INFOSAM: A SAMPLE DATABASE MANAGEMENT SYSTEM (U)

UNCLASSIFIED

CISR-M010-8112-07

N00039-81-C-0663

NL
INFOSAM:
A SAMPLE DATABASE MANAGEMENT SYSTEM

Technical Report #7

By
Bruce Blumberg

December 1981

Principal Investigator:
Professor Stuart E. Madnick

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

This thesis describes the design and implementation of a relational database management system called INFOSAM. Its design is largely based on Madnick's proposed design for a Database Management System <Madnick79>. Its design incorporates the concept of hierarchical decomposition, whereby the system is organized into levels or groups 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 through the services provided by the next lower level. INFOSAM incorporates 3 major levels, which are primarily distinguished by their view of the data. They are the External level, the Nset level and the Internal level. The objective of this stratification is to provide a high degree of data independence for the user, and a low level of functional redundancy in the system.

The objective of this thesis project was twofold. One objective was to implement a sample database management system that could be used as a teaching tool in classes on Database Management Systems. INFOSAM is small enough to be
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
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modules of the Binary Layer</td>
<td>78</td>
</tr>
<tr>
<td>DEFINEB</td>
<td>78</td>
</tr>
<tr>
<td>CREATEB</td>
<td>80</td>
</tr>
<tr>
<td>SELECTF</td>
<td>82</td>
</tr>
<tr>
<td>Concluding Remarks on Internal Level</td>
<td>83</td>
</tr>
<tr>
<td>THE NSET LEVEL</td>
<td>85</td>
</tr>
<tr>
<td>Overview of the Nset Level</td>
<td>86</td>
</tr>
<tr>
<td>Databases of the Nset Level</td>
<td>96</td>
</tr>
<tr>
<td>NSET_CAT</td>
<td>96</td>
</tr>
<tr>
<td>Inter-level Communication Databases</td>
<td>99</td>
</tr>
<tr>
<td>DEF_ARG</td>
<td>100</td>
</tr>
<tr>
<td>DV_ARG</td>
<td>101</td>
</tr>
<tr>
<td>INSERT_ARG</td>
<td>101</td>
</tr>
<tr>
<td>RET_ARG</td>
<td>102</td>
</tr>
<tr>
<td>DOM_RET</td>
<td>105</td>
</tr>
<tr>
<td>The <em>FV_ARG</em> database</td>
<td>107</td>
</tr>
<tr>
<td>Modules of the Nset Level</td>
<td>108</td>
</tr>
<tr>
<td>DEFINEN</td>
<td>108</td>
</tr>
<tr>
<td>DEFINEV</td>
<td>109</td>
</tr>
<tr>
<td>INSERTN</td>
<td>110</td>
</tr>
<tr>
<td>FETCHT</td>
<td>111</td>
</tr>
<tr>
<td>FETCHV</td>
<td>118</td>
</tr>
<tr>
<td>BUILDC</td>
<td>119</td>
</tr>
<tr>
<td>NINIT</td>
<td>119</td>
</tr>
<tr>
<td>Summary of the Nset Level</td>
<td>120</td>
</tr>
<tr>
<td>THE EXTERNAL LEVEL</td>
<td>121</td>
</tr>
<tr>
<td>Logical Overview of External Level</td>
<td>122</td>
</tr>
<tr>
<td>Databases of the External Level</td>
<td>127</td>
</tr>
<tr>
<td>VTABLE</td>
<td>128</td>
</tr>
<tr>
<td>RTABLE</td>
<td>131</td>
</tr>
<tr>
<td>VTABLE</td>
<td>131</td>
</tr>
<tr>
<td>The T1_ARG Database</td>
<td>131</td>
</tr>
<tr>
<td>Communication Databases</td>
<td>133</td>
</tr>
<tr>
<td>The Modules of the External Level</td>
<td>135</td>
</tr>
<tr>
<td>SAM</td>
<td>136</td>
</tr>
<tr>
<td>DEFINE</td>
<td>137</td>
</tr>
<tr>
<td>DEFDOM</td>
<td>137</td>
</tr>
<tr>
<td>DEFREL</td>
<td>138</td>
</tr>
<tr>
<td>DEFVIEW</td>
<td>139</td>
</tr>
<tr>
<td>C'ERY</td>
<td>140</td>
</tr>
<tr>
<td>GETVIEW</td>
<td>141</td>
</tr>
<tr>
<td>SHVIEW</td>
<td>142</td>
</tr>
<tr>
<td>SHREI</td>
<td>142</td>
</tr>
<tr>
<td>SELECT</td>
<td>143</td>
</tr>
<tr>
<td>PROJECT</td>
<td>144</td>
</tr>
<tr>
<td>JOIN</td>
<td>145</td>
</tr>
<tr>
<td>PRINT</td>
<td>147</td>
</tr>
</tbody>
</table>
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 ...........................................................................................................

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 other hand, 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>, <Hsiao77>; 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 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 hierarchically decomposed functions 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

<table>
<thead>
<tr>
<th>FUNCTIONAL HIERARCHY</th>
<th>STORAGE HIERARCHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Responsible for all DBMS functions except device and storage management.</td>
<td>* Responsible for storage and device management.</td>
</tr>
<tr>
<td>* Functional hierarchy of modules</td>
<td>* Physical hierarchy of storage devices.</td>
</tr>
</tbody>
</table>

This approach is similar to other attempts to stratify the design of a DBMS based on either function or view of the data (see <Madrick79> 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 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

- 24 -
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 lowermost 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

- 29 -
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 its 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 its 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 its 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 its own level or the level immediately beneath it.

TABLE 2
Schematic Overview of Infosam

| --------------------------- |   |
|                            |   |
|                            |   |
| EXTERNAL LEVEL             |   |
|                            |   |
|                            |   |
| NSET LEVEL                 |   |
|                            |   |
|                            |   |
| INTERNAL LEVEL             |   |
| Bset Layer                 |   |
| Primitive Layer            |   |

In the following sections we will discuss each level in terms of its data model and its 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

<table>
<thead>
<tr>
<th>Module</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINEB</td>
<td>BSETCAT</td>
</tr>
<tr>
<td>CREATEB</td>
<td></td>
</tr>
<tr>
<td>SELECTF</td>
<td></td>
</tr>
<tr>
<td>PSETCAT</td>
<td>BEU</td>
</tr>
<tr>
<td>IDOS1</td>
<td>INFDNO</td>
</tr>
<tr>
<td>DEFINEP</td>
<td>CREATEP</td>
</tr>
<tr>
<td>SEARCH</td>
<td>FETCH</td>
</tr>
<tr>
<td>CREATE_I</td>
<td>CREATE_E</td>
</tr>
<tr>
<td>INIT_P</td>
<td>MAPSET</td>
</tr>
<tr>
<td></td>
<td>HASH</td>
</tr>
</tbody>
</table>

- Module

- Database

- 38 -
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, it's 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 its 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

<table>
<thead>
<tr>
<th>LENGTH</th>
<th>Pointer Array</th>
<th>NAME</th>
<th>SP.TMP</th>
<th>AP.TMP</th>
<th>NUMPAGE</th>
<th>L.TMP</th>
<th>L.RO32</th>
<th>KEYS</th>
<th>SSESST</th>
<th>PSET.ID</th>
<th>S.CHAIN</th>
<th>DOHLEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>144</td>
<td></td>
<td>PSETCAT</td>
<td>3FFF</td>
<td>3FFF</td>
<td>H</td>
<td>211</td>
<td>64</td>
<td>0000</td>
<td>144</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of the PSET_CAT Pset organization with labels for INDEX, PNAME, and various data fields.](image)
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>
<thead>
<tr>
<th>NAME</th>
<th>SP-HAP</th>
<th>BP-HAP</th>
<th>LNUMFREE</th>
<th>L-KEY</th>
<th>L-POS2</th>
<th>LEN</th>
<th>SET</th>
<th>S-CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARTS</td>
<td>FFFF</td>
<td>FFFF</td>
<td>15</td>
<td>0</td>
<td>2</td>
<td>164</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Screw
- Bolt
- Wrench
TABLE 6

Example of B_tree linkage

| LENGTH | POINTER ARRAY | NAME  | SP-NUM | AL-NUM | L-TYPE | L-POS | L-POS+1 | KEY-POS | SET-LEN | SET-LEN | P-CHAIN | S-CHAIN | B-CHAIN |
|--------|----------------|-------|--------|--------|--------|-------|---------|---------|---------|---------|---------|---------|---------|---------|
|        |                | SUPPER| 4FFF   | 9FFF   | 14     | B     | 2       | 3       | 640     | 0       | 000     | 00      | 00      | 00      |

- 45 -
TABLE 7

Example of Hashing via Scatter Table and Overflow Chaining

<table>
<thead>
<tr>
<th>LENGTH</th>
<th>Pointer Array</th>
<th>NAME</th>
<th>SNAME</th>
<th>AGE</th>
<th>NUMBER</th>
<th>L-TYPE</th>
<th>1-POS</th>
<th>KV1-POS</th>
<th>KV2-POS</th>
<th>S-LEVEL</th>
<th>P-CHAIN</th>
<th>S-CHAIN</th>
<th>DATA LEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>144</td>
<td></td>
<td>CITY</td>
<td>IFFF</td>
<td>9FFE</td>
<td>14</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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 linkages 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 its linkages. 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 its 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,
SUBSET_ID,
P_CHAIN,
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 cannot 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 descendents 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 descendents 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.

- 54 -
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_POSI 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:
A stack structure was chosen because of the dynamic nature of the number of 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-
parameters 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 it's 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 IDS) 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 IDSl 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 its 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 its 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

|.CREATED| TIME| ARRAY| NAME| KNAME| NAME| MAKE| MODEL| SERIAL| SOURCE| NSERIES| NSERIES
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`BSET10` ELEMENT DESCRIBING BSET1

... Smith

... S1
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 domain 2, 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.

- 71 -
TABLE 9

Implementation of a 1 to N Binary Association

<table>
<thead>
<tr>
<th>Length</th>
<th>Parent Array</th>
<th>Name</th>
<th>Object 1</th>
<th>Object 2</th>
<th>Object 3</th>
<th>Object 4</th>
<th>Object 5</th>
<th>Object 6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SLOAN</td>
<td></td>
<td></td>
<td>3:2:1:4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:
- SLOAN
- SLOAN, STUDENT
- MICHAEL
- ANDREW
- VIRGINIA

- 72 -
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

<table>
<thead>
<tr>
<th>NAME</th>
<th>ROLE-SET 1</th>
<th>AP. ROLE-SET 1</th>
<th>AP. ROLE-SET 2</th>
<th>AP. ROLE-SET 2</th>
<th>TYPE</th>
<th>PARENT-TO</th>
<th>NET-MARK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:**
- M: Min
- N: Max

**Diagram:**
- **PSET-0:** Element describing the set
- **PSET-1:** Element describing the set subset
- **LINKSET ELEMENTS:**
- **NET-MARK:** Paris
- **NET-MARK:** London
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 its data model via the databases and modules of the Primitive layer together with layer specific database, BSET_CAT, and 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:

```plaintext
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 DEFINEB

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 DEFINEB 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 DEFINEB module is working on the the actual PSET_CAT entries for the Psets, they are automatically updated to reflect any changes made by the DEFINEB module. DEFINEB 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 its 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 terms 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 other hand, 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

<table>
<thead>
<tr>
<th>Module</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFNEW</td>
<td>NCAT</td>
</tr>
<tr>
<td>DEFINEY</td>
<td>NCATE2</td>
</tr>
<tr>
<td>INSERTN</td>
<td></td>
</tr>
<tr>
<td>FETCHT</td>
<td></td>
</tr>
<tr>
<td>FETCHV</td>
<td></td>
</tr>
<tr>
<td>NINIT</td>
<td>IDS1</td>
</tr>
<tr>
<td>BUILDC</td>
<td>INFO10</td>
</tr>
<tr>
<td>NRECON</td>
<td></td>
</tr>
<tr>
<td>TABN</td>
<td></td>
</tr>
<tr>
<td>NJOIN1</td>
<td></td>
</tr>
</tbody>
</table>
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

PART

P#  PNAME  COLOR  WEIGHT

IMPLEMENTATION OF AN INSTANCE:

ENT NODE #
P1
CAM
BLUE
15
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 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, it's 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

Scheme 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 the restrictions specified
<table>
<thead>
<tr>
<th>INFOSAM: A SAMPLE DATABASE MANAGEMENT SYSTEM. (U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC 81 B BLUMBERG</td>
</tr>
<tr>
<td>UNCLASSIFIED</td>
</tr>
<tr>
<td>CISR-M010-8112-07</td>
</tr>
<tr>
<td>N00039-81-C-0663</td>
</tr>
<tr>
<td>UNCLASSIFIED</td>
</tr>
<tr>
<td>NL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INFOSAM: A SAMPLE DATABASE MANAGEMENT SYSTEM. (U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC 81 B BLUMBERG</td>
</tr>
<tr>
<td>UNCLASSIFIED</td>
</tr>
<tr>
<td>CISR-M010-8112-07</td>
</tr>
<tr>
<td>N00039-81-C-0663</td>
</tr>
<tr>
<td>UNCLASSIFIED</td>
</tr>
<tr>
<td>NL</td>
</tr>
</tbody>
</table>
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 its 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, DVM_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),
   \(3 \text{ K_TYPE},\)
   3 BREL ) BIT(8),
   \(3 \text{ BSETUP},\)
   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),

- 101 -
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 Nsets specified in a request to 16, with a maximum of 10 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, i.e. 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.
TABLE 14

Example use of RET_ARG

Given the following conceptual view of the database:

```
PART
  NAME
  WEIGHT
  COLOR

SP
  P#
  QTY
  S#
  NAME
  CITY

SUPPLIER
  SNAME
  STATUS
```

AND the following sequence of commands:

```
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
```

The following copy of RET_ARG would be passed toペット

<table>
<thead>
<tr>
<th>NUM</th>
<th>NSET</th>
<th>NNUM</th>
<th>NAME</th>
<th>FIELD</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>SP</td>
<td>1</td>
<td>SNAME</td>
<td>'0'B</td>
<td>'00'</td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td>1</td>
<td>STATUS</td>
<td>'0'B</td>
<td>'00'</td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td>1</td>
<td>CITY</td>
<td>'0'B</td>
<td>'00'</td>
</tr>
<tr>
<td></td>
<td>SP</td>
<td>1</td>
<td>CITY</td>
<td>'0'B</td>
<td>'00'</td>
</tr>
<tr>
<td>1</td>
<td>S#</td>
<td>'S1'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>S#</td>
<td>'0'B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>P#</td>
<td>'0'B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>QTY</td>
<td>'0'B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

```plaintext
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 it's 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 BUILDC, 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-

- 111 -
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. It's 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 BUILDC module. Each Nset specified in the request requires a call to BUILDC 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
TABLE 15

Flowchart of NJOIN1 module

When NJOIN1 is called, it is passed L, which indicates the highest order pair to NJOIN to be considered with. In addition, it was passed to NJOIN, values containing all of the input definitions relevant to the entire process. NJOIN then passes an array which contains all of the entries which are searched for restriction points (1, 2, 3) used by the input module and the JN module. Entries are removed, and the result is a controlled variable which has the same species as L. Entries satisfying this rule are placed on a stack called TUPLE.

When L is not empty, it is passed to NJOIN, and P is passed to NJOIN. NJOIN then calls itself recursively with input P and L. If L is not empty, a path is determined for NJOIN, and a path is determined for NJOIN. CALL NJOIN1, passing it (L - 2, 3) and

1. End

2. Add NJOIN to the list of tuples for the input

3. Add NJOIN to the list of tuples for the input

4. Add NJOIN to the list of tuples for the input

5. Add NJOIN to the list of tuples for the input

6. Add NJOIN to the list of tuples for the input

7. Add NJOIN to the list of tuples for the input

8. Add NJOIN to the list of tuples for the input

9. Add NJOIN to the list of tuples for the input

10. Add NJOIN to the list of tuples for the input

11. Add NJOIN to the list of tuples for the input

12. Add NJOIN to the list of tuples for the input

13. Add NJOIN to the list of tuples for the input

14. Add NJOIN to the list of tuples for the input

15. Add NJOIN to the list of tuples for the input

16. Add NJOIN to the list of tuples for the input

17. Add NJOIN to the list of tuples for the input

18. Add NJOIN to the list of tuples for the input

19. Add NJOIN to the list of tuples for the input

20. Add NJOIN to the list of tuples for the input

21. Add NJOIN to the list of tuples for the input

22. Add NJOIN to the list of tuples for the input

23. Add NJOIN to the list of tuples for the input

24. Add NJOIN to the list of tuples for the input

25. Add NJOIN to the list of tuples for the input

26. Add NJOIN to the list of tuples for the input

27. Add NJOIN to the list of tuples for the input

28. Add NJOIN to the list of tuples for the input

29. Add NJOIN to the list of tuples for the input

30. Add NJOIN to the list of tuples for the input

31. Add NJOIN to the list of tuples for the input

32. Add NJOIN to the list of tuples for the input

33. Add NJOIN to the list of tuples for the input

34. Add NJOIN to the list of tuples for the input

35. Add NJOIN to the list of tuples for the input

36. Add NJOIN to the list of tuples for the input

37. Add NJOIN to the list of tuples for the input

38. Add NJOIN to the list of tuples for the input

39. Add NJOIN to the list of tuples for the input

40. Add NJOIN to the list of tuples for the input

41. Add NJOIN to the list of tuples for the input

42. Add NJOIN to the list of tuples for the input

43. Add NJOIN to the list of tuples for the input

44. Add NJOIN to the list of tuples for the input

45. Add NJOIN to the list of tuples for the input

46. Add NJOIN to the list of tuples for the input

47. Add NJOIN to the list of tuples for the input

48. Add NJOIN to the list of tuples for the input

49. Add NJOIN to the list of tuples for the input

50. Add NJOIN to the list of tuples for the input

51. Add NJOIN to the list of tuples for the input

52. Add NJOIN to the list of tuples for the input

53. Add NJOIN to the list of tuples for the input

54. Add NJOIN to the list of tuples for the input

55. Add NJOIN to the list of tuples for the input

56. Add NJOIN to the list of tuples for the input

57. Add NJOIN to the list of tuples for the input

58. Add NJOIN to the list of tuples for the input

59. Add NJOIN to the list of tuples for the input

60. Add NJOIN to the list of tuples for the input

61. Add NJOIN to the list of tuples for the input

62. Add NJOIN to the list of tuples for the input

63. Add NJOIN to the list of tuples for the input

64. Add NJOIN to the list of tuples for the input

65. Add NJOIN to the list of tuples for the input

66. Add NJOIN to the list of tuples for the input

67. Add NJOIN to the list of tuples for the input

68. Add NJOIN to the list of tuples for the input

69. Add NJOIN to the list of tuples for the input

70. Add NJOIN to the list of tuples for the input

71. Add NJOIN to the list of tuples for the input

72. Add NJOIN to the list of tuples for the input

73. Add NJOIN to the list of tuples for the input

74. Add NJOIN to the list of tuples for the input

75. Add NJOIN to the list of tuples for the input

76. Add NJOIN to the list of tuples for the input

77. Add NJOIN to the list of tuples for the input

78. Add NJOIN to the list of tuples for the input

79. Add NJOIN to the list of tuples for the input

80. Add NJOIN to the list of tuples for the input

81. Add NJOIN to the list of tuples for the input

82. Add NJOIN to the list of tuples for the input

83. Add NJOIN to the list of tuples for the input

84. Add NJOIN to the list of tuples for the input

85. Add NJOIN to the list of tuples for the input

86. Add NJOIN to the list of tuples for the input

87. Add NJOIN to the list of tuples for the input

88. Add NJOIN to the list of tuples for the input

89. Add NJOIN to the list of tuples for the input

90. Add NJOIN to the list of tuples for the input

91. Add NJOIN to the list of tuples for the input

92. Add NJOIN to the list of tuples for the input

93. Add NJOIN to the list of tuples for the input

94. Add NJOIN to the list of tuples for the input

95. Add NJOIN to the list of tuples for the input

96. Add NJOIN to the list of tuples for the input

97. Add NJOIN to the list of tuples for the input

98. Add NJOIN to the list of tuples for the input

99. Add NJOIN to the list of tuples for the input

100. Add NJOIN to the list of tuples for the input

101. Add NJOIN to the list of tuples for the input

102. Add NJOIN to the list of tuples for the input

103. Add NJOIN to the list of tuples for the input

104. Add NJOIN to the list of tuples for the input

105. Add NJOIN to the list of tuples for the input

106. Add NJOIN to the list of tuples for the input

107. Add NJOIN to the list of tuples for the input

108. Add NJOIN to the list of tuples for the input

109. Add NJOIN to the list of tuples for the input

110. Add NJOIN to the list of tuples for the input

111. Add NJOIN to the list of tuples for the input

112. Add NJOIN to the list of tuples for the input

113. Add NJOIN to the list of tuples for the input

114. Add NJOIN to the list of tuples for the input

Copy available to DTIC does not permit fully legible production/
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 first 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 BUILDC

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 its 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. 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, it's 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 it's 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

<table>
<thead>
<tr>
<th>DNAME</th>
<th>DATR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supplier</td>
<td>C2</td>
</tr>
<tr>
<td>SNAME</td>
<td>C4</td>
</tr>
<tr>
<td>STATUS</td>
<td>C6 0 50</td>
</tr>
<tr>
<td>CITY</td>
<td>C6</td>
</tr>
<tr>
<td>PE</td>
<td>C2</td>
</tr>
<tr>
<td>PNAME</td>
<td>C8</td>
</tr>
<tr>
<td>COLOR</td>
<td>C3</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>N 2 1 0 20</td>
</tr>
<tr>
<td>QTY</td>
<td>N 3 0 500</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RNAME</th>
<th>DNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>PART</td>
<td>PART</td>
</tr>
<tr>
<td>PART</td>
<td>PNAME</td>
</tr>
<tr>
<td>PART</td>
<td>COLOR</td>
</tr>
<tr>
<td>PART</td>
<td>WEIGHT</td>
</tr>
<tr>
<td>PART</td>
<td>CITY</td>
</tr>
<tr>
<td>SUPPLIER</td>
<td>S4</td>
</tr>
<tr>
<td>SUPPLIER</td>
<td>SNAME</td>
</tr>
<tr>
<td>SUPPLIER</td>
<td>STATUS</td>
</tr>
<tr>
<td>SUPPLIER</td>
<td>CITY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RNAME</th>
<th>V-1D</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPPLIER</td>
<td>V1</td>
</tr>
<tr>
<td>PART</td>
<td>V1</td>
</tr>
<tr>
<td>SP</td>
<td>V2</td>
</tr>
<tr>
<td>SALARIES</td>
<td>V2</td>
</tr>
<tr>
<td>SPECIFICATIONS</td>
<td>V2</td>
</tr>
</tbody>
</table>

Diagram:

- Diagram showing the relationship between DTABLE, RTABLE, and VTABLE.
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, fetchs 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 T1_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 T1_ARG, a new entry is created in T1_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 T1_ARG, and calls FETCHT passing it a copy of the entry. T1_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 how T1_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, NV_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

<table>
<thead>
<tr>
<th>RELNAME</th>
<th>NUM</th>
<th>NSET</th>
<th>N-INDEX</th>
<th>NAME</th>
<th>FETCH</th>
<th>SAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supplier</td>
<td>1</td>
<td>Supplier</td>
<td>1</td>
<td>S#</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NAME</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>STATUS</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CITY</td>
<td>'4'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td>Part</td>
<td>1</td>
<td>Part</td>
<td>1</td>
<td>P#</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PHONE</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>COLOR</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>WEIGHT</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CITY</td>
<td>'4'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td>SP</td>
<td>1</td>
<td>SP</td>
<td>1</td>
<td>S#</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td>Getview Command</td>
<td>1</td>
<td>P#</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>QTY</td>
<td>'4'8</td>
<td>'00'</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>1</td>
<td>Supplier</td>
<td>1</td>
<td>S#</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NAME</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STATUS</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CITY</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td>LONDON</td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>2</td>
<td>Supplier</td>
<td>1</td>
<td>S#</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SNAME</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STATUS</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CITY</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td>LONDON</td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>1</td>
<td>SNAME</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STATUS</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CITY</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td>LONDON</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>2</td>
<td>Supplier</td>
<td>1</td>
<td>S#</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CITY</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td>LONDON</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SNAME</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>STATUS</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CITY</td>
<td>'1'8</td>
<td>'00'</td>
<td>H</td>
<td>LONDON</td>
<td></td>
</tr>
</tbody>
</table>

- 134 -
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 DEEVIEW 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.
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:

DOMA in  <dname>  <d_attr>  where

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

- 137 -
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 for 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 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 vir-
tual 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 TI_ARG. Then, for each restriction specified in the request, the module locates the corresponding domain description in the TI_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 Tl_ARG to indicate that they are not to be retrieved. That is, the Tl_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 Tl_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 for 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 is 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.

- 153 -
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 level's 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 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 Tl_ARG
and all user issued retrieval requests and relational opera-
tions involve Tl_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 Tl_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 Tl_ARG is loaded, the system would
additionally load the upper entries in Tl_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, its 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 types 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 it's strengths and weaknesses. We have seen that it is a relational DBMS, which is designed around the complimentary 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

- 164 -
* 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:

Q

MAXIMUM VALUE:

SQ

D:

D CITY C 6

- 165 -
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.
L:
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:

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

<table>
<thead>
<tr>
<th>DOMAIN</th>
<th>TYPE</th>
<th>LEN</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>======</td>
<td>===</td>
<td>===</td>
<td>===</td>
<td>===</td>
</tr>
</tbody>
</table>

QTY

| NUM | 3 | 0 | 00 |

PP

| CHAR | 2 | -- | -- |

SS

| CHAR | 2 | -- | -- |

Q:

SR SUPPLIER

<table>
<thead>
<tr>
<th>DOMAIN</th>
<th>TYPE</th>
<th>LEN</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 168 -
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CITY</td>
<td>CHAR</td>
<td>6</td>
<td>--</td>
</tr>
<tr>
<td>STATUS</td>
<td>NUM</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>SNAME</td>
<td>CHAR</td>
<td>6</td>
<td>--</td>
</tr>
<tr>
<td>SS</td>
<td>CHAR</td>
<td>2</td>
<td>--</td>
</tr>
</tbody>
</table>

Q: SR PART

<table>
<thead>
<tr>
<th>DOMAIN</th>
<th>TYPE</th>
<th>LEN</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITY</td>
<td>CHAR</td>
<td>6</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>NUM</td>
<td>2</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>COLOR</td>
<td>CHAR</td>
<td>6</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>PNAME</td>
<td>CHAR</td>
<td>6</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

- 169 -
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

<table>
<thead>
<tr>
<th>QTY</th>
<th>PP</th>
<th>SS</th>
</tr>
</thead>
</table>

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:

- 170 -
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

<table>
<thead>
<tr>
<th>QTY</th>
<th>PP</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>340</td>
<td>P4</td>
<td>S4</td>
</tr>
<tr>
<td>51</td>
<td>P5</td>
<td>S1</td>
</tr>
<tr>
<td>100</td>
<td>P3</td>
<td>S1</td>
</tr>
</tbody>
</table>

- 171 -
| 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

<table>
<thead>
<tr>
<th>SNAME</th>
<th>CITY</th>
<th>SS</th>
<th>STATUS</th>
</tr>
</thead>
</table>

L:

SMITH, LONDON, S1, 10

- 172 -
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

<table>
<thead>
<tr>
<th>CITY</th>
<th>STATUS</th>
<th>SNAME</th>
<th>SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATHENS</td>
<td>24</td>
<td>ADAMS</td>
<td>S5</td>
</tr>
<tr>
<td>LONDON</td>
<td>27</td>
<td>CLARK</td>
<td>S4</td>
</tr>
<tr>
<td>LONDON</td>
<td>10</td>
<td>SMITH</td>
<td>S1</td>
</tr>
<tr>
<td>PARIS</td>
<td>14</td>
<td>BLACK</td>
<td>S3</td>
</tr>
</tbody>
</table>

- 173 -
| PART | 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 |

- 174 -
| ROMÉ  | 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:

- 175 -
PRINT T1

<table>
<thead>
<tr>
<th>CITY</th>
<th>WEIGHT</th>
<th>COLOR</th>
<th>PNAME</th>
<th>PP</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARIS</td>
<td>37</td>
<td>BLUE</td>
<td>CAM</td>
<td>P5</td>
</tr>
<tr>
<td>ROME</td>
<td>14</td>
<td>BLUE</td>
<td>SCREW</td>
<td>P3</td>
</tr>
</tbody>
</table>

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

Q:
PRINT T2

<table>
<thead>
<tr>
<th>CITY</th>
<th>WEIGHT</th>
<th>COLOR</th>
<th>PNAME</th>
<th>PP</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARIS</td>
<td>37</td>
<td>BLUE</td>
<td>CAM</td>
<td>P5</td>
</tr>
</tbody>
</table>

Q:
PROJECT PART ON PNAME,COLOR GIVING T3

Q:
PRINT T3

<table>
<thead>
<tr>
<th>COLOR</th>
<th>PNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLUE</td>
<td>SCREW</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>GREEN</td>
<td>BOLT</td>
</tr>
</tbody>
</table>
* 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

<table>
<thead>
<tr>
<th>QTY</th>
<th>PP</th>
<th>SS</th>
<th>CITY</th>
<th>STATUS</th>
<th>SNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>P5</td>
<td>S4</td>
<td>LONDON</td>
<td>27</td>
<td>CLARK</td>
</tr>
</tbody>
</table>
Q:
JOIN T12 AND PART ON PP, CITY GIVING T13
Q:
PRINT T13

<table>
<thead>
<tr>
<th>QTY</th>
<th>PP</th>
<th>SS</th>
<th>CITY</th>
<th>STATUS</th>
<th>SNAME</th>
<th>WEIGHT</th>
<th>COLOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

| 340 | P4 | S4 | LONDON | 27 | CLARK | 24 | RED | SCREW |

Q:
IFS:
-- END OF SESSION --

R;
Appendix B

LISTINGS AND DOCUMENTATION FOR THE INFOSAM SYSTEM
%INCLUDE SAM;     ********************************************SAM:000010
   /**********************************************************
   *                           MODULE DESCRIPTION
   ***********************************************************/
1 0 SAM:  PROCEDURE;
   /**********************************************************
   **** 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
   ***********************************************************/
   ************************************************************
   /**********************************************************
2 1 0 / COMMAND LISTS */
   DCL OP(2) CHAR(6) INITIAL('DEFINE','QUERY'),
         AOP(2) CHAR(6) INITIAL('D','Q');
   /**********************************************************
3 1 0 / SUBROUTINES */
4 1 0 DCL DEFINE ENTRY;
5 1 0 DCL QUERY ENTRY;
   /**********************************************************
6 1 0 / INCLUDE ENINIT:********************************************SAM:00010
   /**********************************************************
5 1 0 DCL NINIT ENTRY;
   /**********************************************************
6 1 0 / MISCELLANEOUS */
   DCL NAME CHAR(6),
       PCATPTR POINTER STATIC EXTERNAL INIT(NULL());
   /**********************************************************
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:');
  GE 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 SESSION */
PUT SKIP(2) LIST ('-- END OF SESSION --');
ENDSAM;
%INCLUDE DEFINE:                              FOR00010

/*------------------------------------------------------------------------*/
/* MODULE DESCRIPTION*/
/*------------------------------------------------------------------------*/
1 0 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 (ONAME, DATTR, RNAM, ID). AND*/
/* CALLS TO DEFINENV 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, DEFFREL, DEFDOM, LEX, DEFINEV, DEFINEN*/
/*------------------------------------------------------------------------*/

/*------------------------------------------------------------------------*/
/* CONTAID LISTS */
DCL O9P(3) CHAR(B) INITIAL('VIEW', 'RELATION',
   'DOMAIN'),
   AOP(3) CHAR(B) INITIAL('V', 'R', 'D');
/* DEFINENV TABLE */
%INCLUDE DEFARG;                              FOR00090
/*------------------------------------------------------------------------*/

/*------------------------------------------------------------------------*/
3 0 DCL 1 DEF_ARG,    /* USED TO DEFINE AN NSET */ DCL00010
IE;

2 NAME BIT(64), /* NAME OF NSET */ DCL00020
2 ATTR BIT(64), /* NUMBER OF ATTRIBUTES */ DCL00030
2 ATTR(20), /* FOR EACH ATTRIBUTE */ DCL00040
3 NAME BIT(64), /* ATTRIBUTE NAME */ DCL00050
3 K_TYPE BIT(8); /* UNIQUE KEY OR NOT */ DCL00060
DCL00070
D 00100
D 00110
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
4 1 0 DCL 1 DI_ARG ,
NAME NAME OF VALUE SET */ DVA000020
2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */ DVA000030
DVA000040
D 00110
D 00120
D 00130
/* DEFINE DICTIONARIES */

IF FLAG THEN UO;

/* DEFINE PSETS */

KEY_LEN = '00111000'B; NAME = UNSPEC('DATTR ');

CALL DEFINE(DV_ARG);

KEY_LEN = '01000000'B; NAME = UNSPEC('DNAME ');

CALL DEFINE(DV_ARG);

KEY_LEN = '01000000'B; NAME = UNSPEC('ID ');

CALL DEFINE(DV_ARG);

/* DEFINE NSETS */

KEY_LEN = '00111000'B; NAME = UNSPEC('DATTR ');

CALL DEFINE(DV_ARG);

KEY_LEN = '01000000'B; NAME = UNSPEC('DNAME ');

CALL DEFINE(DV_ARG);

KEY_LEN = '01000000'B; NAME = UNSPEC('ID ');

CALL DEFINE(DV_ARG);

/* START SUBROJTINE */

DO WHILE(TRUE);

PUT SKIP LIST ('D:');

CALL LEX(N_TOK,VNAME);

IF N_TOK = 0 THEN RETURN;

/* SELECT COMMAND */

SELECT(VNAME(1));

/* DEFINE VIEW */

WHEN(OP(1),AOP(1)) CALL DEFWV(N_TOK,VNAME);

/* DEFINE RELATION */

WHEN(OP(2),AOP(2)) CALL DEFRM(N_TOK,VNAME);

/* DEFINE JOIN */

WHEN(OP(3),AOP(3)) CALL DEFFM(VNAME(2),VNAME(3),VNAME(4),VNAME(5),VNAME(6));

/* INVALID OPERATORS */


50 1 2  OTHERWISE
   GO;
   PUT SKIP(0) EDIT (VNAME(1), ' IS AN INVALID COMMAND')
       (A,A);
51 1 3
      PUT SKIP LIST ('RETYPE COMMAND:');
52 1 3
      GET EDIT (VNAME(1)) (A(B));
53 1 3
      IF VNAME(1) = (B)' THEN GO TO SEL_COM;
54 1 3
      END;
55 1 3
      END;
56 1 2
      END;
57 1 1
      END;
58 1 0
      END DEFINE;
%INCLUDE DEFDOM:*************************************************************************************/
*/
*/

MODULE DESCRIPTION
********************************************************************************/

DEFDOM: PROCEDURE

(TNAME, /* CHAR(8) */, 
CTYPE, /* CHAR(8) */, 
CLEN, /* CHAR(8) */, 
UMIN, /* CHAR(S) */, 
CMAX /* CHAR(M) */);

*******************************************************************************

PURPOSE:

THE PURPOSE OF THIS PROCEDURE IS TO ACCEPT REQUESTS TO DEFINE DOMAINS, VALIDATE THE REQUESTS, THEN ISSUE A CALL TO DEFINED 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 VARG 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. DEFINED 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

UMIN - IF NUMERIC DATA, THE MINIMUM PERMISSABLE VALUE.

