->T_ELEMENT.P_ARRAY(AP_POS(1))=ID1;
/* ESTABLISH INSTANCE OF DOMAIN 2 */
ID2=GET_DOMAIN2(BSET_CAT.NAME(2),BSET_CAT.NAME(2),DATA2,
AP_POS(2));
/* SET PTR SLOT IN LINK SET ELEMENT TO PT TO INSTANCE OF
OF DOMAIN 2 */
ID2A->T_ELEMENT.P_ARRAY(AP_POS(2))=ID2;
/* SET PTR SLOT IN INSTANCE OF DOMAIN 2 TO POINT TO PRIMARY
CAT ENTRY FOR LINK SET */
ID2->T_ELEMENT.P_ARRAY(AP_POS(2))=ID_CAT(2)->T_ELEMENT.
P_ARRAY(ID_CAT(2)->P_CHAIN);
END;
OTHERWISE PUT SKIP EDIT('ERROR TYPE INCORRECT')(A);
END;
RETURN;
/*****
*****/

```

CB 01070
CB 01080
CB 01090
CB 01100
CB 01110
CB 01120
CB 01130
CB 01140
CB 01150
CB 01160
CB 01170
CB 01180
CB 01190
CB 01200
CB 01210
CB 01220
CB 01230
CB 01240
CB 01250
CB 01260
CB 01270
CB 01280
CB 01290
CB 01300
CB 01310
CB 01320
CB 01330
CB 01340
CB 01350
CB 01360
CB 01370
CB 01380
CB 01390
CB 01400
CB 01410
CB 01420
CB 01430
CB 01440
CB 01450
CB 01460
CB 01470
CB 01480
CB 01490
CB 01500
CB 01510
CB 01520
CB 01530
CB 01540
CB 01550

```

74  1  0  GET_DOMAIN2: PROC(NAME2,NAME2A,DATA2,AP_POS) RETURNS(POINTER);          CB 01560
/*.....*/                                                                    CB 01570
*   THIS MODULE IS RESPONSIBLE FOR ESTABLISHING THE APPROPRIATE *           CB 01580
*   INSTANCE OF DOMAIN 2. IT MAY EITHER FIND AN EXISTING OCCURENCE*           CB 01590
*   WHICH IS AVAILABLE, OR IT WILL CREATE A NEW OCCURENCE. *           CB 01600
*   NAME2 CORRESPONDS TO THE PSET IT SHOULD SEARCH FOR AN *           CB 01610
*   EXISTING OCCURENCE, NAME2A IS THE PSET INTO WHICH THE ELEMENT *           CB 01620
*   SHOULD BE INSERTED IF CREATED, DATA2 IS THE VALUE OF THE *           CB 01630
*   INSTNACE, /ND AP_POS CORRESPONDS TO THE BSET'S ALLOCATED *           CB 01640
*   PTR SLOT IN INSTANCES OF DOMAIN 2. A NEW ELEMENT IS CREATED *           CB 01650
*   IF EITHER AN EXISTING OCCURENCE ISN'T FOUND, OR AN OCCURENCE *           CB 01660
*   IS FOUND, BUT THE PTR SLOT IS ALREADY FULL, MEANING THAT THE *           CB 01670
*   ELEMENT IS ALREADY IN THE SET. *           CB 01680
/*.....*/                                                                    CB 01690
75  2  0  DCL (NAME2A,NAME2) BIT(64), DATA2 BIT(*),AP_POS BIT(8),              CB 01700
          ID2 POINTER;                                                         CB 01710
                                                                              CB 01720
/* SEARCH PSET NAME2 FOR ANY OCCURENCES OF DATA2 */                         CB 01730
76  2  0  CALL SEARCH('10'B,NAME2,DATA2,IDX,IDX_POS);                         CB 01740
77  2  0  ID2=NULL();                                                         CB 01750
                                                                              CB 01760
/* FOR EACH OCCURENCE FOUND */                                              CB 01770
78  2  0  DO WHILE(ALLOCATION(IDS1));                                         CB 01780
                                                                              CB 01790
          /* IF 1 TO 1 THEN, THEN THE PTR SLOT IN DOMAIN 2 MUST BE          CB 01800
          NULL IN ORDER FOR IT TO BE USED. IF MORE THAN ONE                CB 01810
          OCCURENCE THEN USE FIRST THAT IS ACCEPTABLE */                   CB 01820
          IF T PE=A1_TO_1                                                    CB 01830
          THEN IF(IDS1->T_ELEMENT.P_ARRAY(AP_POS)=NULL())                   CB 01840
              &ID2=NULL())                                                  CB 01850
              THEN ID2=IDS1;                                                 CB 01860
          ELSE;                                                              CB 01870
                                                                              CB 01880
          /* OTHERWISE, IF PTR SLOT IS NULL() OR (IN THE CASE OF A          CB 01890
          M TO N ) IF THE PTR SLOT POINTS TO THE PRIMARY SET CAT          CB 01900
          ENTRY FOR THE LINK SET , THEN USE IT */                           CB 01910
          ELSE IF (IDS1->T_ELEMENT.P_ARRAY(AP_POS)=NULL()) ;                 CB 01920
          IDS1->T_ELEMENT.P_ARRAY(AP_POS)=ID_CAT(2)->T_ELEMENT.              CB 01930
          P_ARRAY(ID_CAT(2)->P_CHAIN)) &                                     CB 01940
          ID2=NULL()                                                         CB 01950
          THEN ID2=IDS1;                                                      CB 01960
                                                                              CB 01970
          FREE IDS1;                                                         CB 01980
          END;                                                                CB 01990
                                                                              CB 02000
          /* IF NO ACCEPTABLE INSTANCES WERE FOUND, CREATE ONE IN THE      CB 02010
          NAME2A PSET */                                                     CB 02020
          IF ID2=NULL() THEN                                                 CB 02030
          C'LL CREATEP(NAME2A,DATA2,ID2);                                   CB 02040
          CB 02040

```

85 2 0 RETURN(ID2);
86 2 0 END GET_DOMAIN2;
87 1 0 END CREATEB;

CB 02050
CB 02060
CB 02070
CB 02080

1

```

XINCLUDE SELECTF;*****SF 00010
/*****SEL00010
*
*          MODULE      DESCRIPTION
*          *****/
0 SELECTF:
    PROCEDURE( MODE,      /* BIT(2) */
               ID1,       /* PTR */
               NAME1,     /* BIT(64) */
               DATA1,    /* BIT(320) */
               ID2       ); /* PTR */
/*****SEL00100
*****PURPOSE:
*****THE PURPOSE OF THIS MODULE IS RETRIEVE DATA ITEMS
*****WHICH ARE LINKED WITHIN THE BINARY ASSOCIATION SET
*****SPECIFIED BY NAME1 TO THE OCCURENCE OF THE DATA ITEM
*****POINTED TO BY ID1, OR TO THE DATA ITEM SPECIFIED BY
*****DATA1. IT RETURNS AN EXTERNAL STACK OF DATA VALUES
*****WHICH CORRESPONDS TO ALL OF THE ELEMENTS LINKED IN
*****THE BINARY ASSOCIATION SET WITH DATA1.
*****SEL00200
*****METHOD:
*****1) FIRST FETCHS THE BSET_CAT ENTRY FOR NAME1
*****2) IF ID1 IS NULL IT CALL SEARCH USING NAME(1)
*****   FROM BSET_CAT AS THE PSET AND DATA1 AS THE
*****   KEY INTO THE PSET. IF NO VALUE IS FOUND,OR
*****   IF THE POINTER SLOT ALLOCATED TO THIS BINARY
*****   ASSOCIATION IS NULL() THE MODULE RETURNS.
*****3) HAVING ESTABLISHED THE OCCURENCE OF A DATA
*****   VALUE IN DOMAIN 1 IT PROCEEDS TO RETRIEVE
*****   THE ASSOCIATED ITEMS IN DOMAIN 2. THE LOGIC
*****   OF THE RETRIEVAL DEPENDS ON THE TYPE OF BINARY
*****   ASSOCIATION.
*****   1-1 OR N-1 RETRIEVES SINGLE VALUE BY CALLING
*****   FETCH AND PASSING IT THE POINTER
*****   VALUE CONTAINED IN POINTER SLOT
*****   AP_POS(1).
*****   1-N
*****   IN THIS CASE THE POINTER SLOT AP_POS
*****   (1) CONTAINS A POINTER TO A CATALOG
*****   ENTRY FOR THE APPROPRIATE SUBSET WITH
*****   IN DOMAIN2. FETCH IS USED TO RETRIEVE
*****   ALL ELEMENTS OF THAT SUBSET.
*****   M-N
*****   IN THIS CASE POINTER SLOT AP_POS(1)
*****   POINTS TO A CATALOG ENTRY FOR A SUB-
*****   SET WITHIN THE LINKAGE SET MN_NAME.

```

```

*****
*****      IT CALLS THE SEARCH MODULE TO FETCH  SEL00460
*****      POINTERS TO EACH OF THE ELEMENTS IN  SEL00470
*****      THAT SUBSET, AND THEN GOES THROUGH  SEL00480
*****      THE POINTER VALUE CONTAINED IN AP_POSSEL00490
*****      (2) TO RETRIEVE THE ACTUAL DATA ITEMSSEL00500
*****      IN THE SECOND DOMAIN BY PASSING THAT SEL00510
*****      POINTER TO FETCH                      SEL00520
*****                                           SEL00530
*****.....SEL00540
*****      INPUT PARAMETERS:                      SEL00550
*****      1) MODE - USED TO DETERMINE HOW MANY DATA SEL00560
*****            DATA ITEMS TO FETCH:           SEL00570
*****            '01' - FIRST OCCURENCE          SEL00580
*****            '11' - ALL OCCURENCES           SEL00590
*****      2) ID1 - PTR VALUE WHICH CAN BE USED TO SEL00600
*****            IDENTIFY DATA ITEM IN DOMAIN 1. IF IT SEL00610
*****            IS NOT NULL, IT IS ASSUMED TO POINT TO SEL00620
*****            A VALID ITEM, AND THE DATA VALUE SPECIFIED SEL00630
*****            BY DATA1 IS IGNORED.            SEL00640
*****      3) NAME1 - THE NAME OF THE BINARY ASSOCIATION SEL00650
*****            THROUGH WHICH THE LINK IS TO BE MADE. IT SEL00660
*****            MUST CORRESPOND TO AN EXISTING B_SET SEL00670
*****            DEFINITION IN THE BSET_CAT.      SEL00680
*****      4) DATA1 - IF ID1 IS NULL() DATA1 IS USED AS SEL00690
*****            A KEY TO ESTABLISH THE DESIRED OCCURENCE SEL00700
*****            OF DOMAIN 1.                     SEL00710
*****      5) ID2 - NOT USED ON INPUT.            SEL00720
*****                                           SEL00730
*****.....SEL00740
*****      OUTPUT PARAMETERS:                      SEL00750
*****      1) INFO_NO - AN EXTERNAL CONTROLLED STACK OF SEL00760
*****            BIT STRINGS BIT(320) WHICH CORRESPONDS TO SEL00770
*****            THE DATA ITEMS FOUND IN DOMAIN 2 WHICH SEL00780
*****            ARE LINKED TO THE OCCURENCE OF DATA1. NOTE SEL00790
*****            INFO_NO IS CREATED BY THE FETCH MODULE SEL00800
*****            WHICH IS INVOKED BY THIS MODULE.   SEL00810
*****      2) ID2 - PTR VALUE WHICH POINTS TO LAST DATA SEL00820
*****            ELEMENT FOUND IN DOMAIN 2.         SEL00830
*****                                           SEL00840
*****.....SEL00850
*****      CALLS PROCEDURES:                      SEL00860
*****            FETCH, SEARCH                    SEL00870
*****                                           SEL00880
*****.....SEL00890
*****                                           SF 00010
*****                                           SF 00020
*****                                           SF 00030
*****                                           SF 00040
*****                                           SF 00050
*****
*****      /* PARAMETER DECLARATIONS */
*****      DCL NAME1 BIT(64),DATA1 BIT(*),MODE BIT(2), FND BIT(1);

```



