The Interactive Generation of Functional Dependencies

by

WILLIAM CLEON HUNI

E. G. S., University of Nebraska at Omaha
Omaha, Nebraska, 1972

A MASTER'S REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1978

Approved by: Major Professor
Acknowledgment

A special thanks to Dr. Fred J. Maryanski for allowing me to participate in the development and implementation of the system for the automatic generation of third normal form relations, and for his professional attitude during its development. Thanks to Dr. Paul S. Fisher and Edwin W. Basham for serving as members of my committee. The implementation was made possible through a grant from NCR, Inc.
Table of Content

1.0 Introduction ........................................... 1
2.0 Background ............................................. 4
2.1 The Problem of Third Normal Form ................. 5
2.2 Project ................................................. 6
2.3 FDGEN .................................................. 7
3.0 Functional Dependency Generator ................ 9
3.1 The FDGEN Algorithm ................................ 10
3.2 Amplification of the FDGEN Algorithm ........... 17
4.0 Implementation ......................................... 41
4.1 Walk-Through .......................................... 41
4.2 Testing ................................................. 76
5.0 Conclusion ............................................. 94
5.1 Enhancement Considerations ......................... 94

Reference and Bibliography ............................. 97

Appendices

Abstract Code ............................................... A-1
Implementation - FDGEN Code - NCR Level II COBOL .... B-1

List of Figures

System Diagram ............................................ 8
Input File Function and Format ........................ 11
Data Name Table Function and Format ................ 12
Functional Dependency Table Function and Format ... 16
Output Format of Functional Dependencies ............ 17
Hierarchical Structure of Data-Name-Items ............. 19
Input File Example ........................................ 20
Data Name Table Example ................................ 22
Sample Extracts of an Input File and Data Name Table with status 'S' and 'N' ................. 25
Link Stack (Concatenation Stack) Format and Function 28
Storage Stack Format and Function .................... 30
Save Stack Format and Function ........................... 31
Functional Dependency Output File Format and Function 39
Input File .................................................. 44
Concatenation Code Table .................................. 46
Logical Internal View of the Data Name Table ............ 47
Update Section of the Data Name Table ................. 52
Update Logical View of the Data Name Table .......... 55
Logical View of the Concatenation and Storage Stack .. 62
Logical View of the F-D-Table ............................. 62
Current View of the Concatenation and Storage Stacks . 64
Current View of the Save Stack and F-D-Table .......... 68
Current View of the Concatenation and Storage Stacks .... 70
Current Logical View of Internal Tables ............... 72
Final View of Program Generated and Updated Tables ... 74
Functional Dependency Output Format .................. 75
Walk-Through Generated Functional Dependencies .... 75
1.0 INTRODUCTION

Information is a prime commodity in today's hurried existence. Managers make decisions based on the best information available to them at the moment. Strategic planning is based on information compiled from many different sources. Every aspect of modern day life is influenced by information. Information is often related to power.

What is information? It is the meaningful interpretation of data which has been collected and integrated. (1) To meet the needs of today's society data must be collected in great quantities. In order for massive amounts of data to be useful to a user, he must have real-time access to it and have a means of synthesizing the available data into meaningful collections so that it might be interpreted into useful information.

Systems have been developed which provide the user with mass storage of data, quick access to the data, and a means of synthesizing the accessed data. The systems, known as Data Base Management Systems (DBMS), allow a user to retrieve, manipulate, and store data without requiring him to know how the data is organized or stored within the storage device.

The user views the data in prespecified logical groups linked together in some organized structure. The more common DBMS structures are known as network and hierarchy. In such views, data may be represented in three ways: (2)
1. The content of the record (e.g., Maryanski's Dept = CS.);
2. The linkage between records (e.g., Maryanski's Dept record occurs in the hierarchy below the College Record for College of Arts and Science.); and
3. The ordering of records (e.g., All of the department records are stored in alphabetical order.).

Usage of such DBMS requires that the user have knowledge of the representation chosen (e.g., FIND NEXT RECORD of College-Dept Set).

The most recent addition to the DBMS family is represented by the Relational Data Model which makes possible the elimination of the representation-dependency from the user's interface. The relational model represents information at the user's interface only by data value. This representation-independence within a DBMS has generated enough interest within the industrial community that research is being funded within the area.

Critical to the development of a Relational Data Base is a user knowledgeable of the data-items which will comprise the data base. Data-items are the smallest logical unit of data within a data base. Data-items with the same characteristics are categorized by Data-Name-Item (Attribute). Each Data-Name-Item (DNI) has a domain within which are all data-items described by it. Sets of Data-Name-Items are combined to form relations or groups of
related Data-Name-Items. Each relation must possess one or more Data-Name-Item which can be used as a key to reference the other Data-Name-Items of the group. The association between the key Data-Name-Items and other Data-Name-Items in the group is referred to as functional dependence. The non-key Data-Name-Items are said to be functionally dependent upon the key Data-Name-Items. The set of Data-Name-Items is known as a Functional Dependency and the data base developer must be able to recognize them within his data.

The implementation of a DMS based upon the Relational Data Model requires the development of a system which encompasses a minimum of three critical areas.(3)

1) A subsystem to accept the unsophisticated user's data and organize it for processing.
2) A subsystem to interact with the user, thus guiding him in the generation of Functional Dependencies.
3) A subsystem to automatically synthesize the generated Functional Dependencies into Third Normal Form (3NF).

This report presents the design, implementation and testing of the interactive subsystem, Functional Dependency Generator (FDGEN), which enables the user to generate Functional Dependencies (FD).
2.0 BACKGROUND

A user interfaces with a DBMS through an application program or query language. The program provides the user with predefined views of data, independent of their stored configuration. The life expectancy of application programs is significantly increased when used with relational DBMSs. (4)

A relational database provides for the collecting of data in a relation and the formation of logical associations between the relations.

The purpose of a relational database is fourfold: (5)

1. To free the collection of relations from undesirable insertion, update, and deletion dependencies;
2. To reduce the need for restructuring the collection of relations as new types of data are introduced, and thus increase the life span of application programs;
3. To make it feasible to represent any relation in the database; and
4. To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.

Before a Relational Database can fulfill its purpose its relations must be in Third Normal Form.
2.1 The Problem of Third Normal Form

To meet these requirements of a relational data base each data relation must be in Third Normal Form. Each must meet the following test: (6)

1. Each data-item in the relation must be based on a simple domain (it must be atomic);
2. Within each relation data-items are separated into key data-items and non-key data-items. Every non-key data-item must be fully dependent on every key data-item in the relation;
3. Every non-key data-item in the relation must be independent of all other non-key data-items in the relation; and
4. Every key data-item in the relation must be fully dependent on every other key data-item in the relation of which it is a part.

A data relation cannot be tested to see if it is in 3NF before its Functional Dependencies have been identified. A Functional Dependency among the data-items of a relation must be of the following form: (7)

A data-item D of relation C is functionally dependent upon data-item E if, at every instant of time, each B-value in C is associated with only one D-value. The relationship is expressed by the notation B --> D, and says that B determines D or D depends on B. Likewise,
a set of data-items in C may be functionally dependent on another data-item or set of data-items. The data-item (or set of data-items) on the left side of the arrow (B) is called the Determinant.

A user often experiences difficulty trying to determine the Functional Dependencies among the data-items of a data relation, thus increasing the probability that the schema designed would not meet the requirements of a relational data base.

2.2 PROJECT

The automatic generation of 3NF relations is a major step toward the development of relational data bases. NCR, Inc. provided a grant to fund the development of a prototype system which would interactively guide an unsophisticated user in the development of 3NF relations. The system design provided for four subsystems: (8)

1. The User Interface subsystem - An interactive program which reformats input data into hierarchical records, called reports, based on responses to queries from the system and produces an output file of records;
2. The FDGEN subsystem - An interactive program which inputs the data produced by the User Interface subsystem, interactively generates Functional Dependencies and produces as output a file of Functional Dependencies and a hardcopy listing of
Functional Dependencies for the user;
3. The Functional Dependency Analysier (FDA) subsystem - A program which inputs the data produced by the FDGEN subsystem and synthesis the Functional Dependencies into 3NF relations. It produces as output a hardcopy of the 3NF relations; and
4. The Editor subsystem - An interactive program which provides the user with the means to insert, delete, or change any portion of the original input file.

The diagram of the system is pictured in figure 2-1.(9)

2.3 FDGEN

The generation of Functional Dependencies (FD) is not an automatic process. It requires the interaction of a user who is familiar with the data-items input into the system. Data-items entered as a group must relate to each other so that a logical relation can be established. The logically related data-items become the input to the FDGEN. The subsystem defines terms, provides instructions, and directs the user in the manipulation of the related data-items enabling him to form the desired FD. The generated FD's are stored in an output file for later use as input for the FDA.
System Diagram

Figure 2-1
3.0 FUNCTIONAL DEPENDENCY GENERATOR

The Functional Dependency Generator (FDGEN) is an interactive program which accepts as input a file generated by the User Interface Program and produces as output a Functional Dependency Table (F-D-Table) which is stored on disc and used as input for the Functional Dependency Analyzer Program (FDA). (10)

The FDGEN processes the input data and constructs a Data Name Table (DNT) which is used throughout the remainder of the program. The processed data is presented to the interactive user along with the necessary instructions to guide him through the various phases of functional dependency generation. All processed data has associated with it a hierarchical code which allows distinctive data relations to be generated and presented to the user. The user's primary objective is to redefine the generated relations into logical Functional Dependencies (FD). Data manipulation facilities are provided which allow the necessary reconstruction of relations. A hardcopy of the generated FD's is provided to the user upon completion of data manipulation.

An example of a reconstructed relation is illustrated by the following relation with Data-Name-Items (DNI) OWNER, HOUSE, CAR, BUSINESS, and FARM. The user would manipulate the DNI's so that the following FD is generated and output.

OWNER --> HOUSE CAR BUSINESS FARM
The FD says that OWNER identifies the other Data-Name-items
within the relation.

3.1 THE FGEN ALGORITHM

The algorithm for the generation of FD's is separated into eleven logical steps. (11)

Step 1. Read the next record from the Input File into a buffer. The format and function of the Input File are provided in figure 3-1.

Step 2. Delete the first four characters from the buffer. Assign the fifth and sixth characters to Status and level respectively. Separate the remaining data in the buffer and assign each delimited set of characters to a unique position within the DNT. Assign appropriate values to the associated fields. Link each level '0' position to its successor's level '0' position. Mark the end of the DNT. The format and function of the DNT are provided in figure 3-2.

Step 3. When all records have been read and all data has been stored in the DNT display each DNI one at a time. Ask the user to identify all Virtual-Data-Items. Enter a code '3' in the determ-flag field of the DNT for each DNI identified as a Virtual-Data-Item. A Virtual-Data-Item is a DNI which is
Input File Function and Format

Figure 3-1

<table>
<thead>
<tr>
<th>Line no</th>
<th>Status</th>
<th>Level</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXX</td>
<td>X</td>
<td>X</td>
<td>X-70-X</td>
</tr>
</tbody>
</table>

Input File Format

Function of the Input File: To store user input text (DNI's) with their associated attributes (the level of the group, its status, and the group's position within the table (line no)).
Data Name Table Function and Format

Figure 3-2

<table>
<thead>
<tr>
<th>Name</th>
<th>Level-no</th>
<th>Link-code</th>
<th>Determ-flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-10-X</td>
<td>X</td>
<td>XXX</td>
<td>X</td>
</tr>
</tbody>
</table>

Data Name Table Format

Function of the Data Name Table: Initially to store in the proscribed format the input data from the Input File as well as the associated special link codes (concatenation links) which are derived from the associated level-no and status. Updated codes will be entered as DNI's are defined through user interaction.
derivable from other DNI's within the data base. The Virtual-Data-Items selected will be eliminated from the report. Figure 4-4 pictures an example where the DNI PRICE has been identified as a Virtual-Data-Item and a code '3' has been entered.

Step 4. Identify potential Determinants by performing an intersection by DNI cf each report with every other report in the DNI. For each intersected DNI enter a code of '2' in its associated determ-flag field in the DNT. An example of intersected DNI's is pictured in figure 4-5.

Step 5. Display for the user the list of all DNI's of the current report which have associated with them identical link codes. If a displayed DNI has code '2' in its associated determ-flag field display an asterisk to its right. The asterisk identifies a DNI as a potential Determinant of the displayed group. The user must identify one or more (possibly all) DNI's as Determinants of the group. An example of a marked intersected DNI is pictured with its group on page 63.

Step 6. If the current group from which the Determinant was selected has a level greater than the levels of previously considered groups from the current report, then Determinants for the previously considered groups must be concatenated to the current Determinant to form the Determinant Set for
the current group. The concatenated Determinant
Set MODEL, ENG-NC, STYLE, EXTERIOR, and INTERIOR is
shown in the Concatenation Stack pictured in figure
4-8.
(b) If the group from which the Determinant was
selected has a level greater than one but less than
or equal to any level of a previous group from the
current report, then those group Determinants whose
levels are greater than or equal to the current
group's level must be discarded and the remaining
Determinants concatenated to the Determinant of the
current group. This situation can best be
illustrated by example: Consider the Determinant
set in figure 4-8. If the next Determinant
selected by the user has associated with it a
concatenation link code of '22' all Determinants
with codes of '21' would be discarded from the
Concatenation Stack and the new Determinant would
be concatenated with MODEL to form the new Set.

Step 7. If two or more group Determinants have been
concatenated, then display the Determinants and
mark the most current with an asterisk. Provide
the user with the following options:
 a) Elect to leave the determinant configuration
 unchanged.
b) Elect to redefine all non-current Determinants
 as non-Determinants.
c). Elect to select one or more Determinants to be redefined as non-determinants. An example of such a display is pictured on page 67.

Step 8. If two or more group Determinants have been concatenated and they identify themselves, then display the concatenated Determinants and mark with an asterisk those Determinants from the current group. Provide the user with the following options:

a). Elect not to change the determinant configuration.

b). Elect to redefine all non-current Determinants as non-determinants.

c). Elect to select one or more Determinant to be redefined as non-determinants.

The queries which provide the user options are shown on pages 65 and 66. The Determinants are marked the same as those displayed on page 67.

Step 9. Upon selection of an option by the user, add the new FD to the F-E-Table. The FD is added to the table in four steps as follows:

a) Enter the number of Determinants.
Functional Dependency Table Function and Format

Figure 3-3

<table>
<thead>
<tr>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>no. of Determinants</td>
</tr>
<tr>
<td>Determinant Names</td>
</tr>
<tr>
<td>no. of non-determinants</td>
</tr>
<tr>
<td>non-determinant names</td>
</tr>
<tr>
<td>X-10-X</td>
</tr>
</tbody>
</table>

Functional Dependency Table Format

Function of the Functional Dependency Table: To store the Functional Dependencies generated by the program through interaction with the user. Storage will be sequential from the low index and will begin with a numeric (N), followed by (N) Determinant names, another numeric (M), followed by (M) DNI's. This sequence of events is repeated for each FD to be stored. It is the sole source of input for the FTABLE output file, and the secondary output of the FGEN program when written to the line printer.
b) Enter the Determinant names.
c) Enter the number of non-determinants.
d) Enter the non-determinant names.

The F-D-Table format and functions are provided in figure 3-3.

Step 10. Store the F-D-Table on disc for future use as input for the FDA.

Step 11. Restructure the contents of the F-D-Table by FD and output each FD in the format shown in figure 3-4.

OUTPUT FORMAT OF FUNCTIONAL DEPENDENCIES

FIGURE 3-4

DETERMINANT STRING ---> NON-DETERMINANT STRING

The abstract code for the FDGEN algorithm is provided in appendix A.

3.2 AMPLIFICATION OF THE FDGEN ALGORITHM

A detailed explanation of data flow, data manipulation, and user interactions are provided in amplification of the FDGEN Algorithm.

The FDGEN Program accepts as input a current maximum of 200 Data-Name-Items. These DNI's are extracted from the
input file which was produced by the User Interface Program. The file is logically a table of records of four fields. The first field contains a four digit line number which is discarded since it serves no purpose in the FCEGEN Program. The second field contains the status of the text field. The status consist of one alpha-character and is used to initiate link code generation. The third field contains the relative level number of the text in the text field. The level number is relative only within the current report. The level number is used to develop a hierarchy among the groups of DNI's within a report. It is analogous to a PL/1 structure or a CCECL record. A picture of a hierarchical structure of DNI's is provided in figure 3-5.

The last field is the text field which contains the DNI's. The data within the text field is delimited by spaces. At least one space must separate each DNI; however, spaces may also precede the first DNI as well as follow the last one.

The record shown in figure 3-5 is depicted in the format of the Input File and pictured in figure 3-6.

The Input File is read one record at a time until all records have been read. Following the reading of each record the status and level number are extracted and saved for later use, then the DNI's are extracted one at a time from the text field.
Hierarchical Structure of Data-Name-Items

Figure 3-5

Record Automobile
  01 Make
  01 Model
    02 Body-no
    02 Interior
    02 Exterior
    02 Style
    02 Eng-no
      03 Weight
      03 Cost
      03 Price
Input File Example

Figure 3-6

<table>
<thead>
<tr>
<th>LINE</th>
<th>S</th>
<th>L</th>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td></td>
<td>Automobile</td>
</tr>
<tr>
<td>0010</td>
<td>Y</td>
<td>1</td>
<td>Make Model</td>
</tr>
<tr>
<td>0020</td>
<td>Y</td>
<td>2</td>
<td>Body-no Interior Exterior Style Eng-no</td>
</tr>
<tr>
<td>0030</td>
<td>Y</td>
<td>3</td>
<td>Weight Cost Price</td>
</tr>
</tbody>
</table>

LINE: Line Number.
S: Status.
L: Level Number.
Text: Data-Name-Items.
The data extracted from the Input File is used to build the DNT, a sample of which is pictured in figure 3-7. The DNT is the major data structure within the program. Data stored in this table is used to service all other structures of the program.