CMAX - IF NUMERIC DATA, THE MAXIMUM PERMISSABLE VALUE.
OUTPUT PARAMETERS:

NONE, HOWEVER, IT DOES CALL DEFINEV TO DEFINE A VALUE SET DEFO0480 CORRESPONDING TO THE DOMAIN, AND IT DOES CALL INSERTN TO INSERT THE DOMAIN DESCRIPTION INTO THE NSET REPRESENTATION DEFO00490 OF DTABLE.

CALLS PROCEDURES:

INSERTN, DEFINEV, FETCHV

*****************************************************

/* INSERTN TABLE */

DCL 1 INSERT_ARG,

2 NNAME BIT(64) INIT(UNSPEC('DTABLE ')),

2 NATTR BIT(8) INIT('00000010'B),

2 ATTR(20),

3 ANAME BIT(64) INIT(UNSPEC('DNAME ')),

3 VALUE BIT(320):

/* DEFINEV TABLE */

DCL 1 DV_ARG,

2 VNAME BIT(64),

2 KEY_LEN BIT(8);

/* FETCHV TABLE */

DCL 1 FV_ARG,

2 DNAME BIT(64) INIT(UNSPEC('NAME ')),

2 KEY_VAL BIT(160),

2 FOUND BIT(1),

2 DATA BIT(320);

/* TEMPORARY VARIABLES */

DCL TNAME CHAR(8),

1 RECORD,

2 ITYPE BIT(8),

2 TLEN BIT(16),

2 THIN BIT(16),

2 TMAX BIT(16),

DATR BIT(56) DEFINED RECORD,

(CTYPE,CLEN,CMIN,CMAX) CHAR(8),

BTYPE CHAR(1),

(ULEN,MIN,MAX) FIXED BIN(16),

KLEN FIXED BIN(8);

/* SUBROUTINES */

%INCLUDE EINSER;*****************************************************

DEF00380
/* INSERT NSET MODULE */

DCL INSERTN ENTRY(1, 2 BIT(64), 2 BIT(8), 2 (20),
3 BIT(64), 3 BIT(320));

*********************
/* INCLUDE EDEFINE */
*********************

DCL DEFINEV ENTRY(1, 2 BIT(64), 2 BIT(8));

*********************
/* INCLUDE EFETCH */
*********************

DCL FETCHV ENTRY(1, 2 BIT(64), 2 BIT(160),
2 BIT(1), 2 BIT(320));

********************

ON CONDITIONS
BEGIN;

IF TNAME(1) = (8)' ' THEN DO:
    PUT SKIP(I) LIST ('DOMAIN NAME:');
    GET EDIT (TNAME) (A(8));
    IF TNAME = (8)' ' THEN RETURN;
END;

VALIDATE DOMAIN NAME

KEY_VAL = UNSPEC(TNAME);
CALL FETCHV(FV_ARG);
IF FOUND THEN DO;
    PUT SKIP(I) LIST ('DOMAIN ALREADY DEFINED.');
    RETURN;
END;

GET DATA TYPE

BTYPE = CTYPE;
DO WHILE (BTYPE ^= 'C' & BTYPE ^= 'N');
    PUT SKIP(I) LIST ('DATA TYPE:');
    GET EDIT (BTYPE) (A(1));
    IF BTYPE ^= ' ' THEN RETURN;
END;
4
IM
IM
IM
/* GET MAXIMUM LENGTH */
GET
IF VERIFY(CLEN,'0123456789') = 0
THEN IF CLEN = (8)' = 0
THEN BLEN = 0;
ELSE BLEN = BIN(CLEN);
ELSE BLEN = 0;
DO WHILE(BLEN < 1 : BLEN > 40);
PUT SKIP LIST ('MAXIMUM LENGTH: ');
GET LIST (BLEN);
IF 'ULEN = 0 THEN RETURN;
END;

/* CHECK IF NUMERAL DATA TYPE */
IF BTYP = 'N'
THEN DO:
/* GET MINIMUM AND MAXIMUM VALUES */
IF VERIFY(CMIN,'0123456789') = 0 & CMIN = (8)' THEN
CMIN = BIN(CMIN);
ELSE DO;
PUT SKIP LIST ('MINIMUM VALUE: ');
GET LIST (BMIN);
END;
IF VERIFY(CMAX,'0123456789') = 0 THEN
CMAX = (8)' THEN
CMAX = SMIN - 1;
ELSE BMAX = BIN(CMAX);
ELSE BMAX = BMIN - 1;
DO WHILE(BMAX < BMIN);
PUT SKIP LIST ('MAXIMUM VALUE: ');
END;
END;
END;

/* CALL INSERTN TO ADD NEW DOMAIN TO TABLE */
TYP = UNSPEC(BTYP); TLEN = BLEN; TMIN = BMIN; TMAX = BMAX;
VALUE(1) = UNSPEC(TNAM);
VALUE(2) = DATTR;
CALL INSERTN(INSET_ARG);
/* DETERMINE KEY LENGTH */
/* CALL DEFINEV TO CREATE A PSET */
VNAME = UNSPEC(TNAM); KEY_LEN = '01000000'B;
CALL DEFINEV(DV_ARG);

67 1 0  END DEFDOM;

DEF01250
%INCLUDE DEPREL;******************************************************************************

MODULE DESCRIPTION
*******************************************************************************/

DEFREL: PROCEDURE

(N, /* FIXED BIN(15) */
TNAME /* (20) CHAR(8) */);

// PURPOSE:
*****
***** THIS MODULE IS RESPONSIBLE FOR PROCESSING REQUESTS TO DEFINE
***** RELATIONS. IT VALIDATES THE REQUEST ( I.E CHECKS
***** THAT ALL DOMAINS EXIST AND THAT RELATION NOT PREVIOUSLY
***** DEFINED), ISSUES A CALL TO DEFINEN TO DEFINE THE NSET
***** REPRESENTATION OF THE RELATION, AND THEN ISSUES A CALL
***** TO INSERTN TO INSERT RELATION DEFINITION INTO RTABLE.
*****
***** METHOD:
*****
***** LOGIC IS SIMILAR TO THAT OF DEFOOM.NOTES:
***** A) PRE-SPECIFIED COPY OF INSERT_ARG USED TO INSERT TUPLES
***** INTO RTABLE. INSERTN MUST BE CALLED ONCE FOR EACH DOMAIN
***** IN THE RELATION.
*****
***** INPUT PARAMETERS:
*****
***** ARGUMENTS CREATED BY LEX DURING ITS SCAN OF THE COMMAND
***** ISSUED BY USER. ARGUMENTS NOT GIVEN BY USER ARE PASSED
***** AS BLANKS.
***** TNAME- ESSENTIALLY TOKEN CHAIN CREATED BY LEX. CONTAINS
***** NAME OF RELATION AND NAMES OF DOMAINS WITHIN RELATION.
***** N - NUMBER OF TOKENS IN CHAIN.
*****
***** OUTPUT PARAMETERS:
*****
***** NONE, BUT VIA ITS CALL TO INSERTN, IT INSERTS THE RELATION
***** DEFINITION INTO RTABLE, AND VIA ITS CALL TO DEFINEN IT
***** CREATES AN NSET DEFINITION FOR THE NSET REPRESENTATION OF
***** THE RELATION.
*****
***** CALLS PROCEDURES:
*****
***** DEFINEN, INSERTN, FETCHV, LEX

*******************************************************************************/
/
*/  DEFINEN TABLE */

%INCLUDE DEFO00040

DCL 1 DEF_ARG, */ USED TO DEFINE AN NSET */ DCL00010
2 NAME BIT(64), /* NAME OF NSET */ DEFO0020
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

******

/*/ INSERTN TABLE */

%INCLUDE DEFO00040

DCL 1 INSERT_ARG, */ USED TO INSERT AN NSET */ DCL00010
2 NAME BIT(64) INIT(UNSPEC('RTABLE ')), DEFO0060
2 NATTR BIT(8) INIT('00000010'B), DEFO0090
2 ATTR(20), DEFO100
3 ANAME BIT(64) INIT(UNSPEC('RNAME ')), DEFO110
3 VALUE BIT(320); DEFO120
DCL00130
DCL00140
DCL00150

******

/*/ FETCHV TABLE */

%INCLUDE DEFO00040

DCL 1 FV_ARG, */ USED TO FETCH INSTANCES OF A DOMAIN */ DCL00010
2 NAME BIT(64), /* NAME OF DOMAIN */ DEFO0030
2 KEY_VAL BIT(160), /* KEY TO SEARCH ON */ DEFO0040
2 FOUND BIT(1), /* IF FOUND, '1' OTHERWISE '0' */ DEFO0050
2 DATA BIT(320); /* RETRIEVED ELEMENT */ DEFO0060
DCL00070
DCL00160
DCL00170

******

/*/ SUBROUTINES */

%INCLUDE DEFO00040

DCL 1 ENTRY(l). /* DEFINE NSET MODULE */ DCL00010
2 (20), 3 BIT(64), 3 BIT(8)); DCL00180
DCL00190
DCL00200
DCL00100

******

%INCLUDE DEFO00200

/*/ INSERT NSET MODULE */

DCL 1 ENTRY(l). /* INSERT NSET MODULE */ DCL00010
2 (20), 3 BIT(64), 3 BIT(320)); DCL00200
DCL00080
DCL00090

******

%INCLUDE DEFO00210

/*/ USED TO FETCH INSTANCES OF DOMAINS */

DCL 1 ENTRY(l). /* FETCHN INSTANCES OF DOMAINS */ DCL00010
2 (20), 3 BIT(64), 3 BIT(320)); DCL00200
DCL00030
DCL00050
DCL00070

******


%INCLUDE ELEX;******************************************************DEF00220
8 1 0
DCL LEX ENTRY(FIXED BIN(15),(*)) CHAR(8));
*************DEF00010
DECO010

DEFO010

/* MISCELLANEOUS */
9 1 0
DCL TNAME(20) CHAR(8),
TREL CHAR(B),
IND(10) FIXED BIN(15),
THNK FIXED BIN(8),
(1,1,N2) FIXED BIN(15),
TRUE BIT(1) INIT('1'B);
DEFO020

DEFO020

/* START SUBROUTINE */
10 1 0
TREL = TNAME(2);
DEFO030

DEFO030

/* GET RELATION NAME */
11 1 0
IF N = 1 THEN DO;
12 1 1
PUT SKIP LIST('RELATION NAME:');
13 1 1
GET EDIT (TREL) (A(8));
14 1 1
IF TREL = (8)' THEN RETURN;
15 1 1
END;
DEFO040

DEFO040

/* VALIDATE RELATION NAME */
16 1 0
D_NAME = UNSPEC('RNAME '); KEY_VAL = UNSPEC(TREL);
17 1 0
CALL FETCHV(FVARG);
18 1 0
IF FOUND THEN DO:
19 1 1
PUT SKIP LIST('RELATION ALREADY DEFINED.');
20 1 1
RETURN;
21 1 1
END;
DEFO050

DEFO050

/* MOVE DOMAIN NAMES DOWN IN ARRAY */
22 1 0
DO I = 3 TO 12;
23 1 1
TNAME(I-2) = TNAME(I);
24 1 1
END;
DEFO060

DEFO060

/* CHECK FOR CORRECT NUMBER OF DOMAINS */
25 1 0
IF N > 10 THEN PUT SKIP(0) LIST ('TOO MANY DOMAINS SPECIFIED.');
26 1 0
IF N > 10 THEN DO WHILE(TRUE);
27 1 1
IF N < 10 THEN LEAVE;
28 1 1
PUT SKIP(0) LIST ('TOO MANY DOMAINS SPECIFIED.');
29 1 1
CALL LEX(N,TNAME);
30 1 1
CALL LEX(N,TNAME);
31 1 1
IF N = 0 THEN RETURN;
32 1 1
IF N < 10 THEN LEAVE;
33 1 1
PUT SKIP(0) LIST ('TOO MANY DOMAINS SPECIFIED.');
34 1 1
DEFO070

DEFO070

DEFO070

DEFO070

DEFO070

DEFO070
/* VALIDATE DOMAIN NAMES */
D_NAME = UNSPEC('DNAME ');
DO I = 1 TO N;
  DO WHILE(TRUE):
    KEY_VAL = UNSPEC(TNAME(I));
    CALL FETCHV(FV_ARG);
    IF NOT FOUND THEN LEAVE;
    PUT SKIP EDIT ('DOMAIN ',TNAME(I),' UNDEFINED:');
    (A,A,A);
    GET EDIT (TNAME(I)) (A(B));
  END;
END;

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

/* CALL DEFINEN TO CREATE A NEW NSET */
DEF ARG.NNAME = UNSPEC(TREL); TNUM = N; DEF_ARG.NATTR = TNUM; 
DO I = 1 TO N;
  DEF_ARG.ANAME(I) = UNSPEC(TNAME(I)); 
END;

/* CALL DEFINEN TO ADD NEW RELATION TO TABLE */
K_TYPE = '00000000'B;
DO I = 1 TO N2:
  IF VERIFY(TNAME(I), '123456789') = 0 &
    BIN(TNAME(I)) >= 1 & BIN(TNAME(I)) <= N2 
  THEN K_TYPE(BIN(TNAME(I))) = '00000001'B;
END;
END DEFINEN(DEF_ARG);
END DEFREL;
**Purpose:**
This module is responsible for processing requests to define views, where a view is a collection of previously defined relations to which a user may have access. Every view is identified by a unique ID. This module validates the request (checks that the relations exist, and that the view ID is unique), and then inserts the view description into the VTABLE via a call to INSERTN.

**Method:**
The logic is virtually the same as DEFREL and DEFDOM.

**Input Parameters:**
- Arguments passed are created by LEX during its scan of the users command line.
- N - the number of tokens found in the command line.
- TNAME - the token chain created by LEX containing the view ID and the name of the relations contained in the view.

**Output Parameters:**
- None, however, it does insert entries into the VTABLE as a result of its calls to INSERTN. These entries correspond to the view definitions.

**Calls Procedures:**
- INSERTN, FETCHV, LEX

---

```plaintext
/* INSERTN Table */
DCL 1 INSERT_ARG,
  2 NNAME BIT(64) INIT(UNSPEC('VTABLE ')),
  2 *ATT1 BIT(8) INIT('00000010'B),
```
2 ATTR(20),
3 ANAME BIT(64) INIT(UNSPEC('ID '),
    UNSPEC('RNAME ')),
3 VALUE BIT(320);

/* FETCH TABLE */
%INCLUDE FVAR;****************************************************************
/* FETCH' 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' , OTHERWISE '0' */
  2 DATA BIT(320); /* RETRIEVED ELEMENT */
***************

/* SUPROUTINES */
%INCLUDE EINSRN;************************************************************************
/* 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(B));
***************

/* MISCELLANEOUS */
7 1 0 DCL TNAME(20) CHAR(B),
  TVIEW CHAR(B),
  (1,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;
    PUT SKIP LIST('VIEW ID:');
10 1 1 GET EDIT (TVIEW) (A(B));
11 1 1 IF TVIEW = (B) THEN RETURN;
12 1 1 END;
13 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(O) LIST ('TOO MANY RELATIONS SPECIFIED. ');
IF N > 10 ; N < 1 THEN DO WHILE(TRUE):
  PUT SKIP(O) LIST ('RELATION NAMES: ');
  CALL LEX(N,TNAME);
  IF N = 0 THEN RETURN;
  IF N < 11 THEN LEAVE;
  PUT SKIP(O) 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;
    IF NOT FOUND THEN LEAVE;
    PUT SKIP EDIT ('RELATION ',TNAME(I), ' UNDEFINED: ');
    (A,A,A): GET EDIT (TNAME(I))(A(0));
    IF TNAME(I) = (0) 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 INSERTNINSERT_ARG);
%INCLUDE QUERY;******************************************************************************QUE00010
/**** MODULE DESCRIPTION */DOC00310

1 0 QUERY: PROCEDURE;
/******************************************************************************DOC00320
**** PURPOSE: PROMPTS FOR DISPLAY, INSERT, RELATIONAL OPERATOR, AND DOC00330
**** PRINT COMMANDS.
****
**** METHOD: NOT SIGNIFICANT
****
**** INPUT PARAMETERS: DOC00340
****
**** OUTPUT PARAMETERS: DOC00350
****
**** OUTPUT: NONE
****
**** CALLS PROCEDURES:
****
**** GETVIEW, VIEWCAT, SHVIEW, SHREL, JOIN, SELECT,
*****
******************************************************************************DOC00360

/* CURRENT VIEW ID */
DCL C_ID CHAR(8) EXTERNAL,
   GET BIT(1) STATIC EXTERNAL INIT('I'B);QUE00010

/* COMMAND LISTS */
DCL OP(10) CHAR(B) INITIAL('GET_VIEW', 'CUR_VIEW',
   'VIEW_CAT', 'SH_VIEW', 'SH_REL', 'JOIN',
   'SELECT', 'PROJECT', 'PRINT', 'LOAD'),
   AOP(10) CHAR(8) INITIAL('GV', 'CV', 'VC', 'SV', 'SR',
   'J', 'S', 'PJ', 'PR', 'L');QUE00010

/* SUBROUTINES */
%INCLUDE EGETVIEW;******************************************************************************QUE000150

2 1 0
DCL GETVIEW ENTRY(CHAR(B));QUE00020

3 1 0
%INCLUDE ESHVIEW;******************************************************************************QUE000160
DCL SHVIEW ENTRY;

%INCLUDE ESHREL;

DCL SHREL ENTRY(CHAR(B));

%INCLUDE EJOIN;

DCL JOIN ENTRY((*) CHAR(B));

%INCLUDE ESELECT;

DCL SELECT ENTRY((*) CHAR(B));

%INCLUDE EPROJECT;

DCL PROJECT ENTRY((#) CHAR(B));

%INCLUDE EPRINT;

DCL PRINT ENTRY(CHAR(B));

%INCLUDE ELDAD;

DCL LOAD ENTRY(CHAR(B));

%INCLUDE EDELIM;

DCL DELIM ENTRY(BIT(I)) RETURNS(BIT(1));

DCL NAME(20) CHAR(B), N_TOK FIXED BIN(15), TRUE BIT(1) INIT('1'B);

/* START SUBROUTINE */

PUT SKIP(2) LIST ('-- READY FOR QUERIES -- ');

/* GET QUERIES */

DO WHILE(TRUE);

PUT SKIP(2) LIST ('Q: ');

CALL LEX(N_TOK,NAME);

IF N_TOK = 0 THEN RETURN;

/* SELECT COMMAND */

SELECT(NAME(1));

/* GET VIEW */

WHEN(OP(1),AOP(1)) CALL GETVIEW(NAME(2));
Y:

22 1 2 /* CURRENT VIEW */
\hline
\hline
\hline
WHEN(OP(2),AOP(2))
\hline
IF GET
\hline
THEN PUT SKIP LIST ('NO VIEW LOADED YET.');
\hline
ELSE PUT SKIP EDIT (C_ID) (X(10),A);
\hline
\hline
23 1 2 /* VIEW CATALOGUE */
\hline
\hline
\hline
WHEN(OP(3),AOP(3)) CALL VIEWCAT;
\hline
\hline
24 1 2 /* HOW VIEW */
\hline
\hline
\hline
WHEN(OP(4),AOP(4))
\hline
IF GET
\hline
THEN PUT SKIP LIST ('NO VIEW LOADED YET.');
\hline
ELSE CALL SHVIEW;
\hline
\hline
25 1 2 /* SHOW RELATION */
\hline
\hline
\hline
WHEN(OP(5),AOP(5))
\hline
IF GET
\hline
THEN PUT SKIP LIST ('NO VIEW LOADED YET.');
\hline
ELSE CALL SHREL(NAME(2));
\hline
\hline
26 1 2 /* JOIN */
\hline
\hline
\hline
WHEN(OP(6),AOP(6))
\hline
IF DELIM('I'B) = 'O'S THEN CALL JOIN(NAME);
\hline
\hline
27 1 2 /* SELECT */
\hline
\hline
\hline
WHEN(OP(7),AOP(7))
\hline
IF DFILT('O'B) = 'O'S THEN CALL SELECT(NAME);
\hline
\hline
28 1 2 /* PROJECT */
\hline
\hline
\hline
WHEN(OP(8),AOP(8))
\hline
IF DELIM('O'B) = 'O'S THEN CALL PROJECT(NAME);
\hline
\hline
29 1 2 /* PRINT */
\hline
\hline
\hline
WHEN(OP(9),AOP(9)) CALL PRINT(NAME(2));
\hline
\hline
30 1 2 /* LOAD */
\hline
\hline
\hline
WHEN(OP(10),AOP(10))
\hline
IF GET
\hline
THEN PUT SKIP LIST ('NO VIEW LOADED YET.');
\hline
ELSE CALL LOAD(NAME(2));
\hline
\hline
31 1 2 /* INVALID OPERATORS */
\hline
\hline
\hline
OTHERWISE
\hline
\hline
\hline
PUT SKIP EDIT (NAME(1),' IS AN INVALID COMMAND') (A,A); QUE00910
\hline
\hline
32 1 3 PUT SKIP LIST ('RETYPE COMMAND:'); QUE00920
\hline
\hline
33 1 3 GET EDIT (NAME(1)) (A(8)); QUE00930
\hline
\hline
34 1 3 IF NAME(1) = (8) ' THEN GO TO SEL_COM; QUE00940
\hline
\hline
35 1 3 DO;
\hline
\hline
36 1 3 PUT SKIP EDIT (NAME(1),' IS AN INVALID COMMAND') (A,A); QUE00910
\hline
\hline
37 1 3 PUT SKIP LIST ('RETYPE COMMAND:'); QUE00920
\hline
\hline
38 1 3 GET EDIT (NAME(1)) (A(8)); QUE00930
\hline
\hline
39 1 3 IF NAME(1) = (8) ' THEN GO TO SEL COM; QUE00940
\hline
\hline
40 1 3
Y:

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

QUE00950
QUE00960
QUE00970
QUE00980
QUE00990
%INCLUDE GETVIEW;  
/*************************************************************/ GET00010  
*  
*  MODULE DESCRIPTION  
*  1  
*  GETVIEW: PROCEDURE(TNAME /* CHAR(B) */);  
/**** PURPOSE:  
*  LOAD A VIEW FOR QUERIES BY CREATING RELATION TABLES  
*  ALONG WITH THEIR DOMAINS FROM ALL RELATIONS IN THE  
*  PARTICULAR VIEW.  
*  
*  METHOD:  
*  NOT SIGNIFICANT  
*  
*  INPUT PARAMETERS:  
*  1) TNAME - NAME OF VIEW  
*  
*  OUTPUT PARAMETERS:  
*  NONE  
*  
*  CALLS PROCEDURES:  
*  FECHT, FECHTV  
*  
*************************************************************/ GET00010  
/  
*  */ CURRENT VIEW INDEX */  
*  DCL C_ID CHAR(B) EXTERNAL;  
*  GET BIT(1) STATIC EXTERNAL;  
*  
*  */ FECHT TABLE */  
*  %INCLUDE RETARG;  
*  /* USED TO RETRIEVE NSETS */  
DCL 1 RET_ARG,  
*  /* NUMBER OF NSETS */  
2 NUMN BIT(8),  
*  /* NAMES OF NSETS TO BE FETCHED*/  
2 NSET(5) BIT(64),  
*  /* INFO FOR EACH ATTRIBUTE */  
2 ARGS(20),  
*  /* WHICH NSET IS THIS IN */  
3 N_INDEX BIT(8),  
*  /* NAME OF ATTRIBUTE */  
3 NAME BIT(64),  
*  /* RETRIEVE INFORMATION */  
3 RET_INFO.  
*  /* IS IT TO BE FETCHED */  
(4 FETCH,  
*  /* SAME AS PREVIOUSLY */  
4 SAME ) BIT(8).
DEFINED DOMAIN */

4 VALUE BIT(160); /* VALUE TO SEARCH ON OR

NONE */

**************

*/ FETCHT OUTPUT */

%INCLUDE DOMRET;*******************************************************************************/

GET00110

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 */

DCL 1 FV_ARG,

2 D_NAME BIT(64),

2 KEY_VAL BIT(160),

2 FOUND BIT(1).,

2 DUMMY BIT(320);

**************

*/ 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,

R_EL(20) CHAR(8) EXTERNAL,

NDOM(0:20) FIXED BIN(15) EXTERNAL;

**************

*/ SUBROUTINES */

%INCLUDE EFETCHT;*******************************************************************************/

GET00350

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(160)); DEC00120

DECO00110

**************

%INCLUDE EFETCHV;*******************************************************************************/

GET00360

GET00370

8 1 0

DCL FETCHV ENTRY(1, 2 BIT(64), 2 BIT(160),

2 BIT(1), 2 BIT(320));

**************

*/ MISCELLANEOUS */
I4
9 1 0
DCL TNAME CHAR(8),
(BUFF, TEMP) BIT(64),
(I.D.IND) FIXED BIN(15);
/* START SUBROUTINE */
10 1 0
/* GET VIEW ID */
11 1 1
IF TNAME = (8) THEN
12 1 1
PUT SKIP LIST ('VIEW ID:');
13 1 1
GET (I.OIND) FIXED BIN(15);
14 1 1
END;
15 1 0
/* VALIDATE VIEW ID */
16 1 0
D_NAME = UNSPEC('ID '); KEY_VAL = UNSPEC(TNAME);
17 1 0
IF 'FOUND THEN
18 1 1
PUT SKIP EDIT ('ID ', TNAME, 'UNDEFINED.') (A(A,A));
19 1 1
RETURN;
20 1 1
END;
21 1 0
C_ID = TNAME; GET = '0'B;
22 1 0
/* CALL FETCH TO LOAD VIEW */
23 1 0
NUMN = '00000010'B; NSET(1) = UNSPEC('RTABLE ');
24 1 0
NSET(2) = UNSPEC('TABLE ');
25 1 0
N_INDEX(1), N_INDEX(2) = '00000001'B;
26 1 0
N_INDEX(3), N_INDEX(4) = '00000010'B;
27 1 0
NAME(2) = UNSPEC('DNAME ');
28 1 0
NAME(1), NAME(3) = UNSPEC('RNAME ');
29 1 0
NAME(4) = UNSPEC('ID ');
30 1 0
FETCH(4) = '00000000'B;
31 1 0
FETCH(2), FETCH(3), FETCH(1) = '10000000'B;
32 1 0
SAME(1), SAME(2), SAME(4) = '00000000'B;
33 1 0
SAME(3) = '00010000'B;
34 1 0
RET_ARG.VALUE = '01010101'B; RET_ARG.VALUE(4) = UNSPEC(TNAME);
35 1 0
CALL FETCH(RET_ARG);
36 1 0
/* CREATE RET_ARGS FOR RELATIONS */
37 1 0
N1 = '00000001'B; N2 = '00000001'B; N3 = '10000000'B;
38 1 0
N4 = '00000000'B; C3 = '01010101'B;
39 1 0
TEMP = UNSPEC('B ');
40 1 0
R_IND, D_IND = 0;
41 1 0
DO I = 1 TO ALLOCATION(DOMRET)/2:
42 1 1
BUFF = DOM.RET.VALUE;
43 1 1
IF BUFF = TEMP THEN DO;
44 1 1
DDOM(R_IND) = D_IND;
45 1 2
GET00400
GET00410
GET00420
GET00430
GET00440
GET00450
GET00460
GET00470
GET00480
GET00490
GET00500
GET00510
GET00520
GET00530
GET00540
GET00550
GET00560
GET00570
GET00580
GET00590
GET00600
GET00610
GET00620
GET00630
GET00640
GET00650
GET00660
GET00670
GET00680
GET00690
GET00700
GET00710
GET00720
GET00730
GET00740
GET00750
GET00760
GET00770
GET00780
GET00790
GET00800
GET00810
GET00820
GET00830
GET00840
GET00850
GET00860
GET00870
GET00880
50 1 2  R_IND = R_IND + 1;
51 1 2  CI(R_IND, I) = BUFF;
52 1 2  D_IND = 0;
53 1 2  UNSPEC(REL(R_IND)) = BUFF;
54 1 2  TEMP = BUFF;
55 1 2  END;
56 1 1  FREE DOMRET;
57 1 1  D_IND = D_IND + 1;
58 1 1  C2(R_IND, D_IND) = DOMRET.VALUE;
59 1 1  FREE DOMRET;
60 1 1  END;
61 1 0  NDOM(R_IND) = D_IND;
62 1 0  PUT SKIP EDIT ('VIEW LOADED.') (X(10), A);
63 1 0  END GETVIEW;
%INCLUDE SHVIEW;********************************************************************
 MODULE DESCRIPTION

 SHVIEW: PROCEDURE;

 PURPOSE.