```

XINCLUDE BSETCAT;*****SF 00060
/* BSET_CAT TEMPLATE */
3 1 0 DCL BASE BIT(320); /* USED FOR BEU OVERLAY */ BCA00020
1 BSET_CAT DEFINED (BASE); BCA00030
2 SET_NAME BIT(64); /* NAME OF BSET */ BCA00040
2 DOMAIN_INFO(2); /* DOMAIN INFORMATION */ BCA00050
3 NAME BIT(64); /* NAME OF DOMAIN */ BCA00060
3 AP_POS BIT(8); /* PTR SLOT USED FOR LINK */ BCA00070
2 TYPE BIT(8); /* TYPE OF BSET */ BCA00080
2 SUB_ID BIT(8); /* PTR SLOT FOR SUBSET LINK */ BCA00090
2 MN_NAME BIT(64); /* NAME OF M TO N LINK SET */ BCA00100
*****SF 00060
XINCLUDE PSETCAT;*****SF 00070
/* PSET_CAT TEMPLATE */
4 1 0 DCL 1 CAT_ENTRY BASED(P); /* BASED ON ID OF PSET_CAT BEU */ BCA00130
2 LENGTH FIXED BIN(15); /* LENGTH OF CAT ENTRY */ BCA00140
2 P_ARRAY(16) POINTER; /* PTR ARRAY FOR LINKING */ BCA00150
2 DATA; /* INFO ON PSET ORGANIZATION */ BCA00160
3 NAME BIT(64); /* NAME OF PSET */ BCA00170
( 3 SP_MAP; /* MAP OF POINTER ARRAY */ BCA00180
3 AP_MAP ) BIT(16); /* GIVING STATUS OF P_SLOTS */ BCA00190
3 NUMFREE BIT(8); /* NOT USED */ BCA00200
3 SEARCH_INFO; /* LINKAGE INFORMATION */ BCA00210
( 4 L_TYPE; /* TYPE OF LINK (HASHED ETC..) */ BCA00220
4 L_POS1; /* PTR SLOT USED FOR CHAINING */ BCA00230
4 L_POS2; /* ADDITIONAL PTR SLOT FOR LINK */ BCA00240
4 KEY_POS; /* STARTING POSITION OF KEY */ BCA00250
4 KEY_LEN ) BIT(8); /* LENGTH OF KEY */ BCA00260
3 SET_TYPE; /* SET TYPE INFO */ BCA00270
( 4 SUBSET; /* IF PRIMARY OR SUBSET */ BCA00280
4 SUBSET_ID; /* PTR SLOT FOR SUBSET LINK */ BCA00290
4 P_CHAIN; /* PTR SLOT PTS TO PRIMARY DCL */ BCA00300
4 S_CHAIN ) BIT(8); /* SUBSET DCL CHAIN */ BCA00310
3 DATA_LEN BIT(15); /* LENGTH OF ELEMENTS */ BCA00320
*****SF 00070
/* NAMES OF PSET_CAT AND BSET_CAT PSETS */
5 1 0 DCL (P_CAT,B_CAT) BIT(64) STATIC EXTERNAL; SF 00080
*****SF 00090
XINCLUDE BSETSYM;*****SF 00100
/* BSET LINK TYPES */
6 1 0 DCL A1_TO_1 BIT(8) INIT('00000001'B); BCA00480
A1_TO_N BIT(8) INIT('00000010'B); BCA00490
N_TO_1 BIT(8) INIT('00000100'B); BCA00500
M_TO_N BIT(8) INIT('00001000'B); BCA00510
*****SF 00120
XINCLUDE IDS1;*****SF 00130
/* POINTER STACK RETURNED BY SEARCH */
BCA00420

```

```

7   1   0   DCL IDS1 PTR EXTERNAL CONTROLLED;                      BCA00430
*****                                           SF 00140
                                           SF 00150
                                           SF 00160
                                           SF 00170
                                           EFE00020
                                           EFE00030
                                           SF 00170
                                           SF 00180
                                           BCA00700
                                           BCA00710
                                           BCA00720
                                           SF 00180
                                           SF 00190
                                           SF 00200
                                           SF 00210
                                           SF 00220
                                           SF 00230
                                           SF 00240
                                           SF 00250
                                           SF 00260
                                           SF 00270
                                           SF 00280
                                           SF 00290
                                           SF 00300
                                           SF 00310
                                           SF 00320
                                           SF 00330
                                           SF 00340
                                           SF 00350
                                           SF 00360
                                           SF 00370
                                           SF 00380
                                           SF 00390
                                           SF 00400
                                           SF 00410
                                           SF 00420
                                           SF 00430
                                           SF 00440
                                           SF 00450
                                           SF 00460
                                           SF 00470
                                           SF 00480
                                           SF 00490
                                           SF 00500
                                           SF 00510
                                           SF 00520
                                           SF 00530
                                           SF 00540

/* PROCEDURES CALLED */
%INCLUDE EFETCH;*****
/* FETCH PSET MODULE */
8   1   0   DCL FETCH ENTRY(BIT(2),POINTER,BIT(64),BIT(64),BIT(1));
*****
%INCLUDE ESEARCH;*****
/* SEARCH MODULE */
9   1   0   DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER);
*****

/* POINTERS USED TO POINT TO BEUS */
10  1   0   DCL (ID,ID1,ID2,ID_CAT(2)) POINTER ,
IDXX PTR CONTROLLED;
/* TEMPLATE FOR BEU INTERPRETATION */
11  1   0   DCL 1 T_ELEMENT BASED(ID1),
2 LENGTH FIXED BIN(15),
2 P_ARRAY(16) POINTER,
2 DATA BIT(320),
INFO_NO BIT(320) CONTROLLED EXTERNAL;

/* FETCH BSET_CAT ENTRY FOR BSET, AND SET UP BSET_CAT STRUCTURE */
12  1   0   ID=NULL();
13  1   0   CALL FETCH('01'B,ID,B_CAT,NAME1,'1'B);
14  1   0   BASE=INFO_NO;
15  1   0   FREE INFO_NO;

/* IF INSTANCE OF DOMAIN 1 IDENTIFIED BY KEY, GET ID OF INSTANCE */
16  1   0   IF ID1=NULL()
17  1   1   THEN DO;
CALL SEARCH('01'B,BSET_CAT.NAME(1),DATA1,IDXX,
ID_CAT(1));

/* IF INSTANCE FOUND AND IN BSET */
18  1   1   IF ALLOCATION(IDS1)^=0 &
IDS1->T_ELEMENT.P_ARRAY(AP_POS(1))^=NULL()
THEN DO;
ID1=IDS1;
FREE IDS1;
END;

/* INSTANCE NOT FOUND OR NOT IN BSET */
22  1   1   ELSE DO;
IF ALLOCATION(IDS1)^=0 THEN FREE IDS1;
RETURN;
END;
23  1   2
24  1   2
25  1   2

```

26	1	1	END;	SF 00550
			/* SET ID2 TO CONTENTS OF POINTER SLOT FOR BSET */	SF 00560
27	1	0	ID2=ID1->T_ELEMENT.P_ARRAY(AP_POS(1));	SF 00570
				SF 00580
28	1	0	IF TYPE= A1_TO_N THEN	SF 00590
			/* GET ELEMENTS IN SUBSET POINTED TO BY ID2 */	SF 00600
			CALL FETCH(MODE, ID, ID2->CAT_ENTRY.NAME, '0'B, FND);	SF 00610
				SF 00620
29	1	0	ELSE IF TYPE=M_TO_N THEN	SF 00630
			DO	SF 00640
			/* GET ELEMENTS IN LINK SET SUBSET PTED TO BY ID2 */	SF 00650
30	1	1	CALL SEARCH(MODE, ID2->CAT_ENTRY.NAME, '0'B, IDXX,	SF 00660
			ID_CAT(2));	SF 00670
				SF 00680
			/* GO THROUGH ELEMENTS IN LINKSET */	SF 00690
31	1	1	L=ALLOCATION(IDS1);	SF 00700
32	1	1	DO K=1 TO L;	SF 00710
			/* SET ID2 = PTR TO INSTANCE IN DOMAIN 2 */	SF 00720
33	1	2	ID2=IDS1->T_ELEMENT.P_ARRAY(AP_POS(2));	SF 00730
34	1	2	FREE IDS1;	SF 00740
			/* GET INSTANCE OF DOMAIN 2 */	SF 00750
35	1	2	CALL FETCH('01'B, ID2, '0'B, '0'B, FND);	SF 00760
36	1	2	END;	SF 00770
37	1	1	END;	SF 00780
				SF 00790
			/* OTHERWISE GET INSTANCE POINTED TO DIRECTLY BY ID2 */	SF 00800
38	1	0	ELSE CALL FETCH('01'B, ID2, '0'B, '0'B, FND);	SF 00810
39	1	0	RETURN;	SF 00820
40	1	0	END SELECTF;	SF 00830

```

%INCLUDE DEFINEP:*****DP 00010
/*****DEF00010
*      MODULE DESCRIPTION      *DEF00020
*****DEF00030
1  0  DEFINEP:    PROCEDURE      DEF00040
                        (NAME1,    /* BIT(64) */      DEF00050
                        L_TYPE1,    /* BIT(64) */      DEF00060
                        KEY_POS1,    /* BIT(8)  */      DEF00070
                        KEY_LEN1,    /* BIT(8)  */      DEF00080
                        LEN1,        /* BIT(16) */      DEF00090
                        SUBSET1,     /* BIT(8)  */      DEF00100
                        S_ID1,       /* BIT(8)  */      DEF00110
                        P2           /* POINTER */ );    DEF00120
/*****DEF00130
*****  PURPOSE:      DEF00140
*****  THE PURPOSE OF THIS PROCEDURE IS TO CREATE AND MAINTAIN DEF00150
*****  A CATALOGUE OF ALL PRIMARY SETS AND SUBSETS DEFINED IN DEF00160
*****  THE SYSTEM. EVERY PSET AND SUBSET (I.E. A PSET WHICH IS DEF00170
*****  A SUBSET OF ANOTHER PSET) HAS AN ENTRY IN THE PSET DEF00180
*****  CATALOGUE. THE CATALOGUE ENTRY (SEE CAT_ENTRY) SERVES DEF00190
*****  SEVERAL PURPOSES: DEF00200
*****  A) IT CONTAINS INFORMATION ON HOW DEF00210
*****  THE PSET IS ORGANIZED (I.E. THE DEF00220
*****  ACCESS METHOD, HASHED, B_TREE, LINEAR DEF00230
*****  B) INFORMATION ON HOW TO INTERPRET DEF00240
*****  THE CONTENTS OF THE POINTER DEF00250
*****  SLOTS, AND THEIR STATUS (FREE OR ALLOCATED) DEF00260
*****  C) SERVES AS A HEADER TO THE PSET. IT EITHER DEF00270
*****  CONTAINS A POINTER TO THE FIRST ELEMENT IN DEF00280
*****  THE PSET, OR A POINTER TO AN INDEX TO THE DEF00290
*****  SET. DEF00300
*****  HENCE, THE PURPOSE OF THIS MODULE IS TO CREATE THAT DEF00310
*****  CATALOGUE ENTRY, ALLOCATE ANY POINTER SLOTS NECESSARY DEF00320
*****  TO BE USED FOR ORGANIZING THE PSET, AND CREATING ANY DEF00330
*****  SUPPORT STRUCTURES (I.E. A SCATTER TABLE , IF HASHED) DEF00340
*****  *****DEF00350
*****  *****DEF00360
*****  METHOD:      DEF00370
*****  THE OBJECTIVE OF THIS MODULE IS TO CREATE AN ENTRY IN DEF00380
*****  THE P_CAT PSET WHICH REPRESENTS THE P_CAT CATALOGUE DEF00390
*****  ENTRY FOR THE PSET. IN ORDER TO ACCOMPLISH THIS IT IS DEF00400
*****  NECESSARY TO BUILD A TEMPORARY ENTRY CALLED CAT_ENTRY DEF00410
*****  WHICH IS THEN INSERTED INTO THE P_CAT PSET VIA THE DEF00420
*****  CREATEP MODULE. HENCE, THE CATALOGUE ENTRY IS STORED DEF00430
*****  WITHIN A BEU WHICH IS AN ELEMENT WITHIN THE P_CAT PSET. DEF00440
*****  WPENEVER THE CATALOGUE ENTRY IS REQUIRED, THE BEU WHICH DEF00450

```

*****	CONTAINS IT IS FETCHED AND THE CAT_ENTRY STRUCTURE IS	DEF00460
*****	OVERLAID ON DATA PORTION OF THE BEU. THE FOLLOWING	DEF00470
*****	STRATEGY IS EMPLOYED TO CREATE THE CATALOGUE ENTRY.	DEF00480
*****	A) IF THIS IS THE FIRST PSET TO HAVE BEEN DEFINED IT	DEF00490
*****	IS FIRST NECESSARY TO CREATE THE P_CAT PSET. A	DEF00500
*****	MODULE CALLED INIT_P (INTERNAL TO DEFINEP) IS	DEF00510
*****	RESPONSIBLE FOR THIS. INIT_P ESSENTIALLY CREATES	DEF00520
*****	A BEU WHICH CONTAINS THE P_CAT CATALOGUE ENTRY FOR	DEF00530
*****	THE P_CAT PSET, AND THIS BECOMES THE FIRST ENTRY IN	DEF00540
*****	THE P_CAT PSET. THIS ENTRY CONTAINS INFORMATION ON	DEF00550
*****	HOW THE P_CAT PSET IS TO BE ORGANIZED. IN ADDITION,	DEF00560
*****	INIT_P CREATES A SCATTER TABLE FOR THE P_CAT PSET.	DEF00570
*****	B) THE NEXT STEP IS TO RETRIEVE A TEMPLATE FOR THE NEW	DEF00580
*****	CATALOGUE ENTRY. IF THE PSET TO BE DEFINED IS A SUB-	DEF00590
*****	SET OF ANOTHER PSET, THEN A POINTER TO THE CATALOGUE	DEF00600
*****	ENTRY FOR THAT PSET IS RETRIEVED VIA THE SEARCH	DEF00610
*****	ROUTINE. OTHERWISE, THE SEARCH ROUTINE IS CALLED TO	DEF00620
*****	RETURN A POINTER TO THE P_CAT PSET CATALOGUE ENTRY.	DEF00630
*****	THE OBJECTIVE HERE IS THAT IF THE PSET IS A SUBSET OF	DEF00640
*****	ANOTHER PSET THEN ITS CATALOGUE ENTRY MUST REFLECT	DEF00650
*****	THE ORGANIZATION OF THE PSET OF WHICH IT IS A SUBSET.	DEF00660
*****	C) IN EITHER CASE, THE INFORMATION IN THE BEU POINTED	DEF00670
*****	TO BY THE POINTER RETURNED IN (B) IS COPIED INTO A	DEF00680
*****	A TEMPORARY STRUCTURE CALLED CAT_ENTRY.	DEF00690
*****	D) IF THE PSET BEING DEFINED IS NOT A SUBSET THEN CAT_	DEF00700
*****	ENTRY IS MODIFIED TO REFLECT THE INFORMATION PASSED	DEF00710
*****	TO DEFINEP VIA THE INPUT PARAMETERS. THE INFORMATION	DEF00720
*****	INCLUDES: THE NAME, THE KEY POSITION AND LENGTH (BITS),	DEF00730
*****	LENGTH OF THE DATA ELEMENTS, AND THE LINK TYPE (	DEF00740
*****	HASHED, B_TREE, OR LINEAR). DEPENDING ON THE ACCESS	DEF00750
*****	METHOD OR LINK TYPE DESIRED FOR THE PSET IT IS ALSO	DEF00760
*****	NECESSARY TO ALLOCATE POINTER SLOTS. IF THE ACCESS	DEF00770
*****	METHOD IS HASHING VIA A SCATTER TABLE THEN MAPSET IS	DEF00780
*****	CALLED TO ALLOCATE (I.E. RESERVE) A POINTER SLOT	DEF00790
*****	TO BE USED FOR OVERFLOW CHAINING. L_POS2 IN CAT_ENTRY	DEF00800
*****	IS UPDATED TO REFLECT THE POSITION OF THIS PTR SLOT.	DEF00810
*****	IF A B_TREE IS TO BE EMPLOYED, MAPSET IS CALLED TWICE,	DEF00820
*****	ONCE TO RESERVE A PTR SLOT TO BE USED TO CHAIN LEFT	DEF00830
*****	DESCENDENTS, AND ONCE TO RESERVE A SLOT TO BE USED TO	DEF00840
*****	CHAIN RIGHT DESCENDENTS. L_POS1 AND L_POS2 ARE UP-	DEF00850
*****	DATED ACCORDINGLY. IF A SIMPLE LINEAR PTR CHAIN IS	DEF00860
*****	TO BE EMPLOYED THEN MAPSET IS CALLED TO RESERVE A	DEF00870
*****	SINGLE PTR SLOT TO BE USED FOR CHAINING AND L_POS2 IS	DEF00880
*****	UPDATED TO REFLECT THAT POSITION.	DEF00890
*****	F) IF THE PSET TO BE DEFINED IS A SUBSET OF ANOTHER PSET	DEF00900
*****	THEN THE LINK TYPE IS REQUIRED TO BE LINEAR. IF S_ID1	DEF00910
*****	(AN INPUT PARAMETER) IS 0 THEN MAPSET IS CALLED TO	DEF00920
*****	RESERVE A POINTER SLOT TO BE USED TO CHAIN THE ELEMENTS	DEF00930
*****	OF THE SUBSET TOGETHER, AND THIS VALUE IS BOTH RETURNED	DEF00940

