DATA BASE DESIGN PRINCIPLES APPLIED TO A NETWORK MODEL

BY

MARK A. COSTELLO

B.S., Pittsburg State University, 1979

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

Major Professor

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

The author expresses his sincere appreciation to Professor Elizabeth A. Unger for her patience, guidance, and encouragement during this project. The author also expresses his sincere appreciation to Cindy Norman for her faithful assistance during this project.

Chapter 1

INTRODUCTION

1.1 The Problem

1.1.1 Introduction of Complex Data Base Management Software

The data base approach for the storage, maintenance, and retrieval of data is becoming a popular approach to the problem of organizing data using a computer. A 1979 publication by R.L. Nolan indicated that over 60 percent of the total number of data processing installations are committed to the data base approach (NOLA79) for the organization, storage, maintenance, and retrieval of data. The traditional and data base approaches to data management differ in that traditional data management methods organize data in support of a specific service of an organization, introducing unnecessary replication of data, whereas, the data base approach maintains data as a resource for the entire organization offering the opportunities for reduction of data redundancy and the reduction of maintenance, integration, and retrieval costs. Sophisticated data base management software, known as data base management systems (DBMSs), have

been written to support the data base. Although several different classifications of the logical data organization for data bases exist (e.g., hierarchical, network, and relational), the majority of functioning DBMSs are based on the network model (OLLE78). Network DBMSs are considered efficient to operate if the data base is organized properly, but data bases organized according to the network model are complicated to design. This complexity demands a broad spectrum of expertise from those responsible for data base design. Thus, tools to automate or partially automate this design process would be very helpful to those faced with the task of automating the design of a data base structure according to the network model.

1.1.2 Data Base Design Expertise

The central responsibility for data base design is the data base administrator (DBA). As indicated by a well experienced data base consultant and frequently published author, Ronald Ross, the expectations of the DBA are overwhelming.

> "The typical corporate DBA finds himself trapped on every side by a set of expectations totally out of line with reality. On the one hand, DP management expects that the DBA be thoroughly proficient in the technology of the corporate DBMS, so that database systems run smoothly and efficiently. . . . Yet at the same time, application and business analysts (as well as end users) demand that the DBA be so knowledgeable about detailed business operations that all the endless subtleties of "doing business" can be taken into account when creating the database design. . . . still more expectations crowd in on the DBA. Realizing that a crucial measure of any applications's success

> (whether database or not) is the ability to expand and evolve over time--and that maintenance overhead is a primary obstacle to this goal--the DBA is expected to create database designs having inherent stability. . . . Translated, this means that the DBA must also be an expert on the practical application of normalization for the design of large-scale data resources. . . . Finally, there is one last source of expectations. For those corporations that have implemented a data administration function, the DBA is expected to adhere to, and promulgate, the data element (and other) standards which that group produces." (ROSS82)

In the likely event that the DBA cannot completely fulfill all the above expectations, a number of problems may arise. According to Jan Rumberger of TSI, International, the problems are not new ones. He states,

> "The list of problems is a familiar one: DBMSs being used as an access method, DAs unable to get a real handle on element standardization, poor database designs, high subsequent maintenance costs. The results is increasing user and management disenchantment with the costs and promised-but-not-delivered benefits of data base technology." (ROSS82)

These problems can have significant impacts on the productivity of an organization. It is imperative to develop tools which reduce the expertise which must be available to an organization designing a data base and to move the design process from the art that it is currently to a scientific methodologically based process.

1.1.3 Data Base Development Aids and their Shortcomings

Three of the most widely used data base development aids currently available to data base designers are 1) the data dictionary, 2) the normal forms introduced through relational data base theory, and 3) data base development methodologies.

1.1.3.1 Data Dictionary

Due to the sizable volume to data necessary to design, implement, and maintain large computerized systems, the concept of the data dictionaries emerged. Although there does not exist a single definition for a data dictionary, the following is one that most would find acceptable.

> "A DD/D [Data Dictionary/Directory] is a centralized repository of information about data descriptions such as meaning, relationships to other data, responsibility, origin, usage and format. It is a basic tool within the data base environment that assists company management, data base administrators, systems analysts, and applications programmers in effectively planning, controlling, and evaluating the collection, storage and use of the data resource." (UHRO73)

Although data dictionaries are useful data base development aids, they are not without problems. Data dictionaries are designed primarily as maintenance aids and not data base development aids (ROSS82). Furthermore, in instances when data dictionaries are designed to function as pre-implementation utilities they ". . . are concerned with efficient

physical data storage and access. They don't address themselves to the problem of generating an appropriate, complete, accurate, and long lasting schema."(TSIC78)

1.1.3.2 Normalization

The theory of relational data base normalization, as introduced by Edgar Codd of IBM Corporation (CODD70), provides useful criteria for grouping data elements so as to reduce data redundancy and limit data maintenance anomalies. (A more precise definition of normalization is provided in Chapter 3.) Normalization has proven to be a useful design aid no matter which logical data organization model is being used.

The foremost problem with normalization is that traditional normalization techniques (i.e., manual techniques) are complex, cumbersome, and error-prone. Although a synthesis algorithm has been derived to meet the criteria of normalization (BERN76), in only one known case has this process been incorporated as a design aid in an automated data base development process (ROSS82).

1.1.3.3 Data Base Development Methodologies

To provide a more systematic approach to data base design the academic (TSIC78,MOLI79,CHEN77), consultant (YOUR79,MUER80), and corporate (ATRE80,MCEL79) communities have established data base design methodologies. A data

base design methodology eclectically derived from the above sources appears in Exhibit 1.1.

(1) Determine user requirements

(2) Document data element attributes and their inter-relationships

(3) Create normalized records

(4) Establish inter-record relationships

(5) Create a conceptual schema

(6) Transform the conceptual schema into a physical model

(7) Convert the physical model into data definition statements

Exhibit 1.1

While working as a consultant for Performance Development Corporation, Ronald Ross supported data base methodologies similar to the one in Exhibit 1.1 in corporations nationwide. Through his experience he found several major areas of concern.

> "First, the projects tended to become overwhelming simply by the sheer volume of documentation produced. The tasks of monitoring standards, administering revisions, and producing reports often became major stumbling blocks to success. . . . A second problem was that no hard and fast method existed for translating user requirements ("services") into a stable nonredundant database architecture. Almost inevitably, it seemed, headstrong and performance-oriented DBAs ended up doing their own physical designs--which may or may not have either matched the users' requirements or constituted a reliable model of his business." (ROSS82)

These concerns are addressed directly by the research described in this thesis.

1.2 Current Relevant Research

In a search of the literature only one automated integrated approach to data base design was discovered. The system, still under development, is known as FACETS (ROSS82). Ronald Ross, formerly of Performance Development Corporation, was seriously considering automation of the data base design process as early as the mid-1970s. By 1981, Ross had made marketable the initial system components of what will likely become an integrated system for automation of the conceptual data base design process. FACETS is described by the manager of Data Management Products for TSI, International, Jan Rumberger, as "representing a major evolutionary step . . . that will ultimately encourage better--and more creative--results with database development than ever before possible." (ROSS82)

The information required for the use of FACETS can be classified in three categories.

> "Defining the business context of the data base project, primarily (but not exclusively) to answer 'strategic' questions of scoping, planning and higher-level data organization. . . .
>
> Defining the individual requirements that the future database project must satisfy. These requirements are called 'services,' and roughly equate to inputs and outputs the end-users need within the new system.
>
> Developing a logical architecture for the database system. In contrast to the 'local requirements' of the previous area, the logical database architecture represents a 'global' statement about integrated data organization." (ROSS82)

In order to create and maintain the above data, FACETS supports three major processes:

1) The entry and inspection of "Service Local Views" to represent user requirements and obtain dependencies about data elements.

2) The "Relational Generator" module to provide an automation of the normalization process based on information provided by "Service Local Views".

3) The "Database Project Dictionary" to enter, maintain, cross-reference, query, etc. information relevant to the data base development project. (MCCR83)

Thus the FACETS system terminates in the design process with the fifth step, create a conceptual schema (see Exhibit 1.1). FACETS is said to be generalized in that it is designed to collect all the data, data descriptions, and data relationships required in steps six and seven of Exhibit 1.1. Since steps six and seven require knowledge of the specific target DBMS, FACETS requests or creates data not required for a specific DBMS.

1.3 The Solution

In contrast with FACETS, the system described in this paper, known as DB_GEN, has been designed not only to be a useful design aid, but specifically to map the logical entities of a user's data base schema into IDMS data definition language statements (PERR77). That is to say, this tool, DB_GEN, aids the designer through all seven steps of the design process (see Exhibit 1.1), but aids in design steps six and seven only for one specific DBMS, IDMS.

The major objectives of this research are:

1) to orqanize and simplify the data base desiqn process through applied data base development aids and

2) to carry the design process beyond loqical data base design by transforming a conceptual view of the data base into CODASYL data description source statements (specifically those required by the Integrated Data Management System marketed by Culinet (PERR77)).

A generalized system such as FACETS may fail to request, or at least poorly represents, all necessary data for specific DBMSs. Thus, through selection of a specific DBMS the author feels the objectives of this research can be more clearly stated and addressed.

The research objectives of this implemention were met using a single menu-driven interactive PL1/IDMS proqram called DB_GEN. The data flow diagram, depicting the flow of control and data within DB_GEN is presented in Exhibit 1.2.

CONTROL AND DATA FLOW DIAGRAM FOR DB-GEN

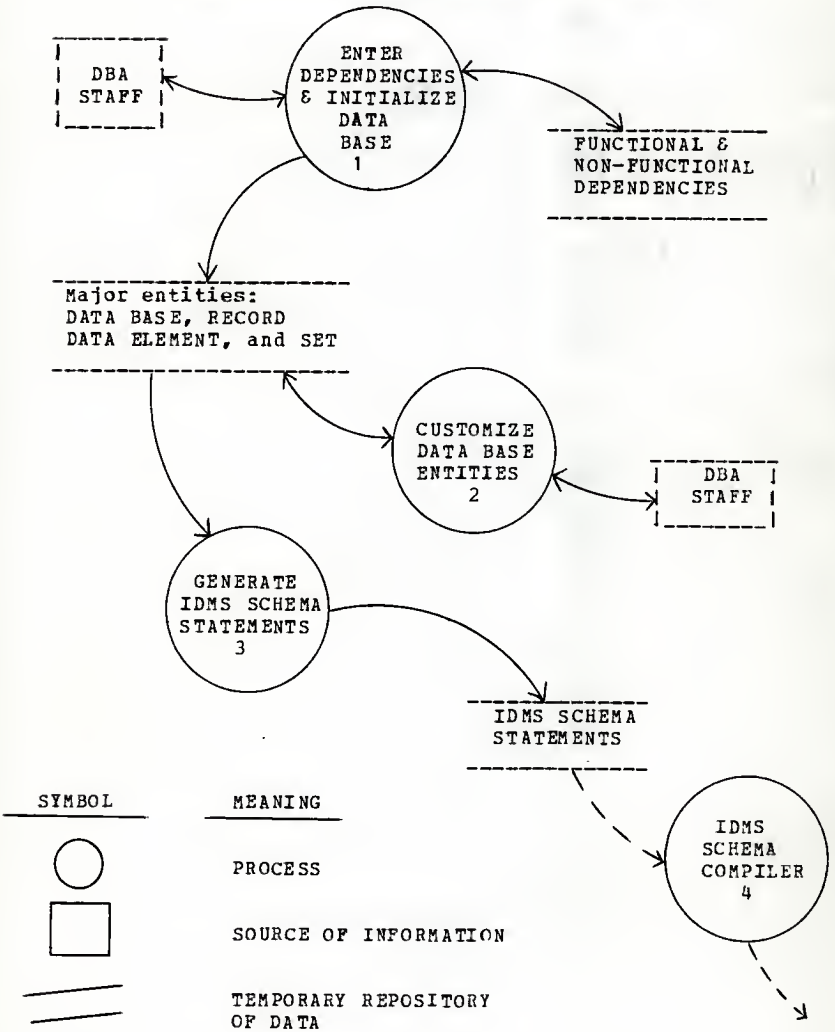


Exhibit 1.2

As illustrated by the control and data flow diagram in Exhibit 1.2, the initial inputs into the system are functional and nonfunctional dependencies. A functional dependency (FD) exists when a given data element value is uniquely identified by the value (or set of values) of one or more other data elements. For example, a functional dependency exists between data elements X and Y if for any value of X there is at most one value of Y (written 'X > Y'). A nonfunctional dependency (NFD) exists when a given data element value is not uniquely identified by the value (or set of values) of one or more other data elements. For example, a nonfunctional dependency exists between data elements R and S if for any value of R there is zero, one, or more values of S (written "R >> S"). Once functional and nonfunctional dependencies are collected, the data base about the data base schema (i.e., a meta data base), is initialized (see process one of Exhibit 1.2). All meta data base entities (DATA BASE, DATA ELEMENT, RECORD, and SET) are customized in process two of Exhibit 1.2 to meet user requirements with respect to IDMS specifications. DB_GEN was designed under the assumption that the data base designer (and the reader of this report) has a good working knowledge of the IDMS generalized data base management system (PERR77). Following customization of data base entities, process three of the control and data flow diagram transforms the data base entities into IDMS data definition language statements acceptable to the IDMS schema compiler shown in process four.

1.4 Guide to this Paper

In order to familiarize the reader with menu perusals, Chapter 2, Design Overview, illustrates the user interaction formats which are consistent throughout the entire implementation. Additional menus indicate how entities that are stored in the meta data base are updated and what types of interactions are made when conflicts between entity attributes exist. Once menu perusals are introduced, an outline of primary activities is used as an overview of the various system features.

A useful generation of entities requires a clear understanding of user needs. Chapter 3, Bernstein's Algorithm and User Requirements, discusses the transformation of user requirements into functional and nonfunctional dependencies and describes how Bernstein's algorithm uses these dependencies to create a relational schema.

Chapter 4, Data Base Initialization and Interpretation, provides a description of how major meta data base entities are established from data element dependencies. Functional dependencies, through the use of Bernstein's algorithm, generate data elements and records. In addition to the generation of data elements and records, functional dependencies also establish 1-to-1 inter-record relationships. Nonfunctional dependencies are modified to represent owner and member records and once modified the respective 1-to-many relationships are generated.

Chapter 5, Data Base Customization, describes the interaction between the data base designer and the system in order to resolve issues about the data in the meta data base that are mechanically unresolvable. Special emphasis is made to default as many DBMS software parameters as possible to minimize the need for customization. In situations where the efficiency of the data base operation is the only concern, parameter selection is made without the ability for the DBA to make modifications. However, the DBA is given the ability to modify all parameters that involve a correct representation of user needs. All modifications are carefully scrutinized for potential conflicts and if such conflicts are found the system responds with helpful advice. To insure the integrity of the meta data base, all propagational changes are carefully updated.

Chapter 6 describes the process of creating an operational schema. Once meta data base entities are customized a transformation must be made from the augmented conceptual view to the IDMS data definition statements. Although this module consists of mostly a reformatting task, CODASYL restrictions, relationship interpretations, and pointer assignments introduce interesting transformations and algorithms.

Chapter 7 summarizes the contributions of this research and concludes with a discussion about future supportive research.

Chapter 2

DESIGN OVERVIEW

The major subsystems of DB_GEN are highlighted in this chapter and covered in detail in later chapters. Additionally, a case study is introduced in this chapter and used throughout this paper as a tool to clarify the use of this system by the data base designer. Followed by the case study introduction, several sample menu traversals through DB_GEN are conducted to inform the reader of how this system is used.

2.1 Implementation Description

2.1.1 DB-GEN Conceptual Schema and Block Diagram

The prevalent tool of data base practitioners to understand and describe their data base requirements better is a conceptual schema (e.g., see Exhibit 2.1). Rectangles of a conceptual schema represent data base entities containing data elements that are bound by a unique identifier. The arrows between entities represent relationships required by user policy. A double-headed arrow indicates a 1-to-many relationship, whereas, a single-headed arrow represents a 1-to-1 relationship. Ironically, this implementation uses

the IDMS network DBMS to maintain data about user IDMS data base schemas. Therefore, the conceptual schema describing the data used by DB_GEN (see Exhibit 2.1) is a data base schema representing data base entities and the relationships between those entities (i.e., a meta data base schema). This meta data base is representative of what has been traditionally called a data dictionary.

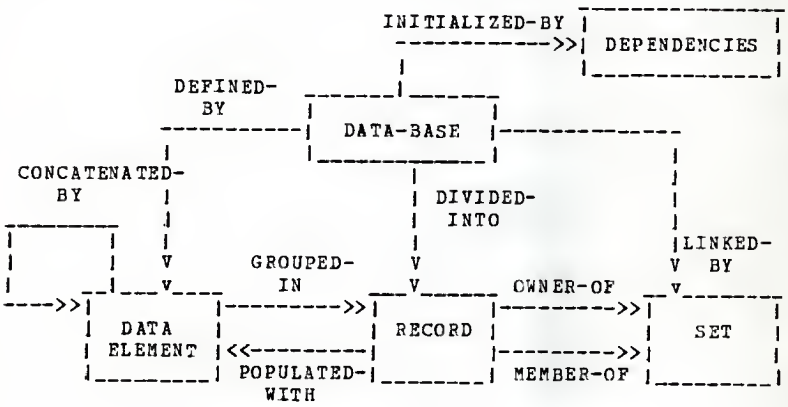


Exhibit 2.1

The block diagram of processes in Exhibit 2.2 illustrates the six major modules and significant sub-modules of DB_GEN. These modules are used by the data base designer during the development of a user's data base schema.

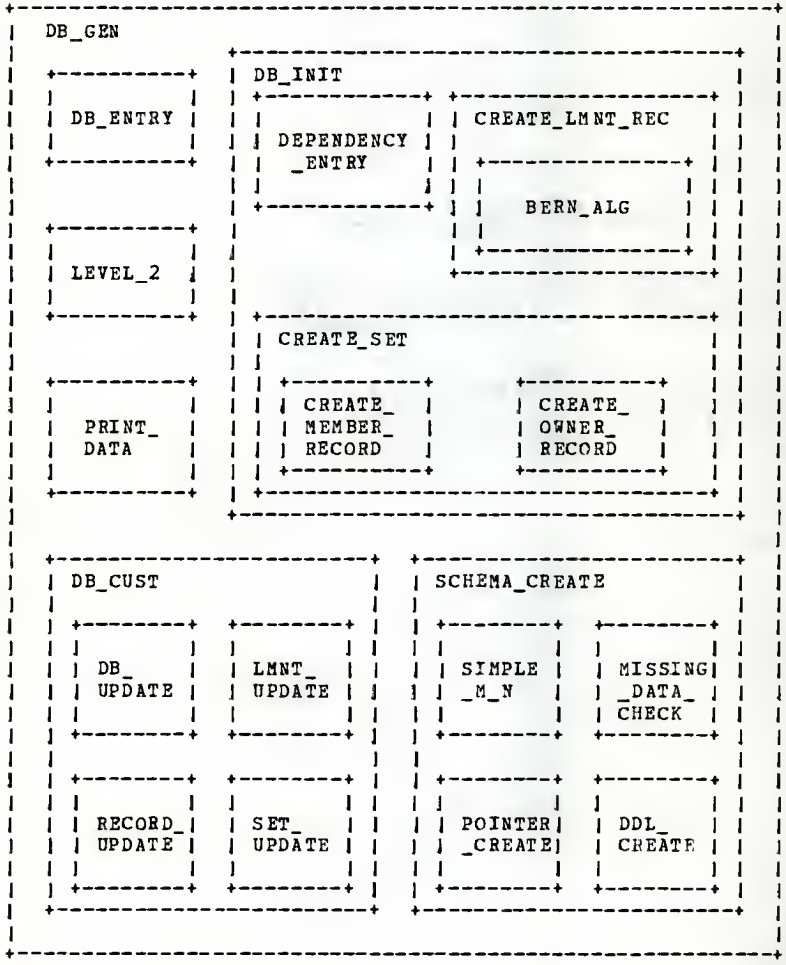


Exhibit 2.2

2.1.2 Input, Output, and Processes of DB_GEN

The data base designer enters DB_GEN in the DB_ENTRY module (see Exhibit 2.2) where a selection of a user's data base schema is made from those that are present in the DATA BASE entity (see Exhibit 2.1). (Note: Throughout the remainder of this section, when there is a reference to an "entity" or "relationship" occurrence, refer to Exhibit 2.1, and when there is a reference to a "module" occurrence, refer to Exhibit 2.2).

All services for development of the user's data base schema are then presented through a primary menu in the LEVEL_2 module.

The first service provided in the development of a user's data base schema is to initialize the major entities of that schema using the DB_INIT module. Functional dependencies (FDs) and nonfunctional dependencies (NFDs) representing user requirements are stored in the DEPENDENCIES entity and are linked to an instance of the data base entity by the INITIALIZED-BY relationship. Once functional and nonfunctional dependencies are present in the meta data base, DATA-ELEMENT, RECORD, and SET entities may be created. Furthermore, the association between DATA-ELEMENT and RECORD is established by the POPULATED-WITH and GROUPED-IN relationships. (The two relationships, POPULATED-WITH and GROUPED-IN, form an M-to-N relationship (i.e., a bi-direc-

tional 1-to-many relationship). This M-to-N structure, also known as a complex relationship, cannot be directly implemented in IDMS. Subsequent chapters discuss how complex relationships like this one are simplified.) In order to avoid redundant record names in the SET entity of Exhibit 2.1, the OWNER-OF and MEMBER-OF relationships establish a SET's owner and member records. (The two relationships, OWNER-OF and MEMBER-OF, constitute a multiple relationship. Multiple relationships allow different relationships with the same owner to point to different member record instances within the same member record.) All entities created during data base initialization, i.e., DATA-ELEMENT, RECORD, and SET entities, are linked respectively to the DATA BASE entity using DEFINED-BY, DIVIDED-INTO, and LINKED-BY relationships.

Following the creation of these entities, customization of record attributes are conducted using the DB_CUST module. The customization module has the capabilities to modify all entities, entity attributes, and relationships created through data base initialization. In addition to these capabilities, sub-elements of a data element can be established using the CONCATENATED-BY relationship. (This situation, where a single data element points to one or more sub-elements, is referred to as an Lii relationship--a link (L) with the same owner and member record (i).)

The final step in the data base development process is to create the data definition statements representing the user's data base from information stored in the meta data base. In situations where complex relationships exist, this module, SCHEMA-CREATE, may create new RECORD and SET instances. Otherwise, this module only retrieves data from the meta data base for reformatting into compilable IDMS data definition language statements.

Any time during the data base development process the PRINT-DATA module may be used to print or display information about any or all data base entities in the meta data base.

2.2 Case Study Introduction

In order to aid the reader of this work a simple case study of the Wampum Brokerage company is now introduced. Although this case study is restricted to two outputs, efforts have been made to ensure the inclusion of some of the most difficult data base design and implementation problems (e.g., M-to-N relationships, data relevant to two or more existing records (i.e., intersection data), 1-to-1 relationships with no inverse, and the potential for second and third normal form violations). Requirements for this case study are altered intermittently in order to emphasize specific situations.

Exhibits 2.3 and 2.4 are examples of two Wampum Brokerage output requirements. Although this case study is limited in scope, existing brokerage houses would have access to similar displays/documents. The "stock activity" display in Exhibit 2.3 would be useful when news broke on any specific stock (e.g., a stock split, extreme quarterly earnings variation, merger or take over, bankruptcy, etc.). The broker would want to have the latest changes in stock price quotes and volume as well as the degree to which clientele are affected and perhaps a list of those clients that are affected most. This display would also be used when clients make queries concerning a specific stock and for the broker to receive updates on how successful previous recommendations concerning a stock have been. The "client activity" display (see Exhibit 2.4) furnishes the broker with necessary client demographic information showing both composite and detailed data. This would be useful when providing financial advice to a client.

*** STOCK ACTIVITY ***

STOCK NAME: INTERNATIONAL BUSINESS MACHINES
STOCK ABBREVIATION: IBM
MOST RECENT QUOTE: 140 1/4
MOST RECENT VOLUME: 1,824,000
LAST UPDATED: 12-01-83 10:24:13
TOTAL CLIENTAL STOCKS: 650
GAIN/LOSS PERCENTAGE WHEN RECOMMENDED: +13.37

CLIENT ID	CLIENT NAME	QUANTITY	PERCENT OF TOTAL INVESTMENT	PHONE
021	JOE SLY	200	100	555-8000
613	TYCOON MARY	150	12	555-6350
419	BAGS MOONIE	100	2	555-6354
414	BULL FRANCIS	100	80	555-0549
812	T HOWELL III	100	50	555-2152

Exhibit 2.3

*** CLIENT ACTIVITY ***

CLIENT NAME: BAGS MOONIE
CLIENT ID: 411
EMPLOYER: KANSAS STATE UNIVERSITY
ANNUAL SALARY: 20000
PHONE: 555-6354
TOTAL INVESTMENT: 16,200.00
CURRENT WORTH: 18,525.00
GAIN/LOSS PERCENTAGE: +14.35

STOCK ABRV	TRANS DATE	BROKER RECMND	PRICE	NO. PUR	AMOUNT	CURRENT WORTH	% GAIN /LOSS
IBM	080180	YES	90.00	50	4500.00	7012.50	+55.83
	100283	NO	150.00	50	7500.00	7012.50	- 6.50
				100	12000.00	14025.00	+16.875
MCI	060182	YES	42.00	100	4200.00	6000.00	+42.86
				100	4200.00	6000.00	+42.86

Exhibit 2.4

2.3 Using DB_GEN

2.3.1 Formats for DB_GEN Interaction

The system has been designed to be user-friendly. It provides information necessary to make user decisions and aid the user in inputting information. User responses are consistent throughout the entire system--a menu number and, optionally, a menu entry. Any time a blank entry is given in response to a menu display, the system responds with the general instructions in Exhibit 2.5.

```
*** GENERAL INSTRUCTIONS ***

TWO FORMATS CAN BE USED --

FORMAT 1:    <MENU_NUMBER>

==> THIS WILL PROVIDE DETAILED INSTRUCTIONS FOR
    ENTERING THE RESPECTIVE INFORMATION

FORMAT 2:     <MENU_NUMBER> <MENU_ENTRY>

==> THE DETAIL INSTRUCTION STEP IS SKIPPED BY
    ADDING THE MENU ENTRY   E.G., 1  STOCK_DATA_BASE


PRESS enter to continue
```

Exhibit 2.5

As a user becomes familiar with the system, intermediate menus can be skipped by giving the appropriate menu number and menu entry. A novice user of the system will find that the system provides adequate guidance for its use.

2.3.2 Data Base Entry--DB_ENTRY

Upon entering DB_GEN, the user must make a selection from the list of existing user data base schema names or create a new user data base schema (see Exhibit 2.6). Exhibit 2.6 illustrates the selection of menu number one; the response of this request is shown in Exhibit 2.7A.

```
** DATA BASE ENTRY **

1)  CREATE DATA BASE
2)  OPTION-DB
3)  BOND-DB
X)  EXIT

MAKE SELECTION ===>
1
```

Exhibit 2.6

```
** CREATE NEW DATA BASE **

1)  DATA BASE NAME:
2)  DATA BASE ADMINISTRATOR:
X)  EXIT

===>
1stock  data base
```

Exhibit 2.7A

```
** CREATE NEW DATA BASE **

1)  DATA BASE NAME:  STOCK-DATA-BASE
2)  DATA BASE ADMINISTRATOR:
X)  EXIT

===>
```

Exhibit 2.7B

The name of the user's data base schema is supplied in Exhibit 2.7A by entering menu number one followed by the menu entry "stock data base." Note that data base name could have been supplied from Exhibit 2.6 and the display in 2.7A would have been bypassed. DB_GEN is already preparing for a clean schema compile by altering the data base name entered in Exhibit 2.7A into an acceptable IDMS data base name by removing blanks and substituting hyphens (see Exhibit 2.7B). These flexibilities and interpretations are consistent but limited. One must be careful to provide user friendliness as well as data integrity.

2.3.3 Update Considerations

Care has also been taken to allow for changes in all entity names (i.e., data base, data element, record, and set names). Exhibit 2.8 illustrates a change of the data base name.

```
        ** CREATE NEW DATA BASE **

   1)  DATA BASE NAME:  STOCK-DATA-BASE
   2)  DATA BASE ADMINISTRATOR:
   X)  EXIT

   ===>
   1 stock db
```

```
        *** CREATE NEW DATA BASE ***

   1)  DATA BASE NAME:  STOCK-DB
   2)  DATA BASE ADMINISTRATOR:
   X)  EXIT

   ===>
```

Exhibit 2.8

DB-GEN has been written to thoroughly, yet efficiently, search all areas where change is required and establish the necessary associations between entities where change propagates. For example, one data element may be in several records, be a candidate key for several other data elements, be part of a concatenated key, and/or be used as a sort or hashing field. All references to this entity attribute are properly modified by the system.

2.3.4 Primary Menu of Services--LEVEL 2

Upon entering a data base, a primary menu of services is provided (see Exhibit 2.9).

```
DATA-BASE:  STOCK_DB

        ** PRIMARY  MENU **

   1)  DATA BASE INITIALIZATION
   2)  DATA ELEMENT UPDATE
   3)  RECORD UPDATE
   4)  SET UPDATE
   5)  DATA BASE UPDATE
   6)  PRINT DATA
   7)  CREATE SCHEMA
   X)  EXIT

   ==>
   2
```

Exhibit 2.9

The primary menu of services (see Exhibit 2.9) serves to guide the data base designer through the design process of a user's data base schema. This Exhibit is used to support an overview of this implementation.

2.3.4.1 Data Base Initialization--DB_INIT

Selection one (i.e., DB_INIT module) utilizes a very limited input of user requirements and Bernstein's Algorithm to perform a data base genesis. Functional dependencies are used to create data elements and normalized records. The third main ingredient, sets, is provided through nonfunctional dependencies. A close study of the application's re-

quirements described in terms of functional and non_functional dependencies provides a skeletal form upon which to build the user's data base schema.

2.3.4.2 Data Base Customization--DB_CUST

The next four selections from the primary menu are used to customize the entites of the user's data base schema. Actually, enough power exists in these four modules to create the data base schema without the use of the DB_INIT module. As data element, record, set, and data base modifications are made, there is constant monitoring for conflicts. Conflicts are answered with an error message followed by advice. For example, if one selects a non-existent record to be a set member, the error message and assistance of Exhibit 2.10 appears.

```
**********************************************************
RECORD SELECTED AS A SET MEMBER DOES NOT EXIST--USE MENU
**********************************************************
```

```
        ** SELECT SET MEMBER **

   1)  STOCK
   2)  CLIENT
   3)  STK_CLNT
   4)  STK_CLNT_TXN
   5)  EMPLR
   X)  EXIT
```

Exhibit 2.10

This list of records (see Exhibit 2.10) can then be used to select the set member; the assumption is that the user incorrectly spelled the record name.

2.3.4.3 Printing of Data Base Information--PRINT_DATA

Selection six from the primary menu allows the user to view the meta data base data in a composite form either via display or hardcopy by using the menu in Exhibit 2.11.

```
*** PRINT DATA BASE INFORMATION ***

    1)  DATA ELEMENT DISPLAY
    2)  DATA ELEMENT HARDCOPY
    3)  RECORD DISPLAY
    4)  RECORD HARDCOPY
    5)  SET DISPLAY
    6)  SET HARDCOPY
    7)  ALL THE ABOVE
    X)  EXIT
```

Exhibit 2.11

2.3.4.4 Schema Creation--SCHEMA_CREATE

The final selection of the primary menu (see Exhibit 2.9) assures an initial check for missing data, displays minor errors, and then makes the conversion to an operational data base schema. Because the previous steps carefully scrutinize attributes of major entities, the user should not be faced with many changes in this final step.

2.3.5 Continued Use of DB_GEN

The data base schema is now ready to be generated for application use. Any future changes due to forgotten or changed user requirements can easily be made through the entity customization modules followed by a regeneration of the schema.

To provide a clearer understanding of the scope of this research, this chapter, Design Overview, highlighted the major services provided by DB_GEN with respect to the data and modules used to perform these services. The following chapter, Bernstein's Algorithm and User Requirements, begins the process of discussing the major parts of this research in detail.

Chapter 3

BERNSTEIN'S ALGORITHM AND USER REQUIREMENTS

In a 1976 publication, "Synthesizing Third Normal Form Relations from Functional Dependencies", Phillip Bernstein proved that a normalized relational schema can be synthesized "from a given set of functional relationships" (BERN76). However, it is not clear how a "given set of functional relationships" is derived. This chapter discusses the translation of user requirements into functional and nonfunctional dependencies and the use of these dependencies in Bernstein's algorithm.

3.1 Decomposition Method for Schema Normalization

Normalization is an integral part of nearly all data base design techniques (CODD70,CODD72,CODD79). Normalization involves a study of data that an organization uses in the relationships and dependencies among that data. The purpose of normalization is to aggregate data items into groups in which the group represents, if possible, only one entity of concern to the user. The output of normalization is a set of data table definitions which is organized to limit data

base redundancy, thus simplifying data maintenance services and enhancing data base integrity. The three steps of normalization may best be defined by one of Codd's colleagues, William Kent.

> "FIRST NORMAL FORM: A relation is in first normal form if none of its domains has elements which are themselves sets.
>
> SECOND NORMAL FORM: A relation in first normal form is in second normal form if every attribute in the complement of a candidate key is fully functionally dependent on that candidate key.
>
> THIRD NORMAL FORM: A relation in second normal form is in third normal form if every attribute in the complement of a candidate key is nontransitively dependent on that candidate key." (KENT73)

Codd's method involves starting with one relation and successfully decomposing it into smaller relations until all relations adhere to the above normalization criteria. It is possible in this decomposition approach to normalization to create a system which no longer represents all the FDs in the original system of FDs. When this occurs the user loses the possibility of referencing some of the information from the data base that is a part of the enterprise's data.

3.2 Synthesis Method for Schema Normalization

Phillip Bernstein's research has revealed that by using FDs, third normal form relations can be synthesized using an algorithm. The synthesis technique for use in normalization has been shown to be much more rigorous and consistent than the original decomposition method. All FDs provided by the designer are guaranteed to be represented in the schema generated by this synthesis method. The algorithm follows.

"ALGORITHM 1

1. (Eliminate extraneous attributes.) Let F be the given set of FDs. Eliminate extraneous attributes from the left side of each FD in F, producing the set G. An attribute is extraneous if its elimination does not alter the closure of the set of FDs.

2. (Find covering.) Find a nonredundant covering H of G.

3. (Partition.) Partition H into groups such that all of the FDs in each group have identical left sides.

4. (Merge equivalent keys.) For each pair of groups, say H1 and H2, with left sides X and Y, respectively, merge H1 and H2 together if there is a bijection X <--> Y in H+.

5. (Construct relations.) For each group, construct a relation consisting of all the attributes appearing in that group. Each set of attributes that appears on the left side of any FD in the group is a key of the relation. (Step 1 guarantees that no such set contains any extra attributes.) All keys found by this algorithm are called synthesized. The set of constructed relations constitutes a schema for the given set of FDs." (BERN76)

Exhibit 3.1

This synthesis technique is used by DB_GEN as the method for normalization of the data base records.

3.3 Translation of User Requirements into FDs

To understand better how Bernstein establishes FDs from known requirements, we re-examine Exhibits 2.3 and 2.4 of

the Wampum Brokerage case study. Output examples are quite useful but fall short of the rigor required to adequately describe the underlying policies of an organization. A starting point for describing entities is the introduction of a unique identifier in the form of a functional dependency for each data element appearing on the output examples. To provide further semantic value to the functional dependencies in Exhibit 3.3, consider the list of standard abbreviations (see Exhibit 3.2).

ABBREVIATION	= ABRV	PHONE	= PH
AMOUNT	= AMT	PRICE	= PRC
ANNUAL	= ANUL	QUANTITY	= QUAN
BROKER	= BRKR	QUOTE	= QUT
CLIENT	= CLNT	RECENT	= RCNT
CURRENT	= CUR	RECOMMEND	= RECMND
DATE	= DTE	SALARY	= SLRY
EMPLOYER	= EMPLR	STOCK	= STK
GAIN_LOSS	= GN_LS	TOTAL	= TOT
IDENTIFICATION	= ID	TRANSACTION	= TXN
INVESTMENT	= INVST	UPDATE	= UPDTE
NUMBER	= NUM	VOLUME	= VOL
PERCENTAGE	= PRCNT	WORTH	= WRTH

Exhibit 3.2

```
STK_NAME                    > STK_ABRV,RCNT_QUT,
                              RCNT_VOL,STK_LAST_UPDTE,
                              TOT_AMT_WHEN_UPDTE,
                              TOT_NUM_WHEN_UPDTE

STK_ABRV                    > STK_NAME

STK_ABRV,CLNT_ID            > CLNT_NAME,CLNT_STK_QUAN,
                              CLNT_STK_PRCNT_INVST,
                              EMPLR_PH

EMPLR_NAME                  > EMPLR_PH

CLNT_ID                     > CLNT_NAME,EMPLR_NAME,
                              ANUL_SLRY,EMPLR_PH

CLNT_ID,STK_ABRV,TXN_DTE    > BRKR_RECMND,CLNT_STK_TXN,
                              NUM_STK_CLNT_PUR
```

Exhibit 3.3

A cursory look at the Wampum Brokerage requirements in terms of FDs show the apparent loss of several data elements (e.g., gain/loss data). In many instances a data element appearing on a user requested output can be derived from other data elements. Therefore, it is not required to store derivable data element values in the user's data base. The data base administrator must weigh the cost of storing and maintaining data elements values that can be derived from other sources against the benefits of faster retrieval if the data elements values are resident in the user's data base. Of the ten derivable items on the output examples only two, CLNT_STK_QUAN and CLNT_STK_PRCNT_INVST, were selected for actual storage. However, the derivation of "GAIN/LOSS PERCENTAGE WHEN RECOMMENDED:" requires the introduction of data elements TOT_AMT_WHEN_RECMND and TOT_NUM_WHEN_RECMND. (See definitions below.)

```
GN_LS_PRCNT_WHEN_RECMND =
  (TOT_AMT_WHEN_RECMND - (TOT_NUM_WHEN_RECMND * RCNT_QUT))
  / (TOT_NUM_WHEN_RECMND * RCNT_QUT * 100)
```

This calculation results in the percent of change with respect to the most recently quoted stock amount. The following definitions should provide additional clarity.