 DISPLAY ALL RELATIONS, PERMANENT AND TEMPORARY, IN
 THE CURRENT VIEW.

 METHOD:

 NOT SIGNIFICANT

 INPUT PARAMETERS:

 NOT SIGNIFICANT

 OUTPUT PARAMETERS:

 NONE

 CALLS PROCEDURES:

 NONE

********************************************************************

 /* RELATION NAMES */
 DCL REL(20) CHAR(8) EXTERNAL,
  R_IND FIXED BIN(15) EXTERNAL,
  I FIXED BIN(15);

 /* PRINT RELATIONS */
 DO 1 = 1 TO R_IND;
  PUT SKIP EDIT (REL(I)) (X(10),A);
  END;
  END SHVIEW;
%INCLUDE SHREL:***************************************************************************$RHR00010
/*=============================================================================*/
* MODULE DESCRIPTION
**==============================================*/
** SHREL: PROCEDURE(TNAME /* CHAR(B) */);**
/** PURPOSE
** DISPLAY ALL DOMAINS WITH THEIR ATTRIBUTES OF A GIVEN
** RELATION.**
**==============================================*/
*** METHOD:
** NOT SIGNIFICANT**
**==============================================*/
*** INPUT PARAMETERS:
** 1) TNAME - NAME OF RELATION**
**==============================================*/
*** OUTPUT PARAMETERS:
** NONE**
**==============================================*/
*** CALLS PROCEDURES:
**==============================================*/
*** FETCHT**
**==============================================*/
**=============================================================================*/
/* FETCHT TABLE */
DCL 1 RET_ARG,
  2 NUMN BIT(B) INIT('00000001'B),
  2 NSET(5) BIT(64) INIT(UNSPEC('DTABLE ')),
  2 ATTR(20),
  3 N_INDEX BIT(8) INIT((2)('00000001'B)),
  3 NAME BIT(64) INIT(UNSPEC('DNAME ')),
  3 RET_INFO,
  4 FETCH BIT(B) INIT('00000000'0B,'10000000'0B),
  4 SAME BIT(B) INIT((2)('00000000'0B)),
  4 FVALUE BIT(160);
/* FETCHT OUTPUT */
%INCLUDE DOMRET;***************************************************************************$RHR00170
/* FETCHT OUTPUT */
%INCLUDE SRLe**********$RHR0020
3 1 0  DCL 1 DOM.RET CONTROLED 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 F3,
    4 N3 BIT(8),
    4 N4 BIT(8),
    4 C3 BIT(100),
    R_IND FIXED BIN(15) EXTERNAL,
    REL(20) CHAR(8) EXTERNAL,
    NDOM(0:20) FIXED BIN(15) EXTERNAL;

        /* DOAIN ATTRIBUTES MASK */

5 1 0  DCL 1 RECORD,
    2 ITYPE BIT(8),
    2 TLEN BIT(16),
    2 TMIN BIT(16),
    2 TMAX BIT(16),
    DATTR BIT(56) DEFINED RECORD,
    TYPE CHAR(1),
    (B,I,E,N,B,M,I,N,B,M,A,X) FIXED BIN(16):

        /* SUBROUTINES */

6 1 0  DCL FETCH 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 CHAR(8),
    (I,J) FIXED BIN(15):

        /* VALIDATE RELATION NAME */

8 1 0  DO I = 1 TO R.IND;
9 1 1  IF TNAME = REL(I) THEN LEAVE;
10 1 1  END;
11 1 0  IF I > R.IND + 1
12 1 1  THEN GO;
13 1 1  PUT SKIP LIST ("RELATION NOT IN VIEW.");
14 1 1  RETURN,
15 1 1  END.

        /* PRINT OUT HEADER */


15 1 0  PUT SKIP EDIT ('DOMAIN','TYPE','LEN','MIN','MAX')
    (X(5),A,X(14),A,X(8),A,X(8),A,X(8),A);
SHR00640
16 1 0  PUT SKIP EDIT ('********','********','********','********')
    (X(5),A,X(14),A,X(8),A,X(8),A,X(8),A);
SHR00650
17 1 0  FVALUE(2) = '01010101'B;
SHR00660
18 1 0  /* PRINT OUT DOMAINS AND ATTRIBUTES */
19 1 1  DO J = 1 TO NDOM(I);
20 1 2  IF N(I,J) = '10000000'B & H4(I,J) = '00000000'B
21 1 2  THE I DO;
22 1 2  FVALUE(1) = C2(I,J);
SHR00720
23 1 2  CALL FETCH(TRET_ARG);
SHR00730
24 1 2  DATR = VALUE;
SHR00740
25 1 2  UNSPEC(TYPE) = BYTE; BLEN = TLEN;
SHR00750
26 1 2  BMIN = TMIN; BMAX = TMAX;
SHR00760
27 1 2  FREE DOM_RET;
SHR00770
28 1 2  UNSPEC(TNAME) = C2(I,J);
SHR00780
29 1 2  IF TYPE = 'C'
29 1 2  THEN PUT SKIP EDIT (TNAME,'CHAR',BLEN,'----','----')
29 1 2  (X(5),A,X(12),A,X(9),F(2),X(9),A,X(9),A);
SHR00790
30 1 2  ELSE PUT SKIP EDIT (TNAME,'NUM',BLEN,BMIN,BMAX)
30 1 2  (X(5),A,X(13),A,X(9),F(2),X(9),F(2),X(9),F(2));
SHR00800
31 1 2  END;
SHR00810
32 1 1  END;
SHR00820
33 1 0  END SHREL;
%INCLUDE LOAD;

***** MODULE  DESCRIPTION *****

1 0 LOAD:  PROCEDURE(TNAME /* CHAR(8) */);

***** PURPOSE: *****
TO INSERT TUPLES OF DATA INTO A GIVEN RELATION.

***** METHOD: *****
NOT SIGNIFICANT

***** INPUT PARAMETERS: *****
1) TNAME - NAME OF RELATION

***** OUTPUT PARAMETERS: *****
NONE

********** CALLS PROCEDURES: **********
FETCH, INSERTN, LEX2

******************************

1/* INSERTN TABLE */
DCL 1 INSERT_ARG,
2 NNAME BIT(64),
2 NATTR BIT(8),
2 ARG(20),
3 INAME BIT(64),
3 IVALUE BIT(320);

2/* FETCHT TABLE */
DCL 1 RET_ARG,
2 NUMN BIT(8),
2 NSET(5) BIT(64),
2 ATTR(20),
3 N_INDEX BIT(8),
3 NAME BIT(64),
3 RET_INFO,
4 FETCH BIT(8),
4 same bit(8),
4 f value bit(160):

/* fetch output */
DCL 1 domain_ret controlled external,
2 d_id fixed bin(15),
2 value bit(320);

/* subroutines */
%include einsert;

/* insert nset module */
DCL insertn entry(1, 2 bit(64), 2 bit(8), 2 (20),
3 bit(64), 3 bit(320));

/* fetch nset module */
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 efetcht: */
%include ex2;

/* lexical analyzer */
DCL lex2 entry(fixed bin(15), (*) char(40), (*) fixed bin(15));

/* domain attributes table */
DCL 1 record,
2 type bit(8),
2 l en bit(16),
2 t mn bit(16),
2 t max bit(16),
data bit(56) defined record,
type(10) char(1),
blend(10) fixed bin(16),
blmin(10) fixed bin(16),
blmax(10) fixed bin(16);

/* current view id */
DCL c_id char(8) external;

/* miscellaneous */
DCL tframe char(8),
dum(20) char(8),
data(20) char(40) init( (20)((40)' ')),
l(20) fixed bin(15),
tmp(20) char(40) init( (20)((40)' ')),
tl(20) fixed bin(15),
st char(80) varying,
s fixed bin(8),
(j,n,n2) fixed bin(15),

/* miscellaneous */
DCL tframe char(8),
dum(20) char(8),
data(20) char(40) init( (20)((40)' ')),
l(20) fixed bin(15),
tmp(20) char(40) init( (20)((40)' ')),
tl(20) fixed bin(15),
st char(80) varying,
s fixed bin(8),
(j,n,n2) fixed bin(15),

/* miscellaneous */
DCL tframe char(8),
dum(20) char(8),
data(20) char(40) init( (20)((40)' ')),
l(20) fixed bin(15),
tmp(20) char(40) init( (20)((40)' ')),
tl(20) fixed bin(15),
st char(80) varying,
s fixed bin(8),
(j,n,n2) fixed bin(15),

/* miscellaneous */
DCL tframe char(8),
dum(20) char(8),
data(20) char(40) init( (20)((40)' ')),
l(20) fixed bin(15),
tmp(20) char(40) init( (20)((40)' ')),
tl(20) fixed bin(15),
st char(80) varying,
s fixed bin(8),
(j,n,n2) fixed bin(15),
TRUE BIT(1) INIT('1'B);

/* START LOAD */

/* GET RELATION NAME */
11 1 0 IF TNAME = (8) THEN DO;
12 1 1 PUT SKIP LIST ('RELATION NAME: ');
13 1 1 GET EDIT (TNAME) (A(B));
14 1 1 IF TNAME = (8) THEN RETURN;
15 1 1 END;

/* VALIDATE RELATION NAME */
16 1 0 NUMN = '00000001'B; NSET(1) = UNSPEC('VTABLE ');
18 1 0 N_INDEX(1),N_INDEX(2) = '00000001'B;
19 1 0 NAME(1) = UNSPEC('TNAME ');
21 1 0 FVALUE(1) = '00000000'B; FVALUE(2) = '10000000'B;
23 1 0 NAME(1),NAME(2) = '00000000'B;
24 1 0 FVALUE(1) = UNSPEC('TNAME ');
26 1 0 CALL FETCHT(RET_ARG);
27 1 0 IF ALLICATION(DOM RET) = 0 THEN DC;
28 1 1 PUT SKIP LIST ('RELATION NOT IN VIEW.');
29 1 1 RETURN;
30 1 1 END;
31 1 0 FREE DOM_RET;

/* GET DOMAINS AND ATTRIBUTES OF RELATION */
32 1 0 NUMN = '00000001'B; NSET(1) = UNSPEC('RTABLE ');
34 1 0 N_INDEX(1),N_INDEX(2) = '00000001'B;
35 1 0 NAME(1) = UNSPEC('TNAME ');
37 1 0 NAME(1) = UNSPEC('RNAME ');
39 1 0 NAME(2),NAME(3) = UNSPEC('DNAME ');
41 1 0 FVALUE(1) = '00000000'B;
43 1 0 CALL FETCHT(RET_ARG);
45 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;
51 1 1 DATTR = VALUE;
53 1 1 UNSPEC(TYPE(I)) = TTYPE; BLEN(I) = TLEN;
55 1 1 BMIN(I) = TMIN; BMAX(I) = TMAX;
56 1 1 FREE DOM_RET;
57 1 0 I = I - 1
/* PRINT OUT DOMAINS */
58 1 0 STR = ';
59 1 0 DO J = 1 TO I;
60 1 1 STR = STR || DOM(J) || ' ;
61 1 1 END;
62 1 0 IF I > 8 THEN DO;
63 1 1 PUT SKIP LIST (SUBSTR(STR,1,73));
64 1 1 PUT SKIP LIST (SUBSTR(STR,73));
65 1 1 END;
66 1 0 ELSE PUT SKIP LIST (STR);

/* INITIALIZE INSERTN VARIABLES */
67 1 0 NNAME = UNSPEC(TNAME);
68 1 0 INAME = UNSPEC(DOM);

/* GET DATA */
70 1 0 DO WHILE(TRUE);
71 1 1 PUT SKIP LIST ('L:');
72 1 1 CALL LEX2(NDATAL);
73 1 1 IF N = 0 THEN RETURN;
74 1 1 IF N = I THEN PUT SKIP LIST ('INCORRECT NUMBER OF DATA ITEMS.');
75 1 1 ELSE DO;
76 1 2 DO J = 1 TO I;
77 1 3 DO WHILE(L(J) > BLEN(J));
78 1 4 PUT SKIP EDIT ('DATA FOR DOMAIN ',DOM(J), ' TOO LONG:') (A,A,A);
79 1 4 CALL LEX2(N2,TEMP,TL);
80 1 4 IF N2 = 0 THEN RETURN;
81 1 4 DATA(J) = TEMP(1);
82 1 4 L(J) = TL(1);
83 1 4 END;
84 1 3 IF TYPE(J) = 'N' THEN DO;
85 1 4 DO WHILE(VERIFY(DATA(J),'-0123456789 ') ^= 0);LOA1430
86 1 5 PUT SKIP EDIT ('DATA FOR DOMAIN ',DOM(J), ' MUST BE NUMERIC!')((3)A);LOA1450
87 1 5 CALL LEX2(N2,TEMP,TL);
88 1 5 IF N2 = 0 THEN RETURN;
89 1 5 DATA(J) = TEMP(1);
90 1 5 END;
91 1 4 DO WHILE(BIN(DATA(J)) < BMIN(J));
92 1 5 PUT SKIP EDIT ('DATA FOR DOMAIN ',DOM(J), ' BELOW MIN: ') (A,A,A);
93 1 5 CALL LEX2(N2,TEMP,TL);
94 1 5 IF N2 = 0 THEN RETURN;
95 1 5 DATA(J) = TEMP(1);
DO WHILE(BIN(DATA(J)) > BMAX(J));
    PUT SKIP EDIT ('DATA FOR DOMAIN ', DOM(J), ' ABOVE MAX:') (A,A,A);
    CALL LEX2(N2, TEMPTL);
    IF N2 = 0 THEN RETURN;
    DATA(J) = TEMP(1);
END;

/* FINALLY, CALL INSERTN */
IVALUE = UNSPEC(DATA);
CALL INSERTN(INSERT_ARG);
END;
%INCLUDE SELECT;***********************SEL00010
******************************************************************************
**
* MODULE DESCRIPTION
**
******************************************************************************

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

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

/* MISCELLANEOUS */
DCL TNAME(20) CHAR(8),
VALUE(20) CHAR(40) EXTERNAL,
ANS CHAR(1),
GIVING FIXED BIN(15) EXTERNAL,
(J1,J1) FIXED BIN(15),
FLAG BIT(1);

/* START SUBROUTINE */

/* VALUE RELATION NAME */

DO I = 1 TO R_IND;
IF TNAME(2) = REL(I) THEN LEAVE;
END;

IF I = R_IND + 1 THEN DO;
PUT LIST ('RELATION NOT FOUND.');
RETURN;
END;

/* COPY OLD RELATION ONTO NEW RELATION */

T1_ARG(R_IND+1) = T1_ARG(I);

/* DO SELECTION */

DO J = 4 TO GIVING - 1;
DO J1 = 1 TO NOOM(I);
IF UNSPEC(TNAME(J)) = C2(I,J1) & N3(I,J1) = '10000000'BSELO0450
THEN DO:
    FLAG * 'O'B;
    IF C3(I,J1) = '01010101'BSELO0490
    THEN C3(R_IND+1,J1) = UNSPEC(VALUE(J));
    ELSE DO:
        PUT SKIP EDIT('DOMAIN ',TNAME(J),
        ' ALREADY SELECTED ON.')((3)A);SELO0510
        ANS * 'Y';SELO0530
        DO WHILE(ANS = 'Y');SELO0540
        PUT SKIP LIST ('IGNORE(Y OR N)');SELO0550
        GET EDIT (ANS) (A(1));SELO0560
        IF ANS = 'Y' | ANS = 'N' THEN DO:
            DO;
            ANS = 'Y';SELO0580
            DO WHILE(ANS = 'Y');SELO0590
            PUT SKIP LIST ('NO NEW RELATION CREATED.'):SELO0600
            RETURN;
        END;
        END;
        GO TO NEXT;
    END;
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(I));
      IF ANS ^= Y | ANS ^= N THEN RETURN;
   END;
   END;
END;

/* CREATE NEW RELATION */
DO J = 1 TO R_IND;
   IF TNAME(GIVING+I) = REL(J) THEN LEAVE;
END;
IF J = R_IND + 1
THEN DO;
   R_IND = R_IND + 1;
   .DOM(R_IND) = .DOM(I);
   REL(R_IND) = TNAME(GIVING+I);
END;
ELSE T1_ARG(J) = T1_ARG(R_IND+1);
END SELECT;
%INCLUDE PROJECT: ****************************************** PRO00010
/*------------------------------------------------------*/

*                       MODULE DESCRIPTION               *
*------------------------------------------------------*/

PROJECT: PROCEDURE(TNAME /*(20) CHAR(B) */):
/****                                                      *
**** PURPOSE: PROJECT A RELATION onto GIVEN DOMAINS.     *
**** METHOD: REFER TO ???? FOR PROJECT METHOD            *
****                                                      *
**** INPUT PARAMETERS:                                   *
**** COMMAND                                             *
****                                                      *
**** OUTPUT PARAMETERS:                                  *
**** NONE                                                *
****                                                      *
**** CALLS PROCEDURES:                                  *
**** NONE                                                *
****                                                      */

/* 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(B) EXTERNAL,
NDOM(0:20) FIXED BIN(15) EXTERNAL:
/* MISCELLANEOUS */
DCL TNAME(20) CHAR(B),

/* PROJECT: PROCEDURE(TNAME /*(20) CHAR(B) */):
/****                                                      *
**** PURPOSE: PROJECT A RELATION onto GIVEN DOMAINS.     *
**** METHOD: REFER TO ???? FOR PROJECT METHOD            *
****                                                      *
**** INPUT PARAMETERS:                                   *
**** COMMAND                                             *
****                                                      *
**** OUTPUT PARAMETERS:                                  *
**** NONE                                                *
****                                                      *
**** CALLS PROCEDURES:                                  *
**** NONE                                                *
****                                                      */
ANS CHAR(1),
GIVING FIXED BIN(15) EXTERNAL,
(I,J,J1) FIXED BIN(15),
FLAG BIT(1):

/* START SUBROUTINE */

/* VALIDATE RELATION NAME */
4 1 1
DO I = 1 TO R_IND;
5 1 1
IF TNAME(2) = REL(I) THEN LEAVE;
6 1 1
END;
7 1 1
IF I = R_IND + 1
THEN DO:
8 1 1
PUT SKIP LIST ('RELATION NOT FOUND.');
9 1 1
RETURN;
10 1 1
END;

/* COPY OLD RELATION ONTO NEW RELATION */
11 1 1
T1_ARG(R_IND+1) = T1_ARG(I);
12 1 1
N3(R_IND+1,*') = '00000000'B;

/* DO PROJECTION */
13 1 1
DO J = 4 TO GIVING - 1;
14 1 1
FLAG = '1'B;
15 1 1
DO J1 = 1 TO NOOM(I);
16 1 2
IF UNSPEC(TNAME(J)) = C2(I,J1) & N3(I,J1) = '10000000'B
THEN DO:
17 1 3
N3(R_IND+1,J1) = '10000000'B;
18 1 3
FLAG = '0'B;
19 1 3
END;
20 1 2
END;
21 1 1
IF FLAG
THEN DO:
22 1 2
PUT SKIP EDIT ('DOMAIN ',TNAME(J),' NOT IN RELATION.'),(A,A,A);
23 1 2
ANS = ' ';
24 1 2
DO WHILE(ANS = 'Y');
25 1 3
PUT SKIP LIST ('IGNORE(Y OR N):');
26 1 3
GET EDIT (ANS) (A(I));
27 1 3
IF ANS = ' ' THEN RETURN;
28 1 3
END;
29 1 2
END;
30 1 1
END;

/* CREATE NEW RELATION */
31 1 0
DO J = 1 TO R_IND;
32 1 1
IF TNAME(GIVING+1) = REL(J) THEN LEAVE;
33 1 1
END;
34 1 0
IF J = P_IND + 1
THEN DO;
    R_IND = R_IND + 1;
    NDOM(R_IND) = NDOM(I);
    REL(R_IND) = TNAME(GIVING+1);
    END;
ELSE
    T1_ARG(J) = T1_ARG(R_IND+1);
END PROJECT;
%INCLUDE JOIN: ************************************************* JOIN00010
   ="/**
    *   
    *   
    *   
    *   
    *   
    *   
    *   
    *   
    *   
    *   
    */
    **
    *   
    *   
    *   
    */
    **
    *   
    *   
    *   
    *   
    */
    **
    *   
    *   
    *   
    *   
    *   
    *   
    */
    **
    *   
    *   
    */
    **

JOIN: PROCEDURE(TNAME /* (20) CHAR(8) */);
    */
    **
    *   
    *   
    *   
    *   
    *   
    */
    **
    *   
    *   
    *   
    *   
    */
    **

PURPOSE:
    JOIN TWO GIVEN RELATIONS AND A NUMBER OF GIVEN
    DOMAINS.

METHOD:
    REFER TO NJOIN1 FOR METHOD

INPUT PARAMETERS:
    NONE

OUTPUT PARAMETERS:
    NONE

CALLS PROCEDURES:
    NONE

*/

*/ 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 */
DCL TNAME(20) CHAR(8),
TEMP BiT(64),
BUFF BiT(8),
TRUE Bit(I): INIT('B'),
GIVING FIXED BiN(15) EXTERNAL,
(I,J,K,L,P1,P2) FIXED BiN(15),
(A,B,C) FIXED BiN(8);

/* START SUBR I T I N E */

/* VALIDATE RELATION 1 */
DO WHILE(TRUE);
DO I = 1 TO R IND;
IF TNAME(2) = REL(I) THEN LEAVE;
END;

/* VALIDATE RELATION 2 */
DO WHILE(TRUE);
DO I = 1 TO R IND;
IF TNAME(4) = REL(I) THEN LEAVE;
END;

/* COPY RELATION 1 INTO NEW RELATION */
T1 ARG(R IND+1) = T1 ARG(P1);

/* CORRECT NUMN */
A = N1(P1); B = N1(P2); C = A + B;
N1(R IND+1) = C;

/* ADD N_INDEXES, ATTRIBUTE NAMES, AND RET_INFO */
DO I = 1 TO NDOM(P2);
B = N2(P2,I); C = A + B;
I N2(R.IND+I,1+NDOM(P1)) = C;
C2(R.IND+I,1+NDOM(P1)) = C2(P2,1);
T3(R.IND+I,1+NDOM(P1)) = T3(P2,1);

/* FIX UP SAME */
IF N4(P2,1) = '00000000'B
THEN DO;
  B = N4(P2,1);
  C = 16*A + B;
  N4(R.IND+1,1+NDOM(P1)) = C;
END;

/* JOIN THE 2 RELATIONS */
DO I = 6 TO GIVING - 1;
  TEMP = UNSPEC(TNAME(I));
  DO J = NDOM(P1)+1 TO NDOM(P1)+NDOM(P2);
    IF TEMP = C2(R.IND+1,J) & N3(R.IND+1,J) = '10000000'B
    THEN DO;
      BUFF = '00000000'B;
      DO K = 1 TO NDOM(P1);
        IF N2(R.IND+1,K) = BUFF
          THEN L = L + 1;
        ELSE L = L;
      END;
      BUFF = N2(R.IND+1,K);
      END;
    THEN DO;
      B = N2(R.IND+1,K);
      C = 16*B + L;
      N4(R.IND+1,J) = C;
      GO TO NEXT;
    END;
END;

PUT SKIP EDIT ('DOMAIN ',TNAME(I),
' NOT IN RELATION 1.') (A.A,A); RETURN;

END;

END;

END;

/* CREATE NEW RELATION */
DO I = 1 TO R.IND;
  IF TNAME(GIVING+1) = REL(I) THEN LEAVE;

RETURN;
END:  
IF I = R_IND + 1 THEN DO:  
R_IND = R_IND + 1;  
REL(R_IND) = TNAME(GIVING+1);  
NDOM(R_IND) = NDOM(P1) + NDOM(P2);  
END:  
ELSE DO:  
T1_ARG(I) = T1_ARG(R_IND+1);  
NDOM(I) = NDOM(P1) + NDOM(P2);  
END;  
END JOIN:
MODULE DESCRIPTION

PROCEDURE (TNAME /* CHAR(8) */);

PURPOSE:

PRINT ALL THE TUPLES OF A GIVEN RELATION.

METHOD:

NOT SIGNIFICANT

INPUT PARAMETERS:

1) TNAME - NAME OF RELATION

OUTPUT PARAMETERS:

NONE

CALLS PROCEDURES:

FETCHT

INPUT PARAMETERS:

1) TNAME - NAME OF RELATION

OUTPUT PARAMETERS:

NONE
3 1 0  /* FETCH OUTPUT */
  DCL 1 DOM_RET CONTROLLED EXTERNAL, /* STACK OF DATA VALUES */
  2 ID FIXED BIN(15), /* NSET AND ATTRIBUTE ID */
  2 VALUE BIT(320); /* DATA VALUE */

4 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;

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

6 1 0  /* SUBROUTINES */
  %INCLUDE EFETCH;***********/
  /* FETCH NSET MODULE */
  DCL FEICHT 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));

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

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

9 1 1
10 1 1 GET EDIT (TNAME) (A(B));
11 1 1 IF TNAME = (B)' THEN RETURN;
12 1 1 END;

13 1 0 DO I = 1 TO R_IND;
14 1 1 IF TNAME = REL(I) THEN LEAVE;
15 1 1 END;
16 1 0 IF I = R_IND + 1 THEN DO:
17 1 1 PU' SKIP LIST ('RELATION NOT FOUND.');
18 1 1 RETURN;
19 1 1 END;

20 1 0 NUMN = '00000001'B; NSET(1) = UNSPEC('DTABLE ');
22 1 0 N_INDEX(1).N_INDEX(2) = '00000001'B;
23 1 0 NAME(1) = UNSPEC('DNAME '); NAME(2) = UNSPEC('DATTR ');
25 1 0 FETCH(1) = '00000000'B; FETCH(2) = '10000000'B;
27 1 0 SAME(1).SAME(2) = '00000000'B; RET_ARG.VALUE(2) = '01010101'B;
29 1 0 STR = ''; K = 0;
31 1 0 DO J = 1 TO NDOM(I);
32 1 1 IF N3(I,J) = '10000000'B & N4(I,J) = '00000000'B THEN DO:
33 1 2 UNSPEC(TMP) = C2(I,J);
34 1 2 STR = STR || TMP || ': ';
35 1 2 RET_ARG.VALUE(1) = C2(I,J);
36 1 2 CALL FETCH(RET_ARG);
37 1 2 DATR = DOM_RET.VALUE;
38 1 2 K = K + 1;
39 1 2 LEN(K) = TLEN;
40 1 2 FREE DOM_RET;
41 1 2 END;
42 1 1 END;

43 1 0 SUM,L = 1;
44 1 0 COUNT = 0;
45 1 0 DO J = 1 TO K;
46 1 1 SUM = SUM + LEN(J) + 1;
47 1 1 COUNT(L) = COUNT(L) + 1;
48 1 1 IF SUM > 80 THEN DO:
49 1 2 COUNT(L) = COUNT(L) - 1;
50 1 2 L = L + 1;
51 1 2 COUNT(L) = 1;
52 1 2 SUM = LEN(J) + 2;
53 1 2 END;
54 1 1 END;
 PRIO0960
 PRIO0970
 PRIO0980
 PRIO0990
 PRIO1000
 PRIO1010
 PRIO1020
 PRIO1030
 PRIO1040
 PRIO1050
 PRIO1060
 PRIO1070
 PRIO1080
 PRIO1090
 PRIO1100
 PRIO1110
 PRIO1120
 PRIO1130
 PRIO1140
 PRIO1150
 PRIO1160
 PRIO1170
 PRIO1180
 PRIO1190
 PRIO1200
 PRIO1210
 PRIO1220
 PRIO1230
 PRIO1240
 PRIO1250

55 1 0 /* PRINT OUT DOMAIN HEADER */
 IF K > 8 THEN DO:
  PUT SKIP LIST (SUBSTR(STR,1,73));
  PUT SKIP LIST (SUBSTR(STR,73));
 END;
 ELSE PUT SKIP LIST (STR);

56 1 1 /* GET THE DATA */
 IF K = 0 THEN PRINT OUT THE DATA */
 RET_ARG = T1_ARG(I);
 CALL FETCHT(RET_ARG);

57 1 0 /* PRINT OUT THE DATA */
 DO J = 1 TO ALLOCATION(DOM_RET)/K;
  M = 0;
  DO J1 = 1 TO L;
    STR = '':
    DO J2 = 1 TO COUNT(L);
      M = M + 1;
      UNSPEC(DATA) = DOM_RET.VALUE;
      STR = STR || SUBSTR(DATA,1,LEN(M)) || '';
    END;
    FREE DOM_RET;
  END;
  PUT SKIP LIST (STR);
 END;

58 1 0 END;

59 1 0 END PRINT;
%INCLUDE LEX; ****************************
/
*
*
* MODULE DESCRIPTION *
*
*
1 0 LEX: PROCEDURE(* FIXED BIN(15) */,
  TOKEN /* (20) CHAR(B) */);
/
***** PURPOSE: LEXICAL ANALYZER FOR USER COMMAND LINES.
*****
***** METHOD: NONE
*****
***** INPUT PARAMETERS: NONE
*****
***** OUTPUT PARAMETERS:
***** 1) I - NUMBER OF TOKENS FOUND
***** 2) TOKEN - TOKEN ARRAY
*****
***** CALLS SUBCLASSES: NONE
*****
*****
/
*/ LEXICAL ANALYZER VARIABLES */
DCL TOKEN(20) CHAR(B),
  LVALUE(20) CHAR(40) EXTERNAL,
  I FIXED BIN(15),
  (AND,ON,GIVING,LAST) FIXED BIN(15) EXTERNAL;
/
*/ INPUT STRING VARIABLES */
DCL STR CHAR(240) VARYING,
  LINE CHAR(80),
  POS FIXED BIN(15);
/
*/ MISCELLANEOUS */
DCL TEMP CHAR(51) VARYING,
  (START,END,EQ) FIXED BIN(15);
START SUBROUTINE */

/* GET INPUT STRING */
5 1 0 LOOP: GET EDIT (LINE) (A(80));
6 1 0 POS = INDEX(LINE,'/');
7 1 0 IF POS = 0 THEN DD;
8 1 0
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 = (8M'; 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;
25 1 1 ELSE DO;
26 1 2 TOKEN(I) = SUBSTR(TEMP,1,EQ-1);
27 1 2 IF SUBSTR(TEMP,EQ+1,1) = ' ';
28 1 3 THEN DO;
29 1 4 STR = SUBSTR(STR,EQ+2);
30 1 3 END = INDEX(STR,','*);
31 1 3 LVALUE(I) = SUBSTR(STR,1,END-1);
32 1 2 END;
33 1 2 ELSE LVALUE(I) = SUBSTR(TEMP,EQ+1);
34 1 2 END;
35 1 2 SELECT(TOKEN(I));
36 1 2 WHEN('AND ') AND = I;
37 1 2 WHEN('ON ') ON = I;
38 1 2 WHEN('GIVING ') GIVING = I;
39 1 2 OTHERWISE;
40 1 2 END;
41 1 1 STR = SUBSTR(STR,END+1);
42 1 1 START = VERIFY(STR, '');
43 1 0 END;
44 1 0 LAST = I.