```

***** AND PLACED IN CAT_ENTRY.S_ID. IF S_ID1 IS NON-ZERO DEF00950
***** THEN THE S_ID1 POINTER SLOT IS ASSUMED TO BE AVAILABLE DEF00960
***** FOR SUBSET CHAINING.(SUBSETS ARE USED PRIMARILY AS A DEF00970
***** MEANS OF IMPLEMENTING 1-N BINARY ASSOCIATIONS. AS DEF00980
***** A RESULT A PRIMARY PSET MAY CONTAIN EXCLUSIVE SUBSETS DEF00990
***** ALL OF WHICH MAY SHARE A COMMON PTR SLOT FOR CHAINING. DEF01000
***** WHEN THE DEFINED MODULE DEFINES A 1-N OR N-1 BSET,IT DEF01010
***** CALLS MAPSET TO RESERVE A PTR SLOT FOR SUBSET CHAINING. DEF01020
***** WHEN THE CREATED MODULE CREATES A 1-N OR N-1 BSET IT DEF01030
***** CALLS DEFINED TO CREATE THE APPROPRIATE SUBSET IF DEF01040
***** NECESSARY AND PASSES IT THE VALUE FOR THE S_ID. ) DEF01050
***** IN ADDITION, IT IS NECESSARY TO MAINTAIN A CHAIN OF DEF01060
***** ALL OF THE SUBSET DEFINITIONS FOR SUBSETS WITHIN A DEF01070
***** GIVEN PSET, AS WELL AS HAVE A PTR IN EACH SUBSET DEF01080
***** CATALOGUE ENTRY WHICH POINTS TO THE PRIMARY PSET DEF01090
***** CATALOGUE ENTRY. TWO PTR SLOTS ARE ALLOCATED FOR THESE DEF01100
***** PURPOSES, AND THE LOCATIONS OF THESE SLOTS ARE CON- DEF01110
***** TAINED IN S_CHAIN AND P_CHAIN RESPECTIVELY. IF THIS DEF01120
***** IS THE FIRST SUBSET TO BE DEFINED WITHIN THE PRIMARY DEF01130
***** PSET THEN MAPSET IS CALLED TWICE TO RESERVE POINTER DEF01140
***** SLOTS FOR THIS, AND BOTH THE SUBSET AND PRIMARY PSET DEF01150
***** CATALOGUE ENTRIES ARE UPDATED TO REFLECT THE NEWLY DEF01160
***** RESERVED PTR SLOTS. FINALLY, NAMEGEN IS CALLED TO DEF01170
***** CREATE A NAME FOR THE SUBSET, AND THIS VALUE IS PLACED DEF01180
***** IN CAT_ENTRY.NAME. DEF01190
***** G) IN EITHER CASE THE NEXT STEP IS TO CONVERT THE DATA DEF01200
***** PORTION OF CAT_ENTRY INTO A BIT STRING VIA THE STRING DEF01210
***** FUNCTION. THIS BIT STRING IS THEN PASSED TO THE CREATEPDEF01220
***** MODULE WHICH CREATES A BEU AND INSERTS IT INTO THE PCATDEF01230
***** PSET. CREATEP RETURNS A PTR TO THE NEWLY CREATED BEU. DEF01240
***** H) IF THE LINK TYPE IS HASHED THEN CREATE_I (AN IN- DEF01250
***** TERNAL PROCEDURE)IS CALLED TO CREATE A SCATTER TABLE DEF01260
***** FOR THE PSET.(EACH PSET WHICH IS HASHED HAS ITS OWN DEF01270
***** SCATTER TABLE WHICH IS IMPLEMENTED AS A BASED STRUCTUREDEF01280
***** WHICH CONTAINS A POINTER ARRAY. WHEN IN USE, AN ENTRY DEF01290
***** IN THE POINTER ARRAY IS EITHER NULL, OR POINTS EITHER DEF01300
***** TO THE BEU CONTAINING THE KEY VALUE OR TO AN OVER- DEF01310
***** FLOW CHAIN.) CREATE_I RETURNS A PTR TO THE SCATTER DEF01320
***** TABLE, AND THIS PTR IS PLACED IN THE L_POS1 PTR SLOT DEF01330
***** THE BEU CONTAINING THE CATALOGUE ENTRY FOR THE PSET. DEF01340
***** I) IF THE PSET IS A SUBSET THEN IT IS NECESSARY TO CHAIN DEF01350
***** THE SUBSET CATALOGUE ENTRY TO THE PRIMARY PSET CATALOGUEDEF01360
***** ENTRY. THIS IS NECESSARY SO THAT IF AN INSERTION IS DEF01370
***** MADE INTO A SUBSET, THEN THE ELEMENT CAN ALSO BE DEF01380
***** INSERTED INTO THE PRIMARY PSET. IN ADDITION, SUBSET DEF01390
***** SET DEFINITIONS FOR A GIVEN PSET ARE CHAINED TOGETHER DEF01400
***** THESE TASKS ARE ACCOMPLISHED HERE BY UPDATING THE P_ DEF01410
***** CHAIN PTR SLOT IN THE BEU CONTAINING THE SUBSET CATALOGDEF01420
***** DEFINITION SO THAT IT POINTS TO THE PRIMARY PSET DEF01430

```

```

*****      DEFINITION, AND IN ORDER TO CHAIN THE SUBSETS TOGETHERDEF01440
*****      THE SUBSET DEFINITION IS INSERTED AT THE FRONT OF THE DEF01450
*****      SUBSET CHAIN. DEF01460
*****      J) THE FINAL STEP IS TO FREE THE TEMPORARY STRUCTURE USED DEF01470
*****      TO BUILD THE CATALOGUE ENTRY. DEF01480
*****      DEF01490
*****      ..... DEF01500
*****      INPUT PARAMETERS: DEF01510
*****      NAME1 - IF THIS IS NOT A SUBSET OF AN EXISTING PSET, DEF01520
*****      THEN NAME1 IS THE NAME OF THE PSET TO BE DEFINED. DEF01530
*****      OTHERWISE, IT IS THE NAME OF THE PRIMARY PSET DEF01540
*****      FOR WHICH THIS IS A SUBSET. DEF01550
*****      L_TYPE1 - THE ACCESS METHOD TO BE EMPLOYED: DEF01560
*****      '00000001'B -HASHING VIA A SCATTER TABLE DEF01570
*****      '00000010'B -B_TREE DEF01580
*****      '00000100'B -LINEAR PTR CHAIN DEF01590
*****      KEY_POS1 - THE STARTING POSITION (IN BITS) OF THE KEY DEF01600
*****      WITHIN THE DATA AREA OF BEUS WITHIN THIS DEF01610
*****      PSET. DEF01620
*****      KEY_LEN1 - THE LENGTH (IN BITS) OF THE KEY, MAXIMUM DEF01630
*****      KEY LENGTH OF 128 BITS. DEF01640
*****      LEN1 - LENGTH OF DATA PORTION OF BEU. (NOT USED IN THIS DEF01650
*****      IMPLEMENTATION SINCE BEUS ARE FIXED SIZE) DEF01660
*****      SUBSET1 - FLAG TO INDICATE IF THIS IS A SUBSET: DEF01670
*****      '0'B - IF NOT A SUBSET DEF01680
*****      '1'B - IF A SUBSET DEF01690
*****      S_ID1 - INDICATES PTR SLOT TO BE USED FOR CHAINING DEF01700
*****      ELEMENTS OF SUBSET. IF NOT A SUBSET, DISREGARDEDDEF01710
*****      P2 - NOT SIGNIFICANT ON INPUT. DEF01720
*****      DEF01730
*****      ..... DEF01740
*****      OUTPUT PARAMETERS: DEF01750
*****      S_ID1 - IF THIS IS A SUBSET AND S_ID1 IS INITIALLY '0'B DEF01760
*****      THEN THIS MODULE RETURNS THE VALUE THAT MAPSET DEF01770
*****      RESERVED TO BE USED FOR CHAINING. DEF01780
*****      P2 - PTR VALUE WHICH POINTS TO THE PSET CATALOGUE ENTRY DEF01790
*****      CREATED. (MOSTLY USED BY THE CREATEB MODULE WHEN DEF01800
*****      IT HAS CALLED THIS MODULE TO DEFINE A PSET DEFINITIONDEF01810
*****      FOR A SUBSET, AND THE PTR TO THIS DEFINITION IS DEF01820
*****      TO BE INSERTED INTO A PTR SLOT OF A BEU IN DOMAIN1 DEF01830
*****      IF 1-N, OR WITHIN DOMAIN2 IF N-1.) DEF01840
*****      DEF01850
*****      ..... DEF01860
*****      PROCEDURES INVOKED: DEF01870
*****      SEARCH, CREATEP, MAPSET, (INIT_P, CREATE_I) INTERNAL DEF01880
*****      DEF01890
*****      ...../ DEF01900
*****      ..... DP 00010
*****      ..... DP 00020
*****      DCL (I_TYPE1,KEY_POS1,KEY_LEN1,SUBSET1,S_ID1) BIT(8),LEN1

```

```

                                BIT(16),NAME1 BIT(64);                                DP 00030
%INCLUDE PSETCAT;*****DP 00040
/* PSET_CAT TEMPLATE */                                BCA00130
3 1 0 DCL 1 CAT_ENTRY BASED(P), /* BASED ON ID OF PSET_CAT BEU */ BCA00140
      2 LENGTH FIXED BIN(15), /* LENGTH OF CAT ENTRY */ BCA00150
      2 P_ARRAY(16) POINTER, /* PTR ARRAY FOR LINKING */ BCA00160
      2 DATA, /* INFO ON PSET ORGANIZATION */ BCA00170
        3 NAME BIT(64), /* NAME OF PSET */ BCA00180
        ( 3 SP_MAP, /* MAP OF POINTER ARRAY, */ BCA00190
          3 AP_MAP ) BIT(16), /* GIVING STATUS OF P_SLOTS */ BCA00200
          3 NUMFREE BIT(8), /* NOT USED */ BCA00210
          3 SEARCH_INFO, /* LINKAGE INFORMATION */ BCA00220
            ( 4 L_TYPE, /* TYPE OF LINK (HASHED ETC..) */ BCA00230
              4 L_POS1, /* PTR SLOT USED FOR CHAINING */ BCA00240
              4 L_POS2, /* ADDITIONAL PTR SLOT FOR LINK */ BCA00250
              4 KEY_POS, /* STARTING POSITION OF KEY */ BCA00260
              4 KEY_LEN ) BIT(8), /* LENGTH OF KEY */ BCA00270
          3 SET_TYPE, /* SET TYPE INFO */ BCA00280
            ( 4 SUBSET, /* IF PRIMARY OR SUBSET */ BCA00290
              4 SUBSET_ID, /* PTR SLOT FOR SUBSET LINK */ BCA00300
              4 P_CHAIN, /* PTR SLOT PTS TO PRIMARY DCL */ BCA00310
              4 S_CHAIN ) BIT(8), /* SUBSET DCL CHAIN */ BCA00320
          3 DATA_LEN BIT(15); /* LENGTH OF ELEMENTS */ BCA00330
      ***** DP 00040
%INCLUDE IDS1;*****DP 00050
/* POINTER STACK RETURNED BY SEARCH */ BCA00420
4 1 0 DCL IDS1 PTR EXTERNAL CONTROLLED; BCA00430
      ***** DP 00050
%INCLUDE PSET_YM;*****DP 00060
/* PSE LINK TYPES */ BCA00540
5 1 0 DCL HASHED BIT(8) INIT('00000001'B), BCA00550
      B_TREE BIT(8) INIT('00000010'B), BCA00560
      LINEAR BIT(8) INIT('00000100'B); BCA00570
      ***** DP 00060
/* PROCEDURES CALLED */ DP 00070
%INCLUDE EHASH;*****DP 00080
/* HASHING MODULE */ DEC00020
6 1 0 DCL HASH ENTRY(BIT(64),FIXED BIN(15)) RETURNS(FIXED BIN(15)); DEC00030
      ***** DP 00080
%INCLUDE EPRINTP;*****DP 00090
/* DIAGNOSTIC PRINT MODULE */ DEC00050
7 1 0 DCL PRINIP ENTRY(POINTER); DEC00060
      ***** DP 00090
%INCLUDE ESEARCH;*****DP 00100
/* SEARCH MODULE */ BCA00700
8 1 0 DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER); BCA00710
      ***** BCA00720
      ***** DP 00100
%INCLUDE ECREATE;*****DP 00110

```