TOT_AMT_WHEN_RECMND = Summation of AMT_CLNT_STK_TXN for each stock when BRKR_RECMND = "yes".

TOT_NUM_WHEN_RECMND = Summation of NUM_CLNT_STK_TXN for each stock when BRKR_RECMND = "yes".

If these data elements were not permanently stored in the data base, it would require traversals of all transactions

for the given stock performing comparisons and summations as described above.

It is important to have a clear statement of user requirements. Without a clear statement of user requirements, an inappropriate transformation of user requirements into functional dependencies is likely to cause data base integrity problems and maintenance anomalies to surface during the use of the data base.

3.4 Synthesize a Normalized Schema for Wampum Brokerage

Instead of a complete manual execution of Bernstein's algorithm using the functional dependencies in Exhibit 3.3, useful instances from the case study are provided for each step of the algorithm. Simplification of the compound right sides of the given functional dependencies is done prior to the initial step of the synthesis algorithm (see Exhibit 3.4A) because an FD of the form X > A,B can always be rewritten as X > A and X > B.

```
A)  STK_NAME                      > STK_ABRV
B)  STK_NAME                      > RCNT_QUT
C)  STK_NAME                      > RCNT_VOL
D)  STK_NAME                      > STK_LAST_UPDTE
E)  STK_NAME                      > TOT_AMT_WHEN_RECMND
F)  STK_NAME                      > TOT_NUM_WHEN_RECMND
G)  STK_ABRV                      > STK_NAME
H)  STK_ABRV,CLNT_ID              > CLNT_NAME
I)  STK_ABRV,CLNT_ID              > CLNT_STK_QUAN
J)  STK_ABRV,CLNT_ID              > CLNT_STK_PRCNT_INVST
K)  STK_ABRV,CLNT_ID              > EMPLR_PH
L)  EMPLR_NAME                    > EMPLR_PH
M)  CLNT_ID                       > CLNT_NAME
N)  CLNT_ID                       > EMPLR_NAME
O)  CLNT_ID                       > ANUL_SLRY
P)  CLNT_ID                       > EMPLR_PH
Q)  CLNT_ID,STK_ABRV,TXN_DTE      > BRKR_RECMND
R)  CLNT_ID,STK_ABRV,TXN_DTE      > CLNT_STK_TXN_PRC
S)  CLNT_ID,STK_ABRV,TXN_DTE      > NUM_STK_CLNT_PUR
```

Exhibit 3.4A

The first step of Bernstein's algorithm, eliminate extraneous attributes, identifies STK_ABRV as an extraneous attribute in the FD H, STK_ABRV,CLNT_ID > CLNT_NAME (see Exhibit 3.4B). As shown in Exhibit 3.4B, the closure (graphically indicated by a plus sign) of CLNT_ID includes CLNT_NAME which indicates that CLNT_ID alone functionally determines CLNT_NAME. The algorithm states that any extraneous attributes on the left side of an FD must be eliminated as illustrated by the removal of STK_ABRV for FD H in the final statement of Exhibit 3.4B.

1) Eliminate extraneous attributes.

H) STK_ABRV,CLNT_ID > CLNT_NAME

Find closure of CLNT_ID:

CLNT_ID+ = CLNT_ID,CLNT_NAME,EMPLR_NAME,
ANUL_SLRY,EMPLR_PH

H) CLNT_ID > CLNT_NAME

Exhibit 3.4B

The second step of Bernstein's algorithm establishes a non-redundant covering from a list of FDs. This is accomplished by removing an FD from the list of FDs and finding the closure of the removed FD's left side using the remaining FDs. If the closure of the removed FD's left side contains the removed FD's right side then the FD is considered redundant. Exhibit 3.4C indicates that the closure of CLNT_ID, the left side of FD P, contains EMPLR_PH, the right side FD P, without the use of FD P. Thus, FD P is considered redundant and is removed from the list of FDs.

2) Find covering.

P) CLNT_ID > EMPLR_PH

Find closure of CLNT_ID without
functional dependency P:

CLNT_ID+ = CLNT_ID,CLNT_NAME,EMPLR_NAME,
ANUL_SLRY,EMPLR_PH

Remove FD P from list of FDs

Exhibit 3.4C

Step three of Bernstein's algorithm partitions the FDs into groups that have identical right sides. Exhibit 3.4D illustrates the creation of six partitions from the remaining FDs.

3) Partition.

STK_NAME	> STK_ABRV,RCNT_QUT, RCNT_VOL,STK_LAST_UPDTE, TOT_AMT_WHEN_RECMND, TOT_NUM_WHEN_RECMND
STK_ABRV	> STK_NAME
CLNT_ID	> CLNT_NAME,ANUL_SLRY
STK_ABRV,CLNT_ID	> CLNT_STK_QUAN, CLNT_STK_PRCNT_INVST
CLNT_ID,STK_ABRV,TXN_DTE	> BRKR_RECMND, CLNT_STK_TXN_PRC, NUM_STK_CLNT_PUR
EMPLR_NAME	> EMPLR_PH

Exhibit 3.4D

It is possible that the left sides of the partitioned groups of FDs in Exhibit 3.4D may be equivalent keys. If left sides of partitioned groups are equivalent keys then step 4 of Bernstein's algorithm requires that they be merged. Equivalent keys exist if the closures of the left sides of partioned groups are equal. Exhibit 3.4E indicates that the closures of STK_NAME and STK_ABRV are equal and the partitions containing these as left sides should be merged.

4) Merge equivalent keys.

Construct the closure of each of the left sides:

```
STK_NAME+ = STK_NAME,STK_ABRV,RCNT_QUT,RCNT_VOL,
            STK_LAST_UPDATE,TOT_AMT_WHEN_RECMND
            TOT_NUM_WHEN_RECMND

STK_ABRV+ = STK_ABRV,STK_NAME,RCNT_QUT,RCNT_VOL,
            STK_LAST_UPDATE,TOT_AMT_WHEN_RECMND
            TOT_NUM_WHEN_RECMND
```

Exhibit 3.4E

The final step of Bernstein's algorithm constructs relations from the merged partitions of the previous step by establishing relation identifiers, enclosing attributes in parentheses, and underlining key attributes as seen in Exhibit 3.4F.

5) Construct relations.

```
R1  (STK_NAME,STK_ABRV,RCNT_QUT,RCNT_VOL,
     STK_LAST_UPDTE,TOT_AMT_WHEN_RECMND,
     TOT_NUM_WHEN_RECMND)

R2 (CLNT_ID,CLNT_NAME,ANUL_SLRY,EMPLR_NAME)

R3 (STK_ABRV,CLNT_ID,CLNT_STK_QUAN,
    CLNT_STK_PRCNT_INVST)

R4 (CLNT_ID,STK_ABRV,TXN_DTE,BRKR_RECMND,
    CLNT_STK_TXN_PRC,NUM_STK_CLNT_PUR)

R5 (EMPLR_NAME,EMPLR_PH)
```

Exhibit 3.4F

Exhibit 3.4F illustrates the output of Bernstein's algorithm with respect to the relational model. The next section describes how the output of Bernstein's algorithm can be applied to the network model.

3.5 Correlation of Output from Bernstein's Algorithm and Network Entities

Each of the five relations in Exhibit 3.4F become an occurrence in the RECORD entity of the conceptual schema in Exhibit 2.1. Likewise, the data elements within the relations in Exhibit 3.4F becomes occurrences in the DATA ELEMENT record of the conceptual schema in Exhibit 2.1. The appropriate links between RECORD and DATA-ELEMENT records in the conceptual schema are established using the POPULATED-BY and GROUPED-IN sets. Subsequent chapters give the details of creating data elements and records from these relations. One can assume at this point, i.e., the completion of Bernstein's algorithm using FDs, that data elements and records for a user's data base requirements have been added to the meta data base that is being manipulated by DB_GEN (see Exhibit 2.1).

3.6 Nonfunctional Dependencies and Bernstein's Algorithm

The functional dependencies shown in Exhibit 3.4A result in the establishment of data elements and records for the network schema being created through the use of DB_GEN. But, as stated by Phillip Bernstein, "Clearly though, not every logical connection in the world is functional." (BERN76) In the section of Bernstein's 1976 paper discussing "The Synthesis Problem in Nonfunctional Relationships", Bernstein never clearly addressed how one determines the

need for a nonfunctional dependency. However, one must assume that a nonfunctional dependency exists when the policy of an organization, for which the data base is being designed, specifies that a specific value of an item, e.g., invoice number, determines a set of instances of another item, e.g., product name. In the case of the Wampum Brokerage case study, the STOCK ACTIVITY display (see Exhibit 2.3) indicates that a given instance of STK_NAME (e.g. International Business Machines), determines a set of instances of CLNT_NAME (e.g., Joe Sly, Tycoon Mary, Bags Moonie, Bull Francis, and T. Howell III). Because Codd's first normal form requires that none of the domains of a relation have elements which are themselves sets, the STK_NAME and CLNT_NAME data elements of Wampum Brokerage must be in separate relations. Therefore, in order to establish the necessary associations between instances of the relation that STK_NAME appears in (i.e., relation R1 of Exhibit 3.4F) and the instances of the relation that CLNT_NAME appears in (i.e., relation R2 of Exhibit 3.4F), a relationship must be established between relation R1 and relation R2 of Exhibit 3.4F. The implementation of such relationships requires some form of a data structure to link the various relations' instances, e.g., pointers are used in the network model and a matching of data element domains is used in the relational model. With respect to the relational schema, Bernstein claimed that ". . . all connections among attributes in a data base description can be represented by FDs. As long as

connections are functional there is of course no problem. Nonfunctional connections require special attention." (BERN76) Bernstein transformed each NFD into an FD by concatenating the right side of an NFD to the left side of the NFD and introduced a unique variable, theta, on the now empty right side of what was an NFD. For example, the NFD described previously for the Wampum Brokerage case study appears as follows:

STK_NAME >> CLNT_NAME

This NFD is transformed into an FD by moving CLNT_NAME to the left side with STK_NAME and placing a unique theta data element, THETA_2, on the right side (see below).

STK_NAME,CLNT_NAME > THETA_2

By adding this FD to the list of FDs in Exhibit 3.4A and applying Bernstein's synthesis algorithm, a new relation is established (see below).

R6 (STK_NAME,CLNT_NAME,THETA_2)

For each instance in the cross-product of the domains of STK_NAME and CLNT_NAME in the above relation, if THETA_2 has the value of "1" then a relationship exists between the respective instances of STK_NAME and CLNT_NAME and if THETA_2 has a value of "0" then a relationship does not exist between the respective instances of STK_NAME and CLNT_NAME.

A complete list of NFDs for the output examples of Wampum Brokerage case study (see Exhibits 2.3 and 2.4) appears in Exhibit 3.5 followed by a transformation of the NFDS into FDs as seen in Exhibit 3.6.

```
 1)  STK_NAME              >> CLNT_ID
 2)  STK_NAME              >> CLNT_NAME
 3)  STK_NAME              >> CLNT_STK_QUAN,
 4)  STK_NAME              >> CLNT_STK_PRCNT_INVST,
 5)  STK_NAME              >> EMPLR_PH
 6)  CLNT_NAME             >> STK_ABRV
 7)  CLNT_ID,STK_ABRV      >> TXN_DTE,
 8)  CLNT_ID,STK_ABRV      >> BRKR_RECMND,
 9)  CLNT_ID,STK_ABRV      >> CLNT_STK_TXN_PRC,
10)  CLNT_ID,STK_ABRV      >> NUM_STK_CLNT_PUR
```

Exhibit 3.5

```
 1)  STK_NAME,CLNT_ID                          > THETA_1
 2)  STK_NAME,CLNT_NAME                        > THETA_2
 3)  STK_NAME,CLNT_STK_QUAN                    > THETA_3
 4)  STK_NAME,CLNT_STK_PRCNT_INVST             > THETA_4
 5)  STK_NAME,EMPLR_PH                         > THETA_5
 6)  CLNT_NAME,STK_ABRV                        > THETA_6
 7)  CLNT_NAME,STK_ABRV,TRANS_DTE              > THETA_7
 8)  CLNT_NAME,STK_ABRV,BRKR_RECMND            > THETA_8
 9)  CLNT_NAME,STK_ABRV,CLNT_STK_TXN_DTE       > THETA_9
10)  CLNT_NAME,STK_ABRV,NUM_STK_CLNT_PUR       > THETA_10
```

Exhibit 3.6

As illustrated in Exhibit 3.7, several new relations have been created from the FDs in Exhibit 3.6 for the establishment of the relationships between the original set of relations in Exhibit 3.4F.

R1 (STK_NAME,STK_ABRV,RCNT_QUT,RCNT_VOL,
STK_LAST_UPDTE,TOT_AMT_WHEN_RECMND,
TOT_NUM_WHEN_RECMND)

R2 (CLNT_ID,CLNT_NAME,ANUL_SLRY,EMPLR_NAME)

R3 (STK_ABRV,CLNT_ID,CLNT_STK_QUAN,
CLNT_STK_PRCNT_INVST,THETA_1)

R4 (CLNT_ID,STK_ABRV,TXN_DTE,BRKR_RECMND,
CLNT_STK_TXN_PRC,NUM_STK_CLNT_PUR,
THETA_7)

R5 (EMPLR_NAME,EMPLR_PH)

R6 (STK_NAME,CLNT_NAME,THETA_2)

R7 (STK_NAME,CLNT_STK_QUAN,THETA_3)

R8 (STK_NAME,CLNT_STK_PRCNT_INVST,THETA_4)

R9 (STK_NAME,EMPLR_NAME,THETA_5)

R10 (CLNT_NAME,STK_ABRV,THETA_6)

R11 (CLNT_ID,STK_ABRV,BRKR_RECMND,THETA_8)

R12 (CLNT_ID,STK_ABRV,CLNT_STK_TXN_PRCNT,THETA_9)

R13 (CLNT_ID,STK_ABRV,NUM_STK_CLNT_PRC,THETA_10)

Exhibit 3.7

By introducing a unique theta for each NFD, one is given the flexibility to introduce multiple relationships. For instance, both relations R7 and R8 have been established as relationships between relations R1 and R3. Generally, multiple relationships are introduced when they are not needed if all the FDs generated from NFDs are introduced. Bernstein provided no insight into the outcome of this method. Thus, the data base designer must be called upon to make

some judgements as to which of the FDs represent the needs of the organization in terms of inherent data structures. In the case of Wampum Brokerage, a data base designer may decide that only THETA_1 in relation R3 and THETA_7 in relation R4 are required for the correct representation of user requirements in a relational model. In reference to Exhibit 3.7, the relationship established between relations R1 and R2 by relations R6 and R10 are represented by the THETA_7 data element in relation R3 because STK_ABRV and STK_NAME represent relation R1 and CLNT_ID and CLNT_NAME represent relation R2. Relations R7 and R8 represent the need for a relationship between relations R1 and R3. Each of the relations R1 and R3 contain STK_NAME. Thus, this relationship already exists without the need of relations R7 and R8. Similarly, relations R11, R12, and R13 represent a need for a relationship between relations R3 and R4. Each of the relations R3 and R4 contain CLNT_ID and STK_ABRV. Therefore, this relationship already exists without the need of relations R11, R12, and R13. Relation R9 indicates a need for a relationship between either relations R1 and R2 or R1 and R5. The reason that a choice exists concerning the establishment of the relationship requested by relation R9 is that EMPLR_NAME appears in both relations R2 and R5. Relation R3 already provides a relationship between relations R1 and R2. Thus, relation R9 is not necessary.

Although internally the network model addresses relationships in a significantly different manner than the relational model, the problems associated with transforming NFDs into sets (the network model term for relationship) necessary for the network model to meet user requirements remain. This transformation process is addressed in more detail in the following chapter, Data Base Initialization and Interpretation.

Chapter 4

DATA BASE INITIALIZATION AND INTERPRETATION

The data base initialization module, DB_INIT, makes use of Bernstein's algorithm to establish user required data base entities (i.e., data elements, records, and sets). However, Bernstein is creating a relational model and the system described in this research is producing a network model. Thus, manipulation of the output of Bernstein's normalization algorithm to transform it from a relational model to a network model must be accomplished. This chapter explains how dependencies among data elements are entered into DB_GEN and how those dependencies are modified by DB_GEN itself and by DB_GEN through interaction with the data base designer to produce the network records and sets required to meet the user's needs.

4.1 Establish Position in DB_GEN

Before pursuing an indepth look into the functions of DB_INIT, the DB_GEN response via menu traversals is given. On entering DB_GEN one must select an existing data base or create a new data base (see Exhibit 4.1). Selection four

generates the primary menu of services with respect to STOCK-DB (see Exhibit 4.2).

```
*** SELECT A DATA BASE ***

    1)  CREATE DATA BASE
    2)  OPTION-DB
    3)  BOND-DB
    4)  STOCK-DB
    X)  EXIT

    MAKE A SELECTION ===> 4
```

Exhibit 4.1

```
***    PRIMARY MENU    ***

    1)  DATA BASE INITIALIZATION
    2)  DATA ELEMENT UPDATE
    3)  RECORD UPDATE
    4)  SET UPDATE
    5)  DATA BASE UPDATE
    6)  PRINT DATA
    7)  SCHEMA CREATOR
    X)  EXIT

    MAKE A SELECTION ===> 1
```

Exhibit 4.2

4.2 Entry of Functional and Nonfunctional Dependencies

Exhibit 4.3 provides the data base administrator with several functions which may be performed and provides enough information to maintain clarity of position and operation for the user. The formats for entry of functional dependencies (FDs) and nonfunctional dependencies (NFDs) are consis-

tent with those of the previous chapter and with most of the literature. If the user were not familiar with the necessary formats, help is provided by selecting the appropriate number of a function and then failing to provide an entry (see Exhibit 4.3). The results of this help feature appear in Exhibit 4.4.

```
*** DATA BASE INITIALIZATION ***

    1)  CREATE FUNCTIONAL DEPENDENCY
    2)  CREATE NON-FUNCTIONAL DEPENDENCY
    3)  INITIALIZE DATA BASE
    DELETE DEPENDENCY . . .
      4)  STK_ABRV    >   STK_NAME
      5)  CLNT_ID     >   CLNT_NAME
      6)  STK_ABRV    >>  CLNT_ID
      .
      .
      .
    X)  EXIT

    MAKE A SELECTION ===> 2
```

Exhibit 4.3

```
*** CREATE NON-FUNCTIONAL DEPENDENCY ***

FORMAT:

    left_side    >    right_side

Where either side can be a concatenation
of several elements separated by commas.

MAKE ENTRY
 ==>  STK_ABRV,CLNT_ID >> TXN_DTE|BRKR_RECMND
```

Exhibit 4.4

All right sides of dependencies are simplified to a single attribute by the introduction of new dependencies (see numbers 7 and 8 of Exhibit 4.5).

```
*** DATA BASE INITIALIZATION ***

    1)  CREATE FUNCTIONAL DEPENDENCY
    2)  CREATE NON _FUNCTIONAL DEPENDENCY
    3)  INITIALIZE DATA BASE
    DELETE DEPENDENCY . .  .
      4)  STK_ABRV     >   STK_NAME
      5)  CLNT_ID      >   CLNT_NAME
      6)  STK_ABRV     >>  CLNT_ID
      7)  STK_ABRV,CLNT_ID  >>  TXN_DTE
      8)  STK_ABRV,CLNT_ID  >>  BRKR_RECMND
      .
      .
      .
    X)  EXIT

    MAKE A SELECTION ===> 1
```

Exhibit 4.5

With the exception of INITIALIZE DATA BASE the remaining selections of Exhibit 4.3 should be self explanatory.

4.3 Initialization of the User's Data Base Schema

INITIALIZE DATA BASE transforms FDs and NFDs into data base entities (i.e., data elements, records, and sets). While the user is creating the inputs representative of user requirements of the data base, INITIALIZE DATA BASE, in conjunction with the print options, should be used frequently as a design aid. However, data base customization should be restricted until user requirements stabilize. When a data base is reinitialized, a complete regeneration of entities

occurs and any previous entity customization is lost. Although DB_INIT provides a fast and easy way to initiate a sound schema, the customization modules in Chapter 5 continue to make major changes in requirements easy to incorporate.

4.3.1 Creation of Data Element and Record Entities

As discussed in Chapter 3, Bernstein's Algorithm and User Requirements, records and data elements are derived from functional dependencies using Bernstein's algorithm.

4.3.2 Creation of Set Entities

The remainder of this chapter: 1) contrasts the relational DBMS's relationship with the network DBMS's set, 2) discusses a necessary enhancement to Bernstein's algorithm to enable the NFD-to-set conversion, 3) explains the NFD-to-set conversion technique, and 4) discusses the creation of the set entity.

4.3.2.1 Contrast the Relational DBMS Relationship with the Network DBMS Set

Functional dependencies deal with intra-record relationships, whereas nonfunctional dependencies are concerned with relationships between records. Because a relational model uses only one structure (i.e., a relation), few interpretations need to be made by Bernstein's algorithm to distinguish relations from relationships. The following steps,

implicitly derived from Bernstein (BERN76), outline the method used to synthesize a relational schema:

1) Enter functional and nonfunctional dependencies
2) Convert NFDs to FDs
3) Execute Bernstein's algorithm

The relational model establishes relationships through a foreign key (for 1-to-1 relationships) or through a separate relation (for 1-to-n and m-to-n relationships) that contains attributes of the relations to be linked. In Exhibit 4.6 the relation entitled STK_CLNT allows users to ask the questions "Given a stock, who are all the clients that own that stock?" and "Given a client, what stocks are owned?"

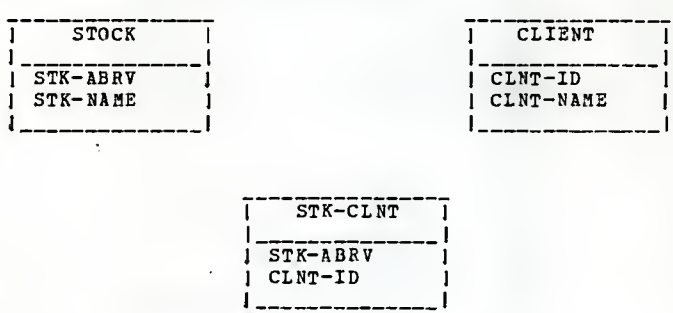


Exhibit 4.6

A network model uses a cyclic pointer structure to establish relationships. Exhibit 4.7 shows a network model representation of the relational model shown in Exhibit 4.6.

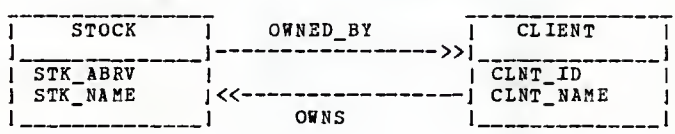


Exhibit 4.7

Network relationships can be described by their respective owner and member records. In general, the left side (LS) of an NFD identifies the owner record and the right side (RS) specifies a member record. Exhibit 4.8 presents two NFDs used to create the graphic schema in Exhibit 4.7.

```
S:  STK_ABRV  >>  CLNT_ID
T:  CLNT_ID   >>  STK_ABRV
```

Exhibit 4.8

When creating the relational model, the NFDs were converted to FDs and all the dependencies were used as input to Bernstein's algorithm. With a network model, one must know which records exist before NFDs can be interpreted. Therefore Bernstein's algorithm is run to completion with FDs only. Candidate keys of the recently created records now provide a means to interpret left and right sides of NFDs into owner and member records. An evaluation of NFDs in Exhibit 4.8 results in the selection of STK_ABRV to be a candidate key of STOCK and, therefore, the owner. Similarly, CLNT_ID is a candidate key of CLIENT and therefore the member. Requiring the user to enter only NFDs whose right and left sides evaluate to a previously derived record is far too restrictive. This is especially true when the user does not have any way of knowing what the candidate keys are when NFDs are entered. Suppose for example, an NFD, S, (see Exhibit 4.8) was changed to STK_ABRV >> CLNT_NAME where CLNT_NAME is not a candidate key. The user's meaning remains clear. CLNT_NAME appears in the CLIENT record as a

non-prime attribute so the same result is expected. This flexibility introduces several NFD interpretation problems. Before addressing these problems, it is necessary to study an enhancement of Bernstein's algorithm.

4.3.2.2 Check for Missing but Implied Dependencies

Converting an NFD to a set requires a minor modification to the output of Bernstein's algorithm. Consider an NFD, STK_ABRV >> EMPLR_PH. The NFD contains an attribute EMPLR_PH which is not a part of any key, i.e., it is a non-prime attribute. In this case a substitution of the respective prime attribute(s) must be made and a search for a matching candidate key performed. With the FDs given in Exhibit 4.9, the non-prime attribute EMPLR_PH appears in two records.

```
                  U)  CLNT_ID      >   EMPLR_PH
NFDs &            V)  EMPLR_NAME  >   EMPLR_PH
FDs in ==>        W)  STK_ABRV    >>  EMPLR_PH

RELATIONS         R1 (CLNT_ID,EMPLR-PH)
  OUT   ==>       R2 (EMPLR_NAME,EMPLR-PH)
```

Exhibit 4.9

In this case, the FDs do not provide enough information to create a schema that would be optimal in a "real world" environment and human intervention is required. The standard form of Bernstein's algorithm produces two relations shown in Exhibit 4.9. There is attribute redundancy as EMPLR_PH appears twice in the relations. The designer now has two

options for the substitution of prime attributes for non-prime attributes: 1) a substitution of the prime attributes CLNT_ID may be made producing STK_ABRV >> CLNT_ID or 2) the prime attribute EMPLR_NAME may be substituted producing STK_ABRV >> EMPLR_NAME. Any non-prime attributes that appear more than once in FDs after application of steps 1 and 2 of Bernstein's algorithm (see Exhibit 3.4C) indicate that additional semantic information is required of the user. Given the FDs of Exhibit 4.9, at least one of the two FDs in Exhibit 4.10 must be true.

```
P)  CLNT_ID      >   EMPLR_NAME
Q)  EMPLR_NAME   >   CLNT_ID
```

Exhibit 4.10

In this case, P is known to be true by the data base designer and a less redundant schema results (see Exhibit 4.11).

```
Relations      R1 (CLNT_ID)
  OUT  ==>     R2 (EMPLR_NAME,EMPLR-PH)
```

Exhibit 4.11

All this concern for removal of a single redundant attribute is questionable. However, another case illuminates the importance of this concept better (see Exhibit 4.12).

```
FDs in ==>     H)  STK_NAME     >   RCNT_QUT
               G)  STK_ABRV     >   RCNT_QUT

RELATIONS      R1 (STK_ABRV,RCNT_QUT)
  OUT  ==>     R2 (STK_NAME,RCNT_QUT)
```

Exhibit 4.12

In this case the data base designer may know that there is a bijection (i.e., STK_ABRV <---> STK_NAME) between the prime attributes STK_NAME and STK_ABRV. This allows the system to produce a single record schema (see Exhibit 4.13).

```
RELATIONS
  OUT  ==>    R1 (STK_NAME,STK_ABRV,RCNT-QUT)
```

Exhibit 4.13

The purpose of Bernstein's 1976 paper was ". . . to develop a provably sound and effective procedure for synthesizing relations satisfying Codd's third normal form from a given set of functional relationships. Also, the schema synthesized by our procedure is shown to contain a minimal number of relations."(BERN76) The FDs added by the data base designer as presented in the above Exhibits must be existing facts. By addressing these facts the data base designer can produce a less redundant schema.

If additional semantic information is required, the data base administrator is expected to respond interactively (see Exhibit 4.14).

```
*** SEMANTIC QUESTION ***

BASED ON GIVEN FDS ONE OF THE FOLLOWING MUST
BE TRUE. CLARIFY SEMANTICS BY SELECTION.

   1)  CLNT_ID     <---> EMPLR_NAME
   2)  EMPLR_NAME  ---> CLNT_ID
   3)  CLNT_ID     ---> EMPLR_NAME

MAKE A SELECTION ===> 3
```

Exhibit 4.14

The case statement in Exhibit 4.15 explains the action to be taken, based on the dependencies in Exhibit 4.9 and the display of Exhibit 4.14.

```
CASE  menu selection
  WHEN   1
    replace either FD U or FD V with both FDs 2 and 3
  WHEN   2
    replace FD V with FD 2
  WHEN   3
    replace FD U with FD 3
END CASE
```

Exhibit 4.15

Without this additional semantic information interpretations of owner and member records from NFDs would be purely arbitrary in some situations. The fact that one produces a less redundant and/or more minimal schema from this added information is a fortunate side effect.

4.3.2.3 Nonfunctional Dependency-to-Set Conversion

4.3.2.3.1 Convert Non-prime Attributes of an NFD to Prime Attributes

Although each non-prime attribute can now be identified by a set of candidate keys representing a single record, concern remains about a mindless substitution of candidate keys for non-primes. This substitution, as a rule, would still model the users' needs but could add unwanted relationships between entities. A subset of FDs and NFDs from the Wampum Brokerage case study is used to illustrate this potential problem (see Exhibit 4.16).

```
FD1:  STK_ABRV            >  STK_NAME
FD2:  CLNT_ID             >  CLNT_NAME
FD3:  STK_ABRV,CLNT_ID    >  STK_CLNT_QUAN
FD4:  CLNT_ID             >  EMPLR_NAME
FD5:  EMPLR_NAME          >  EMPLR_PH

NFD1: STK_ABRV            >> CLNT_NAME
NFD2: STK_ABRV            >> STK_CLNT_QUAN
NFD3: STK_ABRV            >> EMPLR_PH
```

Exhibit 4.16

Any time an FD exists where the RS attribute represents one record and LS attribute(s) represent another record, a 1-to-1 relationship exists. The existence of a 1-to-1 relationship in this situation is based on the definition of a functional dependency which appears in Chapter 1. Functional dependency four (FD4) in Exhibit 4.16 constitutes such an FD. This relationship, derived from FD4, is graphically shown by the single-headed arrow between RECORD2 and RECORD4 in Exhibit 4.17

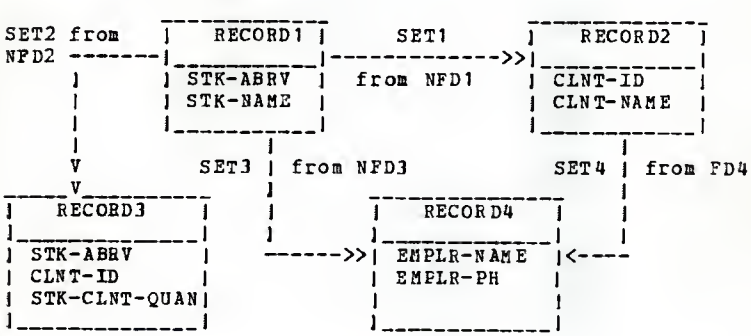


Exhibit 4.17

Apparent from the model shown in Exhibit 4.17 is the existence of a transitive path from RECORD1 to RECORD4 through RECORD2. Nonfunctional dependency NFD3 creates the unwanted transitivity via SET3. In this case, SET3 is useless for creating the STOCK ACTIVITY display and only serves to add complexity to the schema. Should a different user application dictate a path from RECORD1 to RECORD4, that relationship can still be recognized through RECORD2. Transitivity of this type can be eliminated when the RS attribute of an NFD is converted to a member record. The means by which this transitivity is resolved is formally introduced by the high level algorithm in Exhibit 4.18 and expounded upon through the example introduced in Exhibits 4.16 and 4.17.

```
ALGORITHM TO CONVERT NON-PRIME NFD ATTRIBUTES
TO PRIME ATTRIBUTES:

BEGIN ALGORITHM;
prime-substitute <--- non-prime attribute in an NFD;

DO WHILE prime-substitute exists on the RS of an FD and
         the prime-substitute has yet to be considered;

   LOCATE the FD where the prime-substitute attribute
   exists on a RS;

   prime-substitute <--- recently located FD's LS;

END LOOP;
END ALGORITHM;
```

Exhibit 4.18

The algorithm in Exhibit 4.18 removes transitivity from the the data base model by tracing existing FDs back to their left most identifier. This backtracking continues until no further backtracking can be done, or, in the case of a bi-

jection, the prime substitute becomes redundant with respect to previous substitutions. Without the condition checking for redundant substitutions an endless loop could result. In the example presented in Exhibit 4.16, existing FDs trace the non-prime attribute, EMPLR_PH, back to its left most identifier, CLNT_ID as shown below:

CLNT_ID > EMPLR_NAME > EMPLR_PH.

The RS of NFD3 in Exhibit 4.16 becomes CLNT_ID instead of EMPLR_NAME (i.e., NFD3 becomes STK_ABRV >> CLNT_ID) and the transitivity between records is removed. With respect to NFDs, consistency has been established for converting non-prime attributes to prime attributes. Note that the algorithm also converts non-prime attributes on the LS of an NFD to prime attributes. In order for owner records to be interpreted from an NFD's RS and a member record to be interpreted from an NFD's LS, all NFD attributes must be prime attributes.

4.3.2.3.2 Establish Member Record

Several questions concerning the conversion of these modified NFDs to owner and member records remain unanswered. The high level algorithm in Exhibit 4.19 is used to establish the procedure for converting the RS of an NFD to the expected member record. Following the algorithm, an example is provided to give a further understanding of this process.

```
ALGORITHM TO CONVERT AN NFD's RS ATTRIBUTE
TO A MEMBER RECORD:

BEGIN ALGORITHM;

/** INPUT ASSERTION -- all non-prime attributes of the
    given NFD have been converted to prime attributes
    using the algorithm presented in Exhibit 4.18 **/

/** The RS of the NFD is considered for the set member.**/

IF the NFD's RS is equivalent to an existing record's
   candidate key(s)

   LOCATE the record whose keys are equivalent to an
   existing NFD's RS;

   member-record <--- recently located record;

ELSE
   /** The intersection record created by concatenating  **/
   /** the LS and RS of the given NFD is considered       **/
   /** for the set member.                                **/

   CONCATENATE LS and RS attributes of the given NFD;

   IF the concatenated attributes are equivalent to an
      existing record's candidate key(s)

      LOCATE the record whose keys are equivalent to the
      concatenated attribute's;

      member-record <--- recently located record;

   ELSE

      /** There is not enough information to assure a     **/
      /** correct interpretation of the user's NFD as     **/
      /** presented.                                      **/

   ENDIF

END LOOP;
END ALGORITHM;
```

Exhibit 4.19

The algorithm in Exhibit 4.19 first checks to see if the attribute on the RS of the NFD evaluates to an existing record. If it does evaluate to an existing record then that

record is used as the member record. If the attribute on the RS of the NFD does not evaluate to an existing record then the concatenation of all NFD attributes is used to search for an intersection record. If an intersection record exists, it is used as the member record. The existence of an intersection record indicates an M-to-N relationship between the LS and RS attributes of the nonfunctional dependency. As is further clarified in Chapter 6, a complex relationship is simplified using an intersection record. Thus, the use of an intersection record as a member record is acceptable.

The following example uses the NFD, CLNT_ID,STK_ABRV >> TXN_DTE, with respect to the original set of records produced for the stock data base (see Exhibit 3.4F), to help clarify the establishment of the member record. The RS of the NFD does not clearly indicate a member record. TXN_DTE is a prime attribute of RECORD4 in Exhibit 3.4F, so no substitution is necessary. TXN_DTE is not found to be equivalent to any candidate key (i.e., the closure of TXN_DTE does not equal the closure of any candidate key for any record). TXN_DTE must exist as part of some key (it must be a prime attribute) and the only key that could assure semantic value would be the key created by the entire NFD (i.e., the intersection record). The closure of CLNT_ID,STK_ABRV,TXN_DTE is checked against closure of candidate keys for all records and is found to be equivalent to RECORD4 in Exhibit 3.4F and

therefore the member record. If RECORD4 did not exist the NFD, CLNT_ID,STK_ABRV >> TXN_DTE, would have been considered uninterpretable. To further clarify the search for a member record, refer to the graphic models in Exhibit 4.20.