The name field is limited to ten characters; however, it is easily expanded. The level number field may contain any one digit integer while the concatenation link field may contain the characters 'NUL' and any three digit integer. The determ-flag field is restricted to a one digit integer.

The DNT construction is continuous while data is being read from the Input File and the text field data is being separated. As each DNI is extracted it is placed in the name field of the DNT. If a DNI exceeds the allowable number of characters, an error message will be output and all excess characters will be truncated. The level number for the current record is placed in the level number field. The status does not become a part of the Data Name Table, but is used to initiate code updates.

Each status field may contain one of four characters: 'Y', 'C', 'S', or 'N'. The characters 'S' and 'N' alert the program that the normal hierarchical structuring is about to be interrupted. The character 'Y' simply allows the normal hierarchical structuring to continue while the 'C' causes the program to maintain the current level of structuring. (13) The character 'S' instructs the program to maintain the current level but
Sample Data Name Table

Figure 3-7

<table>
<thead>
<tr>
<th>Name</th>
<th>Level Number</th>
<th>Concatenation Link</th>
<th>Determinant Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automobile</td>
<td>0</td>
<td>NUL</td>
<td>0</td>
</tr>
<tr>
<td>Make</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Model</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>Body-no</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>Interior</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>Exterior</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>Style</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>Eng-no</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>Weight</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>Cost</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>Price</td>
<td>3</td>
<td>31</td>
<td>3</td>
</tr>
<tr>
<td>END</td>
<td>XXX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
that the current record is not a continuation of the prior record. The 'N' causes the program to accept the level number provided by the user, but unless the level number is 1, not to consider the current record as a continuation of any prior records that might possess the same level number. These various conditions control the values that are placed in the concatenation link field.

When a DNI has a level number of 0 and no status, it is assumed to be a report name. The name and level number are entered into the table. The concatenation link field is used to link consecutive reports together. The concatenation link field associated with the current report name will receive the characters 'NUL'. If a predecessor report exists, then the address of the current report is placed in the concatenation link field associated with the predecessor report name. If however, the predecessor report exists in name only (no DNI's associated), an error message will be output and the user will be allowed to delete the erroneous report.

The remaining DNI's associated with the current report are entered in sequence of extraction and the associated concatenation link field is loaded from the Concatenation Code Table which is indexed by the level number. The table is reinitialized at the beginning of each new report. It has nine elements and each element contains a two digit integer. The first element contains 11 and each successive element receives the value of its predecessor plus 10.
For each occurrence of a status, 'S' or 'N', the Concatenation Code Table is updated. The update is equivalent to the movement of a pointer to the next sibling in an iteration traversal of a hierarchical tree structure. The table is updated at the current index (level number) and all succeeding elements by adding 1 to the integer value of each element. To illustrate the table's use, a sample Input File with status 'S' and 'N' and the corresponding DNT are pictured in figure 3-8.

The determ-flag field of each entry is initially set to zero. Following the insertion of the last entry, the end of the table is marked by insertion of an 'END' in the name field and 'XXX' in the concatenation link field. See figure 3-7 for an example of a closed table.

The program provides the user with a tutorial of Virtual-Data-Items and the necessary instruction to allow him to identify Virtual-Data-Items. The only user responses throughout this phase of the program are 'YES' and 'NO'.

The DNI's are read from the DNT one at a time and displayed for the user. Each DNI displayed is also identified by report name. This phase, in essence, provides the user with a sequential view of the DNT.

The user is asked to interact with the program by responding with a 'YES' or 'NO' to the question, is this DNI a Virtual-Data-Item. A 'NO' response allows the program to cycle to the next DNI in sequence;
Sample Extracts of an Input File and Data Name Table
with Status of 'S' and 'N'

Figure 3-8

<table>
<thead>
<tr>
<th>Line</th>
<th>S</th>
<th>I</th>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0060</td>
<td>Y</td>
<td>2</td>
<td>text1 text2</td>
</tr>
<tr>
<td>0070</td>
<td>Y</td>
<td>3</td>
<td>text3 text4</td>
</tr>
<tr>
<td>0080</td>
<td>S</td>
<td>3</td>
<td>text5 text6</td>
</tr>
<tr>
<td>0090</td>
<td>N</td>
<td>2</td>
<td>text7 text8</td>
</tr>
</tbody>
</table>

Extract of an Input File

<table>
<thead>
<tr>
<th>Name</th>
<th>L</th>
<th>C-L</th>
<th>D-F</th>
</tr>
</thead>
<tbody>
<tr>
<td>text1</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>text2</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>text3</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>text4</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>text5</td>
<td>3</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>text6</td>
<td>3</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>text7</td>
<td>2</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>text8</td>
<td>2</td>
<td>22</td>
<td>0</td>
</tr>
</tbody>
</table>

Extract of a corresponding Data Name Table Segment
however, if a 'YES' response is recorded, the value in the
determ-flag field associated with the selected DNI is
changed to '3'. The DNI is eliminated from further
consideration by virtue of the code '3'. Following the
update of the code the program cycles to the next DNI.

When all DNI's have been listed and the user has
responded, he is provided the opportunity to verify those
selected Virtual-Data-Items. If a 'YES' response is
recorded, the program searches the DNT in sequential order
for DNI's whose associated determ-flag field contains the
value '3'. For each code '3' found, the program displays
the associated DNI and asks the question; 'is this a
Virtual-Data-Item'. If the user's response is 'YES' the
program continues its search until it reaches the end of the
table, but if the response is 'NO' the value within the
associated determ-flag field is changed to '0'. This
facility allows a user to correct a possible selection
error. If the user had elected not to verify the list of
Virtual-Data-Items by entering 'NO' the program would have
asked if the user desired to review a listing of the DNI's
again. If a 'NO' response is recorded the program continues
to the next phase, but if a 'YES' response is recorded the
program returns to the beginning of the virtual
identification phase and begins the cycle once again. The
only new aspect to this cycle is that for each occurrence of
a Virtual-Data-Item the program will inform the user that
the displayed DNI has previously been selected as a Virtual-
Data-Item. The user may restore Virtual-Data-Items to DNI's only during the verify subphase of the program.

The intersection phase of the program performs a pairwise comparison of every DNI in every report with the DNI's in every other report.(14) Each pairing of DNI's which produces a match causes the value in both associated determ-flag fields to be changed to '2'. DNI's with an associated determ-flag field value of '3' are not considered for pairing. The pairing for comparison is facilitated by the report linkage established in the DNT during the construction of the table. This linkage allows the DNI's of the first report to be paired and compared with the DNI's of all succeeding reports. Upon completion of each current report, its successor report becomes the current report and its DNI's are paired and compared with the DNI's of all succeeding reports. This process is continued until the last report becomes the current report. The last report is identified by having the characters 'NUL' in the determ-flag field associated with the report name. The end of the DNT is identified by the values stored in the name and determ-flag fields, 'END' and 'XXX' respectively.

Four data structures which are used during the determinant identification phase must be identified prior to presenting a detailed explanation of this phase.

1. The Concatenation Stack is a table of records with two fields. Its format and function are shown in figure 3-9.
Link Stack (Concatenation Stack) Format and Function

Figure 3-9

<table>
<thead>
<tr>
<th>E-Name</th>
<th>Lev</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-10-X</td>
<td>XX</td>
</tr>
</tbody>
</table>

Link Stack Format

Function of the Link Stack: To store the selected Determinant and its level code. The level code is synonymous to the concatenation link code of the DNT. It stores the Determinant Set for the current report. It provides a source of input to the E-C-Table.
2. The Storage Stack is a table of elements. The format and function of this structure are provided in figure 3-10.
3. The Save Stack is identical to the Concatenation Stack except that its field names have been changed. Its format and function are presented in figure 3-11.
4. The Functional Dependency Table's function and format are provided in figure 3-3.

The determinant identification phase is highly interactive and requires input from the interacting user in order to generate the resultant FD's. The program cycles through each report and group in the same manner. Different codes associated with various groups may trigger different reactions to each group; however, each group is provided an equal opportunity to activate each facility within this phase.

The program begins this phase by displaying a brief tutorial on Determinants and asking the user if a further explanation is desired. See display samples on page 56. A 'YES' response will provide the user
Storage Stack Format and Function

Figure 3-10

<table>
<thead>
<tr>
<th>ATT</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-1C-X</td>
</tr>
</tbody>
</table>

Storage Stack Format

Function of the Storage Stack: To store DNI's of a particular group as they are extracted from the DNI. To store a copy of the group Determinants as selected by the user. To provide a source of input to the F-D-Table.
Save Stack Format and Function

Figure 3-11

<table>
<thead>
<tr>
<th>S-Name</th>
<th>Slev</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-10-x</td>
<td>XX</td>
</tr>
</tbody>
</table>

Save Stack Format

Function of the Save Stack: To store the Determinant Set associated with the predecessor group of the report being considered. It provides a source of input to the Link Stack, but is seldom used.
with a view of an example of a Determinant as it identifies a set of DNI’s. Upon a signal from the user the program initiates determinant selection. It begins by extracting information from the first report in the DNT. The report name displayed is followed by a set of instructions which explain how to use the facility. See sample displays on pages 57 and 58. To the left of the instructions are displayed all DNI’s from the current report which have the same concatenation link code. A sample display is shown on page 59.

Link codes are generated in a hierarchical sequence beginning with an initial code of '11'. The program compares the generated code with that stored within the detemr-flag field of the DNT. When a match is found the DNI is extracted from the associated name field and displayed in addition to being pushed onto the Storage Stack. The program searches the report sequentially for other matching codes. Associated determ-flag fields are checked at each occurrence of a matched code. If the determ-flag field value is 2 the associated name field values has been matched with another DNI of another report and an asterisk is displayed adjacent to the DNI. When all identical codes have been found and their associated DNI extracted, the user is asked to identify the Determinants which will uniquely identify the remaining DNI’s of the group. The user is instructed to first consider as possible Determinants, all DNI’s which are marked by an asterisk.
The user enters the selected Determinant names one at a time as directed by the program. DNI's spelled incorrectly will not be accepted and will produce an error message. When all Determinants have been entered, the user must respond to the next response by entering 'XX'. See example on page 63. This response signals the program to enter the next segment.

As each Determinant is entered it is pushed onto the Storage Stack and a count of their number is maintained. Additionally, the Determinant is pushed onto the Concatenation Stack following some minor investigations. The investigations consist of determining the level of any previously loaded Determinants. The process is as follows:

1. If no prior Determinant from this report is on the stack, then push the Determinant onto the stack along with its concatenation code (code);
2. If a Determinant has been loaded and its code was '1', then push the Determinant onto the stack with its code;
3. If a Determinant has been loaded and its code is less than or equal to the current Determinant's code, then push the Determinant onto the stack with its code; or
4. If a Determinant has been loaded and its code is greater than the current Determinant's code, then pop the stack until a Determinant is found with a level number value less than the level of the current
Determinant. Push the current Determinant onto the stack with its code.

Following the concatenation of Determinants in the Concatenation Stack, those Determinants which were pushed onto the Storage Stack are deleted along with all matching DNI's on the stack. The remaining DNI's represent the non-determinants of the current group. An example of a Storage Stack before and after deletion of Determinants is pictured in figure 4-8. Following the deletion of Determinants, if the Storage Stack is empty, a special situation exists. A case of all Determinants identifying themselves will have been detected. Two alternatives are presented to the user.

1) Elect to leave the Determinant configuration as it exists; in which case the program will generate a unique symbol and push it onto the Storage Stack. This will identify the FD as one which is self-identifying. The symbol consists of a colon (:) followed by a unique two digit integer. The first integer will be '01'. If Determinants from only one group exist this option is automatically selected. See example in figure 4-20.

2) Elect to change some or all of the Determinants to non-determinants.

   a) The decision to change all non-current Determinants to non-determinants is signified by the user entering 'YES'. This action initiates the movement of DNI's from the Concatenation Stack to the Storage Stack. The value of each Determinant's
code is compared with the current code. If they do not match the Determinant is removed from the Concatenation Stack to the Storage Stack. This process is repeated until all DNI's in the Concatenation Stack have been checked. Following the transfer, the remaining DNI's in the Concatenation Stack are compacted to the bottom of the stack.

b) The decision to individually select those Determinants which are to be restored to non-determinants is initiated simply by pressing the carriage return. This response is answered by a new set of instructions which tell the user how to enter the selected Determinant name and how to signal the program that no further entries are to be made.

As each selected Determinant is entered, the program searches the Concatenation Stack from the bottom until the Determinant is found. If the Determinant is not found a spelling error has been detected and the user is asked to enter the name again.

The identified Determinant is deleted from the Concatenation Stack and pushed onto the Storage Stack. The Concatenation Stack is compacted to insure contiguous storage. The left side count, number of Determinants, is decreased by one and the right side count, number of non-determinants, is increased by one. This process is repeated until the
user enters 'XX'. This signals the program that no more entries are to be made.

A second special case will exist, if following the deletion of Determinants from the Storage Stack, the Storage Stack is not empty and the Concatenation Stack contains a set of concatenated Determinants. This special case may also introduce a third special case which will be discussed on page 37.

The second special case detects the fact that Determinants from more than one group have been concatenated. The program provides the user an opportunity to manipulate Determinants once again.

A list of all Determinants in the Concatenation Stack are displayed along with an asterisk to mark the most recently selected Determinants. The user is offered the same Determinant manipulation options as he was in the first special case. He is permitted to select the marked Determinants as well as the unmarked. See sample display on page 67. This facility allows the user to alter the hierarchical structuring of Determinants within the report by allowing him the option of selecting the Determinant Set for the current group without regard to the generated hierarchical structure. The option is exercised by selecting from the list those Determinants which are to be restored to non-determinants. This action causes the selected DNI's to be transferred to the non-determinant side of the FD.
The third special case can be detected only within the constraints of the second special case. First there must exist a set of concatenated Determinants and secondly, the current group and its predecessor must be at the same level but have different codes. This case is representative of a tree structure where the parent node has two or more successor nodes. The successor nodes are at the same level but neither is a continuation of the other. The normal hierarchical structure is depicted by an n-ary tree structure, but there exists the possibility that the user elected, during the Determinant manipulation of the last group, to change the ancestors in the Determinant Set to non-determinants; breaking the hierarchical chain. This situation would mean that there would not be a hierarchical structuring of Determinants which could be concatenated to the current Determinant to form a complete Determinant Set. The hierarchy, if destroyed, is reconstructed when the third special case is detected. An example of the third special case can be observed in figure 4-21. NOTE: The third FD generated broke the hierarchical chain by defining 'A' as a non-determinant. The chain was restored and broken again in the fourth FD and reconstructed once again in the fifth. This example incorporates the first, second, and third special cases.

The program, upon detecting the third special case, compares the code values of the Determinants in the Save Stack with those of the Concatenation Stack. All
Determinants in the Save Stack with codes smaller than the smallest in the Concatenation Stack are moved to the bottom of the Concatenation Stack. They are moved one at a time to the Concatenation Stack which is now being used as a DEQUE. The current data in the stack is moved one position higher for each addition. All counters are updated to reflect all changes. The hierarchical structure will have been reconstructed following the movement of the required Determinants, thus allowing a continuation of the second special case processing.

The user's interaction with the program ends following the special case processing. The program has continually added FD's to the F-D-Table as the user directed their construction. The end of the F-D-Table is marked by inserting a string of 1's. See sample F-D-Table in figure 4-12.

The completed F-D-Table is copied to the Functional Dependency Output File on disc for future use as input for the FDA. The Output File format and function are provided in figure 3-12. The remaining task of FDGEN is to provide the user with a hardcopy of the FD's generated. See sample output in figure 4-14.

The program begins reading data sequentially from the low end of the F-D-Table. The first element read is an integer (N) which tells the program
Functional Dependency Output File Format and Function

Figure 3-12

F-L-Table-ele
X-10-X

Functional Dependency Output File Format

Function of the Functional Dependency Output File:
To store a copy of the Functional Dependencies stored in the F-L-Table. It is the sole source of input to the FDA Program and the primary output of the FGEN Program.
how many Determinants follow. The Determinants are read and concatenated onto a string delimited by one space. An arrow '--->' is concatenated onto the string following the Determinants. The F-D-Table pointer is incremented by one and the next element is read. Its value is an integer (M) which tells the program the number of non-determinants to follow. If (N+M+1) is not greater than twelve the non-determinants are read and concatenated onto the string which is then written to the line printer. If the value of (N+M+1) is greater than twelve the FE will exceed one line. The number of lines is determined based on a maximum of twelve elements per line. Each line is printed as it is generated from the F-D-Table. The program terminates when it encounters the string of X's which marks the end of the F-D-Table.
4.0 IMPLEMENTATION

The Functional Dependency Generation Program was implemented on an NCR 8250 System using NCR interactive CCBCL version 02.02.02 running under IMOS. A complete copy of the COBOL code is provided in appendix E. Included in the IMOS software package was the operating system, level II COBOL compiler, runtime interpreter, text editor, and interactive debug modules. (15)

Hardware:

The 8250 hardware consist of the following:

1 6850 processor
128K bytes MOS memory
1 cassette drive
2 Hawk 656 disc drives
2 796-101 CRT's
1 349-300 line printer
1 368 card reader
2 Asynchronous adapters (2400 baud)
1 battery back-up.

The program inputs data from an Input File, accepts input interactively, outputs data to a Functional Dependency Output File, and manipulates data in five internal tables. These tables were described in chapter 3.

4.1 WALK-THROUGH

The Functional Dependency Generator (FDGEN) was designed and implemented as an interactive program. The minimum interactions a user might have with the program is
seven, plus one additional for each Data-Name-Item (DNI) entered. The maximum interactions would depend on the number and size of the reports entered as input. In order to provide a clear view of the program in action, a walk-through of an example will be conducted which will include all major phases of the program. The walk-through will also include system queries, user responses, views of internal structures, and the resultant Functional Dependencies (FD) produced. More complex examples will be presented during the explanation of testing. These examples will not be used to show the manipulation of the internal data structures.