```

/* BEU CREATION MODULE */
9 1 0 DCL CREATEE ENTRY(BIT(320),BIT(16),POINTER); ECR00020
***** ECR00030
%INCLUDE ECREATP;***** DP 00110
/* CREATE PSET MODULE */ DP 00120
10 1 0 DCL CREATEP ENTRY(BIT(64),BIT(320),POINTER); BCA00630
***** BCA00640
%INCLUDE EMAPSET;***** DP 00120
/* MAP MAINTENANCE MODULE */ DP 00130
11 1 0 DCL MAPSET ENTRY(BIT(1),FIXED BIN(8),BIT(16),BIT(16),BIT(8)); ECR00050
***** ECR00060
%INCLUDE ENAMEGN;***** DP 00130
/* RANDOM NAME GENERATOR */ DP 00140
12 1 0 DCL NAMEGEN ENTRY(FIXED BIN(15)) RETURNS(BIT(64)); ECR00080
***** ECR00090
/* MISC DCL */ DP 00140
13 1 0 DCL (P1,P2,P3,ID2,P.ID_NEW) POINTER, STR BIT(320); DP 00150
14 1 0 DCL PCATPTR POINTER STATIC EXTERNAL, DP 00160
      IDX PTR CTL, DP 00170
      P_CAT BIT(64) STATIC EXTERNAL; DP 00180
/* INITIALIZE PSET CAT IF NECESSARY */ DP 00190
15 1 0 IF PCATPTR =NULL() DP 00200
      THEN CALL INIT_P; DP 00210
/* GET APPROPRIATE TEMPLATE */ DP 00220
16 1 0 IF SUBSET1='0'B THEN DP 00230
      CALL SEARCH('01'B,P_CAT,P_CAT,IDX,ID2); DP 00240
17 1 0 ELSE CALL SEARCH('01'B,P_CAT,NAME1,IDX,ID2); DP 00250
/* INITIALIZE CAT_ENTRY */ DP 00260
18 1 0 ALLOCATE CAT_ENTRY ; DP 00270
19 1 0 P->CAT_ENTRY.DATA=IDS1->CAT_ENTRY.DATA; DP 00280
/* MODIFY TO REFLECT NEW PSET DEFINITION */ DP 00290
20 1 0 IF SUBSET1='0'B THEN DP 00300
      DO; DP 00310
          CAT_ENTRY.NAME=NAME1; DP 00320
          CAT_ENTRY.KEY_POS=KEY_POS1; DP 00330
          CAT_ENTRY.KEY_LEN=KEY_LEN1; DP 00340
          CAT_ENTRY.DATA_LEN=LEN1; DP 00350
          CAT_ENTRY.L_TYPE=L_TYPE1; DP 00360
          DP 00370
          /* IF HASHED ALLOCATE PTR SLOT FOR OVERFLOW CHAIN */ DP 00380
          IF L_TYPE= HASHED THEN DP 00390
              CALL MAPSET('1'B,2,SP_MAP,AP_MAP,L_POS2); DP 00400
          DP 00410
          /* IF B-TREE ALLOCATE 2 SLOTS FOR RIGHT,LEFT CHAIN */ DP 00420
          ELSE IF L_TYPE =B_TREE THEN DP 00430
              DP 00440
              DP 00450
              DP 00460
              DP 00470
              DP 00480

```

28	1	2	DO;	DP 00490
29	1	2	CALL MAPSET('1'B,2,SP_MAP,AP_MAP,L_POS1);	DP 00500
30	1	2	CALL MAPSET('1'B,2,SP_MAP,AP_MAP,L_POS2);	DP 00510
			END;	DP 00520
			/* IF LINEAR ONLY 1 PTR SLOT USED */	DP 00530
31	1	1	ELSE CALL MAPSET('1'B,1,SP_MAP,AP_MAP,L_POS2);	DP 00540
32	1	1	END;	DP 00550
			/* IF A SUBSET THEN WORK FROM PRIMARY SET CAT ENTRY */	DP 00560
33	1	0	ELSE DO;	DP 00570
			/* IF NO PTR SLOT HAS BEEN ALLOCATED FOR SUBSET	DP 00580
			CHAINING, ALLOCATE A SLOT */	DP 00590
34	1	1	IF S_ID1='0'B THEN	DP 00600
			CALL MAPSET('1'B,1,SP_MAP,AP_MAP,S_ID1);	DP 00610
35	1	1	SUBSET_ID=S_ID1;	DP 00620
			/* IF FIRST SUBSET IN PRIMARY SET ALLOCATE 2 SLOTS,	DP 00630
			1 TO LINK SUBSET CAT WITH PRIMARY CAT, AND 1 TO	DP 00640
			LINK SUBSET CATS */	DP 00650
36	1	1	IF P_CHAIN='0'B THEN	DP 00660
			DO;	DP 00670
37	1	2	CALL MAPSET('1'B,1,SP_MAP,AP_MAP,P_CHAIN);	DP 00680
38	1	2	CALL MAPSET('1'B,1,SP_MAP,AP_MAP,S_CHAIN);	DP 00690
39	1	2	END;	DP 00700
			/* UPDATE PRIMARY CAT ENTRY TO REFLECT ALLOCATIONS */	DP 00710
40	1	1	IDS1->CAT_ENTRY.DATA=P->CAT_ENTRY.DATA;	DP 00720
			/* FINISH DEFINING SUBSET */	DP 00730
41	1	1	CAT_ENTRY.NAME=NAMEGEN(8);	DP 00740
42	1	1	CAT_ENTRY.SUBSET=SUBSET1;	DP 00750
43	1	1	CAT_ENTRY.L_POS2=SUBSET_ID;	DP 00760
44	1	1	L_TYPE=LINEAR;	DP 00770
45	1	1	END;	DP 00780
			/* CREATE ENTRY IN P_SET CATALOG */	DP 00790
46	1	0	STR=STRING(CAT_ENTRY.DATA);	DP 00800
47	1	0	CALL CREATEP(P_CAT,STR,P2);	DP 00810
			/* UPDATE POINTER ARRAYS */	DP 00820
48	1	0	IF L_TYPE = HASHED THEN	DP 00830
			DO;	DP 00840
			/* CREATE SCATTER TABLE AND CHAIN TO CATALOGUE */	DP 00850
49	1	1	CALL CREATE_I(NAME,KEY_LEN,P3);	DP 00860
50	1	1	P2->P_ARRAY(P2->L_POS1)=P3;	DP 00870
51	1	1	END;	DP 00880
			IF SUBSET THEN	DP 00890
52	1	0		DP 00900
				DP 00910
				DP 00920
				DP 00930
				DP 00940
				DP 00950
				DP 00960
				DP 00970

			DO;	DP 00980
			/* UPDATE CATALOGUE CHAINS TO PRIMARY AND OTHER	DP 00990
			SUBSET CATALOGUE ENTRIES */	DP 01000
53	1	1	P2->P_ARRAY(P_CHAIN)=IDS1;	DP 01010
54	1	1	P2->P_ARRAY(S_CHAIN)=IDS1->P_ARRAY(S_CHAIN);	DP 01020
55	1	1	IDS1->P_ARRAY(S_CHAIN)=P2;	DP 01030
56	1	1	END;	DP 01040
				DP 01050
57	1	0	FREE IDS1;	DP 01060
58	1	0	FREE CAT_ENTRY;	DP 01070
59	1	0	RETURN;	DP 01080
				DP 01090
			/*.....	DP 01100
			*/	DP 01110
60	1	0	INIT_P: PROC;	DP 01120
			/*.....	DP 01130
			* THIS MODULE IS RESPONSIBLE FOR INITIALIZING THE PRIMITIVE *	DP 01140
			* LAYER. THIS TASK REQUIRES IT TO INITIALIZE THE PCAT PSET, *	DP 01150
			* USING THE DECLARATION PROVIDED BELOW. *	DP 01160
			*/	DP 01170
				DP 01180
			/* PCAT PSET_CAT ENTRY WITH DESIRED ORGANIZATION */	DP 01190
61	2	0	DCL 1 CAT_ENTRY BASED(P),	DP 01200
			2 LENGTH FIXED BIN(15) INIT(156),	DP 01210
			2 P_ARRAY(16) POINTER,	DP 01220
			2 DATA,	DP 01230
			3 NAME BIT(64),	DP 01240
			3 SP_MAP BIT(16) INIT('0111111111111111'B),	DP 01250
			3 AP_MAP BIT(16) INIT('1111111111111111'B),	DP 01260
			3 NUMFREE BIT(8),	DP 01270
			3 SEARCH_INFO,	DP 01280
			4 L_TYPE BIT(8) INIT('00000001'B),	DP 01290
			4 L_POS1 BIT(8) INIT('00000010'B),	DP 01300
			4 L_POS2 BIT(8) INIT('00000001'B),	DP 01310
			4 KEY_POS BIT(8) INIT('00000001'B),	DP 01320
			4 KEY_LEN BIT(8) INIT('01000000'B),	DP 01330
			3 SET_TYPE,	DP 01340
			4 SUBSET BIT(8) INIT('00000000'B),	DP 01350
			4 SUBSET_ID BIT(8) INIT('00000000'B),	DP 01360
			4 P_CHAIN BIT(8) INIT('00000000'B),	DP 01370
			4 S_CHAIN BIT(8) INIT('00000000'B),	DP 01380
			3 DATA_LEN BIT(16) INIT('0000000011000000'B);	DP 01390
			/* POINTS TO PCAT PSET_CATALOGUE ENTRY */	DP 01400
62	2	0	DCL PCATPTR POINTER STATIC EXTERNAL,	DP 01410
				DP 01420
			/* STRUCTURE OF SCATTER TABLE */	DP 01430
			1 T_INDEX BASED(INDEX_PTR),	DP 01440
			2 NAME BIT(64),	DP 01450
			2 TEST_LEN FIXED BIN(15),	DP 01460

```

                2 PTR_TO_ENTRY(50) POINTER,
                (P, INDEX_PTR) POINTER, STR BIT(240),
                P_CAT BIT(64) STATIC EXTERNAL;

/* START OF PROCEDURE */
63 2 0 ALLOCATE CAT_ENTRY SET(P);
64 2 0 P_CAT=UNSPEC('P_CAT ');
65 2 0 CAT_ENTRY.NAME=P_CAT;
66 2 0 STR=STR NG(CAT_ENTRY.DATA);

/* CREATE BEU CONTAINING CAT ENTRY */
67 2 0 CALL CREATEE(STR,DATA_LEN,ID_NEW);

/* CREATE SCATTER TABLE FOR PSET AND INSERT PTR INTO SLOT */
68 2 0 CALL CREATE_I(CAT_ENTRY.NAME,KEY_LEN,INDEX_PTR);
69 2 0 ID_NEW->P_ARRAY(L_POS1)=INDEX_PTR;

/* UPDATE SCATTER TABLE TO REFLECT NEW ENTRY */
70 2 0 POS=HASH(CAT_ENTRY.NAME,INDEX_PTR->TEST_LEN);
71 2 0 INDEX_PTR->T_INDEX.PTR_TO_ENTRY(POS)=ID_NEW;

/* SET PCATPTR TO ID OF CAT ENTRY */
72 2 0 PCATPTR=ID_NEW;
73 2 0 FREE CAT_ENTRY;
74 2 0 RETURN;
75 2 0 END INIT_P;

/*****
*****/
76 1 0 CREATE_I: PROC(NAME1,LEN,ID_RETURN);
/*****
* THIS MODULE IS RESPONSIBLE FOR CREATING THE SCATTER *
* TABLE USED TO IMPLEMENT HASHING. IT RETURNS A POINTER *
* TO THE SCATTER TABLE THAT IT CREATED. *
*****/

/* STRUCTURE USED FOR SCATTER TABLE */
77 2 0 DCL 1 INDEX_BASED(ID_RETURN),
        2 NAME_ENTRY BIT(64),
        2 TEST_LEN FIXED BIN(15),
        2 PTR_TO_ENTRY(50) POINTER INIT((50) NULL()),

        ID_RETURN POINTER,POS FIXED BIN(15),
        NAME1 BIT(64),LEN BIT(8);

/* ALLOCATE STRUCTURE TO BE USED AS A SCATTER TABLE */
78 2 0 ALLOCATE INDEX SET(ID_RETURN);
79 2 0 INDEX.NAME_ENTRY=NAME1;

```

DP 01470
 DP 01480
 DP 01490
 DP 01500
 DP 01510
 DP 01520
 DP 01530
 DP 01540
 DP 01550
 DP 01560
 DP 01570
 DP 01580
 DP 01590
 DP 01600
 DP 01610
 DP 01620
 DP 01630
 DP 01640
 DP 01650
 DP 01660
 DP 01670
 DP 01680
 DP 01690
 DP 01700
 DP 01710
 DP 01720
 DP 01730
 DP 01740
 DP 01750
 DP 01760
 DP 01770
 DP 01780
 DP 01790
 DP 01800
 DP 01810
 DP 01820
 DP 01830
 DP 01840
 DP 01850
 DP 01860
 DP 01870
 DP 01880
 DP 01890
 DP 01900
 DP 01910
 DP 01920
 DP 01930
 DP 01940
 DP 01950