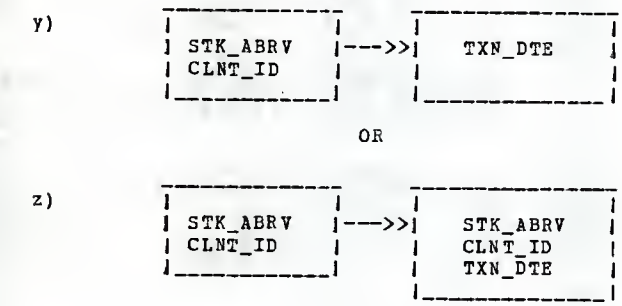


Exhibit 4.20

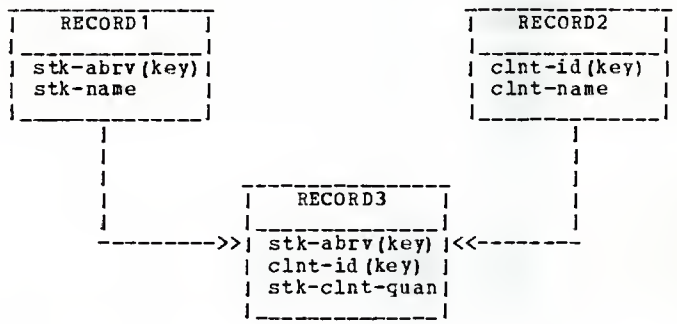
Option y is first considered, but TXN_DTE is not found to be a candidate key of any record in Exhibit 3.4F. STK_ABRV,CLNT_ID,TXN_DTE is a candidate key for RECORD4 and, therefore, represents the member record as shown by option z.

4.3.2.3.3 Establish Owner Record

The LS of an NFD, unlike the RS, does not necessarily identify a single record. The process to reduce an NFD's LS to an exact and minimal set of owners uses recursive tree traversals. Starting with the entire LS as a potential owner record, until all LSs are matched, recursive calls generate combinations of potential key attributes. If a wrong

path is taken, the process is backed up to where the initial combination was found and starts again at that point. Based on the input assertion that all LSs are prime attributes, this tree traversal algorithm should eventually find a set of combinations of the LS attributes whose closures are equal to the closures of a respective set of records. An exception case that deserves special attention exists when a set of LS combinations evaluate to an owner record that is the same record as the member. This type of relationship (i.e., an Lii) is not allowed by CODASYL and, when respective attribute combinations are found in the tree, they must be bypassed.

Consider the graphic depiction of STOCK_DB and NFD-G in Exhibit 4.21 as an example illustrating this procedure.



NFD-G: STK_ABRV,CLNT_ID >> STK_CLNT_QUAN

Exhibit 4.21

From NFD-G (see Exhibit 4.21) two sets are derived by the tree traversal algorithm. The LS of NFD-G, STK_ABRV,CLNT_ID, initially evaluates to the same record as the member record (i.e., RECORD3). Because an Lii relationship is not allowed, RECORD3 is bypassed and the tree traversal algorithm partitions the LS into STK_ABRV and CLNT_ID. STK_ABRV is a candidate key of RECORD1 and therefore an owner record. CLNT_ID is a candidate key of RECORD2 and also an owner record.

4.3.2.4 Create Set Entities in User's Data Base

Once owner and member records are determined, the 1:many sets are created. A pass of the FDs is made to determine if LS and RS closures are equivalent to candidate key closures of separate records. If so, a 1-to-1 relationship, such as SET5 of Exhibit 4.22, is created. The conceptual schema in Exhibit 4.22 shows the outcome of DB_INIT for the Wampum Brokerage case study in terms of its major entities.

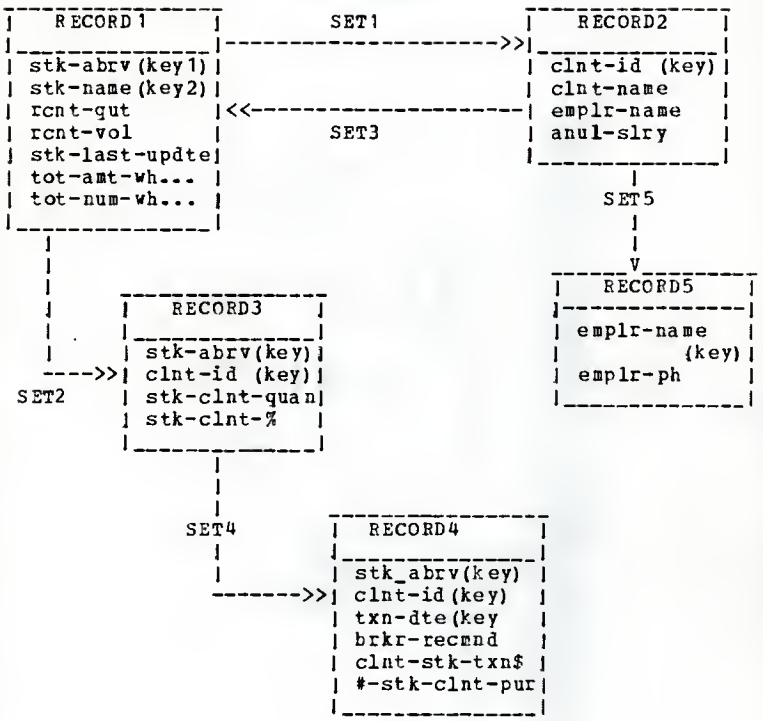


Exhibit 4.22

4.4 Status of User's Data Base Schema

At this point in the design process, the meta data base may contain some sets which still need to be modified to fit within the constraints of the CODASYL model. For instance, in Exhibit 4.22 the complex relationship between RECORD1 and RECORD2 (an N-to-M relationship) is not acceptable. The changes which still must be made are accomplished using the

next system module (DB_CUST). These modifications are postponed until the schema is to be created. The next chapter shows how these recently created entities can be customized to meet user requirements in a better way.

Chapter 5

DATA BASE CUSTOMIZATION

It is unlikely that all entities created by DB_INIT correctly and completely describe user needs. DB_INIT makes several assumptions that can lead to incorrect or inefficient code. For example, the defaulting of a data element's type is likely to be wrong as often as right. Therefore the need exists to modify entities to model the users' domain more correctly. The data base customization module (DB_CUST) is designed to lead the data base designer easily through the processes of adding, deleting and changing characteristics of the data elements, records, and sets created by DB_INIT. DB_CUST meets these requirements through the following services:

1) modification of all entities via a user-friendly, menu-driven, interactive system,

2) data base design assistance when the user is in doubt about available options, and

3) real-time conflict checks on all applicable entries.

The current assumed status, with respect to the data base design process, is that a data base has just been initialized and is ready for customization as shown in Exhibit 4.22.

5.1 Establish Position in DB-GEN

Selections 2-5 of of the primary menu in Exhibit 5.1 comprise the options available in DB_CUST.

```
|                                                |
|              ** PRIMARY  MENU **               |
|                                                |
|     1)  DATA BASE INITIALIZATION               |
|     2)  DATA ELEMENT UPDATE                    |
|     3)  RECORD UPDATE                          |
|     4)  SET UPDATE                             |
|     5)  DATA BASE UPDATE                       |
|     6)  PRINT DATA                             |
|     7)  CREATE SCHEMA                          |
|     X)  EXIT                                   |
|                                                |
|     ==>                                        |
|     2                                          |
|________________________________________________|
```

Exhibit 5.1

As described in earlier chapters, there can be an undetermined number of meta data bases under DB_GEN. Substantial effort has been made to keep each of the data base's entities separate so like-named entities of one data base can exist and be manipulated without affecting those of another data base. Although difficult to implement and costly in terms of input-output requests, meta data base separation is a necessity for integrity purposes.

5.2 Data Base Update

Through selection five, DATA BASE UPDATE (see Exhibit 5.1), the data base name can be changed with full confidence that all subordinate entities will remain intact. The data

base name is the only required attribute of the DATA BASE record. Available, but not required, is the ability to assign a data base administrator to each meta data base. Once a meta data base has been established, there is little need for the DATA BASE UPDATE module.

5.3 Data Element Update

Data elements form the basic building blocks for any data base. Evident from Chapter 4, Data Base Initialization and Interpretation, is the fact that data base generation is accomplished only from knowledge of how data elements relate to one another. Although much can be told about a data element, DB_GEN requests only the most basic data element information required for the schema generation (i.e., name, definition, type, and format).

The conceptual schema in Exhibit 2.1 illustrates how a data element participates with other data base entities. The CONCATENATED-BY relationship of DATA ELEMENT onto itself, allows for group-level data elements (a data element comprised of sub-elements). Group level data elements cannot be described in terms of functional dependencies, therefore, they must be described using the customization module. The GROUPED_IN and POPULATED_WITH relationships are initially created by DB_INIT and remain for use in this schema unless removed through use of the RECORD_UPDATE module. A selected few of the data elements are customized, avoiding an

exhaustive trace of data element customization for the entire organization which is not appropriate for this study. Selection two of the primary menu, DATA ELEMENT UPDATE (see Exhibit 5.1), lists all the data elements created by DB_INIT (see Exhibit 5.2).

```
** DATA ELEMENT UPDATE **

 1)  CREATE ELEMENT        16)  STK-NAME
 2)  ANUL-SLRY             17)  TOT-AMT-WHEN-REC
 3)  BRKR-RECMND           18)  TOT-NUM-WHEN-REC
 4)  CLNT-ID               19)  TXN-DTE
 5)  CLNT-NAME              X)  EXIT
 6)  CLNT-STK-PRCNT-I
 7)  CLNT-STK-QUAN
 8)  CLNT-STK-TXN-PRC
 9)  EMPLR-PH
10)  EMPLR-NAME
11)  NUM-STK-CLNT-PUR
12)  RCNT-QUT
13)  RCNT-VOL
14)  STK-ABRV
15)  STK-LAST-UPDTE

MAKE SELECTION ===>
6
```

Exhibit 5.2

Each of the four major data base entities use the same format for presentation of the entities of concern. The services available from Exhibit 5.2 are:

1) the creation of a new entity (selection 1),

2) the updating of existing entities (valid selection other than "1" or "X"), and

3) the removal of an existing entity (selection of an existing entity followed by "DELETE").

This menu system provides a comprehensive approach for entity maintenance; however, several shortcuts have been created to reduce menu traversals and therefore increase machine and manpower performance. For example, the data base designer can assign a data element name to a newly created data element by entering the data element name following menu number "1" in Exhibit 5.2.

Data element customization might begin by renaming selection six of Exhibit 5.2. Truncation by DB_INIT has left that name less than descriptive. Exhibits 5.3 and 5.4 track the name changing process.

```
** CHANGE OR DELETE DATA ELEMENT **

 1) ELEMENT NAME: CLNT-STK-PRCNT-I
 2) DEFINITION:
 3) TYPE: CHARACTER
 4) TOTAL SIZE: 010
 X) EXIT

 MAKE SELECTION ===>
 1CLNT-STK-INVST
```

Exhibit 5.3

```
** CHANGE OR DELETE DATA ELEMENT **

 1) ELEMENT NAME: CLNT-STK-INVST
 2) DEFINITION:
 3) TYPE: CHARACTER
 4) TOTAL SIZE: 010
 X) EXIT

 MAKE SELECTION ===>
 3numeric
```

Exhibit 5.4

Suppose the TYPE attribute of CLNT_STK_INVST needs to be modified from CHARACTER to NUMERIC and given an applicable format. Exhibits 5.4-5.6 illustrate the process the data base designer must use to accomplish this task. In Exhibit 5.4 a menu number of three followed by a menu entry of "numeric", indicates the TYPE attribute is to be modified from CHARACTER to NUMERIC as shown in Exhibit 5.5. Exhibit 5.5, inturn, changes the TOTAL SIZE attribute from ten to three (see Exhibit 5.6).

```
** CHANGE OR DELETE DATA ELEMENT **

 1) ELEMENT NAME: CLNT-STK-INVST
 2) DEFINITION:
 3) TYPE: NUMERIC
 4) TOTAL SIZE: 010
 5) FRACTION SIZE: 0
 X) EXIT

 MAKE SELECTION ===>
 4 3
```

Exhibit 5.5

```
** CHANGE OR DELETE DATA ELEMENT **

 1) ELEMENT NAME: CLNT-STK-INVST
 2) DEFINITION:
 3) TYPE: NUMERIC
 4) TOTAL SIZE: 003
 5) FRACTION SIZE: 0
 X) EXIT

 MAKE SELECTION ===>
 x
```

Exhibit 5.6

To illustrate other features, let us assume that a user has requested additional information which requires STK_LAST_UPDTE to be partitioned into STK_DAY_UPDTE and STK_TIME_UPDTE. These sub-elements have been created and appear in exhibits 5.7 and 5.8. The two new data elements must be created before STK_LAST_UPDTE could add these as sub-elements. If an attempt were made to divide STK_LAST_UPDTE into sub-elements prior to their creation, an error message would be displayed and a list of all valid data elements would be made available. The adding of sub-elements is accomplished by entering menu number four followed by the sub-element name (see Exhibit 5.9). If the data base designer is not sure of the sub-element to be added, the menu entry can be left blank and a list of existing data elements appears for selection.

```
    **  CREATE NEW DATA ELEMENT  **

 1) ELEMENT NAME: STK-TIME-UPDTE
 2) DEFINITION:
 3) TYPE: NUMERIC
 4) TOTAL SIZE: 006
 5) FRACTION SIZE: 0
 X) EXIT

 MAKE SELECTION ===>
```

Exhibit 5.7

```
    **  CREATE NEW DATA ELEMENT  **

 1) ELEMENT NAME: STK-DAY-UPDTE
 2) DEFINITION:
 3) TYPE: NUMERIC
 4) TOTAL SIZE: 006
 5) FRACTION SIZE: 0
 X) EXIT

 MAKE SELECTION ===>
```

Exhibit 5.8

```
     ** CHANGE OR DELETE DATA ELEMENT **

 1) ELEMENT NAME: STK-LAST-UPDTE
 2) DEFINITION:
 3) TYPE: CONCATENATED
 4) ADD SUB ELEMENT
 DELETE SUB ELEMENT . . .
   5) STK-DAY-UPDTE
   6) STK-TIME-UPDTE
 X) EXIT

 MAKE SELECTION ===>
```

Exhibit 5.9

5.4 Record Update

Similar to DATA ELEMENT, the RECORD entity is very tightly coupled within the DB_GEN data base (see Exhibit 2.1). Populated with data elements and linked to sets for which it is the owner and/or member, RECORD functions as an interface entity for the data base. Upon entry into the RECORD_UPDATE module, the need to clarify the generic record names is most apparent (see Exhibit 5.10). Changing RECORD attributes, as one might suspect, is similar to DATA ELEMENT attribute changes. Selection of a record displays the defaulted record attributes and, most importantly, the data elements linked to that record by DB_INIT (see Exhibit 5.11). By viewing the data elements within a record a more descriptive record name can likely be created (see Exhibit 5.12).

```
** RECORD UPDATE **

    1)  CREATE RECORD
    2)  RECORD1
    3)  RECORD2
    4)  RECORD3
    5)  RECORD4
    6)  RECORD5
    X)  EXIT

    MAKE SELECTION ===>
    3
```

Exhibit 5.10

```
** CHANGE OR DELETE RECORD **

1) RECORD NAME: RECORD3
2) RECORD LOCATION MODE: CALC
3) RECORD DUPLICATE OPTION: DN
4) RECORD CALC KEY OR VIA SET:
5) ADD DATA ELEMENT TO RECORD3
DELETE DATA ELEMENT . . .
    6) CLNT-STK-INVST
    7) CLNT-STK-QUAN
    8) CLNT-ID
    9) STK-ABRV
X) EXIT

MAKE SELECTION ===>
1 STK-CLNT
```

Exhibit 5.11

```
** RECORD UPDATE **

1)  CREATE RECORD
2)  CLIENT
3)  EMPLR
4)  STK-CLNT
5)  STK-CLNT-TXN
6)  STOCK
X)  EXIT


MAKE SELECTION ==>
```

Exhibit 5.12

The record attribute of most concern is the LOCATION MODE. Assignment of this attribute directly influences the remaining two attributes (i.e., the DUPLICATE OPTION and CALC KEY OR VIA SET). In Exhibit 5.13 STK_NAME has been chosen as the direct access key. (Note: If one wanted a concatenated CALC key, it would be necessary to create such an element using DATA ELEMENT UPDATE. Only one data element name is

accepted as a CALC key.) A change of the LOCATION MODE from CALC to VIA is made in Exhibit 5.13 and 5.14. This change forces suppression of the no longer applicable DUPLICATE OPTION. Exhibit 5.14 illustrates entry of an erroneous set for the VIA SET parameter (STK_CLNT must be a member record in the set chosen.) Selection of an invalid set name for VIA SET results in an error message followed by a help feature which lists the set in which STK-CLNT functions as a member record (see Exhibit 5.15).

```
** CHANGE OR DELETE RECORD **

1) RECORD NAME: STK-CLNT
2) RECORD LOCATION MODE: CALC
3) RECORD DUPLICATE OPTION: DN
4) RECORD CALC KEY OR VIA SET: STK-NAME
5) ADD DATA ELEMENT TO RECORD3
DELETE DATA ELEMENT . . .
    6) CLNT-STK-INVST
    7) CLNT-STK-QUAN
    8) CLNT-ID
    9) STK-ABRV
X) EXIT

MAKE SELECTION ===>
2 VIA
```

Exhibit 5.13

```
** CHANGE OR DELETE RECORD **

1) RECORD NAME: STK-CLNT
2) RECORD LOCATION MODE: VIA
3) RECORD CALC KEY OR VIA SET:
4) ADD DATA ELEMENT TO RECORD3
DELETE DATA ELEMENT . . .
    5) CLNT-STK-INVST
    6) CLNT-STK-QUAN
    7) CLNT-ID
    8) STK-ABRV
X) EXIT

MAKE SELECTION ===>
3 SET3
```

Exhibit 5.14

```
** SELECT VIA SET **

1)  SET2
X)  EXIT

==> 1
```

Exhibit 5.15

Additional RECORD UPDATE capabilities are shown by conducting a common data base optimization. To include a stand alone 1-to-1 relationship into its owner record is often a good tradeoff of increased redundancy for improved efficiency. This type of relationship exists between CLIENT and EMPLR in the Wampum Brokerage system (see Exhibit 4.22). The EMPLR record is deleted from the data base (see Exhibit 5.16) and EMPLR_PH is linked to the CLIENT record (see Exhibit 5.17 - 5.18). EMPLR_NAME previously existed in CLIENT

as a foreign key so it was not necessary to add EMPLR_NAME to CLIENT.

```
        ** RECORD UPDATE **

   1)  CREATE RECORD
   2)  CLIENT
   3)  EMPLR
   4)  STK-CLNT
   5)  STK-CLNT-TXN
   6)  STOCK
   X)  EXIT

   MAKE SELECTION ===>
   3  DELETE
```

Exhibit 5.16

```
      ** CHANGE OR DELETE RECORD **

    1) RECORD NAME: CLIENT
    2) RECORD LOCATION MODE: CALC
    3) RECORD DUPLICATE OPTION: DN
    4) RECORD CALC KEY OR VIA SET: CLNT-NAME
    5) ADD DATA ELEMENT TO CLIENT
    DELETE DATA ELEMENT . . .
        6) ANUL-SLRY
        7) EMPLR-NAME
        8) CLNT-NAME
        9) CLNT-ID
    X) EXIT

    MAKE SELECTION ===>
    5 EMPLR-PH
```

Exhibit 5.17

```
** CHANGE OR DELETE RECORD **

1) RECORD NAME: CLIENT
2) RECORD LOCATION MODE: CALC
3) RECORD DUPLICATE OPTION: DN
4) RECORD CALC KEY OR VIA SET: CLNT-NAME
5) ADD DATA ELEMENT TO CLIENT
DELETE DATA ELEMENT . . .
    6) ANUL-SLRY
    7) EMPLR-NAME
    8) CLNT-NAME
    9) CLNT-ID
   10) EMPLR-PH
X) EXIT

MAKE SELECTION ===>
```

Exhibit 5.18

5.5 Set Update

The SET_UPDATE module uses the customization software previously discussed. After more meaningful names are selected (see Exhibit 5.19), few decisions concerning a set remain due to the fact that selections 2-5 are derived during data base initialization (see Exhibit 5.20).

```
** SET UPDATE **

1)  CREATE SET
2)  OWNED-BY
3)  OWNS
4)  STK-CLNT-SET
5)  STK-CLNT-TXN-SET
X)  EXIT

MAKE SELECTION ===>
2
```

Exhibit 5.19

```
** CHANGE OR DELETE SET **

    1)  SET NAME:  OWNED-BY
    2)  SET OWNER:  STOCK
    3)  SET MEMBER:  CLIENT
    4)  SET VALUE:  1 TO MANY
    5)  SET INVERSE VALUE:  1 TO MANY
    6)  SET MEMBERSHIP:  MANDATORY AUTOMATIC
    7)  SET ORDER:  FIRST
    X)  EXIT
```

Exhibit 5.20

Any changes made to OWNER or MEMBER set attributes are verified by DB_CUST to assure that the the owner and member records that are selected exist and are disjoint (remember Lii sets are not allowed). If a set was established by an NFD, the SET VALUE is 1-to-Many and if it was derived from an FD its SET VALUE is 1-to-1. The SET INVERSE for 1-to-Many SET VALUE is assumed 1-to-1 unless one or both of the following are true:

1) The closure of the concatenation of owner and member candidate keys is equal to the closure of another record (i.e., an intersection record exists between the owner and member records).

2) At least one other record exists that has the opposite relationship of owner and member records of the set in question (i.e., an M-to-N relationship exists).

The SET INVERSE for a set with a SET VALUE of 1-to-1 is assumed to be 1-to-Many. Otherwise, the member and owner records are the same.

Possibly the most mystifying of all IDMS parameters is SET MEMBERSHIP. To help offset the perplexities of SET MEMBERSHIP, excerpts from an IDMS programmer's guide (CADY80) supplement the SET MEMBERSHIP help feature (see Exhibit 5.21).

```
        ** SET MEMBERSHIP VALUES **

    1)  MANDATORY AUTOMATIC --
        DISCONNECTION FROM SET ONLY BY ERASING RECORD
        CONNECTION TO SET AUTOMATIC WHEN STORED

    2)  MANDATORY MANUAL -
        DISCONNECTION FROM SET ONLY BY ERASING RECORD
        CONNECTION TO SET VIA "CONNECT" STATEMENT

    3)  OPTIONAL AUTOMATIC -
        DISCONNECTION FROM SET VIA "DISCONNECT" STATEMENT
        CONNECTION TO SET AUTOMATIC WHEN STORED

    4)  OPTIONAL MANUAL -
        DISCONNECTION FROM SET VIA "DISCONNECT" STATEMENT
        CONNECTION TO SET VIA "CONNECT" STATEMENT

    X)  EXIT

    ====>
```

Exhibit 5.21

With respect to SET ORDER, a final series of menu traversals graphically summarizes DB_CUST's capabilities. Exhibit 5.22 illustrates a user's request for the SET ORDER option. A selection of ASCENDING (see Exhibit 5.23) causes the "** CHANGE OR DELETE SET **" display to add the SORT ELEMENT option (see Exhibit 5.24). A sort field must be a data element present in the member record. In the likely event one

can not remember the spelling for the sort field data element, the help feature lists all possible options (see Exhibit 5.25).

```
** CHANGE OR DELETE SET **

1)  SET NAME:  OWNED-BY
2)  SET OWNER:  STOCK
3)  SET MEMBER:  CLIENT
4)  SET VALUE:  1 TO MANY
5)  SET INVERSE VALUE:  1 TO MANY
6)  SET MEMBERSHIP:  MANDATORY AUTOMATIC
7)  SET ORDER:  FIRST
X)  EXIT

====>
7
```

Exhibit 5.22

```
** SET ORDER VALUES **

1)  FIRST
2)  LAST
3)  NEXT
4)  PRIOR
5)  ASCENDING
6)  DESCENDING
X)  EXIT

====>
5
```

Exhibit 5.23

```
** CHANGE OR DELETE SET **

 1)  SET NAME:  OWNED-BY
 2)  SET OWNER:  STOCK
 3)  SET MEMBER:  CLIENT
 4)  SET VALUE:  1 TO MANY
 5)  SET INVERSE VALUE:  1 TO MANY
 6)  SET MEMBERSHIP:  MANDATORY AUTOMATIC
 7)  SET ORDER:  ASCENDING
 8)  SET SORT ELEMENT:
 9)  SET DUPLICATE OPTION:  DUPLICATES NO
 X)  EXIT

====>
8
```

Exhibit 5.24

```
** SELECT SORT ELEMENT **

1)  CLNT-ID
2)  CLNT-NAME
3)  ANUL-SLRY
4)  EMPLR-NAME
5)  EMPLR-PH
X)  EXIT

MAKE SELECTION ===>
```

Exhibit 5.25

5.6 Data Base Customization as a Maintenance Aid

Once the data base administrator feels confident that the data base entities have been properly customized, it is time to create the schema. It is likely however, that the first few attempts at schema creation will find missing data or unforeseen conflicts. These problems, in conjunction with requirement changes, may cause several revisits to DB_CUST.

Chapter 6

SCHEMA CREATION

Upon entry of the user's data base name, the initial menu entry of the system, edit checks and minor enhancements are performed to insure a clean IDMS schema compilation. Validation and conflict checking continue throughout the data base initialization and customization process, thus assuring many strong input assertions for the actual creation of the IDMS schema. Therefore, SCHEMA_CREATE, the module that creates schema source code, does not require user interaction to reformat entities into IDMS data definition statements. Of most interest is the way SCHEMA_CREATE:

1) simplifies M-to-N relationships to meet CODASYL (and IDMS) requirements,

2) generates pointer positions within records by simulating the IDMS "clock rule" algorithm (PERR77), and

3) establishes 1-to-1 relationships via owner pointers and foreign keys.

6.1 Check for Missing Data

Before addressing the actual schema creation, a missing data sub-module (MISSING_DATA_CHECK) must be successfully run. This module only delineates required missing data. If

missing data is detected, the data base administrator is notified (see Exhibit 6.1) and schema compilation is aborted.

```
*** REQUIRED BUT MISSING DATA ***

DATA ELEMENT:                    ATTRIBUTE:
-------------                    ---------
STK_ABRV                         FORMAT
STK_CLNT_QUAN                    TYPE

RECORDS:
-------
CLIENT                           CALC KEY
STK_CLNT_TXN                     VIA SET

SET:
---
OWNED_BY                         SET MEMBERSHIP
```

Exhibit 6.1

6.2 Verify Entity Customization

After missing data requirements are met through the use of DB_CUST, a scan of all entities is made to verify customization. As discussed in Chapter 5, Data Base Customization, nearly all entities require some customization. Unlike the missing data check, this routine generates only warning messages (see Exhibit 6.2) and then continues to the next process.

```
*** WARNING--ENTITIES NOT CUSTOMIZED ***

DATA ELEMENT:      RECORD:        SET
-------------      ------         ---
STK_NAME           RECORD3        SET4
CLIENT_NAME
```

Exhibit 6.2

6.3 Simplify Complex Relationships

Complex relationships (M-to-N relationships) are banned by CODASYL data base management system specifications. The user-required sets from the Wampum Brokerage System represent such a relationship. (see Exhibit 6.3).

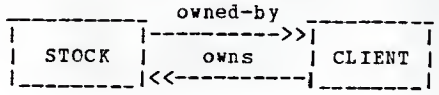


Exhibit 6.3

In all situations the solution lies in the creation of an intersection record (see Exhibit 6.4). The system automatically detects this situation by evaluating SET VALUE and SET INVERSE VALUE attributes of a set entity and creates the necessary intersection record and accompanying sets.

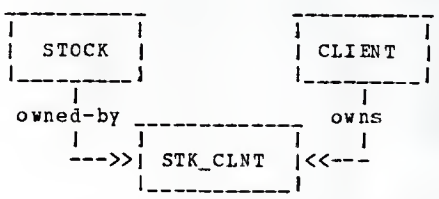


Exhibit 6.4

Traversals from STOCK to CLIENT are now made by the following statements:

```
OBTAIN NEXT SET(OWNED_BY);
OBTAIN OWNER SET(OWNS);
```

From CLIENT to STOCK just the opposite statements are required.

```
OBTAIN NEXT SET(OWNS);
OBTAIN OWNER SET(OWNED_BY):
```

Overtly, the solution seems flawless. But, if the user requires other services additional sets made be needed. For example, perhaps a CLIENT wishes to know which of his stocks have made the most money (see Exhibit 6.5).

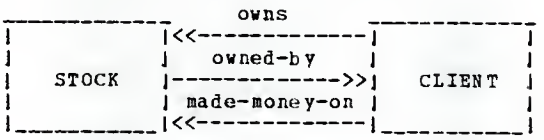


Exhibit 6.5

Sets OWNS and OWNED-BY address reciprocating questions, are correctly modeled by Exhibit 6.4, and can be easily implemented. However, given the existence of the OWNS and OWNED-BY sets, the implementation of the MADE-MONEY-ON set is less apparent. If a second intersection record were created, software could not distinguish reciprocating relationships like OWNS and OWNED-BY from non-reciprocating relationships like OWNED-BY and MADE-MONEY-ON. When multiple sets exist between records, the interpretation of these sets traditionally require human input. However, by again referring to Bernstein's research (BERN76), a different approach provides a solution to this problem without human intervention.

Recall from Chapter 3, Bernstein's Algorithm and User Requirements, that each of Bernstein's NFDs are converted to FDs by concatenating on the LS both the RS and LS attributes and creating a new RS , namely a unique theta attribute. Each theta represents a set and the value of theta (either "yes" or "no") indicates an association between current records. This concept, applied to a network model, limits intersection records to one, and set names become data elements within the intersection record (see Exhibit 6.6).

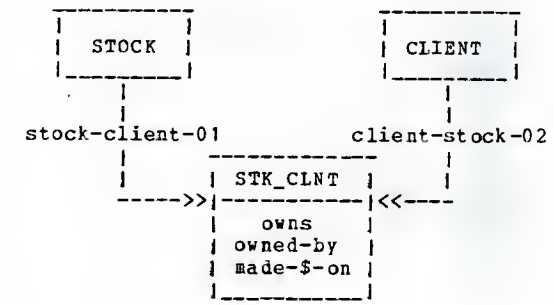


Exhibit 6.6

This simplification technique provides a consistent and useful method for solving M-to-N relationships, and by using IDMS' "logical record facility", traversals remain quite readable. For example, a traversal from STOCK to CLIENT would read

```
OBTAIN NEXT SET(STOCK_CLIENT_01) WHERE (OWNS = 'YES');
OBTAIN OWNER SET(CLIENT_STOCK_02);
```

Should the data base administrator find substantial difficulties with this transformation, any type of record/set configuration can be created via DB_CUST.

6.4 Establish 1-to-1 Relationships

The implementation of a 1-to-1 relationship is analogous to a traditional table lookup operation. Use of an IDMS set for this type of relationship is poor use of the software. A set (and all its pointers) should not be introduced if at most there is to be one occurrence of the member record. If an inverse relationship exists, there is no problem; the owner pointer provides the needed relationship. If the inverse relationship is non-existent, SCHEMA_CREATOR adds a member record candidate key to the owning record, providing it does not already exist (see Exhibit 6.7). The 1-to-1 relationship can now be accomplished by matching like-keys (see EMPLR_NAME in Exhibit 6.7).

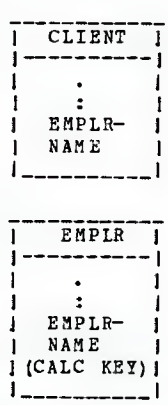


Exhibit 6.7

6.5 Generate Set Pointers

IDMS establishes pointers within records by a peculiar technique known as the "clock rule" (PERR77). Exhibit 6.8 graphically presents each record of the Wampum Brokerage conceptual schema spiraled twice in their 12 hour clocks.

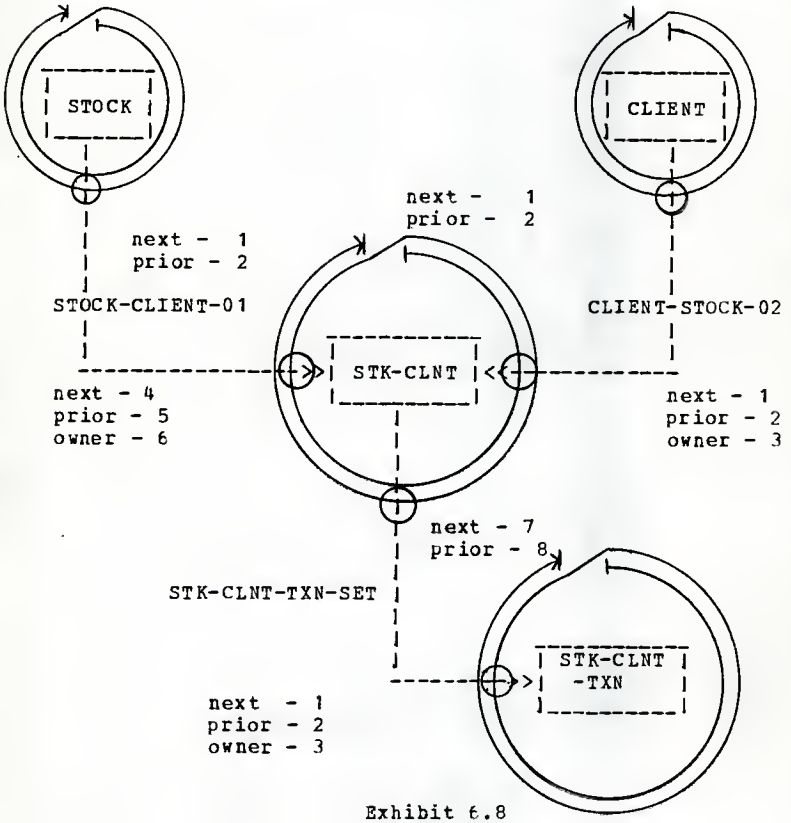


Exhibit 6.8

Starting at top center (12 o'clock), circle the graphical representation of a record twice in a clockwise direction. On the A.M. spiral, assign pointer positions for all sets in which the record participates as a member in the order the sets are encountered. On the P.M. spiral, assign pointer positions for all sets in which the record participates as an owner in the order the sets are encountered.

SCHEMA_CREATE simulates the clock rule algorithm in much the same way it is done graphically. Possibly the best form of explanation is a high level algorithm (see Exhibit 6.9).

```
DECLARE tables--member-next-pointer, member-prior-pointer,
                member-owner-pointer,owner-next-pointer,
                owner-prior-pointer

BEGIN ALGORITHM;

FOR EACH record DO;
  initialize pointer to 1;

  FOR EACH set DO;  /** simulates a.m. spiral **/

    IF member of current set = current record
       /** comment - assign member record pointers **/
       member-next-pointer (current set) <-- pointer;
       member-prior-pointer (current set) <-- pointer + 1;
       member-owner-pointer (current set) <-- pointer + 2;
       increment pointer by 3;
    END IF;

  END LOOP;

  FOR EACH set DO;  /** simulates p.m. spiral **/

    IF owner of current set = current record
      /** comment--assign owner record pointers **/
      owner-next-pointer (current set) <-- pointer;
      owner-prior-pointer (current set) <-- pointer + 1;
      increment pointer by 2;
    END IF;

  END LOOP;
END LOOP;
END ALGORITHM;
```

Exhibit 6.9

A detailed narrative explanation of the above algorithm would only serve to distort its clarity. In brief, the two inner loops simulate the member record and owner record pointer assignments for each record represented by the outer loop. The algorithm establishes all possible pointers (i.e., next, prior, and owner). Although this default option lacks praise for storage efficiency, there need not be any concern by the programmer about the existence of a pointer or any need for the data base administrator to perform a possibly costly regeneration of an operational system due to additional pointer requirements.