LOOKING FOR SUBSTR(STR,1,END-1);
%INCLUDE LEX2:
/**
 * MODULE DESCRIPTION
 */
PROCEDURE 1 /
** 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
 */
/**
 */
PROCEDURE 2 /
** FIXED BIN(15) */,
DCL DATA(10) CHAR(40),
L(10) FIXED BIN(15),
(I,LAST) FIXED BIN(15);
/**
 * INPUT STRING VARIABLES */
DCL STR CHAR(240) VARYING,
LINE CHAR(80),
POS FIXED BIN(15);
/**
 * START SUBROUTINE */
/**
 * GET INPUT STRING */
LOOP:
    GET EDIT (LINE) (A(80));
    POS = INDEX(LINE, '/');
    IF POS ^= 0 THEN DO;
    END;
    STR = STR || SUBSTR(LINE, 1, POS);   
    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;
    DATA(I) = SUBSTR(STR, 1, LAST); 
    L(I) = LAST;
    STR = SUBSTR(STR, LAST + 2);
END;
END LEX2;
DELIM: PROCEDURE (TYPE /* BIT(1) */ )
RETURNS(BIT(1));

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.

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.

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) EXTERNAL;

OUTPUT PARAMETERS:

RETURNS A '1','0' VALUE TO INDICATE IF ANY ERRORS WERE DISCOVERED. '0'B INDICATES NO ERRORS WERE DETECTED.

CALLS PROCEDURES:

NONE

RELATIONAL OPERATOR INDEXES */

DCL (AND,ON,GIVING,LAST) FIXED BIN(15) EXTERNAL;

MISCELLANEOUS */
DCL TYPE BIT(1);

/* START SUBROUTINE */

/* CHECK MISSING DELIMITERS */
IF TYPE = '1'B
THEN IF AND = 0
THEN DO;
   PUT SKIP LIST('AND KEYWORD MISSING.');
   RETURN('1'B);
   END;
IF ON = 0
THEN DO:
   PUT SKIP LIST('ON KEYWORD MISSING.');
   RETURN('1'B);
   END;
IF GIVING = 0
THEN DO:
   PUT SKIP LIST('GIVING KEYWORD MISSING.');
   RETURN('1'B);
   END;

/* CHECK MISSING RELATION AND DOMAIN NAMES */
IF TYPE = '1'B
THEN DO:
   IF AND = 2
THEN DO:
      PUT SKIP LIST('FIRST RELATION NAME MISSING.');
      RETURN('1'B);
      END;
   IF AND > 3
THEN DO:
      PUT SKIP LIST('TOO MANY NAMES FOR FIRST RELATION.');
      RETURN('1'B);
      END;
   IF ON - AND < 2
THEN DO:
      PUT SKIP LIST('SECOND RELATION MISSING.');
      RETURN('1'B);
      END;
   IF ON - AND > 2
THEN DO:
      PUT SKIP LIST('TOO MANY NAMES FOR SECOND RELATION.');
   END;
ELSE DO:
   IF LN = 2
THEN DO:
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 THEN DO;                                    DEL00590
40 1 2  PUT SKIP LIST('TOO MANY NAMES FOR FIRST RELATION.');  DEL00600
41 1 2  RETURN('1'B);                                            DEL00610
42 1 2  END;                                                   DEL00620
43 1 1  EN';                                                   DEL00630
44 1 0  IF GIVING - ON < 2 THEN DO;                            DEL00640
45 1 1  PUT SKIP LIST('DOMAINS ARE MISSING.');                 DEL00650
46 1 1  RETURN('1'B);                                            DEL00660
47 1 1  END;                                                   DEL00670
48 1 0  IF LAST - GIVING < 1 THEN DO;                           DEL00680
49 1 1  PUT SKIP LIST('NEW RELATION NAME MISSING.');           DEL00690
50 1 1  RETURN('1'B);                                            DEL00700
51 1 1  END;                                                   DEL00710
52 1 0  IF LAST - GIVING > 1 THEN DO;                           DEL00720
53 1 1  PUT SKIP LIST('TOO MANY NAMES FOR NEW RELATION.');     DEL00730
54 1 1  RETURN('1'B);                                            DEL00740
55 1 1  END;                                                   DEL00750
56 1 0  /* HERE, NO SYNTAX ERROR WAS FOUND */                   DEL00760
57 1 0  RETURN('0'B);                                            DEL00770
58 1 0  END DELIM;                                              DEL00780
59 1 0  END;
%INCLUDE DEFINEN: \********************************************DEFO0010
/*****************************/ DEFO0010

MODULE DESCRIPTION

DEFO0020

DEFO0030

DEFO0040

DEFO0050

DEFO0060

DEFO0070

DEFO0080

DEFO0090

DEFO0100

DEFO0110

DEFO0120

DEFO0130

DEFO0140

DEFO0150

DEFO0160

DEFO0170

DEFO0180

DEFO0190

DEFO0200

DEFO0210

DEFO0220

DEFO0230

DEFO0240

DEFO0250

DEFO0260

DEFO0270

DEFO0280

DEFO0290

DEFO0300

DEFO0310

DEFO0320

DEFO0330

DEFO0340

DEFO0350

DEFO0360

DEFO0370

DEFO0380

DEFO0390

DEFO0400

DEFO0410

DEFO0420

DEFO0430

DEFO0440

DEFO0450

DEFINEN:: PROCEDURE

(DEF_ARG /* 1,
  2 BIT(64),
  2 BIT(8),
  2 (20),
  3 BIT(64),
  3 BIT(8) */);

PURPOSE:

THIS PROCEDURE CREATES AN ENTRY IN THE NSET CATALOGUE
CORRESPONDING TO THE NSET TO BE DEFINED. IN ADDITION IT
IS RESPONSIBLE FOR DEFINING THE PSET WHICH ACTS AS THE
ENTITY NODE OF THE NSET. IT IS ALSO RESPONSIBLE FOR
DEFINING THE BINARY ASSOCIATIONS (I.E. BSETS) BETWEEN
THE ENTITY NODE AND ITS ATTRIBUTES.

METHOD:

THE PROCEDURE BEGINS BY FILLING THE NCAT STRUCTURE WITH
THE CORRESPONDING ARGUMENTS PASSED TO IT BY DEF_ARG. THIS
INCLUDES: THE NSET NAME, THE NUMBER OF ATTRIBUTES,
THE NAME OF EACH ATTRIBUTE, AND WHETHER THE ATTRIBUTE
HAS A 1 TO 1 RELATIONSHIP TO THE ENTITY NODE. IT THEN
CALLS DEFINEP TO DEFINE A PSET WHICH CORRESPONDING TO
THE ENTITY NODE. THE ENTITY NODE IS GIVEN THE NAME
OF THE NSET, IS LINKED VIA HASHING, AND HAS A 32 BIT
KEY. THE NEXT STEP IS TO CREATE BSET DEFINITIONS BETWEEN
THE ENTITY NODE AND ITS ATTRIBUTES. FOR EACH ATTRIBUTE
IT DETERMINES IF A 1 TO 1 OR A N TO 1 LINK IS TO BE
DEFINED BETWEEN THE ENTITY NODE AND ATTRIBUTE. IT THEN
CALLS NAMEGEN TO CREATE A NAME FOR THE BSET, AND THEN
CALLS DEFINEU TO CREATE THE BSET DEFINITION. IN ADDITION,
IT CALLS DEFINEU USING THE EQUIVALENT OPTION TO DEFINE
THE RECIPROCAL LINK (I.E. BETWEEN THE ATTRIBUTE AND THE
ENTITY NODE). ONCE THIS IS DONE IT UPDATES NCAT TO REFLECT
THE BSETS CREATED. FINALLY IT USES THE INFORMATION IN
NCAT TO FILL INSERT_ARG, AND CALLS INSERT, PASSING IT
INSERT_ARG, TO CREATE AN ENTRY IN THE NSET CATALOGUE,
CORRESPONDING TO THE NSET DEFINITION CREATED BY DEFINEN.

INPUT PARAMETERS:

-
DEFARG CONTAINS INFO ON NSET TO BE DEFINED

2 NAME NAME OF NSET TO BE CREATED
2 NATR NUMBER OF ATTRIBUTES
2 ATTR(20) UP TO 20 ATTRIBUTES MAY BE IN AN NSET
3 ANAME NAME OF ATTRIBUTE, CORRESPONDING TO
PREVIOUSLY DEFINED PSET.
3 K_TYPE '00000001'B IF UNIQUE, '00000000'B OTHERWISE.

OUTPUT PARAMETERS:
NONE RETURNED.

PROCEDURES INVOKED:
DEFINEB.DEFINEP, NAMEGENINSERTN

%INCLUDE NCAT;*******************

%INCLUDE DEFARG;*******************

%INCLUDE INSERTA;*******************
**USED TO CREATE ATTRIBUTE STRING FOR INSERT */

5 1 0
DCL 1 ATTR_TEMP DEFINED ATTR_STR,
   2 NAME BIT(64),
   (2 K_TYPE,
   2 BREL ) BIT(8),
   (2 BSETUP,
   2 BSETDOWN ) BIT(64),
   AT' R_STR BIT(203);

/*
** SET LINK TYPES */

6 1 0
DCL A1_TO_1 BIT(6) INIT('00000001'B),
   A1_TO_N BIT(8) INIT('00000010'B),
   M_TO_N BIT(8) INIT('00000100'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);

/* PROCEEDURES CALLED */

9 1 0
DCL DEFINE_ENTRY (BIT(64),BIT(8),BIT(8),BIT(8),BIT(8),
                   BIT(8),BIT(8),BIT(8));

/* DEFINE PSET MODULE */

10 1 0
DCL DEFINED_ENTRY (BIT(64),BIT(64),BIT(64),BIT(8),BIT(1));

/* INSERT NSET MODULE */

11 1 0
DCL INSERTN_ENTRY (1, 2 BIT(64), 2 BIT(8), 2 (20),
                   3 BIT(64), 3 BIT(32));

/* 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;
NCAT.NATTR=DEF_ARG.NATTR;

DO K=1 TO NCAT.NATTR;
   NCAT.ANAMIE(K)=OEFARG.ANAME(K);
   NCAT.K_TYPE(K)=DEF_ARG(K).K_TYPE;
END;

/* CREATE ENTITY NODE PSET DEFINITION */
CALL DEFINEP(NCAT.NNAME,'00000001'B,'00000001'B,'0010000'B,'0'B,'0'B,'0'B,'1D');</

/* SET UP BSET DEFINITIONS FOR EACH ATTRIBUTE – ENTITY LINK */
DO J=1 TO NCAT.NATTR;
   IF NCAT.K_TYPE(J)=UNIQUE THEN DO:
      NCAT.BREL(J)=A1_TO_1;
      RECIP_BREL=A1_TO_1;
   END;
   ELSE DO:
      NCAT.BREL(J)=N_TO_1;
      RECIP_BREL=N_TO_N;
   END;
/* GENERATE NAME FOR BSETDOWN */
NCAT(J).BSETDOWN = NAMEGEN(B);

TEMP = NCAT(J).BSETDOWN;

/* DEFINE BSETDOWN */
CALL DEFINEBIT(TEMP,NCAT.NNAME,NCAT.ATTR(J).ANAME,
               NCAT.BREL(J),'0'B);

/* DEFINE BSETUP AS RECIPROCAL OF BSETDOWN */
CALL DEFINEB(TEMP,NCAT.ATTR(J).ANAME,NCAT.NNAME,
              RECIP_BREL,'1'B);
NCAT(J).BSETUP = TEMP;

/* SET UP INSERT_ARG FOR CALL TO INSERT NCAT INTO NSET_CAT */
INSERT_ARG.NNAME = NSETCAT;

L=1+NCAT.NATTR;
INSERT_ARG.NATTR=BIN(L);
INSERT_ARG.NAME(N)=N_ATTRIB;
INSERT_ARG.VALUE(N)=NCAT.NNAME;

/* CREATE BIT STRING REP OF ATTRIBUTE DESCRIPTION */
DO J=1 TO NCAT.NATTR;
   INSERT_ARG(J+1).NAME=N_ATTRIB;
   ATTR_TEMP=NCAT.ATTR(J);
   INSERT_ARG(J+1).VALUE=ATTR_STR;
44 1 1 END;

        /* INSERT INTO NSET_CAT */
45 1 0 CALL INSERTN(INSERT_ARG);
46 1 0 RETURN;
47 1 0 END DEFINEN;
%INCLUDE INSERTN: PROCEDURE

MODULE DESCRIPTION

INSERT: PROCEDURE (INSERT_ARG /* 1,
2 BIT(64),
2 BIT(8),
2 (20),
3 BIT(64),
3 BIT(320) */);

 PURPOSE:

 THIS MODULE IS RESPONSIBLE FOR INSERTING A TUPLE INTO A PREVIOUSLY DEFINED NSET. IT RELIES ON TWO SOURCES OF INFO REGARDING THE NSET, THE NSET DEFINITION CONTAINED IN THE NSET CATALOGUE AND THE INFORMATION CONTAINED IN INSERT_ARG. IT IS ALSO RESPONSIBLE FOR MAINTAINING THE INTEGRITY OF THE DATABASE (I.E. IT WILL NOT INSERT A TUPLE IF THERE IS ALREADY AN EXISTING OCCURRENCE OF A SUPPOSEDLY UNIQUE KEY WITHIN THE NSET.

 METHOD:

 THE PROCEDURE BEGINS BY CALLING BUILDQ WHICH FETCHES THE NSET CATALOGUE ENTRY FOR THE NSET SPECIFIED IN INSERT_ARG. IT THEN GOES THROUGH NCAT TO SEE IF ANY ATTRIBUTES ARE DEFINED TO BE UNIQUE. IF SO IT GETS THE VALUE TO BE INSERTED FROM INSERT_ARG AND CALLS SELECTF WHICH CHECKS TO SEE IF THE VALUE EXISTS ALREADY IN THE NSET. IF SO, INSERTN PRINTS AN ERROR MESSAGE AND RETURNS. OTHERWISE, IT CONTINUES TO CHECK ANY OTHER ATTRIBUTES WHICH ARE DEFINED TO BE UNIQUE. ONCE THE VALIDITY OF THE INSERT HAS BEEN DETERMINED, A UNIQUE TAG VALUE IS GENERATED FOR THE NSET NODE AND CREATEP IS CALLED TO CREATE THE ENTITY NODE. IT THEN GOES THROUGH INSERT_ARG AND CALLS CREATEP PASSING IT THE SET NAME ASSOCIATED WITH THE ATTRIBUTE AND THE ENTITY Node, A POINTER TO THE NEWLY CREATED ENTITY NODE, AND THE VALUE TO BE INSERTED. CREATEP IS RESPONSIBLE FOR ESTABLISHING THE BINARY ASSOCIATION BETWEEN THE ENTITY NODE AND ATTRIBUTE, AS WELL AS CREATING THE ATTRIBUTE NODE IF NECESSARY.

 INPUT PARAMETERS:
1 INSERT_ARG CONTAINS INFORMATION FOR INSERT
2 NNAME NAME OF PREVIOUSLY DEFINED NSET.
3 NATTR NUMBER OF ATTRIBUTE-VALUES TO BE INSERTED.
4 NOTE THAT MULTIPLE VALUES OF A PARTICULAR
5 ATTRIBUTE MAY BE INSERTED.
6 AT ONE TIME.
7 3 NAME NAME OF PREVIOUSLY DEFINED PSET CORRESPONDING
8 TO NAME OF AN ATTRIBUTE IN THE NSET DEFINITION.
9 3 VALUE VALUE FOR ATTRIBUTE, UP TO 320 BITS LONG.

OUTPUT PARAMETERS:
NONE RETURNED

PROCEDURES INVOKED:
CREATEP, CREATEB, BUILDCC.

%INCLUDE NCAT;..................... /* NSET CATALOGUE ENTRY */
DCL 1 NCAT.
2 NNAME BIT(64), /* NAME OF NSET */
3 NATTR BIT(8), /* NUMBER OF ATTRIBUTES */
4 ATTR(20), /* UP TO 20 ATTRIBUTES */
5 3 ANAME BIT(64), /* ATTRIBUTE NAME */
6 (3 K_TYPE, /* UNIQUE KEY OR NOT */
7 3 BREL) BIT(8), /* TYPE OF BSET */
8 (3 BSETUP, /* BSET(NNODE->ATTR) */
9 3 BSETDOWN) BIT(64); /* BSET(N_NODE->ATTR) */

%INCLUDE INSERTARG;.............. /* USED TO INSERT INTO NSET */
DCL 1 INSERTARG.
2 NNAME BIT(64), /* NAME OF NSET */
3 NATTR BIT(8), /* NUMBER OF ATTRIBUTES */
4 ATTR(20), /* FOR EACH ATTRIBUTE */
5 3 NAME BIT(64), /* NAME OF ATTRIBUTE */
6 3 VALUE BIT(320); /* VALUE TO BE INSERTED */

%INCLUDE INFOUND;................. /* DATA STACK RETURNED BY FETCH */
DCL NCONT BIT(320) EXTERNAL CONTROLLED;

/* PROCEDURES CALLED */

%INCLUDE ECREATP;

/* CREATE ISP MODULE */

DCL CREATEP ENTRY(BIT(64), BIT(320), POINTER);

%INCLUDE ECREATP;

/* CREATE ISP MODULE */

DCL CREATEP ENTRY(BIT(64), POINTER, BIT(320), BIT(320), POINTER);

%INCLUDE ECREATP;

/* SET RETRIEVAL MODULE */

DCL SELECTF ENTRY(BIT(2), POINTER, BIT(64), BIT(320), POINTER);

%INCLUDE EUSEL;

/* SET RETRIEVAL MODULE */

DCL BUILDC 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 ENAMEGN;

/* RANDOM NAME GENERATOR */

DCL NAMEGEN ENTRY(FIXED BIN(t5)) RETURNS(BIT(64));

/* FILL NCAT WITH NSET DEFINITION */

CALL BUILDCINSERT_ARG.NNAME, NCAT);

/* CHECK FOR DUPLICATE KEY VALUES */

DO J = 1 TO NCAT.NATTR;

IF NCAT.K_TYPE(J)=UNIQUE THEN DO;

ID2 = NULL();

/* GET CORRESPONDING ENTRY IN INSERT_ARG */

DO L = 1 TO INSERT_ARG.NATTR;

IF INSERT_ARG.NAME(L)=NCAT.ANAME(J) THEN LEAVE;

END;

/* SEE IF INSTANCE ALREADY EXISTS */

CALL SELECTF('01'B, ID2, BSETUP(J), VALUE(L), ID);

K=ALLOCATION(INFONb);

IF K ^=O THEN DO;

PUT SKIP EDIT('REQUEST IGNORED, DUPLICATE ' NAMEGEN, 'KEY FOUND IN RELATION')(A);

FREE INFOND;

ECR00090

ECR00100

ECR00110

ECR00120

ECR00130

ECR00140

ECR00150

ECR00160

ECR00170

ECR00180

ECR00190

ECR00200

ECR00210

ECR00220

ECR00230

ECR00240

ECR00250

ECR00260

ECR00270

ECR00280

ECR00290

ECR00300

ECR00310

ECR00320

ECR00330

ECR00340

ECR00350

ECR00360

ECR00370

ECR00380

ECR00390

ECR00400

ECR00410
RETURN;
END;
END;

/* INSERTION ROUTINE */
/* GENERATE TAG FOR ENTITY NODE INSTANCE */
NNN=NAMEGEN(B); NSET_ID=NNN;

/* CREATE INSTANCE OF ENTITY NODE */
CALL CREATEP(NCAT,NNAME,NSET_ID,N_ID);

/- BUILD VALUE NODES */
DO K=1 TO INSERT_ARG.NATTR;

/- FIND CORRESPONDING ENTRY IN INSERT_ARG */
DO J=1 TO NCAT.NATTR ;
  IF NCAT.ANAME(J)=INSERT_ARG.NAME(K)
    THEN LEAVE;
END;

/- LINK INSTANCE OF ATTRIBUTE TO INSTANCE OF ENTITY */
CALL CREATEB(BSETDOWN(J),N_ID,NSET_ID,INSERT_ARG(K).VALUE,INSO0640 ID2);

END;
RETURN;
END INSERTN;
%INCLUDE FETCHT;**************************************************************************
//**************************************************************************

** MODULE DESCRIPTION **
**************************************************************************

10 FETCHT: PROCEDURE
(RET_ARG /* 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) */ );

**************************************************************************

****** PURPOSE:******

THIS MODULE IS RESPONSIBLE FOR INTERPRETING THE REQUEST PASSED TO IT THROUGH RET_ARG, AND RETURNING THE REQUESTED TUPLES. AS CURRENTLY SET-UP IT ASSUMES A RELATIONAL ORGANIZATION TO THE NSET TUPLES TO BE RETRIEVED. IT IS CURRENTLY SET UP TO HANDLE SELECT, PROJECT AND JOIN ON UP TO 5 NSETS OVER A TOTAL OF 20 ATTRIBUTES.

****** METHOD:******

1) IT BEGINS BY GOING THROUGH RET_ARG.NSET(1) AND FOR EACH NSET SPECIFIED IT CALLS BUILD TO FETCH THE NSET DEFINITION CONTAINED IN THE NSET CATALOGUE.BUILD CALLS BUILD TO FETCH THE INFORMATION CONTAINED IN NCAT AND FETCH MERGES THE INFORMATION CONTAINED IN NCAT AND RET_ARG INTO A STRUCTURE CALLED NCAT2. AT THE END OF THIS PROCESS, NCAT2 CONTAINS ALL THE INFORMATION NECESSARY TO PROCESS THE REQUEST.

2) THE NEXT STEP IS TO INITIALIZE A STRUCTURE CALLED NLIST WHICH IS USED TO HOLD THE VALUES FOR THE NSET NODES WHICH MEET THE SELECT RESTRICTIONS FOUND IN RET_ARG.

3) THE NEXT STEP IS A 2 PASS PHASE IN WHICH THE SELECT RESTRICTIONS ARE TAKEN INTO ACCOUNT. IN PASS 1 IT GOES THROUGH ALL THE ATTRIBUTES IN NCAT2 AND CHECKS TO SEE IF A VALUE HAS BEEN SPECIFIED AND B) WHETHER THE ATTRIBUTE UNIQUELY DEFINES A TUPLE. IF SO, IT CALLS SEARCH WHICH IF IT EXISTS RETURNS A POINTER TO THE OCCURRENCE OF THE ATTRIBUTE VALUE. IT THEN CALLS SELECT PASSING IT THE POINTER TO THE ATTRIBUTE VALUE, AND THE NAME OF THE BSET WHICH DESCRIBES THE ATTRIBUTE-ENTITY.
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 id_list and nlist(i). the 2 nd 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, fetcht 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 selectf
and the join restrictions are contained in tab. fet00750

6) the last step is to fetch the actual attribute values fet00760
which are linked to the nodes in tab and which are fet00770
requested in ret_arg (i.e. for which ret_info.fetch = fet00780
'1'). this is accomplished by going through tab a row fet00790
at a time and using selectf to fetch the attribute fet00800
associated with the entity nodes contained in tab if t fet00810
attribute value is to be fetched. the fetched values fet00820
are placed on a stack called dom_ret (ctl ext). each fet00830
entry in dom_ret contains an id which identifies the fet00840
nset and domain with which this value is to be fet00850
associated as well as the actual data item. fet00860

*********************************************************************************************************************************************
input parameters:
*****************************************************************************
fet00900
1 ret_arg fet00910
2 numn number of nsets involved (up to 5) fet00920
2 nset(s) name of each nset (must be previously defined)fet00930
2 ATTR(20) A TOTAL OF TWENTY MAY BE SPECIFIED.

3 N_INDEX INDEX TO NSET NAME IN NSET

3 NAME NAME OF ATTRIBUTE

3 RET_INFO USED TO CONTROL FETCH

4 FETCH ('1' IF YES, '0' IF NOT)

4 SAME (IF ATTRIBUTE IS THE SAME AS A PREVIOUS

ATTRIBUTE (I.E. A JOIN SITUATION) SAME

SHOULD EQUAL ( N_INDEX OF THE NSET WHICH

CONTAINS THE PREVIOUSLY SPECIFIED ATTRIBUTE),

* 1G) WHICH ATTRIBUTE IT WAS WITHIN THAT

NSET (I.E. THE THIRD ATTRIBUTE DEFINED WITHIN)

IN THE NSET DEFINITION OF THAT NSET).

OTHERWISE SAME = 0.)

4 VALUE (IF ATTRIBUTE IS TO BE RESTRICTED

ON A CERTAIN VALUE, VALUE SHOULD

EQUAL THAT VALUE, OTHERWISE IT

MUST EQUAL '01010101' WHICH

INDICATES THAT NO RESTRICT IS WANTED.

OUTPUT PARAMETERS:

1 DOM,- RET CTL EXT

2 0_ID IDENTIFIES NSET AND ATTRIBUTE FROM WHICH

THIS VALUE WAS FETCHED. ID CONVENTION IDENTICAL

TO THAT USED IN RET_ARG.SAME.

2 VALUE THE ACTUAL DATA ITEM FOUND. FIXED LENGTH OF 320.

PROcedures Invoked:

SELECTF, SEARCH, FETCH, BUILD, NRECON, NJOIN1, TABN

%INCLUDE NCAT;*************************/

DCL 1 NCAT, /* NSET CATALOGUE ENTRY */
DCL0010
2 NNAME BIT(64), /* NAME OF NSET */
DCL0020
2 NATTR BIT(0), /* NUMBER OF ATTRIBUTES */
DCL0030
2 ATTR(20), /* UP TO 20 ATTRIBUTES */
DCL0040
3 ANAME BIT(64), /* ATTRIBUTE NAME */
DCL0050
(3 K_TYPE, /* UNIQUE KEY OR NOT */
DCL0060
3 BREL BIT(8), /* TYPE OF BSET */
DCL0070
3 BSETUP, /* BSET(ATTR->N_NODE) */
DCL0080
3 BSETDOWN) BIT(64); /* BSET(N_NODE->ATTR) */
DCL0090

%INCLUDE RETARG;*************************/

/* RETRIEVAL INFORMATION */
DCL 1 RET_ARG, /* USED TO RETRIEVE NSETS */
  2 NUMN BIT(8), /* NUMBER OF NSETS */
  2 NSET(5) BIT(64), /* NAMES OF NSETS TO BE FETCHED */
  2 ARGS(20), /* INFO FOR EACH ATTRIBUTE */
  3 N_INDEX BIT(8), /* WHICH NSET IS THIS IN */
  3 NAME BIT(64), /* NAME OF ATTRIBUTE */
  3 RET_INFO, /* RETRIEVE INFORMATION */
  4 SAME ) BIT(8), /* SAME AS PREVIOUSLY DEFINED DOMAIN */
  4 VALUE BIT(160); /* VALUE TO SEARCH ON OR 
                     NONE */

***************
/* MISC DCL */
DCL NOT_GIVEN BIT(8) INIT('01010101'B),
  (IG,IG2,IGPOS) POINTER,
  N_TAG BIT(32),
  1 FIXED BIN(8),
  TEMPI CHAR(8),
  TEMP2 BIT(32);

/* STRUCTURE HOLDS NSET_CAT DEFINITIONS AND RET_INFO FOR EACH NSET */
DCL 1 NSET(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 BSETUP BIT(64),
    3 BSETDOWN BIT(64),
    3 RET_INFO,
    4 FETCH BIT(8),
    4 SAME BIT(8),
    4 VALUE BIT(160);

/* HOLDS ENTITY NODE TAGS WHICH MEET SELECT RESTRICTIONS */
DCL 1 NLIST(5) STATIC EXT,
  2 NUM FIXED BIN(15),
  2 NODE(50) BIT(32),

/* RECTANGULARIZED VERSION OF TUPLE */
  1 TAB(5) STATIC EXT,
  2 ROW_NUM FIXED BIN(15),
  2 ROW(50) BIT(32),

/* HOLDS ENTITY NODE TAGS WHICH MEET SELECT AND JOIN CRIT */
  1 TUPLE_CTL EXT.
2 D_ID FIXED BIN(15),
2 NODE BIT(32).

/* DOMAIN VALUES RETURNED TO EXTERNAL LEVEL */
1 DM_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);
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:**************************************************************************************

******************************************************************************
#include ENRECO;******************************************************************************

******************************************************************************
/* RECTANGULARIZATION MODULE */

******************************************************************************
/* ENTITY NODE RESTRICTION MODULE */

******************************************************************************
/* BUILD NSET CAT */

K=1;
DO I=1 TO RET_ARG.NUMN;
I=1+I;
/* GET NSET CAT ENTRY FOR NSET */
CALL BUILD(CAT(RET_ARG.NSET(I), NCAT);
NCAT2(I).NNAME=NCAT.NNAME;
NCAT2(I).NATTR=NCAT.NATTR;
JJ=0;
/* FOR EACH ATTRIBUTE IN NSET */
DO I=I+1 TO 20 WHILE(N_INDEX(I)=IK);
JJ=JJ+I;
/* FIND RET_ARG ENTRY FOR ATTRIBUTE */
DO J=1 TO NCAT2(I).NATTR;
IF RET_ARG.NAME(I)=NCAT.ANAME(I)
THEN LEAVE;
END;
/* ENTER REST OF INFO FOR Attribute */
NCAT2(I).A_NAME(JJ)=NCAT.A_NAME(I);
NCAT2(I).K_TYPE(I)=NCAT.K_TYPE(I);
NCAT2(I).BREL(JJ)=NCAT.BREL(I);
NCAT2(I).BSETU(JJ)=NCAT.BSETU(I);
NCAT2(I).BSETD(JJ)=NCAT.BSETD(I);
NCAT2(I).ATTR(JJ).RET_INFO=RET_ARG(I).RET_INFO;
END;
K=I;
END;
/* END OF BUILD CAT SEQUENCE */

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

DO JJ=1 TO 50;
NLST(J).NODE(JJ)=0;
END;
END;

/* RESTRICTION PHASE */
/* FOR EACH NSET IN NCAT2 */
DO J=1 TO RET_ARG.NUMN;
RE;TRCT='O';
END; 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;

/* SEARCH */
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;
END;

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

/* CALL SELECTF */
CALL SELECTF('11'B,NCAT2(J).BSETUP(K),
'0B',102);

/* 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(RESTRCT,NLIST(J),IDLIST);
END;
/* IF NO RESTRICTIONS ON NSET GET ALL ENTITY NODES */

IF RESTRICT='O'B
   THEN DO;
      ID=NULL();
      CALL FETCH('11'B.ID.NCAT2(u).NAME,'O'B,'1'B);
      NLIST(J).NUM=ALLOCATION(INFO_ND);
      DO I=1 TO NLIST(J).NUM;
         NLIST(J).NODE(I)=INFO_ND;
   END;
   FREE INFO_ND;
/* BEGIN JOIN LOGIC */

IF NUMN>1 THEN DO;
   L=NUMN;
   CALL NJOIN1 TO HANDLE JOIN LOGIC /
   NUM_NODES=NJOIN1(L);
   END;
   ELSE DO;
      DO K=1 TO NLIST.NUM(1);
      ALLOCATE TUPLE;
      TUPLE.O_1D=I;
      TUPLE.NODE=NLIST(1).NODE(K);
      END;
      END;
      END;
      END;
      END;
      END;
/* RECTANGULARIZE CONTENTS OF TUPLE FOR RELATION */

CALL TABN;
/* CONSTRUCT DOMAIN STACK */

DO I=1 TO TAB(1).ROW_NUM;
   DO II=1 TO NUMN;
      DO III=1 TO NCAT2(II).NATTR;
         CALL SELECTF('O1'B.ID.NCAT2(II).BSETDOWN(III),TAB(II).ROW(I).102);
      END;
   END;
   END;
/* IF TO BE FETCHED AND NOT FETCHED ALREADY */

IF NCAT2(II).FETCH(III)='I'B &
   NCAT2(II).SAME(III)='00000000'B
   THEN DO;
      ID=NULL();
      CALL SELECTF('O1'B.ID.NCAT2(II).BSETDOWN(III),TAB(II).ROW(I),ID2);
      END;
*/
/* PLACE ON TEMPORARY STACK */

ALLOCATE D_TEMP;
D_TEMP.D_ID=11*16+111;
D_TEMP.VALUE=INFO_ND;
FREE INFO_ND;
END;

/* REVERSE ORDER OF ELEMENTS ON STACK */

DO WHILE(ALLOCATION(D_TEMP)"=0);
ALLOCATE DOM_RET;
DOM_RET = D_TEMP;
FREE D_TEMP;
END;

END;
**%INCLUDE**

/*NJJOIN1:*********************************************************************/

MODULE DESCRIPTION

**** */

FOR00010 (L /*FIXED BIN(15)*/)

RETURNS(FIXED BIN(15))

RECURSIVE;

FOR00060

/**** PURPOSE: ****/

THIS MODULE IS RESPONSIBLE FOR HANDLING JOINS BETWEEN ONE OR MORE NSETS.