The sample walk-through will use as input the following data which is depicted in a format that would be viewed by a user at a CRT.

```
Report Name
0000 Automobile
0010 Model Make       Y
0020 Eng-no Style Exterior Interior Body-no  Y
0030 Body-no Year Weight Cost Price      Y
0040 END

Report Name
0000 Engine
0010 Eng-no No-cyl Displac Eng-wt Eng-cost HF  Y
0020 END
```

The displayed reports are entered by the user one line at a time. These reports are used by the User Interface Program to generate the Input File. The Y's displayed at the extreme right are system generated based on user
responses to system queries. They are used by the system to generate the status for the particular group of DNI's. A view of the Input File as generated from the above data is pictured in figure 4-1.

The FDGEN is executed by NCR 8250 JCL. The following is a complete walk-through of the FDGEN Program using the input data displayed in the Input File and picturing all system queries and user responses.

The first system query is in the form of an introductory instruction statement.

ANY INFORMATION THAT APPEARS ON THE SCREEN WITHOUT ANY INSTRUCTIONS MAY BE REMOVED BY DEPRESSING NEW LINE ** PRESS NEW LINE.

The user must respond by pressing 'NEW LINE', thus allowing execution to continue.

The Input File provides the second input to the FDGEN. The file is automatically accessed and read a record at a time until the entire file has been input. As each record is read the first four characters are discarded, leaving the group's status, level number, and associated DNI's. The program also discards all END's which are preceded by two blanks. The status is used to initiate the update of the Concatenation Code Table. The level becomes an entry in the
## Input File

### Figure 4-1

<table>
<thead>
<tr>
<th>LN</th>
<th>SL</th>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>O</td>
<td>Automobile</td>
</tr>
<tr>
<td>0010</td>
<td>Y1</td>
<td>Model Make</td>
</tr>
<tr>
<td>0020</td>
<td>Y2</td>
<td>Eng-No Style Exterior Interior Body-No</td>
</tr>
<tr>
<td>0030</td>
<td>Y3</td>
<td>Body-No Year Weight Cost Price</td>
</tr>
<tr>
<td>0040</td>
<td></td>
<td>END</td>
</tr>
<tr>
<td>0050</td>
<td></td>
<td>Engine</td>
</tr>
<tr>
<td>0060</td>
<td>Y1</td>
<td>Eng-No No-Cyl Displac Eng-wt Eng-ccst HP</td>
</tr>
<tr>
<td>0080</td>
<td></td>
<td>END</td>
</tr>
</tbody>
</table>

**IN:** Line number  
**S:** Status  
**I:** Level Number  
**TEXT:** Data-Name-Item
Data Name Table (DNT) and the current index to the Concatenation Code Table, from which is extracted the concatenation link entry to the DNT. A status of 'N' or 'S' is required to initiate the update of the Concatenation Code Table. The Concatenation Code Table is reinitialized at each determination of a new report. The initialized table is pictured at figure 4-2.

The DNI is constructed independently of user interaction. The end of the table is marked by inserting an 'END' in the last name field and 'XXX' in the last concatenation link field. The logical internal view of the table is pictured in figure 4-3.

If a report name or DNI has been entered that exceeds the maximum allowable number of characters the program will alert the user of the error by displaying the following error message.

```
DATA NAME 'DATA-NAME-ITEM' EXCEEDS THE ALLOWABLE NUMBER OF CHARACTERS, EXCESS CHARACTERS WILL BE TRUNCATED.
```

The user's response is to press 'NEW LINE'. The program truncates the excess characters by assignment and continues.

If a report name was entered and was not followed by
Concatenation Code Table

Figure 4-2

<table>
<thead>
<tr>
<th>Concat-code</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>41</td>
</tr>
<tr>
<td>51</td>
</tr>
<tr>
<td>61</td>
</tr>
<tr>
<td>71</td>
</tr>
<tr>
<td>81</td>
</tr>
<tr>
<td>91</td>
</tr>
</tbody>
</table>

Indexed by level number
## Logical Internal View of the Data Name Table

**Figure 4-3**

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Level-No</th>
<th>Conc-Link</th>
<th>Determin-Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Automobile</td>
<td>0</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Model</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Make</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Eng-no</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>Style</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>Exterior</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>Interior</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>Body-no</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>Body-no</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>Year</td>
<td>3</td>
<td>31</td>
<td>C</td>
</tr>
<tr>
<td>11</td>
<td>Weight</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>Cost</td>
<td>3</td>
<td>31</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>Price</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>Engine</td>
<td>0</td>
<td>NUL</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>Eng-no</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>No-Cyl</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>Displac</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>Eng-wt</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>Eng-cost</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>HP</td>
<td>1</td>
<td>11</td>
<td>C</td>
</tr>
<tr>
<td>21</td>
<td>END</td>
<td></td>
<td>XXX</td>
<td></td>
</tr>
</tbody>
</table>
at least one DNI an error condition will exist. The program
detects the error and queries the user as follows.

'REPORT NAME'

ERROR -- THIS REPORT HAS NO ENTRIES. DO YOU WISH
IT ELIMINATED - YES/NO.

The program is not designed to accept null reports. If
the user responds by entering

NO

the program will generate erroneous information as output;
whereas, if the user's response is

YES

the null report is eliminated from the DNT by overlaying.

The DNT is the primary source of information for the
remainder of the program. The DNT is first used during the
interactive identification of Virtual-Data-Items (VDI). The
following user instructions are provided.
USER INSTRUCTIONS

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

VIRTUAL DATA ITEM IDENTIFICATION PHASE

THE USER MUST IDENTIFY ALL VIRTUAL DATA ITEMS WHICH HAVE BEEN ENTERED AS DATA-NAME-ITEMS. THIS WILL BE DONE BY ENTERING A YES FOR EACH IDENTIFIED VIRTUAL DATA ITEM. THE SYSTEM WILL DISPLAY ONE POTENTIAL VIRTUAL DATA ITEM AT A TIME. YOU MUST ENTER A YES(Y) IF IT IS A VDI AND A NO (N) IF NOT.

DO YOU DESIRE AN EXPLANATION OF A VIRTUAL DATA ITEM? ENTER YES OR NO.

If the user's response is 'NO' the program will continue, on the other hand, a 'YES' response will produce the following description and example of a VDI.

A VIRTUAL DATA ITEM IS A DATA-NAME-ITEM WHICH IS DERIVABLE BY COMPUTATION INVOLVING DATA-NAME-ITEMS FROM WITHIN THE SAME DATA BASE. THIS SAYS THAT THE DATA-NAME-ITEMS CAN BE DERIVED FROM OTHER DATA-NAME-ITEMS THAT ARE WITHIN THE SAME DATA BASE OR POSSIBLY THE SAME REPORT.

AN EXAMPLE OF A VIRTUAL DATA ITEM WOULD BE TOTAL-HOURS-WORKED WHEN WITHIN THE SAME REPORT THE FOLLOWING DATA-NAME-ITEMS ARE AVAILABLE:
HOURS-WORKED-PER-WEEK
WEEKS-WORKED-PER-YEAR

IF YOU REQUIRE ADDITIONAL ASSISTANCE ENTER HELP
If the user enters 'HELP' the program will respond by providing the phone number and name of the development coordinator for the system, otherwise the user pressed 'NEW LINE' and the system will respond by displaying this message.

WE ARE READY TO START

The user simply presses 'NEW LINE' when he is ready to begin the VDI identification phase.

The program begins by displaying the report name associated with the DNI's to be presented to the user for possible selection. The user must respond to each newly presented DNI. The program cycles until all DNI's in the DNT have been presented. The following sequence is repeated for each DNI presented to the user.

REFCRT TO BE CHECKED FOR VIRTUAL DATA ITEMS IS 'REFCRT NAME'

'DATA-NAME-ITEM'

IS THIS DATA-NAME-ITEM A VDI? (YES or NO)

A user response of 'NO' allows the process to continue,
but a response of 'YES' interrupts the cycling process long enough to tag the selected VDI. The tagging is accomplished by inserting the code '3' into the associated determ-flag field for the selected DNI. In our example the DNI 'PRICE' will be selected as a VDI. The updated section of the DNI is pictured in figure 4-4.

After setting the tag, the program continues the cycling process. Upon completion, the user is presented with the following query.

ALL REPORTS AND THEIR RESPECTIVE DATA-NAME-ITEMS HAVE BEEN DISPLAYED. DO YOU WISH TO VERIFY THE LIST OF VIRTUAL DATA ITEMS. ENTER 'YES' OF 'NO'.

A user response of 'YES' will produce the following results.

PRICE

IF THIS IS A VALID VIRTUAL DATA ITEM ENTER YES OTHERWISE NO.

In the example only one VDI was selected. Had there been more, the above display would be repeated until all selected VDI's had been displayed and the user provided the opportunity to verify their status.
Updated Section of the Data Name Table

Figure 4-4

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Level-No</th>
<th>Conc-Link</th>
<th>Determ-Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Cost</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>Price</td>
<td>3</td>
<td>31</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>Engine</td>
<td>0</td>
<td>NUL</td>
<td>0</td>
</tr>
</tbody>
</table>
If the user responds with a 'YES' the program continues; however, if the response is 'NO' the status of the VDI is changed to that of a non-VDI. This is accomplished by changing the tag, code '3', back to a code of '0'. This is the only place in the program where an earlier decision may be reversed. 'PRICE', will remain a VDI in this example.

When all selected VDI's have been displayed and the user has responded to all queries, the system will present its next query.

DO YOU WISH TO REVIEW THE LIST OF DATA-NAME-ITEMS OF EACH REPORT AGAIN TO INSURE THAT ALL VIRTUAL DATA ITEMS HAVE BEEN IDENTIFIED. ENTER YES OR NO.

A 'NO' response to this query will terminate the virtual identification phase. A 'YES' response will cause all pointers to be reset to the beginning and the user will be given the opportunity to repeat the initial identification process again. This recycling of events includes another opportunity to verify the selected list as well as another review of all DNI's. The only addition to additional cycles will be the system's identification of all previously selected VDI's. This is depicted through a display of the previously selected VDI 'PRICE' when it is presented to the user as follows.
REJECT TO BE CHECKED FOR VIRTUAL DATA ITEMS IS 'REPORT NAME'

'PRICE' IDENTIFIED AS A VDI

IS THIS DATA-NAME-ITEM A VDI? (YES OR NC)

A 'NC' response will not change an identified VDI to a non-VDI; whereas, a 'YES' response will change a non-VDI to a VDI. An identified VDI can only be changed to a non-VDI during the verification of the VDI's.

The intersection phase of the program is not interactive. The purpose of this phase is to identify those DNI's which appear in more than one report. The identified DNI's are tagged by entering a code '2' in the determ-flag field associated with the DNI. If the DNI has been selected as a VDI, it is not considered during intersection. A view of the updated DNI following intersection is pictured in figure 4-5.

The determinant identification phase depends heavily upon user interactions. Upon entering this phase the program displays the following user requirements.
Updated Logical View of the Data Name Table

**Figure 4-5**

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
<th>Level-No</th>
<th>Conc-Link</th>
<th>Determ-Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Automobile</td>
<td>0</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Model</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Make</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Eng-no</td>
<td>2</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Style</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>Exterior</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>Interior</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>Body-no</td>
<td>2</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>Body-no</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>Year</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>Weight</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>Cost</td>
<td>3</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>Price</td>
<td>3</td>
<td>31</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>Engine</td>
<td>0</td>
<td>NUL</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>Eng-no</td>
<td>1</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>No-Cyl</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>Displac</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>Eng-wt</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>Eng-cost</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>HP</td>
<td>1</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>END</td>
<td>XXX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
USER REQUIREMENTS
***************

DETERMINANT IDENTIFICATION PHASE

YOU MUST IDENTIFY FROM EACH GROUP OF DATA-NAME-ITEMS DISPLAYED - THOSE ITEMS WHICH WILL UNIQUELY IDENTIFY THE REMAINING ITEMS WITHIN THE GROUP. THOSE DATA-NAME-ITEMS WHICH IDENTIFY THE REMAINING ITEMS ARE CALLED DETERMINANTS. EACH DETERMINANT IDENTIFIED MUST BE ENTERED UPON REQUEST. SPELLING IS CRITICAL. ANY QUESTIONS? YES/NO

A user response of 'YES' will produce the following example and explanation.

THE FOLLOWING DATA-NAME-ITEMS ARE DISPLAYED****
WAREHOUSE-NO, SPACE-AVAILABLE, ITEMS-STORED,
STORAGE-LEVEL**** THE SINGLE DATA ITEM WHICH COULD IDENTIFY THE OTHER ITEMS WOULD BE WAREHOUSE-NO ****IT IS THE DETERMINANT****

a 'NO' response, or following the display, the following statement is presented to the user.

ENTER GO WHEN YOU ARE READY TO START.

When the user is ready to begin the identification phase he must press 'NEW LINE'. This action allows the
program to continue. It produces the following message.

GRE E TO BE DISPLAYED ARE FROM REPORT 'REPORT NAME'

Displayed immediately following the report name is a variable explanation of what is to come. The display is variable because the variables WORD, ARTICLE, ONE-ARE-MORE, and VERB are assigned the correct words to make the explanation grammatically correct. The explanation is displayed only during the listing of groups from the first report.

'WORD' OF THE LISTED DATA-NAME-ITEMS MATCHED OTHER DATA-NAME-ITEMS IN OTHER REPORTS. THIS MEANS THAT YOU HAVE 'ARTICLE' CLUE AS TO WHICH DATA-NAME-ITEMS MIGHT BE 'ONE-ARE-MORE'

The values the variables may receive are:

WORD: None, One, Several;
ARTICLE: no, a, several;
ONE-ARE-MORE: a Determinant, Determinant;
VERB: is, are.

The following is a continuation of the above. The first text is used if no DNI's were intersected in the current report, otherwise the second text is used.
First Text

PLEASE ENTER THE DATA-NAME-ITEM WHICH MEETS THE REQUIREMENTS OF A DETERMINANT FOR THE GROUP. THERE CAN BE MORE THAN ONE DETERMINANT PER GROUP. FAILURE TO IDENTIFY AT LEAST ONE DETERMINANT WILL CAUSE AN ERROR CONDITION. THE SYSTEM WILL REQUEST THAT YOU ENTER EACH SELECTED DETERMINANT NAME ONE AT A TIME WHEN INDICATED. IF ALL DETERMINANT NAMES HAVE BEEN ENTERED, ENTER XX WHEN THE NEXT REQUEST APPEARS. EACH NAME ENTERED MUST BE SPELLED EXACTLY AS IT APPEARS ON THE SCREEN.

Second Text

THE DATA-NAME-ITEMS WHICH ALSO APPEAR IN OTHER REFERRS 'VERE' MARKED BY *. WHEN CONSIDERING WHICH DATA-NAME-ITEMS IDENTIFY THE REMAINING ITEMS IN THE GROUP--FIRST CONSIDER THE MARKED ITEMS. THERE CAN BE MORE THAN ONE DETERMINANT PER GROUP. FAILURE TO IDENTIFY AT LEAST ONE DETERMINANT WILL CAUSE AN ERROR CONDITION. THE SYSTEM WILL REQUEST THAT YOU ENTER EACH SELECTED DETERMINANT NAME ONE AT A TIME WHEN INDICATED. IF ALL DETERMINANT NAMES HAVE BEEN ENTERED, ENTER XX WHEN THE NEXT REQUEST APPEARS. EACH NAME ENTERED MUST BE SPELLED EXACTLY AS IT APPEARS ON THE SCREEN.

Within the current report the program searches and displays to the left of the above instructions the list of DNI's possessing the same concatenation link code. The program will generate matching codes in sequence, thus allowing for a sequential presentation of groups within the current report. When a generated code can not be matched, new codes are generated until a match is found, or until a logical number of levels beyond the last matched level have been searched an found not to exist. At this point the program will have completed a report cycle and will select the next
report as the current report to be considered.

The following view, less instructions, is seen by the user.

GROUP TO BE DISPLAYED ARE FROM REPORT AUTOMOBILE

MODEL
MAKE

INSTRUCTIONS
NOT SHOWN

Immediately following the above display, the system requests that the user select and enter a Determinant. The user enters 'MODEL'.

ENTER THE DETERMINANT NAME. MODEL

If the user spells the entry incorrectly, the program detects the error and presents the following message.

YOU SPELLED IT WRONG**TRY AGAIN**PLEASE.
PRESS NEW LINE
The press 'NEW LINE' response causes the incorrect response to be disregarded and a new request for a Determinant. If the user inadvertently enters a Determinant name twice the program detects the error and presents the following message.

DUPLICATE DETERMINANT ENTRY -- ENTER A DETERMINANT NAME ONLY ONCE PER UNIQUE GROUP. PRESS NEW LINE

The press 'NEW LINE' response causes the incorrect response to be disregarded and the system to ask for the next Determinant name.

A user wishing to delete a group from a report may do so by not identifying a determining Determinant for the group. This editing process is accomplished by entering 'XX' when asked to enter the Determinant name. The 'XX' response causes the program to recycle to the beginning of the identification phase, thus by-passing the Load FDTABLE Section. The system detects the actions and displays this message.

A DETERMINANT HAS NOT BEEN IDENTIFIED FOR THE LAST GROUP--GROUP IS DISCARDED
At the same time as the DNI's were being displayed on the screen they were being pushed onto the Storage Stack. The selected Determinant, 'MCPEL', was pushed onto the Storage Stack as well as the Concatenation Stack. Their internal logical view is pictured in figure 4-6.

Duplicate DNI's are deleted from the Storage Stack and the first FD is loaded into the F-D-Table. A picture of its logical view is at figure 4-7.