6.6 Create Data Definition Statements

The remaining function of SCHEMA_CREATE is the reformatting of the application's data base entities into compileable IDMS data definition language statements (see Exhibit 6.10). In SCHEMA DESCRIPTION, a substitution of the data base name and the current date for SCHEMA NAME and DATE (see lines 7000 and 8000 of Exhibit 6.10) is made. Although much potential exists for intelligently tuning the data base through AREA and FILE DESCRIPTIONS, this research does not address these issues. AREAs and FILEs are defaulted to one each (see lines 16000 and 24000 of Exhibit 6.10). The major reformatting work pertains to the RECORD and SET DESCRIPTIONs. Each existing record and set is obtained from the designed data base and with careful evaluation of each entity's parameters, the entity is converted to IDMS data definition source statements.

```
001000*
002000*****************************************************
003000*          ** SCHEMA DESCRIPTION **                 *
004000*****************************************************
005000*
006000 SCHEMA DESCRIPTION.
007000 SCHEMA NAME IS WDB.
008000 DATE.  12/02/83.
009000 INSTALLATION.  KSU
010000*
011000*****************************************************
012000*          ** FILE DESCRIPTION **                   *
013000*****************************************************
014000*
015000 FILE DESCRIPTION.
016000 FILE NAME IS IDMS-FILE1          ASSIGN TO SYS010.
017000 FILE NAME IS JOURNAL             ASSIGN TO SYS009.
018000*
019000*****************************************************
020000*          ** AREA DESCRIPTION **                   *
021000*****************************************************
022000*
023000 AREA DESCRIPTION.
024000 AREA NAME IS DB-AREA
025000       RANGE IS 1001 THRU 1100
026000       WITHIN FILE IDMS-FILE1 FROM 1 THRU 100.
027000*
028000*****************************************************
029000*          ** RECORD DESCRIPTION **                 *
030000*****************************************************
031000*
032000 RECORD DESCRIPTION.
033000 RECORD NAME STOCK.
034000 RECORD ID 100.
035000 LOCATION MODE CALC USING STK-ABRV DUPLICATES LAST.
036000 WITHIN DB-AREA AREA.
037000       05  STK-NAME                PIC X(16).
038000       05  STK-ABRV                PIC X(4).
039000       05  RCNT-QUT                PIC 999V9999.
040000       05  RCNT-VOL                PIC 9(7).
041000       05  STK-LAST-UPDTE.
042000           07  STK-DAY-UPDTE       PIC X(6).
043000           07  STK-TIME-UPDTE      PIC X(6).
044000       05  TOT-AMT-WHEN-REC        PIC 999V9999.
045000       05  TOT-NUM-WHEN-REC        PIC 9999.
046000*
047000 RECORD NAME CLIENT.
048000 RECORD ID 200.
049000 LOCATION MODE CALC USING CLNT-NAME DUPLICATES LAST.
050000 WITHIN DB-AREA AREA.
051000      05  CLNT-ID                  PIC X(9).
052000      05  CLNT-NAME                PIC X(25).
053000      05  ANUL-SLRY                PIC 9(6).
054000*
055000 RECORD NAME STK-CLNT
```

```
056000 RECORD ID 300.
057000 LOCATION MODE CALC USING STK-CLNT-KEY DN.
058000 WITHIN DB-AREA AREA.
059000       05  STK-CLNT-KEY.
060000           07  STK-ABRV          PIC X(4).
061000           07  CLNT-ID           PIC X(9).
062000       05  CLNT-STK-QUAN         PIC 9(5).
063000       05  CLNT-STK-INVST        PIC 999V99.
064000       05  OWNES                 PIC X(3).
065000       05  OWNED-BY              PIC X(3).
066000*
067000 RECORD NAME STK-CLNT-TXN.
068000 RECORD ID 400.
069000 LOCATION MODE CALC USING TXN-KEY DN.
070000 WITHIN DB-AREA AREA.
071000       05  TXN-KEY.
072000           07 STK-ABRV           PIC X(4).
073000           07 CLNT-ID            PIC X(9).
074000           07 TXN-DTE            PIC X(6).
075000       05  BRKR-RECMND           PIC X(3).
076000       05  CLNT-STK-TXN-PRC      PIC 999V9999.
077000       05  NUM-STK-CLNT-PUR      PIC 9(4).
078000*
079000*****************************************************
080000*            ** SET DESCRIPTION **                  *
081000*****************************************************
082000*
083000 SET DESCRIPTION.
084000 SET NAME STOCK-CLIENT-01  .
085000 ORDER IS SORTED          .
086000 MODE                  CHAIN .
087000 OWNER     STOCK       NEXT POSITION  1 PRIOR POSITION 2.
088000 MEMBER    STK-CLNT    NEXT POSITION  3 PRIOR POSITION 4
089000                       LINKED OWNER
090000                       OWNER POSITION 5
091000                       MANDATORY AUTOMATIC
092000                       ASCENDING KEY IS STK-CLNT-KEY
093000                       DUPLICATES NOT ALLOWED.
094000*
095000 SET NAME CLIENT-STOCK-02 .
096000 ORDER IS SORTED          .
097000 MODE                  CHAIN .
098000 OWNER     CLIENT      NEXT POSITION  1 PRIOR POSITION 2.
099000 MEMBER    STK-CLNT    NEXT POSITION  1 PRIOR POSITION 2
100000                       LINKED OWNER
101000                       OWNER POSITION 3
102000                       MANDATORY AUTOMATIC
103000                       ASCENDING KEY IS STK-CLNT-KEY
104000                       DUPLICATES NOT ALLOWED.
105000*
106000 SET NAME STK-CLNT-TXN-SET.
107000 ORDER IS SORTED          .
108000 MODE                   CHAIN.
109000 OWNER  STK-CLNT       NEXT POSITION 7 PRIOR POSITION 8.
110000 MEMBER STK-CLNT-TXN NEXT POSITION 1 PRIOR POSITION 2
```

```
111000                  LINKED OWNER
112000                  OWNER POSITION 3
113000                  MANDATORY AUTOMATIC
114000                  ASCENDING KEY IS TXN-KEY
115000                  DUPLICATES NOT ALLOWED.
```

Exhibit 6.10

6.7 Results of Schema Creation

Once the process of handling each record and set is completed, a CMS file containing the respective IDMS statements is created. The filename is assigned the data base name and the filetype is SCHMA (improperly spelled as required the by IDMS/CMS EXEC available at the Kansas State University computing center). One can be assured that the existing schema will compile successfully! If the data base designer prefers, the source file created by DB_GEN can be edited further before it is compiled.

Chapter 7

SUMMARY AND CONCLUSIONS

7.1 Contributions of this Research

This research shows that applied data base design aids (e.g., data dictionary and normalization) can be integrated into an encompassing automated data base design tool to assist the data base designer. It is no longer necessary for the data base designer to manually control large volumes of data produced through the data base design process, manually conduct the normalization process for hundreds of dependencies, re-execute schema compilation due to syntax errors or conflicting parameters, or develop his own data base design methodology through trial and error.

Although this research relies heavily on research by others (YOUR79,ROSS82,CODD70,BERN76), several areas lacked formal guidance. One such area was the transformation of NFDs into sets. Research being done with NFDs pertains strictly to the relational model (e.g., BERN76) and no NFD-to-set transformation processes were found for the network model. A second area of limited guidance was the transformation of a conceptual schema into a physical CODASYL network schema.

Several authors have developed useful conceptual-to-network model transformations (ATRE80,CHEN77), but none provide insight into the perplexing area of transforming multiple and complex relationships into CODASYL sets without user intervention. A final area of importance is that the entire design process is encompassed in a user friendly interactive menu-driven system that constantly assists the data base designer in the development of the user's data base schema. Although another automated data base design aid exists (ROSS82), DB_GEN is the only known data base design aid that produces an operational data base schema.

7.2 Status of Implementation

DB_GEN is currently operational with the following modules (refer to Exhibit 2.2): DB_ENTRY, LEVEL_2, DB_CUST, and UTILITY_RTNS.

7.3 Enhancements to DB_GEN

Retrospection on the system created, DB_GEN, reveals potential improvements. Any significant change to user requirements once the data base is customized leaves the user with a decision between reinitializing all data base entities (and losing customized information) and adding the new requirements without the design power of DB_INIT. Obviously, neither option is in the user's best interest. A better solution would be to retain attributes of previous entities and attempt to match similar reinitialized entites. Data

elements (through unique names), records (using candidate key closures), and sets (by their owner and member records) could be reestablished or, if not found, could be added to the data base.

7.4 Continued Research

Another potential improvement deals with the two algorithms in Chapter 5 that convert NFDs to CODASYL sets. Although the algorithms are sound, the author feels that continued research may reveal a technique to further interpret NFDs that are judged uninterpretable by DB_GEN. Such an expansion of either one or both of the algorithms would require further study of the intentions of the data base designer at the time an NFD was deemed necessary based on a user requirement.

Although this research addresses important areas of automated data base modeling and implementation, much work remains. Earlier stages of the design life cycle could associate FDs and NFDs with specific user requirements and associate these with organizational entities. User requirements could be supplemented with response time requirements, frequency estimates, security measures, integrity constraints, and user priorities. Organizational areas also need to collect information. By associating user requirements with an organizational area's priority, budget, volatility, staff experience, hardware and software availabili-

ty, etc., potential data base projects can be staged in an order most beneficial to the entire organization (COHE79). And if DBMS software becomes abundant, other software will be able to choose the DBMS that best fits an organization's requirements. Once a DBMS is selected, initialized, and made operational, live statistics can be kept to tune the data base management system for efficiency. For example, groups of data often accessed together can automatically be stored contiguously for better performance, and if statistics indicate substandard data base response time requirements, schema modifications can automatically be made to improve efficiency (e.g., add a secondary index) in order of user priorities.

BIBLIOGRAPHY

(ATRE80) Atre, S. (1980) Data Base Structured Techniques for Design, Performance and Management. New York: John Wiley & Sons.

(BERN76) Bernstein, Phillip (1976) "Synthesis Third Normal Form Relations From Functional Dependencies," ACM Transactions on Data Base Systems Vol. 1, No. 4, December.

(CADY80) Cady, William, Martin, Esther, eds. (1980) IDMS Programmer's Reference Guide - PL/I Rev. 0, Release 6.5, Wellesley: Cullinane Corporation, August 1980.

(CHEN77) Chen, Perter (1977) The Entity-Relationship Approach to Logical Data Base Design QED Monograph Series.

(CHEN76) Chen, Perter (1976) "The Entity-Relationship Model - Towards a Unified View of Data," ACM Transactions of Database Systems Vol. 1, No. 1, March, pp. 9-36.

(CODD70) Codd, Edgar (1970) "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM Vol. 13, No. 7, June, pp. 377-387.

(CODD72) Codd, E.F. (1972) "Further Normalization of the Data Base Relational Model," Data Base Systems (Courant Computer Science Symposium) Prentice-Hall, Englewood Cliffs, NJ.

(CODD79) Codd, E.F. (1979) "Extending the Database Relational Model to Capture More Meaning," ACM Transactions on Database Systems Vol. 4, No. 4, December, pp. 397-434.

(COHE79) Cohen, Leo (1979) Pre-Data Base Survey Princeton: Performance Development Corporation.

(DATE81) Date, C. J. (1981) An Introduction of Database Systems (third edition), Reading: Addison-Wesley.

(HAMM81) Hammer, M., McLeod, D. (1981) "Database Description with SDM: A Semantic Database Model," ACM Transactions on Database Systems Vol. 6, No. 3, September.

(KENT73) Kent, William (1973) "A Primer of Normal Forms," IBM Technical Report TR 02.600, December 17.

(KENT83) Kent, William (1983) "A Simple Guide to Five Normal Forms in Relational Database Theory," Communications of the ACM Vol. 26, No. 2, February, pp. 120-125.

(MCAR83) McCarthy, Elizabeth (1983) Telephone interview. TSI, International, 27 December.

(MART77) Martin, James (1977) Computer Data-Base Organization (second edition), Englewood Cliffs: Prentice-Hall.

(MCEL79) Mcelreath, T. Jack (1979) "Chapter 8: Defining the system; Chapter 9: Data Definition; Chapter 10: General Data Base Design," In IMS Design and Implementation Techniques Wellesley: QED Information Science.

(MEUR80) Meurer. Thomas F. (1980) "Solving the Mystery of Data Base Design," Computerworld/Extra! Vol. 14, No. 38, September 17, pp. 48-51.

(MOLI79) Molina, Francisco Walter (1979) "A Practical Data Base Design Method," Data Base Vol. 11, No. 1, Summer, pp. 3-11.

(OLLE78) Olle, William T. (1978) The CODASYL Approach to Data Base Management New York: John Wiley & Sons.

(ORR77A) Orr, Kenneth T. (1977) "Start with the Output," Infosystems October, pp. 86-88.

(ORR78B) Orr, Kenneth T. (1977) "The Logic of Data Base Structure," Infosystems September, pp. 58-59, 96.

(ORR77C) Orr, Kenneth T. (1978) "Procedures for Structured Data Base Design," Infosystems June, pp. 78-82.

(PERR77) Perron, Bob, writer (1977) IDMS Concepts and Facilities, Wellesley: Cullinane Corporation.

(ROSS82) Ross, Ronald G. (1982) "Solving the Current DBA Crisis," Data Base Newsletter.

(TSIC78) Tsichritzis, D.C., Lochovsky, F.H. (1978) "Designing the Data Base," Datamation Vol. 24, No. 8, August, pp. 147-151.

(UHRO73) Uhrowczid, P.P. (1978) "Data Dictionary/Directories," IBM Systems Journal Vol. 12, No. 4, pp. 332-350.

(ULLM82) Ullman, J. D. (1982) Principles of Database Systems (second edition). Rockville: Computer Science Press.

(YOUR79) Data Base Modelling and Design Workshop (second edition). (1979) New York: Yourdon Press.

Schema Data Definition Statements

```
001000*
002000*****************************************************************
003000*          ** SCHEMA DESCRIPTION **                             *
004000*****************************************************************
005000*
006000 SCHEMA DESCRIPTION.
007000 SCHEMA NAME IS DB-GEN-DB VERSION IS 1.
008000 DATE.  12/02/83.
009000 INSTALLATION.  KSU
012000*
013000 REMARKS.  THIS DATA BASE SCHEMA IS USED TO SUPPORT THE
014000           INPUT-OUTPUT REQUIREMENTS FOR THIS IMPLEMENTATION.
018000*
019000*****************************************************************
020000*          ** FILE DESCRIPTION **                               *
021000*****************************************************************
022000*
023000 FILE DESCRIPTION.
024000 FILE NAME IS IDMS-FILE1        ASSIGN TO SYS010.
025500 FILE NAME IS JOURNAL           ASSIGN TO SYS009.
026000*
027000*****************************************************************
028000*          ** AREA DESCRIPTION **                               *
029000*****************************************************************
030000*
031000 AREA DESCRIPTION.
032000 AREA NAME IS DB-AREA
033000      RANGE IS 1001 THRU 1100
034000      WITHIN FILE IDMS-FILE1 FROM 1 THRU 100.
035000*
036000*****************************************************************
037000*          ** RECORD DESCRIPTION **                             *
038000*****************************************************************
039000*
040000 RECORD DESCRIPTION.
041000 RECORD NAME DATA-BASE.
042000 RECORD ID 100.
043000 LOCATION MODE CALC USING DB-NAME DUPLICATES NOT ALLOWED.
044000 WITHIN DB-AREA AREA.
045000      05  DB-NAME                PIC X(16).
046000      05  DBA                    PIC X(4).
047000      05  DATE-CREATED.
048000          07  YEAR-CREATED       PIC X(2).
049000          07  MONTH-CREATED      PIC X(2).
050000          07  DAY-CREATED        PIC X(2).
051000      05  DATE-CHANGED.
```

```
052000           07  YEAR-CHANGED        PIC X(2).
053000           07  MONTH-CHANGED       PIC X(2).
054000           07  DAY-CHANGED         PIC X(2).
055000*
056000 RECORD NAME RE-CORD.
057000 RECORD ID 200.
058000 LOCATION MODE CALC USING REC-ID DUPLICATES LAST.
059000 WITHIN DB-AREA AREA.
060000      05  REC-ID                   PIC X(4).
061000      05  REC-NAME                 PIC X(16).
062000      05  REC-STRG-MODE            PIC X(2).
063000      05  REC-LCTN-MODE            PIC X(4).
064000      05  REC-DUP-OPTION           PIC X(2).
065000      05  REC-CALC-VIA             PIC X(16).
066000      05  REC-AREA                 PIC X(16).
067000*
068000 RECORD NAME DATA-ELEMENT
069000 RECORD ID 300.
070000 LOCATION MODE CALC USING LMNT-NAME DUPLICATES LAST.
071000 WITHIN DB-AREA AREA.
072000      05  LMNT-NAME                PIC X(16).
073000      05  LMNT-DEF                 PIC X(55).
074000      05  LMNT-TYPE                PIC X(17).
075000      05  TOTAL-SIZE               PIC 9(3).
076000      05  FRACTION-SIZE            PIC 9(1).
077000*
078000 RECORD NAME SE-T.
079000 RECORD ID 400.
080000 LOCATION MODE CALC USING SET-NAME DUPLICATES LAST.
081000 WITHIN DB-AREA AREA.
082000      05  SET-NAME                 PIC X(16).
083000      05  SET-LINK                 PIC X(3).
084000      05  SET-MEM                  PIC X(2).
085000      05  SET-ORDER                PIC X(5).
086000      05  SET-SORT-LMNT            PIC X(16).
087000      05  SET-DUP-OPTION           PIC X(2).
088000      05  SET-VALUE                PIC X(2).
089000      05  SET-INVRS-VAL            PIC X(2).
090000*
091000 RECORD NAME LMNT-REC.
092000 RECORD ID 500.
093000 LOCATION MODE VIA POPULATED-WITH SET.
094000 WITHIN DB-AREA AREA.
095000      05  LMNT-REC-DUMMY           PIC X(8).
096000*
097000 RECORD NAME FD-AND-NFD.
098000 RECORD ID 600.
099000 LOCATION MODE CALC USING LEFT-SIDE DUPLICATES NOT ALLOWED.
100000 WITHIN DB-AREA AREA.
101000      05  LEFT-SIDE                PIC X(16).
101100      05  HOW-MANY                 PIC X(2).
101200      05  RIGHT-SIDE               PIC X(16).
102000*
103000 RECORD NAME CONCAT-LMNT.
```

```
104000 RECORD ID 700.
105000 LOCATION MODE CALC USING CONCAT-FIELD DN.
106000 WITHIN DB-AREA AREA.
107000      05  CONCAT-FIELD          PIC X(16).
108000*
109000*****************************************************************
110000*           ** SET DESCRIPTION **                               *
111000*****************************************************************
112000*
113000 SET DESCRIPTION.
114000 SET NAME DEFINED-BY      .
115000 ORDER IS SORTED          .
116000 MODE                 CHAIN.
117000 OWNER    DATA-BASE        NEXT POSITION  3.
118000 MEMBER   DATA-ELEMENT     NEXT POSITION  1
119000                           LINKED OWNER
120000                           OWNER POSITION 2
121000                           MANDATORY AUTOMATIC
122000                           ASCENDING KEY IS LMNT-NAME
123000                           DUPLICATES NOT ALLOWED.
124000*
125000 SET NAME DIVIDED-INTO    .
126000 ORDER IS SORTED          .
127000 MODE                 CHAIN.
128000 OWNER    DATA-BASE           NEXT POSITION  2.
129000 MEMBER   RE-CORD              NEXT POSITION  1
130000                              LINKED OWNER
131000                              OWNER POSITION 2
132000                              MANDATORY AUTOMATIC
133000                              ASCENDING KEY IS REC-NAME
134000                              DUPLICATES NOT ALLOWED.
135000*
136000 SET NAME LINKED-BY.
137000 ORDER IS SORTED          .
138000 MODE                 CHAIN.
139000 OWNER    DATA-BASE           NEXT POSITION  1.
140000 MEMBER   SE-T                 NEXT POSITION  5
141000                              LINKED OWNER
142000                              OWNER POSITION 6
143000                              MANDATORY AUTOMATIC
144000                              ASCENDING KEY IS SET-NAME
145000                              DUPLICATES NOT ALLOWED.
146000*
147000 SET NAME CONCAT-WITH.
148000 ORDER IS SORTED          .
149000 MODE                 CHAIN.
150000 OWNER    DATA-ELEMENT        NEXT POSITION  4.
151000 MEMBER   CONCAT-LMNT          NEXT POSITION  1
152000                              MANDATORY AUTOMATIC
153000                              ASCENDING KEY IS CONCAT-FIELD
154000                              DUPLICATES NOT ALLOWED.
155000*
156000 SET NAME POPULATED-WITH.
157000 ORDER IS FIRST           .
```

```
158000 MODE                    CHAIN.
159000 OWNER     RE-CORD                NEXT POSITION  5.
160000 MEMBER    LMNT-REC               NEXT POSITION  1
161000                                  MANDATORY AUTOMATIC.
164000*
165000 SET NAME GROUPED-IN.
166000 ORDER IS FIRST          .
167000 MODE                    CHAIN.
168000 OWNER     DATA-ELEMENT           NEXT POSITION  3.
169000 MEMBER    LMNT-REC               NEXT POSITION  2
170000                                  LINKED OWNER
171000                                  OWNER POSITION 3
172000                                  MANDATORY AUTOMATIC.
173000*
174000 SET NAME OWNER-OF.
175000 ORDER IS FIRST          .
176000 MODE                    CHAIN.
177000 OWNER     RE-CORD                NEXT POSITION  3.
178000 MEMBER    SE-T                   NEXT POSITION  3
179000                                  LINKED OWNER
180000                                  OWNER POSITION 4
181000                                  OPTIONAL MANUAL
182000*
183000 SET NAME MEMBER-OF.
184000 ORDER IS FIRST          .
185000 MODE                    CHAIN.
186000 OWNER     RE-CORD                NEXT POSITION  4.
187000 MEMBER    SE-T                   NEXT POSITION  1
188000                                  LINKED OWNER
189000                                  OWNER POSITION 2
190000                                  OPTIONAL MANUAL
191000*
192000 SET NAME INITIALIZED-BY.
193000 ORDER IS SORTED         .
194000 MODE                    CHAIN.
195000 OWNER     DATA-BASE              NEXT POSITION  4.
196000 MEMBER    FD-AND-NFD             NEXT POSITION  1
197000                                  LINKED OWNER
198000                                  OWNER POSITION 2
199000                                  MANDATORY AUTOMATIC
200000                                  ASCENDING KEY IS LEFT-SIDE
201000                                  DUPLICATES NOT ALLOWED.
```

Device Media Control Statements

```
000100 DEVICE-MEDIA DESCRIPTION.
000200 DEVICE-MEDIA NAME IS MDMCL OF SCHEMA NAME DB-GEN-DB.
000300 AUTHOR.                MARK COSTELLO.
000400 DATE.                  09/13/83.
000500 INSTALLATION.          KSU.
000600 REMARKS.               DMCL FOR DB_GEN.
000700
000800 BUFFER SECTION.
000900     BUFFER NAME IS IDMS-BUFFER
001000     PAGE CONTAINS 496 CHARACTERS
001100     BUFFER CONTAINS 100 PAGES.
001200
001300 AREA SECTION.
001400     COPY DB-AREA AREA.
```

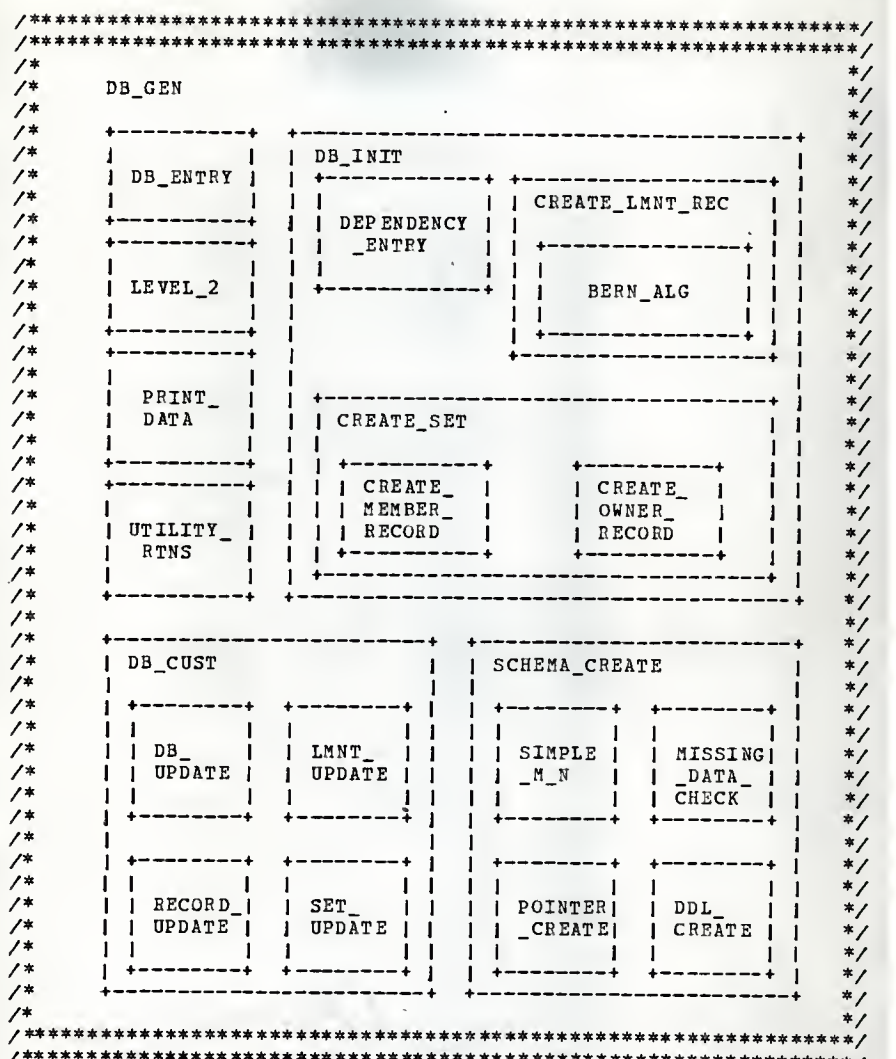
Subschema Data Defintion Statements

```
000100 ADD SUBSCHEMA NAME IS MSUB
000200       OF SCHEMA NAME IS DB-GEN-DB
000300         DMCL NAME IS MDMCL.
000400 ADD  AREA     DB-AREA.
000500 ADD  RECORD   DATA-BASE.
000600 ADD  RECORD   RE-CORD.
000700 ADD  RECORD   SE-T.
000800 ADD  RECORD   DATA-ELEMENT.
000900 ADD  RECORD   LMNT-REC.
000950 ADD  RECORD   CANDIDATE-KEY.
000975 ADD  RECORD   CONCAT-LMNT.
001000 ADD  SET      DEFINED-BY.
001050 ADD  SET      DIVIDED-INTO.
001100 ADD  SET      LINKED-BY.
001200 ADD  SET      MEMBER-OF.
001300 ADD  SET      OWNER-OF.
001325 ADD  SET      POPULATED-WITH.
001350 ADD  SET      GROUPED-IN.
001375 ADD  SET      DETERMINED-BY.
001400 ADD  SET      CONCAT-WITH.
001500 GENERATE.
```

Data Manipultion Source Statements

```
/*DMLIST*/
/*SCHEMA_COMMENTS*/
DB_GEN:  PROC OPTIONS (MAIN);

/*****************************************************************/
/*****************************************************************/
/*  This program automates the data base design process through */
/*  the use of applied data base design principles.  The output */
/*  of this implementation is operational IDMS schema data      */
/*  definition statements representing a user's data base       */
/*  schema.                                                     */
/*                                                              */
/*  written by: Mark Costello                                   */
/*                                                              */
/*  date:  December 1983                                        */
/*                                                              */
/*  The block diagram on the next page illustrates the major    */
/*  modules for this program.  Each module will be broken down  */
/*  further into sub-modules.                                   */
/*****************************************************************/
/*****************************************************************/
```


DB_GEN
DB_ENTRY
LEVEL_2
PRINT_ DATA
UTILITY_ RTNS
DB_INIT
DEPENDENCY _ENTRY
CREATE_LMNT_REC
BERN_ALG
CREATE_SET
CREATE_ MEMBER_ RECORD
CREATE_ OWNER_ RECORD
DB_CUST
DB_ UPDATE
LMNT_ UPDATE
RECORD_ UPDATE
SET_ UPDATE
SCHEMA_CREATE
SIMPLE _M_N
MISSING _DATA_ CHECK
POINTER _CREATE
DDL_ CREATE

```
/*****************************************************************/
/* This is the main driver module.  To precede from this module */
/* one must either create a new data base, or select             */
/* an existing data-base for enhancement or study.               */
/*****************************************************************/
        DCL (IDMS,ABORT) OPTIONS (INTER,ASM) ENTRY;
        DCL  CLRSCR ENTRY;
        DCL (MSUB SUBSCHEMA, DB-GEN-DB SCHEMA VERSION 1)
            MODE (BATCH);
        INCLUDE IDMS (SUBSCHEMA_DESCRIPTION);
        INCLUDE IDMS (SUBSCHEMA_BINDS);
        READY AREA (DB_AREA) PROTECTED UPDATE;
        CALL IDMS_STATUS;
        /*  IF I GET A MESSAGE 0966 FROM HERE THEN I MUST UNLOCK
            MY "AREA(S)".  THIS HAPPENS IF THE PROGRAM ABENDS.
            EITHER RE "IDMSINIT DBASE" (WHICH LOSES WHAT IS IN
            THE DATA BASE) OR USE THE "PFIX UTILITY" (SAVES
            WHAT IS IN THE DATA BASE).  SEE NOTES.          */

        INCLUDE IDMS (IDMS_STATUS);
        DCL     (OK,REC_FOUND)              CHAR(4) INIT ('0000'),
                EDIT_OUT                    CHAR(72),
                MENU_ENTRY                  CHAR(69),
                MSG                         CHAR(60),
                ENTER_KEY                   CHAR(1),
                (I,J,COUNT)                 FIXED DEC(3),
                DISPLAY_TBL(500)            CHAR(72),
                DOMAIN_TBL(500)             CHAR(16),
                (SAVE_DB_NAME,SAVE_NAME)    CHAR(16),
                DB_KEY_TBL(500)             FIXED BINARY(31),
                MENU_NUM                    CHAR(3);

        MENU_ENTRY = ' ';
        CALL DB_ENTRY (MENU_NUM,MENU_ENTRY);
        DO WHILE  (MENU_NUM  ¬=  'X');
           IF  MENU_NUM  =  '1'
           THEN
              CALL  NEW_DB (MENU_NUM,MENU_ENTRY);
           ELSE DO;
                OBTAIN CALC RECORD (DATA_BASE);
                CALL IDMS_STATUS;
                CALL LEVEL_2 (MENU_ENTRY);
           END;
           CALL DB_ENTRY (MENU_NUM,MENU_ENTRY);
        END;

        FINISH;
```

```
DB_ENTRY:  PROC (MENU_NUM, MENU_ENTRY);
/*****************************************************************/
/* This routine displays the primary menu and accepts the reply.*/
/* The primary menu consists of an option to start a new data    */
/* base, select a data-base that already exists, or exit from    */
/* the system.  The MENU_NUM parameter returns an "x", "1" or a */
/* valid menu number.   (Note:  "x" stands for exit)             */
/* If MENU_NUM is a valid menu number then the respective        */
/* data base will be made "current".                             */
/*****************************************************************/
/*                                                               */
/*    +------------+       +----------+     +------------+      */
/*    |            |       |          |     |            |      */
/*    |  DB_ENTRY  |       |  NEW_DB  |     | DB_UP_MENU |      */
/*    |            |       |          |     |            |      */
/*    +------------+       +----------+     +------------+      */
/*                                                               */
/*****************************************************************/

        DCL             SLCT_NUM                FIXED DEC(3),
                        MENU_NUM                CHAR(3),
                        MENU_ENTRY              CHAR(69);

        /*      LOAD  NAME TABLE WITH DATA BASES     */

        DISPLAY_TBL = ' ';
        PUT STRING (DISPLAY_TBL(1)) EDIT ('1)  CREATE DATA BASE')
                                         (X(4),A);

        OBTAIN FIRST RECORD (DATA_BASE) AREA (DB_AREA);
        IF ERROR_STATUS ¬= '0307'
        THEN CALL IDMS_STATUS;
        COUNT = 1;
        DO WHILE (ERROR_STATUS = REC_FOUND);
          COUNT = COUNT + 1;
          PUT STRING (DISPLAY_TBL(COUNT)) EDIT (COUNT,')  ',
            DB_NAME) (X(2),F(3),2(A));
          OBTAIN NEXT RECORD (DATA_BASE) AREA (DB_AREA);
          IF ERROR_STATUS ¬= '0307'
          THEN CALL IDMS_STATUS;
        END;
        PUT STRING (DISPLAY_TBL(COUNT + 1))
          EDIT ('    X)  EXIT') (A);

        /*      DISPLAY MENU  /  ACCEPT EDITED REPLY   */

        DB_NAME = ' ';
        CALL GEN_MENU (MENU_NUM,MENU_ENTRY,
             '** DATA BASE ENTRY **', COUNT,SLCT_NUM,3);

        /*   SET CURRENCY FOR RESPECTIVE DATA BASE  */

        IF ¬(MENU_NUM = 'X'  |  MENU_NUM = '1')
        THEN DO;
```

```
          DB_NAME = SUBSTR(DISPLAY_TBL(SLCT_NUM),9,16);
          END;
END DB_ENTRY;

NEW_DB:   PROC (MENU_NUM,MENU_ENTRY);
/*****************************************************************/
/* This module establishes a new user's data base for develop-   */
/* ment.  Fields of the DATA BASE structure are assigned values */
/* and then the DATA BASE occurrence is stored in DB_GEN's data */
/* base.                                                         */
/*****************************************************************/

        DCL       SLCT                CHAR(72),
                  MENU_NUM            CHAR(3),
                  STATUS              CHAR(4),
                  MENU_ENTRY          CHAR(69),
                  D                   CHAR(6),
                  DATE                BUILTIN;

        DATA_BASE = ' ';
        IF MENU_ENTRY = ' '
        THEN
          CALL DB_UP_MENU (MENU_NUM,MENU_ENTRY,
                           '** CREATE NEW DATA BASE **');
        ELSE
          MENU_NUM = '1';
        DO WHILE (MENU_NUM ¬= 'X');
          IF MENU_NUM = '1'  THEN DO;
               DB_NAME = MENU_ENTRY;
               CALL EDIT_NAME (DB_NAME,STATUS);
               IF STATUS = 'GOOD'
               THEN DO;
                 FIND CALC RECORD (DATA_BASE);
                 IF ERROR_STATUS ¬= '0326'
                 THEN CALL IDMS_STATUS;
                 IF ERROR_STATUS = REC_FOUND
                 THEN DO;
                   PUT STRING (MSG) EDIT
                     (DB_NAME,' ALREADY EXISTS') (2(A));
                   CALL MESSAGES;
                   DB_NAME = ' ';
                   END;
                 END;
               ELSE
                 DB_NAME = ' ';
               END;
          ELSE IF MENU_NUM = '2'  THEN
               DBA = MENU_ENTRY;
          CALL DB_UP_MENU (MENU_NUM,MENU_ENTRY,
                           '** CREATE NEW DATA BASE **');
        END;

        IF  DB_NAME = ' '
        THEN DO;
```

```
            MSG = 'DATA BASE NAME IS BLANK -- NO ADD MADE';
            CALL MESSAGES;
            RETURN;
            END;

          D = DATE;
          YEAR_CREATED, YEAR_CHANGED = SUBSTR(D,1,2);
          MONTH_CREATED, MONTH_CHANGED = SUBSTR(D,3,2);
          DAY_CREATED, DAY_CHANGED = SUBSTR(D,5,2);
          STORE RECORD (DATA_BASE);
          CALL IDMS_STATUS;
END NEW_DB;

DB_UP_MENU:  PROC (MENU_NUM,MENU_ENTRY,MENU_MSG);
/*****************************************************************/
/* This module displays the fields to be added or modified       */
/* for the DATA BASE structure.                                  */
/*****************************************************************/

          DCL       SLCT_NUM                  FIXED DEC(3),
                    SLCT                      CHAR(72),
                    STATUS                    CHAR(4),
                    MENU_NUM                  CHAR(3),
                    MENU_ENTRY                CHAR(69),
                    MENU_MSG                  CHAR(30);

          STATUS = 'BAD';
          DO WHILE (STATUS = 'BAD');
            CALL CLRSCR;
            PUT STRING (EDIT_OUT) EDIT (MENU_MSG)
                                       (X(15),A);
            DISPLAY (EDIT_OUT);
            CALL BLANK_LINE(2);
            PUT STRING (EDIT_OUT) EDIT
                ('1)  DATA BASE NAME:  ',DB_NAME) (2(A));
            DISPLAY (EDIT_OUT);
            PUT STRING (EDIT_OUT) EDIT
                ('2)  DATA BASE ADMINISTRATOR:  ',DBA) (2(A));
            DISPLAY (EDIT_OUT);
            PUT STRING (EDIT_OUT) EDIT ('X)  EXIT') (A);
            DISPLAY (EDIT_OUT);
            CALL BLANK_LINE(2);
            DISPLAY ('===>')   REPLY (SLCT);
            CALL EXAMINE_ENTRY (SLCT,MENU_NUM,MENU_ENTRY,SLCT_NUM,
                                STATUS,2);
          END;
END DB_UP_MENU;
```

```
LEVEL_2:  PROC (MENU_ENTRY);
/*****************************************************************/
/* This module calls the modules representing the primary        */
/* services of DB_GEN.                                           */
/*****************************************************************/
/*                                                               */
/*      LEVEL_2                                                  */
/*                                                               */
/*      +--------------+                                         */
/*      |              |                                         */
/*      | PRIMARY_MENU |                                         */
/*      |              |                                         */
/*      +--------------+                                         */
/*                                                               */
/*****************************************************************/
       DCL  MENU_NUM                    CHAR(3),
            MENU_ENTRY                  CHAR(69),
            DEL_SW                      CHAR(1),
            SLCT_2                      CHAR(72);

       IF MENU_ENTRY = 'DELETE'
       THEN DO;
         PUT STRING (MSG) EDIT
             (DB_NAME,' DATA BASE DELETED')(2(A));
         ERASE RECORD (DATA_BASE) ALL;
         CALL IDMS_STATUS;
         CALL MESSAGES;
         RETURN;
         END;

       CALL PRIMARY_MENU;
       DO WHILE  (SLCT_2  ¬=  'X');
          IF  MENU_NUM  =  '1' THEN
              CALL  LMNT_UPDATE (MENU_ENTRY);
          ELSE IF  MENU_NUM  =  '2'  THEN
              CALL  RECORD_UPDATE (MENU_ENTRY);
          ELSE IF  MENU_NUM  =  '3'  THEN
              CALL  SET_UPDATE (MENU_ENTRY);
          ELSE IF  MENU_NUM  =  '4'  THEN
              DO;
                CALL CHG_DEL_DB (DEL_SW);
                IF DEL_SW = 'D'
                THEN
                    RETURN;
              END;
          ELSE IF  MENU_NUM  =  '5'  THEN
              CALL  PRINT_DATA;
          ELSE IF  MENU_NUM  =  '6'  THEN
              CALL  CREATE_SCHEMA;
          ELSE DO;
              MSG = 'ERROR:  LEVEL_2 CASE STATEMENT';
              CALL MESSAGES;
              END;
          CALL PRIMARY_MENU;
```