WHEN ORIGINALLY CALLED BY FETCHT IT MAKES USE OF THE INFORMATION CONTAINED IN NCAT2 TO DETERMINE WHAT DOMAINS ARE TO BE JOINED, AND IT USES THE NSET NODES CONTAINED IN NLIST AS ITS UNIVERSE OF POSSIBLE TUPLES. AS CURRENTLY IMPLEMENTED IT WILL ONLY HANDLE JOINS ON ELEMENTS HAVING EQUAL VALUES.

ALSO, IT REQUIRES THAT ALL JOINS BE EXPRESSED IN TERMS OF PREVIOUS NSETS. THAT IS, WHEN AN ATTRIBUTE IN NSET L IS DEFINED TO BE THE SAME AS A DOMAIN IN ANOTHER NSET J, J MUST BE LESS THAN L. THIS RESTRICTION DOES NOT COMPROMISE THE GENERALITY OF THE JOIN LOGIC. IT DOES, HOWEVER, RESTRICT THE MANNER IN WHICH THE JOIN IS EXPRESSED.

/**** METHOD: ****/

THIS IS A RECURSIVE PROCEDURE WHICH BUILDS TUPLES A TUPLE AT A TIME, AND CREATES A STACK OF NSET NODES WHICH CORRESPOND TO THE JOINED TUPLES. THE STRATEGY EMPLOYED IS AS FOLLOWS:

1) IT BEGINS BY CREATING A TEMPORARY COPY OF THE RELEVANT NSET NODES IN A CONTROLLED STRUCTURE CALLED NLTEMP IF NO PREVIOUS ALLOCATIONS OF NLTEMP EXIST THEN A COPY OF NLIST IS MADE. OTHERWISE, A COPY OF THE MOST RECENT ALLOCATION OF NLTEMP IS MADE.

2) IT THEN GOES THROUGH THE NODES CONTAINED IN NLTEMP(L) WHICH CORRESPONDS TO THE NSET NODES OF THE LAST NSET TO BE JOINED. FOR EACH NODE IT GOES THROUGH THE FOLLOWING LOOP:

A) IT PLACES A COPY OF THE NODE ON THE TOP OF A STACK CALLED TUPLE.

B) IT THEN GOES THROUGH ALL THE ATTRIBUTES OF THAT NSET NODE TO SEE IF ANY ATTRIBUTE IS DEFINED IN NCAT2 TO BE THE SAME AS AN ATTRIBUTE OF ANOTHER NSET. IF SO:

1) IT DETERMINES THE NSET AND DOMAIN WITHIN THAT
NSET by DECOMPOSING NCAT2.SAME FOR THE ATTRIBUTE. THIS IS REFERRED TO AS JNSET AND J_DOMAIN RESPECTIVELY.

2) IT THEN FINDS THE OCCURRENCE OF THE ATTRIBUTE WHICH IS LINKED TO THE NSET NODE ON THE TUPLE STACK. THIS IS ACCOMPLISHED VIA A CALL TO SELECTF.

3) IT THEN RETRIEVES ALL OF THE NSET NODES IN JNSET WHICH ARE ASSOCIATED WITH THAT OCCURRENCE OF J_DOMAIN. THIS IS ACCOMPLISHED THROUGH A CALL TO SELECTF, PASSING IT THE VALUE OF THE ATTRIBUTE NODE AND THE NAME OF THE BSET WHICH LINKS THE J_DOMAIN ATTRIBUTE IN JNSET TO THE ENTITY NODE IN JNSET.

4) IF THE NUMBER OF NSET NODES FOUND = 0 THEN NO JOIN IS POSSIBLE WITH THE NSET NODE CURRENTLY ON THE TOP OF THE STACK AND SO YOU POP THE STACK AND DROP OUT OF THE LOOP. OTHERWISE THE NSET NODES FOUND ARE COMPARED WITH THE NSET NODES CONTAINED IN NTEMP(J,NSET) AND A TEMPORARY STRUCTURE IS CREATED WHICH CONTAINS THE INTERSECTION OF NTEMP(J,NSET) AND THE NSET NODES FOUND. IF THE INTERSECTION IS EMPTY, NO JOIN IS POSSIBLE, THE STACK IS POPPED AND YOU DROP OUT OF THE LOOP. OTHERWISE, NTEMP(J,NSET) EQUAL TO THE TEMPORARY STRUCTURE.

5) THIS LOOP IS CONTINUED FOR ALL OF THE ATTRIBUTES ASSOCIATED WITH THE NSET NODE ON THE TOP OF THE STACK. AT THE END OF THE LOOP NTEMP WILL CONTAIN ONLY THOSE NSET NODES WHICH ARE CONSISTENT WITH JOINS WITH THE LTH NSET AND FOR THE PARTICULAR OCCURRENCE OF THE NSET NODE.

C) IT THEN CHECKS THE CURRENT VALUE OF L, IF L > 1 THEN NJOIN CALLS ITSELF, PASSING ITSELF L-1. THE EFFECT OF THIS IS TO PERFORM THE JOIN LOGIC ON THE NODES IN NTEMP FOR NSETS (L-1) TO L-1. WHEN NJOIN RETURNS, IT RETURNS THE NUMBER OF NSET NODES WHICH IT ADDED TO THE TUPLE STACK. IF NO NODES WERE ADDED THEN NO JOIN IS POSSIBLE AND THE STACK IS POPPED. OTHERWISE THE STACK CONTAINS 1 OR MORE COMPLETE TUPLES ASSOCIATED WITH THE ORIGLAL NODE ON THE TOP OF THE STACK. IT THEN PROCEDS TO THE NEXT NSET NODE ON NTEMP(L).

D) IF L = 2 THEN THE CONTENTS OF NTEMP(1) ARE PLACED ON THE TOP OF THE STACK. IF NTEMP IS EMPTY THEN NO JOIN IS POSSIBLE AND THE STACK IS POPPED. IN ANY EVENT IT THEN PROCEDS TO THE NEXT NSET NODE ON NTEMP(L)

3) AFTER GOING THROUGH ALL THE NSET NODES IN NTEMP(L) IT FREES THE CURRENT ALLOCATION OF NTEMP AND RETURNS
PL/I OPTIMIZING COMPILER

%INCLUDE NJJOIN1;

STMT LEV NT

***** 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 NLST (CREATED BY FETCHT)
***** NCA12 (ALSO CREATED BY FETCHT)

*****
***** OUTPUT PARAMETERS:
***** 1 TUPLE CTL EXT
***** 2 D_ID BIT(64) 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 OCCURRENCE
***** OF AN NSET.

*****
***** PROCEDURES INVOKED:
***** ELECTF, NJJOIN

*********************************************************************************/

/ * RETRIEVAL AND NSET ORGANIZATION INFO */
DCL 1 NCA12(64) EXTERNAL,
2 NNAME BIT(64),
2 ATTR BIT(8),
2 ATTR(20),
3 A_NAME BIT(64),
3 K_TYPE BIT(B),
3 BREL BIT(B),
3 SETUP BIT(64),
3 BSETDOWN BIT(64),
3 RETINFO,
4 FETCH BIT(B),
4 SADT BIT(B),
4 VALUE BIT(160);

/ * HOLDS ENTITY NODES THAT SATISFIED RESTRICTIONS */
DCL 1 NLST(5) STATIC EXTERNAL,
2 NUM FIXED BIN(15),
2 NAGE(50) BIT(32);

/ * 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);

/* STACK OF ENTITY NODES MEETING JOIN AND SELECT RESTRICTIONS */

5 1 0
DCL 1 TUPLE CTL EXT,
  2 NODE BIT(32);

6 1 0
DCL 1 TEMP(5),
  2 NUM FIXED BIN(15),
  2 NODE(50)BIT(32);

7 1 0
DCL 1 T_LIST,
  2 NUM FIXED BIN(15),
  2 NODE(50)BIT(32);

%INCLUDE IDSI;

8 1 0
DCL IDSI PTR EXTERNAL CONTROLLED;

%INCLUDE INFDQUD;

9 1 0
DCL INFO.NODE BIT(320) EXTERNAL CONTROLLED;

%INCLUDE ENJOIN;

10 1 0
DCL (10,ID2,IDPOS) POINTER INIT( NULL()),N_TAG CHAR(8);

11 1 0
DCL IDAK POINTER CONTROLLED;

12 1 0
DCL (ID,FND,STATUS) BIT(1) INIT('1'),
  TMP1 BIT(320);

/* PROCEDURES CALLED */

%INCLUDE ESELECT;

13 1 0
DCL NJ(INI ENTRY RETURNS(FIXED BIN(15));

%INCLUDE ESEARCH;

14 1 0
DCL SELECTF ENTRY(BIT(2),POINTER,BIT(64),BIT(320),POINTER);

/* FETCH PSET MODULE */

15 1 0
DCL FETCH ENTRY(BIT(2),POINTER,BIT(64),BIT(64),BIT(1));

%INCLUDE ESEARCH;

16 1 0
DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER);

/* IF FIRST CALL, THEN COPY NLIST INTO NLTEMP */

17 1 0
IF ALLOCATION(NLTEMP) *0
  THEN 00;
TEMP=NLIST;
ALLOCATE NLTEMP;
NLTEMP=TEMP;
END;

/* FOR EACH ELEMENT IN THE HIGHEST ORDER NLIST */
DO I=1 TO NLTEMP(L).NL;
IF ALLOCATION(NLTEM)=0 THEN DO:
ALLOCATE NLTEMP;
END;
NLTEMP=NLIST;
END;

/* PLACE ELEMENT ON TOP OF TUPLE STACK */
ALLOCATE TUPLE;
NODES_ADDED=L;
TUPLE.NODE=NLTEMP(L).NODE(I);

/* INPREPARATION FOR JOIN CHECKING GET FRESH COPY OF NLTEMP */
TLP=L;
ALLOCATE NLTEMP;
NLTEM=TEMP;

/* FOR EACH ATTRIBUTE OF THE LTH NSET */
DO II = 1 TO NCAT2(L).NATTR;
/* IF ATTRIBUTE THE SAME AS A PREVIOUS ATTRIBUTE */
IF NCAT2(L).SAME(II)="00000000'B" THEN DO:
/* CALCULATE WHICH NSET AND DOMAIN */
J_NSET=(BOOL(NCAT2(L).SAME(II),"11110000'B", "0001'B"))/16;
J_DOMAIN=BOOL(NCAT2(L).SAME(II),"0001111'B", "0001'B");
/* GET INSTANCE OF ATTRIBUTE IN THE LTH NSET WHICH IS ASSOCIATED WITH THE I TH ENTITY */
NODE *=
IDNULL();
CALL SELECTF("11'B,ID,NCAT2(L).BSETDOWN(II), NLTEM(NODE(I),102); 
TEMP=INFO_NODE;
FREE INFO_NODE;
/* GET ASSOCIATED INSTANCES OF ENTITY NODES IN THE J_NSET TH WHICH SHARE THE COMMON ATTRIBUTE VALUE */
ID=NUll();
CALL SELECTF("11'B,ID,NCAT2(J_NSET).BSETUP (J_DOMAIN),TEMP1,102);
NUM_FND = ALLOCATION(INFO_ND);

IF NUM_FND = 0

/* IF NONE FOUND, THIS ENTITY NODE CAN NOT BE JOINED, HENCE REMOVE FROM TUPLE */
THEN DO:
FREE TUPLE;
NODES_ADDED = 0;
END:
ELSE DO;

/* OTHERWISE GET INTERSECTION OF NODES FOUND AND NODES IN CURRENT COPY OF NLTEMP FOR THAT NSET, PUT UNION IN T_LIST */
T_LIST.NUM = 0;
DO J = 1 TO NUM_FND;
FND = '0' B;
DO LL = 1 TO NLTEMP(J_NSET).NUM WHILE (~FND);
IF INFO_ND = NLTEMP(J_NSET).NODE(LL)
THEN FND = '1' B;
END;

IF FND
THEN DO:
T_LIST.NUM = T_LIST.NUM + 1;
FREE INFO_ND;
END;

/* IF INTERSECTION IS EMPTY, NO JOIN */
IF T_LIST.NUM = 0
THEN DO:
FREE TUPLE;
NODES_ADDED = 0;
END;

ELSE NLTEMP(J_NSET) = T_LIST;
END;

END:

/* IF JOIN ATTEMPT UNSUCCESSFUL FOR THIS NODE GO ON TO NEXT NODE */
IF NODES_ADDED = 0
THEN LEAVE;
END;

/* OTHERWISE PERFORM JOIN LOGIC ON L-1TH NSET */
IF NODES_ADDED = 0
THEN DO:
/* IF MORE THAN 1 NSET REMAINS CALL NUJOIN */
IF L > 2
THEN ADDED = NUJOIN(L - 1);
ELSE DO;
    /* OTHERWISE NLTEMP(1) WILL CONTAIN ONLY THOSE ENTITY NODES FOR NSET I THAT SATISFY THE JOIN LOGIC, HENCE THEY ARE PLACED ON THE TUPLE STACK*/
    DO K=1 TO NLTEMP(1).NUM;
        ALLOCATE TUPLE;
        TUPLE.D_ID=1;
        TUPLE.NODE=NLTEMP(1).NODE(K);
        END;
        ADDED=NLTEMP(1).NUM;
    END;
    /* IF NONE ADDED THEN JOIN WAS NOT SATISFIED NODE SHOULD BE REMOVED FROM TUPLE STACK */
    IF ADDED=0 THEN DO:
        FREE TUPLE;
        NODES_ADDED=0;
    END;
    /* UPDATE NODE_ADDED */
    ELSE NODES_ADDED=NODES_ADDED+ADDED;
    END;
    /* FREE CURRENT ALLOCATION OF NLTEMP WHICH WAS USED FOR THIS INSTANCE OF THE ENTITY NODE */
    FREE NLTEMP;
END NJOIN1;
%INCLUDE T ABN:itchen----------------------------------------------------------------------------------------TAB00010
/**
 * MODULE DESCRIPTION
 * FORO0020
 * FORO0030
 * FORO0040
 *--------------------------------------------------------------------------------------------------------*/
0 TABN: PROCEDURE;
/*
 * PURPOSE:
 * FORO0070
 * THIS MODULE TAKES THE STACK OF ENTITY NODES CREATED BY
 * FORO0080
 * FETCHT AND CREATES A TABLE STRUCTURE WHICH REFLECTS
 * FORO0090
 * THE RELATIONAL MODEL. FOR EXAMPLE, THE STACK MAY LOOK
 * FORO100
 * AS FOLLOWS: 121212123, WHERE THE NUMBERS CORRESPOND TO THE
 * FORO110
 * NODES TO WHICH THE NODES BELONG. THAT STACK IS IMPLICITLY
 * FORO120
 * SPECIFYING THE FOLLOWING RELATIONAL STRUCTURE:
 * FORO130
 * 123
 * 123
 * 123
 * 123
 * FORO140
 * FORO150
 * FORO160
 * FORO170
 * THIS MODULE IS RESPONSIBLE FOR PERFORMING THAT CONVERSION.
 *--------------------------------------------------------------------------------------------------------*/
/* METHOD:
 * FORO180
 * SEE COMMENTS IN PROGRAM
 *--------------------------------------------------------------------------------------------------------*/
/* INPUT PARAMETERS:
 * FORO190
 * USES THE TUPLE STACK CREATED BY FETCHT:
 * FORO200
 * 1 TUPLE CTL EXT,
 * FORO210
 * 2 0_ID FIXED BIN(15),
 * FORO220
 * 2 NODE BIT(32);
 *--------------------------------------------------------------------------------------------------------*/
/* OUTPUT PARAMETERS:
 * FORO230
 * RETURNS THE FOLLOWING EXTERNAL STRUCTURE:
 * FORO240
 * 1 TAB(5) STATIC EXTERNAL,
 * FORO250
 * 2 ROW_NUM FIXED BIN(15),
 * FORO260
 * 2 ROW(50) BIT(32);
 *--------------------------------------------------------------------------------------------------------*/
/* CALLS PROCEDURES:
 * FORO270
 *--------------------------------------------------------------------------------------------------------*/
/*
 * TAB TO HOLD ENTITY NODE TAGS */
PUBLIC 1 TAB(5) STATIC EXTERNAL,
DCL 1 TAB(5) STATIC EXTERNAL,
2 ROW_NUM FIXED BIN(15) INIT ((5) 0),
2 ROW(50) BIT(32);
2 ROW(50) BIT(32).

/* STACK CONTAINING ENTITY NODE TAGS */
1 TUPLE CTL EXT,
2 D_ID FIXED BIN(15),
2 NODE BIT(32);

3 1 0
DO WHILE(ALLOCATIOT(TUPLE)EQ0);

/* ADD ENTITY NODE TAGS FOR NSET 1 TO COL 1 */
4 1 1
DO WHILE(ALLOCATIOT(TUPLE)EQ0&TUPLE.D_IDEQ1);
5 1 2
ROW_NUM(1)=ROW_NUM(1)+1;
6 1 2
TAB(1).ROW(ROW_NUM(1))=TUPLE.NODE;
7 1 2
FREE TUPLE;
8 1 2
END;
9 1 1
LAST_COL=1;

/* FILL ROWS FOR REMAINING NSETS */
10 1 1
DO WHILE(ALLOCATIOT(TUPLE)EQ0&TUPLE.D_IDEQ1):

/* FILL SO NUMBER OF ENTRIES SAME AS PREVIOUS COLUMN */
11 1 2
DO K=1 TO (ROW_NUM(LAST_COL)-ROW_NUM(TUPLE.D_ID));
12 1 3
ROW_NUM(TUPLE.D_ID)=ROW_NUM(TUPLE.D_ID)+1;
13 1 3
TAB(TUPLE.D_ID).ROW(ROW_NUM(TUPLE.D_ID))=TUPLE.NODE;
14 1 3
END;
15 1 2
LAST_COL=TUPLE.D_ID;

/* POP TOP OF TUPLE STACK */
16 1 2
FREE TUPLE;
17 1 2
END;
18 1 1
END;
19 1 0
RETURN;
20 1 0
END TBN4;
%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

/**/ MODULE DESCRIPTION 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 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

FORO0010 (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: FOR00010
***** THIS IS A SUPPORT MODULE FOR THE FETCHT 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: FOR00010
***** 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='O'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  IF ARG_NODE.NUM=0 & MODE='O'B
            THEN DO:
                 DO J=1 TO WORK_NODE(1).NUM;
                 ARG_NODE.NODES(J)=WORK_NODE(1).NODES(J);
            END;

6   1   0  ARG_NODE.NUM=WORK_NODE(1).NUM;

7   1   1  ELSE DO:

8   1   2  ARG_NODE.NUM=WORK_NODE(1).NUM;

9   1   2  END;

10  1   1  RETURN;

11  1   1  END;

12  1   1  END;

13  1   0  ELSE DO:

14  1   1  DO J=1 TO ARG_NODE.NUM;

15  1   2  FND='O'B;

16  1   2  DO J=1 TO WORK_NODE(1).NUM WHILE("FND");

17  1   3  IF ARG_NODE(J).NODES=WORK_NODE(1).NODES(J)
            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;
%INCLUDE BU100010
/***************************************************************************/

BUILDC: PROCEDURE

(NAME1, /* BIT(64) */
  NCAT /* 1, */
    2 BIT(64),
    2 BIT(8),
    2 (20),
    3 BIT(64),
    3 BIT(8),
    3 BIT(64),
    3 BIT(86);)

***************************************************************************/

**** PURPOSE:
****
**** THIS MODULE IS RESPONSIBLE FOR FETCHING THE NSET_CAT ENTRY FOR NSET NAME1, AND FOR RETURNING THE INFORMATION IN THE NCAT STRUCTURE.

**********************************************************************************************/

**** METHOD:
****
**** SEE COMMENTS IN THE CODE. ITS FAIRLY STRAIGHTFORWARD EXCEPT FOR THE USE OF ATTR_TEMP. SINCE THE ATTR DESCRIPTION FOR AN ATTRIBUTE IS STORED AS A BIT STRING, IT IS NECESSARY TO USE A STRING OVERLAY TO MAP THE CONTENTS OF THE BIT STRING TO NCAT.ATTR, AND THAT IS THE PURPOSE OF ATTR_TEMP.

**********************************************************************************************/

**** INPUT PARAMETERS:
****
**** NAME1 - NAME OF A PREVIOUSLY DEFINED NSET.

**********************************************************************************************/

**** OUTPUT PARAMETERS:
****
**** NCAT - SEE OTHER DESCRIPTIONS OF NCAT.

**********************************************************************************************/

**** CALLS PROCEDURES:
****
**** SELECTF, SEARCH

**********************************************************************************************/

%INCLUDE NCAT;***************************************************************************/

DCL 1 NCAT, /* NSET CATALOGUE ENTRY */

DCL000020

DCL000030
2 ATTR(20),  /* UP TO 20 ATTRIBUTES */ DCL00040
  3 ANAME BIT(64),  /* ATTRIBUTE NAME */ DCL00050
  (3 K_TYPE,  /* UNIQUE KEY OR NOT */ DCL00070
   3 BREL) BIT(8),  /* TYPE OF BSET */ DCL00070
  (3 BSETUP,  /* BSET(ATTR->N_NODE) */ DCL00080
   3 BSETDOWN) BIT(64);  /* BSET(N_NODE->ATTR) */ DCL00090

***************

/* OV'RLAID ON INFO_ND TO EXTRACT ATTRIBUTE DESCRIPTION */
3 1 0
DCL 1 A TO 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 INFOUND:*******************************/
/* DATA STACK RETURNED BY FETCH */
5 1 0
DCL INFO_ND BIT(320) EXTERNAL CONTROLLED;

***************

/* MISC DECLARATION */
6 1 0
DCL (IDXX.ID,I1)POINTER INIT(NULL()),
NAME 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 ESELCCF:*******************************/
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('01B,N_NAME,NAME1,IDXX,ID);

12 1 0
ID=IDXX;

13 1 0
ID=NULL();

14 1 0
FREE ID1;
/* GET THE ASSOCIATED ENTITY NODE */
CALL SELECTF('11'!B.ID1.NSETB1,'0'B.IDXX);
IDNODE=INFO_ND;
FREE INFO_ND;
/* GET THE ASSOCIATED INSTANCES OF ITS ATTRIBUTE DESCRIPTIONS*/
CALL SELECTF('11'!B.ID,NSETB2,IDNODE, ID1);
L=ALLOCATION(INFO_ND);
/* BUILD NCAT */
NCAT.NATTR= BIN(L);
NCAT.NNAME=NAME1;
DO J=1 TO NCAT.NATTR;
/* MAP ATTR_STR ONTO INFO_ND */
ATTR_STR=INFO_ND;
NCAT(J).ATTR=ATTR_TEMP;
FREE INFO_ND;
END;
RETURN;
END BUILD;
PROCEDURE NINIT:

PURPOSE: THIS MODULE IS RESPONSIBLE FOR INITIALIZING THE NSET-CAT.

METHOD:

NONE. THOUGH IT DOES CREATE VIA CALLS TO THE INTERNAL LEVEL THE PSETS AND BSETS NECESSARY TO IMPLEMENT THE NSET CAT NSET, IT DOES NOT ISSUE THE APPROPRIATE CALLS TO THE INTERNAL LEVEL TO DEFINE THE PSETS AND BSETS BY WHICH THE NSET-CAT IS IMPLEMENTED, AND THEN INSERT AN NSET-CAT ENTRY FOR THE NSET-CAT NSET INTO ITSELF.

PROCEDURES:

DEFP, CREATEP, DEFINE.
%INCLUDE EDEF-INB:..*..*..*********..*.***********.****.***********NINOl40

/* DEFINE BSET MODULE */
DCL DEFINED ENTRY(BIT(64),BIT(64),BIT(64),BIT(64),BIT(64),BIT(1));
%INCLUDE ECET:.

/* CREATE BSET MODULE */
DCL CREATES ENTRY(BIT(64).BIT(64).BIT(64),BIT(320).BIT(320),BIT(320),POINTER);
%INCLUDE ERAP~

/* START OF PROCEDURE - INITIALIZE VARIABLES */
UNIQUE-'00000001'B;
N_NAME=UNSPEC('NSETNAME');
NSETCAT=UNSPEC('NSETCAT ');
N_ATTR=UNSPEC('NSETATTR ');
NSETB1=UNSPEC('NSETB1 ');
NSETB2=UNSPEC('NSETB2 ');

/* DEFINE THE NSETNAME PSET FOR THE NSETCAT */
CALL DEFINEP(N_NAME,'00000001'B,'00000001'B,'00100000'B,'00000000101100'B,'0'B,'0'B,0PTR);

/* DEFINE THE ENTITY NODE PSET FOR THE NSET_CAT */
CALL DEFINEP(NSETCAT,'00000001'B,'00000001'B,'00100000'B,'00000000100000'B,'0'B,'0'B,0PTR);

/* DEFINE THE ATTRIBUTE DESCRIPTION PSET FOR THE NSET_CAT */
CALL DEFINEP(N_ATTR,'00000001'B,'00000001'B,'00100000'B,'00000000111000'B,'0'B,'0'B,0PTR);

/* DEFINE THE NSETNAME-NSETCAT ENTITY NODE BSET */
CALL DEFINEB(NSETB1,N_NAME,NSETCAT,'00000001'B,'0'B);

/* NOW DEFINE ITS RECIPROCAL */
TEMP=NSETB1;
CALL DEFINEB(TEMP,NSETCAT,N_NAME,'00000001'B,'1'B);

/* DEFINE THE ENTITY NODE - ATTR DESCRIPTION BSET */
CALL DEFINEB(NSETB2,NSETCAT,N_ATTR,'0000000100'B,'0'B);

/* NOW INSERT THE NSET_CAT CAT ENTRY FOR ITSELF INTO THE NSET_CAT, FIRST CREATE AN INSTANCE OF THE NSET NAME */
CALL CREATEP(N_NAME,NSETCAT,PTR);

/* CREATE AN ASSOCIATED INSTANCE OF THE ENTITY NODE */
CALL CREATEB(NSETB1,PTR,'0'B,'0'B,0PTR2);

/* CREATE PSET MODULE */
DCL CREATEP ENTRY(BIT(64),BIT(320),BIT(320),POINTER);
%INCLUDE ERAP~
/* CREATE ATTR DESCRIPTION FOR N_NAME */
NAME=N_NAME;
K_TYPE='000000001'B;
BREL='00000001'B;
BSETUP=NSETB1;
BSETDOWN = TUP;

/* CREATE AN ASSOCIATED INSTANCE OF THE ATTR DESCRIPTION */
CALL CREATEB(NSETB2,PTR2,'O'B,ATTR_STR,PTR);

/* CREATE ATTR DESCRIPTION FOR N_ATTR */
NAME=N_ATTR;
K_TYPE='000000008'B;
BREL='000000010'B;
BSETDOWN=NSETB2;

/* CREATE AN ASSOCIATED INSTANCE OF THE N_ATTRIB DESCRIPTION */
CALL CREATE3(NSETB2,PTR2,'O'B,ATTR_STR,PTR);

RETURN;
END NINIT;
%INCLUDE DEFINEV: 

/****************************************************************************

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
DEFINEP. 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 DEFINEP
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:

DEFINEP

******************************************************************************/
%INCLUDE DVARG;  
/* STRUCTURE USED TO PASS NAME OF VALUE SET TO BE DEFINED */  
2 1 0  
DCL 1 DV_ARG,  
 2 NAME BIT(64), /* NAME OF VALUE SET */  
 2 KEY_LEN BIT(8); /* LENGTH OF KEY FIELD */  
***************  
/* DEFAULTS USED TO DEFINE PSETS */  
3 1 0  
DCL 1 SYS_DEFAULT,  
 2 LINK BIT(8) INIT('00000001'B),  
 2 LEN BIT(16) INIT('0000000001000000'B),  
 2 KEY_POS BIT(8) INIT('00000000'B),  
 2 SUBSET BIT(8) INIT('00000000'B),  
 2 S_ID BIT(8) INIT('00000000'B),  
 2 ID PTR;  
***************  
/* PROCEDURE CALLED */  
%INCLUDE EDEFINP;  
***************

4 1 0  
DCL DEFINEP_ENTRY(BIT(64),BIT(8),BIT(8),BIT(8),BIT(8),BIT(8));  
***************  
/* CALL DEFINEP MODULE */  
5 1 0  
CALL DEFINEP(NAME,LINK,KEY_POS,KEY_LEN,LEN,SUBSET,S_ID,  
  ID);  
6 1 0  
RETURN;  
7 1 0  
END DEFINEP;
%include fetchv:

MODULE DESCRIPTION

PURPOSE:

METHOD:

INPUT PARAMETERS:

OUTPUT PARAMETERS:

CALLS PROCEDURES:

%include FVARG:

/* 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'; OTHERWISE '0' */

/* FVARG */

/* FETCH */
2 DATA BIT(320); /* RETRIEVED ELEMENT */

**************
/* MISC DECLARATIONS */
3 1 0 DCL ID PTR INIT(NULL()),
   FND BIT(1) INIT('0');
**%INCLUDE INFOUND;******************************************************************************************
**FETO0070
4 1 0 DCL INFO_NO BIT(320) EXTERNAL CONTROLLED;
**************
**%INCLUDE EFETCH;******************************************************************************************
**FETO0080
5 1 0 DCL FETCH ENTRY(BIT(2), POINTER, BIT(64), BIT(64), BIT(1));
**************
/* CALL FETCH TO RETRIEVE ELEMENT */
6 1 0 CALL FETCH('01'B, ID, D_NAME, KEY_VAL, FND);
7 1 0 IF ALLOCATION(INFO_NO)>0 THEN DO;
   DATA = INFO_NO;
   FREE INFO_NO;
   FOUND = '1'B;
11 1 1 END;
12 1 0 ELSE FOUND = '0'B;
13 1 0 RETURN;
14 1 0 END FETCHV;
**MODULE DESCRIPTION**

**DEFO0010**

**SETNAMEI**, /* BIT(64) */
**DOMAIN1**, /* BIT(64) */
**DOMAIN2**, /* BIT(64) */
**TYPE**, /* 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 IS RESPONSIBLE FOR UPDATING A PSET'S PSET 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, I-N, N-1, AND 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 DEFINEP IN ORDER TO DEFINE THE PSET.

2) IF EQUIV='I' 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
THE CALLING MODULE.

3) OTHERWISE, IT IS NECESSARY TO BUILD THE BSET DEFINITION
IT BEGINS BY USING THE SEARCH PROCEDURE TO SEE IF DOMAIN
AND DOMAINT2 EXIST. IF ONE OR THE OTHER IS NOT FOUND AN
ERROR MESSAGE IS PRINTED AND THE PROCEDURE RETURNS. IF
A DOMAIN EXISTS IT CALLS MAPSET, PASSING IT THE APMAP AND
SPRAT FOR THE PART OF THE PSET CATALOGUE ENTRY,
AND MAPSET FINDS A FREE POINTER SLOT, UPDATES THE AP AND
SP MAPS AND RETURNS THE NUMBER OF THE SLOT.
THE NUMBER IS INSERTED INTO THE APPROPRIATE AP_POS
WITHIN BSET_CAT.