The program recycles to the beginning of the determinant identification phase, clears the Storage Stack, generates the next sequential search code '21', and begins its sequential search within the current report area of the DNT. The next view presented to the user is the current report name, a report of the instruction since report one is still current, and a listing of the next group in the sequence.
Logical View of the Concatenation and Storage Stacks

Figure 4-6

<table>
<thead>
<tr>
<th>D-Name</th>
<th>Lev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>11</td>
</tr>
</tbody>
</table>

Concatenation Stack

<table>
<thead>
<tr>
<th>Attr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
</tr>
<tr>
<td>Make</td>
</tr>
<tr>
<td>Model</td>
</tr>
</tbody>
</table>

Storage Stack

attr: Blanks

before duplicates are deleted

Logical View of the F-D-Table

Figure 4-7

<table>
<thead>
<tr>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>Model</td>
</tr>
<tr>
<td>01</td>
</tr>
</tbody>
</table>

F-D-Table
Each 'ENTER THE DETERMINANT NAME.' required a user response. The tables built and updated thus far during this cycle are pictured in figure 4-8.

Prior to updating the F-D-Table the program checks for two special cases which if detected will allow the user the opportunity to restructure the current FD.

The first special case exists when the Determinant Set does not identify any non-determinants. The example being used does not activate this facility; however, the user's options are the same as those provided in the second special case. An explanation and example of the first special case was presented on pages 34, 35, and 36. The queries for this case are shown on page 65.
Current View of the Concatenation and Storage Stacks

Figure 4-8

<table>
<thead>
<tr>
<th>D-Name</th>
<th>Lev</th>
<th>Attr</th>
<th>Attr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal</td>
<td>21</td>
<td>Internal</td>
<td>Body-nc</td>
</tr>
<tr>
<td>Exterior</td>
<td>21</td>
<td>Exterior</td>
<td>bbbbbb</td>
</tr>
<tr>
<td>Style</td>
<td>21</td>
<td>Style</td>
<td>bbbbb</td>
</tr>
<tr>
<td>Eng-no</td>
<td>21</td>
<td>Eng-no</td>
<td>bbbbb</td>
</tr>
<tr>
<td>Model</td>
<td>11</td>
<td>Body-no</td>
<td>bbbbb</td>
</tr>
</tbody>
</table>

Concatenation Stack

Storage Stack After duplicates are deleted

Storage Stack before duplicates are deleted
The second special case is detected when there exists a Determinant Set which consist of Determinants from more than one group. This case is detected as can be seen by referring to the Concatenation Stack pictured in figure 4-8. If the second special case is detected the program immediately checks for the third special case. This case does not exist in the example but was explained on page 37.

The program responds to a first special case situation by querying the user as follows.

SPECIAL SITUATION

THE FOLLOWING DATA-NAME-ITEMS HAVE BEEN CONCATENATED TO FORM A DETERMINANT. THE DETERMINANT AS STRUCTURED IS SELF-IDENTIFYING.

LIST OF DETERMINANTS NOT SHOWN

IF THE DATA-NAME-ITEMS MARKED * UNIQUELY IDENTIFIES ANY OF THE OTHER LISTED ITEMS **ENTER YES** IF NO -- THE RELATION IS ASSUMED CORRECT--PRESS NEW LINE.

A 'YES' response continues the above query.

ARE ALL UNMARKED DETERMINANTS IDENTIFIED BY THE MARKED DETERMINANTS? ENTER YES ** OTHERWISE PRESS NEW LINE.
A 'YES' response will transfer all unmarked Determinants from the Concatenation Stack to the Storage Stack, update all counters, and load the generated FD into the F-D-Table. A response of 'NEW LINE' produces the following continuation to the query.

LIST THOSE DATA-NAME-ITEMS WHICH ARE IDENTIFIED. ENTER XX TO SIGNAL NO MORE ENTRIES ARE TO BE MADE.

Following these instructions the user enters the selected Determinants and the program transfers each entry from the Concatenation Stack to the Storage Stack. Upon entry of 'XX', the generated FD is loaded into the F-D-Table. Any entry that is spelled incorrectly will cause an error condition to exist. The program responds with the following error message.

YOU SPelled IT WRCNG**TRY AGAIN**PLEASE PRESS NEW LINE

The second special case, when detected, causes a user query similar to the first special case. The walk-through will provide several examples of this case.
ARE ANY OF THE FOLLOWING DETERMINANTS UNIQUELY IDENTIFIED BY THE DETERMINANTS MARKED WITH THE *. ENTER 'YES'**OTHERWISE PRESS NEW LINE.

* INTERIOR
* EXTERIOR
* STYLE
* ENG-NO
* MODEL

The response selected is 'NEW LINE'. This response generates no further queries of the user. The program saves the set of concatenated Determinants in the Save Stack and loads the F-D-Table. An internal view of the Save Stack and an updated view of the F-D-Table is pictured in figure 4-9.

The program updates its search code to '31' and recycles to the beginning of the current report in the DNT. The Storage Stack is cleared and the user is queried again.

GROUP TO BE DISPLAYED ARE FROM REPORT AUTOMOBILE

<table>
<thead>
<tr>
<th>BODY-NO</th>
<th>INSTRUCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>NCT</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>SHOWN</td>
</tr>
</tbody>
</table>

ENTER THE DETERMINANT NAME.  BODY-NO
ENTER THE DETERMINANT NAME.  XX
Current View of the Save Stack and F-D-Table

Figure 4-9

<table>
<thead>
<tr>
<th>S-Name</th>
<th>Slev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interior</td>
<td>21</td>
</tr>
<tr>
<td>Exterior</td>
<td>21</td>
</tr>
<tr>
<td>Style</td>
<td>21</td>
</tr>
<tr>
<td>Eng-no</td>
<td>21</td>
</tr>
<tr>
<td>Model</td>
<td>11</td>
</tr>
</tbody>
</table>

List

- Body-no
- Interior
- Exterior
- Style
- Eng-no
- Model

Save Stack

F-D-Table

- 05
- Make
- 01
- Model
- 01
The user selected 'BCDY-NC' to be the Determinant for this group while keeping in mind that all higher level Determinants for the current report are being concatenated to form the Determinant Set. The internal data structures as created and updated are pictured in figure 4-10.

The second special case is once again detected. The program immediately queries the user as before.

ARE ANY OF THE FOLLOWING DETERMINANTS UNICELY IDENTIFIED BY THE DETERMINANTS MARKED WITH THE *.
Enter 'YES'**OTHERWISE NEW LINE.

* ECDY-NO
  INTERIOR
  EXTERIOR
  STYLE
  ENG-NO
  MODEL

The selected response is 'YES'. This causes the query to be continued.

ARE ALL UNMARKED DETERMINANTS IDENTIFIED BY THE MARKED DETERMINANTS? ENTER 'YES'** OTHERWISE PRESS NEW LINE.
Current View of the Concatenation and Storage Stack

Figure 4-10

<table>
<thead>
<tr>
<th>D-Name</th>
<th>Lev</th>
<th>Attr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body-no</td>
<td>31</td>
<td>Body-no</td>
</tr>
<tr>
<td>Interior</td>
<td>21</td>
<td>Cost</td>
</tr>
<tr>
<td>Exterior</td>
<td>21</td>
<td>Weight</td>
</tr>
<tr>
<td>Style</td>
<td>21</td>
<td>Year</td>
</tr>
<tr>
<td>Eng-no</td>
<td>21</td>
<td>Body-no</td>
</tr>
<tr>
<td>Model</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Storage Stack

Concatenation Stack before duplicates are deleted

Storage Stack after duplicates are deleted
The selected response is 'YES'. This response causes all unmarked Determinants to be transferred from the Concatenation Stack to the Storage Stack. The Concatenation Stack contents are compacted to the low end and its contents saved in the Save Stack. The generated PD is loaded into the P-D-Table. The logical views of the internal tables (less DNT) are pictured in figure 4-11.

The program recycles to the beginning of the determinant identification section; is unable to match a search code; clears the Save, Concatenate, and Storage Stacks; and enters the next sequential report within the DNT. The new search code selected is '11', and the user is queried as follows.

GROUPS TO BE DISPLAYED ARE FROM REPORT ENGINE

<table>
<thead>
<tr>
<th>ENG-NC</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO-CYL</td>
<td></td>
</tr>
<tr>
<td>DISFLAC</td>
<td></td>
</tr>
<tr>
<td>ENG-WT</td>
<td></td>
</tr>
<tr>
<td>ENG-CCST</td>
<td></td>
</tr>
<tr>
<td>HP</td>
<td></td>
</tr>
</tbody>
</table>

ENTER THE DETERMINANT NAME.  ENG-NO

ENTER THE DETERMINANT NAME.  XX
Current Logical View of Internal Tables

Figure 4-11

<table>
<thead>
<tr>
<th>D-Name</th>
<th>Lev</th>
<th>Attr</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body-no</td>
<td>31</td>
<td>Interior, Exterior, Style, Eng-no, Model, Cost, Weight, Year</td>
<td>08 Body-no 01 Interior 01 Exterior 01 Style 05 Eng-no 01 Model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Concatenation Stack</td>
<td>Storage Stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Save Stack</td>
<td>F-D-Table</td>
</tr>
</tbody>
</table>
No special cases are detected. The program generated and updated tables are pictured in figure 4-12. The program is unable to match additional search codes within the last report of the DNT. This condition terminates the determinant identification phase.

The program's last section is the print F-D-Table. Within this section each FD within the F-D-Table is extracted one at a time, reformatted as pictured in figure 4-13 and written to the line printer. This output provides the user with a readable hardcopy of the generated FD. The hardcopy may be used by the user to implement changes to the Input File and regenerate the FD's. The hardcopy out of the user generated FD's for the example are pictured in figure 4-14.
Final View of Program Generated and Updated Tables

Figure 4-12

<table>
<thead>
<tr>
<th>L-Name</th>
<th>Lev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eng-no</td>
<td>11</td>
</tr>
</tbody>
</table>

Concatenation Stack

<table>
<thead>
<tr>
<th>S-Name</th>
<th>Slev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>00</td>
</tr>
</tbody>
</table>

Save Stack

<table>
<thead>
<tr>
<th>Attr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eng-no</td>
</tr>
<tr>
<td>HP</td>
</tr>
<tr>
<td>Eng-cost</td>
</tr>
<tr>
<td>Eng-wt</td>
</tr>
<tr>
<td>Displac</td>
</tr>
<tr>
<td>No-Cyl</td>
</tr>
<tr>
<td>Eng-no</td>
</tr>
</tbody>
</table>

Storage Stack before duplicates are deleted

<table>
<thead>
<tr>
<th>Attr</th>
</tr>
</thead>
<tbody>
<tr>
<td>HP</td>
</tr>
<tr>
<td>Eng-cost</td>
</tr>
<tr>
<td>Eng-wt</td>
</tr>
<tr>
<td>Displac</td>
</tr>
<tr>
<td>No-Cyl</td>
</tr>
<tr>
<td>bbbbbb</td>
</tr>
</tbody>
</table>

List

| XXXXXXXXXX |
| HP         |
| Eng-cost   |
| Eng-wt     |
| Displac    |
| No-Cyl     |
| 05          |
| Eng-no     |
| 01          |
| Interior   |
| Exterior   |
| Style      |
| Eng-no     |
| Model      |
| Cost       |
| Weight     |
| Year       |
| 08          |
| Body-no    |
| 01          |
| Body-no    |
| 01          |
| Interior   |
| Exterior   |
| Style      |
| Eng-no     |
| Model      |
| 05          |
| Make       |
| 01          |
| Model      |
| 01          |

Storage Stack after duplicates are deleted

F-D-Table
Functional Dependency Output Format

Figure 4-13

DETERMINANT SET ---> NON-DETERMINANT SET

Walk-Through Generated Functional Dependencies

Figure 4-14

Model ---> Make
Model Eng-no Style Exterior Interior ---> Body-nc
Body-nc ---> Year Weight Cost Model Eng-no Style
Exterior Interior
Eng-nc ---> No-Cyl Displac Eng-wt Eng-cost HP
4.2 TESTING

The FDGEN Program was designed in a top-down configuration. The top-down design enhanced modular implementation which simplified the syntactical debugging process and enhanced modular testing. Testing was conducted in three phases; syntactical debug testing, program testing, and system testing.

The syntactical debug phase was conducted immediately following insertion of each module into the system. The only tccl used during this phase was the COECL compiler which provided a limited set of diagnostic outputs. During the program testing phase two input files were used. They are pictured in figure 4-15 and figure 4-16.

Test file 1 and 2 were used to debug the logical construction of the program prior to insertion of the special case modules. The COBOL compiler was once again the major debugging tool used during this phase. The compiler diagnostics were augmented by an abundance of output statements. The statements were positioned so as to provide the tester with a complete internal view of all data at and around the point of concern. This procedure was continued until all paths had been tested for which the test data had been designed and all errors had been corrected.

The system test phase incorporated test files 1 and 2 in addition to
<table>
<thead>
<tr>
<th>LN SL</th>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0Inventory</td>
</tr>
<tr>
<td>0010Y1</td>
<td>Product+ Product-Description</td>
</tr>
<tr>
<td>0020C1</td>
<td>Quantity Manufacturer Shipping-Weight</td>
</tr>
<tr>
<td>0030C3</td>
<td>END</td>
</tr>
<tr>
<td>0040</td>
<td>0Customer</td>
</tr>
<tr>
<td>0050Y1</td>
<td>Customer+ Customer-Name Address Phone</td>
</tr>
<tr>
<td>0060</td>
<td>END</td>
</tr>
<tr>
<td>0070C0</td>
<td>0Invoice</td>
</tr>
<tr>
<td>0080Y1</td>
<td>Invoice+ Customer+ Data-of-Invoice</td>
</tr>
<tr>
<td>0090CC1</td>
<td>Gross-Amount</td>
</tr>
<tr>
<td>0100Y2</td>
<td>Product+ Product-Description</td>
</tr>
<tr>
<td>0110CC2</td>
<td>Cost</td>
</tr>
<tr>
<td>0120</td>
<td>END</td>
</tr>
<tr>
<td>0130</td>
<td>0Accounts- Receivable</td>
</tr>
<tr>
<td>0140Y1</td>
<td>Customer+</td>
</tr>
<tr>
<td>0150Y2</td>
<td>Invoice+ Amount-Receivable</td>
</tr>
<tr>
<td>0160N1</td>
<td>Customer-Total</td>
</tr>
<tr>
<td>0170</td>
<td>END</td>
</tr>
<tr>
<td>0180</td>
<td>0Backlog</td>
</tr>
<tr>
<td>0190Y1</td>
<td>Product+</td>
</tr>
<tr>
<td>0200Y2</td>
<td>Customer+</td>
</tr>
<tr>
<td>0210Y3</td>
<td>Invoice+ Amount-Backordered</td>
</tr>
<tr>
<td>0220N2E1</td>
<td>for-Customer</td>
</tr>
<tr>
<td>0230N1</td>
<td>Total-for-Product</td>
</tr>
<tr>
<td>0240</td>
<td>END</td>
</tr>
<tr>
<td>0250</td>
<td>END</td>
</tr>
</tbody>
</table>

+: Number
<table>
<thead>
<tr>
<th>LN SL</th>
<th>TEXT</th>
<th>LN SL</th>
<th>TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0Areport</td>
<td>0270</td>
<td>X1A X1 B</td>
</tr>
<tr>
<td>0010</td>
<td>Y1A</td>
<td>0280</td>
<td>END</td>
</tr>
<tr>
<td>0020</td>
<td>Y2B1 B2</td>
<td>0290</td>
<td>0Hreport</td>
</tr>
<tr>
<td>0030</td>
<td>S2C C2</td>
<td>0300</td>
<td>Y1B</td>
</tr>
<tr>
<td>0040</td>
<td>N1D1</td>
<td>0310</td>
<td>Y2C X2</td>
</tr>
<tr>
<td>0050</td>
<td>Y2E1 E2</td>
<td>0320</td>
<td>END</td>
</tr>
<tr>
<td>0060</td>
<td>N1F</td>
<td>0330</td>
<td>0Ireport</td>
</tr>
<tr>
<td>0070</td>
<td>Y2G1</td>
<td>0340</td>
<td>Y1K1</td>
</tr>
<tr>
<td>0080</td>
<td>END</td>
<td>0350</td>
<td>C1K2</td>
</tr>
<tr>
<td>0090</td>
<td>0Breport</td>
<td>0360</td>
<td>C1K3</td>
</tr>
<tr>
<td>0100</td>
<td>Y1H I</td>
<td>0370</td>
<td>C1K4</td>
</tr>
<tr>
<td>0110</td>
<td>END</td>
<td>0380</td>
<td>C1K5</td>
</tr>
<tr>
<td>0120</td>
<td>0Creport</td>
<td>0390</td>
<td>C1K6</td>
</tr>
<tr>
<td>0130</td>
<td>Y1I J</td>
<td>0400</td>
<td>Y2L1</td>
</tr>
<tr>
<td>0140</td>
<td>END</td>
<td>0410</td>
<td>C2K4</td>
</tr>
<tr>
<td>0150</td>
<td>0Dreport</td>
<td>0420</td>
<td>C2K5</td>
</tr>
<tr>
<td>0160</td>
<td>Y1H J</td>
<td>0430</td>
<td>END</td>
</tr>
<tr>
<td>0170</td>
<td>END</td>
<td>0440</td>
<td>0Error</td>
</tr>
<tr>
<td>0180</td>
<td>0Error</td>
<td>0450</td>
<td>Y1A</td>
</tr>
<tr>
<td>0190</td>
<td>Y1X1</td>
<td>0460</td>
<td>END</td>
</tr>
<tr>
<td>0200</td>
<td>X2X2 A D</td>
<td>0470</td>
<td>0Strange</td>
</tr>
<tr>
<td>0210</td>
<td>END</td>
<td>0480</td>
<td>Y1M</td>
</tr>
<tr>
<td>0220</td>
<td>0Preport</td>
<td>0490</td>
<td>Y2N</td>
</tr>
<tr>
<td>0230</td>
<td>Y1C A</td>
<td>0500</td>
<td>N1O</td>
</tr>
<tr>
<td>0240</td>
<td>Y2D X1 X2</td>
<td>0510</td>
<td>END</td>
</tr>
<tr>
<td>0250</td>
<td>END</td>
<td>0520</td>
<td>END</td>
</tr>
<tr>
<td>0260</td>
<td>0Greport</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
test file 3 which is pictured in figure 4-17. The system test phase was designed to test the flow and correct manipulation of data through the four subsystems: User Interface, Editor, FDGEN, and FDA. The primary test objectives of the FDGEN subsystem were that the data placed into the Input File was correctly read; that the input data was accurately presented to the interactive user; that the data as defined by the user was correctly manipulated by the subsystem; and that the F-D-Table produced (Output File) was properly generated as output, both as user output as well as input for the FDA.