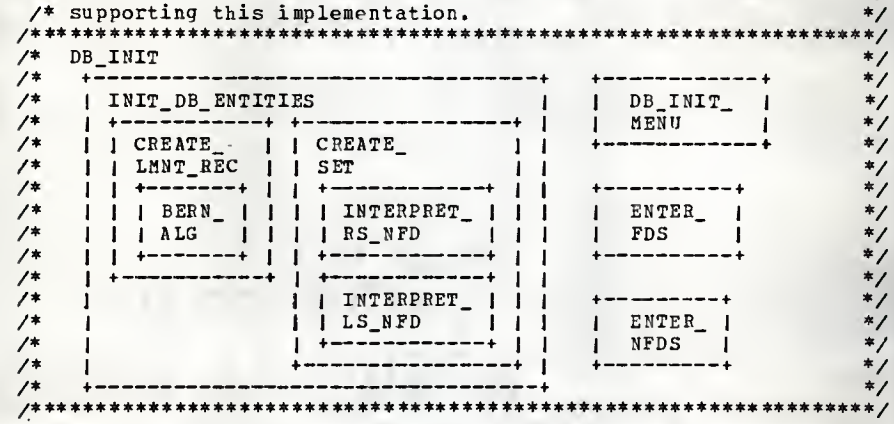
```
        END;

PRIMARY_MENU:  PROC;
/*****************************************************************/
/* This module displays the primary menu of services of DB_GEN. */
/*****************************************************************/

        DCL       SLCT_STATUS                CHAR(4),
                  SLCT_NUM                   FIXED DEC(3);

        SLCT_STATUS = 'BAD';
        DO WHILE (SLCT_STATUS = 'BAD');
              MSG = '** PRIMARY  MENU **';
              CALL  MENU_HEAD;
              DISPLAY  ('1)  DATA BASE INITIALIZATION');
              DISPLAY  ('2)  DATA ELEMENT UPDATE     ');
              DISPLAY  ('3)  RECORD UPDATE           ');
              DISPLAY  ('4)  SET UPDATE              ');
              DISPLAY  ('5)  DATA BASE UPDATE        ');
              DISPLAY  ('6)  PRINT DATA              ');
              DISPLAY  ('7)  CREATE SCHEMA           ');
              DISPLAY  ('X)  EXIT                    ');
              CALL BLANK_LINE(2);
              DISPLAY  ('==>')  REPLY  (SLCT_2);
              CALL EXAMINE_ENTRY (SLCT_2,MENU_NUM,MENU_ENTRY,
                                  SLCT_NUM,SLCT_STATUS,7);
        END;
    END PRIMARY_MENU;
END LEVEL_2;
```

```
DB_INIT:  PROC;
/*****************************************************************/
/* This is a major global module of DB_GEN.  Once completed it   */
/* will accept functional and nonfunctional dependencies and     */
/* use those dependencies to create instances of the             */
/* DATA-ELEMENT, RECORD, and SET structures.  The establishment  */
/* of these structure instances is described in the thesis       */
/* supporting this implementation.                               */
/*****************************************************************/
```




```
/*****************************************************************/

        MSG = 'CREATE_SCHEMA TO BE COMPLETED';
        CALL MENU_HEAD;
        DISPLAY ('PRESS ENTER TO CONTINUE') REPLY (ENTER_KEY);
        END  CREATE_SCHEMA;
```

```
/*****************************************************************/
/*          **  The DB_CUST modules starts here. **              */
/*                                                               */
/* The module name DB_CUST only serves as a logical grouping     */
/* of modules and is not itself an actual module.  The modules   */
/* within DB_CUST follow.                                        */
/*****************************************************************/

LMNT_UPDATE:  PROC(MENU_ENTRY);
/*****************************************************************/
/* This module displays all existing DATA_ELEMENT occurrences    */
/* for the "current" DATA_BASE structure and allows the user     */
/* to add a new DATA_ELEMENT occurrence, change and existing     */
/* DATA_ELEMENT occurrence, or delete an existing DATA_ELEMENT   */
/* occurrence.                                                   */
/*****************************************************************/
/*                                                               */
/*   LMNT_UPDATE                                                 */
/*                                                               */
/* +-----------------------------------------+  +------------+ */
/* |  NEW_LMNT                               |  |CHG_DEL_LMNT| */
/* | +-----------+ +----------+ +-----------+ |  | +-------+  | */
/* | | NEW_      | | LMNT_    | | NEW_CONCAT| |  | | LMNT_ |  | */
/* | | LMNT_NAME | | NEW_MENU | | _FIELD    | |  | | CHG_  |  | */
/* | +-----------+ +----------+ | +------+  | |  | | MENU  |  | */
/* |                            | | DUP_ |  | |  | +-------+  | */
/* |                            | | CK   |  | |  | +-------+  | */
/* |                            | +------+  | |  | | CHG_  |  | */
/* |                            +-----------+ |  | | LMNT_ |  | */
/* +------------------------------------------+  | | NAME  |  | */
/*      +--------+  +--------+       +-----------+ +-------+  | */
/*      | TOTAL_ |  | FRCTN_ |       |  +--------------------+  | */
/*      | SIZE_  |  | SIZE_  |       |  | CHG_CONCAT_FIELD   |  | */
/*      | RTN    |  | RTN    |       |  |    +--------+      |  | */
/*      +--------+  +--------+       |  |    | DUP_   |      |  | */
/*   +----------+  +----------+      |  |    | CK     |      |  | */
/*   | LMNT_    |  | LMNT_    |      |  |    +--------+      |  | */
/*   | TYPE_RTN |  | DEF_RTN  |      |  +--------------------+  | */
/*   +----------+  +----------+      +--------------------------+ */
/*                                                               */
/*****************************************************************/

        DCL       MENU_ENTRY              CHAR(69),
                  SLCT_NUM                FIXED DEC(3),
                  MENU_NUM                CHAR(3);

  /* IF DATA ELEMENT NAME SUPPLIED THEN SKIP DATA ELEMENT MENU */

        IF MENU_ENTRY ¬= ' '
        THEN DO;
          LMNT_NAME = MENU_ENTRY;
          SAVE_DB_NAME = DB_NAME;
          OBTAIN CALC RECORD (DATA_ELEMENT);
```

```
        DO WHILE (ERROR_STATUS = REC_FOUND);
          OBTAIN OWNER SET (DEFINED_BY);  CALL IDMS_STATUS;
          IF SAVE_DB_NAME = DB_NAME
          THEN
            ERROR_STATUS = 'FWND';
          ELSE
            OBTAIN DUPLICATE RECORD (DATA_ELEMENT);
        END;
        IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
        THEN CALL IDMS_STATUS;
        IF (ERROR_STATUS = 'FWND')
        THEN DO;
          MENU_ENTRY = ' ';
          CALL CHG_DEL_LMNT (MENU_ENTRY);
          RETURN;
          END;
        ELSE DO;
          DB_NAME = SAVE_DB_NAME;
          OBTAIN CALC RECORD (DATA_BASE);
          PUT STRING (MSG) EDIT (MENU_ENTRY,
              ' IS NOT AN EXISTING DATA ELEMENT') (A(16),A);
          CALL MESSAGES;
          END;
        END;

/*    LOAD TABLE WITH DATA ELEMENT NAMES    */

      MENU_NUM = ' ';
      DO WHILE (MENU_NUM ¬= 'X');
        DISPLAY_TBL = ' ';

        PUT STRING (DISPLAY_TBL(1)) EDIT ('1)  CREATE ELEMENT')
                                         (X(4),A);
        OBTAIN FIRST SET (DEFINED_BY);
        IF ERROR_STATUS ¬= '0307'
        THEN CALL IDMS_STATUS;
        COUNT = 1;
        DO WHILE (ERROR_STATUS = REC_FOUND);
          COUNT = COUNT + 1;
          PUT STRING (DISPLAY_TBL(COUNT)) EDIT (COUNT,')  ',
              LMNT_NAME)  (X(2),F(3),2(A));
          DB_KEY_TBL(COUNT) = DBKEY;
          OBTAIN NEXT SET (DEFINED_BY);
          IF ERROR_STATUS ¬= '0307'
          THEN CALL IDMS_STATUS;
        END;
        PUT STRING (DISPLAY_TBL(COUNT + 1))
            EDIT ('    X)  EXIT') (A);

      /*    DISPLAY MENU   /  ACCEPT EDIT REPLY   */

        CALL GEN_MENU(MENU_NUM,MENU_ENTRY,
             '** DATA ELEMENT UPDATE **',
             COUNT,SLCT_NUM,3);
```

```
        IF MENU_NUM = 'X'
        THEN
          RETURN;

      /*  CONTINUE TO UPDATE DATA ELEMENTS UNTIL EXIT    */

          IF SLCT_NUM = 1
          THEN
            CALL NEW_LMNT (MENU_ENTRY);
          ELSE DO;
            LMNT_NAME = SUBSTR(DISPLAY_TBL(SLCT_NUM),9,16);
            OBTAIN RECORD (DATA_ELEMENT)
                          DBKEY (DB_KEY_TBL(SLCT_NUM));
            CALL IDMS_STATUS;
            CALL CHG_DEL_LMNT (MENU_ENTRY);
            END;
      END;
NEW_LMNT:    PROC (MENU_ENTRY);
/*****************************************************************/
/* This module establishes a new DATA_ELEMENT occurrence by      */
/* defaulting the DATA_ELEMENT fields and calling the modules    */
/* to update these fields based on the user's MENU_ENTRY.        */
/* Once updating is complete, the new DATA_ELEMENT occurrence    */
/* is stored.                                                    */
/*****************************************************************/

   DCL            MENU_NUM                 CHAR(3),
                  MENU_ENTRY               CHAR(69),
                  SLCT_NUM                 FIXED(3),
                  NUM_CONCAT               FIXED(3),
                  CONCAT_TBL(20)           CHAR(16),
                  (LOW_LIMIT,UP_LIMIT)     FIXED(3);

   /********* INITIALIZE AND DEFAULT RECORD FIELDS  ************/
   SLCT_NUM = 0;
   NUM_CONCAT = 0;
   CONCAT_TBL = ' ';
   UP_LIMIT        = 5;
   LOW_LIMIT       = 5;
   LMNT_NAME       = ' ';
   LMNT_DEF        = ' ';
   LMNT_TYPE       = 'CHARACTER';
   TOTAL_SIZE      = 10;
   FRACTION_SIZE   = 0;

/*    AS AN ELEMENT NAME BEEN PROVIDED FROM SECONDARY MENU  */

   IF MENU_ENTRY = ' '
   THEN
     CALL LMNT_NEW_MENU (MENU_NUM,MENU_ENTRY,SLCT_NUM);
   ELSE
     MENU_NUM = '1';
```

```
/*   ADD ELEMENT ATTRIBUTES UNTIL EXIT                          */

   DO WHILE (MENU_NUM ¬= 'X');
     IF MENU_NUM = '1' THEN
       CALL NEW_LMNT_NAME (MENU_ENTRY,LMNT_NAME);
     ELSE IF MENU_NUM = '2' THEN
       CALL LMNT_DEF_RTN (MENU_ENTRY,LMNT_DEF);
     ELSE IF MENU_NUM = '3' THEN
       CALL LMNT_TYPE_RTN (MENU_ENTRY,LMNT_TYPE);
     /*-------------------------------------------------*/
     IF (LMNT_TYPE = 'CHARACTER' & MENU_NUM = '4') THEN
           CALL TOTAL_SIZE_RTN (MENU_ENTRY,TOTAL_SIZE);
     ELSE IF (LMNT_TYPE = 'NUMERIC' & MENU_NUM = '4') THEN
           CALL TOTAL_SIZE_RTN (MENU_ENTRY,TOTAL_SIZE);
     ELSE IF (LMNT_TYPE = 'NUMERIC' & MENU_NUM = '5') THEN
           CALL FRCTN_SIZE_RTN (MENU_ENTRY,FRACTION_SIZE);
     ELSE IF (LMNT_TYPE = 'CONCATENATED' &
           (MENU_NUM >= '4' & SLCT_NUM < LOW_LIMIT)) THEN
           CALL NEW_CONCAT_FIELD (SLCT_NUM,MENU_ENTRY);
     CALL LMNT_NEW_MENU (MENU_NUM,MENU_ENTRY,SLCT_NUM);
   END;

   IF LMNT_NAME = ' '
   THEN DO;
     MSG = 'ELEMENT NAME IS BLANK -- NO ADD MADE';
     CALL MESSAGES;
     RETURN;
     END;
   STORE RECORD (DATA_ELEMENT);  CALL IDMS_STATUS;

   /******  STORE AND CONNECT DATA ELEMENT WITH ITS *********/
   /******  CONCATENATED FIELDS                     *********/
   DO I = 1 TO NUM_CONCAT;
     CONCAT_FIELD = CONCAT_TBL(I);
     STORE RECORD (CONCAT_LMNT);
   END;

LMNT_NEW_MENU:  PROC (MENU_NUM,MENU_ENTRY,SLCT_NUM);
/*****************************************************************/
/* This module displays the various DATA_ELEMENT fields that     */
/* can be updated for a selected DATA_ELEMENT occurrence.        */
/*****************************************************************/

   DCL            MENU_NUM                CHAR(3),
                  MENU_ENTRY              CHAR(69),
                  (I,SLCT_NUM)            FIXED(3);

   /***  LOAD DISPLAY TABLE ***/

   DISPLAY_TBL = ' ';

   PUT STRING (DISPLAY_TBL(1)) EDIT
       ('  1) ELEMENT NAME: ',LMNT_NAME) (2(A));
   PUT STRING (DISPLAY_TBL(2)) EDIT
```

```
        (' 2) DEFINITION: ',LMNT_DEF) (2(A));
    PUT STRING (DISPLAY_TBL(3)) EDIT
        (' 3) TYPE: ',LMNT_TYPE) (2(A));
    /*-------------------------------------------------*/
    IF (LMNT_TYPE = 'CHARACTER' | LMNT_TYPE = ' ') THEN
      DO;
      PUT STRING (DISPLAY_TBL(4)) EDIT
          (' 4) TOTAL SIZE: ',TOTAL_SIZE) (2(A));
      UP_LIMIT = 4;
      END;
    ELSE IF LMNT_TYPE = 'NUMERIC' THEN
      DO;
      PUT STRING (DISPLAY_TBL(4)) EDIT
          (' 4) TOTAL SIZE: ',TOTAL_SIZE) (2(A));
      PUT STRING (DISPLAY_TBL(5)) EDIT
          (' 5) FRACTION SIZE: ',FRACTION_SIZE) (2(A));
      UP_LIMIT = 5;
      END;
    ELSE IF LMNT_TYPE = 'CONCATENATED' THEN
      DO;
      DISPLAY_TBL(4) = ' 4) ADD SUB ELEMENT';
      DISPLAY_TBL(5) = ' DELETE SUB ELEMENT . . .';
      UP_LIMIT = 4;
      I = 0;
      DO WHILE (CONCAT_TBL(I+1) ¬= ' ');
        I = I+1;
        UP_LIMIT = UP_LIMIT + 1;
        PUT STRING (DISPLAY_TBL(I+5)) EDIT
            (' ',UP_LIMIT,') ',CONCAT_TBL(I)) (A,F(3),2(A));
      END;
      NUM_CONCAT = I;
      END;
    DISPLAY_TBL(UP_LIMIT + 1) = ' X) EXIT';
    CALL GEN_MENU (MENU_NUM,MENU_ENTRY,'** CREATE NEW ELEMENT **',
                   UP_LIMIT,SLCT_NUM,1);
  END LMNT_NEW_MENU;

NEW_LMNT_NAME:    PROC (MENU_ENTRY,LMNT_NAME);
/*****************************************************************/
/* This module allows the user to assign a data element name     */
/* to a newly created DATA_ELEMENT occurrence.  Before the name  */
/* is accepted, a check is made to verify that it does not       */
/* already exist.                                                */
/*****************************************************************/

    DCL             MENU_ENTRY              CHAR(69),
                    (LMNT_NAME,SAVE_NAME)   CHAR(16),
                    STATUS                  CHAR(4);

    LMNT_NAME = MENU_ENTRY;
    CALL EDIT_NAME (LMNT_NAME,STATUS);
    IF STATUS = 'GOOD'
    THEN DO;
      SAVE_NAME = LMNT_NAME;
```

```
      SAVE_DB_NAME = DB_NAME;
      OBTAIN CALC RECORD (DATA_ELEMENT);
      DO WHILE (ERROR_STATUS = REC_FOUND);
        OBTAIN OWNER SET (DEFINED_BY);
        IF SAVE_DB_NAME = DB_NAME
        THEN
          ERROR_STATUS = 'FWND';
        ELSE
          OBTAIN DUPLICATE RECORD (DATA_ELEMENT);
      END;
      IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
      THEN CALL IDMS_STATUS;
      IF ERROR_STATUS = 'FWND'
      THEN DO;
        PUT STRING (MSG) EDIT (LMNT_NAME,' ALREADY EXISTS') (2(A));
        CALL MESSAGES;
        LMNT_NAME = ' ';
        END;
      ELSE DO;
        DB_NAME = SAVE_DB_NAME;
        OBTAIN CALC RECORD (DATA_BASE);
        END;
      END;
    ELSE
      LMNT_NAME = ' ';
  END NEW_LMNT_NAME;

  NEW_CONCAT_FIELD:    PROC (SLCT_NUM,MENU_ENTRY);
/*****************************************************************/
/* This module adds and deletes sub-elements of a concatenated   */
/* data element.  If the user isn't sure which DATA_ELEMENT      */
/* occurrences to make sub-elements, a blank MENU_ENTRY will     */
/* list all DATA_ELEMENT occurrences for the respective DATA_    */
/* BASE occurrence.                                              */
/*****************************************************************/

    DCL             MENU_NUM            CHAR(3),
                    MENU_ENTRY          CHAR(69),
                    SAVE_NAME           CHAR(16),
                    (SLCT_NUM,I)        FIXED(3),
                    STATUS              CHAR(4);

    SAVE_NAME = LMNT_NAME;
    IF (SLCT_NUM = 4 & LMNT_TYPE = 'CONCATENATED')
    THEN DO;
      /*********  ADD NEW SUB ELEMENT  ***********/
      IF MENU_ENTRY ¬= ' '
      THEN DO;
        /******* SUB ELEMENT SUPPLIED VIA MENU_ENTRY ******/
        CONCAT_FIELD = MENU_ENTRY;
        IF CONCAT_FIELD = LMNT_NAME
        THEN DO;
          MSG = 'ELEMENT NAME & SUB ELEMENT ARE EQUAL - USE MENU';
          CALL MESSAGES;
```

```
    END;
  ELSE DO;
    /** SEE IF CONCAT_FIELD IS AN EXISTING DATA ELEMENT ***/
    SAVE_DB_NAME = DB_NAME;
    LMNT_NAME = CONCAT_FIELD;
    OBTAIN CALC RECORD (DATA_ELEMENT);
    DO WHILE (ERROR_STATUS = REC_FOUND);
      OBTAIN OWNER SET (DEFINED_BY); CALL IDMS_STATUS;
      IF SAVE_DB_NAME = DB_NAME
      THEN
        ERROR_STATUS = 'FWND';
      ELSE
        OBTAIN DUPLICATE RECORD (DATA_ELEMENT);
    END;
    IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
    THEN CALL IDMS_STATUS;
    /*================================================*/
    IF ERROR_STATUS = 'FWND'
    THEN DO;
      /*** VALID DATA ELEMENT BUT DOES IT ALREADY EXIST ***/
      CALL DUP_CK(CONCAT_TBL,CONCAT_FIELD,STATUS);
      IF STATUS = 'GOOD'
      THEN DO;
        /**** SUB ELEMENT DOESN'T ALREADY EXIST  *****/
        NUM_CONCAT = NUM_CONCAT + 1;
        CONCAT_TBL(NUM_CONCAT) = CONCAT_FIELD;
        END;
      ELSE DO;
        /**** SUB ELEMENT DOES ALREADY EXIST  *****/
        PUT STRING (MSG) EDIT (CONCAT_FIELD,
            ' ALREADY EXISTS AS A SUB ELEMENT')
            (2(A));
        CALL MESSAGES;
        END;
      LMNT_NAME = SAVE_NAME;
      RETURN;
      END;
    ELSE DO;
      /** SUB ELEMENT SELECTED ISN'T A VALID DATA ELEMENT **/
      PUT STRING (MSG) EDIT (CONCAT_FIELD,
        ' DATA ELEMENT DOES NOT EXIST - USE MENU') (2(A));
      CALL MESSAGES;
      END;
    END;
  END;
  /** LIST ALL DATA ELEMENTS TO SELECT A SUB ELEMENT *****/
  DISPLAY_TBL = ' ';
  OBTAIN FIRST SET (DEFINED_BY);
  IF ERROR_STATUS = '0307'
  THEN DO;
    MSG = 'NO DATA ELEMENTS TO SELECT FROM !';
    CALL MESSAGES;
    LMNT_NAME = SAVE_NAME;
    RETURN;
```

```
        END;
      COUNT = 0;
      DO WHILE (ERROR_STATUS = REC_FOUND);
        COUNT = COUNT + 1;
        PUT STRING (DISPLAY_TBL(COUNT)) EDIT
          (COUNT,')  ',LMNT_NAME,'  ') (F(3),3(A));
        DOMAIN_TBL(COUNT) = LMNT_NAME;
        OBTAIN NEXT SET (DEFINED_BY);
      END;
      IF ERROR_STATUS ¬= '0307' THEN CALL IDMS_STATUS;
      DISPLAY_TBL(COUNT+1) = '  X)  EXIT';
      CALL GEN_MENU (MENU_NUM,MENU_ENTRY,'* ADD DATA ELEMENT *',
                     COUNT,SLCT_NUM,3);
      IF MENU_NUM ¬= 'X'
      THEN DO;
        /*** VALID DATA ELEMENT BUT DOES IT ALREADY EXIST ***/
        CALL DUP_CK(CONCAT_TBL,DOMAIN_TBL(SLCT_NUM),STATUS);
        IF STATUS = 'GOOD'
        THEN DO;
          /**** SUB ELEMENT DOESN'T ALREADY EXIST  *****/
          NUM_CONCAT = NUM_CONCAT + 1;
          CONCAT_TBL(NUM_CONCAT) = DOMAIN_TBL(SLCT_NUM);
          END;
        ELSE DO;
          /**** SUB ELEMENT DOESN'T ALREADY EXIST  *****/
          PUT STRING (MSG) EDIT (DOMAIN_TBL(SLCT_NUM),
              ' ALREADY EXISTS AS A SUB ELEMENT')
              (2(A));
          CALL MESSAGES;
          END;
        END;
      END;
    ELSE DO;
      /********  REMOVE A SUB ELEMENT  ***********/
      NUM_CONCAT = NUM_CONCAT - 1;
      DO I = (SLCT_NUM-4) TO NUM_CONCAT;
        CONCAT_TBL(I) = CONCAT_TBL(I+1);
      END;
      CONCAT_TBL(NUM_CONCAT+1) = ' ';
      END;
 DUP_CK:    PROC(SRCH_TBL,SRCH_FLD,STATUS);
/*****************************************************************/
/* This module is used when creating a new concatenated DATA_    */
/* ELEMENT occurrence to make sure that the same sub-element     */
/* is not used twice for a single concatenated DATA_ELEMENT      */
/* occurrence.                                                   */
/*****************************************************************/

  /** IF SRCH_FLD IS FOUND IN SRCH_TBL THEN STATUS IS SET TO  ***/
  /** "BAD."  IF IT IS NOT FOUND STATUS IS SET TO "GOOD."     ***/

   DCL            (SRCH_TBL(20),SRCH_FLD)  CHAR(16),
                  STATUS                   CHAR(4),
                  I                        FIXED(3);
```

```
    STATUS = 'GOOD';
    DO I = 1 TO 50;
      IF SRCH_TBL(I) = SRCH_FLD
      THEN DO;
        STATUS = 'BAD';
        RETURN;
        END;
    END;
  END DUP_CK;

  LMNT_NAME = SAVE_NAME;
 END NEW_CONCAT_FIELD;
END NEW_LMNT;

CHG_DEL_LMNT:  PROC (MENU_ENTRY);
/*****************************************************************/
/* This module changes and deletes DATA_ELEMENT fields of the    */
/* "current" DATA_ELEMENT occurrence.                            */
/*****************************************************************/
        DCL       MENU_ENTRY          CHAR(69);

        MSG = 'CHG_DEL_LMNT TO BE COMPLETED';
        CALL MENU_HEAD;
        DISPLAY ('PRESS ENTER TO CONTINUE') REPLY (ENTER_KEY);
        END  CHG_DEL_LMNT;


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

 LMNT_DEF_RTN:    PROC (MENU_ENTRY,LMNT_DEF);
/*****************************************************************/
/* This module allows the data base designer to enter a data     */
/* element definition for the "current" DATA_ELEMENT             */
/* occurrence.                                                   */
/*****************************************************************/

        DCL       MENU_ENTRY              CHAR(69),
                  LMNT_DEF                CHAR(55);


   IF MENU_ENTRY = ' '
   THEN DO;
     MSG = '**  ENTER ELEMENT DEFINITION **';
     CALL MENU_HEAD;
     DISPLAY ('====>') REPLY (LMNT_DEF);
     END;
   ELSE
     LMNT_DEF = MENU_ENTRY;
 END LMNT_DEF_RTN;
```

```
 LMNT_TYPE_RTN:    PROC (MENU_ENTRY,LMNT_TYPE);
/*****************************************************************/
/* This module updates the LMNT_TYPE field in the DATA_ELEMENT   */
/* structure.  If no MENU_ENTRY parameter is inputted into this  */
/* module, a list of possible values will be displayed for the   */
/* data base designer to make a decision.                        */
/*****************************************************************/

        DCL       MENU_ENTRY                CHAR(69),
                  LMNT_TYPE                 CHAR(17),
                  THE_VALUE                 CHAR(16),
                  I                         FIXED(2);

/*   LOAD DISPLAY TABLE AND DOMAIN TABLE                          */

 DISPLAY_TBL(1)='1)   CHARACTER    '; DOMAIN_TBL(1)='CHARACTER    ';
 DISPLAY_TBL(2)='2)   NUMERIC      '; DOMAIN_TBL(2)='NUMERIC      ';
 DISPLAY_TBL(3)='3)   CONCATENATED'; DOMAIN_TBL(3)='CONCATENATED';
 DISPLAY_TBL(4)='X)   EXIT         ';

/* IF NO MENU_ENTRY THEN CALL ROUTINE TO SELECT A VALID ENTRY */

   THE_VALUE = LMNT_TYPE;
   LMNT_TYPE = MENU_ENTRY;
   IF LMNT_TYPE = ' '
   THEN DO;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** ELEMENT TYPE VALUES **',3);
     LMNT_TYPE = THE_VALUE;
     END;
   ELSE DO;

/*  THERE WAS A MENU_ENTRY -- RETURN IF VALID MENU_ENTRY          */

     DO I = 1 TO 3;
       IF DOMAIN_TBL(I) = LMNT_TYPE
       THEN
         RETURN;
     END;

/* THERE MUST HAVE BEEN AN INVALID ENTRY TO GET TO THIS POINT.
WRITE AN ERROR MESSAGE AND CALL ROUTINE TO SELECT A VALID ENTRY */

     PUT STRING (MSG) EDIT (LMNT_TYPE,' IS NOT AN ACCEPTABLE ',
                'VALUE FOR ELEMENT TYPE -- USE MENU')  (3(A));
     CALL MESSAGES;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** ELEMENT TYPE VALUES **',3);
     LMNT_TYPE = THE_VALUE;
     END;
END LMNT_TYPE_RTN;
```

```
 TOTAL_SIZE_RTN:     PROC (MENU_ENTRY,TOTAL_SIZE);
/*****************************************************************/
/* This module updates the TOTAL_SIZE field in the DATA_ELEMENT */
/* structure.  If no MENU_ENTRY parameter is inputted into this */
/* module, a list of possible values will be displayed for the  */
/* data base designer to make a decision.                       */
/*****************************************************************/

   DCL    MENU_ENTRY                  CHAR(69),
          TOTAL_SIZE  PICTURE         '(3)9',
          DIGITS                      CHAR(10) INIT ('0123456789'),
          I                           FIXED(3);

   IF MENU_ENTRY = ' '
   THEN DO;
     PUT STRING (MSG) EDIT
         ('ENTER TOTAL SIZE OF ',LMNT_NAME)  (2(A));
     CALL MESSAGES;
     RETURN;
     END;

   I = 69;
   DO WHILE (SUBSTR(MENU_ENTRY,I,1) = ' ');
     I = I - 1;
   END;

   IF I > 3
   THEN DO;
     MSG = 'ENTRY TO LARGE';
     CALL MESSAGES;
     RETURN;
     END;

   IF VERIFY(SUBSTR(MENU_ENTRY,1,I),DIGITS) = 0
   THEN DO;
     DO WHILE (SUBSTR(MENU_ENTRY,3,1) = ' ');
       SUBSTR(MENU_ENTRY,3,1) = SUBSTR(MENU_ENTRY,2,1);
       SUBSTR(MENU_ENTRY,2,1) = SUBSTR(MENU_ENTRY,1,1);
       SUBSTR(MENU_ENTRY,1,1) = ' ';
     END;
     GET STRING (SUBSTR(MENU_ENTRY,1,3)) EDIT
        (TOTAL_SIZE)  (F(3));
     END;
   ELSE DO;
        MSG = 'NON NUMERIC ENTRY - REENTER';
        CALL MESSAGES;
        END;
END TOTAL_SIZE_RTN;
```

```
 FRCTN_SIZE_RTN:    PROC (MENU_ENTRY,FRACTION_SIZE);
/*****************************************************************/
/* This module updates the FRCTN_SIZE field in the DATA_ELEMENT */
/* structure.  If no MENU_ENTRY parameter is inputted into this */
/* module, a list of possible values will be displayed for the  */
/* data base designer to make a decision.                        */
/*****************************************************************/
   DCL     MENU_ENTRY                CHAR(69),
           FRACTION_SIZE  PICTURE    '(1)9',
           DIGITS                    CHAR(10) INIT ('0123456789'),
           I                         FIXED(3);

   IF MENU_ENTRY = ' '
   THEN DO;
     PUT STRING (MSG) EDIT
         ('ENTER FRACTION SIZE OF ',LMNT_NAME) (2(A));
     CALL MESSAGES;
     RETURN;
     END;

   I = 69;
   DO WHILE (SUBSTR(MENU_ENTRY,I,1) = ' ');
     I = I - 1;
   END;

   IF I > 1
   THEN DO;
     MSG = 'ENTRY TO LARGE';
     CALL MESSAGES;
     RETURN;
     END;

   IF VERIFY(SUBSTR(MENU_ENTRY,1,I),DIGITS) = 0
   THEN DO;
     DO WHILE (SUBSTR(MENU_ENTRY,3,1) = ' ');
       SUBSTR(MENU_ENTRY,3,1) = SUBSTR(MENU_ENTRY,2,1);
       SUBSTR(MENU_ENTRY,2,1) = SUBSTR(MENU_ENTRY,1,1);
       SUBSTR(MENU_ENTRY,1,1) = ' ';
     END;
     GET STRING (SUBSTR(MENU_ENTRY,1,3)) EDIT
        (FRACTION_SIZE) (F(3));
     END;
   ELSE DO;
       MSG = 'NON NUMERIC ENTRY - REENTER';
       CALL MESSAGES;
       END;
 END FRCTN_SIZE_RTN;
END LMNT_UPDATE;
```

```
RECORD_UPDATE:  PROC(MENU_ENTRY);
/*****************************************************************/
/* This module displays all existing RE_CORD occurrences         */
/* for the "current" DATA_BASE structure and allows the user     */
/* to add a new RE_CORD occurrence, change and existing          */
/* RE_CORD occurrence, or delete an existing RE_CORD             */
/* occurrence.                                                   */
/*****************************************************************/
/*                                                               */
/*    RECORD_UPDATE                                              */
/*                                                               */
/*                                                               */
/* +-------------------------------------------------------+     */
/* |  NEW_RECORD                                           |     */
/* | +-----------+  +-----------+  +---------+  +---------+ |   */
/* | | NEW_      |  | REC_      |  | CALC_   |  | REC_    | |   */
/* | | REC_NAME  |  | NEW_MENU  |  | VIA_NEW |  | LMNT_NEW| |   */
/* | +-----------+  +-----------+  +---------+  +---------+ |   */
/* +-------------------------------------------------------+     */
/*                                                               */
/*            +-------------+      +-----------+                 */
/*            | LCTN_       |      | REC_      |                 */
/*            | MODE_RTN    |      | DUP_RTN   |                 */
/*            +-------------+      +-----------+                 */
/*                                                               */
/* +-------------------------------------------------------+     */
/* |  CHG_DEL_REC                                          |     */
/* | +-----------+  +-----------+  +---------+  +---------+ |   */
/* | | REC_      |  | REC_      |  | CALC_   |  | CHG_    | |   */
/* | | CHG_MENU  |  | LMNT_CHG  |  | VIA_CHG |  | REC_NAME| |   */
/* | +-----------+  +-----------+  +---------+  +---------+ |   */
/* +-------------------------------------------------------+     */
/*                                                               */
/*****************************************************************/

        DCL       MENU_ENTRY                CHAR(69),
                  (SLCT_NUM,NUM_REC_LMNTS)  FIXED DEC(3),
                  MENU_NUM                  CHAR(3),
                  REC_LMNT_TBL(500)         CHAR(16);

        IF MENU_ENTRY ¬= ' '
        THEN DO;
          REC_NAME = MENU_ENTRY;
          SAVE_DB_NAME = DB_NAME;
          OBTAIN CALC RECORD (RE_CORD);
          DO WHILE (ERROR_STATUS = REC_FOUND);
            OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
            IF SAVE_DB_NAME = DB_NAME
            THEN
              ERROR_STATUS = 'FWND';
            ELSE
              OBTAIN DUPLICATE RECORD (RE_CORD);
          END;
          IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
          THEN CALL IDMS_STATUS;
```

```
      IF (ERROR_STATUS = 'FWND')
      THEN DO;
        MENU_ENTRY = ' ';
        CALL CHG_DEL_REC (MENU_ENTRY);
        RETURN;
        END;
      ELSE DO;
        DB_NAME = SAVE_DB_NAME;      /********* NEW ********/
        OBTAIN CALC RECORD (DATA_BASE); /****** NEW ***/
        PUT STRING (MSG) EDIT (MENU_ENTRY,
                ' IS NOT AN EXISTING RECORD') (A(16),A);
        CALL MESSAGES;
        END;
      END;

/*    LOAD TABLE WITH RECORDS  NAMES       */
   MENU_NUM = ' ';
   DO WHILE (MENU_NUM ¬= 'X');
     DISPLAY_TBL = ' ';

     PUT STRING (DISPLAY_TBL(1)) EDIT ('1)  CREATE RECORD')
                                      (X(4),A);
     PUT STRING (DISPLAY_TBL(2)) EDIT ('2)  3NF ALL ELEMENTS')
                                      (X(4),A);
     PUT STRING (DISPLAY_TBL(3)) EDIT ('3)  3NF NEW ELEMENTS')
                                      (X(4),A);
     OBTAIN FIRST SET (DIVIDED_INTO);
     IF ERROR_STATUS ¬= '0307'
     THEN CALL IDMS_STATUS;
     COUNT = 3;
     DO WHILE (ERROR_STATUS = REC_FOUND);
       COUNT = COUNT + 1;
       PUT STRING (DISPLAY_TBL(COUNT))
           EDIT (COUNT,')  ',REC_NAME)
                 (X(2),F(3),2(A));
       DB_KEY_TBL(COUNT) = DBKEY;
       OBTAIN NEXT SET (DIVIDED_INTO);
       IF ERROR_STATUS ¬= '0307'
       THEN CALL IDMS_STATUS;
     END;
     PUT STRING (DISPLAY_TBL(COUNT + 1))
         EDIT ('    X)  EXIT') (A);

   /*    DISPLAY MENU   /  ACCEPT EDIT REPLY   */

     CALL GEN_MENU (MENU_NUM,MENU_ENTRY,
          '   ** RECORD UPDATE **',COUNT,SLCT_NUM,3);

   /*  CONTINUE TO UPDATE RECORDS UNTIL EXIT   */

       IF MENU_NUM = 'X'
       THEN
         RETURN;
```

```
          IF SLCT_NUM = 1 THEN
            CALL NEW_RECORD (MENU_ENTRY);
          ELSE IF SLCT_NUM = 2 THEN
            CALL ALL_LMNTS_3NF;
          ELSE IF SLCT_NUM = 3 THEN
            CALL NEW_LMNTS_3NF;
          ELSE DO;
            REC_NAME = SUBSTR(DISPLAY_TBL(SLCT_NUM),9,16);
            OBTAIN RECORD (RE_CORD) DBKEY (DB_KEY_TBL(SLCT_NUM));
            CALL IDMS_STATUS;
            CALL CHG_DEL_REC (MENU_ENTRY);
            END;
      END;

NEW_RECORD:  PROC(MENU_ENTRY);
/*****************************************************************/
/* This module establishes a new RE_CORD occurrence by           */
/* defaulting the RE_CORD occurrence fields and calling the      */