4) IF THE BSET IS M-N IT IS NECESSARY TO DEFINE A LINK
SET, THROUGH WHICH TO LINK THE TWO DOMAINS. THIS IS
DONE VIA A CALL TO DEFINEP, PASSING IT A PSETNAME CREATED
BY A CALL TO NAMEGEN. IN THIS IMPLEMENTATION, THE
SAME POINTER SLOTS IN THE LINK SET ARE ALLOCATED AS IN
THE ACTUAL DOMAINS. FOR EXAMPLE, WHATEVER AP_POS(1)
HAPRTS TO BI, THAT POINTER SLOT IN THE LINK SET IS ALSO
ALLOCATED. THIS REQUIRES SOME MODIFICATION TO THE
AP AND SP MAPS OF THE LINK SET, AS WELL AS TO THE POINTER
SLOT USED TO LINK ELEMENTS OF THE LINK SET TOGETHER.

Finally, IT IS NECESSARY TO ALLOCATE A POINTER SLOT
TO BE USED TO CHAIN SUBSETS.

5) IF THE BSET TYPE IS EITHER 1-N OR N-1 IT IS NECESSARY
TO ALLOCATE A POINTER SLOT TO BE USED TO CHAIN ELEMENTS
WITHIN A SUBSET TOGETHER. THIS IS DONE VIA A CALL TO
MAPSET.

6) THE FINAL STEP IS TO CREATE THE APPROPRIATE ENTRY IN THE
BSET CATALOGUE. THIS IS ACCOMPLISHED THROUGH A CALL TO
CREATEP, PASSING IT THE INFORMATION CONTAINED IN THE
BSET_CAT STRUCTURE BUILT DURING THIS PROCEDURE.

Input Parameters:

- SET_NAME1 - IT IS INTERPRETED IN TWO WAYS DEPENDING ON
  THE VALUE OF EQUIV. IF EQUIV = '0'B THEN
  SET_NAME1 IS THE NAME OF THE BSET TO BE
  DEFINED. OTHERWISE IT IS INTERPRETED AS THE
  NAME OF THE BSET, WHICH IS THE RECIPROCAL OF
  THE BSET TO BE DEFINED.
- DOMAIN1 - THE NAME OF THE FIRST PSET IN THE BSET.
  MUST BE PREVIOUSLY DEFINED.
- DOMAIN2 - THE NAME OF THE SECOND PSET IN THE BSET. ALSO
  MUST BE PREVIOUSLY DEFINED.
- TYPE - TYPE OF BINARY ASSOCIATION:
  1-1 - '00000000'B
  1-N - '00000001'B
  N-1 - '00000010'B
  M-N - '00000011'B
EQUIV: IF EQUIV = 'I'B MEANS THAT THE BSET TO BE DEFINED IS THE RECIPROCAL OF THE PREVIOUSLY DEFINED BSET IDENTIFIED BY SET_NAME1, OTHERWISE NO RECIPROCAL HAS BEEN PREVIOUSLY DEFINED.

OUTPUT PARAMETERS:
- IF EQUIV = 'I'B THEN THIS PROCEDURE RETURNS THE NAME OF NEW BSET IN SET_NAME1, OTHERWISE NO ARGUMENTS ARE IDENTIFIED BY SETNAME1, OTHERWISE NO RECIPROCAL HAS BEEN PREVIOUSLY DEFINED.

PROCEDURES INVOKED:
- MAPSET, DEFINEP, FETCH, CREATEP, NAMEGEN, SEARCH

***************

-- PARAMETER DECLARATIONS */

2 1 0  DCL (SET_NAME1,DOMAIN1,DOMAIN2) BIT(64), TYPE1 BIT(8), SUCCESS BIT(1), BSET_FLAG BIT(1) STATIC INIT('OSB), FREE_SLOT BIT(8), EQUIV BIT(1);

%INCLUDE BSETCAT;  

%INCLUDE PSETCAT;  

-- a;SLO.
LPOS2, /* ADDITIONAL PTR SLOT FOR LINK */ BCA0250
4 KEY_POS, /* STARTING POSITION OF KEY */ BCA0260
4 KEY_LEN ) BIT(8), /* LENGTH OF KEY */ BCA0270
3 SET_TYPE, /* SET TYPE INFO */ BCA0280
4 SUBSET, /* IF PRIMARY OR SUBSET */ BCA0290
4 SUBSET_ID, /* PTR SLOT FOR SUBSET LINK */ BCA0300
4 P_CHAIN, /* PTR SLOT PTS TO PRIMARY DCL*/ BCA0310
4 S_CHAIN ) BIT(8), /* SUBSET DCL CHAIN */ BCA0320
3 DATA_LEN BIT(15); /* LENGTH OF ELEMENTS */ BCA0330

********************
5 1 0 DCL (P_CAT.B_CAT) BIT(64) STATIC EXTERNAL;
6 1 0 DCL (A1_TO_1,A1_TO_N,N_TO_1,N_TO_N) BIT(8) STATIC EXT;

%INCLUDE IDS1:*****************************/POINTER STACK RETURNED BY SEARCH */
7 1 0 DCL IDS1 PTR EXTERNAL CONTROLLED;

%INCLUDE INFOUND:************************DATA STACK RETURNED BY FETCH */
8 1 0 DCL INFO_ND BIT(320) EXTERNAL CONTROLLED;

%INCLUDE EDEFINP:**************************/PROCEDURES CALLED */
9 1 0 DCL DEFINEP ENTRY(BIT(64),BIT(8),BIT(8),BIT(8),BIT(8)) BCA0590

%INCLUDE ECREATF:**************************CREATE PSET MODULE */
10 1 0 DCL CREATEP ENTRY(BIT(64),BIT(320),POINTER)

%INCLUDE EFETCH:***********************FETCH PSET MODULE */
11 1 0 DCL FETCH ENTRY(BIT(2),POINTER,BIT(64),BIT(64),BIT(1))

%INCLUDE ESEARCH:**********************SEARCH MODULE */
12 1 0 DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER)

%INCLUDE EMAPSFT:*********************MAP MAINTENANCE MODULE */
13 1 0 DCL MAPSET ENTRY(BIT(1),FIXED BIN(B),BIT(16),BIT(16),BIT(8))
%INCLUDE ENAMEGN:*************************/RANDOM NAME GENERATOR */
DCL NAMEGEN ENTRY(FIXED BIN(15)) RETURNS(Bit(64));

/* MISC DECLARATIONS */
DCL (ID1, ID2) ID CAT(2)) POINTER INIT(NULL),
IDX PTR CTRL;
/* TEMPORARY STRUCTURE FOR EQUIVALENCE OPERATION */
1 TEMP_INFO,
2 NAMEBit(64),
2 AP_POSI Bit(B);

/* IF THIS IS THE FIRST CALL TO DEFINE, DEFINE THE BSETCAT */
IF "BSET_FLAG" THEN

DO;
B_CAT = UNSPEC('BSET_CAT');
CALL DEFINEP(B_CAT,'00000001'B,'00000001'B,
'01000000'B,'000000100010111'B,
'0000000000'B,'0000000000'B,ID1);

BSET_FLAG='1'B;
A1.TO='00000001'B;
A1.TO.N='000000000'B;
N.TO='000000000'B;
M.TO='000000000'B;
END;

/* IF EQUIV FLAG SET, MEANS BSET TO BE DEFINED IS A RECIPROCAL */
IF "EQUIV" THEN

DO;

/* GET BSETCAT ENTRY FOR SET_NAME1 TO USE AS A TEMPLATE */
CALL FETCH('01'B, ID1, B_CAT, SET_NAME1, '0'B);
BASE=INFO.NO;
FREE INFO.NO;

/* MODIFY DOMAIN_INFO TO REFLECT RECIPROCAL */
TEMP_INFO=DOMAIN_INFO(1);
DOMAIN_INFO(1)=DOMAIN_INFO(2);
DOMAIN_INFO(2)=TEMP_INFO;

/* GENERATE NAME FOR BSET */
SET_NAME1=NAMEGEN();
SET_NAME=SET_NAME1;
TYPE=TYPE1;

/* INSERT NEW CAT ENTRY INTO BSETCAT PSET */
CALL CREATEP(B_CAT, BASE, ID1);
11 RETURN;

/* OTHERWISE */

/* BUILD CAT ENTRY FOR NEW BSET */

SET_NAME=SET_NAME;

DOMAIN_INFO.NAME(1)=DOMAIN1;

DOMAIN_INFO.NAME(2)=DOMAIN2;

TYPE=TYPE1;

/* VERIFY EXISTENCE OF EACH DOMAIN, AND ALLOCATE PTR SLOT */

DO J=1 TO 2;

CALL SEARCH('01'B,P_CAT,DOMAIN_INFO.NAME(J),IDX_ID);

IF ALLOCATION(IDS1)='0'B THEN DO;

PUT SKIP EDIT('DOMAIN',J, 'DOESNT EXIST ERROR')

(A,F(7),A);

SUCCESS='0'B;

RETURN;

END;

/* ID_CAT(J) PTS TO PSET CAT ENTRY FOR DOMAIN */

FREE IDS;

/* ALLOCATE A FREE POINTER SLOT TO BE USED FOR IMPLEMENTING BSET, UPDATE PSETCAT TO REFLECT SLOT ALLOCATED. IF NONE AVAILABLE, PRINT ERROR */

CALL MAPSET('t'B,2,ID_CAT(J)->SPMAP.ID_CAT(J)->APMAP.

FREE_SLOT);

IF 'FREE_SLOT'='00000000'B THEN DO;

PUT SKIP EDIT('NO MORE FREE SLOTS IN DOMAIN',J)

(A,F(7));

SUCCESS='FREE_SLOT';

RETURN;

END;

/* PLACE POSITION OF ALLOCATED POINTER SLOT IN BSETCAT ENTRY FOR BSET */

AP_POS(J)=FREE_SLOT;

END;

IF TYPE=M TO N THEN DO;

/* IF M TO N, NECESSARY TO DEFINE A Link Set TO BE USED TO IMPLEMENT BSET */

MN_NAME=NAMEGEN();

CALL DEFINEP(MN_NAME,'000000000'B,'00000001'B,

'00010000'B,'00000001'B,'0'B,'0'B.ID_CAT(2));
/* MODIFY ALLOCATION OF PTR SLOTS IN LINK SET SO THAT
SAME PTR SLOTS CAN BE USED AS SPECIFIED FOR EACH
OF THE DOMAINS */

62 1 1 ID_CAT(2)->L_PTR='00000111'B;
63 1 1 SUBSTR(ID_CAT(2)->SP_MAP,AP_PTR(1),1)='0'B;
64 1 1 SUBSTR(ID_CAT(2)->SP_MAP,AP_PTR(2),1)='0'B;
65 1 1 ID_CAT(2)->AP_MAP=ID_CAT(2)->SP_MAP;
66 1 1 CALL MAPSET('1'B,1,ID_CAT(2)->SP_MAP,ID_CAT(2)->
AP_MAP,SUB_ID);
67 1 1 ELSE IF TYPE=A1_TO_N;TYPE=N_TO_1 THEN
68 1 0 ENI;)
69 1 1 /* IF EITHER 1 TO N OR N TO 1, IT IS NECESSARY TO
ALLOCATE A POINTER SLOT TO BE USED TO CHAIN
ELEMENTS IN A SUBSET TOGETHER */
70 1 1 IF TYPE=A1_TO_N THEN
71 1 1 K=2;
72 1 1 ELSE K=1;
73 1 2 /* FINDS FREE PTR SLOT AND RESERVES IT BY UPDATING
PSETCAT ENTRY FOR DOMAIN TO REFLECT ALLOCATION */
74 1 2 CALL MAPSET('1'B,1,ID_CAT(K)->SP_MAP,ID_CAT(K)->
AP_MAP,SUB_ID);
75 1 2 IF SUB_ID='0'B THEN
76 1 2 DO:
77 1 2 /* INSERT ENTRY INTO THE PSETCAT PSET */
78 1 0 CALL CREATEP(B_CAT.BASE,101);
79 1 0 RETURN;
80 1 0 END DEFINE;}
%INCLUDE C:

CREATE PROCEDURE (B_SET, /* BIT(64) */
ID1, /* POINTER */
DATA1, /* BIT(320) */
DATA2, /* BIT(320) */
ID2 /* POINTER */);

PURPOSE:

1) The BSET_CAT entry for B_SET is first retrieved from BSET_CAT using the FETCH procedure. This in effect places the BSET_CAT information for this BSET into the BSET_CAT structure within the procedure.

2) The user has the choice of specifying either the actual DATA1 in the first domain or by specifying a pointer which points to the DATA1 item. If the user chooses the former, then the search procedure is invoked to return a pointer to the occurrence of DATA1 in DOMAIN1. If no value is found which matches DATA1 then an error message is printed and the procedure returns.

3) If the binary association is 1 to 1 then an internal procedure called GET_DOMAIN2 is called which returns a pointer which points to an occurrence of DATA2 in DOMAIN2. GET_DOMAIN2 either finds an existing occurrence of DATA2 which was eligible (i.e. its AP_POS(2) was NULL) or it creates a new entry via the CREATEP routine. In either event, pointer slot (AP_POS(1)) in the BEU containing DATA1 is updated to point to DATA2, and the FOR0010

4) If the BSET is 1-N then AP_POS(1) should point to the PSET catalogue definition for that subset. The procedure FOR0050
FIRST CHECKS TO SEE IF THAT POINTER SLOT IS NULL OR NOT. IF SO, THIS IS THE FIRST ENTRY FOR THIS SUBSET AND IT IS NECESSARY TO CREATE THE SUBSET DEFINITION WITHIN THE PSET CATALOGUE. THIS IS DONE VIA A CALL TO THE DEFINEP PROCEDURE, PASSING IT THE NAME OF THE PSET CONTAINING THE SUBSET (I.E. DOMAIN2), AS WELL AS THE POINTER SLOT ALLOCATED BY THE DEFINEB PROCEDURE TO BE USED FOR CHAINING ELEMENTS OF THE SUBSET TOGETHER. THE DEFINEP PROCEDURE RETURNS A POINTER TO THE CATALOG ENTRY CREATED AND THIS POINTER IS PLACED IN APPOS(1) OF THE BEU CONTAINING DATA1. IF THE POINTER SLOT WAS ORIGINALLY NON-NULL THEN IT ALREADY POINTED TO THE CATALOGUE ENTRY. THE NEXT STEP IS TO CHAIN DATA2 TO DATA1. THIS CONSISTS OF SEVERAL TASKS. FIRST GET_DOMAIN2, A CALL TO DEFINEP, PASSING IT THE NAME OF THE PSET CONTAINING DATA2 IN DOMAIN2 AND CHAINS IT INTO THE SUBSET, OR IF IT CREATES AN OCCURRENCE OF DATA2 WITHIN THE SUBSET, THIS ALSO HAS THE EFFECT OF INSERTING IT INTO DOMAIN2. IN EITHER EVENT IT RETURNS A POINTER TO THE BEU WHICH CONTAINS DATA2, AND CREATEB UPDATES THE APPOS(2) POINTER SLOT POINTED TO BY THAT POINTER TO POINT TO THE OCCURRENCE OF DATA1 IN DOMAIN1.

5) IF THE BSET IS N-1 THE LOGIC IS VERY SIMILAR BUT IN REVERSE. IT FIRST CHECKS TO SEE IF AN OCCURRENCE OF DATA EXISTS, AND IF NOT IT CREATES AN OCCURRENCE OF DATA. IT THEN CHECKS TO SEE IF THE AP_POS(2) POINTER SLOT IN THE BEU CONTAINING DATA1 IS NULL OR NOT. IF SO, IT IS NECESSARY TO CREATE A CATALOGUE ENTRY FOR THE SUBSET TO BE CREATED IN DOMAIN1, AND THIS IS ACCOMPLISHED VIA A CALL TO DEFINEP. IF IT WAS NECESSARY TO CREATE THE CATALOGUE ENTRY THEN THE AP_POS(2) POINTER SLOT OF DATA1 IS UPDATED TO POINT TO IT. THE PROCEDURE THEN CHAINS THE OCCURRENCE OF DATA1 INTO THE SUBSET, AND UPDATES THE AP_POS(1) POINTER SLOT WITHIN THE BEU CONTAINING DATA1 SO THAT IT POINTS TO THE OCCURRENCE OF DATA2.

6) IF THE BSET IS M-N THE FOLLOWING STRATEGY IS FOLLOWED: A) IT FIRST CHECKS TO SEE IF THE AP_POS(1) POINTER SLOT IN DATA1 IS NULL OR NOT. IF IT IS THEN IT CALLS DEFINEP TO CREATE A SUBSET WITHIN THE PSET IDENTIFIED BY MN_NAME AND WHICH ACTS AS A LINK SET BETWEEN DOMAIN1 AND DOMAIN2. IT THEN UPDATES THE AP_POS(1) POINTER SLOT SO THAT IT POINTS TO THE SUBSET CATALOGUE DEFINITION.

B) IT THEN CREATES AN ENTRY IN THE SUBSET OF THE LINK SET VIA A CALL TO CREATEP (NOTE THIS ALSO CREATES AN ENTRY IN THE PRIMARY LINK SET PSET).

C) IT THEN CALLS GET_DOMAIN2 WHICH EITHER FINDS AN ELIGIBLE OCCURRENCE OF DATA2 IN DOMAIN2 OR IT CREATES A NEW OCCURRENCE, AND IN ANY EVENT RETURNS A POINTER TO DATA2.
TO THE OCCURRENCE OF DATA2.

- THE FINAL TASK IS TO CHAIN THE NEWLY CREATED ENTRY FOR00950
  AP.POS(1) POINTER SLOT OF THE ENTRY IS UPDATED TO
  POINT TO THE OCCURRENCE OF DATA1, AND THE AP.POS(2)
  POINTER SLOT IS UPDATED TO POINT TO THE OCCURRENCE
  OF DATA2. FINALLY, THE AP.POS(2) POINTER SLOT OF THE
  BEU CONTAINING DATA2 IS UPDATED TO POINT TO THE
  CATALOGUE ENTRY FOR THE MN_NAME PSET.

INPUT PARAMETERS:

* BSET - NAME OF A PREVIOUSLY DEFINED BINARY SET
* ID1 - A POINTER WHICH IS EITHER NULL OR POINTS
  TO AN OCCURRENCE OF DATA1.
* DATA1 - A PREVIOUSLY CREATED ELEMENT WITHIN DOMAIN1
  NOTE: IF ID1 IS NOT NULL, DATA1 IS DISREGARDED.
* DATA2 - AN ELEMENT WITHIN DOMAIN2 WHICH EITHER EXISTS
  OR IS TO BE CREATED BY THIS PROCEDURE.
* ID2 - NOT SIGNIFICANT ON INPUT

OUTPUT PARAMETERS:

* ID2 - A POINTER WHICH POINTS TO THE OCCURRENCE OF DATA2
  WHICH WAS LINKED TO DATA1.

PROCEDURES INVOKED:

* DEFINEP, CREATEP, SEARCH, FETCH, GET_DOMAIN2 (INTERNAL)

%INCLUDE BSETCAT;
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 */
4 1 0 DCL T_ELEMENT BASED(ID), BCA00340
 2 LENGTH FIXED BIN(15), BCA00350
 2 P_ARRAY(16) POINTER, BCA00360
 2 INFO, BCA00370
 3 DATA BIT(320); BCA00380

5 1 0 DCL (P_CAT,B_CAT) BIT(64) STATIC EXTERNAL; BCA00390
%INCLUDE BSETSYM; **************************** BCA00400
/* BSET LINK TYPES */ BCA00410
6 1 0 DCL A1_TU_1 BIT(8) INIT('00000001'B), BCA00420
A1_TU_N BIT(8) INIT('00000010'B), BCA00430
A1_TU_1 BIT(8) INIT('00000100'B), BCA00440
A1_TU_N BIT(8) INIT('00001000'B); BCA00450

****************
%INCLUDE IDS1; **************************** BCA00460
/* POINTER STACK RETURNED BY SEARCH */ BCA00470
7 1 0 DCL IDS1 PTR EXTERNAL CONTROLLED; BCA00480
%INCLUDE INFOUND; **************************** BCA00490
/* DATA STACK RETURNED BY FETCH */ BCA00500
8 1 0 DCL INFO_ND BIT(320) EXTERNAL CONTROLLED; BCA00510

****************
/* PROCEDURES CALLED */
%INCLUDE EDEFINP; **************************** BCA00520
/* DEFINE PSET MODULE */ BCA00530
9 1 0 DCL DEFINE ENTRY(BIT(64),BIT(8),BIT(8),BIT(8),BIT(8),BIT(8), BCA00540
BIT(8),BIT(8),BIT(8),BIT(8),BIT(8)); BCA00550

%INCLUDE ECREAT; **************************** BCA00560
* CREATE PSET MODULE */ BCA00570
DCL CREATEP ENTRY(BIT(64),BIT(320),POINTER);

%INCLUDE EFETCH;

/* FETCH PSET MODULE */

DCL FETCH ENTRY(BIT(2),POINTER,BIT(64),BIT(64),BIT(1));

%INCLUDE ESPEARCH;

/* SEARCH MODULE */

DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER);
34 1 1 WHEN(A1_TO_N)
   DO;
   /* IF FIRST ELEMENT IN SUBSET, DEFINE SUBSET, AND CHAIN TO 
      INSTANCE OF DOMAIN 1 */
   35 1 2 IF ID1->T_ELEMENT.P_ARRAY(AP_POS(1))=NULL() THEN
      DO:
      CALL DEFINEP(BSET_CAT.NAME(2),'00000100'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 SUBSET CAT ENTRY */
      ELSE ID_CAT(2)=ID1->T_ELEMENT.P_ARRAY(AP_POS(1));
   END;
   WHEN(N_TO_1)
   DO;
   /* ESTABLISH INSTANCE OF DOMAIN 2 */
   CALL SEARCH('01',BSET_CAT.NAME(2),DATA2,IDX2, ID_POS);
   IF ALLOCATION(IDS1)= 0 THEN CALL CREATEP(BSET_CAT.NAME(2),DATA2, ID2);
   ELSE DO:
      ID2=IDS1;
      FREE IDS1;
      END;
      /* IF FIRST SUBSET ENTRY, DEFINE SUBSET AND CHAIN TO INSTANCE 
         OF DOMAIN 2 */
      IF ID2->T_ELEMENT.P_ARRAY(AP_POS(2))=NULL() THEN
         DO:
         CALL DEFINEP(DOMAIN_INFO.NAME(1),'00000100'B,'0'B, 
         '0'B,'1'B, SUB_ID,ID_CAT(1));
         ID2->T_ELEMENT.P_ARRAY(AP_POS(2))=ID_CAT(1);
         END;
      /* CHAIN INSTANCE OF DOMA In INTO THE SUBSET POINTED TO 
         BY THE INST NACE OF DOMAIN 2 */
      ID1->T_ELEMENT.P_ARRAY(SUB_ID)=ID2->T_ELEMENT.P_ARRAY(SUB_ID);
      ID2->T_ELEMENT.P_ARRAY(AP_POS(2))->T_ELEMENT.P_ARRAY( 

SUB_ID) = ID1;

/* CHAIN INSTANCE OF DOMAIN 2 TO DOMAIN 1 */
ID1->T_ELEMENT.P_ARRAY(AP_POS(1)) = ID2;

56 1 2 END;

57 1 2 WHEN(M_TO_N)
DO;

/* IF FIRST ELEMENT IN DOMAIN 2 TO BE LINKED TO INSTANCE OF 0
* DOMAIN 1, THEN DEFINE LINK SET SUBSET AND CHAIN TO INSTANCE 0
* 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;
60 1 3 CALL DEFINEP(MN_NAME, '00000100'B.iO'B.B.'O8.61 .
SUB_ID.ID_CAT(2));
61 1 3 ID1->T_ELEMENT.P_ARRAY(AP_POS(1)) = ID_CAT(2);
62 1 3 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));

63 1 2 END;

64 1 2 J = J + 1;
65 1 2 CALL CREATEP(ID_CAT(2)->CAT ENTRY NAME, '0'B.IDCAT(2));

/* CREATE INSTANCE OF LINK SET ELEMENT */
66 1 2 ID2A->T ELEMENT.P_ARRAY(AP_POS(1)) = ID1;

/* ESTABLISH INSTANCE OF DOMAIN 2 */
ID2A->T ELEMENT.P_ARRAY(AP_POS(2)) = ID2;

67 1 2 ID2->T_ELEMENT.P_ARRAY(AP_POS(2)) = ID_CAT(2)->T_ELEMENT.
P_ARRAY(ID_CAT(2)->P_CHAIN);
68 1 2 END;

/* SET PTR SLOT IN INSTANCE OF DOMAIN 2 TO POINT TO PRIMARY CAT ENTRY FOR LINK SET */
69 1 2 ID2->T_ELEMENT.P_ARRAY(AP_POS(2)) = ID_CAT(2)->T_ELEMENT.
P_ARRAY(ID_CAT(2)->P_CHAIN);
70 1 2 END;

71 1 1 OTHERWISE PUT SKIP EDIT('ERROR TYPE INCORRECT')(A);
72 1 1 END;
73 1 0 RETURN;

**************************************************************************
**************************************************************************
GET_DOMAIN2: PROC(NAME2,NAME2A,DATA2,AP_POS) RETURNS(POINTER);

/* THIS MODULE IS RESPONSIBLE FOR ESTABLISHING THE APPROPRIATE
INSTANCE OF DOMAIN 2. IT MAY EITHER FIND AN EXISTING OCCURRENCE
WHICH IS AVAILABLE, OR IT WILL CREATE A NEW OCCURRENCE.
NAME2 CORRESPONDS TO THE PSET IT SHOULD SEARCH FOR AN
EXISTING OCCURRENCE, NAME2A IS THE PSET INTO WHICH THE ELEMENT
SHOULD BE INSERTED IF CREATED, DATA2 IS THE VALUE OF THE
INSTANCE, AND AP_POS CORRESPONDS TO THE SET'S ALLOCATED
PTR SLOT IN INSTANCES OF DOMAIN 2. A NEW ELEMENT IS CREATED IF EITHER
AN EXISTING OCCURRENCE ISN'T FOUND, OR AN OCCURRENCE IS FOUND, BUT THE PTR SLOT IS ALREADY FULL, MEANING THAT THE ELEMENT IS ALREADY IN THE SET. */

DCL (NAME2A,NAME2) BIT(64), DATA2 BIT(*), AP_POS BIT(8), ID2 POINTER;

/* SEARCH PSET NAME2 FOR ANY OCCURRENCES OF DATA2 */
CALL SEARCH('10'BNAME2,DATA2,IDXX,ID.POS);

/* FOR EACH OCCURRENCE FOUND */
DO WHILE( ALLOCATION(IDSI));

/* IF 1 TO 1 THEN, THEN THE PTR SLOT IN DOMAIN 2 MUST BE
NULL IN ORDER FOR IT TO BE USED. IF MORE THAN ONE OCCURRENCE THEN USE FIRST THAT IS ACCEPTABLE */
IF TPE=AI_TO_1 THEN IF(IDSI->T_ELEMENT.P_ARRAY(AP_POS)=NULL()) THEN ID2=IDSI;
ELSE;

/* OTHERWISE, IF PTR SLOT IS NULL() OR (IN THE CASE OF A M TO N) IF THE PTR SLOT POINTS TO THE PRIMARY SET CAT ENTRY FOR THE LINK SET, THEN USE IT */
ELSE IF (IDSI->T_ELEMENT.P_ARRAY(AP.POS)=NULL()) |
IDSI->T_ELEMENT.P_ARRAY(AP.POS)=10_CAT(2)->T_ELEMENT.P_ARRAY(10_CAT(2)->P_CHAIN)) &
ID2=NULL()) THEN ID2=IDSI;
FREE IDSI;
END;

/* IF NO ACCEPTABLE INSTANCES WERE FOUND, CREATE ONE IN THE NAME2A PSET */
IF ID2=NULL() THEN CALL CREATEP(NAME2A,DATA2,ID2);

/* SEARCH PSET NAME2 FOR ANY OCCURRENCES OF DATA2 */
CALL SEARCH('10'BNAME2,DATA2,IDXX,ID.POS);

/* FOR EACH OCCURRENCE FOUND */
DO WHILE( ALLOCATION(IDSI));

/* IF 1 TO 1 THEN, THEN THE PTR SLOT IN DOMAIN 2 MUST BE
NULL IN ORDER FOR IT TO BE USED. IF MORE THAN ONE OCCURRENCE THEN USE FIRST THAT IS ACCEPTABLE */
IF TPE=AI_TO_1 THEN IF(IDSI->T_ELEMENT.P_ARRAY(AP_POS)=NULL()) THEN ID2=IDSI;
ELSE;

/* OTHERWISE, IF PTR SLOT IS NULL() OR (IN THE CASE OF A M TO N) IF THE PTR SLOT POINTS TO THE PRIMARY SET CAT ENTRY FOR THE LINK SET, THEN USE IT */
ELSE IF (IDSI->T_ELEMENT.P_ARRAY(AP.POS)=NULL()) |
IDSI->T_ELEMENT.P_ARRAY(AP.POS)=10_CAT(2)->T_ELEMENT.P_ARRAY(10_CAT(2)->P_CHAIN)) &
ID2=NULL()) THEN ID2=IDSI;
FREE IDSI;
END;

/* IF NO ACCEPTABLE INSTANCES WERE FOUND, CREATE ONE IN THE NAME2A PSET */
IF ID2=NULL() THEN CALL CREATEP(NAME2A,DATA2,ID2);
85 2 0  RETURN(ID2);
86 2 0  END GET_DOMAIN2;
87 1 0  END CREATED;
THE PURPOSE OF THIS MODULE IS TO RETRIEVE DATA ITEMS WHICH ARE LINKED WITHIN THE BINARY ASSOCIATION SET SPECIFIED BY NAME1 TO THE OCCURRENCE 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.

METHOD:
1) FIRST FETCHES THE SET_CAT ENTRY FOR NAME1.
2) IF ID1 IS NULL IT CALLS SEARCH USING NAME1 FROM SET_CAT AS THE PSET AND DATA1 AS THE BSET_CAT. IF THE POINTER SLOT ALLOCATED TO THIS BINARY ASSOCIATION IS NULL() THE MODULE RETURNS.
3) HAVING ESTABLISHED THE OCCURRENCE 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 APPOS(I).
1-N IN THIS CASE THE POINTER SLOT APPOS(I) CONTAINS A POINTER TO A CATALOG ENTRY FOR THE APPROPRIATE SUBSET WITHIN THE LINKAGE SET. FETCH IS USED TO RETRIEVE ALL ELEMENTS OF THAT SUBSET.
M-N IN THIS CASE POINTER SLOT APPOS(I) POINTS TO A CATALOG ENTRY FOR A SUBSET WITHIN THE LINKAGE SET WHICH CORRESPONDS TO ALL OF THE ELEMENTS LINKED IN THE BINARY ASSOCIATION SET.
IT CALLS THE SEARCH MODULE TO FETCH SEL00460 POINTERS TO EACH OF THE ELEMENTS IN SEL00470 THAT SUBSET, AND THEN GOES THROUGHSEL00480 THE POINTER VALUE CONTAINED IN AP_POSSEL00490 (2) TO RETRIEVETHETA CTUAL DATA ITEMSSEL00500 IN THE SECOND DOMAIN BY PASSING THATSEL00510 POINTER TO FETCH SEL00520

INPUT PARAMETERS:

1) MODE - USED TO DETERMINE HOW MANY DATA ITEMS TO FETCH:
   '01' - FIRST OCCURRENCE
   '11' - ALL OCCURRENCES

2) ID1 - PTR VALUE WHICH CAN BE USED TO IDENTIFY DATA ITEM IN DOMAIN 1. IF IT IS NOT NULL, IT IS ASSUMED TO POINT TO A VALID ITEM, AND THE DATA VALUE SPECIFIED BY DATAI IS IGNORED.

3) NAME1 - THE NAME OF THE BINARY ASSOCIATION THROUGH WHICH THE LINK IS TO BE MADE. IT MUST CORRESPOND TO AN EXISTING BSET DEFINITION IN THE BSET.CAT.