The FDGEN test schema is presented at figure 4-18. It insures that each path through which data may traverse is tested for each condition which is allowed to occur along that path. The test path diagram is pictured at figure 4-19. Each test case consist of a group of test segments. Each segment consist of one or more routines which are to be tested using one or more conditions. Segments without stated conditions are tested by virtue of the fact that the data must pass through their routines. The test segments are as follows:

<table>
<thead>
<tr>
<th>SEGMENT</th>
<th>ROUTINE</th>
<th>CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Farse-Record</td>
<td>1. Reports with groups with status of C, N, Y, and S.</td>
</tr>
<tr>
<td></td>
<td>Up-Level</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Build-D-N-Table</td>
<td>2. ENI's with more than ten characters.</td>
</tr>
<tr>
<td></td>
<td>Fill-it</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Virtual-elim</td>
<td>1. Select, verify, and review VDI's.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Select, verify, restore, and review VDI's.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Select, review, and select</td>
</tr>
</tbody>
</table>
IN SL TEXT

0000 0Test1
0010X1SSAN-ID STR-ADDRESS CITY-ADDRESS ZIP-CDOE TELEPHONE
0020C1MAR-STATUS VEH-STATE LIC-NO MAKE YEAR DECAL-NC
0030C1PEC-FR-ORG SOC-FR-ORG REL-EMPLMT EMP-ON-CAM EMP-OF-CAM
0040C1TUIT-ASST SVC-BRANCH TOI-SAL
0050Y2THIS IS A JUNK TEST GROUP
0060 END
0070 0TEST2
0080X1SSAN-ID STU-NAME CITY-BIRTH ST-BIRTH DATE-BIRTH
0090C1HS-NAME HS-CITY HS-STATE DR-GRAD NOK-NAME NOK-ADD
0100C1NOK-CITY NOK-STATE NOK-ZIP NOK-TELE DT-ENBOLL
0110C1PBC1-GR-DT ST-COLLEGE LEVEL-ID PRI-MAJOR SEC-MAJOR
0120C1ADV-NAME DEG-SOUGHT NO-SIST NC-BIRTH SGC-FR-ORG
0130C1PBC1-FR-ORG TUIT-ASST MAR-STATUS
0140Y2STU-NAME F-NAME M-NAME
0150 END
0160 0TEST3
0170X1SPECIAL TEST TC SHOW
0180Y2THAT
0190Y3THE WCRKS
0200N1THAT WHAT HAPPENS IS
0210 END
0220 0TEST7
0230X1MCDEL MAKE
0240Y2ENG-NO STYLE EXTERIOR INTERIOR BODY-NC
0250Y3BCEY-NO YEAR WEIGHT COST PRICE
0260 END
0270 0TEST9
0280Y1ENG-NO NO-CYL DISEL ENG-WT ENG-COST HF
0290 END
0300 0TEST46
0310Y1STYLE EXTERIOR INTERIOR
0320Y2EXTERIOR NO-DOORS PAINT-CCDE TRIM-PK
0330S2INTERIOR CARGO-SF UPHOLSTERY SEATS ASSARY-PK
0340 END
0350 0TEST58
0360Y1PAINT-CODE COLOR-1 TCNE-NC
0370Y2TCNE-NO COLOR-2
0380 END
0390 END
# Test Schema

**Figure 4-18**

## TEST CASES

<table>
<thead>
<tr>
<th>Case No</th>
<th>Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A12 E1 C D E12 F H I J K U U2 V X Z</td>
</tr>
<tr>
<td>2</td>
<td>A1 B3 C D E1 F H I J K U U2 V X Y</td>
</tr>
<tr>
<td>3</td>
<td>A1 E1 C D E1 G H I J L U U1 V W</td>
</tr>
<tr>
<td>4</td>
<td>A1 B1 C D E1 F H M N U U2 V W</td>
</tr>
<tr>
<td>5</td>
<td>A1 E1 C D E1 F H M O U U2 V W</td>
</tr>
<tr>
<td>6</td>
<td>A1 B1 C D E1 F H M P Q U U2 V W</td>
</tr>
<tr>
<td>7</td>
<td>A1 E1 C D E1 F H M P R S U U2 V W</td>
</tr>
<tr>
<td>8</td>
<td>A1 B1 C D E1 F H M P R T U U2 V W</td>
</tr>
<tr>
<td>9</td>
<td>A1 E1 C D E1 F H U U2 V W</td>
</tr>
</tbody>
</table>
FDGEN Test Path Diagram
Figure 4-19
WDI's.

C Report-Intersect
Intersect
Loop2

D Determ-identify
CleadFD
Make-null
Cat-Clear

E Savclear
Fill-it
Check-Name
Check-DName
Set-Key

F Load-Cat-Stack
Push-Stack
Cat-union

G Load-Cat-Stack
Pcp-Stack
Push-Stack
Cat-union

H Build-F-D-Table
De-lete
Remove
Check-List

I Spec-Case
Disp-Det

J Next-Q

K E-New-Relation
L-tc-R
Pack-it-Down
Move-Det2

L Sel-items

1. Two or more reports with two
   or more groups.
2. Misspell Determinants.
3. Enter duplicate Determinants
   for the same group.
4. Non-selection of Determinant
   for one group.

1.Y Two groups with at least one
   Determinants each. The level
   of the second must be greater
   than the first.
2.N Two Determinants of the same
   level but separated by Determin-
   ants of different levels.
3.C Continued group.

1.S Two groups with levels not
   equal to one, both groups
   separate but of the same
   level.
2.N Two groups - one with a level
   greater than one, the second
   Determinant selected with a
   level less than the first.

Select a Determinant.

Determinants identify themselves.
The set remains unchanged.

Elect to change the set
configuration.

Elect to move all Determinants
which have been concatenated to
the last group's Determinant.

Elect to move one or more of
<table>
<thead>
<tr>
<th>Move-Det-Rt</th>
<th>Search-Stack Compact</th>
<th>the Determinants which were concatenated to the last group's Determinant. Vary the number moved.</th>
</tr>
</thead>
<tbody>
<tr>
<td>M Spec-Case2</td>
<td></td>
<td>Two or more Determinants of different groups concatenated together.</td>
</tr>
<tr>
<td>N Sav-More Transfer-Det Make-Room Savclear New-Union-Main New-union Compact</td>
<td></td>
<td>The current Determinant level must equal the successor level. Use status S and levels 1 and 2 intermixed.</td>
</tr>
<tr>
<td>O Sav-Last</td>
<td></td>
<td>The current Determinant level is not equal to the successor level.</td>
</tr>
<tr>
<td>F Disp-Det</td>
<td></td>
<td>The current Determinant level is not equal to the successor level.</td>
</tr>
<tr>
<td>Q Sav-Last</td>
<td></td>
<td>Elect not to change the set configuration.</td>
</tr>
<tr>
<td>R Sav-Last Next-Cues</td>
<td></td>
<td>Elect to change the set configuration.</td>
</tr>
<tr>
<td>S Sel-items Move-Det-Rt Search-Stack Compact</td>
<td></td>
<td>Elect to move one or more of the Determinants which was concatenated to the last group's Determinants. Vary the size of the set moved.</td>
</tr>
<tr>
<td>T B-New-Relation L-to-R Pack-it-D Move-Deter</td>
<td></td>
<td>Elect to move all Determinants not of the current group.</td>
</tr>
<tr>
<td>U Lcad1</td>
<td></td>
<td>Self identifying Determinant.</td>
</tr>
<tr>
<td>U1</td>
<td></td>
<td>Non-determinants exist.</td>
</tr>
<tr>
<td>U2 Lcad2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V Next-Proc Lcad-Table Last-Proc Print-F-D-Table Genout</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Build-L-S  
W  Build-R-S  Less than twelve Determinants and non-determinants.
X  Build-R-S  More than twelve Determinants and non-determinants.
     More-Lines
Y  Print-it  More than 24 Determinants and non-determinants.
       Build-R-S
       Build-R-S
Z  Build-R-S  Twelve or less Determinants and non-determinants.

The user's output of input test file 1 is pictured in figure 4-20. The file specifically tested the following segments and associated test conditions: A1(C,N,Y), A2, B1, B2, B3, C, D, E1, F1, F3, G2, H, I, L, M, F, R, S, U, U1, U2, V, and W.

The user's output of input test file 2, as manipulated to test segments A1(C,N,S,Y), C, D, E2, F1, F2, F3, G1, G2, H, I, J, M, N, O, P, Q, R, S, T, U, U1, U2, V, and W, is pictured in figure 4-21.

Input test file 3 is separated into test reports. Each report name specifies the specific test case to be tested. The file must be input three times inorder to test all conditions. The user's output is presented in figures 4-22, 4-23, and 4-24. The Output File generated from the F-D-Table is pictured in figure 4-25. This file is stored on disc an is later used as input for the FDA subsystem.
PRODUCT+  --->  PRODUCT-DE QUANTITY  MANUFACTUR SHIPPING-W
CUSTOMER+  --->  CUSTOMER-ADDRESS  PHONE
INVOICE+  --->  CUSTOMER+  DATE-OF-IN  GROSS-AMOUNT
PRODUCT+  --->  PRODUCT-DE COST  INVOICE+
CUSTOMER+  --->  :01
CUSTOMER+ INVOICE+  --->  AMOUNT-REC
PRODUCT+  --->  :02
PRODUCT+ CUSTOMER+  --->  :03
CUSTOMER+ INVOICE+  --->  AMOUNT-BAC PRODUCT+
BL-FOR-CUS  --->  :04
Test Run 1 -- Input Test File 3

Figure 4-22

SSAN-ID ---> STR-ADDRS CITY-ADDRS ZIP-CODE TELEPHONE
MAR-STATUS VEH-STATE LIC-NO MAKE YEAR DECAL-NO PRO-
FR-ORG SOC-FR-ORG REL-EMPLMT EMF-ON-CAM EMP-CF-CAM TUIT-ASST
SVC-BRANCH TOT-SAL
IS THIS ---> SSAN-ID TEST
SSAN-ID ---> STU-NAME CITY-BIRTH ST-BIRTH DATE-BIRTH
HS-NAME HS-CITY HS-STATE DR-GRAD NOK-NAME NCK-ADD NOK-CITY
NCK-STATE NOK-ZIP NOK-TEL DT-ENROLL PROJ-GH-DT ST-
COLLEGE LEVEL-ID PRI-MAJOR SEC-MAJOR ADV-NAME DEG-SOUGHT
NC-SIST NC-BROTH SOC-FR-ORG PRO-FR-ORG TUIT-ASST MAR-
STATUS
STU-NAME ---> F-NAME M-NAME SSAN-ID
SPECIAL TEST ---> 10 SHOW THAT WHAT HAPPENS IS
SPECIAL TEST THAT ---> 301
THAT THE ---> SPECIAL TEST WCRKS
MCDEL ---> MAKE
MCDEL BODY-NO ---> ENG-NC STYLE EXTERIOR INTERIOR
MCDEL BCLY-NO ---> YEAR WEIGHT COST PRICE
ENG-NO ---> NO-CYL DIESL ENG-WT ENG-COST HE
EXTERIOR STYLE ---> INTERIOR
EXTERIOR STYLE ---> NC-DACRS PAINT-CCDE TBIM-PK
STYLE EXTERIOR INTERIOR ---> CARGO-SP UPHOLSTERY
SEATS ASSARY-PK
PAINT-CCDE ---> CCLR-1 TONE-NO
TONE-NO ---> COLR-2 PAINT-CODE
Figure 4-23

SSAN-ID --> STR-ADDRS CITY-ADDRS ZIP-CODE TELEPHONE
MAR-STATUS VEH-STATE LTC-NO MAKE YEAR DECAL-NO PRO-
FR-ORG SCC-FR-ORG REL-EMPLMT EMP-ON-CAM EMP-CF-CAM TUIT-ASST
SVC-BRANCH TOT-SAL
SSAN-ID --> STU-NAME CITY-BIRTH ST-BIRTH DATE-BIRTH
HS-NAME HS-CITY HS-STATE GR-GRAD NCK-NAME NCK-ADD NCK-
CITY NOK-STATE NOK-ZIP NOK-ICELE DT-ENROLL PROJ-GR-DATE
ST-CCOLLEGE LEVEL-ID PRH-MAJOR SCC-MAJOR ADV-NAME DEG-SOUGHT
NC-SIST NC-BROTHER SCC-FR-ORG PRO-FR-ORG TUIT-ASST MAR-STATUS
STU-NAME --> F-NAME M-NAME SSAN-ID
SPECIAL TEST --> TO SHOW THAT WHAT HAPPENS IS
SPECIAL TEST THAT --> :01
THAT THE --> SPECIAL TEST WORKS
MCDEL --> MAKE
MCDEL INTERIOR EXTERIOR STYLE ENG-NO --> BODY-NO
MCDEL INTERIOR EXTERIOR STYLE BODY-NO --> YEAR
WEIGHT COSI PRICE ENG-NO
ENG-NO --> NO-CYL DISPL ENG-WT ENG-COST HP
STYLE --> EXTERIOR INTERIOR
STYLE EXTERIOR --> NO-DOORS PAINT-CODE TRIM-FLK
STYLE INTERIOR --> CARGO-SE UPHOLSTERY SEATS
ASSAY-FLK
PAINT-CODE --> CCLCR-1 TCNE-NO
TCNE-NO --> COLCR-2 PAINT-CODE
SSAN-ID ---> STR-ADDRS CITY-ADDRS ZIP-CODE TELEPHONE
MAR-STATUS VEH-STATE LIC-NC MAKE YEAR DECAL-NO PRO-
FR-CRG SCC-FR-ORG REL-EMPLMT EMP-ON-CAM EMP-OF-CAM TUIT-ASSIS
SVC-BRANCH TOT-SAL

THIS ---> IS A JUNK TEST GROUP SSAN-ID
STU-NAME SSAN-ID ---> CITY-BIRTH ST-BIRTH DATE-BIRTH
HS-NAME HS-CITY HS-STATE DR-GRAD NOK-NAME NOK-ADD
NOK-CITY NOK-STATE NOK-ZIP NOK-TELE DT-ENRCELL PROJ-GR-
DT ST-COLLEGE LEVEL-LE PRI-MAJOR SEC-MAJOR ADV-NAME DEG-
SUGHT NO-SIST NO-BROTH SCC-FR-ORG ERC-FR-CRG TUIT-ASSIST
SPECIAL TEST ---> TO SHOW THAT WHAT HAPPENS IS
SPECIAL TEST THAT ---> :01
THAT THE ---> WORKS TEST SPECIAL
MODEL ---> MAKE
MODEL STYLE ENG-NO ---> EXTERIOR INTERIOR BODY-NO
BODY-NO ---> YEAR WEIGHT COST PRICE MODEL STYLE
ENG-NO
ENG-NO ---> NO-CYL LISPL ENG-WT ENG-COST HP
INTERIOT EXTERIOR ---> STYLE
EXTERIOR ---> NO-LCORS PAINT-CODE TRIM-PK INTERIOR
EXTERIOR INTERIOR ---> CARG-SF UHCLSTERY SEATS
ASSARY-EK
PAINT-CODE ---> COLOR-1 TCNE-NC
TCNE-NO ---> COLCH-2 PAINT-CODE
Output File Generated From F-D-Table

Figure 4-25

<table>
<thead>
<tr>
<th>001</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSAN-ID</td>
</tr>
<tr>
<td>018</td>
</tr>
<tr>
<td>STR-ADDR</td>
</tr>
<tr>
<td>CITY-ADDR</td>
</tr>
<tr>
<td>ZIP-CODE</td>
</tr>
<tr>
<td>TELEPHONE</td>
</tr>
<tr>
<td>MAR-STATUS</td>
</tr>
<tr>
<td>VEH-STATE</td>
</tr>
<tr>
<td>LIC-NO</td>
</tr>
<tr>
<td>MAKE</td>
</tr>
<tr>
<td>YEAR</td>
</tr>
<tr>
<td>DECAI-NO</td>
</tr>
<tr>
<td>ERO-FR-ORG</td>
</tr>
<tr>
<td>SOC-FR-ORG</td>
</tr>
<tr>
<td>REL-EMPLMT</td>
</tr>
<tr>
<td>EMP-CN-CAM</td>
</tr>
<tr>
<td>EMP-NF-CAM</td>
</tr>
<tr>
<td>TUIT-ASSST</td>
</tr>
<tr>
<td>SVC-BRANCH</td>
</tr>
<tr>
<td>TOT-SAL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>002</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
</tr>
<tr>
<td>THIS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>002</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSAN-ID</td>
</tr>
<tr>
<td>TEST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>001</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSAN-ID</td>
</tr>
<tr>
<td>028</td>
</tr>
<tr>
<td>STU-NAME</td>
</tr>
<tr>
<td>CITY-BIRTH</td>
</tr>
<tr>
<td>ST-BIRTH</td>
</tr>
<tr>
<td>DATE-BIRTH</td>
</tr>
<tr>
<td>HS-NAME</td>
</tr>
<tr>
<td>HS-CITY</td>
</tr>
<tr>
<td>HS-STATE</td>
</tr>
<tr>
<td>LR-GRAD</td>
</tr>
<tr>
<td>NCK-NAME</td>
</tr>
<tr>
<td>NOK-ADD</td>
</tr>
<tr>
<td>NOK-CITY</td>
</tr>
<tr>
<td>NOK-STATE</td>
</tr>
<tr>
<td>NOK-ZIP</td>
</tr>
<tr>
<td>NOK-TEL</td>
</tr>
<tr>
<td>DT-ENRCLL</td>
</tr>
<tr>
<td>PROJ-GR-DT</td>
</tr>
<tr>
<td>ST-CCILLGE</td>
</tr>
</tbody>
</table>
Figure 4-25 Continued