/* modules to update the fields based on the user's MENU_ENTRY.  */
/* Once updating is complete, the new RE_CORD occurrence is      */
/* stored.                                                       */
/*****************************************************************/

   DCL            MENU_ENTRY                    CHAR(69),
                  MENU_NUM                      CHAR(3),
                  (SLCT_NUM,I)                  FIXED DEC (3);

/*    INITIALIZE AND DEFAULT RECORD FIELDS    */

   REC_LMNT_TBL = ' ';
   NUM_REC_LMNTS = 0;
   RE_CORD = ' ';
   REC_NAME = ' ';
   REC_STRG_MODE = 'F';
   REC_DUP_OPTION = 'DN';
   REC_LCTN_MODE = 'CALC';
   REC_CALC_VIA = ' ';

/*  HAS A RECORD NAME BEEN PROVIDED FROM SECONDARY MENU      */

   IF MENU_ENTRY = ' '
   THEN
     CALL REC_NEW_MENU (MENU_NUM,MENU_ENTRY,SLCT_NUM);
   ELSE
     MENU_NUM = '1';

/*  ADD RECORD ATTRIBUTES UNTIL EXIT                         */

   DO WHILE (MENU_NUM ¬= 'X');
     IF MENU_NUM = '1' THEN
       CALL NEW_REC_NAME (MENU_ENTRY,REC_NAME);
     ELSE IF MENU_NUM = '2' THEN
       CALL STRG_MODE_RTN (MENU_ENTRY,REC_STRG_MODE);
     ELSE IF MENU_NUM = '3' THEN
```

```
        CALL LCTN_MODE_RTN (MENU_ENTRY,REC_LCTN_MODE);
      ELSE IF REC_LCTN_MODE = 'CALC'
      THEN DO;
        IF MENU_NUM = '4' THEN
          CALL REC_DUP_RTN (MENU_ENTRY,REC_DUP_OPTION);
        ELSE IF MENU_NUM = '5' THEN
          CALL CALC_VIA_NEW (MENU_ENTRY,REC_CALC_VIA);
        ELSE IF MENU_NUM >= '6' THEN
          CALL REC_LMNT_NEW (SLCT_NUM,MENU_ENTRY);
        END;
      ELSE DO;
        IF MENU_NUM = '4' THEN
          CALL CALC_VIA_NEW (MENU_ENTRY,REC_CALC_VIA);
        ELSE IF MENU_NUM >= '5' THEN
          CALL REC_LMNT_NEW (SLCT_NUM,MENU_ENTRY);
        END;
      CALL REC_NEW_MENU (MENU_NUM,MENU_ENTRY,SLCT_NUM);
    END;

    IF REC_NAME = ' '
    THEN DO;
      MSG = 'RECORD NAME IS BLANK -- NO ADD MADE';
      CALL MESSAGES;
      RETURN;
      END;

    STORE RECORD (RE_CORD);

    DO I = 1 TO NUM_REC_LMNTS;
      LMNT_NAME = REC_LMNT_TBL(I);
      SAVE_DB_NAME = DB_NAME;
      OBTAIN CALC RECORD (DATA_ELEMENT);  CALL IDMS_STATUS;
      OBTAIN OWNER SET (DEFINED_BY);      CALL IDMS_STATUS;
      DO WHILE (SAVE_DB_NAME ¬= DB_NAME);
        OBTAIN CALC RECORD (DATA_ELEMENT);  CALL IDMS_STATUS;
        OBTAIN OWNER SET (DEFINED_BY);      CALL IDMS_STATUS;
      END;
      STORE RECORD (LMNT_REC);  CALL IDMS_STATUS;
    END;

NEW_REC_NAME:  PROC (MENU_ENTRY,REC_NAME);
/*****************************************************************/
/* This module allows the user to assign a record name to a      */
/* newly created RE_CORD occurrence.  Before the record name     */
/* is accepted, a check is made to verify it does not already    */
/* exist.                                                        */
/*****************************************************************/

    DCL       MENU_ENTRY                CHAR(69),
              (REC_NAME,SAVE_NAME)      CHAR(16),
              STATUS                    CHAR(4);

    REC_NAME = MENU_ENTRY;
    CALL EDIT_NAME (REC_NAME,STATUS);
```

```
    IF STATUS = 'GOOD'
    THEN DO;
      SAVE_NAME = REC_NAME;
      SAVE_DB_NAME = DB_NAME;
      OBTAIN CALC RECORD (RE_CORD);
      DO WHILE (ERROR_STATUS = REC_FOUND);
        OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
        IF SAVE_DB_NAME = DB_NAME
        THEN
          ERROR_STATUS = 'FWND';
        ELSE
          OBTAIN DUPLICATE RECORD (RE_CORD);
      END;
      IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
      THEN CALL IDMS_STATUS;

      IF ERROR_STATUS = 'FWND'
      THEN DO;
        PUT STRING (MSG) EDIT (REC_NAME,' ALREADY EXISTS')(2(A));
        CALL MESSAGES;
        REC_NAME = ' ';
        END;
      ELSE DO;
        DB_NAME = SAVE_DB_NAME;
        OBTAIN CALC RECORD (DATA_BASE);
        END;
      END;
    ELSE
      REC_NAME = ' ';
  END NEW_REC_NAME;

 REC_NEW_MENU:  PROC (MENU_NUM,MENU_ENTRY,SLCT_NUM);
/*****************************************************************/
/* This module displays the RE_CORD fields that can be updated  */
/* for a new RE_CORD occurrence.  It also displays the DATA_     */
/* ELEMENT occurrences that are liked to this RE_CORD occurrence*/
/* for updating (i.e., add or delete).                           */
/*****************************************************************/

        DCL       MENU_NUM              CHAR(3),
                  MENU_ENTRY            CHAR(69),
                  (I,SLCT_NUM)          FIXED DEC(3);

     /*  LOAD DISPLAY TABLE  */


    DISPLAY_TBL = ' ';
    PUT STRING (DISPLAY_TBL(1)) EDIT
     ('1) RECORD NAME: ',REC_NAME)(2(A));
    PUT STRING (DISPLAY_TBL(2)) EDIT
     ('2) RECORD STORAGE MODE: ',REC_STRG_MODE)(2(A));
    PUT STRING (DISPLAY_TBL(3)) EDIT
     ('3) RECORD LOCATION MODE: ',REC_LCTN_MODE)(2(A));
    IF REC_LCTN_MODE = 'CALC'
```

```
    THEN DO;
      COUNT = 7;
      PUT STRING (DISPLAY_TBL(4)) EDIT
        ('4) RECORD DUPLICATE OPTION: ',REC_DUP_OPTION)(2(A));
      PUT STRING (DISPLAY_TBL(5)) EDIT
        ('5) RECORD CALC KEY OR VIA SET: ',REC_CALC_VIA)(2(A));
      PUT STRING (DISPLAY_TBL(6)) EDIT
        ('6) ADD DATA ELEMENT TO ',REC_NAME)(2(A));
      END;
    ELSE DO;
      COUNT = 6;
      PUT STRING (DISPLAY_TBL(4)) EDIT
        ('4) RECORD CALC KEY OR VIA SET: ',REC_CALC_VIA)(2(A));
      PUT STRING (DISPLAY_TBL(5)) EDIT
        ('5) ADD DATA ELEMENT TO ',REC_NAME)(2(A));
      DISPLAY_TBL(6) = 'DELETE DATA ELEMENT . . .';
      END;
    DO I = 1 TO NUM_REC_LMNTS;
      COUNT = COUNT + 1;
      PUT STRING (DISPLAY_TBL(COUNT)) EDIT
       (COUNT-1,') ',REC_LMNT_TBL(I)) (X(2),F(3),2(A));
    END;
    DISPLAY_TBL(COUNT+1) = 'X) EXIT';
    CALL GEN_MENU (MENU_NUM,MENU_ENTRY,'** CREATE NEW RECORD **',
                   COUNT-1,SLCT_NUM,2);
  END REC_NEW_MENU;

CALC_VIA_NEW:     PROC (MENU_ENTRY,REC_CALC_VIA);
/*****************************************************************/
/* This module assigns a value to the REC_CALC_VIA field of a    */
/* new RE_CORD occurrence.  Because it is a new RE_CORD          */
/* occurrence, it cannot be a member of any set at this time.    */
/* Thus, a proper via set value is not allowed.  The calc value */
/* can be any DATA_ELEMENT occurrence associated with this       */
/* RE_CORD occurrence.                                           */
/*****************************************************************/
        DCL       MENU_NUM                        CHAR(3),
                  MENU_ENTRY                      CHAR(69),
                  REC_CALC_VIA                    CHAR(16),
                  (SLCT_NUM,I)                    FIXED DEC(3);

 /*     VIA SET IS NOT POSSIBLE FOR A NEW RECORD          */

   IF REC_LCTN_MODE = 'VIA'
   THEN DO;
     MSG = 'RECORD IS NOT MEMBER OF ANY SET - ',
           'NO ACCEPTABLE VALUE';
     CALL MESSAGES;
     RETURN;
     END;

/* SEE IF MENU_ENTRY IS A RECORD DATA ELEMENT  */

   DO I = 1 TO NUM_REC_LMNTS;
```

```
      IF REC_LMNT_TBL(I) = MENU_ENTRY
      THEN DO;
        REC_LCTN_MODE = 'CALC';     /*  IN CASE IT WAS BLANK  */
        REC_CALC_VIA = MENU_ENTRY;
        RETURN;
        END;
    END;
    DISPLAY_TBL(I) = '  X)  EXIT';

/*  ARE THERE DATA ELEMENTS FOR A CALC KEY                    */

    IF NUM_REC_LMNTS = '0'
    THEN DO;
      MSG = 'RECORD CONTAINS NO DATA ELEMENTS - " ',
            'NO ACCEPTABLE VALUE';
      CALL MESSAGES;
      RETURN;
      END;

/*  DISPLAY DATA ELEMENTS FOR CALC KEY SOLUTION               */

    CALL GEN_MENU (MENU_NUM,MENU_ENTRY,'** SELECT CALC KEY **',
                   NUM_REC_LMNTS,SLCT_NUM,3);
    IF MENU_NUM = 'X'
    THEN
      RETURN;
    REC_CALC_VIA = REC_LMNT_TBL(SLCT_NUM);
  END CALC_VIA_NEW;

REC_LMNT_NEW:  PROC (SLCT_NUM,MENU_ENTRY);
/*****************************************************************/
/* This module links existing DATA_ELEMENT occurrences to a      */
/* new RE_CORD occurrence.  If the MENU_ENTRY parameter does     */
/* not contain a valid data element name then a complete list    */
/* of data element names is displayed for the user to select     */
/* from.                                                         */
/*****************************************************************/

    DCL             MENU_NUM            CHAR(3),
                    MENU_ENTRY          CHAR(69),
                    (SLCT_NUM,START)    FIXED(3);

    IF (SLCT_NUM = 5 | (SLCT_NUM = 6 & REC_LCTN_MODE = 'CALC'))
    THEN DO;
      IF MENU_ENTRY ¬= ' '
      THEN DO;
        LMNT_NAME = MENU_ENTRY;
        SAVE_DB_NAME = DB_NAME;
        OBTAIN CALC RECORD (DATA_ELEMENT);
        DO WHILE (ERROR_STATUS = REC_FOUND);
          OBTAIN OWNER SET (DEFINED_BY);  CALL IDMS_STATUS;
          IF SAVE_DB_NAME = DB_NAME
          THEN
            ERROR_STATUS = 'FWND';
```

```
      ELSE
        OBTAIN DUPLICATE RECORD (DATA_ELEMENT);
    END;
    IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
    THEN CALL IDMS_STATUS;
    IF ERROR_STATUS = 'FWND'
    THEN DO;
      NUM_REC_LMNTS = NUM_REC_LMNTS + 1;
      REC_LMNT_TBL(NUM_REC_LMNTS) = LMNT_NAME;
      RETURN;
      END;
    ELSE DO;
      PUT STRING (MSG) EDIT
      (LMNT_NAME,
      ' DATA ELEMENT DOES NOT EXIST - USE MENU')(2(A));
      CALL MESSAGES;
      END;
    END;

  /*******  LIST ALL DATA ELEMENTS TO SELECT FROM     *******/
  DISPLAY_TBL = ' ';
  OBTAIN FIRST SET (DEFINED_BY);
  IF ERROR_STATUS = '0307'
  THEN DO;
    MSG = 'NO DATA ELEMENTS TO CHOOSE FROM!';
    CALL MESSAGES;
    RETURN;
    END;
  COUNT = 0;
  DO WHILE (ERROR_STATUS = REC_FOUND);
    COUNT = COUNT + 1;
    PUT STRING (DISPLAY_TBL(COUNT)) EDIT
              (COUNT,')  ',LMNT_NAME,'  ')(F(3),3(A));
    DOMAIN_TBL(COUNT) = LMNT_NAME;
    OBTAIN NEXT SET (DEFINED_BY);
  END;
  IF ERROR_STATUS ¬= '0307'
  THEN CALL IDMS_STATUS;
  DISPLAY_TBL(COUNT+1) = '  X)  EXIT';
  CALL GEN_MENU (MENU_NUM,MENU_ENTRY,
     '** ADD DATA ELEMENT TO A RECORD **',COUNT,SLCT_NUM,3);

  IF MENU_NUM ¬= 'X'
  THEN DO;
      NUM_REC_LMNTS = NUM_REC_LMNT + 1;
      REC_LMNT_TBL(NUM_REC_LMNTS) = DOMAIN_TBL(SLCT_NUM);
    END;
  END;
ELSE DO;
  NUM_REC_LMNTS = NUM_REC_LMNT - 1;
  IF REC_LCTN_MODE = 'CALC'
  THEN
    START = SLCT_NUM - 6;
  ELSE
```

```
        START = SLCT_NUM - 5;
      DO I = START TO NUM_REC_LMNTS;
        REC_LMNT_TBL(I) = REC_LMNT_TBL(I+1);
      END;
      END;
 END REC_LMNT_NEW;
END NEW_RECORD;

 CHG_DEL_REC:  PROC (MENU_ENTRY);
/*****************************************************************/
/* This module changes and deletes RE_CORD fields of the         */
/* "current" RE_CORD occurrence.                                 */
/*****************************************************************/

   DCL              DEL_SW              CHAR(1),
                    SAVE_NAME           CHAR(16),
                    MENU_NUM            CHAR(3),
                    MENU_ENTRY          CHAR(69);

   IF MENU_ENTRY = 'DELETE'
   THEN DO;
     FIND CURRENT RECORD (RE_CORD);     CALL IDMS_STATUS;
     ERASE RECORD (RE_CORD) PERMANENT; CALL IDMS_STATUS;
     RETURN;
     END;

   SAVE_NAME = REC_NAME;
   CALL REC_CHG_MENU (MENU_NUM,MENU_ENTRY,SLCT_NUM);
   DO WHILE (MENU_NUM ¬= 'X');
     IF MENU_NUM = '1' THEN
       CALL CHG_REC_NAME (MENU_ENTRY,REC_NAME);
     ELSE IF MENU_NUM = '2' THEN
       CALL STRG_MODE_RTN (MENU_ENTRY,REC_STRG_MODE);
     ELSE IF MENU_NUM = '3' THEN
       CALL LCTN_MODE_RTN (MENU_ENTRY,REC_LCTN_MODE);
     ELSE IF REC_LCTN_MODE = 'CALC'
     THEN DO;
       IF MENU_NUM = '4' THEN
         CALL REC_DUP_RTN (MENU_ENTRY,REC_DUP_OPTION);
       ELSE IF MENU_NUM = '5' THEN
         CALL CALC_VIA_CHG (MENU_ENTRY,REC_CALC_VIA);
       ELSE IF MENU_NUM >= '6' THEN
         CALL REC_LMNT_CHG (SLCT_NUM,MENU_ENTRY);
       END;
     ELSE DO;
       IF MENU_NUM = '4' THEN
         CALL CALC_VIA_CHG (MENU_ENTRY,REC_CALC_VIA);
       ELSE IF MENU_NUM >= '5' THEN
         CALL REC_LMNT_CHG (SLCT_NUM,MENU_ENTRY);
       END;
     CALL REC_CHG_MENU (MENU_NUM,MENU_ENTRY,SLCT_NUM);
   END;

   IF REC_NAME = ' '
```

```
    THEN DO;
      CALL CLRSCR;
      CALL BLANK_LINE(5);
      DISPLAY ('CONFIRM DELETE BY TYPING "D"');
      CALL BLANK_LINE(5);
      DISPLAY ('=====>')  REPLY (DEL_SW);
      IF DEL_SW = 'D'
      THEN DO;
        FIND CURRENT RECORD (RE_CORD);    CALL IDMS_STATUS;
        ERASE RECORD (RE_CORD) PERMANENT; CALL IDMS_STATUS;
        RETURN;
        END;
      ELSE DO;
        MSG = 'DELETE REQUEST ABORTED';
        CALL MESSAGES;
        REC_NAME = SAVE_NAME;
        END;
      END;
    ELSE DO;
      FIND CURRENT RECORD (RE_CORD);    CALL IDMS_STATUS;
      MODIFY RECORD (RE_CORD);          CALL IDMS_STATUS;
      END;

CHG_REC_NAME:     PROC (MENU_ENTRY,REC_NAME);
/*****************************************************************/
/* This module changes the record name of the "current" RE_CORD */
/* occurrence based on the MENU_ENTRY.                           */
/*****************************************************************/

    DCL       MENU_ENTRY                CHAR(69),
              REC_NAME                  CHAR(16),
              STATUS                    CHAR(4);

    REC_NAME = MENU_ENTRY;
    CALL EDIT_NAME (REC_NAME,STATUS);
    IF STATUS = 'GOOD'
    THEN DO;
      IF SAVE_NAME = REC_NAME
      THEN
        ENTER_KEY = ' ';
      ELSE DO;
        SAVE_DB_NAME = DB_NAME;
        OBTAIN CALC RECORD (RE_CORD);
        DO WHILE (ERROR_STATUS = REC_FOUND);
          OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
          IF SAVE_DB_NAME = DB_NAME
          THEN
            ERROR_STATUS = 'FWND';
          ELSE
            OBTAIN DUPLICATE RECORD (RE_CORD);
        END;
        IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
        THEN CALL IDMS_STATUS;
```

```
        IF ERROR_STATUS = 'FWND'
        THEN DO;
          PUT STRING (MSG) EDIT (REC_NAME,' ALREADY EXISTS')(2(A));
          CALL MESSAGES;
          REC_NAME = SAVE_NAME;
          END;
        ELSE DO;
          DB_NAME = SAVE_DB_NAME;
          OBTAIN CALC RECORD (DATA_BASE);
          IF REC_NAME ¬= ' '
          THEN
            SAVE_NAME = REC_NAME;
          END;
        END;
      END;
    ELSE
      REC_NAME = SAVE_NAME;
  END CHG_REC_NAME;

REC_CHG_MENU:   PROC (MENU_NUM,MENU_ENTRY,SLCT_NUM);
/*****************************************************************/
/* This module displays the RE_CORD fields that can be updated */
/* for an existing RE_CORD occurrence.  It also displays the    */
/* DATA_ELEMENT occurrences that are linked to this RE_CORD     */
/* occurrence for updating (i.e., add or delete).               */
/*****************************************************************/

        DCL       MENU_NUM            CHAR(3),
                  MENU_ENTRY          CHAR(69),
                  SLCT_NUM            FIXED(3);

/*   LOAD DISPLAY TABLE    */

    DISPLAY_TBL = ' ';
    PUT STRING (DISPLAY_TBL(1)) EDIT
            ('1) RECORD NAME: ',REC_NAME)(2(A));
    PUT STRING (DISPLAY_TBL(2)) EDIT
            ('2) RECORD STORAGE MODE: ',REC_STRG_MODE)(2(A));
    PUT STRING (DISPLAY_TBL(3)) EDIT
            ('3) RECORD LOCATION MODE: ',
                REC_LCTN_MODE)(2(A));
    IF REC_LCTN_MODE = 'CALC'
    THEN DO;
      COUNT = 7;
      PUT STRING (DISPLAY_TBL(4)) EDIT
        ('4) RECORD DUPLICATE OPTION: ',REC_DUP_OPTION)(2(A));
      PUT STRING (DISPLAY_TBL(5)) EDIT
        ('5) RECORD CALC KEY OR VIA SET: ',
            REC_CALC_VIA)(2(A));
      PUT STRING (DISPLAY_TBL(6)) EDIT
        ('6) ADD DATA ELEMENT TO ',REC_NAME)(2(A));
      DISPLAY_TBL(7) = 'DELETE DATA ELEMENT . . .';
      END;
    ELSE DO;
```

```
    COUNT = 6;
    PUT STRING (DISPLAY_TBL(4)) EDIT
      ('4) RECORD CALC KEY OR VIA SET: ',
           REC_CALC_VIA) (2(A));
    PUT STRING (DISPLAY_TBL(5)) EDIT
      ('5) ADD DATA ELEMENT TO ',REC_NAME) (2(A));
    END;
  OBTAIN FIRST SET (POPULATED_WITH);
  IF ERROR_STATUS ¬= '0307'
  THEN DO;
    CALL IDMS_STATUS;
    OBTAIN OWNER SET (GROUPED_IN);  CALL IDMS_STATUS;
    END;
  DO WHILE (ERROR_STATUS = OK);
    COUNT = COUNT + 1;
    PUT STRING (DISPLAY_TBL(COUNT)) EDIT
               (COUNT-1,') ',LMNT_NAME) (X(2),F(3),2(A));
    OBTAIN NEXT SET (POPULATED_WITH);
    IF ERROR_STATUS ¬= '0307'
    THEN DO;
      CALL IDMS_STATUS;
      OBTAIN OWNER SET (GROUPED_IN);  CALL IDMS_STATUS;
      END;
  END;
  DISPLAY_TBL(COUNT+1) = 'X) EXIT';
  CALL GEN_MENU (MENU_NUM,MENU_ENTRY,
       '** CHANGE OR DELETE RECORD **',COUNT-1,SLCT_NUM,2);
END REC_CHG_MENU;

CALC_VIA_CHG:    PROC (MENU_ENTRY,REC_CALC_VIA);
/*************************************************************/
/*  This module changes the value of an existing RE_CORD     */
/*  occurrence's REC_CALC_VIA field based on the value of    */
/*  the MENU_ENTRY.  This module assures that the value      */
/*  assigned to REC_CALC_VIA is an acceptable value based    */
/*  the REC_LCTN_MODE field.                                 */
/*************************************************************/

   DCL     MENU_NUM                                  CHAR(3),
           MENU_ENTRY                                CHAR(69),
           REC_CALC_VIA                              CHAR(16),
           (CALC_SLCT_TBL(500),VIA_SLCT_TBL(500))  CHAR(24),
           (SLCT_NUM,NUM_CALC,NUM_VIA)             FIXED (3);

  /*Load VIA_SLCT_TBL with valid VIA Set values */
  VIA_SLCT_TBL  = ' ';
  NUM_VIA = 0;
  OBTAIN FIRST SET (MEMBER_OF);
  DO WHILE (ERROR_STATUS = OK);
    NUM_VIA = NUM_VIA + 1;
    IF MENU_ENTRY = SET_NAME
    THEN DO;
      REC_LCTN_MODE = 'VIA';
      REC_CALC_VIA = MENU_ENTRY;
```

```
    RETURN;
    END;
  ELSE DO;
    PUT STRING (VIA_SLCT_TBL(NUM_VIA)) EDIT
      (NUM_VIA,')   ',SET_NAME,'   ') (F(3),3(A));
    OBTAIN NEXT SET (MEMBER_OF);
    END;
END;
VIA_SLCT_TBL(NUM_VIA+1) = '  X)   EXIT';
IF ERROR_STATUS ¬= '0307' THEN CALL IDMS_STATUS;

/*  Load CALC_SLCT_TBL with valid CALC_KEY values */
CALC_SLCT_TBL  = ' ';
NUM_CALC = 0;
OBTAIN FIRST SET (POPULATED_WITH);
IF ERROR_STATUS ¬= '0307'
THEN DO;
  CALL IDMS_STATUS;
  OBTAIN OWNER SET (GROUPED_IN);  CALL IDMS_STATUS;
  END;
DO WHILE (ERROR_STATUS = OK);
  NUM_CALC = NUM_CALC + 1;
  IF MENU_ENTRY = LMNT_NAME
  THEN DO;
    REC_LCTN_MODE = 'CALC';
    REC_CALC_VIA = MENU_ENTRY;
    RETURN;
    END;
  ELSE DO;
    PUT STRING (CALC_SLCT_TBL(NUM_CALC)) EDIT
      (NUM_CALC,')   ',LMNT_NAME,'   ') (F(3),3(A));
    OBTAIN NEXT SET (POPULATED_WITH);
    IF ERROR_STATUS ¬= '0307'
    THEN DO;
      CALL IDMS_STATUS;
      OBTAIN OWNER SET (GROUPED_IN);  CALL IDMS_STATUS;
      END;
    END;
END;
CALC_SLCT_TBL(NUM_CALC+1) = '  X)   EXIT';

IF REC_LCTN_MODE = ' '
THEN DO;
  MSG = 'MAKE AN ENTRY IN RECORD LOCATION MODE FIRST';
  CALL MESSAGES;
  RETURN;
  END;

IF MENU_ENTRY  ¬= ' '
THEN DO;
  PUT STRING (MSG) EDIT (MENU_ENTRY,
          ' IS INVALID CALC KEY OF VIA SET - USE MENU')
            (A(16),A);
  CALL MESSAGES;
```

```
      END;

    /*  Display valid CALC_KEY fields if any */
    IF REC_LCTN_MODE = 'CALC'
    THEN DO;
      IF NUM_CALC = 0
      THEN DO;
        MSG='RECORD CONTAINS NO DATA ELEMENTS -
             NO ACCEPTABLE VALUE';
        CALL MESSAGES;
        RETURN;
        END;
      DISPLAY_TBL = CALC_SLCT_TBL;
      CALL GEN_MENU (MENU_NUM,MENU_ENTRY,
                     '** SELECT CALC KEY **',
                     NUM_CALC,SLCT_NUM,3);
      IF MENU_NUM = 'X'
      THEN DO;
        REC_CALC_VIA = ' ';
        RETURN;
        END;
      REC_CALC_VIA = SUBSTR(CALC_SLCT_TBL(SLCT_NUM),7,16);
      END;
    ELSE DO;

      /*  Dispaly valid VIA_SE fields if any */
      IF NUM_VIA = 0
      THEN DO;
        MSG='RECORD IS NOT MEMBER OF ANY SET -
             NO ACCEPTABLE VALUE';
        CALL MESSAGES;
        RETURN;
        END;
      DISPLAY_TBL = VIA_SLCT_TBL;
      CALL GEN_MENU (MENU_NUM,MENU_ENTRY,
                     '** SELECT VIA SET **',
                     NUM_VIA,SLCT_NUM,3);
      IF MENU_NUM = 'X'
      THEN DO;
        REC_CALC_VIA = ' ';
        RETURN;
        END;
      REC_CALC_VIA = SUBSTR(VIA_SLCT_TBL(SLCT_NUM),7,16);
      END;
  END CALC_VIA_CHG;

REC_LMNT_CHG:  PROC (SLCT_NUM,MENU_ENTRY);
/*************************************************************/
/* This module links existing DATA_ELEMENT occurrences to    */
/* an existing RE_CORD occurrence.  If the MENU_ENTRY        */
/* parameter does not contain a valid data element name      */
/* then a complete list of data element names is displayed   */
/* for the user to select from.                              */
/*************************************************************/
```

```
DCL             MENU_NUM            CHAR(3),
                MENU_ENTRY          CHAR(69),
                SLCT_NUM            FIXED(3);

IF (SLCT_NUM = 5 | (SLCT_NUM = 6 &
                    REC_LCTN_MODE = 'CALC'))
THEN DO;
  IF MENU_ENTRY ¬= ' '
  THEN DO;
    LMNT_NAME = MENU_ENTRY;
    SAVE_DB_NAME = DB_NAME;
    OBTAIN CALC RECORD (DATA_ELEMENT);
    DO WHILE (ERROR_STATUS = REC_FOUND);
      OBTAIN OWNER SET (DEFINED_BY);  CALL IDMS_STATUS;
      IF SAVE_DB_NAME = DB_NAME
      THEN
        ERROR_STATUS = 'FWND';
      ELSE
        OBTAIN DUPLICATE RECORD (DATA_ELEMENT);
    END;
    IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
    THEN CALL IDMS_STATUS;
    IF ERROR_STATUS = 'FWND'
    THEN DO;
      STORE RECORD (LMNT_REC);  CALL IDMS_STATUS;
      RETURN;
      END;
    ELSE DO;
      PUT STRING (MSG) EDIT
      (LMNT_NAME,
       ' DATA ELEMENT DOES NOT EXIST - USE MENU')(2(A));
      CALL MESSAGES;
      END;
    END;

  /*****  LIST ALL DATA ELEMENTS TO SELECT FROM ******/
  DISPLAY_TBL = ' ';
  OBTAIN FIRST SET (DEFINED_BY);
  IF ERROR_STATUS = '0307'
  THEN DO;
    MSG = 'NO DATA ELEMENTS TO CHOOSE FROM!';
    CALL MESSAGES;
    RETURN;
    END;
  COUNT = 0;
  DO WHILE (ERROR_STATUS = REC_FOUND);
    COUNT = COUNT + 1;
    PUT STRING (DISPLAY_TBL(COUNT)) EDIT
               (COUNT,')  ',LMNT_NAME,'  ')(F(3),3(A));
    DOMAIN_TBL(COUNT) = LMNT_NAME;
    OBTAIN NEXT SET (DEFINED_BY);
  END;
  IF ERROR_STATUS ¬= '0307'
```

```
      THEN CALL IDMS_STATUS;
      DISPLAY_TBL(COUNT+1) = '  X)  EXIT';

      CALL GEN_MENU (MENU_NUM,MENU_ENTRY,
         '** ADD DATA ELEMENT TO A RECORD **',
          COUNT,SLCT_NUM,3);

      IF MENU_NUM ¬= 'X'
      THEN DO;
        LMNT_NAME = DOMAIN_TBL(SLCT_NUM);
        FIND CALC RECORD (DATA_ELEMENT);
        STORE RECORD (LMNT_REC);  CALL IDMS_STATUS;
        END;
      END;
    ELSE DO;
      LMNT_NAME = SUBSTR(DISPLAY_TBL(SLCT_NUM+1),8,16);
/**   FIND CALC RECORD (LMNT_REC);  CONFLICT W/ SCHEMA==> VIA */
      ERASE RECORD (LMNT_REC) PERMANENT;  CALL IDMS_STATUS;
      END;
  END REC_LMNT_CHG;
END  CHG_DEL_REC;

STRG_MODE_RTN:  PROC (MENU_ENTRY,REC_STRG_MODE);
/*****************************************************************/
/* This module updates the REC_STRG_MODE field in the RE_CORD    */
/* structure.  If no MENU_ENTRY parameter is inputted into this */
/* module, a list of possible values will be displayed for the   */
/* data base designer to make a decision.                        */
/*****************************************************************/

   DCL       MENU_ENTRY                CHAR(69),
             REC_STRG_MODE             CHAR(2),
             THE_VALUE                 CHAR(16),
             I                         FIXED DEC(2);

/*   LOAD DISPLAY TABLE AND DOMAIN TABLE                          */

 DISPLAY_TBL(1) = '1)  FIXED              ';  DOMAIN_TBL(1) = 'F';
 DISPLAY_TBL(2) = '2)  VARAIBLE           ';  DOMAIN_TBL(2) = 'V';
 DISPLAY_TBL(3) = '3)  COMPRESSED         ';  DOMAIN_TBL(3) = 'C';
 DISPLAY_TBL(4) = 'X)  EXIT               ';

/*  IF NO MENU_ENTRY THEN CALL ROUTINE TO SELECT A VALID ENTRY  */

   THE_VALUE = REC_STRG_MODE;
   REC_STRG_MODE = MENU_ENTRY;
   IF REC_STRG_MODE = ' '
   THEN DO;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
          '** RECORD STORAGE MODES **',3);
     REC_STRG_MODE = THE_VALUE;
     END;
   ELSE DO;
```

```
/*  THERE WAS A MENU_ENTRY -- RETURN IF VALID MENU_ENTRY          */

     DO I = 1 TO 3;
       IF DOMAIN_TBL(I) = REC_STRG_MODE
       THEN
         RETURN;
     END;

/*THERE MUST HAVE BEEN AN INVALID ENTRY TO GET TO THIS POINT.
  WRITE AN ERROR MESSAGE AND CALL ROUTINE TO SELECT A VALID ENTRY*/

     PUT STRING (MSG) EDIT (REC_STRG_MODE,' IS NOT ACCEPTABLE ',
         'FOR RECORD STORAGE MODE -- USE MENU')  (3(A));
     CALL MESSAGES;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** RECORD STORAGE MODES **',3);
     REC_STRG_MODE = THE_VALUE;
     END;
END STRG_MODE_RTN;

LCTN_MODE_RTN:  PROC (MENU_ENTRY,REC_LCTN_MODE);
/*********************************************************************/
/* This module updates the REC_LCTN_MODE field in the RE_CORD    */
/* structure.  If no MENU_ENTRY parameter is inputted into this */
/* module, a list of possible values will be displayed for the  */
/* data base designer to make a decision.                        */
/*********************************************************************/

   DCL       MENU_ENTRY                CHAR(69),
             REC_LCTN_MODE             CHAR(4),
             (THE_VALUE,SAVE_VALUE)    CHAR(16),
             I                         FIXED DEC(2);

/*   LOAD DISPLAY TABLE AND DOMAIN TABLE                          */

 DISPLAY_TBL(1) = '1)  CALC KEY      ';  DOMAIN_TBL(1) = 'CALC';
 DISPLAY_TBL(2) = '2)  VIA SET       ';  DOMAIN_TBL(2) = 'VIA';
 DISPLAY_TBL(3) = 'X)  EXIT          ';

/*  IF NO MENU_ENTRY THEN CALL ROUTINE TO SELECT A VALID ENTRY  */

   SAVE_VALUE = REC_LCTN_MODE;
   THE_VALUE =  REC_LCTN_MODE;
   REC_LCTN_MODE = MENU_ENTRY;
   IF REC_LCTN_MODE = ' '
   THEN DO;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** RECORD LOCATION MODES **',2);
     REC_LCTN_MODE = THE_VALUE;
     END;
   ELSE DO;

/*  THERE WAS A MENU_ENTRY -- RETURN IF VALID MENU_ENTRY          */
```

```
      DO I = 1 TO 2;
        IF DOMAIN_TBL(I) = REC_LCTN_MODE
        THEN
          RETURN;
      END;

/* THERE MUST HAVE BEEN AN INVALID ENTRY TO GET TO THIS POINT.
 WRITE AN ERROR MESSAGE AND CALL ROUTINE TO SELECT A VALID ENTRY*/

      PUT STRING (MSG) EDIT (REC_LCTN_MODE,' IS NOT ACCEPTABLE ',
         'FOR RECORD LOCATION MODE -- USE MENU')   (3(A));
      CALL MESSAGES;
      CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
           '** RECORD LOCATION MODES **',2);
      REC_LCTN_MODE = THE_VALUE;
      END;
      IF SAVE_VALUE ¬= REC_LCTN_MODE
      THEN
        REC_CALC_VIA = ' ';
END LCTN_MODE_RTN;

REC_DUP_RTN:  PROC (MENU_ENTRY,REC_DUP_OPTION);
/*****************************************************************/
/* This module updates the REC_DUP_OPTION field in the RE_CORD   */
/* structure.  If no MENU_ENTRY parameter is inputted into this */
/* module, a list of possible values will be displayed for the   */
/* data base designer to make a decision.                        */
/*****************************************************************/

   DCL        MENU_ENTRY                CHAR(69),
              REC_DUP_OPTION            CHAR(2),
              THE_VALUE                 CHAR(16),
              I                         FIXED DEC(2);

/* LOAD DISPLAY TABLE AND DOMAIN TABLE                           */

 DISPLAY_TBL(1) = '1'   DUPLICATES FIRST      ';
                        DOMAIN_TBL(1) = 'DF';
 DISPLAY_TBL(2) = '2'   DUPLICATES LAST       ';
                        DOMAIN_TBL(2) = 'DL';
 DISPLAY_TBL(3) = '3'   DUPLICATES NOT ALLOWED';
                        DOMAIN_TBL(3) = 'DN';
 DISPLAY_TBL(4) = 'X'   EXIT                  ';

/*  IF NO MENU_ENTRY THEN CALL ROUTINE TO SELECT A VALID ENTRY  */

   THE_VALUE = REC_DUP_OPTION;
   REC_DUP_OPTION = MENU_ENTRY;
   IF REC_DUP_OPTION = ' '
   THEN DO;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** RECORD DUPLICATE OPTIONS **',3);
     REC_DUP_OPTION = THE_VALUE;
     END;
```

```
   ELSE DO;

/*  THERE WAS A MENU_ENTRY -- RETURN IF VALID MENU_ENTRY          */

     DO I = 1 TO 3;
       IF DOMAIN_TBL(I) = REC_DUP_OPTION
       THEN
         RETURN;
     END;

/* THERE MUST HAVE BEEN AN INVALID ENTRY TO GET TO THIS POINT.
 WRITE AN ERROR MESSAGE AND CALL ROUTINE TO SELECT A VALID ENTRY*/

     PUT STRING (MSG) EDIT (REC_DUP_OPTION,' IS NOT ACCEPTABLE ',
         'FOR RECORD DUPLICATE OPTION -- USE MENU')  (3(A));
     CALL MESSAGES;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                  '** RECORD DUPLICATE OPTIONS **',3);
     REC_DUP_OPTION = THE_VALUE;
     END;
 END REC_DUP_RTN;
END RECORD_UPDATE;
```

```
SET_UPDATE:  PROC(MENU_ENTRY);
/*****************************************************************/
/* This module displays all existing SET occurrences for the     */
/* "current" DATA_BASE occurrence and allows the user to add     */
/* a new SET occurrence, change an existing SET occurrence, or   */
/* delete an existing SET occurrence.                            */
/*****************************************************************/
/*                                                               */
/*  SET_UPDATE                                                   */
/*                                                               */
/* +---------------+  +---------------+                         */
/* | NEW_SET       |  | CHG_DEL_SET   |                         */
/* |  +----------+ |  |  +----------+ |                         */
/* |  | NEW_     | |  |  | CHG_     | |                         */
/* |  | NAME_RTN | |  |  | NAME_RTN | |                         */
/* |  +----------+ |  |  +----------+ |                         */
/* +---------------+  +---------------+                         */
/*                                                               */
/* +---------+  +----------+  +--------+  +------------+        */
/* | SET_UP  |  | MEM_     |  | LINK_  |  | SET_       |        */
/* | MENU    |  | OWN_RTN  |  | RTN    |  | VALUE_RTN  |        */
/* +---------+  +----------+  +--------+  +------------+        */
/*                                                               */