```
80 2 0      TEST_LEN=LEN;  
81 2 0      RETURN;  
82 2 0 END CREATE_I;  
83 1 0 END DEFINEP;
```

```
DP 01960  
DP 01970  
DP 01980  
DP 01990  
DP 02000
```

1

```

XINCLUDE CREATEP;*****CRP00010
/*****CRE00010
*CRE00020
*CRE00030
*MODULE DESCRIPTION*CRE00040
*****CRE00050
0 CREATEP: PROCEDURECRE00060
      (NAME1, /* BIT(64) */CRE00070
        DATA1, /* BIT(320) */CRE00080
        ID_NEW /* PTR */ );CRE00090
/*****CRE00100
*****PURPOSE:CRE00110
*****THIS PROCEDURE IS RESPONSIBLE FOR INSERTING AN ELEMENTCRE00120
*****CONTAINING THE BIT STRING DATA1 INTO THE PSET NAME1.CRE00130
*****NAME1 MAY BE EITHER A PRIMARY PSET OR A SUBSET OF AN- CRE00140
*****OTHER PSET. IF IT IS A SUBSET, THEN THE ELEMENT MUST BE CRE00150
*****INSERTED INTO BOTH THE SUBSET AND THE PRIMARY PSET.CRE00160
*****CRE00170
*****CRE00180
*****METHOD:CRE00190
*****THIS MODULE INSERTS THE DATA ITEM INTO THE PSET VIA THE CRE00200
*****FOLLOWING STRATEGY:CRE00210
*****A) THE FIRST TASK IS TO RETRIEVE THE P_CAT CATALOGUE CRE00220
*****FOR THE NAME1 PSET. THIS IS ACCOMPLISHED BY CALLING CRE00230
*****SEARCH, WHICH RETURNS A POINTER TO THE BEU CONTAINING CRE00240
*****THE CATALOGUE ENTRY FOR THE PSET. USING A PTR OVER- CRE00250
*****LAY, CAT_ENTRY (A TEMPORARY STRUCTURE) IS OVERLAID CRE00260
*****ON THE BEU. THIS CATALOGUE ENTRY CONTAINS INFORMATION CRE00270
*****NECESSARY TO INSERT THE ELEMENT AND PERFORM CHAINING. CRE00280
*****B) IF THE PSET IS NOT SUBSET, THEN CREATEE IS CALLED, CRE00290
*****PASSING IT DATA1. CREATEE IS THE MODULE WHICH IS CRE00300
*****ACTUALLY RESPONSIBLE FOR CREATING A BEU WHICH CON- CRE00310
*****TAINS A COPY OF DATA1 IN ITS DATA PORTION. CREATEE CRE00320
*****RETURNS A PTR TO THE NEWLY CREATED BEU. THE SEARCH CRE00330
*****ROUTINE IS THEN CALLED, PASSING IT THE NAME OF THE CRE00340
*****PSET AND DATA1. SEARCH RETURNS VIA ID_POS (THE LAST CRE00350
*****PARAMETER) A PTR WHICH POINTS TO THE BEU TO WHICH CRE00360
*****THE NEW BEU SHOULD BE CHAINED. FINALLY, CHAIN ( AN CRE00370
*****INTERNAL PROCEDURE) IS CALLED, PASSING IT POINTERS CRE00380
*****TO THE CATALOGUE ENTRY AND THE NEW BEU. CHAIN IS CRE00390
*****RESPONSIBLE FOR THE ACTUAL INSERTION OF THE BEU INTO CRE00400
*****THE PSET, I.E. CHAINING IF LINEAR OR B_TREE LINK TYPE, CRE00410
*****OTHERWISE, UPDATING PSET'S SCATTER TABLE OR CHAINING CRE00420
*****INTO AN OVERFLOW CHAIN. CRE00430
*****C) IF THE PSET IS A SUBSET OF ANOTHER PSET THEN IT IS CRE00440
*****NECESSARY TO INSERT THE ELEMENT INTO BOTH THE SUBSET & CRE00450

```

```

THE PRIMARY PSET. THE STRATEGY EMPLOYED IS AS FOL-
LOWED:
1) THE PTR VALUE IN THE P_CHAIN PTR SLOT OF THE PSET
CATALOGUE ENTRY IS USED AS A PTR TO THE PRIMARY
PSET CATALOGUE ENTRY.
2) THE SEARCH PROCEDURE IS CALLED, PASSING IT THE
PRIMARY PSET NAME POINTED TO BY THE PTR IN P_CHAIN,
AND DATA1 AS THE KEY. THE PURPOSE HERE IS TO
SEARCH THE PRIMARY PSET TO FIND WHERE TO INSERT
THE BEU CONTAINING DATA1.
3) THE NEXT STEP IS TO CALL CREATEE, PASSING IT DATA1.
AS DESCRIBED ABOVE CREATEE CREATES A BEU CONTAINING
DATA1 AND RETURNS A POINTER TO THE NEW BEU.
CREATEE AS DESCRIBED ABOVE.
4) CHAIN IS THEN CALLED TO INSERT THE BEU CREATED BY
CREATEE INTO THE PRIMARY PSET, USING THE VALUE OF
ID_POS RETURNED BY THE PREVIOUS CALL TO SEARCH.
5) SEARCH IS CALLED AGAIN, THIS TIME TO FIND THE VALUE
OF ID_POS WITHIN THE SUBSET (I.E. WHICH ELEMENT
WITHIN THE SUBSET TO CHAIN THE NEW ELEMENT), AND
THEN CHAIN IS CALLED TO INSERT THE BEU INTO THE
SUBSET.

*****
***** INPUT PARAMETERS:
***** NAME1 - NAME OF PSET INTO WHICH ELEMENT IS TO BE INSERTED
***** DATA1 - BIT STRING TO BE INSERTED
***** ID_NEW - NOT SIGNIFICANT ON INPUT.
*****
***** OUTPUT PARAMETERS:
***** ID_NEW - POINTER TO BEU CREATED.
*****
***** CALLS PROCEDURES:
***** SEARCH, CREATEE, CHAIN (INTERNAL)
*****
*****/
*****
%INCLUDE PSETCAT;*****
/* PSET_CAT TEMPLATE */
DCL 1 CAT_ENTRY BASED(P), /* BASED ON ID OF PSET_CAT BEU */
2 LENGTH FIXED BIN(15), /* LENGTH OF CAT ENTRY */
2 P_ARRAY(16) POINTER, /* PTR ARRAY FOR LINKING */
2 DATA, /* INFO ON PSET ORGANIZATION */
3 NAME BIT(64), /* NAME OF PSET */
( 3 SP_MAP, /* MAP OF POINTER ARRAY, */
3 AP_MAP ) BIT(16), /* GIVING STATUS OF P_SLOTS */

```

```

3 NUMFREE BIT(8),      /* NOT USED */          BCA00210
3 SEARCH_INFO,         /* LINKAGE INFORMATION */      BCA00220
  ( 4 L_TYPE,          /* TYPE OF LINK (HASHED ETC..) */ BCA00230
    4 L_POS1,          /* PTR SLOT USED FOR CHAINING */ BCA00240
    4 L_POS2,          /* ADDITIONAL PTR SLOT FOR LINK */ BCA00250
    4 KEY_POS,         /* STARTING POSITION OF KEY */   BCA00260
    4 KEY_LEN ) BIT(8), /* LENGTH OF KEY */         BCA00270
3 SET_TYPE,           /* SET TYPE INFO */          BCA00280
  ( 4 SUBSET,          /* IF PRIMARY OR SUBSET */    BCA00290
    4 SUBSET_ID,       /* PTR SLOT FOR SUBSET LINK */ BCA00300
    4 P_CHAIN,         /* PTR SLOT PTS TO PRIMARY DCL */ BCA00310
    4 S_CHAIN ) BIT(8), /* SUBSET DCL CHAIN */       BCA00320
3 DATA_LEN BIT(15);   /* LENGTH OF ELEMENTS */     BCA00330
*****
3 1 0      DCL P_CAT BIT(64) STATIC EXTERNAL;      CRP00030
                                           CRP00040
                                           CRP00050
%INCLUDE IDS1;*****CRP00060
/* POINTER STACK RETURNED BY SEARCH */          BCA00420
4 1 0      DCL IDS1 PTR EXTERNAL CONTROLLED;      BCA00430
*****
                                           CRP00060
                                           CRP00070
%INCLUDE PSETSYM;*****CRP00080
/* PSET LINK TYPES */                          BCA00540
5 1 0      DCL HASHED BIT(8) INIT('00000001'B),  BCA00550
           B_TREE BIT(8) INIT('00000010'B),      BCA00560
           LINEAR BIT(8) INIT('00000100'B);      BCA00570
*****
/* PROCEDURES CALLED BY CREATEP */              CRP00090
%INCLUDE ECREATEF;*****CRP00100
/* IEU CREATION MODULE */                      ECR00020
6 1 0      DCL CREATEE ENTRY(BIT(320),BIT(16),POINTER); ECR00030
*****
                                           CRP00100
%INCLUDE ESEARCH;*****CRP00110
/* SEARCH MODULE */                            BCA00700
7 1 0      DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER); BCA00710
                                           BCA00720
                                           CRP00110
%INCLUDE EHASH;*****CRP00120
/* HASHING MODULE */                          DEC00020
8 1 0      DCL HASH ENTRY(BIT(64),FIXED BIN(15)) RETURNS(FIXED BIN(15)); DEC00030
*****
                                           CRP00120
%INCLUDE EMAPSET;*****CRP00130
/* MAP MAINTENANCE MODULE */                  ECR00050
9 1 0      DCL MAPSET ENTRY(BIT(1),FIXED BIN(8),BIT(16),BIT(16),BIT(8)); ECR00060
*****
                                           CRP00130
%INCLUDE EPRINTP;*****CRP00140
/* DIAGNOSTIC PRINT MODULE */                  DEC00050
10 1 0     DCL PRINTP ENTRY(POINTER);           DEC00060
*****
                                           CRP00140

```


11	1	0	/* MISC DECLARATIONS */ DCL NAME1 BIT(64), DATA1 BIT(*), (ID_RETURN, ID_POS, P, P_PTR, ID_NEW) POINTER, IDXX POINTER CONTROLLED ;	CRP00150 CRP00160 CRP00170 CRP00180 CRP00190 CRP00200 CRP00210 CRP00220 CRP00230 CRP00240 CRP00250 CRP00260
12	1	0	/* GET PTR TO PCAT CATALOGUE ENTRY FOR PSET */	CRP00270
13	1	0	CALL SEARCH('01'B, P_CAT, NAME1, IDXX, ID_POS);	CRP00280
14	1	0	P=IDS1;	CRP00290
15	1	0	FREE IDS1;	CRP00300
			ID_TO_DATE=ALLOCATION(IDS1);	CRP00310
16	1	0	/* IF A PRIMARY SET CREATE NEW ELEMENT AND CHAIN IT INTO SET */	CRP00320
			IF SUBSET='0'B THEN	CRP00330
17	1	1	DO;	CRP00340
18	1	1	CALL CREATEE(DATA1, DATA_LEN, ID_NEW);	CRP00350
19	1	1	CALL SEARCH('01'B, NAME1, DATA1, IDXX, ID_POS);	CRP00360
20	1	1	CALL CHAIN(P, ID_NEW);	CRP00370
			/* ADJUST IDS1 TO REFLECT ANY DUPLICATES FOUND */	CRP00380
21	1	1	IF ALLOCATION(IDS1)>ID_TO_DATE	CRP00390
			THEN FREE IDS1;	CRP00400
			END;	CRP00410
22	1	0	/* IF A SUBSET INSERT INTO BOTH SUBSET AND PRIMARY SET */	CRP00420
			ELSE DO;	CRP00430
23	1	1	/* LOCATE POSITION WHERE TO INSERT ELEMENT IN PRIMARY */	CRP00440
24	1	1	P_PTR=P_ARRAY(P_CHAIN);	CRP00450
			CALL SEARCH('01'B, P_PTR->NAME, DATA1, IDXX, ID_POS);	CRP00460
25	1	1	/* ADJUST IDS1 TO REFLECT ANY DUPLICATES FOUND */	CRP00470
			IF ALLOCATION(IDS1)>ID_TO_DATE THEN	CRP00480
			FREE IDS1;	CRP00490
			/* CREATE BEU CONTAINING ELEMENT AND CHAIN INTO PRIMARY	CRP00500
26	1	1	SET */	CRP00510
27	1	1	CALL CREATEE(DATA1, DATA_LEN, ID_NEW);	CRP00520
			CALL CHAIN(P_PTR, ID_NEW);	CRP00530
28	1	1	/* LOCATE POSITION IN SUBSET, ADJUST IDS1, AND CHAIN ELEMENT	CRP00540
29	1	1	INTO SUBSET */	CRP00550
			CALL SEARCH('01'B, NAME1, DATA1, IDXX, ID_POS);	CRP00560
30	1	1	IF ALLOCATION(IDS1)>ID_TO_DATE	CRP00570
			THEN FREE IDS1;	CRP00580
			CALL CHAIN(P, ID_NEW);	CRP00590
31	1	1	END;	CRP00600
32	1	0	RETURN;	CRP00610
			/* *****	CRP00620
			***** */	CRP00630