LEVEL-ID
PRI-MAJOR
SBC-MAJOR
ALV-NAME
DEG-DGUGHT
NO-SIST
NO-BRCH
SOC-FR-ORG
FRO-FR-ORG
TUIT-ASST
MAR-STATUS
001
STU-NAME
003
F-NAME
M-NAME
SSAN-ID
002
SPECIAL
TEST
006
TO
SHOW
THAT
WHAT
HAPPENS
IS
003
SPECIAL
TEST
THAT
001
 waking
THAT
THE
003
SPECIAL
TEST
WORKS
001
MODEL
001
MAKE
002
MODEL
BODY-NO
004
ENG-NO
STYLE
Figure 4-25 Continued

EXTERIOR
INTERIOR  002
MODEL
BODY-NO  004
YEAR
WEIGHT
COST
PRICE  001
ENG-NO  005
NC-CYL
DISPL
ENG-WT
ENG-COST
HP  002
EXTERIOR
STYLE  001
INTERIOR  002
EXTERIOR
STYLE  003
NC-DCORS
PAINT-CODE
TRIM-PK  003
STYLE
EXTERIOR
INTERIOR  004
CARGO-SP
UPHCLSTERY
SEATS
ASSARY-PK  001
PRINT-CODE  002
COLOR-1
TONE-NO  001
TONE-NO  002
COLOR -2
PAINT-CODE
5.0 CONCLUSION

To the unsophisticated user a Relational Data Model is difficult to conceptualize. It presents him with a unique task of data relation development which normally would be beyond his capability. The FDGEN as implemented provides the user with a sophisticated tool to aid him in the development of a Relational Data Base. The user is able to enter his data in logical segments and transform it into sets of related data which can be manipulated into Functional Dependencies. The manipulation of the data is guided by the FDGEN Program but directed by the user. This process allows for maximum interaction with the user, the one who knows most about the data and its interrelationship. The final product is a set of FD's which is provided to the user and is made available for synthesis into 3NF Relations by the FDA Program of the Third Normal Form Synthesis System. (16)

5.1 ENHANCEMENT CONSIDERATIONS

The FDGEN was designed to provide its user with a copy of its final product - a list of the Functional Dependencies generated. A user after reviewing the list of FD's and discovering that incorrect FD's were generated might not be able to recall the interactive query responses which produced the incorrect FD's. This would mean that the user would have to regenerate the FD's and hope that he did not make the same error again.
The possibility of trial and error manipulations by a user can be eliminated by making a few modifications to the FDGEN design. The essence of the modification would be to provide the user with a copy of each query and his response to that query. This modification would allow the user to map a new course of action prior to regeneration of the FD's.

The modifications can be accomplished by assigning each query as a constant to an elementry data item; replacing each query in the Procedure Division with its equivalent elementry data item; and following each query, insert the assignment of the query to the output file for output to the line printer. All query responses are output to the line printer also. The modification can be implemented without increasing the current program size.

A second enhancement which is applicable for inclusion in the FDGEN Subsystem is a data relation preprocessor module. The preprocessor module would extend the capability of the unsophisticated user with respect to the generation of FD's. It would allow him to enter all related data as logical sets. This means that data could be entered while in First Normal Form (1NF). In order for the data to be in 1NF all Data-Name-Items in the relation must be atomic. The current system restricts each relation to those Data-Name-Items which will be used for the construction of one FD. Included in the FD may be concatenated Determinants from other relations within the hierarchical report. The
relation as used is synonymous to a group.

The addition of a preprocessor module to the PDGEN Program would require major design modifications to the input modules of the program. The preprocessor would be interactive and would replace the current Parse-Record module and its submodules.
Reference


5. Tsichritzis, pp. 42.


8. Maryanski, pp. 2.


15. Maryanski, pp. 2-5.

Bibliography


APPENDIX-A

FDGEN ABSTRACT CODE

The FDGEN is of modular construction. All major modules are supported by submodules. All variables are global to all modules.

Standard Declarations are assumed.

Main Module:

FDGEN:

Begin.

Do until all records have been read.

Read a record. Discard line number. If record \texttt{=} END then parse remainder of record.

END.

Insert end marker at end of Data-Name-Table. Mark all Virtual-Data-Items. Mark all Data-Name-Items which appear in more than one report. Identify Determinants. Print output table.

END.

END FDGEN.

Sub-Module:

- Parse remainder of record:

Parse-Record:

Begin.

Separate record into status, level-number, and remainder of the record.

If record has no entries than error. If end of record then return. If level-number is
equal to or less than the last level-number and is not a continuation then increase the concatenation code by 1 for all remaining levels.

Do until end of record.

Check length of first Data-Name-Item in record. If length > 10 then error, truncate remaining characters. Remove first Data-Name-Item from record. Continue construction of the Data-Name-Table.

END.

END.

END Parse-Record.

-Increase concatenation code:
Up-level:

Begin.

with level-number as index. From current index-increment each concatenation code by 1 - Tc end of table.

END.

END up-level.

-Continue construction of the Data-Name-Table:

Build-D-N-Table:

Begin.
Increment table index by 1. Assign Data-Name-Item to name field (fld). Assign level-number to level-no fld. Assign 0 to Detlem-flag fld.

Begin.

Reinitialize concatenation-list. Assign NULL to concatenation-link fld. If not first report then assign index of new report to last report's concatenation-link fld ELSE assign concatenation-code to concatenation-link fld.

END.

END.

END Build-D-N-Table.

-Reinitialize concatenation-list:

Fill-it:

Begin.

Index <-- 1.

Assign 11 to first indexed location.

Do 8 times.

Increment index by 1. Increment assign value by 10. Assign new value to concatenation-code (index).

END.

END.

END Fill-it.
Mark all Virtual-Data-Items:

Virtual-Elim:

Begin.

Index ← 1. Display instructions and a tutorial of a Virtual-Data-Item.

Do until end of Data-Name-Table.

Display report name.

Do until report name changes.

If level-no = 0 then error. Display value of name fld. If Determ-Flag fld = 3 then name previously identified. Ask if name is a Virtual-Data-Item. If yes then assign 3 to Determ-Flag fld.

Increment index by 1.

END.

END.

END.

END Virtual-Elim.

Mark all Data-Name-Items which appear in more than one report:

Report-Intersect:

Begin:

Select first report.

Do until selected report's conc-link fld = NUL.

If level-no fld = 0 then conc-link is index to next report. Compare each name fld of the
current report with each name fld of all subsequent reports within the Data-Name-Table. Select the next report in sequence as the current report.

END.

END.

END Report-Intersect.

-Compare each name field of the current report...

Intersect:

Begin.

Select name fld to be compared. If name fld previously matched then return. Select beginning index from which to begin comparing. Compare name flds until end of table.

END.

END Intersect.

-Compare name fields until end of table:

Loop2:

Begin.

Select name fld to be compared.

Do until end of table.

If name flds match and selected name is not a report name or a Virtual-Data-Item then assign 2 to the Determ-Flag fld. Select next name fld to be compared.
END.

END.

END Loop2.

-Identify Determinants:

Determine-Identify:

Begin.

Display instructions and a tutorial of a Determinant. Clear the Functional Dependency Table, S-Stack Table, and the L-Stack Table. Initialize indices.

Do until end of Data-Name-Table.

Reinitialize concatenation-list.

Do until end of current report.

Increment index by 1. If level-no fld = 0 then conc-link fld is index to next report. Display report name.

Do until all unique conc-link fld values have been found.

Display all name fld values of current report with the same conc-link fld value. Assign each selected name fld value to the S-Stack Table. If name fld value is an intersected item then display asterisk by name.

END.
Display instructions as to Determinant selection.

Do until no more selections.

Accept selected determinant. If spelled incorrectly then error ELSE If duplicate entry then error ELSE assign reply to S-Stack Table. Set Determ-Flag fld to 1.

END.

If no Determinant selected for displayed group then error--discard group ELSE assign reply to L-Stack Table and count all new entries. Build next part of the Functional Dependency Table (FDT).

END.

END.

Prepare to write the FDT to disk.

END.

END Determ-Identify.

-Duplicate entry:

Check-DName:

Begin.

If reply is = to any previous reply for the displayed group then error.

END.

END Check-DName.
-Spelled incorrectly:

Check-Name.

Begin.

If reply does not match any ATTR fld value in S-Stack then error.

END.

END Check-Name.

-Prepare to write the FDT to disk:

Load-table:

Begin.

Assign one record at a time from the FDTABLE in Working Storage to a file name. Write the file to disk until all files have been written.

END.

END Load-table.

-Assign reply to L-Stack Table and count the number of new entries:

Load-Cat-Stack:

Begin.

Digit gets first digit of conc-link fld value.

Case.

Digit > last key: Push all Determinants on L-Stack.

Digit = 1: Push all Determinants on L-Stack.
Digit = last key and last key — 1:
   Pop Determinants off L-Stack until Lev fld value < digit. Push all Determinants on L-Stack.

Digit < last key:
   Pop Determinants off L-Stack until Lev fld value < digit. Push all Determinants on L-Stack.

END Case.

Last key <-- digit.

END.

END Load-Cat-Stack.

NOTE: Last key is initially 1.

-Push all Determinants on L-Stack:
Push-Stack:
Begin.
   Do count times.
      Increment index of L-Stack by 1. D-Name fld <-- Attr fld value. Decrement index of S-Stack by 1.
END.
END.
END Push-Stack.

-Pop all Determinants off L-Stack until Lev fld value < digit:
Pop-Stack:
  Begin.
  Do until Lev fld value < digit or value = 1. Decrease L-Stack index by 1.
  END.
END Pop-Stack.

-Set Determin-Flag field to 1:
Set-Key:
  Begin.
  Set index to beginning of current report.
  Do until all name flds of current report have been compared.
  Increment index of Data-Name-Table by 1. If not a Virtual-Data-Item and name fld value matches Attr field value of S-Stack then assign 1 to Determin-Flag fld.
  END.
END.
END Set-Key.

-Build next part of the Functional Dependency Table:
Build-F-D-Table:
  Begin.
  Status <-- 0. Save L-Stack and S-Stack indices.
  Delete Determinants from S-Stack. Save the number
of Data-Name-Items in Hold-no. If the last level number was greater than 1 then special-case2 Save the last level number referenced. Increment F-D-Table index by 1. Assign number of Determinants in L-Stack to list fld.

(The number must be right justified)
Load that number of Determinants into F-D-Table. If a Determinant identifies itself then build a unique symbol consisting of a colon (:) followed by a unique 2-digit number (i.e., :01).
Increment index by 1. Assign number of Data-Name-Items in S-Stack to F-D-Table. Load that number of Data-Name-Items into F-D-Table. If S-Stack is empty then load the special symbol.

END.
END Build-F-D-Table.

-Special-case1:
Spec-Case:

Begin.


END.
END Spec-Case.

-Display L-Stack:
Disp-Det:
   Begin.
   Display each Determinant. Count the number of
   Determinants.
   END.
   END Disp-Det.

   Ask next question:
Next-Q:
   Begin.
   Ask question. If yes then move all Determinants
   whose level number does not match the saved level
   number to the right side. Compact L-Stack.
   If no then select the Determinant to be moved.
   END.
   END Next-Q.

   Move all Determinants whose level number...
B-New-Relation:
   Begin.
   Initialize L-Stack index to 1. Determine the
   number of Determinants to be moved. Move that
   many Determinants to the right side.
   END.
   END B-New-Relation.

   Move that many Determinants to the right side:
L-to-R:

Begin.

Increment S-Stack index by 1. ATTR fld <-- D-Name fld value. Increment L-Stack index by 1.

END.
END L-to-R.

-Compact L-Stack:
Pack-it-down:

Begin.

Move all remaining D-Name fld values to the low end of L-Stack.

END.
END Pack-it-down.

-Move all remaining D-Name...:
Move-Det2:

Begin.

Compact to the low end of L-Stack all values in L-Stack. Level fld <-- 11.

END.
END Move-Det2.

-Select the Determinant to be moved.
Sel-items:

Begin.

Display instructions.
Do until no more Determinants are selected. ATTR
fld ← D-Name fld value selected.
END.
END.
END Sel-items.

ATTR field ← D-Name field value selected:
Move-det-rt:
Begin.
Set a flag. L-Stack index ← 1. Search L-Stack
for the selected Determinant. If flag still set
then spelling error. L-Stack index ← 1.
Compact L-Stack again. Increment the size of S-
Stack by 1.
END.
END Move-det-rt.

Search L-Stack for the selected Determinant:
Search-stack:
Begin.
If D-Name fld value selected then ATTR fld ← D-
Name fld value Change flag setting. Increment L-
Stack index by 1. ELSE increment L-Stack index by
1.
END.
END Search-Stack.
-Compact L-Stack again:
Compact:

Begin.
Move all remaining D-Name and Level fld values to the low end of L-Stack.

END.
END Compact.

-Special-case2:
Spec-case2:

Begin.
If the level saved is equal to the last level referenced and that level is not 1 then STATUS <-- S. If STATUS is S then move all Determinants from Save-Stack to L-Stack which have higher levels than the highest level listed in Level fld. L-stack index <-- 0. Union all D-Name fld values. If only one Determinant exists then Save L-Stack in Save-Stack ELSE Ask question.

Display L-Stack. If yes then save L-Stack in Save-Stack. Ask next question in sequence. ELSE save L-Stack in Save-Stack.

END.
END Spec-case2.

-Move all Determinants from Save-Stack...:
Sav-move:
Begin.

Save stack index <-- 1. Transfer all Determinants which meet the requirements. Save-Stack index <-- 1. Clear Save-Stack.

END.

END Sav-move.

-Transfer all Determinants which meet the requirements:

Transfer-det:

Begin.

Set L-Stack index to top of stack. Save-Stack index <-- 1. If S-Name has a higher level than the tested D-Name then Move all L-Stack values one position higher. L-Stack index <-- 1. D-Name fld <-- S-Name fld value. ELSE check the next S-Name fld values level.

END.

END Transfer-det.

-Move all L-Stack values one position higher:

Make-room:

Begin.

Move each element of L-Stack one position higher so that the low position will be vacant.

END.

END Make-room.
Clear Save-Stack:
Saveclear:
    Begin.
        Fill Save-Stack with spaces.
    END.
END Saveclear.

Union all D-Name field values:
New-union-main:
    Begin.
        Increment L-Stack index by 1. Match <-- D-Name fld value. Flag <-- 1. Perform union operation.
        If flag is 3 then compact L-Stack again ELSE decrement L-Stack index by 1.
    END.
END New-union-main.

Perform union operation:
New-union:
    Begin.
        Increment L-Stack index by 1. If match equals D-Name fld value than flag gets 3.
    END.
END New-union.

Save L-Stack in Save-Stack:
Sav-last:
Begin.

Save-Stack <-- L-Stack.

END.

END Sav-last.

-Ask next question in sequence:

Next-Ques:

Begin.

Display instructions. If yes then move all Determinants whose level number does not match the saved level number to the right side.

Pack L-Stack again. If no then select the Determinant to be moved.

END.

END Next-Ques.

-Pack L-Stack again:

Pack-It-D:

Begin.

Move all remaining D-Name and Level fld values to the low end of L-Stack.

END.

END Pack-It-D.

-Move all remaining D-Name and Level field value...:

Move-Determ:

Begin.
Compact to the low end of L-Stack all values in L-Stack.

END.

END Move-Determ.

-Load that number of Determinants into F-D-Table:
Load1:
Begin.
L-Stack index <-- 1.
Do until L-Stack is empty.
Increment F-D-Table index by 1. List fld <-- D-Name fld value. Increment L-Stack index by 1.
END.
END.

END Load1.

-Delete Determinants from S-Stack:
Delete:
Begin.
Delete each Determinant assigned to L-Stack from S-Stack twice.
END.
END Delete. (Delete includes Remove and Check-List)

-Load that number of Data-Name-Items into F-D-Table:
Load2:
Begin.

S-Stack index <-- 1.

Do until S-Stack is empty.

Increment F-D-Table index by 1. List fld <-- ATTR fld value. Increment S-Stack index by 1.

END.

END.

END load2.

-Print output file:

Print-F-D-Table:

Begin.

F-D-Table index <-- 1. Do until end of F-D-Table.

Build left side of output. Add arrow to output. Build right side of output. Print output.

END.

END.

END Print-F-D-Table.

- Build left side of output:

Build-L-S:

Begin.

Increment F-D-Table index by 1.

Do list fld value times.

Generate a string of Determinants separated
by blanks. Increment F-D-Table index by 1.

END.

END.

END Build-L-S.

-Build right side of output:

Build-R-S:

Begin.

Increment F-I-Table index by 1.

Do list fld value times.

Continue generating the string consisting of Determinants and the arrow by adding Data-Name-Items to the string. Increment F-D-Table index by 1.

END.

END.