4) DATAI - IF ID1 IS NULL() DATAI IS USED AS A KEY TO ESTABLISH THE DESIRED OCCURRENCE OF DOMAIN 1.

5) ID2 - NOT USED ON INPUT.

OUTPUT PARAMETERS:

1) INFON0 - AN EXTERNAL CONTROLLED STACK OF BIT STRINGS BIT(320) WHICH CORRESPONDS TO THE DATA ITEMS FOUND IN DOMAIN 2 WHICH ARE LINKED TO THE OCCURRENCE OF DATAI. NOTE INFO_NO IS CREATED BY THE FETCH MODULE WHICH IS INVOKED BY THIS MODULE.

2) ID2 - PTR VALUE WHICH POINTS TO LAST DATA ELEMENT FOUND IN DOMAIN 2.

CALLS PROCEDURES:

FETCH, SEARCH

/* PARAMETER DECLARATIONS */

DCL NAME1 BIT(64), DATAI BIT(*), MODE BIT(2), FND BIT(1);
%INCLUDE BSETCAT;------------------------------------------SF 00060

/* BSET_CAT TEMPLATE */
DCL BASE BIT(320), /* USED FOR BEU OVERLAY */
1 BSET_CAT DEFINED (BASE), BCA00030
2 SET_NAME BIT(64), /* NAME OF BSET */ BCA00040
2 DOMAIN_INFO(2), /* DOMAIN INFORMATION */ BCA00060
3 NAME BIT(64), /* NAME OF DOMAIN */ BCA00070
3 AP_POS BIT(8), /* PTR SLOT USED FOR LINK */ BCA00080
2 TYPE BIT(8), /* TYPE OF BSET */ BCA00090
2 SUB_ID BIT(8), /* PTR SLOT FOR SUBSET LINK */ BCA00100
2 MN_NAME BIT(64); /* NAME OF M TO N LINK SET */ BCA00110

%INCLUDE PSETCAT;------------------------------------------SF 00070

/* PSET_CAT TEMPLATE */
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 */ BCA00180
3 NUMFREE BIT(8), /* NOT USED */ BCA00210
3 SEARCH_INFO, /* LINKAGE INFORMATION */ BCA00220
( 4 LTYPE, /* 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

%INCLUDE BSETSYM;------------------------------------------SF 00090

/* NAMES OF PSET_CAT AND BSET_CAT PSETS */
DCL (P_CAT,B_CAT) BIT(64) STATIC EXTERNAL; BCA00410

%INCLUDE IDS1;------------------------------------------SF 00120

/* BSET LINK TYPES */
DCL A1_TO_1 BIT(8) INIT('000000001'B), BCA00480
A1_TO_N BIT(8) INIT('000000010'B), BCA00490
M_TO_1 BIT(8) INIT('000000100'B), BCA00500
M_TO_N BIT(8) INIT('000010000'B); BCA00520

%INCLUDE IDS1;------------------------------------------SF 00140

/* POINTER STACK RETURNED BY SEARCH */
BCA00420
DCL IDS1 PTR EXTERNAL CONTROLLED;

--

DCL FETCH ENTRY(Bit(2), POINTER, Bit(64), Bit(64), Bit(1));

--

DCL SEARCH ENTRY(Bit(2), Bit(64), Bit(64), POINTER, POINTER);

--

DCL (ID, ID1, ID2, ID, CAT(2)) POINTER,
IDXX PTR EXTERNAL CONTROLLED;

/* POINTERS USED TO POINT TO BEUS */

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 */

/* IF INSTANCE OF DOMAIN 1 IDENTIFIED BY KEY, GET ID OF INSTANCE */

IF ID1=NULL();

THEN DO;

CALL SEARCH('01'B, BSET_CAT, NAME1, '1'B);

FREE INFO_NO;

IF INSTANCE FOUND AND IN BSET THEN DO;

IF ALLOCATION(ID1)='0' &

IDS1->T_ELEMENT.P_ARRAY(AP_POS(1))='NULL()

THEN ID1=IDS1;

FREE IDS1;

END;

ELSE DO;

IF ALLOCATION(IDS1)='0' THEN FREE IDS1;

RETURN;

END;
END;

/* SET ID2 TO CONTENTS OF POINTER SLOT FOR BSET */

ID2*ID1->T_ELEMENT.P_ARRAY(AP_POS(1));

IF TYPE= A1_TO_N THEN

/* GET ELEMENTS IN SUBSET POINTED TO BY ID2 */

CALL FETCH(MODE, ID, ID2->CAT_ENTRY_NAME, 'O'B,FND);

ELSE IF TYPE=M_TO_N THEN

DO

/* GET ELEMENTS IN LINK SET SUBSET PTED TO BY ID2 */

CALL SEARCH(MODE, ID2->CAT_ENTRY_NAME, 'O'B, IDXX,

ID_CAT(2));

/* GO THROUGH ELEMENTS IN LINKSET */

L=ALLOCATION(IDS1);

DO K=1 TO L;

/* SET ID2 = PTR TO INSTANCE IN DOMAIN 2 */

ID2*IDS1->T_ELEMENT.P_ARRAY(AP_POS(2));

FREE IDS1;

/* GET INSTANCE OF DOMAIN 2 */

CALL FETCH('01'B, ID2, 'O'B, 'O'B,FND);

END;

END:

/* OTHERWISE GET INSTANCE POINTED TO DIRECTLY BY ID2 */

ELSE CALL FETCH('O1'B, ID2, 'O'B, 'O'B,FND);

RETURN;

END SELECTF;
%INCLUDE DEFINEP;******************************************************************

MODULE DESCRIPTION

1

0 DEFINEP: PROCEDURE

(NAME1, /* BIT(64) */
L_TYPE1, /* BIT(64) */
KEY_POS1, /* BIT(8) */
KEY_LEN1, /* BIT(8) */
LEN1, /* BIT(16) */
SUBSET1, /* BIT(8) */
S_ID1, /* BIT(8) */
P2 /* POINTER */);

/)******************************************************************

 Purposes:

 The purpose of this procedure is to create and maintain a catalogue
 of all primary sets and subsets defined in the system. Every pset and subset
 (i.e. a pset which is a subset of another pset) has an entry in the pset
 catalogue. The catalogue entry (see cat_entry) serves several purposes:

 A) It contains information on how the pset is organized (i.e. the
    access method, hashed, b_tree, linear

 B) Information on how to interpret the contents of the pointer

 C) Serves as a header to the pset. It either contains a pointer to the
    first element in the pset, or a pointer to an index to the
    set.

 HENCE, THE PURPOSE OF THIS MODULE IS TO CREATE THAT

 CATALOGUE ENTRY, ALLOCATE ANY POINTER SLOTS NECESSARY
 TO BE USED FOR ORGANIZING THE PSET, AND CREATING ANY
 SUPPORT STRUCTURES (i.e. a scatter table, if hashed)

 Method:

 The objective of this module is to create an entry in the
 p_cat pset which represents the p_cat catalogue
 entry for the pset. In order to accomplish this it is
 necessary to build a temporary entry called cat_entry
 which is then inserted into the p_cat pset via the
 create module. Hence, the catalogue entry is stored
 within a beu which is an element within the p_cat pset.
 Whenever the catalogue entry is required, the beu which
CONTAINS IT IS FETCHED AND THE CAT_ENTRY STRUCTURE IS
OVERLAID ON DATA PORTION OF THE BEU. THE FOLLOWING
STRATEGY IS EMPLOYED TO CREATE THE CATALOGUE ENTRY.
A) IF THIS IS THE FIRST PSET TO HAVE BEEN DEFINED IT
IS FIRST NECESSARY TO CREATE THE P_CAT PSET. A
MODULE CALLED INIT P (INTERNAL TO DEFINE P) IS
RESPONSIBLE FOR THIS. INIT P ESSENTIALLY CREATES
A BEU WHICH CONTAINS THE P_CAT CATALOGUE ENTRY FOR
THE P_CAT PSET, AND THIS BECOMES THE FIRST ENTRY IN
THE P_CAT PSET. THIS ENTRY CONTAINS INFORMATION ON
HOW THE P_CAT PSET IS TO BE ORGANIZED. IN ADDITION,
INIT P CREATES A SCATTER TABLE FOR THE P_CAT
PSET. 

B) THE NEXT STEP IS TO RECLAIM A TEMPLATE FOR THE
NEW CATALOGUE ENTRY. IF THE PSET TO BE DEFINED IS
A SUBSET OF ANOTHER PSET, THEN A POINTER TO THE CATALOGUE
ENTRY FOR THAT PSET IS RETRIEVED VIA THE SEARCH ROUTINE. OTHERWISE, THE SEARCH ROUTINE IS CALLED TO
RETURN A POINTER TO THE P_CAT PSET CATALOGUE ENTRY. THE OBJECTIVE HERE IS THAT IF THE PSET IS A SUBSET OF ANOTHER PSET THEN ITS CATALOGUE ENTRY MUST REFLECT THE ORGANIZATION OF THE PSET OF WHICH IT IS A SUBSET.

C) IN EITHER CASE, THE INFORMATION IN THE BEU POINTED TO BY THE POINTER RETURNED IN (B) IS COPIED INTO A TEMPORARY STRUCTURE CALLED CAT_ENTRY.

D) IF THE PSET BEING DEFINED IS NOT A SUBSET THEN CAT_ENTRY IS MODIFIED TO REFLECT THE INFORMATION PASSED TO DEFINE P VIA THE INPUT PARAMETERS. THE INFORMATION INCLUDES: THE NAME, THE KEY POSITION AND LENGTH (BITS), LENGTH OF THE DATA ELEMENTS, AND THE LINK TYPE (HASHED, B_TREE, OR LINEAR). DEPENDING ON THE ACCESS METHOD OR LINK TYPE DESIRED FOR THE PSET IT IS ALSO NECESSARY TO ALLOCATE POINTER SLOTS. IF THE ACCESS METHOD IS HASHING VIA A SCATTER TABLE THEN MAPSET IS CALLED TO ALLOCATE (I.E. RESERVE) A POINTER SLOT TO BE USED FOR OVERFLOW CHAINING. L_POS2 IN CAT_ENTRY IS UPDATED TO REFLECT THE POSITION OF THIS PTR SLOT. IF A B_TREE IS TO BE EMPLOYED, MAPSET IS CALLED TO RESERVE A PTR SLOT TO BE USED TO CHAIN LEFT DESCENDENTS, AND ONCE TO RESERVE A PTR SLOT TO BE USED TO CHAIN RIGHT DESCENDENTS. L_POS1 AND L_POS2 ARE UPDATED ACCORDINGLY. IF A SIMPLE LINEAR PTR CHAIN IS DESIRED, MAPSET IS CALLED TO RESERVE A SINGLE PTR SLOT TO BE USED FOR CHAINING AND L_POS2 IS UPDATED TO REFLECT THAT POSITION.

E) IF THE PSET TO BE DEFINED IS A SUBSET OF ANOTHER PSET THEN THE LINK TYPE IS REQUIRED TO BE LINEAR. IF S_ID1 (AN INPUT PARAMETER) IS 0 THEN MAPSET IS CALLED TO RESERVE A POINTER SLOT TO BE USED TO CHAIN THE ELEMENTS OF THE SUBSET TOGETHER, AND THIS VALUE IS BOTH RETURNED.
AND PLACED IN CAT_ENTRY.S_ID. IF S_ID IS NON-ZERO DEF00950
THEN THE S_ID POINTER SLOT IS ASSUMED TO BE AVAILABLE DEF00960
FOR SUBSET CHAINING. (SUBSETS ARE ASSIGNED PRIMARY PSET ID) DEF01210
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 DEFINEP 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 CREATEP DEF01220
FUNCTION WHICH CREATES A BEU AND INSERTS IT INTO THE PCAT DEF01230
PSET. CREATEP RETURNS A PTR TO THE NEWLY CREATED BEU. DEF01240
H) IF THE LINK TYPE IS HASHED THEN CREATE_1 (AN IN- DEF01250
TERM 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 A XOR-DEF01310
FLOW CHAIN.) CREATE_1 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
DEFINITIONS FOR A GIVEN PSET ARE CHAINED TOGETHER DEF01400
THREE TASKS ARE ACCOMPLISHED HERE BY UPDATING THE P DEF01410
CHAIN PTR SLOT IN THE BEU CONTAINING THE SUBSET CATALOGUEDEF01420
DEFINITION SO THAT IT POINTS TO THE PRIMARY PSET DEF01430
DEFINITION, and in order to chain the subsets together the subset definition is inserted at the front of the subset chain.

J) The final step is to free the temporary structure used to build the catalogue entry.

**INPUT PARAMETERS:**

- **NAME1** - if this is not a subset of an existing PSET, then NAME1 is the name of the PSET to be defined. Otherwise, it is the name of the primary PSET for which this is a subset.
- **L_TYPE1** - the access method to be employed:
  - '00000001'B - hashing via a scatter table
  - '00000010'B - B-TREE
  - '00000100'B - linear PTR chain
- **KEY_POS1** - the starting position (in bits) of the key within the data area of BEUS within this PSET.
- **KEY_LEN1** - the length (in bits) of the key, maximum KEY_LEN1 of 128 bits.
- **LEN1** - length of data portion of BEU. (not used in this implementation since BEUs are fixed size)
- **SUBSET1** - flag to indicate if this is a subset:
  - '0'B - if not a subset
  - '1'B - if a subset
- **S_ID1** - indicates PTR slot to be used for chaining elements of subset. If not a subset, disregarded.
- **F2** - not significant on input.

**OUTPUT PARAMETERS:**

- **S_ID1** - if this is a subset and S_ID1 is initially '0'B then this module returns the value that MAPSET_RESERVED is reserved to be used for chaining.
- **P2** - PTR value which points to the PSET catalogue entry created. (mostly used by the CREATEP MODULE when it has called this module to define a PSET definition for a subset, and the PTR to this definition is to be inserted into the PTR slot of a BEU in domain1 if 1-N, or within domain2 if N-1.)

**PROCEDURES INVOKED:**

SEARCH, CREATEP, MAPSET, (INIT_P, CREATE_I) INTERNAL

```
DCL (L_TYPE1, KEY_POS1, KEY_LEN1, SUBSET1, S_ID1) BIT(8), LEN1
```
BIT(16), NAME1 BIT(64);

/* SET Cats Template */

DCL 1 CAT_ENTRY BASED(P), /* LENGTH OF ENTRY */ BCA00140
2 LENGTH FIXED BIN(15), /* LENGTH OF CAT */ 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 (HASED ETC.) */ BCA00230
4 L_POS1, /* PTR SLOT 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

**************************************************************************

/* Pointer Stack Returned by Search */

DCL IDS1 PTR EXTERNAL CONTROLLED;

**************************************************************************

/* Set Types */

DCL H_SHED BIT(8) INIT('00000001'B),
B_TREE BIT(8) INIT('00000010'B),
LINEAR BIT(8) INIT('00000100'B);

**************************************************************************

/* Procedures Called */

DCL HASH ENTRY(BIT(64), FIXED BIN(15)) RETURNS(FIXED BIN(15));
DCL HASH ENTRY(BIT(2), BIT(64), BIT(64), POINTER, POINTER);

**************************************************************************

%INCLUDE ECREATEidot
/* BEU CREATION MODULE */

DCL CREATEE ENTRY(BIT(320),BIT(16),POINTER);

**************

%INCLUDE ECREATP;******

/* CREATE PSET MODULE */

DCL CREATEP ENTRY(BIT(64),BIT(320),POINTER);

**************

%INCLUDE EMAPSET;******

/* TAP MAINTENANCE MODULE */

DCL MAPSET ENTRY(BIT(1),FIXED BIN(8),BIT(16),BIT(16),BIT(8));

**************

%INCLUDE ENAMEGN;**********

/* NAME GENERATOR */

DCL NAMEGEN ENTRY(FIXED BIN(15)) RETURNS(BIT(64));

**************

/* MISC DCL */

DCL (P1,P2,P3,ID2,P,ID_NEW) POINTER, STR BIT(320);

DCL PCATPTR POINTER STATIC EXTERNAL,

IDX PTR CTL,

P_CAT BIT(64) STATIC EXTERNAL;

/* INITIALIZE PSET CAT IF NECESSARY */

IF PCATPTR = NULL() THEN CALL INITP;

/* GET APPROPRIATE TEMPLATE */

IF SUBSETI='0'8 THEN CALL SEARCH('O1'B,PCAT,PCAT,IDX,ID2):
ELSE CALL SEARCH('O11'B,PCAT,NAME1,IDX,ID2);

/* INITIALIZE CAT ENTRY */

ALLOCATE CAT_ENTRY;

P->CAT_ENTRY.DATA=IDSI->CAT_ENTRY.DATA;

/* MODIFY TO REFLECT NEW PSET DEFINITION */

IF SUBSETI='O'B THEN DO;

CAT_ENTRY.NAME=NAME1;

CAT_ENTRY.KEY_POS=KEY_POS1;

CAT_ENTRY.KEY_LEN=KEY_LEN1;

CAT_ENTRY.DATA_LEN=LEN1;

CAT_ENTRY.L_TYPE=L_TYPE1;

/* IF HASHED ALLOCATE PTR SLOT FOR OVERFLOW CHAIN */

IF L_TYPE= MASED THEN CALL MAPSET('1'B,2,SP_MAP,AP_MAP,L_POS2):

/* IF B-TREE ALLOCATE 2 SLOTS FOR RIGHT,LEFT CHAIN */

ELSE IF L_TYPE =B_TREE THEN
DO;
CALL MAPSET('1'B,2,SP_MAP,AP_MAP,L_POS1);
END;

/* IF LINEAR ONLY 1 PTR SLOT USED */
ELSE CALL MAPSET('1'B,1,SP_MAP,AP_MAP,L_POS2);

/* IF A SUBSET THEN WORK FROM PRIMARY SET CAT ENTRY */
ELSE CALL MAPSET('1'B,1,SP_MAP,AP_MAP,L_POS2);

CALL MAPSET('1'B,1,SP_MAP,AP_MAP,P_CHAIN);

CALL CREATE_I(NAMEKEYLEN,P3);
P2->P_ARRAY(P2->L_POS2)+P3;

END;

IF SUBSET THEN
DO;
/* UPDATE CATALOGUE CHAINS TO PRIMARY AND OTHER */
SUBSET CATALOGUE ENTRIES */
P2->P_ARRAY(S_CHAINE)IDS1;  
IDS1->P_ARRAY(S_CHAIN)=P2;
END;

57 1 0 FREE IDS1;
58 1 0 FREE CAT_ENTRY;
59 1 0 RETURN;

60 1 0 INIT_P: PROC;
/* THIS MODULE IS RESPONSIBLE FOR INITIALIZING THE PRIMITIVE */
/* LAYER. THIS TASK REQUIRES IT TO INITIALIZE THE PCAT PSET, */
/* USING THE DECLARATION PROVIDED BELOW. */

61 2 0 DCL PCATPTR POINTER STATIC EXTERNAL.
/* PCAT PSET_CAT_ENTRY WITH DESIRED ORGANIZATION */
1305 1 2 LENGTH FIXED BIN(15) INIT(156),
2 P_ARRAY(16) POINTER,
2 DATA,
3 NAME BIT(64),
3 SP_MAP BIT(16) INIT('0111111111111111'B),
3 AP_MAP BIT(16) INIT('1111111111111111'B),
3 NUMFREE BIT(8),
3 SEARCH_INFO,
4 L_TYPE BIT(8) INIT('00000001'B),
4 L_POS1 BIT(8) INIT('00000010'B),
4 L_POS2 BIT(8) INIT('00000001'B),
4 KEY_POS BIT(8) INIT('00000001'B),
4 KEY_LEN BIT(8) INIT('01000000'B),
3 SET_TYPE,
4 SUBSET BIT(8) INIT('000000000'B),
4 SUBSET_ID BIT(8) INIT('000000000'B),
4 P_CHAIN BIT(8) INIT('000000000000'B),
4 S_CHAIN BIT(8) INIT('0000000000000000'B),
3 DATA_LEN BIT(16) INIT('0000000110000000'B);
/* POINTS TO PCAT PSET_CAT_ENTRY */

62 2 0 DCL PCATPTR POINTER STATIC EXTERNAL.
/* STRUCTURE OF SCATTER TABLE */
1 T_INDEX BASED(INDEX_PTR),
2 NAME BIT(64),
2 TEST_LEN FIXED BIN(15),
2 PTR_TO_ENTRY(50) POINTER,
   (P, INDEX_PTR) POINTER, STR BIT(240),
   P_CAT BIT(64) STATIC EXTERNAL;

   */ START OF PROCEDURE */
   ALLOCATE CAT_ENTRY SET(P);
   P_CAT=UNSPEC('P_CAT ');
   CAT_ENTRY.NAME=P_CAT;
   STR=STR NG(CAT_ENTRY.DATA);

   /* CREATE BEU CONTAINING CAT ENTRY */
   CALL CREATEE(STR,DATALEN,IDNEW);

   /* CREATE SCATTER TABLE FOR PSET AND INSERT PTR INTO SLOT */
   CALL CREATE_I(CAT_ENTRY.NAME,KEY_LEN,INDEX_PTR);

   /* UPDATE SCATTER TABLE TO REFLECT NEW ENTRY */
   POS-HASH(CAT_ENTRY.NAME,INDEX_PTR->TEST_LEN);
   INDEX_PTR->INDEX_PTR_TO_ENTRY(POS)->ID_NEW;

   /* SET PCATPTR TO ID OF CAT ENTRY */
   PCATPTR=IDNEW;
   FREE CAT_ENTRY;
   RETURN;
   END INITP;

   76 1 0 CREATE_I: PROC(NAME1,LEN,IDRETURN);
   /************************************************************/
   • 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 */
   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 */
   ALLOCATE INDEX_SET(ID_RETURN);

   79 2 0 INDEX.NAME_ENTRY=NAME1;
80 2 0 TEST LEN=LEN;
81 2 0 RETURN;
82 2 0 END CREATE I;
83 1 0 END DEFINEP;
**INCLUDE CREATEP;******************************************************************************
*CREO0000
*CREO0010
*CREO0020
*CREO0030
*CREO0040
*CREO0050
*CREO0060
*CREO0070
*CREO0080
*CREO0090
*CREO0100
*CREO0110
*CREO0120
*CREO0130
*CREO0140
*CREO0150
*CREO0160
*CREO0170
*CREO0180
*CREO0190

**DEFINE MODULE DESCRIPTION******************************************************************************
CREO0020
CREO0021
CREO0030
CREO0040
CREO0050
CREO0060
CREO0070
CREO0080
CREO0090
CREO0100
CREO0110
CREO0120
CREO0130
CREO0140
CREO0150
CREO0160
CREO0170
CREO0180
CREO0190

**CREATEP: PROCEDURE******************************************************************************
CREO0020
CREO0021
CREO0030
CREO0040
CREO0050
CREO0060
CREO0070
CREO0080
CREO0090
CREO0100
CREO0110
CREO0120
CREO0130
CREO0140
CREO0150
CREO0160
CREO0170
CREO0180
CREO0190

**NAME1, */ BIT(64) */
CREO0030
CREO0040
CREO0050
CREO0060
CREO0070
CREO0080
CREO0090
CREO0100
CREO0110
CREO0120
CREO0130
CREO0140
CREO0150
CREO0160
CREO0170
CREO0180
CREO0190

**DATA1, */ BIT(320) */
CREO0040
CREO0050
CREO0060
CREO0070
CREO0080
CREO0090
CREO0100
CREO0110
CREO0120
CREO0130
CREO0140
CREO0150
CREO0160
CREO0170
CREO0180
CREO0190

**IDNEW /* PTR */
CREO0050
CREO0060
CREO0070
CREO0080
CREO0090
CREO0100
CREO0110
CREO0120
CREO0130
CREO0140
CREO0150
CREO0160
CREO0170
CREO0180
CREO0190

** PURPOSE:******************************************************************************
CREO0020
CREO0021
CREO0030
CREO0040
CREO0050
CREO0060
CREO0070
CREO0080
CREO0090
CREO0100
CREO0110
CREO0120
CREO0130
CREO0140
CREO0150
CREO0160
CREO0170
CREO0180
CREO0190

**CONSTRUCTING AN ELEMENT CONTAINING THE BIT STRING NAME1 INTO THE PSET NAME1. CREO0040
**THE TABLE IS EITHER A PRIMARY PSET OR A SUBSET OF ANOTHER PSET. IF IT IS A SUBSET, THEN THE ELEMENT MUST BE INSERTED INTO BOTH THE SUBSET AND THE PRIMARY PSET. CREO0050

**METHOD:******************************************************************************
CREO0020
CREO0021
CREO0030
CREO0040
CREO0050
CREO0060
CREO0070
CREO0080
CREO0090
CREO0100
CREO0110
CREO0120
CREO0130
CREO0140
CREO0150
CREO0160
CREO0170
CREO0180
CREO0190

**A) THE FIRST TASK IS TO RETRIEVE THE P_SET CATALOGUE FOR THE NAME1 PSET. THIS IS ACCOMPLISHED BY CALLING CREO0030
**SEARCH, WHICH RETURNS A POINTER TO THE BEU CONTAINING CREO0040
**THE CATALOGUE ENTRY FOR THE PSET USING A PTR OVERLAPPED CREO0050
**ON THE BEU. THIS CATALOGUE ENTRY CONTAINS INFORMATION NECESSARY TO INSERT THE ELEMENT AND PERFORM CHAINING. CREO0060

**B) IF THE PSET IS NOT SUBSET, THEN CREATEE IS CALLED, PASSING IT DATA1. CREATEE IS THE MODULE WHICH IS CREO0070
**ACTUALLY RESPONSIBLE FOR CREATING A BEU WHICH CONTAINS A COPY OF DATA1 IN ITS DATA PORTION. CREATEE CREO0080
**RETURNS A PTR TO THE NEWLY CREATED BEU. THE SEARCH CREO0090
**ROUTINE IS THEN CALLED, PASSING IT THE NAME OF THE CREO0100
**PSET AND DATA1. SEARCH RETURNS VIA IDPOS (THE LAST CREO0110
**PARAMETER) A PTR WHICH POINTS TO THE BEU TO WHICH CREO0120
**THE NEW BEU SHOULD BE CHAINED. FINALLY, CHAIN (AN CREO0130
**INTERNAL PROCEDURE) IS CALLED, PASSING IT POINTERS TO THE CATALOGUE ENTRY AND THE NEW BEU. CHAIN IS CREO0140
**RESPONSIBLE FOR THE ACTUAL INSERTION OF THE BEU INTO CREO0150
**THE PSET, I.E., CHAINING IF LINEAR OR B_TREE LINK TYPE, CREO0160
**OTHERWISE, UPDATING PSET'S SCATTER TABLE OR CHAINING INTO AN OVERFLOW CHAIN. CREO0170

**C) IF THE PSET IS A SUBSET OF ANOTHER PSET THEN IT IS CREO0180
**NECESSARY TO INSERT THE ELEMENT INTO BOTH THE SUBSET &
THE PRIMARY PSET. THE STRATEGY EMPLOYED IS AS FOLLOWS:

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.

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.

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;
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 SLOTS 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
%INCLUDE IDS1;***************************************************************************CRP00060
/* POINTER STACK RETURNED BY SEARCH */ BCA00420
4 1 0  DCL IDS1 PTR EXTERNAL CONTROLLED; BCA00430
%INCLUDE PSETSYM;***************************************************************************CRP00080
/* PROCTYPES CALLED BY CREATEP */ CRP00090
5 1 0  DCL HASHED BIT(8) INIT('00000001'B), BCA00540
B_TREE BIT(8) INIT('00000010'B), BCA00550
LINEAR BIT(8) INIT('00000100'B); BCA00560
%INCLUDE ECRAFI;***************************************************************************CRP00100
/* I EU CREATION MODULE */ ECR00020
6 1 0  DCL CREATEE ENTRY(BIT(320),BIT(16),POINTER); ECR00030
%INCLUDE ESEARCH;***************************************************************************CRP00110
/* SEARCH MODULE */ BCA00700
7 1 0  DCL SEARCH ENTRY(BIT(2).BLT(64),BIT(64).POINTER.POINTER); BCA00710
%INCLUDE EHASH;***************************************************************************CRP00120
/* HASHING MODULE */ DEC00020
8 1 0  DCL HASH ENTRY(BIT(64),FIXED BIN(15)) RETURNS(FIXED BIN(15)); DEC00030
%INCLUDE EMAPSET;***************************************************************************CRP00130
/* MAP MAINTENANCE MODULE */ CRP00140
9 1 0  DCL MAPSET ENTRY(BIT(1).FIXED BIN(8),BIT(16),BIT(16),BIT(8)); ECR00060
%INCLUDE EPRINP;***************************************************************************CRP00150
/* DIAGNOSTIC PRINT MODULE */ DEC00050
10 1 0  DCL PRINTP ENTRY(POINTER); DC60060
*****************************************************************************
DCL NAME BIT(64), DATA1 BIT(*), (ID_RETURN, ID_POS, P, P_PTR), ID_NEW) POINTER, IDXX POINTER CONTROLLED;

/* GET PTR TO PCAT CATALOGUE ENTRY FOR PSET */
CALL SEARCH('O1'B, P_CAT, NAME1, IDXX, ID_POS);
P=IDS1;
FREE IDS1;
ID_TO_DATE=ALLOCATION(IDS1);

/* IF A PRIMARY SET CREATE NEW ELEMENT AND CHAIN IT INTO SET */
IF SUBSET='O'B THEN
  DO:
    CALL CREATEE(DATA1, DATA_LEN, ID_NEW);
  CALL SEARCH('O1'B, NAME1, DATA1, IDXX, ID_POS);
  CALL CHAIN(P, ID_NEW);
  /* ADJUST IDS1 TO REFLECT ANY DUPLICATES FOUND */
  IF ALLOCATION(IDS1)>ID_TO_DATE THEN
    FREE IDS1;
  ELSE DO:
    /* IF A SUBSET INSERT INTO BOTH SUBSET AND PRIMARY SET */
    P_PTR=P_ARRAY(P_CHAIN);
    CALL SEARCH('O1'B, P_PTR->NAME, DATA1, IDXX, ID_POS);
    /* ADJUST IDS1 TO REFLECT ANY DUPLICATES FOUND */
    IF ALLOCATION(IDS1)>ID_TO_DATE THEN
      FREE IDS1;
    /* CREATE BBU CONTAINING ELEMENT AND CHAIN INTO PRIMARY SET */
    CALL CREATEE(DATA1, DATA_LEN, ID_NEW);
    CALL CHAIN(P_PTR, ID_NEW);
    /* LOCATE POSITION IN SUBSET, ADJUST IDS1, AND CHAIN ELEMENT INTO SUBSET */
    CALL SEARCH('O1'B, NAME1, DATA1, IDXX, ID_POS);
    IF ALLOCATION(IDS1)>ID_TO_DATE THEN
      FREE IDS1;
    CALL CHAIN(P, ID_NEW);
    END;

RETURN;

******************************************************************************/
CHAIN: PROC(CAT_PTR, DATA_PTR);

/* THIS PROCEDURE IS RESPONSIBLE FOR CHAINING ELEMENTS INTO PREVIOUSLY DEFINED PSETS, GIVEN CAT_PTR WHICH POINTS TO THE PSET CAT ENTRY, DATA_PTR WHICH POINTS TO THE BEU TO BE INSERTED, AND ID_POS (A GLOBAL VARIABLE) WHICH POINTS TO THE BEU TO WHICH THIS BEU SHOULD BE CHAINED. */

DCL (CAT_PTR, DATA_PTR, INDEX_PTR) POINTER;
DCL 1 T_INDEX BASED(INDEX_PTR),
    2 NAME BIT(64),
    2 TEST_LEN FIXED BIN(15),
    2 PTR_TO_ENTRY(50) POINTER;
DCL POS FIXED BIN(15);
DCL 1 T_ELEMENT BASED(DATA_PTR),
    2 LENGTH FIXED BIN(15),
    2 P_ARRAY(16) POINTER,
    2 INFO BIT(320); */

IF FIRST ELEMENT IN PSET *
IF ID_POS = NULL()
    THEN DO;
        IF CAT_PTR->L_TYPE = HASHED
            THEN DO:
                /* UPDATE APPROPRIATE SCATTER TABLE ENTRY */
                INDEX_PTR=CAT_PTR->P_ARRAY(CAT_PTR->L_POS1);
                POS=HASH(DATA_PTR->T_ELEMENT.INFO,TEST_LEN);
                PTR_TO_ENTRY(POS)=DATA_PTR;
                END;
            /* OTHERWISE CHAIN TO CATALOGUE ENTRY */
            ELSE CAT_PTR->P_ARRAY(CAT_PTR->L_POS2)=DATA_PTR;
        END;
    ELSE CAT_PTR->P_ARRAY(CAT_PTR->L_POS2)=DATA_PTR;
    RETURN;
END;

/* NOT FIRST ELEMENT IN PSET */
ELSE IF CAT_PTR->L_TYPE=B_TREE
    THEN DO:
        /* IF ELEMENT IS LESS THAN ELEMENT POINTED TO BY ID_POS, CHAIN AS A LEFT DESCENDANT */
        IF SUBSTR(ID_POS->T_ELEMENT.INFO,CAT_PTR->KEY_POS,CAT_PTR->KEY_LEN) > SUBSTR(DATA_PTR->T_ELEMENT.INFO,CAT_PTR->KEY_POS,CAT_PTR->KEY_LEN) THEN
            ID_POS->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS1)=DATA_PTR;
        END;
    ELSE IF CAT_PTR->L_TYPE=B_TREE
        THEN DO:
            /* CHAIN TO CATALOGUE ENTRY */
            CAT_PTR->P_ARRAY(CAT_PTR->L_POS2)=DATA_PTR;
        END;
    RETURN;
END;
/* OTHERWISE IT MUST BE A RIGHT DESCENDENT, AND CHAIN ACCORDINGLY */

49 2 1
   ELSE DO:
      CRP0140
50 2 2
   DATA_PTR->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2); =
   CRP0150
   ID_POS->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2);
   CRP0160
51 2 2
   ID_POS->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2)
   = DATA_PTR;
   CRP0160
52 2 2
   END;
   CRP0170
53 2 1
   END;
   CRP0180

/* IF HASHED OR LINEAR CHAIN TO ELEMENT POINTED TO BY ID_POS */

54 2 0
   ELSE DO:
      CRP0190
55 2 1
   DATA_PTR->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2); =
   CRP0200
   ID_POS->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2);
   CRP0210
56 2 1
   ID_POS->T_ELEMENT.P_ARRAY(CAT_PTR->L_POS2); = DATA_PTR;
   CRP0220
57 2 1
   END;
   CRP0230
58 2 0
   RETURN;
   CRP0240
59 2 0
   END CHAIN;
   CRP0250

60 1 0
   END CREATEP;
%include search: ................................................................. sea00010

**** MODULE DESCRIPTION ****

search: procedure:

mode, /* bit(2) */
set_name, /* bit(64) */
data, /* bit(*) */
idx, /* ptr */
ids, /* ptr */
)

purposes:
this module is responsible for retrieving the ids of 1
or more elements within the pset pset_name, given the
key specified by data, and the search mode specified by
mode. it returns the ids of the elements found in a ctl
structure called ids1.

methods:
the search method depends on both the pset organization
hashed, b-tree, or linear, and the search mode ( first
element which matches key, all elements which match key,
or all elements within the set ). general logic is as
follows:

a) retrieves psetcat entry for pset identified by
   pset_name. this is done via a hash search of the
   pcat pset. via a string overlay, cat_entry is overlaid

b) on the beu containing the psetcat entry for the pset. for00300
   if the search mode is first, or all elements which
   match key, h_search, b_search, or l_search is called
   depending on the pset organization. these are internal for00330
   routines which return 2 pointers, id which is a pointer for00340
   to the first occurrence of the element, and id_chain for00350
   which is a ptr to the last occurrence of the element
   found. ( if no element was found which
   matched the key, id_chain points to the beu to which
   the element should be chained). id51 is set equal to
   the pointer found

   c) if all elements which match key was specified, then
      a linear search is invoked starting at the first
      first match and going until no other matches are possible. for00420
      if linear organization this means until the end of the set for00440
      if hashed, to the end of the overflow chain, if b_tree
UNTIL THE RIGHT DESCENDENTS NO LONGER MATCH. EVERY ID RETURNED IS PLACED ON THE IDSI STACK.