```

33  1  0  CHAIN:  PROC(CAT_PTR,DATA_PTR);                                CRP00640
/*.....*/                                                                CRP00650
*   THIS PROCEDURE IS RESPONSIBLE FOR CHAINING ELEMENTS INTO          * CRP00660
*   PREVIOUSLY DEFINED PSETS, GIVEN CAT_PTR WHICH POINTS TO THE      * CRP00670
*   PSET CAT ENTRY, DATA_PTR WHICH POINTS TO THE BEU TO BE         * CRP00680
*   INSERTED, AND ID_POS ( A GLOBAL VARIABLE ) WHICH POINTS TO THE * CRP00690
*   BEU TO WHICH THIS BEU SHOULD BE CHAINED.                        * CRP00700
/*.....*/                                                                CRP00710
34  2  0          DCL (CAT_PTR,DATA_PTR,INDEX_PTR) POINTER;          CRP00720
35  2  0          DCL  1 T_INDEX BASED(INDEX_PTR),                   CRP00730
                      2 NAME BIT(64),                                CRP00740
                      2 TEST_LEN FIXED BIN(15),                      CRP00750
                      2 PTR_TO_ENTRY(50) POINTER;                    CRP00760
36  2  0          DCL POS FIXED BIN(15);                             CRP00770
37  2  0          DCL  1 T_ELEMENT BASED(DATA_PTR),                  CRP00780
                      2 LENGTH FIXED BIN(15),                        CRP00790
                      2 P_ARRAY(16) POINTER,                          CRP00800
                      2 INFO BIT(320);                                CRP00810
38  2  0          /* IF FIRST ELEMENT IN PSET */                     CRP00820
                      IF ID_POS = NULL()                             CRP00830
                      THEN DO;                                       CRP00840
39  2  1              IF CAT_PTR->L_TYPE = HASHED                     CRP00850
                      THEN DO;                                       CRP00860
40  2  2                  /* UPDATE APPROPRIATE SCATTER TABLE ENTRY */ CRP00870
41  2  2                  INDEX_PTR=CAT_PTR->P_ARRAY(CAT_PTR->L_POS1); CRP00880
42  2  2                  POS=HASH(DATA_PTR->T_ELEMENT.INFO,TEST_LEN); CRP00890
43  2  2                  PTR_TO_ENTRY(POS)=DATA_PTR;               CRP00900
                      END;                                           CRP00910
                      /* OTHERWISE CHAIN TO CATALOGUE ENTRY */       CRP00920
44  2  1                  ELSE CAT_PTR->P_ARRAY(CAT_PTR->L_POS2)=DATA_PTR; CRP00930
45  2  1                  RETURN;                                     CRP00940
46  2  1              END;                                           CRP00950
                      /* NOT FIRST ELEMENT IN PSET */               CRP00960
47  2  0          ELSE IF CAT_PTR->L_TYPE=B_TREE                     CRP00970
                      THEN DO;                                       CRP00980
                      /* IF ELEMENT IS LESS THAN ELEMENT POINTED TO BY CRP00990
                      ID_POS,CHAIN AS A LEFT DESCENDANT */           CRP01000
48  2  1          IF SUBSTR(ID_POS->T_ELEMENT.INFO,CAT_PTR->KEY_POS, CRP01010
                      CAT_PTR->KEY_LEN) > SUBSTR(DATA_PTR->T_ELEMENT.INFO, CRP01020
                      CAT_PTR->KEY_POS,CAT_PTR->KEY_LEN) THEN         CRP01030
                      ID_POS->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS1)=DATA_PTR; CRP01040
                      CRP01050
                      CRP01060
                      CRP01070
                      CRP01080
                      CRP01090
                      CRP01100
                      CRP01110
                      CRP01120

```

			/* OTHERWISE IT MUST BE A RIGHT DESCENDENT, AND CHAIN	CRP01130
			ACCORDINGLY */	CRP01140
49	2	1	ELSE DO;	CRP01150
50	2	2	DATA_PTR->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2)=	CRP01160
			ID_POS->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2);	CRP01170
51	2	2	ID_POS->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2)	CRP01180
			= DATA_PTR;	CRP01190
52	2	2	END;	CRP01200
53	2	1	END;	CRP01210
			/* IF HASHED OR LINEAR CHAIN TO ELEMENT POINTED TO BY	CRP01220
			ID_POS */	CRP01230
54	2	0	ELSE DO;	CRP01240
55	2	1	DATA_PTR->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2)=	CRP01250
			ID_POS->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2);	CRP01260
56	2	1	ID_POS->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2)=DATA_PTR;	CRP01270
57	2	1	END;	CRP01280
58	2	0	RETURN;	CRP01290
59	2	0	END CHAIN;	CRP01300
				CRP01310
				CRP01320
60	1	0	END CREATEP;	CRP01330
				CRP01340

1

```

%INCLUDE SEARCH;.....SEA00010
/.....FOR00010
*.....FOR00020
*.....FOR00030
*.....MODULE      DESCRIPTION      *.....FOR00040
*.....FOR00050
0 SEARCH: PROCEDURE
    (MODE,          /* BIT(2) */      FOR00060
      PSET_NAME,    /* BIT(64) */     FOR00070
      DATA1,       /* BIT(*) */      FOR00080
      IDXX,         /* PTR */         FOR00090
      ID_CHAIN      /* PTR */         FOR00100
    );
/.....FOR00110
.....PURPOSE:
.....THIS MODULE IS RESPONSIBLE FOR RETRIEVING THE IDS OF 1 FOR00120
.....OR MORE ELEMENTS WITHIN THE PSET PSET_NAME, GIVEN THE FOR00130
.....KEY SPECIFIED BY DATA1, AND THE SEARCH MODE SPECIFIED BY FOR00140
.....MODE. IT RETURNS THE IDS OF THE ELEMENTS FOUND IN A CTL FOR00150
.....STRUCTURE CALLED IDS1. FOR00160
.....FOR00170
.....FOR00180
.....FOR00190
.....METHOD:
.....FOR00200
.....FOR00210
.....THE SEARCH METHOD DEPENDS ON BOTH THE PSET ORGANIZATION FOR00220
.....(HASHED, B_TREE OR LINEAR), AND THE SEARCH MODE ( FIRST FOR00230
.....ELEMENT WHICH MATCHES KEY, ALL ELEMENTS WHICH MATCH KEY, FOR00240
.....OR ALL ELEMENTS WITHIN THE SET ). GENERAL LOGIC IS AS FOR00250
.....FOLLOWS: FOR00260
.....A) RETRIEVES PSET_CAT ENTRY FOR PSET IDENTIFIED BY FOR00270
.....PSET_NAME. THIS IS DONE VIA A HASH SEARCH OF THE FOR00280
.....PCAT_PSET. VIA A STRING OVERLAY, CAT_ENTRY IS OVERLAID FOR00290
.....ON THE BEU CONTAINING THE PSET_CAT ENTRY FOR THE PSET. FOR00300
.....B) IF THE SEARCH MODE IS FIRST, OR ALL ELEMENTS WHICH FOR00310
.....MATCH KEY, H_SEARCH, B_SEARCH, OR L_SEARCH IS CALLED FOR00320
.....DEPENDING ON THE PSET ORGANIZATION. THESE ARE INTERNAL FOR00330
.....ROUTINES WHICH RETURN 2 POINTERS, ID1 WHICH IS A POINTER FOR00340
.....TO THE FIRST OCCURENCE OF THE ELEMENT, AND ID_CHAIN FOR00350
.....WHICH IS A PTR TO THE LAST OCCURENCE OF THE ELEMENT FOR00360
.....FOUND. (IF NO ELEMENT WAS FOUND WHICH FOR00370
.....MATCHED THE KEY, ID_CHAIN POINTS TO THE BEU TO WHICH FOR00380
.....THE ELEMENT SHOULD BE CHAINED). IDS1 IS SET EQUAL TO FOR00390
.....THE POINTER FOUND FOR00400
.....C) IF ALL ELEMENTS WHICH MATCH KEY WAS SPECIFIED, THEN FOR00410
.....A LINEAR SEARCH IS INVOKED STARTING AT THE FIRST FOR00420
.....FIRST MATCH AND GOING UNTIL NO OTHER MATCHES ARE POSSIBLE. FOR00430
.....IF LINEAR ORGANIZATION THIS MEANS UNTIL THE END OF THE SET FOR00440
.....IF HASHED, TO THE END OF THE OVERFLOW CHAIN. IF B_TREE FOR00450

```

```

***** UNTIL THE RIGHT DESCENDENTS NO LONGER MATCH. EVERY ID FOR00460
***** RETURNED IS PLACED ON THE IDS1 STACK. FOR00470
***** D) IF THE IDS OF ALL ELEMENTS IN THE SET ARE TO BE FETCHED FOR00480
***** 3 OTHER ROUTINES ARE CALLED LINEAR_H, LINEAR_B, AND FOR00490
***** LINEAR_L. LINEAR_H PERFORMS A LINEAR SEARCH OF THE SET'S FOR00500
***** SCATTER TABLE AND OVERFLOW CHAINS, AND RETURNS THE IDS OF FOR00510
***** ALL ELEMENTS FOUND. LINEAR_B PERFORMS AN IN_ORDER FOR00520
***** TRAVERSAL OF A BINARY TREE, AND RETURNS THE IDS FOUND. FOR00530
***** LINEAR_L PERFORMS A SIMPLE LINEAR SEARCH. FOR00540
***** ..... FOR00550
***** INPUT PARAMETERS: FOR00560
***** MODE - SPECIFIES THE SEARCH MODE: FOR00570
***** '01'B - FIRST ELEMENT WHICH MATCHES KEY. FOR00580
***** '10'B - ALL ELEMENTS WHICH MATCH KEY. FOR00590
***** '11'B - ALL ELEMENTS IN SET. FOR00600
***** PSET_NAME - NAME OF PSET TO BE SEARCHED FOR00610
***** DATA1 - KEY TO SEARCH ON, IF MODE = '11'B THEN FOR00620
***** NOT USED. FOR00630
***** IDXX - NOT CURRENTLY USED. FOR00640
***** ID_CHAIN - NOT USED ON INPUT FOR00650
***** ..... FOR00660
***** ..... FOR00670
***** OUTPUT PARAMETERS: FOR00680
***** IDS1 - A CONTROLLED STACK OF POINTERS CORRESPONDING FOR00690
***** TO THE RETRIEVED IDS. FOR00700
***** ID_CHAIN - A PTR VALUE WHICH POINTS TO BEU TO WHICH FOR00710
***** ELEMENT SHOULD BE CHAINED IF NOT FOUND, FOR00720
***** OTHERWISE POINTS TO THE LAST OCCURENCE FOUND FOR00730
***** DURING SEARCH. FOR00740
***** ..... FOR00750
***** ..... FOR00760
***** CALLS PROCEDURES: FOR00770
***** HASH, (H_SEARCH, L_SEARCH, B_SEARCH, LINEAR_H, LINEAR_B FOR00780
***** LINEAR_L) INTERNAL FOR00790
***** ..... FOR00800
***** ..... SEA00010
***** ..... SEA00020
***** ..... SEA00030
***** ..... SEA00040
***** ..... BCA00130
/* PSET_CAT TEMPLATE */
%INCLUDE PSETCAT:..... BCA00140
/* PSET_CAT TEMPLATE */ BCA00150
DCL 1 CAT_ENTRY BASED(P), /* BASED ON ID OF PSET_CAT BEU */ BCA00160
2 LENGTH FIXED BIN(15), /* LENGTH OF CAT ENTRY */ BCA00170
2 P_ARRAY(16) POINTER, /* PTR ARRAY FOR LINKING */ BCA00180
2 DATA, /* INFO ON PSET ORGANIZATION */ BCA00190
3 NAME BIT(64), /* NAME OF PSET */ BCA00200
( 3 SP_MAP, /* MAP OF POINTER ARRAY, */ BCA00210
3 AP_MAP ) BIT(16), /* GIVING STATUS OF P_SLOTS */ BCA00220
3 NUMFREE BIT(8), /* NOT USED */
3 SEARCH_INFO, /* LINKAGE INFORMATION */

```

2 1 0

```

( 4 L_TYPE,          /* TYPE OF LINK (HASHED ETC..) *//BCA00230
 4 L_POS1,          /* PTR SLOT USED FOR CHAINING *//BCA00240
 4 L_POS2,          /* ADDITIONAL PTR SLOT FOR LINK *//BCA00250
 4 KEY_POS,         /* STARTING POSITION OF KEY *//BCA00260
 4 KEY_LEN ) BIT(8), /* LENGTH OF KEY *//BCA00270
3 SET_TYPE,         /* SET TYPE INFO *//BCA00280
( 4 SUSSET,         /* IF PRIMARY OR SUBSET *//BCA00290
 4 SUSSET_ID,       /* PTR SLOT FOR SUBSET LINK *//BCA00300
 4 P_CHAIN,         /* PTR SLOT PTS TO PRIMARY DCL *//BCA00310
 4 S_CHAIN ) BIT(8), /* SUBSET DCL CHAIN *//BCA00320
3 DATA_LEN BIT(15); /* LENGTH OF ELEMENTS *//BCA00330
*****
%INCLUDE BEU;*****SEA00040
/* BEU TEMPLATE *//BCA00350
3 1 0 DCL 1 T_ELEMENT BASED(Q),BCA00360
      2 LENGTH FIXED BIN(15),BCA00370
      2 P_ARRAY(16) PTR,BCA00380
      2 INFO,BCA00390
      3 DATA BIT(320);BCA00400
*****SEA00050
/* MISC DECLARATIONS *//SEA00060
4 1 0 DCL MODE BIT(2),SEA00070
      FIRST BIT(2) INIT('01'B),SEA00080
      ALL_SET BIT(2) INIT('11'B),SEA00090
      ALL_SAME BIT(2) INIT('10'B);SEA00100
%INCLUDE PSET5.M;*****SEA00110
/* PSET LINK TYPES *//BCA00540
5 1 0 DCL HASHED BIT(8) INIT('00000001'B),BCA00550
      B_TREE BIT(8) INIT('00000010'B),BCA00560
      LINEAR BIT(8) INIT('00000100'B);BCA00570
*****SEA00110
/* POINTS TO PCAT ENTRY FOR PCAT PSET DESCRIPTION *//SEA00120
6 1 0 DCL PCATPTR POINTER EXTERNAL STATIC;SEA00130
SEA00140
SEA00150
7 1 0 DCL (P,Q, ID_CHAIN, ID1, ID_START ) POINTER;SEA00160
SEA00170
SEA00180
/* POINTER STACK RETURNED BY SEARCH *//SEA00190
%INCLUDE IDS1;*****SEA00200
/* POINTER STACK RETURNED BY SEARCH *//BCA00420
8 1 0 DCL IDS1 PTR EXTERNAL CONTROLLED;BCA00430
*****SEA00200
SEA00210
/* MISC DECLARATIONS *//SEA00220
9 1 0 DCL IDXX POINTER CONTROLLED;SEA00230
10 1 0 DCL T_DATA1 BIT(128) VARYING;SEA00240
11 1 0 DCL P_CAT BIT(64) STATIC EXTERNAL,SEA00250
      PSE_NAME BIT(34), DATA1 BIT(*),SEA00260

```