END Build-R-S.
IDENTIFICATION DIVISION.
PROGRAM-ID. F-D-GENERATOR.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. NCP-CENTURY-8200.
OBJECT-COMPUTER. NCP-CENTURY-8200.
INPUT-OUTPUT SECTION.
FILE-COMPLETE.
SELECT USERI-FILE ASSIGN TO DISC.
SELECT F-D-TABLE ASSIGN TO DISC.
SELECT F-D-PRINT ASSIGN TO PRINTER.
DATA DIVISION.
FILE SECTION.
FD USERI-FILE BLOCK CONTAINS 6 RECORDS, RECORD CONTAINS 76 CHARACTERS, LABEL RECORDS ARE STANDARD.
FD USERI-RECORD PIC X(76).
FD F-D-TABLE BLOCK CONTAINS 51 RECORDS, RECORD CONTAINS 10 CHARACTERS, LABEL RECORDS ARE STANDARD.
FD F-D-TABLE-LELE PIC X(10).
FD F-D-PRINT LABEL RECORDS ARE OMITTED.
PRINT-LINE PIC X(132).
WORKING-STORAGE SECTION.
FDTABLE.
02 LIST PIC X(10) OCCURS 250 TIMES INDEXED BY F-D-INDEX.
DATA-N OCCURS 200 TIMES INDEXED BY ANINDEX.
05 NAME PIC X(10).
05 LEVEL-NO PIC 9.
05 CONC-LINK PIC XXX.
05 DETERM-FLAG PIC 9.
S-STACK.
02 ATTR PIC X(10) OCCURS 50 TIMES INDEXED BY STACK-INDEX.
02 CAT-STACK OCCURS 20 TIMES INDEXED BY CAT-INDEX.
04 D-NAME PIC X(10).
04 LEV PIC 99.
01 CONC-LIST.
02 CONC-CODE PIC 99 OCCURS 9 TIMES.
01 SAVE-STACK.
02 SA-SP OCCURS 20 TIMES INDEXED BY S-INDEX.
04 S-NAME PIC X(10).
04 SLEV PIC 99.
77 RECONST-RECORD PIC X(76).
77 HOLD PIC X(72).
77 INTERMED-STRING PIC X(70).
77 CHAR-COUNT PIC 999.
77 STAT-HOLD PIC X.
77 LEVEL-HOLD PIC 9.
77 ATTRIBUTE-STRING PIC X(70).
77 BREAK PIC X VALUE IS SPACES.
77 COUNT-1 PIC 99.
77 COUNT-2 PIC 99.
77 WORD-COUNT PIC 9.
77 RESIDUE PIC X(69).
01 ATTRIBUTE PIC X(10).
77 NEXT-LOC PIC 999 VALUE 0.
77 FIND PIC 999 VALUE 1.
77 LOOK PIC 9 VALUE 1.
77 HOLD-LEVEL PIC 4.
77 MAIN-LEVEL PIC 99 VALUE 11.
77 LEVEL-CODE PIC 99.
77 COUNT PIC 999 VALUE 0.
77 KOUNT PIC 999 VALUE 0.
77 REPLY PIC X(11).
77 NEXT-RPT PIC XXX VALUE "1".
77 THAT-MANY PIC 999 VALUE 0.
77 CK-NEXT PIC 999.
77 KNT PIC 999.
THREE3 PIC 9 VALUE 3.
TWO2 PIC 9 VALUE 2.
KEY3 PIC XX.
KEY4 PIC X.
KEY5 PIC X.
NUMBERS PIC 99.
SET-CHECK PIC 9 VALUE 0.
ADD-ON PIC 9.
DET-COUNT PIC 99.
HOLD-DET-COUNT PIC 99.
DET-KOUNT PIC 99.
LEV-HOLD PIC 99.
SAVETOP PIC 99 VALUE 0.
KNIT PIC 99.
KNITTY PIC 99.
KOUNTY PIC 99.
HOL-LEV PIC 9.
SAVE-LEVEL PIC 9 VALUE 0.
PROCEDURE DIVISION.
BEGIN1.
DISPLAY "ANY INFORMATION THAT APPEARS ON THE SCREEN WITHOUT"
"ANY INSTRUCTIONS MAY BE", LINE 10, ERASE.
DISPLAY "REMOVED BY DEPRESSING NEW LINE.**PRESS NEW LINE.."
ACCEPT REPLY.
IF REPLY IS NOT EQUAL TO "DKUJRK" DISPLAY "", ERASE.
OPEN INPUT USERI-FILE.
OPEN OUTPUT F-D-PRINT.
READ USERI-FILE INTO RECONST-RECORD AT FND GO TO V-E.
MOVE 5 TO CHAR-COUNT.
UNSTRING RECONST-RECORD INTO HOLD WITH POINTER CHAR-COUNT.
IF HOLD IS EQUAL TO " END" GO TO RF.
PERFORM PARSE-RECORD THRU NEXT-RECORD.
GO TO PE.
V-E.
CLOSE USERI-FILE.


149* 001490  SET ANINDEX UP BY 1.
150* 001500  MOVE "END" TO NAME(ANINDEX).
151* 001510  MOVE "XXX" TO CONC-LINK(ANINDEX).
152* 001520  PERFORM VIRTUAL-ELIM THRU VIRTUAL-END.
153* 001530
154* 001540  PERFORM REPORT-INTERSECT THRU THE-BOTTOM.
155* 001550
156* 001560  OPEN OUTPUT F-D-TABLE.
157* 001570  PERFORM DETERM-IDENTIFY THRU LAST-PROC.
158* 001580
159* 001590
160* 001600
161* 001610  PERFORM PRINT-F-D-TABLE THRU THATS-ALL.
162* 001620
163* 001630  CLOSE F-D-TABLE.
164* 001640  CLOSE F-D-PRINT.
165* 001650  STOP RUN.
166* 001660
167* 001670
168* 001680  PARSE-RECORD.
169* 001690  UNSTRING HOLD INTO STAT-HOLD, LEVEL-HOLD, INTERMED-STRING.
170* 001700  MOVE 0 TO COUNT-1.
171* 001710  INSPECT INTERMED-STRING TALLYING COUNT-1 FOR LEADING BREAK.
172* 001720  IF COUNT-1 IS EQUAL TO 0
173* 001730  MOVE INTERMED-STRING TO ATTRIBUTE-STRING
174* 001740  ELSE ADD 1 TO COUNT-1
175* 001750  MOVE COUNT-1 TO CHAR-COUNT
176* 001760  UNSTRING INTERMED-STRING INTO ATTRIBUTE-STRING
177* 001770  WITH POINTER CHAR-COUNT.
178* 001780  MOVE 1 TO CHAR-COUNT, K-TRACK.
179* 001790  IF LEVEL-HOLD IS EQUAL TO 0 IF LEVEL-HOLD IS EQUAL TO
180* 001800  SET-TRAP MOVE 1 TO SET-TRAP
181* 001810  ELSE DISPLAY NAME(ANINDEX), POSITION 35
182* 001820  DISPLAY "ERROR--THIS REPORT HAS NO ENTRIES."
183* 001830  POSITION 23
184* 001840  DISPLAY "DO YOU WISH IT ELIMINATED-YES/NO."
185* 001850  POSITION 23
ACCEPT REPLY, POSITION 40

IF REPLY IS EQUAL TO "Y" OR
REPLY IS EQUAL TO "YE" OR

REPLY IS EQUAL TO "YES"
SET ANINDEX DOWN BY 1.

IF LEVEL-HOLD IS GREATER THAN 0 MOVE 0 TO SET-TRAP.

IF LEVEL-HOLD IS EQUAL TO " " OR ATTRIBUTE-STRING
IS EQUAL TO "END" GO TO NEXT-RECORD.

IF LEVEL-HOLD IS GREATER THAN 0

IF STAT-HOLD IS EQUAL TO "N" OR STAT-HOLD IS EQUAL TO
"S"

SUBTRACT LEVEL-HOLD FROM 10 GIVING THAT-MANY

PERFORM UP-LEVEL THAT-MANY TIMES

MOVE 11 TO CONC-CODE(1)

SUBTRACT THAT-MANY FROM 10 GIVING LEVEL-HOLD.

SEPERATE.

UNSTRING ATTRIBUTE-STRING DELIMITED BY BREAK INTO RESIDUE;
COUNT IN COUNT-1 WITH POINTER K-TRACK.

IF COUNT-1 IS LESS THAN 11 GO TO NEXT-LINE.

DISPLAY "DATA-NAME", LINE 5, POSITION 5, ERASE.

DISPLAY RESIDUE, LINE 5, POSITION 15.

DISPLAY "EXCEEDS THE ALLOWABLE NUMBER OF CHARACTERS;"
LINE 5, POSITION 35.

DISPLAY "EXCESS CHARACTERS WILL BE TRUNCATED;", LINE 7,

ACCEPT REPLY.

IF REPLY IS NOT EQUAL TO "ZYJPU" DISPLAY " ", ERASE.

NEXT-LINE.

MOVE 0 TO WORD-COUNT.

UNSTRING ATTRIBUTE-STRING DELIMITED BY ALL BREAK INTO
ATTRIBUTE, COUNT IN COUNT-1, WITH POINTER CHAR-COUNT
TALLYING IN WORD-COUNT.

IF COUNT-1 IS EQUAL TO 0 OR WORD-COUNT IS EQUAL TO 0
GO TO NEXT-RECORD.

PERFORM BUILT-ON-TABLE THRU TAIL-END.
223* 002230  GO TO SEPERATE,
224* 002240 NEXT-RECORD.
225* 002250 EXIT.
226* 002260 UP-LEVEL.
227* 002270 ADD 1 TO CONC-CODE(LEVEL-HOLD).
228* 002280 ADD 1 TO LEVEL-HOLD.
229* 002290 BUILD-D-N-TABLE.
230* 002300 ADD 1 TO NEXT-LOC.
231* 002310 SET ANINDEX TO NEXT-LOC.
232* 002320 MOVE ATTRIBUTE TO NAME(ANINDEX).
233* 002330 MOVE LEVEL-HOLD TO LEVEL-NO(ANINDEX).
234* 002340 MOVE 0 TO DETERM-FLAG(ANINDEX).
235* 002350
236* 002360
237* 002370 IF LEVEL-HOLD IS NOT EQUAL TO 0 GO TO NEXT-STEP.
238* 002380 MOVE 11 TO MAIN-LEVEL.
239* 002390 MOVE 1 TO LEVEL-HOLD.
240* 002400 PERFORM FILL-IT 9 TIMES.
241* 002410 MOVE 0 TO LEVEL-HOLD.
242* 002420
243* 002430 MOVE "NIL" TO CONC-LINK(ANINDEX).
244* 002440 SET ANINDEX TO FIND.
245* 002450
246* 002460 IF NEXT-LOC IS NOT EQUAL TO 1 MOVE NEXT-LOC TO
247* 002470 CONC-LINK(ANINDEX).
248* 002480 MOVE NEXT-LOC TO FIND.
249* 002490 SET ANINDEX TO NEXT-LOC.
250* 002500 GO TO TAIL-END.
251* 002510 NEXT-STEP.
252* 002520
253* 002530 MOVE CONC-CODE(LEVEL-HOLD) TO CONC-LINK(ANINDEX).
254* 002540 TAIL-END.
255* 002550 EXIT.
256* 002560 THE END OF BUILD-DATA-NAME-TABLE.
257* 002570 FILL-IT.
258* 002580 MOVE MAIN-LEVEL TO CONC-CODE(LEVEL-HOLD).
259* 002590 ADD 1 TO LEVEL-HOLD.
ADD 10 TO MAIN-LEVEL.

DISPLAY "USER INSTRUCTIONS": LINE 4, POSITION 32, ERASE.

DISPLAY "***************": LINE 5, POSITION 32.

DISPLAY "VIRTUAL DATA ITEM IDENTIFICATION PHASE": LINE 7,
POSITION 21.

DISPLAY "THE USER MUST IDENTIFY ALL VIRTUAL DATA ITEMS WHICH
HAVE BEEN ENTERED AS DATA-NAME-ITEMS. THIS WILL BE DONE BY EN
TERING": LINE 11, POSITION 10.

DISPLAY "A YES FOR EACH IDENTIFIED VIRTUAL DATA ITEM. THE SY
"STEM WILL": LINE 12, POSITION 10.

DISPLAY "DISPLAY ONE POTENTIAL VIRTUAL DATA ITEM AT A TIME.
"YOU": LINE 13, POSITION 10.

DISPLAY "MUST ENTER A YES (Y) IF IT IS A VDI AND A NO (N) IF
"NOT": LINE 14, POSITION 10.

DISPLAY "DO YOU DESIRE AN EXPLANATION OF A VIRTUAL DATA ITEM?
"ENTER YES OR NO": LINE 17, POSITION 10.

ACCEPT REPLY, LINE 19, POSITION 5.

IF REPLY IS EQUAL TO "Y" OR

REPLY IS EQUAL TO "YES"

DISPLAY "A VIRTUAL DATA ITEM IS A DATA-NAME-ITEM WHICH IS
"DERIVABLE": LINE 5, POSITION 5, ERASE.

DISPLAY "BY COMPUTATION INVOLVING DATA-NAME-ITEMS FROM WI
"THIN THE": LINE 6, POSITION 5.

DISPLAY "SAME DATA BASE. THIS SAYS THAT THE DATA-NAME-ITE
"M CAN": LINE 7, POSITION 5.

DISPLAY "BE DERIVED FROM OTHER DATA-NAME-ITEMS THAT ARE
"WITHIN THE": LINE 8, POSITION 5.

DISPLAY "SAME DATA BASE OR POSSIBLY THE SAME REPORT."
LINE 9: POSITION 5
DISPLAY "AN EXAMPLE OF A VIRTUAL DATA ITEM WOULD BE TOTAL
"HOURS-WORKED", LINE 12, POSITION 5
LINE 13, POSITION 5
DISPLAY "WHEN WITH IN THE SAME REPORT THE FOLLOWING DATA-
"NAME-ITEMS ARE AVAILABLE", LINE 15, POSITION 10
DISPLAY "HOURS-WORKED-PER-WEEK", LINE 16, POSITION 10
DISPLAY "WEEKS-WORKED-PER-YEAR", LINE 17, POSITION 10
DISPLAY "IF YOU REQUIRE ADDITIONAL ASSISTANCE ENTER HELP."
LINE 20, POSITION 15
ACCEPT REPLY, LINE 21, POSITION 5.
LINE 22, POSITION 5
IF REPLY IS EQUAL TO "H" OR
IF REPLY IS EQUAL TO "HEL" OR
REPLY IS EQUAL TO "HELP" DISPLAY
"PLEASE CALL RICH AT 8-316-657-6339"
LINE 5, POSITION 10, ERASE
ELSE DISPLAY "WE ARE READY TO START.", LINE 5,
POSITION 29, ERASE.
ACCEPT REPLY.
IF REPLY IS NOT EQUAL TO "JDTIFH" DISPLAY "", ERASE.
BEGIN.
SET ANINDEX TO 1.
STRT.
IF CONC-LINK(ANINDEX) IS EQUAL TO "XXX" GO TO FINISH.
DISPLAY "REPORT TO BE CHECKED FOR VIRTUAL DATA ITEMS IS ",
LINE 5, POSITION 12, ERASE.
DISPLAY NAME(ANINDEX), LINE 5, POSITION 59.
SET ANINDEX UP BY 1.
VIRT-CHECK.
IF LEVEL-NO(ANINDEX) IS EQUAL TO 0
DISPLAY "ERROR-REPORT WITH NO ENTRIES HAS BEEN FOUND"
ACCEPT REPLY.
IF REPLY IS NOT EQUAL TO "KDEKFOG" DISPLAY "", ERASE
GO TO STRT.
DISPLAY NAME(ANINDEX), LINE 7, POSITION 35.
DISPLAY ", LINE 7, POSITION 50.
IF DETERM-FLAG(ANINDEX) IS EQUAL TO 3 DISPLAY "THIS IS A VDI"
   + LINE 7, POSITION 50.

DISPLAY "IS THIS DATA-NAME-ITEM A VDI? (YES OR NO)"*, LINE 9,
   + POSITION 10.

ACCEPT REPLY*, LINE 9, POSITION 52.

IF REPLY IS EQUAL TO "Y" OR

   + REPLY IS EQUAL TO "YES" OR

   + REPLY IS EQUAL TO "YES"
   + MOVE 3 TO DETERM-FLAG(ANINDEX)
   + ADD 1 TO COWNT
   + DISPLAY " ", LINE 1, ERASE.

SET ANINDEX UP BY 1.

IF LEVEL-NO(ANINDEX) IS EQUAL TO 0 GO TO STRT.

IF CONC-LINK(ANINDEX) IS NOT EQUAL TO "XXX"

   + GO TO VIRT-CHECK.

FINISH.

MOVE 0 TO KOUNT.

DISPLAY "ALL REPORTS AND THEIR RESPECTIVE DATA-NAME-ITEMS HAV
   + E BEEN"*, LINE 5, POSITION 10, ERASE.

DISPLAY "DISPLAYED, DO YOU WISH TO VERIFY THE LIST OF VIRTUA
   + L DATA "*, LINE 6, POSITION 10.

ACCEPT REPLY LINE 7, POSITION 35.

IF REPLY IS EQUAL TO "N" OR

   + REPLY IS EQUAL TO "NO" GO TO LAST-CHANCE.

SET ANINDEX TO 1.

LOOP.

IF COWNT IS EQUAL TO KOUNT GO TO LAST-CHANCE.

IF CONC-LINK(ANINDEX) IS EQUAL TO "XXX" GO TO LAST-CHANCE.