/* +---------+  +----------+  +--------+  +------------+        */
/* | ORDER_  |  | SORT_    |  | MEM_   |  | DUP_       |        */
/* | RTN     |  | LMNT_RTN |  | RTN    |  | OPTION_RTN |        */
/* +---------+  +----------+  +--------+  +------------+        */
/*****************************************************************/

        DCL       MENU_ENTRY                 CHAR(69),
                  SLCT_NUM                   FIXED DEC(3),
                  MENU_NUM                   CHAR(3),
                  (SET_OWNER,SET_MEMBER)     CHAR(16);

        IF MENU_ENTRY ¬= ' '
        THEN DO;
          SET_NAME = MENU_ENTRY;
          SAVE_DB_NAME = DB_NAME;
          OBTAIN CALC RECORD (SE_T);
          DO WHILE (ERROR_STATUS = REC_FOUND);
            OBTAIN OWNER SET (LINKED_BY);  CALL IDMS_STATUS;
            IF SAVE_DB_NAME = DB_NAME
            THEN
              ERROR_STATUS = 'FWND';
            ELSE
              OBTAIN DUPLICATE RECORD (SE_T);
          END;
          IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
          THEN CALL IDMS_STATUS;
          IF (ERROR_STATUS = 'FWND')
          THEN DO;
            MENU_ENTRY = ' ';
            CALL CHG_DEL_SET (MENU_ENTRY);
            RETURN;
```

```
        END;
      ELSE DO;
        DB_NAME = SAVE_DB_NAME;
        OBTAIN CALC RECORD (DATA_BASE);
        PUT STRING (MSG) EDIT (MENU_ENTRY,
                ' IS NOT AN EXISTING SET') (A(16),A);
        CALL MESSAGES;
        END;
      END;

/*    LOAD TABLE WITH SET NAMES       */

   MENU_NUM = ' ';
   DO WHILE (MENU_NUM ¬= 'X');

     DISPLAY_TBL = ' ';

     PUT STRING (DISPLAY_TBL(1)) EDIT ('1)  CREATE SET')
                                      (X(4),A);
     OBTAIN FIRST SET (LINKED_BY);
     IF ERROR_STATUS ¬= '0307'
     THEN CALL IDMS_STATUS;
     COUNT = 1;
     DO WHILE (ERROR_STATUS = REC_FOUND);
       COUNT = COUNT + 1;
       PUT STRING (DISPLAY_TBL(COUNT)) EDIT (COUNT,')  ',
                    SET_NAME)
                   (X(2),F(3),2(A));
       DB_KEY_TBL(COUNT) = DBKEY;
       OBTAIN NEXT SET (LINKED_BY);
       IF ERROR_STATUS ¬= '0307'
       THEN CALL IDMS_STATUS;
     END;
     PUT STRING (DISPLAY_TBL(COUNT + 1)) EDIT ('    X)  EXIT')
                                              (A);

   /*    DISPLAY MENU   /  ACCEPT EDIT REPLY   */

     CALL GEN_MENU (MENU_NUM,MENU_ENTRY,'  ** SET UPDATE **',
                    COUNT,SLCT_NUM,3);

       IF MENU_NUM = 'X'
       THEN
         RETURN;

       IF SLCT_NUM = 1
       THEN
         CALL NEW_SET (MENU_ENTRY);
       ELSE DO;
         OBTAIN RECORD (SET) DBKEY (DB_KEY_TBL(SLCT_NUM));
         CALL IDMS_STATUS;
         CALL CHG_DEL_SET (MENU_ENTRY);
         END;
   END;
```

```
NEW_SET:  PROC (MENU_ENTRY);
/*****************************************************************/
/* This module established a new SET occurrence by defaulting  */
/* the SET occurrence fields and calling the modules to update */
/* the fields based on the data base designer's MENU_ENTRY.    */
/* Once updating is complete, the new SET occurrence is stored.*/
/*****************************************************************/

   DCL       STATUS                  CHAR(4),
             MENU_NUM                CHAR(3),
             MENU_ENTRY              CHAR(69);

/*  INITIALIZE AND DEFAULT SET FIELDS                            */

   SE_T           =  ' ';
   SET_OWNER      =  ' ';
   SET_MEMBER     =  ' ';
   SET_LINK       =  'NPO';
   SET_MEM        =  'MA';
   SET_ORDER      =  'FIRST';
   SET_SORT_LMNT  =  ' ';
   SET_DUP_OPTION =  'DN';
   SET_VALUE      =  '1M';
   SET_INVRS_VAL  =  '11';

/*  HAS A SET NAME BEEN PROVIDED FROM SECONDARY MENU?            */

   IF MENU_ENTRY = ' '
   THEN
     CALL SET_UP_MENU (MENU_NUM,MENU_ENTRY,
                       '** CREATE NEW SET **');
   ELSE
     MENU_NUM = '1';

/*  ALLOW ADDITION OF SET ATTRIBUTES UNTIL EXIT                  */

   DO WHILE (MENU_NUM ¬= 'X');
     IF MENU_NUM = '1' THEN
       CALL NEW_SET_NAME (MENU_ENTRY,SET_NAME);
     ELSE IF MENU_NUM = '2' THEN
       CALL MEM_OWN_RTN (MENU_ENTRY,SET_OWNER,SET_MEMBER,
                         '** SELECT SET OWNER **');
     ELSE IF MENU_NUM = '3' THEN
       CALL MEM_OWN_RTN (MENU_ENTRY,SET_MEMBER,SET_OWNER,
                         '** SELECT SET MEMBER **');
     ELSE IF MENU_NUM = '4' THEN
       CALL SET_VALUE_RTN (MENU_ENTRY,SET_VALUE);
     ELSE IF MENU_NUM = '5' THEN
       CALL SET_VALUE_RTN (MENU_ENTRY,SET_INVRS_VAL);
     ELSE IF MENU_NUM = '6' THEN
       CALL LINK_RTN (MENU_ENTRY,SET_LINK);
     ELSE IF MENU_NUM = '7' THEN
       CALL MEM_RTN (MENU_ENTRY,SET_MEM);
```

```
      ELSE IF MENU_NUM = '8' THEN
        CALL ORDER_RTN (MENU_ENTRY,SET_ORDER);
      ELSE IF MENU_NUM = '9' THEN
        CALL SORT_LMNT_RTN (MENU_ENTRY,SET_SORT_LMNT);
      ELSE IF MENU_NUM = '10' THEN
        CALL DUP_OPTION_RTN (MENU_ENTRY,SET_DUP_OPTION);
      CALL SET_UP_MENU (MENU_NUM,MENU_ENTRY,
                        '** CREATE NEW SET **');
    END;
    IF SET_NAME = ' '
    THEN DO;
      MSG = 'SET NAME IS BLANK -- NO ADD MADE';
      CALL MESSAGES;
      RETURN;
      END;
    STORE RECORD (SE_T);      CALL IDMS_STATUS;

   /*****  CONNECT SET_OWNER TO RESPECTIVE RECORD **************/

    IF SET_OWNER ¬= ' '
    THEN DO;
      REC_NAME = SET_OWNER;
      SAVE_DB_NAME = DB_NAME;
      OBTAIN CALC RECORD (RE_CORD);  CALL IDMS_STATUS;
      OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
      DO WHILE (SAVE_DB_NAME ¬= DB_NAME);
        OBTAIN DUPLICATE RECORD (RE_CORD);  CALL IDMS_STATUS;
        OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
      END;
      CONNECT RECORD (SE_T)  SET (OWNER_OF);  CALL IDMS_STATUS;
      END;

   /*****  CONNECT SET_MEMBER TO RESPECTIVE RECORD *************/

    IF SET_MEMBER ¬= ' '
    THEN DO;
      REC_NAME = SET_MEMBER;
      SAVE_DB_NAME = DB_NAME;
      OBTAIN CALC RECORD (RE_CORD);  CALL IDMS_STATUS;
      OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
      DO WHILE (SAVE_DB_NAME ¬= DB_NAME);
        OBTAIN DUPLICATE RECORD (RE_CORD);  CALL IDMS_STATUS;
        OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
      END;
      CONNECT RECORD (SE_T)  SET (MEMBER_OF);  CALL IDMS_STATUS;
      END;

NEW_SET_NAME:  PROC (MENU_ENTRY,SET_NAME);
/*****************************************************************/
/* This module allows the data base designer to assign a set     */
/* name to a newly created SET occurrence.  Before the set       */
/* name is accepted, a check is made to verify that it does      */
/* not already exist.                                            */
/*****************************************************************/
```

```
   DCL        MENU_ENTRY                CHAR(69),
              (SET_NAME,SAVE_NAME)      CHAR(16),
              STATUS                    CHAR(4);

   SET_NAME = MENU_ENTRY;
   CALL EDIT_NAME (SET_NAME,STATUS);
   IF STATUS = 'GOOD'
   THEN DO;
     SAVE_NAME = SET_NAME;
     SAVE_DB_NAME = DB_NAME;
     OBTAIN CALC RECORD (SE_T);
     DO WHILE (ERROR_STATUS = REC_FOUND);
       OBTAIN OWNER SET (LINKED_BY);  CALL IDMS_STATUS;
       IF SAVE_DB_NAME = DB_NAME
       THEN
         ERROR_STATUS = 'FWND';
       ELSE
         OBTAIN DUPLICATE RECORD (SE_T);
     END;
     IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
     THEN CALL IDMS_STATUS;

     IF ERROR_STATUS = 'FWND'
     THEN DO;
       PUT STRING (MSG) EDIT (SET_NAME,' ALREADY EXISTS')(2(A));
       CALL MESSAGES;
       SET_NAME = ' ';
       END;
     ELSE DO;
       DB_NAME = SAVE_DB_NAME;
       OBTAIN CALC RECORD (DATA_BASE);
       END;
     END;
   ELSE
     SET_NAME = ' ';
  END NEW_SET_NAME;
END NEW_SET;

CHG_DEL_SET:  PROC (MENU_ENTRY);
/*****************************************************************/
/* This module changes and deletes SET fields of the "current" */
/* SET occurrence.                                               */
/*****************************************************************/

   DCL        DEL_SW                                  CHAR(1),
              (SAVE_NAME,SAVE_OWNER,SAVE_MEMBER)      CHAR(16),
              MENU_NUM                                CHAR(3),
              MENU_ENTRY                              CHAR(69);


   SAVE_OWNER, SAVE_MEMBER = ' ';
   IF MENU_ENTRY = 'DELETE'
   THEN DO;
```

```
      PUT STRING (MSG) EDIT (SET_NAME,' SET DELETED') (2(A));
      OBTAIN CURRENT RECORD (SE_T);
      ERASE RECORD (SE_T);
      CALL IDMS_STATUS;
      CALL MESSAGES;
      RETURN;
      END;


/*       OBTAIN VALUES FOR SET_OWNER & SET_MEMBER IF EXISTENT    */

   OBTAIN CURRENT RECORD (SE_T);
   IF SET (OWNER_OF) MEMBER
   THEN DO;
     OBTAIN OWNER SET (OWNER_OF);  CALL IDMS_STATUS;
     SAVE_OWNER,SET_OWNER = REC_NAME;
     END;
   ELSE
     SET_OWNER = ' ';

   OBTAIN CURRENT RECORD (SE_T);
   IF SET (MEMBER_OF) MEMBER
   THEN DO;
     OBTAIN OWNER SET (MEMBER_OF);  CALL IDMS_STATUS;
     SAVE_MEMBER,SET_MEMBER = REC_NAME;
     END;
   ELSE
     SET_MEMBER = ' ';

/*  MAKE CHANGES TO SET INFO UNTIL EXIT                          */

   SAVE_NAME = SET_NAME;
   CALL SET_UP_MENU (MENU_NUM,MENU_ENTRY,
                     '** CHANGE OR DELETE SET **');

   DO WHILE (MENU_NUM ¬= 'X');
     IF MENU_NUM = '1' THEN
       DO;
       CALL CHG_SET_NAME (MENU_ENTRY,SET_NAME);
       IF SET_NAME ¬= ' '
       THEN
         SAVE_NAME = SET_NAME;
       END;
     ELSE IF MENU_NUM = '2' THEN
       CALL MEM_OWN_RTN (MENU_ENTRY,SET_OWNER,SET_MEMBER,
                         '** SELECT SET OWNER **');
     ELSE IF MENU_NUM = '3' THEN
       CALL MEM_OWN_RTN (MENU_ENTRY,SET_MEMBER,SET_OWNER,
                         '** SELECT SET MEMBER **');
     ELSE IF MENU_NUM = '4' THEN
       CALL SET_VALUE_RTN (MENU_ENTRY,SET_VALUE);
     ELSE IF MENU_NUM = '5' THEN
       CALL SET_VALUE_RTN (MENU_ENTRY,SET_INVRS_VAL);
     ELSE IF MENU_NUM = '6' THEN
```

```
      CALL LINK_RTN (MENU_ENTRY,SET_LINK);
    ELSE IF MENU_NUM = '7' THEN
      CALL MEM_RTN (MENU_ENTRY,SET_MEM);
    ELSE IF MENU_NUM = '8' THEN
      CALL ORDER_RTN (MENU_ENTRY,SET_ORDER);
    ELSE IF MENU_NUM = '9' THEN
      CALL SORT_LMNT_RTN (MENU_ENTRY,SET_SORT_LMNT);
    ELSE IF MENU_NUM = '10' THEN
      CALL DUP_OPTION_RTN (MENU_ENTRY,SET_DUP_OPTION);
    CALL SET_UP_MENU (MENU_NUM,MENU_ENTRY,
                      '** CREATE NEW SET **');
  END;

/*  IF BLANK SET_NAME THE CONFIRM DELETION AND EITHER DELETE OR
    ABORT DELETE REQUEST.                                      */

  IF SET_NAME = ' '
  THEN DO;
    CALL CLRSCR;
    CALL BLANK_LINE(5);
    DISPLAY ('CONFIRM DELETE BY TYPING "D"');
    CALL BLANK_LINE(5);
    DISPLAY ('====>') REPLY (DEL_SW);
    IF DEL_SW = 'D'
    THEN DO;
      PUT STRING (MSG) EDIT (SAVE_NAME,' SET DELETED') (2(A));
      FIND CURRENT RECORD (SE_T);
      ERASE RECORD (SE_T);
      CALL IDMS_STATUS;
      CALL MESSAGES;
      RETURN;
      END;
    ELSE DO;
      MSG = 'DELETE REQUEST ABORTED';
      CALL MESSAGES;
      SET_NAME = SAVE_NAME;
      END;
    END;

  IF SAVE_OWNER ¬= SET_OWNER
  THEN DO;
    /******** DELETE PREVIOUS SET_OWNER **************/
    IF SAVE_OWNER ¬= ' '
    THEN DO;
      SAVE_DB_NAME = DB_NAME;
      REC_NAME = SAVE_OWNER;
      OBTAIN CALC RECORD (RE_CORD);  CALL IDMS_STATUS;
      OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
      DO WHILE (SAVE_DB_NAME ¬= DB_NAME);
        OBTAIN DUPLICATE RECORD (RE_CORD);  CALL IDMS_STATUS;
        OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
      END;
      DISCONNECT RECORD (SE_T)  SET (OWNER_OF);
      END;
```

```
  /***************** END ****************************/

  /************ ADD NEW SET_OWNER ********************/
  IF SET_OWNER ¬= ' '
  THEN DO;
    REC_NAME = SET_OWNER;
    OBTAIN CALC RECORD (RE_CORD);  CALL IDMS_STATUS;
    OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
    DO WHILE (SAVE_DB_NAME ¬= DB_NAME);
      OBTAIN DUPLICATE RECORD (RE_CORD);  CALL IDMS_STATUS;
      OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
    END;
    CONNECT RECORD (SE_T) SET (OWNER_OF);
    END;
    /****************** END ***************************/
  END;

IF SAVE_MEMBER ¬= SET_MEMBER
THEN DO;
  /**** DELETES PREVIOUS SET MEMBER  *****************/
  IF SAVE_MEMBER ¬= ' '
  THEN DO;
    SAVE_DB_NAME = DB_NAME;
    REC_NAME = SAVE_MEMBER;
    OBTAIN CALC RECORD (RE_CORD);  CALL IDMS_STATUS;
    OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
    DO WHILE (SAVE_DB_NAME ¬= DB_NAME);
      OBTAIN DUPLICATE RECORD (RE_CORD);  CALL IDMS_STATUS;
      OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
    END;
    DISCONNECT RECORD (SE_T)  SET (MEMBER_OF);
    END;
  /************** END ******************************/

  /********* ADD NEW SET MEMBER **********************/
  IF SET_MEMBER ¬= ' '
  THEN DO;
    REC_NAME = SET_MEMBER;
    OBTAIN CALC RECORD (RE_CORD);  CALL IDMS_STATUS;
    OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
    DO WHILE (SAVE_DB_NAME ¬= DB_NAME);
      OBTAIN DUPLICATE RECORD (RE_CORD);  CALL IDMS_STATUS;
      OBTAIN OWNER SET (DIVIDED_INTO);  CALL IDMS_STATUS;
    END;
    CONNECT RECORD (SE_T) SET (MEMBER_OF);
    END;
    /*************** END ****************************/
  END;

FIND CURRENT RECORD (SE_T);  CALL IDMS_STATUS;
MODIFY RECORD (SE_T);  CALL IDMS_STATUS;
```

```
CHG_SET_NAME:  PROC (MENU_ENTRY,SET_NAME);
/*****************************************************************/
/*  This module changes the set name of the "current" SET        */
/*  occurrence based on the MENU_ENTRY.                          */
/*****************************************************************/

   DCL        MENU_ENTRY                CHAR(69),
              SET_NAME                  CHAR(16),
              STATUS                    CHAR(4);

   SET_NAME = MENU_ENTRY;
   CALL EDIT_NAME (SET_NAME,STATUS);
   IF STATUS = 'GOOD'
   THEN DO;
     IF SAVE_NAME = SET_NAME
     THEN
       ENTER_KEY = ' ';       /*  NULL STATEMENT  */
     ELSE DO;
          SAVE_DB_NAME = DB_NAME;
          OBTAIN CALC RECORD (SE_T);
          DO WHILE (ERROR_STATUS = REC_FOUND);
            OBTAIN OWNER SET (LINKED_BY);  CALL IDMS_STATUS;
            IF SAVE_DB_NAME = DB_NAME
            THEN
              ERROR_STATUS = 'FWND';
            ELSE
              OBTAIN DUPLICATE RECORD (SE_T);
          END;
          IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
          THEN CALL IDMS_STATUS;
       IF (ERROR_STATUS = 'FWND')
       THEN DO;
         PUT STRING (MSG) EDIT (SET_NAME,' ALREADY EXISTS')
                                (2(A));
         CALL MESSAGES;
         SET_NAME = SAVE_NAME;
         END;
       ELSE DO;
         DB_NAME = SAVE_DB_NAME;
         OBTAIN CALC RECORD (DATA_BASE);
         IF SET_NAME ¬= ' '
         THEN
           SAVE_NAME = SET_NAME; /*IN CASE ABORT A DELETE WE WANT
         END;                    /*TO KNOW THE LAST GOOD SET_NAME*/
       END;
     END;
   ELSE
     SET_NAME = SAVE_NAME;
  END CHG_SET_NAME;
END CHG_DEL_SET;
```

```
SET_UP_MENU:  PROC (MENU_NUM,MENU_ENTRY,MENU_MSG);
/*****************************************************************/
/* This module displays the SET fields that can be updated for  */
/* SET occurrences.                                              */
/*****************************************************************/

   DCL          MENU_MSG                      CHAR(30),
                MENU_NUM                      CHAR(3),
                MENU_ENTRY                    CHAR(69),
                STATUS                        CHAR(4),
                SLCT                          CHAR(72),
                (SLCT_NUM,NUM_ATTRIBUTES)     FIXED DEC(3);

   STATUS = 'BAD';
   DO WHILE (STATUS = 'BAD');
     MSG = MENU_MSG;
     CALL MENU_HEAD;
     PUT STRING (EDIT_OUT) EDIT
         (' 1)  SET NAME:  ', SET_NAME) (2(A));
     DISPLAY (EDIT_OUT);
     PUT STRING (EDIT_OUT) EDIT
         (' 2)  SET OWNER:  ', SET_OWNER) (2(A));
     DISPLAY (EDIT_OUT);
     PUT STRING (EDIT_OUT) EDIT
         (' 3)  SET MEMBER:  ', SET_MEMBER) (2(A));
     DISPLAY (EDIT_OUT);
     PUT STRING (EDIT_OUT) EDIT
         (' 4)  SET VALUE:  ', SET_VALUE) (2(A));
     DISPLAY (EDIT_OUT);
     PUT STRING (EDIT_OUT) EDIT
         (' 5)  SET INVERSE VALUE:  ', SET_INVRS_VAL) (2(A));
     DISPLAY (EDIT_OUT);
     PUT STRING (EDIT_OUT) EDIT
         (' 6)  SET LINKAGE:  ', SET_LINK) (2(A));
     DISPLAY (EDIT_OUT);
     PUT STRING (EDIT_OUT) EDIT
         (' 7)  SET MEMBERSHIP:  ', SET_MEM) (2(A));
     DISPLAY (EDIT_OUT);
     PUT STRING (EDIT_OUT) EDIT
         (' 8)  SET ORDER:  ', SET_ORDER) (2(A));
     DISPLAY (EDIT_OUT);
     IF (SET_ORDER = 'ASC' | SET_ORDER = 'DES')
     THEN DO;
       NUM_ATTRIBUTES = 10;
       PUT STRING (EDIT_OUT) EDIT
           (' 9)  SET SORT ELEMENT:  ', SET_SORT_LMNT) (2(A));
       DISPLAY (EDIT_OUT);
       PUT STRING (EDIT_OUT) EDIT
         ('10)  SET DUPLICATE OPTION:  ', SET_DUP_OPTION) (2(A));
     DISPLAY (EDIT_OUT);
       END;
     ELSE
       NUM_ATTRIBUTES = 8;
     DISPLAY (' X)  EXIT');
```

```
      CALL BLANK_LINE(2);
      DISPLAY ('====>') REPLY (SLCT);
      CALL EXAMINE_ENTRY (SLCT,MENU_NUM,MENU_ENTRY,SLCT_NUM,
                          STATUS,NUM_ATTRIBUTES);
   END;
END SET_UP_MENU;

MEM_OWN_RTN:  PROC (MENU_ENTRY,CHG_REC,TEST_REC,MENU_MSG);
/*****************************************************************/
/* This module verifies that the CHG_REC parameter exists as     */
/* an occurrence of the RE_CORD structure of the user's data     */
/* base and that it is different than TEST_REC.  This is         */
/* done to avoid having identical owner and member records for   */
/* the same SET occurrence.                                      */
/*****************************************************************/

/*  THIS ROUTINE VERIFIES THAT THE CHOSEN RECORD EXISTS AND THAT
    IT IS DIFFERENT THAN THE OTHER RECORD OF THAT SET           */


   DCL        MENU_NUM                 CHAR(3),
              MENU_ENTRY               CHAR(69),
              (CHG_REC,TEST_REC)       CHAR(16),
              (SUB,TBL_SIZE)           FIXED DEC(3),
              MENU_MSG                 CHAR(40);

/*  LOAD DISPLAY_TBL AND DOMAIN_TBL WITH EXISTING RECORDS      */

  DISPLAY_TBL = ' ';
  OBTAIN FIRST SET (DIVIDED_INTO);
  IF ERROR_STATUS = '0307'
  THEN DO;
    PUT STRING (MSG) EDIT ('NO RECORDS TO SELECT FROM!')(A);
    CALL MESSAGES;
    RETURN;
    END;
  TBL_SIZE = 0;
  DO WHILE (ERROR_STATUS = OK);
    TBL_SIZE = TBL_SIZE + 1;
    PUT STRING (DISPLAY_TBL(TBL_SIZE)) EDIT
              (TBL_SIZE,')  ',REC_NAME)(X(2),F(3),2(A));
    DOMAIN_TBL(TBL_SIZE) = REC_NAME;
    OBTAIN NEXT SET (DIVIDED_INTO);
  END;
  IF ERROR_STATUS ¬= '0307'
  THEN CALL IDMS_STATUS;
  DISPLAY_TBL(TBL_SIZE+1) = '    X)  EXIT';

/*  IF NO SELECTION THEN GIVE LIST OF RECORDS                  */

  IF MENU_ENTRY = ' '
  THEN DO;
    CALL GEN_MENU (MENU_NUM,MENU_ENTRY,MENU_MSG,TBL_SIZE,
                   SLCT_NUM,3);
```

```
    IF MENU_NUM = 'X'
    THEN DO;
      CHG_REC = ' ';
      RETURN;
      END;
    DO WHILE (DOMAIN_TBL(SLCT_NUM) = TEST_REC);
      MSG = 'THE SAME RECORD CANNOT BE BOTH MEMBER & OWNER';
      CALL MESSAGES;
      CALL GEN_MENU (MENU_NUM,MENU_ENTRY,MENU_MSG,
                     TBL_SIZE,SLCT_NUM,3);
      IF MENU_NUM = 'X'
      THEN DO;
        CHG_REC = ' ';
        RETURN;
        END;
    END;
    CHG_REC = DOMAIN_TBL(SLCT_NUM);
    RETURN;
    END;

/*  A MEMBER/OWNER RECORD WAS GIVEN -- VERIFY                    */

  CHG_REC = MENU_ENTRY;
  DO SUB = 1 TO TBL_SIZE  WHILE (CHG_REC ¬= TEST_REC);
    IF CHG_REC = DOMAIN_TBL(SUB)
    THEN
      RETURN;     /* GOOD SELECTION -- RETURN */
  END;

/*  A NON ACCEPTABLE RECORD WAS GIVEN -- DISPLAY ERROR AND GIVE */
/*  A LIST OF VALID RECORDS TO CHOSE FROM.                      */

  IF CHG_REC = TEST_REC
  THEN DO;
    MSG = 'THE SAME RECORD CANNOT BE BOTH MEMBER & OWNER';
    CALL MESSAGES;
    END;
  ELSE DO;
    PUT STRING (MSG) EDIT
               (CHG_REC,' NOT AN EXISTING RECORD -- USE MENU')
               (2(A));
    CALL MESSAGES;
    END;

  CALL GEN_MENU (MENU_NUM,MENU_ENTRY,MENU_MSG,TBL_SIZE,
                 SLCT_NUM,3);
  IF MENU_NUM = 'X'
  THEN DO;
    CHG_REC = ' ';
    RETURN;
    END;

  DO WHILE (DOMAIN_TBL(SLCT_NUM) = TEST_REC);
    MSG = 'THE SAME RECORD CANNOT BE BOTH MEMBER & OWNER';
```

```
     CALL MESSAGES;
     CALL GEN_MENU (MENU_NUM,MENU_ENTRY,MENU_MSG,TBL_SIZE,
                    SLCT_NUM,3);
     IF MENU_NUM = 'X'
     THEN DO;
       CHG_REC = ' ';
       RETURN;
       END;
   END;
   CHG_REC = DOMAIN_TBL(SLCT_NUM);
END MEM_OWN_RTN;

LINK_RTN:  PROC (MENU_ENTRY,SET_LINK);
/*****************************************************************/
/* This module updates the SET_LINK field in the SET             */
/* structure.  If no MENU_ENTRY parameter is inputted into this */
/* module, a list of possible values will be displayed for the  */
/* data base designer to make a decision.                        */
/*****************************************************************/

   DCL       MENU_ENTRY                CHAR(69),
             SET_LINK                  CHAR(3),
             THE_VALUE                 CHAR(16),
             I                         FIXED DEC(2);

/*   LOAD DISPLAY TABLE AND DOMAIN TABLE                         */

  DISPLAY_TBL(1) = '1)  NEXT            ';  DOMAIN_TBL(1) = 'N  ';
  DISPLAY_TBL(2) = '2)  NEXT PRIOR      ';  DOMAIN_TBL(2) = 'NP ';
  DISPLAY_TBL(3) = '3)  NEXT OWNER      ';  DOMAIN_TBL(3) = 'NO ';
  DISPLAY_TBL(4) = '4)  NEXT PRIOR OWNER';  DOMAIN_TBL(4) = 'NPO';
  DISPLAY_TBL(5) = 'X)  EXIT            ';

/*  IF NO MENU_ENTRY THEN CALL ROUTINE TO SELECT A VALID ENTRY  */

   THE_VALUE = SET_LINK;
   SET_LINK = MENU_ENTRY;
   IF SET_LINK = ' '
   THEN DO;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** SET LINKAGE VALUES **',4);
     SET_LINK = THE_VALUE;
     END;
   ELSE DO;

/*  THERE WAS A MENU_ENTRY -- RETURN IF VALID MENU_ENTRY        */

     DO I = 1 TO 4;
       IF DOMAIN_TBL(I) = SET_LINK
       THEN
         RETURN;
     END;

/* THERE MUST HAVE BEEN AN INVALID ENTRY TO GET TO THIS POINT.
```

```
WRITE AN ERROR MESSAGE AND CALL ROUTINE TO SELECT A VALID ENTRY */

     PUT STRING (MSG) EDIT (SET_LINK,' IS NOT AN ACCEPTABLE ',
               'VALUE FOR SET LINKAGE -- USE MENU')  (3(A));
     CALL MESSAGES;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** SET LINKAGE VALUES **',4);
     SET_LINK = THE_VALUE;
     END;
END LINK_RTN;

SET_VALUE_RTN:  PROC (MENU_ENTRY,SET_VALUE);
/*****************************************************************/
/* This module updates the SET_VALUE field in the SET            */
/* structure.  If no MENU_ENTRY parameter is inputted into this */
/* module, a list of possible values will be displayed for the  */
/* data base designer to make a decision.                        */
/*****************************************************************/

   DCL       MENU_ENTRY                 CHAR(69),
             SET_VALUE                  CHAR(2),
             THE_VALUE                  CHAR(16),
             I                          FIXED DEC(2);

/*   LOAD DISPLAY TABLE AND DOMAIN TABLE                          */

   DISPLAY_TBL(1) = '1)  1 TO 1          ';  DOMAIN_TBL(1) = '11';
   DISPLAY_TBL(2) = '2)  1 TO MANY       ';  DOMAIN_TBL(2) = '1M';
   DISPLAY_TBL(3) = 'X)  EXIT            ';

/*  IF NO MENU_ENTRY THEN CALL ROUTINE TO SELECT A VALID ENTRY  */

   THE_VALUE = SET_VALUE;
   SET_VALUE = MENU_ENTRY;
   IF SET_VALUE = ' '
   THEN DO;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** SET VALUE OPTIONS **',2);
     SET_VALUE = THE_VALUE;
     END;
   ELSE DO;

/*  THERE WAS A MENU_ENTRY -- RETURN IF VALID MENU_ENTRY         */

     DO I = 1 TO 2;
       IF DOMAIN_TBL(I) = SET_VALUE
       THEN
         RETURN;
     END;

/*  THERE MUST HAVE BEEN AN INVALID ENTRY TO GET TO THIS POINT.
WRITE AN ERROR MESSAGE AND CALL ROUTINE TO SELECT A VALID ENTRY*/

     PUT STRING (MSG) EDIT (SET_VALUE,' IS NOT AN ACCEPTABLE ',
```

```
                'VALUE -- USE MENU')   (3(A));
     CALL MESSAGES;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** SET VALUE OPTIONS **',2);
     SET_VALUE = THE_VALUE;
     END;
END SET_VALUE_RTN;

MEM_RTN:  PROC (MENU_ENTRY,SET_MEM);
/*****************************************************************/
/* This module updates the SET_MEM field in the SET              */
/* structure.  If no MENU_ENTRY parameter is inputted into this  */
/* module, a list of possible values will be displayed for the   */
/* data base designer to make a decision.                        */
/*****************************************************************/

   DCL       MENU_ENTRY                 CHAR(69),
             SET_MEM                    CHAR(2),
             THE_VALUE                  CHAR(16),
             I                          FIXED DEC(2);

/*   LOAD DISPLAY TABLE AND DOMAIN TABLE                         */

DISPLAY_TBL(1) = '1)  MANDATORY AUTOMATIC';  DOMAIN_TBL(1) = 'MA';
DISPLAY_TBL(2) = '2)  MANDATORY MANUAL    ';  DOMAIN_TBL(2) = 'MM';
DISPLAY_TBL(3) = '3)  OPTIONAL AUTOMATIC  ';  DOMAIN_TBL(3) = 'OA';
DISPLAY_TBL(4) = '4)  OPTIONAL MANUAL     ';  DOMAIN_TBL(4) = 'OM';
DISPLAY_TBL(5) = 'X)  EXIT                ';

/*  IF NO MENU_ENTRY THEN CALL ROUTINE TO SELECT A VALID ENTRY  */

   THE_VALUE = SET_MEM;
   SET_MEM = MENU_ENTRY;
   IF SET_MEM = ' '
   THEN DO;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** SET MEMBERSHIP VALUES **',4);
     SET_MEM = THE_VALUE;
     END;
   ELSE DO;

/*  THERE WAS A MENU_ENTRY -- RETURN IF VALID MENU_ENTRY        */

     DO I = 1 TO 4;
       IF DOMAIN_TBL(I) = SET_MEM
       THEN
         RETURN;
     END;

/*  THERE MUST HAVE BEEN AN INVALID ENTRY TO GET TO THIS POINT.
WRITE AN ERROR MESSAGE AND CALL ROUTINE TO SELECT A VALID ENTRY*/

     PUT STRING (MSG) EDIT (SET_MEM,' IS NOT AN ACCEPTABLE ',
                'VALUE FOR SET MEMBERSHIP -- USE MENU')  (3(A));
```

```
     CALL MESSAGES;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** SET MEMBERSHIP VALUES **',4);
     SET_MEM = THE_VALUE;
     END;
END MEM_RTN;


DUP_OPTION_RTN:  PROC (MENU_ENTRY,SET_DUP_OPTION);
/*****************************************************************/
/* This module updates the SET_DUP_OPTION field in the SET       */
/* structure.  If no MENU_ENTRY parameter is inputted into this */
/* module, a list of possible values will be displayed for the  */
/* data base designer to make a decision.                        */
/*****************************************************************/

   DCL       MENU_ENTRY                CHAR(69),
             SET_DUP_OPTION            CHAR(2),
             THE_VALUE                 CHAR(16),
             I                         FIXED DEC(2);

/*   LOAD DISPLAY TABLE AND DOMAIN TABLE                          */

 DISPLAY_TBL(1) =
            '1)  DUPLICATES FIRST       ';  DOMAIN_TBL(1) = 'DF';
 DISPLAY_TBL(2) =
            '2)  DUPLICATES LAST        ';  DOMAIN_TBL(2) = 'DL';
 DISPLAY_TBL(3) =
            '3)  DUPLICATES NOT ALLOWED';  DOMAIN_TBL(3) = 'DN';
 DISPLAY_TBL(4) =
            'X)  EXIT                   ';

/*  IF NO MENU_ENTRY THEN CALL ROUTINE TO SELECT A VALID ENTRY  */

   THE_VALUE = SET_DUP_OPTION;
   SET_DUP_OPTION = MENU_ENTRY;
   IF SET_DUP_OPTION = ' '
   THEN DO;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** SET DUPLICATE OPTIONS **',3);
     SET_DUP_OPTION = THE_VALUE;
     END;
   ELSE DO;

/*  THERE WAS A MENU_ENTRY -- RETURN IF VALID MENU_ENTRY         */

     DO I = 1 TO 3;
       IF DOMAIN_TBL(I) = SET_DUP_OPTION
       THEN
         RETURN;
     END;

/*  THERE MUST HAVE BEEN AN INVALID ENTRY TO GET TO THIS POINT.
WRITE AN ERROR MESSAGE AND CALL ROUTINE TO SELECT A VALID ENTRY */
```

```
      PUT STRING (MSG) EDIT (SET_DUP_OPTION,' IS NOT ACCEPTABLE ',
                  'FOR SET DUPLICATE OPTION -- USE MENU')   (3(A));
      CALL MESSAGES;
      CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                       '** SET DUPLICATE OPTIONS **',3);
      SET_DUP_OPTION = THE_VALUE;
      END;
END DUP_OPTION_RTN;

ORDER_RTN:  PROC (MENU_ENTRY,SET_ORDER);
/*****************************************************************/
/* This module updates the SET_ORDER field in the SET            */
/* structure.  If no MENU_ENTRY parameter is inputted into this  */
/* module, a list of possible values will be displayed for the   */
/* data base designer to make a decision.                        */
/*****************************************************************/
   DCL        MENU_ENTRY                CHAR(69),
              SET_ORDER                 CHAR(5),
              THE_VALUE                 CHAR(16),
              I                         FIXED DEC(2);

/*   LOAD DISPLAY TABLE AND DOMAIN TABLE                          */

   DISPLAY_TBL(1) = '1)  FIRST       ';  DOMAIN_TBL(1) = 'FIRST';
   DISPLAY_TBL(2) = '2)  LAST        ';  DOMAIN_TBL(2) = 'LAST ';
   DISPLAY_TBL(3) = '3)  NEXT        ';  DOMAIN_TBL(3) = 'NEXT ';
   DISPLAY_TBL(4) = '4)  PRIOR       ';  DOMAIN_TBL(4) = 'PRIOR';
   DISPLAY_TBL(5) = '5)  ASCENDING   ';  DOMAIN_TBL(5) = 'ASC  ';
   DISPLAY_TBL(6) = '6)  DESCENDING  ';  DOMAIN_TBL(6) = 'DES  ';
   DISPLAY_TBL(7) = 'X)  EXIT        ';

/*  IF NO MENU_ENTRY THEN CALL ROUTINE TO SELECT A VALID ENTRY    */
   THE_VALUE = SET_ORDER;
   SET_ORDER = MENU_ENTRY;
   IF SET_ORDER = ' '
   THEN DO;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** SET ORDER VALUES **',6);
     SET_ORDER = THE_VALUE;
     END;
   ELSE DO;

/*  THERE WAS A MENU_ENTRY -- RETURN IF VALID MENU_ENTRY          */

     DO I = 1 TO 6;
       IF DOMAIN_TBL(I) = SET_ORDER
       THEN
         RETURN;
     END;

/*  THERE MUST HAVE BEEN AN INVALID ENTRY TO GET TO THIS POINT.
WRITE AN ERROR MESSAGE AND CALL ROUTINE TO SELECT A VALID ENTRY */
```

```
     PUT STRING (MSG) EDIT (SET_ORDER,' IS NOT AN ACCEPTABLE ',
                  'VALUE FOR SET ORDER -- USE MENU')  (3(A));
     CALL MESSAGES;