```

/* EXTERNAL PROCEDURES CALLED */
(PRINTP,PRINTIT) EXTERNAL ENTRY;
%INCLUDE EHASH;*****
/* HASHING MODULE */
12 1 0 DCL HASH ENTRY(BIT(64),FIXED BIN(15)) RETURNS(FIXED BIN(15));
*****

/* GET ID FOR PSET_CAT ENTRY FOR PSET */
13 1 0 P=PCATPT;
14 1 0 CALL H_SEARCH(PSET_NAME,ID1,ID_CHAIN);

/* OVERLAY CAT_ENTRY ON PSET_CAT BEU FOR PSET */
15 1 0 P=ID1;

/* EXTRACT KEY FIELD FROM DATA STRING PASSED TO SEARCH */
16 1 0 T_DATA1=SUBSTR(DATA1,KEY_POS,KEY_LEN);

/* DISPATCH APPROPRIATE INTERNAL SEARCH ROUTINE */
17 1 0 IF MODE=FIRST ;MODE=ALL_SAME
      THEN DO;
18 1 1   IF L_TYPE=HASHED THEN
          CALL H_SEARCH(T_DATA1,ID1,ID_CHAIN);
19 1 1   ELSE IF L_TYPE=B_TREE THEN
          CALL B_SEARCH(T_DATA1,ID1,ID_CHAIN);
20 1 1   ELSE CALL L_SEARCH(T_DATA1,CAT_ENTRY.P_ARRAY(L_POS2)
          ,ID1,ID_CHAIN);

      /* IF VALUE FOUND PUT POINTER ON STACK */
21 1 1   IF ID1=NULL() THEN
          DO;
22 1 2       ALLOCATE IDS1;
23 1 2       IDS1=ID1;
24 1 2       END;

      /* OTHERWISE RETURN */
25 1 1   ELSE RETURN;

26 1 1   IF MODE=ALL_SAME THEN
          /* DO LINEAR SEARCH FOR REMAINING ELEMENTS */
          DO WHILE(ID1=NULL());
          /* SET START TO NEXT BEU AFTER LAST MATCH */
          ID_START=ID1->T_ELEMENT.P_ARRAY(L_POS2);
          CALL L_SEARCH(T_DATA1,ID_START,ID1,ID_CHAIN);
          IF ID1=NULL() THEN
              /* IF MATCH FOUND PUT ID ON STACK */
              DO;
              ALLOCATE IDS1;
              IDS1=ID1;

```

```

SEA00270
SEA00280
SEA00290
DEC00020
DEC00030
SEA00290
SEA00300
SEA00310
SEA00320
SEA00330
SEA00340
SEA00350
SEA00360
SEA00370
SEA00380
SEA00390
SEA00400
SEA00410
SEA00420
SEA00430
SEA00440
SEA00450
SEA00460
SEA00470
SEA00480
SEA00490
SEA00500
SEA00510
SEA00520
SEA00530
SEA00540
SEA00550
SEA00560
SEA00570
SEA00580
SEA00590
SEA00600
SEA00610
SEA00620
SEA00630
SEA00640
SEA00650
SEA00660
SEA00670
SEA00680
SEA00690
SEA00700
SEA00710
SEA00720

```

32	1	3		ID_START=ID1;	SEA00730
33	1	3		END;	SEA00740
34	1	2		END;	SEA00750
35	1	1		RETURN;	SEA00760
36	1	1		END;	SEA00770
				/* IF ALL OF SET TO BE RETRIEVED DISPATCH APPROPRIATE ROUTINE */	SEA00780
37	1	0		IF MODE=ALL_SET THEN	SEA00790
				DO;	SEA00800
38	1	1		IF L_TYPE =HASHED THEN	SEA00810
				CALL LINEAR_H(IDS1);	SEA00820
39	1	1		ELSE IF L_TYPE=B_TREE THEN	SEA00830
				CALL LINEAR_B(IDS1);	SEA00840
40	1	1		ELSE CALL LINEAR_L(CAT_ENTRY.P_ARRAY(L_POS2),IDS1);	SEA00950
41	1	1		END;	SEA00860
42	1	0		RETURN;	SEA00870
				/* UTILITY SUBROUTINES USED BY SEARCH */	SEA00880
43	1	0	L_SEARCH:	PROC(T_DATA1,START,ID1A,ID1B);	SEA00890
44	2	0		DCL T_DATA1 BIT(*),	SEA00900
				(START,ID1A,ID1B) POINTER;	SEA00910
				ID1B=NULL();	SEA00920
45	2	0		/* FOLLOW POINTER CHAIN LINKING ELEMENTS IN PSET */	SEA00930
46	2	0		DO ID1A=START REPEAT ID1A->T_ELEMENT.P_ARRAY(L_POS2)	SEA00940
				WHILE(ID1A=NULL());	SEA00950
47	2	1		ID1B=ID1A;	SEA00960
48	2	1		IF T_DATA1=SUBSTR(ID1A->T_ELEMENT.DATA,KEY_POS,KEY_LEN)	SEA00970
				THEN RETURN;	SEA00980
49	2	1		END;	SEA00990
50	2	0		RETURN;	SEA01000
51	2	0	END L_SEARCH;		SEA01010
52	1	0	H_SEARCH:	PROC(T_DATA1,ID1A,ID1B);	SEA01020
53	2	0		DCL T_DATA1 BIT(*), (ID1A,ID1B) POINTER,	SEA01030
				Q POINTER, INDEX_POS FIXED BIN,	SEA01040
				/* TEMPLATE FOR SCATTER TABLE */	SEA01050
				1 T_INDEX BASED(Q),	SEA01060
				2 NAME BIT(64),	SEA01070
				2 L_HASH FIXED BIN(15),	SEA01080
				2 PTR_TO_ENTRY(50) POINTER;	SEA01090
54	2	0		ID1B=NULL();	SEA01100
				/* HASH INTO SCATTER TABLE USING KEY, AND LENGTH IN T_INDEX */	SEA01110
55	2	0		INDEX_POS=HASH(T_DATA1,P->CAT_ENTRY.P_ARRAY(L_POS1)->L_HASH);	SEA01120
				/* CHECK BEU POINTED TO BY T-INDEX ENTRY, IF NO MATCH FOLLOW	SEA01130
				OVERFLOW CHAIN */	SEA01140
					SEA01150
					SEA01160
					SEA01170
					SEA01180
					SEA01190
					SEA01200
					SEA01210

56	2	0	DO ID1A=CAT_ENTRY.P_ARRAY(L_POS1)->PTR_TO_ENTRY(INDEX_POS)	SEA01220
			REPEAT ID1A->T_ELEMENT.P_ARRAY(L_POS2)	SEA01230
			WHILE (ID1A=NULL());	SEA01240
57	2	1	ID1B=ID1A;	SEA01250
58	2	1	IF T_DATA1=SUBSTR(ID1A->T_ELEMENT.DATA,KEY_POS,KEY_LEN)	SEA01260
			THEN RETURN;	SEA01270
59	2	1	END;	SEA01280
60	2	0	RETURN;	SEA01290
61	2	0	END H_SEARCH;	SEA01300
				SEA01310
62	1	0	B_SEARCH: PROC(T_DATA1, ID1A, ID1B);	SEA01320
63	2	0	DCL T_DATA1 BIT(*), (ID1A, ID1B) POINTER;	SEA01330
64	2	0	ID1B=NULL();	SEA01340
				SEA01350
65	2	0	ID1A=CAT_ENTRY.P_ARRAY(L_POS2);	SEA01360
66	2	0	DO WHILE (ID1A=NULL());	SEA01370
67	2	1	ID1B=ID1A;	SEA01380
68	2	1	IF T_DATA1=SUBSTR(ID1A->T_ELEMENT.DATA,KEY_POS,KEY_LEN)	SEA01390
			THEN RETURN;	SEA01400
69	2	1	ELSE IF SUBSTR(ID1A->T_ELEMENT.DATA,KEY_POS,KEY_LEN)>T_DATA1	SEA01410
			THEN ID1A=ID1A->T_ELEMENT.P_ARRAY(L_POS1);	SEA01420
			ELSE ID1A=ID1A->T_ELEMENT.P_ARRAY(L_POS2);	SEA01430
70	2	1		SEA01440
71	2	1	END;	SEA01450
72	2	0	RETURN;	SEA01460
73	2	0	END B_SEARCH;	SEA01470
				SEA01480
74	1	0	LINEAR_L: PROC(START, IDS11);	SEA01490
75	2	0	DCL (ID, START) POINTER, IDS11 POINTER CONTROLLED;	SEA01500
76	2	0	DO ID=START REPEAT ID->T_ELEMENT.P_ARRAY(L_POS2)	SEA01510
			WHILE (ID=NULL());	SEA01520
77	2	1	ALLOCATE IDS11;	SEA01530
78	2	1	IDS11=ID;	SEA01540
79	2	1	END;	SEA01550
80	2	0	RETURN;	SEA01560
81	2	0	END LINEAR_L;	SEA01570
				SEA01580
82	1	0	LINEAR_H: PROC(IDS11);	SEA01590
83	2	0	DCL IDS11 POINTER CONTROLLED, ID POINTER,	SEA01600
			1 T_INDEX BASED(Q),	SEA01610
			2 NAME BIT(64),	SEA01620
			2 HASH_LEN FIXED BIN(15),	SEA01630
			2 PTR_TO_ENTRY(50) POINTER,	SEA01640
			Q POINTER;	SEA01650
84	2	0	Q=CAT_ENTRY.P_ARRAY(L_POS1);	SEA01660
85	2	0	DO J=1 TO 50;	SEA01670
86	2	1	DO ID=Q->PTR_TO_ENTRY(J) REPEAT	SEA01680
			ID->T_ELEMENT.P_ARRAY(L_POS2)	SEA01690
			WHILE (ID=NULL());	SEA01700
87	2	2	ALLOCATE IDS11;	

88	2	2	IDS11=ID;	SEA01710
89	2	2	END;	SEA01720
90	2	1	END;	SEA01730
91	2	0	RETURN;	SEA01740
92	2	0	END LINEAR_H;	SEA01750
93	1	0	LINEAR_B: PROC(IDS11);	SEA01760
94	2	0	DCL (IDS11, ITEMP) POINTER CONTROLLED,	SEA01770
			Q POINTER, FLAG BIT(1);	SEA01780
95	2	0	Q=DAT_E TRY.P_ARRAY(L_POS2);	SEA01790
96	2	0	FLAG='1 B;	SEA01800
			/* DO A INORDER DEPTH FIRST TRAVERSAL */	SEA01810
97	2	0	DO WHILE (FLAG);	SEA01820
			/* GET TO BOTTOM ELEMENT IN BRANCH */	SEA01830
98	2	1	DO WHILE(Q~=NULL());	SEA01840
			/* TEMPORARY STACK FOR NODES PASSED ON WAY DOWN */	SEA01850
99	2	2	ALLOCATE ITEMP;	SEA01860
100	2	2	ITEMP=Q;	SEA01870
101	2	2	Q=Q->T_ELEMENT.P_ARRAY(L_POS1);	SEA01880
102	2	2	END;	SEA01890
			/* WORK YOUR WAY BACK UP BRANCH */	SEA01900
103	2	1	IF ALLOCATION(ITEMP)~=0 THEN	SEA01910
			DO;	SEA01920
104	2	2	ALLOCATE IDS11;	SEA01930
105	2	2	IDS11=ITEMP;	SEA01940
106	2	2	Q=ITEMP->T_ELEMENT.P_ARRAY(L_POS2);	SEA01950
107	2	2	FREE ITEMP;	SEA01960
108	2	2	END;	SEA01970
109	2	1	ELSE RETURN;	SEA01980
110	2	1	END;	SEA01990
111	2	0	END LINEAR_B;	SEA02000
112	1	0	END SEARCH;	SEA02010
				SEA02020
				SEA02030
				SEA02040

```

%INCLUDE FETCH;*****SEA02060
/*****SEA02070
* FET00010
* FET00020
* FET00030
* FET00040
* FET00050
* FET00060
* FET00070
* FET00080
* FET00090
* FET00100
* FET00110
* FET00120
* FET00130
* FET00140
* FET00150
* FET00160
* FET00170
* FET00180
* FET00190
* FET00200
* FET00210
* FET00220
* FET00230
* FET00240
* FET00250
* FET00260
* FET00270
* FET00280
* FET00290
* FET00300
* FET00310
* FET00320
* FET00330
* FET00340
* FET00350
* FET00360
* FET00370
* FET00380
* FET00390
* FET00400
* FET00410
* FET00420
* FET00430
* FET00440

1 0  FETCH:  PROCEDURE
        (MODE1,      /* BIT(2) */
         ID,          /* PTR */
         NAME,        /* BIT(64) */
         KEY,         /* BIT(*) */
         FND          /* BIT(1) */ );

/*****
*****  PURPOSE:
*****  THIS MODULE IS RESPONSIBLE FOR RETRIEVING THE DATA POR-
*****  OF BEUS CONTAINED WITHIN A PARTICULAR PSET.  DEPENDING
*****  ON THE MODE, IT WILL EITHER RETURN ALL OF THE ELEMENTS
*****  IN A PSET, OR JUST THE FIRST ELEMENT WHICH MATCHES THE
*****  KEY, OR ALL OF THE ELEMENTS IN THE SET WHICH MATCH THE
*****  KEY.  IT RETURNS A CONTROLLED STACK OF BIT STRINGS COR-
*****  RESPONDING TO THE DATA ELEMENTS FOUND.
*****