D) IF THE IDS OF ALL ELEMENTS IN THE SET ARE TO BE FETCHED SCATTER TABLE AND OVERFLOW CHAINS, AND RETURNS THE IDS OF ALL ELEMENTS FOUND. LINEAR_L PERFORMS A LINEAR SEARCH OF THE SET'S IDTS FOR THE IDS FOUND.

**INPUT PARAMETERS:**
- MODE - SPECIFIES THE SEARCH MODE:
  - '01' - FIRST ELEMENT WHICH MATCHES KEY.
  - '10' - ALL ELEMENTS WHICH MATCH KEY.
  - '11' - ALL ELEMENTS IN SET.
- PSET_NAME - NAME OF PSET TO BE SEARCHED.
- IDXX - NAME OF PSE SET ORGANIZATION.
- ID_CHAIN - A PIR VALUE WHICH POINTS TO BEU TO WHICH ELEMENT SHOULD BE CHAINED IF NOT FOUND, OTHERWISE Points TO THE LAST OCCURRENCE FOUND DURING SEARCH.

**OUTPUT PARAMETERS:**
- IDSI - A CONTROLLED STACK OF POINTERS CORRESPONDING TO THE RETRIEVED IDS.
- ID_CHAIN - A PIR VALUE WHICH POINTS TO BEU TO WHICH ELEMENT SHOULD BE CHAINED IF NOT FOUND, OTHERWISE POINTS TO THE LAST OCCURRENCE FOUND DURING SEARCH.

**CALLS PROCEDURES:**
- HASH, (H_SEARCH, L_SEARCH, B_SEARCH, LINEAR_H, LINEAR_B, LINEAR_L) INTERNAL

/* PSET_CAT TEMPLATE */

%INCLUDE PSETCAT
(4 L_TYPE, /* TYPE OF LINK (HASHED ETC..) */ DCA00330
4 L_POS1, /* PTR SLOT USED FOR CHAINING */ DCA00240
4 L_POS2, /* ADDITIONAL PTR SLOT FOR LINK */ DCA00250
4 KEY_POS, /* STARTING POSITION OF KEY */ DCA00260
4 KEY_LEN ) BIT(8), /* LENGTH OF KEY */ DCA00270
3 SET_TYPE, /* SET TYPE INFO */ DCA00280
4 SUBSET, /* IF PRIMARY OR SUBSET */ DCA00290
4 SUBSET_ID, /* PTR SLOT FOR SUBSET LINK */ DCA00300
4 P_CHAIN, /* PTR SLOT PTS TO PRIMARY DCL */ DCA00310
4 S_CHAIN ) BIT(8), /* SUBSET DCL CHAIN */ DCA00320
3 DATA_LEN BIT(15); /* LENGTH OF ELEMENTS */ DCA00330

******************************************************************************
%INCLUDE SEU;******************************************************************************

3 1 0 DCL 1 ELEMENT BASED(Q), DCA00350
2 LENGTH FIXED BIN(15), DCA00360
2 P_ARRAY(16) PTR, DCA00370
2 INFO, DCA00380
3 DATA BIT(320); DCA00390

******************************************************************************
/* MISC DECLARATIONS */
4 1 0 DCL MODE BIT(2), DCA00070
FIRST BIT(2) INIT('01'B), DCA00080
ALL_SET BIT(2) INIT('11'B), DCA00090
ALL_SAME BIT(2) INIT('10'B); DCA00100

%INCLUDE PSETS W;******************************************************************************

5 1 0 DCL HASHED BIT(H) INIT('00000001'B), DCA00540
B_TREE BIT(U) INIT('00000010'B), DCA00550
LI_EAR BIT(B) INIT('00000100'B); DCA00560

******************************************************************************
/* PSET LINK TYPES */
6 1 0 DCL PCATPTR POINTER EXTERNAL STATIC; DCA00130
7 1 0 DCL (P.Q.ID_CHA.IND1.IDS TART ) POINTER; DCA00140
8 1 0 DCL :ID51 PTR EXTERNAL CONTROLLED; DCA00150

%INCLUDE IDS1;******************************************************************************

9 1 0 DCL IDXX POINTER CONTROLLED; DCA00170
10 1 0 DCL T_DATA BIT(128) VARYING; DCA00180
11 1 0 DCL P.CAT BIT(64) STATIC EXTERNAL, DCA00200
PSET_NAME BIT(34),DATA1 BIT(1), DCA00210

%INCLUDE PSET LINK TYPES */
6 1 0 DCL PCATPTR POINTER EXTERNAL STATIC; DCA00130
7 1 0 DCL (P.Q.ID.CHAIN.ID1.IDS TART ) POINTER; DCA00140
8 1 0 DCL :ID51 PTR EXTERNAL CONTROLLED; DCA00150

%INCLUDE IDS1;******************************************************************************

9 1 0 DCL IDXX POINTER CONTROLLED; DCA00170
10 1 0 DCL T_DATA BIT(128) VARYING; DCA00180
11 1 0 DCL P.CAT BIT(64) STATIC EXTERNAL, DCA00200
PSET_NAME BIT(34),DATA1 BIT(1), DCA00210

%INCLUDE IDS1;******************************************************************************

9 1 0 DCL IDXX POINTER CONTROLLED; DCA00170
10 1 0 DCL T_DATA BIT(128) VARYING; DCA00180
11 1 0 DCL P.CAT BIT(64) STATIC EXTERNAL, DCA00200
PSET_NAME BIT(34),DATA1 BIT(1), DCA00210

%INCLUDE IDS1;******************************************************************************

9 1 0 DCL IDXX POINTER CONTROLLED; DCA00170
10 1 0 DCL T_DATA BIT(128) VARYING; DCA00180
11 1 0 DCL P.CAT BIT(64) STATIC EXTERNAL, DCA00200
PSET_NAME BIT(34),DATA1 BIT(1), DCA00210

%INCLUDE IDS1;******************************************************************************

9 1 0 DCL IDXX POINTER CONTROLLED; DCA00170
10 1 0 DCL T_DATA BIT(128) VARYING; DCA00180
11 1 0 DCL P.CAT BIT(64) STATIC EXTERNAL, DCA00200
PSET_NAME BIT(34),DATA1 BIT(1), DCA00210

%INCLUDE IDS1;******************************************************************************

9 1 0 DCL IDXX POINTER CONTROLLED; DCA00170
10 1 0 DCL T_DATA BIT(128) VARYING; DCA00180
11 1 0 DCL P.CAT BIT(64) STATIC EXTERNAL, DCA00200
PSET_NAME BIT(34),DATA1 BIT(1), DCA00210

%INCLUDE IDS1;******************************************************************************

9 1 0 DCL IDXX POINTER CONTROLLED; DCA00170
10 1 0 DCL T_DATA BIT(128) VARYING; DCA00180
11 1 0 DCL P.CAT BIT(64) STATIC EXTERNAL, DCA00200
PSET_NAME BIT(34),DATA1 BIT(1), DCA00210

%INCLUDE IDS1;******************************************************************************

9 1 0 DCL IDXX POINTER CONTROLLED; DCA00170
10 1 0 DCL T_DATA BIT(128) VARYING; DCA00180
11 1 0 DCL P.CAT BIT(64) STATIC EXTERNAL, DCA00200
PSET_NAME BIT(34),DATA1 BIT(1), DCA00210

%INCLUDE IDS1;******************************************************************************

9 1 0 DCL IDXX POINTER CONTROLLED; DCA00170
10 1 0 DCL T_DATA BIT(128) VARYING; DCA00180
11 1 0 DCL P.CAT BIT(64) STATIC EXTERNAL, DCA00200
PSET_NAME BIT(34),DATA1 BIT(1), DCA00210

%INCLUDE IDS1;******************************************************************************

9 1 0 DCL IDXX POINTER CONTROLLED; DCA00170
10 1 0 DCL T_DATA BIT(128) VARYING; DCA00180
11 1 0 DCL P.CAT BIT(64) STATIC EXTERNAL, DCA00200
PSET_NAME BIT(34),DATA1 BIT(1), DCA00210

%INCLUDE IDS1;******************************************************************************

9 1 0 DCL IDXX POINTER CONTROLLED; DCA00170
10 1 0 DCL T_DATA BIT(128) VARYING; DCA00180
11 1 0 DCL P.CAT BIT(64) STATIC EXTERNAL, DCA00200
PSET_NAME BIT(34)
/* EXTERNAL PROCEDURES CALLED */
(PRINTP,PRINTIT) EXTERNAL ENTRY;
%INCLUDE EHASH;**********************************************************************
/* HASHING MODULE */
DCL HASH ENTRY(BIT(64),FIXED BIN(15)) RETURNS(FIXED BIN(15));
**********************************************************************

/* GET _N FOR PSET_CAT ENTRY FOR PSET */
CALL H_SEARCH(PSET_NAME, PSET, ID1, ID_CHAIN);

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

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

/* DISPATCH APPROPRIATE INTERNAL SEARCH ROUTINE */
IF MODE=FIRST ;MODE=ALL_SAME THEN DO;
IF L_TYPE=HASHED THEN
   CALL H_SEARCH(T_DATA1, ID1, ID_CHAIN);
ELSE IF L_TYPE=B_TREE THEN
   CALL B_SEARCH(T_DATA1, ID1, ID_CHAIN);
ELSE CALL L_SEARCH(T_DATA1, CAT_ENTRY, P_ARRAY(L_POS2), ID1, ID_CHAIN);
/
/* IF VALUE FOUND PUT POINTER ON STACK */
IF ID1=NULL() THEN
   DO;
      ALLOCATE IDS1;
      IDS1=ID1;
   END;
      
/* OTHERWISE RETURN */
ELSE RETURN;

IF MODE=ALL_SAME THEN
   /* DO LINEAR SEARCH FOR REMAINING ELEMENTS */
   DO WHILE(IS1=ID1); 
      /* 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;
         END;
      
      /* DO LINEAR SEARCH FOR REMAINING ELEMENTS */
      DO WHILE(IS1=ID1);
      /* 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;
         END;
   END;
ID_START=ID1;
END;
RETURN;
END;
/* IF ALL OF SET TO BE RETRIEVED DISPATCH APPROPRIATE ROUTINE */
IF MODE=ALL_SET THEN
DO:
IF L_TYPE = HASHED THEN
CALL LINEAR_H(IDS1);
ELSE IF L_TYPE = B_TREE THEN
CALL LINEAR_B(IDS1);
ELSE CALL LINEAR_L(CAT_ENTRY.P_ARRAY(L_POS2), IDS1);
END;
RETURN;

UTILITY SUBROUTINES USED BY SEARCH */
L_SEARCH: PROC(T_DATA1.START, IDIA, IDIB);
DCL T_DATA1 BIT(*),
(START, IDIA, IDIB) POINTER;
IDIB=NULL();
/* FOLLOW POINTER CHAIN LINKING ELEMENTS IN PSET */
DO IDIA=START REPEAT IDIA->T_ELEMENT.P_ARRAY(L_POS2)
WHILE(IDIA%NULL());
IDIB=IDIA;
IF T_DATA1=SUBSTR(IDIA->T_ELEMENT.DATA.KEY_POS.KEY_LEN)
THEN RETURN;
END;
RETURN;
END L_SEARCH;
H_SEARCH: PROC(T_DATA1, IDIA, IDIB);
DCL T_DATA1 BIT(*), (IDIA, IDIB) POINTER,
Q POINTER, INDEX_POS FIXED BIN,
/* TEMPLATE FOR SCATTER TABLE */
1 T_INDEX BASED(Q),
2 NAME BIT(64),
2 L_HASH FIXED BIN(15),
2 PTR_TO_ENTRY(50) POINTER;
IDIB=NULL();
/* HASH INTO SCATTER TABLE USING KEY, AND LENGTH IN T_INDEX */
INDEX_POS=HASH(T_DATA1, P->CAT_ENTRY.P_ARRAY(L_POS1)->L_HASH);
/* CHECK BEU POINTED TO BY T_INDEX ENTRY, IF NO MATCH FOLLOW OVER LOW CHAIN */
DO IDIA=CATENTRY.P_ARRAY(LPos1)->PTR_TO_ENTRY(INDEX-POS)
REPEAT IDIA=IDIA->T_ELEMENT.P_ARRAY(LPos2)
WHILE(IDIA=NULL());

IF T_DATA1=SUBSTR(IDIA->T_ELEMENT.DATA.KEYPos.KEY_LEN)
THEN RETURN;
END;

BSEARCH:
PROC(T_DATA1,IDIA,IDIB);
DCL T DATA1 BIT(*), (IDIA,IDIB) POINTER;
IDIB=IDIA;
DO WHILE (IDIB=NULL());
IF T_DATA1=SUBSTR(IDIB->T_ELEMENT.DATA.KEYPos.KEY_LEN)
THEN RETURN;
ELSE IF SUBSTR(IDIB->T_ELEMENT.DATA.KEYPos.KEY_LEN)>T_DATA1
THEN IDIA=IDIB->T_ELEMENT.P_ARRAY(LPos1);
ELSE IDIA=IDIA->T_ELEMENT.P_ARRAY(LPos2);
END;
RETURN;

LINEAR_L:
PROC(START,IOS11);
DCL (ID,START) pointer, IDS11 pointer controlled;
DO ,=START REPEAT ID->T_ELEMENT.P_ARRAY(LPos2)
WHILE (ID=NULL());
ALLOCATE IDS11;
ID11=ID;
RETURN;

LINEAR_H:
PROC(IDS11);
DCL IDS11 pointer controlled, ID POINTER;
1 T_INDEX BASED(0),
2 NAME BIT(4),
2 HASH_LEN FIXED BIN(15),
2 PTR_TO_ENTRY(50) POINTER,
Q POINTER;
Q=CAT_ENTRY.P_ARRAY(LPos1);
DO J=1 TO 50;
DO ID=Q->PTR_TO_ENTRY(J) REPEAT
ID->T_ELEMENT.P_ARRAY(LPos2)
WHILE (ID=NULL());
ALLOCATE IDS11;
LINEAR_B: PROC(IDS11);
DCL (IDS11,IDTEMP) POINTER CONTROLLED,
C POINTER,FLAG BIT(1);
Q=SAT_ETRY.P_ARRAY(L_POST);
FLAG='I B;
/* DO A INORDER DEPTH FIRST TRAVERSAL */
DO WHILE (FLAG);

/* GET TO BOTTOM ELEMENT IN BRANCH */
DO WHILE(Q=NULL());
/* TEMPORARY STACK FOR NODES PASSED ON WAY DOWN */
ALLOCATE IDTEMP;
IDTEMP=Q;
Q=Q->T_ELEMENT.P_ARRAY(L_POST);
END;

/* WORK YOUR WAY BACK UP BRANCH */
IF ALLOCATION(IDTEMP)='O THEN
DO;
ALLOCATE IDS11;
IDS11=IDTEMP;
Q=IDTEMP->T_ELEMENT.P_ARRAY(L_POST);
FREE IDTEMP;
END;
ELSE RETURN;
END;
END LINEAR_B;
END SEARCH;
**MODULE DESCRIPTION**

```plaintext
PROCEDURE FETOO010
(MODE, /* BIT(2) */
ID, /* PTR */
NAME, /* BIT(64) */
KEY, /* BIT(*) */
FND /* BIT(1) */);
```

**PURPOSE:**

This module is responsible for retrieving the data portion 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 corresponding 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) 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:

- MODE - 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, locating INFO_NO, and setting the current allocation of INFO_NO 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...
ISTAKEN, IN THIS CASE THE ID IS ASSUMED TO POINT TO THE DESIRED BEU AND NO SEARCHING IS NECESSARY. THE BEU TEMPLATE IS USED TO EXTRACT THE DATA PORTION OF THE BEU POINTED TO BY ID, AND AN ALLOCATION OF INFO_NO IS CREATED TO RETURN THE DATA.

INPUT PARAMETERS:
- MODE - THE RETRIEVAL MODE (SAME AS FOR SEARCH ROUTINE)
- '01'G - FIRST ELEMENT MATCHING KEY
- '10'G - ALL ELEMENTS MATCHING KEY
- '11'G - ALL ELEMENTS IN SET, REGARDLESS.
- ID - A PTR WHICH IS EITHER NULL OR POINTS TO A BEU CONTAINING THE DESIRED DATA. NOTE:
  - IF ID ISN'T NULL THEN FETCH ASSUMES THAT IT IS A VALID BEU REFERENCE.
- NAME - THE NAME OF THE PSET WHICH CONTAINS THE ELEMENTS TO BE FETCHED. IT MAY BE A SUBSET OF ANOTHER PSET. IN EITHER CASE, HOWEVER,
  - THE PSET MUST HAVE BEEN PREVIOUSLY DEFINED
- KEY - KEY TO BE SEARCHED ON WITHIN PSET.
  - AND LENGTH TAKEN FROM P_CAT ENTRY.
  - IF ALL ELEMENTS TO BE FETCHED THEN KEY DISREGARDED
- FND - NOT SIGNIFICANT ON INPUT.

OUTPUT PARAMETERS:
- FND - IF ATLEAST 1 ELEMENT WAS FETCHED THE EQUAL TO '1'B, OTHERWISE EQUALS '0'B
- INFQ-ND - AN EXT CIL BIT STRING OF LENGTH(320) USED TO RETURN THE DATA ELEMENTS FOUND. SINCE, SEARCH RETURNS A CONTROLLED STACK OF POINTERS WHICH ARE USED TO CREATE INFO_NO, THE ELEMENTS OF THE INFO_NO STACK ARE IN THE SAME ORDER AS THE DATA ELEMENTS FOUND IN THE PSET.
- CALLS PROCEDURES:
  - SEARCH.

%INCLUDE BEU;
2 INFO.
3 DATA BIT(320): *************

/ * STACK TO RETURN FOUND ELEMENTS */

%INCLUDE INFOND:**********************

/ * DATA STACK RETURNED BY FETCH */

4 1 0 DCL INFO_ND BIT(320) EXTERNAL CONTROLLED;

%INCLUDE IDS1; **********

/ * POINTER STACK RETURNED BY SEARCH */

5 1 0 DCL IDS1 PTR EXTERNAL CONTROLLED;

/ * MISC DECLARATIONS */

6 1 0 DCL SEARCH ENTRY(BIT(2),BIT(64),BIT(64),POINTER,POINTER); *************

7 1 0 IF ID=NULL() THEN DO:

/ * GET IDS OF ITEMS IN SET TO BE FETCHED */

8 1 1 CALL SEARCH(MODEL,NAME,KEY,IDX,Z):

9 1 1 L=ALLOCATION (IDS1):

10 1 1 IF L>0 THEN FND='1'B;

11 1 1 ELSE FND='0'B;

/ * FETCH DATA CONTENTS OF BEUS POINTED TO BY IDS1 */

12 1 1 DO J=1 TO L;

/ * PUT ITEMS ON STACK */

13 1 2 ALLOCATE INFO_ND;

14 1 2 INFO_ND=IDS1->DATA;

15 1 2 FREE IDS1;

16 1 2 END:

17 1 1 RETURN;

18 1 1 END;

/ * IF ONLY 1 ELEMENT WAS TO BE FETCHeD */

19 1 0 ALLOCATE INFO_ND;

20 1 0 INFO_ND=ID->DATA;

21 1 0 FND='1'B;

22 1 0 RETURN;

23 1 0 END FETCH;
%INCLUDE CREATEE;******************************CRE00010
***********************************************************************CRE00010
* CRE00020
* CRE00030
***********************************************************************CRE00040
* CRE00050
* CRE00060
* CRE00070
***********************************************************************CRE00080
* CRE00090
* CRE00100
* CRE00110
* CRE00120
* CRE00130
* CRE00140
* CRE00150
* CRE00160
* CRE00170
* CRE00180
* CRE00190
* CRE00200
* CRE00210
* CRE00220
* CRE00230
* CRE00240
* CRE00250
* CRE00260
* CRE00270
* CRE00280
* CRE00290
* CRE00300
* CRE00310
* CRE00320
* CRE00330
* CRE00340
* CRE00350
* CRE00360
* CRE00370
* CRE00380
* CRE00390
* CRE00400
* CRE00410
* CRE00420
* CRE00430
* CRE00440
* CRE00450

MODULE DESCRIPTION
***********************************************************************CRE00010

PROGRAM CREATEE: PROCEDURE CREATEE

STR, /* INPUT */
ZLEN, /* FIXED BIN(15) */
ID_CREATED /* PTR */;

***********************************************************************CRE00020

PURPOSE:

THIS MODULE IS RESPONSIBLE FOR CREATING THE BASIC STORED
UNIT OF INFORMATION IN THE SYSTEM, CALLED A BINARY EN-
CODING UNIT. CONCEPTUALLY A BEU LOOKS AS FALLS:

<table>
<thead>
<tr>
<th>LENGTH</th>
<th>POINTER ARRAY</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

WHERE: LENGTH - LENGTH OF RELEVANT DATA IN DATA AREA
BEU (IN BITS). IN THIS IMPLEMENTATION THE
DATA AREA IS FIXED LENGTH, SO THE LENGTH
FIELD INDICATES THE PORTION OF THAT DATA
AREA WHICH IS VALID. IN A FUTURE IM-
PLEMENTATION THE LENGTH FIELD WOULD IN-
DICATE THE ACTUAL LENGTH OF THE DATA AREA

POINTER ARRAY - THIS POINTER ARRAY IS USED TO IM-
PLEMENT BINARY ASSOCIATIONS BETWEEN BEU
AS WELL AS TO LINK BEUS WITHIN A COMMON
SET. DEFINED AND DEFINED ARE RESPONSIBLE
FOR MANAGING THE POINTER ARRAYS, I.E.
ALLOCATING POINTER SLOTS TO VARIOUS PUR-
POSES. THE STATUS OF POINTER SLOTS IS
CONTAINED IN THE PSET'S P_CAT CATALOGUE
ENTRY. IN THIS IMPLEMENTATION THE PTR
ARRAY HAS FIXED EXTENTS (16). FUTURE IM-
PLEMENTATIONS MAY WANT THIS TO BE VARIABLE

DATA - A FIXED LENGTH BIT STRING CONTAINING THE
ACTUAL DATA. NOTE: MODULES WHICH ACCESS
THE BEU ARE RESPONSIBLE FOR INTERPRETING ITS
CONTENTS. THIS IS OFTEN DONE BY OVERLAYING
BASED STRUCTURES ON THE DATA AREA.

METHOD:

THE APPROACH IS VERY SIMPLE:

A) IT BEGINS BY ALLOCATING A BASED STRUCTURE CALLED
**ELEMENT**. **ELEMENT** IS DECLARED AS FOLLOWS:

1 ELEMENT.
2 LENGTH FIXED BIN(15),
2 P_ARRAY(16) PTR INIT((16) NULL()),
2 INFO BIT(320);
B) ELEMENT.LENGTH AND ELEMENT.INFO ARE SET EQUAL TO
ZLEN AND STR RESPECTIVELY.
C) RETURNS A PTR (ID_CREATED) WHICH POINTS TO THE NEWLY
ALLOCATED BEU.

---

**INPUT PARAMETERS:**
- SIR - BIT STRING OF A LENGTH NOT EXCEEDING 320 BITS.
- ZLEN - LENGTH (IN BITS) OF THE ELEMENT TO BE INSERTED.
- ID_CREATED - NOT SIGNIFICANT ON INPUT.

**OUTPUT PARAMETERS:**
- ID_CREATED - A PTR WHICH POINTS TO THE BEU CREATED.

**CALLS PROCEDURES:**
- NONE

---

```
2 1 0      DCL STR BIT(*), ZLEN BIT(16), ID_CREATED POINTER;
           /* JEU STRUCTURE */
3 1 0      DCL 1 ELEMENT BASED(ID_CREATED),
           2 LENGTH FIXED BIN(15),
           2 P_ARRAY(16) POINTER INIT((16) NULL()),
           2 INFO BIT(320) INIT((320)'0'B);
4 1 0      ALLOCATE ELEMENT;
5 1 0      ELEMENT.LENGTH=ZLEN;
6 1 0      ELEMENT.INFO=SIR;
7 1 0      RETURN;
8 1 0      END CREATEE;
```
MAPSET: PROC(MODE, TYPE, MAP1, MAP2, FREE);

DCL (TYPE, 1) FIXED BIN(8), FREE BIT(8),
  (MAP1, MAP2, MAP(2)) BIT(16), MODE BIT(1);

MAP(1)=MAP1;
MAP(2)=MAP2;
DO I=1 TO 16;
  IF SUBSTR(MAP(TYPE), I, 1)=MODE
    THEN DO:
      SUBSTR(MAP1, I, 1)=("MODE");
      SUBSTR(MAP2, I, 1)=("MODE");
      FREE=BIN(I);
    RETURN;
  END;
END;

FREE=0;
RETURN;
END MAPSET;
HASH: PROC(STR,NUMCHAR) RETURNS(FIXED BIN(15));

DCL STR BIT(*), NUMCHAR FIXED BIN(15), TEMPCHAR BIT(16),
(NUM,VALUE) FIXED BIN(31) INIT(0), NAME CHAR(8);

UNSPEC(NAME)=STR;
K=NUMCHAR;
DO J=1 OK BY 16:
TEMPCHAR=SUSSTR(STR,J,16);
NUM=TEMPCHAR;
VALUE=NUM+VALUE;
END;
VALUE=MOD(VALUE,51);
IF VALUE=0 THEN VALUE =1;
RETURN(VALUE);

END HASH;
PRINTIT: PROC(IFLAG,PPP):
DCL (P,PPP) POINTER;
DCL 1 T_ELEMENT BASED(P),
2 LEN FIXED BIN(15),
2 P_ARRAY(16) POINTER,
2 DATA BIT(320), IFLAG FIXED BIN(15);
IF PPP=NULL() THEN
  PUT L1!T('POINTER IS NULL');
ELSE PUT SKIP EDIT(IFLAG,SUBSTR(PPP->DATA,1,96),SUBSTR(PPP->DATA,
  97,96))(F(3),B(96),SKP(1),B(96));
RETURN:
END PRINTIT:
STMT LEV NT

0 PRINTP: PROC(P);
1 DCL P POINTER, A BIT(31), TRY FIXED BIN(31):
2 A=UNSPEC(P);
3 TRY=A;
4 PUT EDIT(TRY) (F(0));
5 RETURN;
6 END PRINTP:
International conference on very large data bases, 1977.

Kung, A. and Tsichritzis, D., Multiple view support
within the ANSI/SPARC framework. Proceedings
CISR report. 5, Sloan school of management, MIT, 1980.

Functional hierarchy of the INFORMIX database computer.
Hsu, Hsu, M., A preliminary architectural design for the
Sloan school of management, MIT, June 1980.

Huff, Hutt, S. L., A systematic methodology for designing the

Haleo, Haleo, D., Kerr, D. S., Madnick, S. E., Computer
addison-wesley publishing co, 1977.

Date: date, c.j., An introduction to database systems.
1979
capture more meaning, ACM TODS, vol 4, no. 4, december
codd: codd, E.F., Extending database relational model to
very large data bases, 1980.

Performance, proceedings international conference on
concurrency control on distributed database system
Badal: Badal, D. Z., The analysis of the effects of
1975
report, February 1975
ANSI/SPARC: ANSI/SPARC study group on DBMS intern

M. et al., System R , ACM TODS, Vol 1,

To DBMS design and Evaluation. CISR-99, Sloan school of
Approach to the design of complex systems: Application
Andreu: Andreu, R. and Madnick, S. E. A systematic