     CALL SLCT_VALUE (DISPLAY_TBL,DOMAIN_TBL,THE_VALUE,
                      '** SET ORDER VALUES **',6);
     SET_ORDER = THE_VALUE;
     END;
END ORDER_RTN;

SORT_LMNT_RTN:  PROC (MENU_ENTRY,SET_SORT_LMNT);
/*****************************************************************/
/* This module updates the SET_SORT_LMNT field in the SET        */
/* structure.  If no MENU_ENTRY parameter is inputted into this  */
/* module, a list of possible values will be displayed for the   */
/* data base designer to make a decision.                        */
/*****************************************************************/

   DCL        MENU_NUM                        CHAR(3),
              MENU_ENTRY                      CHAR(69),
              SET_SORT_LMNT                   CHAR(16),
              (SUB,TBL_SIZE)                  FIXED DEC(3);


   IF SET_MEMBER = ' '
   THEN DO;
     MSG = ' SET MEMBER ENTRY MUST FIRST BE MADE';
     CALL MESSAGES;
     RETURN;
     END;

/* LOAD DISPLAY_TBL AND DOMAIN_TBL WITH DATA ELEMENTS OF
   MEMBER SET*/

  REC_NAME = SET_MEMBER;
  OBTAIN CALC RECORD (RE_CORD);
  CALL IDMS_STATUS;
  OBTAIN FIRST SET (POPULATED_WITH);
  IF ERROR_STATUS = '0307'
  THEN DO;
    PUT STRING (MSG) EDIT
        ('NO ELEMENTS IN ',SET_MEMBER,' TO BE SELECTED FROM!')
         (3(A));
    CALL MESSAGES;
    RETURN;
    END;
  DO TBL_SIZE = 1 TO 999  WHILE (ERROR_STATUS = OK);
    PUT STRING (DISPLAY_TBL(TBL_SIZE)) EDIT
        (TBL_SIZE,')   ',LMNT_NAME) (X(2),F(3),2(A));
    DOMAIN_TBL = LMNT_NAME;
    OBTAIN NEXT SET (POPULATED_WITH);
  END;
  IF ERROR_STATUS ¬= '0307'
  THEN CALL IDMS_STATUS;
  DISPLAY_TBL(TBL_SIZE + 1) = '  X)  EXIT';
```

```
/*IF NO SELECTION THEN GIVE LIST OF DATA ELEMENTS FOR
  MEMBER RECORD */

  IF MENU_ENTRY = ' '
  THEN DO;
    CALL GEN_MENU (MENU_NUM,MENU_ENTRY,
                   '** SELECT SORT ELEMENT **',
                   TBL_SIZE,SLCT_NUM,3);
    SET_SORT_LMNT = DOMAIN_TBL (SLCT_NUM);
    RETURN;
    END;

/* A SET_SORT_LMNT WAS GIVEN -- SEE IF IT IS ACCEPTABLE          */

  SET_SORT_LMNT = MENU_ENTRY;
  DO SUB = 1 TO TBL_SIZE;
    IF SET_SORT_LMNT = DOMAIN_TBL(SUB)
    THEN
      RETURN;
  END;

/*  A NON ACCEPTABLE SET_SORT_LMNT WAS GIVEN.  DISPLAY ERROR AND*/
/*  GIVE A LIST OF VALID CHOICES                                 */

  PUT STRING (MSG) EDIT
    (SET_SORT_LMNT,' NOT FOUND IN ',SET_MEMBER,' - USE MENU')
    (4(A));
  CALL MESSAGES;
  CALL GEN_MENU (MENU_NUM,MENU_ENTRY,'** SELECT SORT ELEMENT **',
                 TBL_SIZE,SLCT_NUM,3);
  SET_SORT_LMNT = DOMAIN_TBL(SLCT_NUM);
 END SORT_LMNT_RTN;
END SET_UPDATE;
```

```
CHG_DEL_DB:  PROC (DEL_SW);
/*****************************************************************/
/*  This module displays the values of the "current" DATA_BASE   */
/*  occurrence and allows the data base designer to change the   */
/*  DATA_BASE occurrence's fields or delete the DATA_BASE        */
/*  occurrence's fields.                                         */
/*****************************************************************/

/*******************/
/*                 */
/*   CHG_DEL_DB    */
/*                 */
/*******************/

        DCL         MENU_NUM                CHAR(3),
                    MENU_ENTRY              CHAR(69),
                    SAVE_NAME               CHAR(16),
                    DEL_SW                  CHAR(1),
                    STATUS                  CHAR(4),
                    D                       CHAR(6),
                    DATE                    BUILTIN;

 /* DISPLAY MENU AND MAKE CHANGES TO CURRENT DATA BASE RECORD */

        DEL_SW = ' ';
        SAVE_NAME = DB_NAME;
        CALL DB_UP_MENU (MENU_NUM,MENU_ENTRY,
                         '** CHANGE OR DELETE DATA BASE **');
        DO WHILE (MENU_NUM ¬= 'X');
          IF MENU_NUM = '1'
          THEN DO;
            DB_NAME = MENU_ENTRY;
            CALL EDIT_NAME (DB_NAME,STATUS);
            IF STATUS = 'GOOD'
            THEN DO;
              IF SAVE_NAME = DB_NAME
              THEN
                  ENTER_KEY = ' ';     /* NULL STATEMENT  */
              ELSE DO;
                FIND CALC RECORD (DATA_BASE);
                IF ERROR_STATUS ¬= '0326'
                THEN CALL IDMS_STATUS;
                IF ERROR_STATUS = REC_FOUND
                THEN DO;
                  PUT STRING (MSG) EDIT (DB_NAME,
                              ' ALREADY EXISTS') (2(A));
                  CALL MESSAGES;
                  DB_NAME = SAVE_NAME;
                  END;
                ELSE DO;
                  IF DB_NAME ¬= ' '
                  THEN
                    SAVE_NAME = DB_NAME;  /* A GOOD DB_NAME */
                  END;
```

```
            END;
          END;
        ELSE
          DB_NAME = ' ';
        END;
      ELSE
        DBA = MENU_ENTRY;
      CALL DB_UP_MENU (MENU_NUM,MENU_ENTRY,
                       '** CHANGE OR DELETE DATA BASE **');
    END;

/*  IF BLANK DB_NAME THEN CONFIRM DELETION AND EITHER DELETE OR
    ABORT DELETE REQUEST                                        */

    IF DB_NAME = ' '
    THEN DO;
      CALL CLRSCR;
      CALL BLANK_LINE (5);
      DISPLAY ('CONFIRM DELETE BY TYPING "D" ');
      CALL BLANK_LINE (5);
      DISPLAY ('====>') REPLY (DEL_SW);
      IF DEL_SW = 'D'
      THEN DO;
        PUT STRING (MSG) EDIT (SAVE_NAME,' DATA BASE DELETED')
                    (2(A));
        FIND CURRENT RECORD (DATA_BASE);
        ERASE RECORD (DATA_BASE) ALL;
        CALL IDMS_STATUS;
        CALL MESSAGES;
        RETURN;
        END;
      ELSE DO;
        MSG = 'DELETE REQUEST ABORTED';
        CALL MESSAGES;
        DB_NAME = SAVE_NAME;
        END;
      END;

/*  MAKE CHANGES TO CURRENT DATA BASE RECORD        */

    D = DATE;
    YEAR_CHANGED = SUBSTR(D,1,2);
    MONTH_CHANGED = SUBSTR(D,3,2);
    DAY_CHANGED = SUBSTR(D,5,2);
    MODIFY RECORD (DATA_BASE);
    CALL IDMS_STATUS;

END CHG_DEL_DB;
/*****************************************************************/
/**           DB_CUST module ends here.                         **/
/*****************************************************************/
```

```
 CREATE_SCHEMA:  PROC;
/*************************************************************/
/* This module, when completed, will create an opera-        */
/* tional schema based on information stored in DB_GEN's     */
/* data base. Five sub-systems used to accomplish this       */
/* are: 1) Missing data check, 2) Validate customiza-        */
/* tion, 3) Simplify complex relationships, 4) Create        */
/* pointer, and 5) Create DDL statements.                    */
/*************************************************************/
/*                                                           */
/*   CREATE_SCHEMA                                           */
/*                                                           */
/*  +--------------+  +-------------+  +--------------+      */
/*  | MISSING_     |  | ENTITY_     |  | DDL_         |      */
/*  | DATA_CHK     |  | CUST_CK     |  | CREATE       |      */
/*  | +----------+ |  | +---------+ |  | +---------+  |      */
/*  | | LMNT_    | |  | | LMNT_   | |  | | SCHEMA_ |  |      */
/*  | | DATA_CHK | |  | | CUST_CK | |  | | DSCR    |  |      */
/*  | +----------+ |  | +---------+ |  | +---------+  |      */
/*  | +----------+ |  | +---------+ |  | +---------+  |      */
/*  | | REC_     | |  | | REC_    | |  | | FILE_   |  |      */
/*  | | DATA_CHK | |  | | CUST_CK | |  | | DSCR    |  |      */
/*  | +----------+ |  | +---------+ |  | +---------+  |      */
/*  | +----------+ |  | +---------+ |  | +---------+  |      */
/*  | | SET_     | |  | | SET_    | |  | | AREA_   |  |      */
/*  | | DATA_CHK | |  | | CUST_CK | |  | | DSCR    |  |      */
/*  | +----------+ |  | +---------+ |  | +---------+  |      */
/*  +--------------+  +-------------+  | +---------+  |      */
/*                                     | | RECORD_ |  |      */
/*  +----------+   +-------------+     | | DSCR    |  |      */
/*  |          |   |             |     | +---------+  |      */
/*  | SIMPLE_  |   |  POINTER_   |     | +------+     |      */
/*  | M_N      |   |  CREATE     |     | | SET_ |     |      */
/*  |          |   |             |     | | DSCR |     |      */
/*  +----------+   +-------------+     | +------+     |      */
/*                                     +--------------+      */
/*                                                           */
/*************************************************************/

   MSG = 'CREATE_SCHEMA TO BE COMPLETED';
   CALL MENU_HEAD;
   DISPLAY ('PRESS ENTER TO CONTINUE') REPLY (ENTER_KEY);
  END  CREATE_SCHEMA;
```

```
 PRINT_DATA:  PROC;
/*****************************************************************/
/*  This module will print or display information describing     */
/*  a user's data base schema.                                   */
/*****************************************************************/
/*                                                               */
/*   PRINT_DATA                                                  */
/*                                                               */
/*    +---------+   +-----------+    +----------+               */
/*    | LMNT_   |   | LMNT_     |    | RECORD_  |               */
/*    | PRINT   |   | DISPLAY   |    | PRINT    |               */
/*    +---------+   +-----------+    +----------+               */
/*                                                               */
/*    +---------+  +-------+  +---------+  +-----------------+  */
/*    | RECORD_ |  | SET_  |  | SET_    |  | ALL_ENTITIES_   |  */
/*    | DISPLAY |  | PRINT |  | DISPLAY |  |     PRINT       |  */
/*    +---------+  +-------+  +---------+  +-----------------+  */
/*                                                               */
/*****************************************************************/

        MSG = 'PRINT_DATA TO BE COMPLETED';
        CALL MENU_HEAD;
        DISPLAY ('PRESS ENTER TO CONTINUE') REPLY (ENTER_KEY);
        END  PRINT_DATA;
```

```
/*****************************************************************/
/**             The UTILITY_RTNS start here                     **/
/*****************************************************************/
/*                                                               */
/*  UTILITY_RTNS                                                 */
/*                                                               */
/*   +--------+  +--------+  +----------+  +--------+  +------+  */
/*   | GEN_   |  | SLCT_  |  | EXAMINE_ |  | EDIT_  |  | GEN_ |  */
/*   | MENU   |  | VALUE  |  |  ENTRY   |  | NAME   |  | INST |  */
/*   +--------+  +--------+  +----------+  +--------+  +------+  */
/*                                                               */
/*   +---------+  +--------+  +---------+  +----------+  +-------+*/
/*   | MESSA-  |  | MENU_  |  | BLANK_  |  | ASRRIC_  |  | DIV_  |*/
/*   |  GES    |  | HEAD   |  |  LINE   |  |  LINE    |  | ENTRY |*/
/*   +---------+  +--------+  +---------+  +----------+  +-------+*/
/*                                                               */
/*****************************************************************/


 DIV_ENTRY:       PROC (MENU_ENTRY,PART_TBL,STATUS);
/*****************************************************************/
/*  This module is used to divide a multiple MENU_ENTRY,         */
/*  separated by semi-colons, into parts and store the           */
/*  in the PART_TBL.                                             */
/*****************************************************************/

       DCL       MENU_ENTRY                CHAR(69),
                 PART_TBL(20)              CHAR(16),
                 STATUS                    CHAR(4),
                 (START,SUB,I,J)           FIXED(3);

   /********  REMOVE ALL LEADING ";"          *************/
   DO WHILE (SUBSTR(MENU_ENTRY,1,1) = ';');
     DO I = 1 TO 68  WHILE (SUBSTR(MENU_ENTRY,I,70-I) ¬= ' ');
       SUBSTR(MENU_ENTRY,I,1) = SUBSTR(MENU_ENTRY,I+1,1);
     END;
     SUBSTR(MENU_ENTRY,I+1,1) = ' ';
   END;

   /***  REDUCE ANY MULTIPLE CONTIGUOUS ";" TO SINGLE ";" ******/
   DO I = 2 TO 68  WHILE (SUBSTR(MENU_ENTRY,I,70-I) ¬= ' ');
     DO WHILE (SUBSTR(MENU_ENTRY,I,1)      = ';'   &
               SUBSTR(MENU_ENTRY,(I+1),1) = ';');
       DO J = I TO 68  WHILE (SUBSTR(MENU_ENTRY,J,70-J) ¬= ' ');
         SUBSTR(MENU_ENTRY,J,1) = SUBSTR(MENU_ENTRY,J+1,1);
       END;
       SUBSTR(MENU_ENTRY,J+1,1) = ' ';
     END;
   END;

   /***** LEFT JUSTIFY ALL ENTRIES AGAINST ";"   *********/
   DO I = 2 TO 68  WHILE (SUBSTR(MENU_ENTRY,I,70-I) ¬= ' ');
     IF SUBSTR(MENU_ENTRY,I,1) = ';'
     THEN DO;
```

```
    DO WHILE (SUBSTR(MENU_ENTRY,I+1,1) = ' ');
      DO J = (I+1) TO 68
        WHILE (SUBSTR(MENU_ENTRY,I,70-I) ¬= ' ');
        SUBSTR(MENU_ENTRY,J,1) = SUBSTR(MENU_ENTRY,J+1,1);
      END;
      SUBSTR(MENU_ENTRY,J+1,1) = ' ';
    END;
    END;
END;

/*** ONCE VERIFIED AS VALID LOAD TABLE WITH ENTRIES ********/
START    = 1;
SUB      = 0;
PART_TBL = ' ';
STATUS   = 'GOOD';
DO I = 2 TO 68  WHILE (SUBSTR(MENU_ENTRY,I,70-I) ¬= ' ');
  IF SUBSTR(MENU_ENTRY,I,1) = ';'
  THEN DO;
    SUB = SUB + 1;
    PART_TBL(SUB) = SUBSTR(MENU_ENTRY,START,I-START);
    /****** SEE IF CAND_FIELD IS AN EXISTING DATA ELEMENT ****/
    LMNT_NAME = PART_TBL(SUB);
    SAVE_DB_NAME = DB_NAME;
    OBTAIN CALC RECORD (DATA_ELEMENT);
    DO WHILE (ERROR_STATUS = REC_FOUND);
      OBTAIN OWNER SET (DEFINED_BY); CALL IDMS_STATUS;
      IF SAVE_DB_NAME = DB_NAME
      THEN
        ERROR_STATUS = 'FWND';
      ELSE
        OBTAIN DUPLICATE RECORD (DATA_ELEMENT);
    END;
    IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
    THEN CALL IDMS_STATUS;
    IF ERROR_STATUS ¬= 'FWND'
    THEN DO;
      PUT STRING (MSG) EDIT
        ('ENTRY ',SUB,' DOES NOT EXIST - USE MENU') (A,F(3),A);
      CALL MESSAGES;
      STATUS = 'BAD';
      RETURN;
      END;
    ELSE
      START = I + 1;
    END;
END;

/** SEE IF THERE WAS A FINAL ENTRY.  KEYING ON ";" AND FINAL */
/** ENTRY LIKELY WILL NOT BE FOLLOWED BY A  ";".             */
IF SUBSTR(MENU_ENTRY,START,I-START) ¬= ' '
THEN DO;
  SUB = SUB+1;
  PART_TBL(SUB) = SUBSTR(MENU_ENTRY,START,I-START);
  /******** SEE IF CAND_FIELD IS AN EXISTING DATA ELEMENT ****/
```

```
      LMNT_NAME = PART_TBL(SUB);
      SAVE_DB_NAME = DB_NAME;
      OBTAIN CALC RECORD (DATA_ELEMENT);
      DO WHILE (ERROR_STATUS = REC_FOUND);
        OBTAIN OWNER SET (DEFINED_BY); CALL IDMS_STATUS;
        IF SAVE_DB_NAME = DB_NAME
        THEN
          ERROR_STATUS = 'FWND';
        ELSE
          OBTAIN DUPLICATE RECORD (DATA_ELEMENT);
      END;
      IF (ERROR_STATUS ¬= '0326' & ERROR_STATUS ¬= 'FWND')
      THEN CALL IDMS_STATUS;
      IF ERROR_STATUS ¬= 'FWND'
      THEN DO;
        PUT STRING (MSG) EDIT
          ('ENTRY ',SUB,' DOES NOT EXIST - USE MENU') (A,F(3),A);
        CALL MESSAGES;
        STATUS = 'BAD';
        RETURN;
        END;
      END;
  END DIV_ENTRY;

GEN_MENU:   PROC (MENU_NUM,MENU_ENTRY,MENU_MSG,TBL_SIZE,
                  SLCT_NUM,NUM_COLS);
/*****************************************************************/
/*  This module displays a menu that is provided in the global   */
/*  varible, DISPLAY_TBL.  The size of the DISPLAY_TBL is        */
/*  identified by the TBL_SIZE parameter.  The user makes a      */
/*  selection that is divided into MENU_NUM and MENU_ENTRY.      */
/*  The contents of DISPLAY_TBL is displayed on the screen       */
/*  in 1, 2, or 3 columns depending on the value of the          */
/*  NUM_COLS parameter.                                          */
/*****************************************************************/

        DCL       MENU_NUM                    CHAR(3),
                  MENU_ENTRY                  CHAR(69),
                  SLCT                        CHAR(72),
                  MENU_MSG                    CHAR(40),
                  (TBL_SIZE,NUM_SCREENS)      FIXED DEC(3),
                  (SLCT_NUM,I,J,NUM_CHOICES)  FIXED DEC(3),
                  SLCT_STATUS                 CHAR(4),
                  NUM_COLS                    FIXED DEC(1);

   /*   DISPLAY MENU  /  ACCEPT REPLY          */

        SLCT = ' ';

        IF NUM_COLS = 1
        THEN
          NUM_CHOICES = 15;
        ELSE IF NUM_COLS = 2 THEN
          NUM_CHOICES = 30;
```

```
        ELSE IF NUM_COLS = 3 THEN
          NUM_CHOICES = 45;
        ELSE
          DISPLAY ('PROGRAMMER ERROR IN GEN_MENU ')
            REPLY (ENTER_KEY);

        IF MOD(TBL_SIZE,(NUM_CHOICES - 1)) = 0
        THEN
          NUM_SCREENS = (TBL_SIZE/(NUM_CHOICES - 1)) - 1;
        ELSE
          NUM_SCREENS = (TBL_SIZE/(NUM_CHOICES - 1));
        SLCT_STATUS = 'BAD';
        DO WHILE (SLCT_STATUS = 'BAD');
          DO I = 0 TO NUM_SCREENS  WHILE (SLCT = ' ');
            MSG = MENU_MSG;
            CALL MENU_HEAD;
            DO J = 1 TO 15;
              IF NUM_COLS = 1 THEN
                PUT STRING (EDIT_OUT) EDIT
                  (DISPLAY_TBL(J+(I*NUM_CHOICES))) (A);
              ELSE IF NUM_COLS = 2 THEN
                PUT STRING (EDIT_OUT) EDIT
                  (DISPLAY_TBL(J+(I*NUM_CHOICES)),
                   DISPLAY_TBL(J + 15 + (I*NUM_CHOICES)))
                  (A(48),A(24));
              ELSE IF NUM_COLS = 3 THEN
                PUT STRING (EDIT_OUT) EDIT
                  (DISPLAY_TBL(J+(I*NUM_CHOICES)),
                   DISPLAY_TBL(J + 15 + (I*NUM_CHOICES)),
                   DISPLAY_TBL(J + 30 + (I*NUM_CHOICES)))
                  (3(A(24)));
              DISPLAY (EDIT_OUT);
            END;
            CALL BLANK_LINE(1);
            IF I <  NUM_SCREENS    /* I.E.  MORE DATA    */
            THEN
              DISPLAY ('MAKE SELECTION OR PRESS ENTER ===>')
                       REPLY (SLCT);
            ELSE
              DISPLAY ('MAKE SELECTION ===>')  REPLY (SLCT);
          END;
          CALL EXAMINE_ENTRY (SLCT,MENU_NUM,MENU_ENTRY,SLCT_NUM,
                              SLCT_STATUS,TBL_SIZE);
        END;
END GEN_MENU;

SLCT_VALUE:  PROC (DISPLAY_TBL,DOMAIN_TBL,
                   THE_VALUE,MENU_MSG,TBL_SIZE);
/*****************************************************************/
/* This module displays a menu that is provided in the DISPLAY_ */
/* TBL.  The user enters a number representing a menu selection */
/* and the value of that selection is moved to the output       */
/* parameter, THE_VALUE.                                        */
/*****************************************************************/
```

```
   DCL        DISPLAY_TBL(500)          CHAR(72),
              DOMAIN_TBL(500)           CHAR(16),
              THE_VALUE                 CHAR(16),
              MENU_MSG                  CHAR(30),
              SLCT                      CHAR(72),
              (TBL_SIZE,I,SLCT_NUM)     FIXED DEC(3),
              STATUS                    CHAR(4),
              MENU_NUM                  CHAR(3),
              MENU_ENTRY                CHAR(69);

   STATUS = 'BAD';
   DO WHILE (STATUS = 'BAD');
     MSG = MENU_MSG;
     CALL MENU_HEAD;
     DO I = 1 TO TBL_SIZE;
       DISPLAY (DISPLAY_TBL(I));
     END;
     DISPLAY (DISPLAY_TBL(I));
     CALL BLANK_LINE(2);
     DISPLAY ('====>')  REPLY (SLCT);
     CALL EXAMINE_ENTRY (SLCT,MENU_NUM,MENU_ENTRY,SLCT_NUM,
                         STATUS,TBL_SIZE);
   END;
   IF MENU_NUM ¬= 'X'
   THEN
     THE_VALUE = DOMAIN_TBL(SLCT_NUM);
END SLCT_VALUE;

EXAMINE_ENTRY:  PROC (SLCT,MENU_NUM,MENU_ENTRY,SLCT_NUM,
                      SLCT_STATUS,UP_LIMIT);
/*****************************************************************/
/* This module examines an entry made by the data base designer.*/
/* The SLCT parameter contains the value inputted by the data    */
/* base designer which is divided into MENU_NUM and MENU_ENTRY.  */
/* SLCT_NUM is the numeric equivalent to the MENU_NUM which is   */
/* of type character.                                            */
/*****************************************************************/

        DCL       SLCT                      CHAR(72),
                  MENU_ENTRY                CHAR(69),
                  MENU_NUM                  CHAR(3),
                  (UP_LIMIT,SLCT_NUM,I,J)   FIXED DEC(3),
                  SLCT_STATUS               CHAR(4);

   /*    CHECK FOR NULL ENTRY BY USER       */

        IF SLCT = ' '
        THEN DO;
          CALL GEN_INST;
          SLCT_STATUS= 'BAD';
          RETURN;
          END;
```

```
/*   LEFT JUSTIFY SLCT -- USER MIGHT SPACE BEFORE ENTRY   */

     DO WHILE (SUBSTR(SLCT,1,1) = ' ');
       DO I = 1 TO 71;
         SUBSTR(SLCT,I,1) = SUBSTR(SLCT,I+1,1);
         SUBSTR(SLCT,I+1,1) = ' ';
       END;
     END;

/*   CHECK FOR EXIT REQUEST          */

     IF SUBSTR(SLCT,1,1) = 'X'
     THEN DO;
       MENU_NUM = 'X';
       SLCT_STATUS = 'GOOD';
       RETURN;
       END;

/*   ROTATE RIGHT ALL CHARACTERS OUT OF FIRST 3 POSTITIONS THAT
     ARE NOT ' ' OR NUMERIC -- SET UP MENU_NUM                */

     DO J = 1 TO 3;
       IF (SUBSTR(SLCT,J,1) ¬= ' '  &
         ¬(SUBSTR(SLCT,J,1) >= '0'  &
           SUBSTR(SLCT,J,1)  <= '9'))
       THEN DO;
         DO I = 72 TO (J+1) BY -1;
           SUBSTR(SLCT,I,1) = SUBSTR(SLCT,I-1,1);
           SUBSTR(SLCT,I-1,1) = ' ';
         END;
         END;
     END;

/*    DIVIDE SLCT INTO MENU_NUM AND MENU_ENTRY    */

     GET STRING (SLCT) EDIT (MENU_NUM,MENU_ENTRY) (A(3),A(69));

/*   LEFT JUSTIFY MENU_ENTRY                      */

     IF MENU_ENTRY ¬= ' '
     THEN DO;
       DO WHILE (SUBSTR(MENU_ENTRY,1,1) = ' ');
         DO I = 1 TO 68;
           SUBSTR(MENU_ENTRY,I,1) = SUBSTR(MENU_ENTRY,I+1,1);
           SUBSTR(MENU_ENTRY,I+1,1) = ' ';
         END;
       END;
     END;

/*   EMBEDDED BLANKS -- MAKE POSITION 3 OF MENU_NUM PART OF
     MENU_ENTRY                                          */

     IF SUBSTR(MENU_NUM,3,1) ¬= ' '
     THEN DO;
```

```
            DO J = 69 TO 2 BY -1;
              SUBSTR(MENU_ENTRY,J,1) = SUBSTR(MENU_ENTRY,(J-1),1);
            END;
            SUBSTR(MENU_ENTRY,1,1) = SUBSTR(MENU_NUM,3,1);
            SUBSTR(MENU_NUM,3,1) = ' ';
            END;

    /*   CONVERT CHARACTER (MENU_NUM) TO NUMBER (SLCT_NUM)    */

         GET STRING (MENU_NUM) EDIT (SLCT_NUM) (F(3));

    /*   CHECK UPPER & LOWER LIMITS.  NOTE:  ALL BLANKS WOULD
      CONVERT TO ZERO AND WOULD THEREFORE BE INVALID.          */

         IF (SLCT_NUM > 0  & SLCT_NUM <= UP_LIMIT)
         THEN
           SLCT_STATUS = 'GOOD';
         ELSE DO;
           SLCT = ' ';     /*  NECESSARY TO REDISPLAY SELECTIONS */
           MSG = 'NOT A VALID MENU NUMBER ';
           CALL MESSAGES;
           SLCT_STATUS = 'BAD';
           END;
END EXAMINE_ENTRY;

EDIT_NAME:   PROC (NAME, STATUS);
/*****************************************************************/
/* This module receives NAME as input and attempts to make it    */
/* a compilable field name.  If the name cannot be made valid,   */
/* then the output parameter, STATUS, is set to "BAD".           */
/*****************************************************************/

         DCL   NAME                    CHAR(16),
               STATUS                  CHAR(4),
               (END_POS,I,J)           FIXED DEC(3);

         STATUS = 'GOOD';

         IF NAME = ' '
         THEN
             RETURN;


    /*  LEFT JUSTIFY NAME                                         */

         DO WHILE (SUBSTR(NAME,1,1) = ' ');
            DO I = 1 TO 15;
               SUBSTR(NAME,I,1) = SUBSTR(NAME,I+1,1);
               SUBSTR(NAME,I+1,1) = ' ';
            END;
         END;

    /*  LOCATE ENDING POSITION OF NAME.                           */
```

```
      END_POS = 16;
      DO I = 16 TO 1 BY -1  WHILE (SUBSTR(NAME,I,1) = ' ');
        END_POS = I-1;
      END;

   /*  VERIFY 1ST CHARACTER AS ALPABETIC                          */

      IF (SUBSTR(NAME,1,1) < 'A' | SUBSTR(NAME,1,1) > 'Z')
      THEN DO;
        MSG = 'FIRST POSITION OF DATA BASE NAME NOT ALPABETIC';
        CALL MESSAGES;
        PUT STRING (MSG) EDIT
          ('YOUR ENTRY WAS ==> ',NAME) (2(A));
        DISPLAY (MSG);
        STATUS = 'BAD';
        RETURN;
      END;

   /*  REDUCE IMBEDDED BLANKS TO DASHES   */

      I = 2;
      DO WHILE (I < END_POS);
        IF (SUBSTR(NAME,I,1) = ' ' | SUBSTR(NAME,I,1) = '-' |
            SUBSTR(NAME,I,1) = '_')
        THEN DO;
          SUBSTR(NAME,I,1) = '-';
          IF (SUBSTR(NAME,I+1,1) = ' ' |
              SUBSTR(NAME,I+1,1) = '-' |
              SUBSTR(NAME,I+1,1) = '_')
          THEN DO;
            DO J = I+1 TO END_POS - 1;
              SUBSTR(NAME,J,1) = SUBSTR(NAME,J+1,1);
              SUBSTR(NAME,J+1,1) = ' ';
            END;
            END_POS = END_POS - 1;
            END;
          ELSE
            I = I + 1;
          END;
        ELSE
          I = I + 1;
      END;

   /*  VERIFY POSITIONS 2 THROUGH END_POS AS A DASH, NUMBER, OR
       LETTER                                                     */

        DO I = 2 TO END_POS;
          IF ¬ (SUBSTR(NAME,I,1) = '-' |
                (SUBSTR(NAME,I,1) >= 'A' &
                 SUBSTR(NAME,I,1) <= '9'))
          THEN DO;
            PUT STRING (MSG) EDIT ('ERROR -- INVALID ',
             'CHARACTER IN POSITION',I,' OF ',NAME) (2(A),
              F(2),2(A));
```

```
            CALL MESSAGES;
            STATUS = 'BAD';
            RETURN;
          END;
        END;
      END EDIT_NAME;

GEN_INST:  PROC;
/*****************************************************************/
/*  This module displays the general instructions for inter-     */
/*  acting with DB_GEN.                                          */
/*****************************************************************/
/*

    SOME GENERAL USER INSTRUCTIONS FOR USING THIS PROGRAM
                                                                  */
 CALL CLRSCR;
 DISPLAY ('          ** GENERAL INSTRUCTIONS');
 CALL BLANK_LINE(2);
 DISPLAY (' TWO FORMATS CAN BE USED --');
 DISPLAY (' ');
 DISPLAY (' FORMAT 1:     <MENU_NUM>');
 DISPLAY (' ');
 DISPLAY
     ('==> THIS WILL PROVIDE DETAILED INSTRUCTIONS FOR ENTERING');
 DISPLAY ('    THE RESPECTIVE INFORMATION');
 DISPLAY (' ');
 DISPLAY (' FORMAT 2:      <MENU_MUN> <MENU_ENTRY>');
 DISPLAY (' ');
 DISPLAY
    ('==> THE DETAIL INSTRUCTION STEP IS SKIPPED BY ADDING THE ');
 DISPLAY ('    MENU ENTRY   E.G.  1  PART_NUMBER ');
 CALL BLANK_LINE(2);
 DISPLAY ('PRESS ENTER TO CONTINUE') REPLY (ENTER_KEY);
 END GEN_INST;

MESSAGES:  PROC;
/*****************************************************************/
/*  This module writes a message to the screen.  It displays     */
/*  the value of the global varible, MSG, and waits for the user*/
/*  to respond before continuing.                                */
/*****************************************************************/
          CALL CLRSCR;
          CALL BLANK_LINE(3);
          CALL ASTRICK_LINE(2);
          CALL BLANK_LINE(1);
          DISPLAY (MSG);
          CALL BLANK_LINE(1);
          CALL ASTRICK_LINE(2);
          CALL BLANK_LINE(3);
          DISPLAY ('PRESS ENTER TO CONTINUE') REPLY (ENTER_KEY);
          END MESSAGES;

MENU_HEAD:  PROC;
```

```
/*****************************************************************/
/* This module clears the screen and prints the "current" data   */
/* base name in the upper right hand corner of the screen.       */
/*****************************************************************/

        CALL CLRSCR;
        OBTAIN CALC RECORD (DATA_BASE);
        IF ERROR_STATUS = '0326'
        THEN
          DB_NAME = '????';
        ELSE
          CALL IDMS_STATUS;
        PUT STRING (EDIT_OUT) EDIT ('DATA-BASE:  ',DB_NAME)
                   (X(40),A,A);
        DISPLAY (EDIT_OUT);
        CALL BLANK_LINE(1);
        PUT STRING (EDIT_OUT) EDIT (MSG) (X(10),A);
        DISPLAY (EDIT_OUT);
        CALL BLANK_LINE(1);
        END MENU_HEAD;

BLANK_LINE:  PROC(NUM_LINE);
/*****************************************************************/
/* This module prints a blank line - used for screen formatting.*/
/*****************************************************************/
   DCL  (NUM_LINE,I,J)                FIXED DEC(3);

   DO I = 1 TO NUM_LINE;
      DISPLAY (' ');
   END;
   END BLANK_LINE;

ASTRICK_LINE:  PROC(NUM_LINE);
/*****************************************************************/
/* This module prints an astrick line - used for screen          */
/* formatting.                                                   */
/*****************************************************************/
   DCL  (NUM_LINE,I,J)                FIXED DEC(3);

 DO I = 1 TO NUM_LINE;
   DISPLAY ('************************************************');
 END;
 END ASTRICK_LINE;
 MSG = 'NORMAL PROGRAM TERMINATION';
 CALL MESSAGES;
 END DB_GEN;
```

DATA BASE DESIGN PRINCIPLES APPLIED TO A NETWORK MODEL

BY

MARK A. COSTELLO

B.S., Pittsburg State University, 1979

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

ABSTRACT

This thesis describes the automation of the data base design process by using the principles of schema normalization, the data dictionary concept, and a sound data base design methodology. The implementation of this data base design tool aids the data base designer in the monumental task of designing the user's data base schema.

This implementation uses an interactive menu driven system to aid the data base administrator throughout the entire data base design process. The initial step collects only necessary data (i.e., functional and nonfunctional dependencies) to generate the major data base entities. Once the major entities are generated the data base administrator is able to interactively customize the entities to best describe the users' needs. Finally the actual data base management system data definition statements representing the users' data base are generated.

This paper describes an implementation of this process using the PL/I Optimizing Compiler supported by the IDMS version 5.7 generalized data base management system. The system operates under IBM's CP/CMS operating system.