*****  METHOD.
*****  THIS MODULE IS BASICALLY AN INTERFACE TO THE SEARCH MODULE
*****  THE SEARCH MODULE IS RESPONSIBLE FOR RETRIEVING PTRS TO
*****  THE BEUS CONTAINING THE DESIRED DATA, AND THE FETCH MODULE
*****  SIMPLY EXTRACTS THE DATA PORTION OF THOSE BEUS AND RETURNS
*****  A CONTROLLED STACK OF BIT STRINGS, CORRESPONDING TO THOSE
*****  DATA ELEMENTS, CALLED INFO_ND.  THE STRATEGY IS AS FOLLOWS:
*****  A) IF ID (AN INPUT PARAMETER) IS NULL, THEN THE SEARCH
*****  ROUTINE IS CALLED TO RETURN A STACK OF PTRS TO THE
*****  RELEVANT BEUS.  SEARCH IS PASSED:
*****      MODE1 - THE DESIRED RETRIEVAL MODE.
*****      NAME - NAME OF THE PSET INVOLVED (
*****              MAY BE A SUBSET OF ANOTHER PSET).
*****      KEY - KEY TO SEARCH ON.
*****      IDXX,Z - NOT SIGNIFICANT.
*****  ONCE SEARCH HAS RETURNED, FETCH GOES THROUGH THE PTR
*****  STACK BY USING THE TOP OF THE STACK AS A PTR TO A BEU,
*****  USING A BEU TEMPLATE TO EXTRACT THE DATA PORTION, AL-
*****  LOCATING INFO_ND, AND SETTING THE CURRENT ALLOCATION
*****  OF INFO_ND TO EQUAL THE EXTRACTED DATA.  THE TOP OF THE
*****  PTR STACK IS THEN POPPED AND THE PROCESS CONTINUES UNTIL
*****  THE PTR STACK IS EMPTY.
*****  B) IF ID IS NOT NULL, THEN AN ALTERNATIVE APPROACH IS

```


		2 INFO,	BCA00390
		3 DATA BIT(320);	BCA00400
		*****	SEA02090
		/* STACK TO RETURN FOUND ELEMENTS */	SEA02100
		%INCLUDE INFO;*****	SEA02110
		/* DATA STACK RETURNED BY FETCH */	SEA02120
3	1	0 DCL INFO_NO BIT(320) EXTERNAL CONTROLLED;	BCA00450
		*****	BCA00460
		%INCLUDE IDS1; *****	SEA02120
		/* POINTER STACK RETURNED BY SEARCH */	SEA02130
4	1	0 DCL IDS1 PTR EXTERNAL CONTROLLED;	BCA00420
		*****	BCA00430
		/* MISC DECLARATIONS */	SEA02130
5	1	0 DCL MODE1 BIT(2), FND BIT(1), (KEY,NAME) BIT(*),IDXX POINTER	SEA02140
		CONTROLLED, (ID,Q,Z) PTR;	SEA02150
		/* PROCEDURES CALLED */	SEA02160
		%INCLUDE ESEARCH;*****	SEA02170
		/* SEARCH MODULE */	SEA02180
6	1	0 DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER);	BCA00700
		*****	BCA00710
			BCA00720
7	1	0 IF ID=NULL()	SEA02180
		THEN DO;	SEA02190
		/* GET IDS OF ITEMS IN SET TO BE FETCHED */	SEA02200
8	1	1 CALL SEARCH(MODE1,NAME,KEY,IDXX,Z);	SEA02210
9	1	1 L=ALLOCATION (IDS1);	SEA02220
10	1	1 IF L>0 THEN FND='1'B;	SEA02230
11	1	1 ELSE FND='0'B;	SEA02240
			SEA02250
			SEA02260
			SEA02270
		/* FETCH DATA CONTENTS OF BEUS POINTED TO BY IDS1 */	SEA02280
12	1	1 DO J=1 TO L;	SEA02290
		/* PUT ITEMS ON STACK */	SEA02300
13	1	2 ALLOCATE INFO_NO ;	SEA02310
14	1	2 INFO_NO=IDS1->DATA;	SEA02320
15	1	2 FREE IDS1;	SEA02330
16	1	2 END;	SEA02340
17	1	1 RETURN;	SEA02350
18	1	1 END;	SEA02360
		/* IF ONLY 1 ELEMENT WAS TO BE FETCHED */	SEA02370
19	1	0 ALLOCATE INFO_NO;	SEA02380
20	1	0 INFO_NO=ID->DATA;	SEA02390
21	1	0 FND='1'B;	SEA02400
22	1	0 RETURN;	SEA02410
23	1	0 END FETCH;	SEA02420

```

%INCLUDE CREATEE;*****CRP00010
/*****CRE00010
*CRE00020
*MODULE DESCRIPTION*CRE00030
*****CRE00040
0 CREATEE: PROCEDURE ( STR , /* BIT(*) */CRE00050
ZLEN, /* FIXED BIN(15) */CRE00060
ID_CREATED /* PTR */ );CRE00070
CRE00080
/*****CRE00090
*****PURPOSE:CRE00100
*****THIS MODULE IS RESPONSIBLE FOR CREATING THE BASIC STORED CRE00110
*****UNIT OF INFORMATION IN THE SYSTEM, CALLED A BINARY EN- CRE00120
*****CODING UNIT. CONCEPTUALLY A BEU LOOKS AS FOLLOWS: CRE00130
*****CRE00140
*****LENGTH | POINTER ARRAY | DATA |CRE00150
*****-----|-----|-----|CRE00160
*****WHERE: LENGTH - LENGTH OF RELEVANT DATA IN DATA AREA CRE00170
*****BEU (IN BITS). IN THIS IMPLEMENTATION THE CRE00180
*****DATA AREA IS FIXED LENGTH, SO THE LENGTH CRE00190
*****FIELD INDICATES THE PORTION OF THAT DATA CRE00200
*****AREA WHICH IS VALID. IN A FUTURE IM- CRE00210
*****PLEMENTATION THE LENGTH FIELD WOULD IN- CRE00220
*****DICATE THE ACTUAL LENGTH OF THE DATA AREA CRE00230
*****POINTER ARRAY - THIS POINTER ARRAY IS USED TO IM- CRE00240
*****PLEMENT BINARY ASSOCIATIONS BETWEEN BEU CRE00250
*****AS WELL AS TO LINK BEUS WITHIN A COMMON CRE00260
*****PSET. DEFINEP AND DEFINED ARE RESPONSIBLE CRE00270
*****FOR MANAGING THE POINTER ARRAYS, I.E. CRE00280
*****ALLOCATING POINTER SLOTS TO VARIOUS PUR- CRE00290
*****POSES. THE STATUS OF POINTER SLOTS IS CRE00300
*****CONTAINED IN THE PSET'S P_CAT CATALOGUE CRE00310
*****ENTRY. IN THIS IMPLEMENTATION THE PTR CRE00320
*****ARRAY HAS FIXED EXTENTS (16). FUTURE IM- CRE00330
*****PLEMENTATIONS MAY WANT THIS TO BE VARIABLE CRE00340
*****DATA - A FIXED LENGTH BIT STRING CONTAINING THE CRE00350
*****THE ACTUAL DATA. NOTE: MODULES WHICH ACCESS CRE00360
*****THE BEU ARE RESPONSIBLE FOR INTERPRETING ITS CRE00370
*****CONTENTS. THIS IS OFTEN DONE BY OVERLAYING CRE00380
*****BASED STRUCTURES ON THE DATA AREA. CRE00390
*****CRE00400
*****CRE00410
*****CRE00420
*****METHOD:CRE00430
*****THE APPROACH IS VERY SIMPLE:CRE00440
*****A) IT BEGINS BY ALLOCATING A BASED STRUCTURE CALLED CRE00450

```



```

1      0  MAPSET: PROC(MODE,TYPE, MAP1,MAP2,FREE);
2      1  0      DCL  (TYPE,I) FIXED BIN(8),FREE BIT(8),
                   (MAP1,MAP2,MAP(2)) BIT(16), MODE BIT(1);
3      1  0      MAP(1)=MAP1;
4      1  0      MAP(2)=MAP2;
5      1  0      DO I=1 TO 16;
6      1  1          IF SUBSTR(MAP(TYPE),I,1)=MODE
                       THEN DO;
7      1  2              SUBSTR(MAP1,I,1)=(~MODE);
8      1  2              SUBSTR(MAP2,I,1)=(~MODE);
9      1  2              FREE=BIN(I);
10     1  2              RETURN;
11     1  2          END;
12     1  1      END;
13     1  0      FREE=0;
14     1  0      RETURN;
15     1  0  END MAPSET;

```

```

UT100010
UT100020
UT100030
UT100040
UT100050
UT100060
UT100070
UT100080
UT100090
UT100100
UT100110
UT100120
UT100130
UT100140
UT100150
UT100160
UT100170

```



```

1      0  HASH:  PROC(STR,NUMCHAR) RETURNS(FIXED BIN(15));
2      1  0      DCL STR BIT(*), NUMCHAR FIXED BIN(15),TEMPCHAR BIT(16),
                (NUM,VALUE) FIXED BIN(31) INIT(0),NAME CHAR(8);

3      1  0      UNSPEC(NAME)=STR;
4      1  0      K=NUMCHAR;
5      1  0      DO J=1 0 K BY 16;
6      1  1      TEMPCHAR=SUBSTR(STR,J,16);
7      1  1      NUM=TEMPCHAR;
8      1  1      VALUE=NUM+VALUE;
9      1  1      END;
10     1  0      VALUE=MOD(VALUE,51);
11     1  0      IF VALUE=0 THEN VALUE =1;
12     1  0      RETURN(VALUE);
13     1  0  END HASH;

```

```

UT100190
UT100200
UT100210
UT100220
UT100230
UT100240
UT100250
UT100260
UT100270
UT100280
UT100290
UT100300
UT100310
UT100320
UT100330
UT100340

```

1	0	PRINTIT: PROC(IFLAG,PPP);	UT100360
2	1	0 DCL (P,PPP) POINTER;	UT100370
3	1	0 OCL 1 T_ELEMENT BASED(P),	UT100380
		2 LEN FIXED BIN(15),	UT100390
		2 P_ARRAY(16) POINTER,	UT100400
		2 DATA BIT(320), IFLAG FIXED BIN(15);	UT100410
4	1	0 IF PPP=NULL() THEN	UT100420
		PUT LIST('POINTER IS NULL');	UT100430
5	1	0 ELSE PUT SKIP EDIT(IFLAG,SUBSTR(PPP->DATA,1,96),SUBSTR(PPP->DATA,	UT100440
		97,96))(F(3),B(96),SKIP(1),B(96));	UT100450
6	1	0 RETURN;	UT100460
7	1	0 END PRINTIT;	UT100470

STMT LEV NT

```
1      0  PRINTP: PROC(P);  
2      1  0      DCL P POINTER,  A BIT(31), TRY FIXED BIN(31);  
3      1  0      A=UNSPEC(P);  
4      1  0      TRY=A;  
5      1  0      PUT EDIT(TRY) (F(8));  
6      1  0  RETURN;  
7      1  0      END PRINTP;
```

UT100490
UT100500
UT100510
UT100520
UT100530
UT100540
UT100550
UT100560

- Andreu77: Andreu, R. and Madnick, S.E. A Systematic Approach to the Design of Complex Systems: Application to DBMS Design and Evaluation. CISR-99, Sloan School of Management, MIT, March 1977.
- Astrahan: Astrahan, M.M. et al. 'System R' ACM TODS, Vol 1, No. 2, June 1976.
- ANSI/SPARC: ANSI/X3/SPARC study group on DBMS interim report, February 1975
- Badal: Badal, D.Z., 'The Analysis of the effects of Concurrency control on distributed database system performance', Proceedings International Conference on Very Large Data Bases 1980.
- Codd: Codd, E.F., 'Extending database relational model to capture more Meaning' ACM TODS, vol 4, no. 4, December 1979
- Date: Date, C.J., An Introduction to Database Systems. Addison-Wesley Publishing Co, 1977
- Hsiao: Hsiao, D.K., Kerr, D.S., Madnick, S.E., Computer Security Academic Press, 1979.
- Huff: Huff, S.L. 'A Systematic Methodology for Designing the Architecture of Complex Software Systems' Ph.D. thesis, Sloan School of Management, MIT, June 1980
- Hsu: Hsu M., 'A Preliminary Architectural Design for the Functional Hierarchy of the INFOPLEX Database Computer', CISR report 5, Sloan School of Management, MIT 1980.
- Klug: Klug, A. and Tsichritzis, D., 'Multiple View Support within the ANSI/SPARC framework' Proceedings International conference on Very Large Data Bases, 1977

Madnick75: Madnick, S.E., 'Design of a General Hierarchical Storage System', IEEE International Conference Proceedings, 1975.

Madnick79: Madnick S.E., 'The INFOPLEX Database Computer: Concepts and Directions' Proceedings, IEEE Computer Conference, February 1979

Parnas: Parnas, D.L. 'On the Design and Development of Program Families'. IEEE Transactions on Software Engineering, March 1976

Reis: Reis, D.R. and Stonebraker, M., 'Effects of Locking Granularity in a Database Management System', ACM TODS

vol. 2, No. 3, September 1977

Senko: Senko, M.E. 'Data Structures and accessing in database systems: II. Information Organization', and 'III. Data representations and Data Independent Accessing Model', IBM Systems Journal, No.1 1973

Yao: Yao, S.B., 'Optimization of Query Evaluation Algorithms', ACM TODS Vol. 4, No. 2, June 1979.

FILM
8-