IF DETERM-FLAG(ANINDEX) IS EQUAL TO THREE

   + DISPLAY NAME(ANINDEX)*, LINE 10 POSITION 35, ERASE

DISPLAY "IF THIS IS A VALID VIRTUAL DATA ITEM ENTER YES OTHER
"WISE NO." LINE 12, POSITION 10
ACCEPT REPLY LINE 14, POSITION 20
IF REPLY IS EQUAL TO "N" OR
REPLY IS EQUAL TO "NO" MOVE 0 TO DETERM-FLAG(ANINDEX)
DISPLAY "", ERASE
ELSE DISPLAY "", ERASE
ADD 1 TO KOUNT
SET ANINDEX UP BY 1
GO TO LOOP.
SET ANINDEX UP BY 1.
GO TO LOOP.
GO TO LOOP.
LAST-CHANCE.
DISPLAY "DO YOU WISH TO REVIEW THE LIST OF DATA-NAME-ITEMS"
"OF EACH REPORT", LINE 5, Position 10, ERASE.
DISPLAY "AGAIN TO INSURE THAT ALL VIRTUAL-DATA-ITEMS HAVE"
"BEEN IDENTIFIED.", LINE 6, Position 10.
DISPLAY "ENTER YES OR NO", Line 7, Position 10.
ACCEPT REPLY Line 7, Position 30.
IF REPLY IS EQUAL TO "Y" OR
REPLY IS EQUAL TO "YE" OR
REPLY IS EQUAL TO "YES" GO TO BEGIN2.
VIRTUAL-END.
EXIT.
END OF VIRTUAL-IDENTIFY
REPORT-INTERSECT.
DISPLAY "", ERASE.
SET ANINDEX TO 1.
LOOP-1.
IF LEVEL-NO(ANINDEX) IS EQUAL TO 0 MOVE CONC-LINK(ANINDEX)
TO NEXT-RPT.
IF NEXT-RPT IS EQUAL TO "NULL" GO TO THE-BOTTOM.
SET SUB-ANINDEX TO ANINDEX.
MOVE NEXT-RPT TO CONVERT1.
SUBTRACT SUB-ANINDEX + 1 FROM CONVERT1 GIVING THAT-MANY.
408*  004080  MOVE NEXT-RPT TO CK-NEXT.
409*  004090  ADD 1 TO CK-NEXT.
410*  004100  SET KNT TO ANINDEX.
411*  004110  PERFORM INTFRSFT THRU INTERSECT-END THAT-MANY TIMES.
412*  004120  SET ANINDEX TO CONVERTI.
413*  004130  GO TO LOOP-1.
414*  004140  THE-BOTTOM.
415*  004150  EXIT.
416*  004160  INTERSECT.
417*  004170  ADD 1 TO KNT.
418*  004180  SET ANINDEX TO KNT.
419*  004190  IF DETERM-FLAG(ANINDEX) IS EQUAL TO TWO2 GO TO INTERSECT-END.
420*  004200  MOVE NAME(ANINDEX) TO MATCH.
421*  004210  
422*  004220  SET ANINDEX TO CK-NEXT.
423*  004230  
424*  004240  PERFORM LOOP-2 UNTIL CONC-LINK(ANINDEX) IS EQUAL TO "XXX".
425*  004250  INTERSECT-END.
426*  004260  EXIT.
427*  004270  LOOP-2.
428*  004280  IF MATCH IS EQUAL TO NAME(ANINDEX) AND LEVEL-NO(ANINDEX)
429*  004290  IS NOT EQUAL TO 0 AND DETERM-FLAG(ANINDEX) IS NOT EQUAL
430*  004300  TO THREE3 MOVE TWO2 TO DETERM-FLAG(ANINDEX)
431*  004310  SET NEXT-LOC TO ANINDEX
432*  004320  SET ANINDEX TO KNT
433*  004330  MOVE TWO2 TO DETERM-FLAG(ANINDEX)
434*  004340  SET ANINDEX TO NEXT-LOC
435*  004350  SET ANINDEX UP HY 1
436*  004360  ELSE SET ANINDEX UP BY 1.
437*  004370  END LOOP-2.
438*  004380  END INTERSECT.
439*  004390  END REPORT-INTERSECT.
440*  004400  
441*  004410  DETERM-IDENTIFY.
442*  004420  DISPLAY "USER REQUIREMENTS", LINE 5, POSITION 31, ERASE.
443*  004430  DISPLAY "*******************", LINE 6, POSITION 31.
444*  004440  DISPLAY "DETERMINANT IDENTIFICATION PHASE", LINE 8, POSITION
25.
DISPLAY "YOU MUST IDENTIFY FROM EACH GROUP OF DATA-NAME-ITEMS
" DISPLAYED-THOSE". LINE 10*, POSITION 5.
DISPLAY "ITEMS WHICH WILL UNIQUELY IDENTIFY THE REMAINING ITE
" MS WITHIN", LINE 11*, POSITION 5.
DISPLAY "THE GROUP, THOSE DATA-NAME-ITEMS WHICH IDENTIFY THE
" REMAINING ITEMS", LINE 12*, POSITION 5.
DISPLAY "ARE CALLED DETERMINANTS. EACH DETERMINANT IDENTIFIED
" MUST HE", LINE 13*, POSITION 5.
DISPLAY "ENTERED UPON REQUEST. SPELLING IS CRITICAL. ANY OUES
" TIONS? YES/NO", LINE 14*, POSITION 5.
ACCEPT REPLY, LINE 15*, POSITION 5.
IF REPLY IS EQUAL TO "N" OR
REPLY IS EQUAL TO "NO" GO TO SKIP-LAST.
DISPLAY "THE FOLLOWING DATA-NAME-ITEMS ARE DISPLAYED****
"WAREHOUSE-NO", LINE 5*, POSITION 10, ERASE.
DISPLAY "SPACE-AVAILABLE, ITEMS-STORRED, STORAGE-LEVEL****
LINE 6*, POSITION 10.
DISPLAY "THE SINGLE DATA ITEM WHICH COULD IDENTIFY THE OTHER
" ITEMS WOULD", LINE 8*, POSITION 10.
DISPLAY "BE WAREHOUSE-NO"****IT IS THE DETERMINANT*****
LINE 9*, POSITION 10.

467* 004670 SKIP-LAST.
469* 004690 DISPLAY "ENTER GO WHEN YOU ARE READY TO BEGIN.", LINE 16.
470* 004700 ACCEPT REPLY, LINE 17, POSITION 15.
471* 004710 IF REPLY IS EQUAL TO "G" OR
472* 004720 IF REPLY IS EQUAL TO "GO" DISPLAY " ", ERASE.
474* 004740 SET F-D-INDEX TO 1.
475* 004750 PERFORM CLEARFD 250 TIMES.
476* 004760 SET F-D-INDEX TO 0.
477* 004770 SET AF INDEX TO 1.
478* 004780 SET STACK-INDEX TO 1.
479* 004790 PERFORM MAKE-NULL 50 TIMES.
480* 004800 SET CAT-INDEX TO 1.
481* 004810 PERFORM CAT-CLEAR 20 TIMES.
482* 004820 MAIN-LOOP.
483* 004830  SET CAT-INDEX TO 0.
484* 004840  SET S-INDEX TO 1.
485* 004850  PERFORM SAVCLEAR 20 TIMES.
486* 004860   MOVE 0 TO SAVETOP.
487* 004870   MOVE 1 TO LAST-LEV.
488* 004880   MOVE CONC-LINK(ANINDEX) IS EQUAL TO "XXX"
489* 004890   MOVE "END" TO NAME(ANINDEX)
490* 004900   SET F-D-INDEX UP BY 1
491* 004910   MOVE "XXXXXXXXX" TO LIST(F-D-INDEX)
492* 004920   PERFORM NEXT-PROC
493* 004930   GO TO LAST-PROC.
494* 004940
495* 004950   MOVE 11 TO MAIN-LEVEL.
496* 004960   MOVE 1 TO LEVEL-HOLD.
497* 004970   PERFORM FILL-IT 9 TIMES.
498* 004980   MOVE 1 TO LEVEL-HOLD.
499* 004990
500* 005000
501* 005010   SET TOPP TO ANINDEX.
502* 005020   IF LEVEL-NOT(ANINDEX) IS EQUAL TO 0 MOVE CONC-LINK
503* 005030     (ANINDEX) TO NEXT-RPT.
504* 005040   DISPLAY "GROUPS TO BE DISPLAYED ARE FROM REPORT", LINE 3
505* 005050   POSITION 16, ERASE.
506* 005060   DISPLAY NAME(ANINDEX), LINE 3, POSITION 55.
507* 005070 SECOND-LOOP.
508* 005080   MOVE 0 TO KOUNT, DOUBLE-CK, SET-FLAG, SET-CHECK.
509* 005090   MOVE 7 TO LINE-NO.
510* 005100 SECOND-PART.
511* 005110
512* 005120   MOVE CONC-CODE(LEVEL-HOLD) TO KEY1.
513* 005130   SET STACK-INDEX TO 0.
514* 005140 THIRD-LOOP.
515* 005150   ADD 1 TO LINE-NO.
516* 005160 THIRD-LOOP1.
517* 005170   SET ANINDEX UP BY 1.
518* 005180*LOOKING FOR END OF TABLE
519*  005190  SET KNT TO ANINDEX.
520*  005200  IF KNT IS EQUAL TO NEXT-RPT OR
521*  005210    LEVEL-NO(ANINDEX) IS EQUAL TO 0 OR
522*  005220    CONC-LINK(ANINDEX) IS EQUAL TO "XXX"
523*  005230    GO TO TEST-GROUP.
524*  005240*GROUP EXIT
525*  005250  IF CONC-LINK(ANINDEX) IS NOT EQUAL TO KEY1 GO TO THIRD-LOOP1.
526*  005260*SKIP VIRTUAL DATA ITEM
527*  005270  IF DETERM-FLAG(ANINDEX) IS EQUAL TO THREF3 GO TO THIRD-LOOP1.
528*  005280  MOVE 0 TO DOUBLE-CK, SET-CHECK.
529*  005290    MOVE 1 TO SET-FLAG.
530*  005300  DISPLAY NAME(ANINDEX), LINE LINE-NO, POSITION 5.
531*  005310*LOAD STACK
532*  005320  SET STACK-INDEX UP BY 1.
533*  005330  MOVE NAME(ANINDEX) TO ATTR(STACK-INDEX).
534*  005340*CHECK FOR INTERSECTED DATA-NAME-ITEMS
535*  005350  IF DETERM-FLAG(ANINDEX) IS EQUAL TO TWO2 DISPLAY "###".
536*  005360  LINF LINE-NO, POSITION 16.
537*  005370    ADD 1 TO KOUNT.
538*  005380  GO TO THIRD-LOOP.
539*  005390   TEST-GROUP.
540*  005400  MOVE 0 TO L-S-COUNT.
541*  005410  IF SET-FLAG IS EQUAL TO 1 GO TO NEXT-GROUP.
542*  005420   ADD 1 TO LEVEL-HOLD.
543*  005430    IF LEVEL-HOLD IS GREATER THAN B GO TO SET-GROUP.
544*  005440  SET ANINDEX TO T0PP.
545*  005450  MOVE 0 TO SET-FLAG.
546*  005460  GO TO SECOND-PART.
547*  005470   SET-GROUP.
548*  005480  ADD 1 TO SET-CHECK.
549*  005490  MOVE 2 TO LEVEL-HOLD.
550*  005500  MOVE 8 TO THAT-MANY.
551*  005510   PERFORM UP-LEVEL THAT-MANY TIMES.
552*  005520  MOVE 2 TO LEVEL-HOLD.
553*  005530  IF SET-CHECK IS EQUAL TO 4 GO TO MAIN-LOOP.
554*  005540  SET ANINDEX TO T0PP.
555*  005550  GO TO SECOND-PART.
BUILD WORDING FOR DISPLAY TO FOLLOW.
NEXT-GROUP.
IF TOPP IS GREATER THAN 1 GO TO GET-NAME.
SET AMINDEX TO TOPP.
IF KOUNT IS GREATER THAN 1 MOVE 3 TO KOUNT.
PERFORM FIX-WORD1 UNTIL KOUNT NOT EQUAL TO 0.
PERFORM FIX-WORD2 UNTIL KOUNT NOT EQUAL TO 1.
PERFORM FIX-WORD3 UNTIL KOUNT NOT EQUAL TO 3.
GO TO EXPLAN1.

FIX-WORD1.

MOVE "NONE" TO WORD.
MOVE "NO" TO ARTICAL.
MOVE "DETERMINANTS" TO ONE-OR-MORE.
MOVE 99 TO KOUNT.

FIX-WORD2.

MOVE "A" TO ARTICAL.
MOVE "A DETERMINANT" TO ONE-OR-MORE.
MOVE "ONE" TO WORD.
MOVE "IS" TO VPRF.
MOVE 100 TO KOUNT.

FIX-WORD3.

MOVE "A" TO ARTICAL.
MOVE "ARE" TO VPRF.
MOVE "SEVERAL" TO WORD.
MOVE "DETERMINANT" TO ONE-OR-MORE.
MOVE 100 TO KOUNT.

EXPLAN1.

DISPLAY WORD; LINE 5, POSITION 20.
DISPLAY "OF THE LISTED DATA-NAME-ITEMS MATCHED OTHER DATA--
LINE 5, POSITION 28.
DISPLAY "NAME-ITEMS IN OTHER REPORTS. THIS MEANS THAT YOU
"HAVE": LINE 6, POSITION 20.
DISPLAY ARTICAL; LINE 7, POSITION 20.
DISPLAY "CLUE AS TO WHICH DATA-NAME-ITEM/S MIGHT BE": LINE 7
593× 005930  POSITION 23.
594× 005940  DISPLAY ONE-OR-MORE, LINE 8, POSITION 20.
595× 005950*CHECK THE NUMBER OF DATA-NAME-ITEMS MARKED.
596× 005960
597× 005970  IF KCOUNT IS EQUAL TO 100 GO TO MORE-THAN-ONE.
598× 005980  DISPLAY "PLEASE ENTER THE DATA-NAME-ITEM WHICH MEETS THE"
599× 005990  LINE 10, POSITION 20.
600× 006000  DISPLAY "REQUIREMENTS OF A DETERMINANT FOR THIS GROUP. THERE"
601× 006010  LINE 11, POSITION 20.
602× 006020  DISPLAY "CAN BE MORE THAN ONE DETERMINANT PER GROUP. FAILURE"
603× 006030  LINE 12, POSITION 20.
604× 006040  DISPLAY "TO IDENTIFY AT LEAST ONE DETERMINANT WILL CAUSE AN"
605× 006050  LINE 13, POSITION 20.
606× 006060  DISPLAY "ERROR CONDITION. THE SYSTEM WILL REQUEST THAT YOU"
607× 006070  LINE 14, POSITION 20.
608× 006080  DISPLAY "ENTER EACH SELECTED DETERMINANT NAME ONE AT A TIME"
609× 006090  LINE 15, POSITION 20.
610× 006100  DISPLAY "WHEN INDICATED. IF ALL DETERMINANT NAMES HAVE BEEN"
611× 006110  LINE 16, POSITION 20.
612× 006120  DISPLAY "ENTERED, ENTER XX WHEN THE NEXT REQUEST APPEARS."
613× 006130  LINE 17, POSITION 20.
614× 006140  DISPLAY "EACH NAME ENTERED MUST BE SPelled EXACTLY AS IT"
615× 006150  LINE 18, POSITION 20.
616× 006160  DISPLAY "APPEARS ON THE SCREEN.; LINE 19, POSITION 20.
617× 006170*MORE THAN ONE DATA-NAME-ITEM WAS UNDERLINED.
618× 006180 MORE-THAN-ONE.
619× 006190*A CHECK TO INSURE THAT THE CORRECT MESSAGE WAS Displayed
620× 006200  IF KCOUNT IS EQUAL TO 99 GO TO GET-NAME.
621× 006210  DISPLAY "THE DATA-NAME-ITEM/S WHICH ALSO APPEAR IN OTHER Repo"
622× 006220-  "RTS", LINE 10, POSITION 20.
623× 006230  DISPLAY "RTS", LINE 11, POSITION 20.
624× 006240  DISPLAY "MARKED BY *. WHEN CONSIDERINO WHICH DATA-NAME-ITEMS"
625× 006250  LINE 11, POSITION 20.
626× 006260  DISPLAY "IDENTIFY THE REMAINING ITEMS IN THE GROUP--FIRST"
627× 006270  LINE 12, POSITION 20.
628× 006280  DISPLAY "CONSIDER THE MARKED ITEMS. THERE CAN BE MORE"
629× 006290  LINE 13, POSITION 20.
DISPLAY "THAN ONE DETERMINANT PER GROUP. FAILURE TO"
LINE 14, POSITION 20.
DISPLAY "IDENTIFY AT LEAST ONE DETERMINANT WILL CAUSE AN"
LINE 15, POSITION 20.
DISPLAY "ERROR CONDITION. THE SYSTEM WILL REQUEST THAT YOU"
LINE 16, POSITION 20.
DISPLAY "ENTER EACH SELECTED DETERMINANT NAME ONE AT A TIME"
LINE 17, POSITION 20.
DISPLAY "WHEN INDICATED. IF ALL DETERMINANT NAMES HAVE BEEN"
LINE 18, POSITION 20.
DISPLAY "ENTERED, ENTER XX WHEN THE NEXT REQUEST APPEARS."
LINE 19, POSITION 20.
DISPLAY "EACH NAME ENTERED MUST BE SPelled EXACTLY AS IT"
LINE 20, POSITION 20.
DISPLAY "APPARrs ON THE SCREEN", LINE 21, POSITION 20.
ACCEPT DETERMINANT NAMES FROM CRT
GET-NAME.
DISPLAY "ENTER THE DETERMINANT NAME.", POSITION 20.
ACCEPT REPLY, POSITION 48.
IF REPLY IS EQUAL TO "X" OR
REPLY IS EQUAL TO "XX" OR
REPLY IS EQUAL TO "XXX" DISPLAY "", ERASE
GO TO OUT-OF-NAME.
SET TOP-STACK TO STACK-INDEX.
MOVE 1 TO LOOK.
SET STACK-INDEX TO 1.
PERFORM CHECK-NAME TOP-STACK TIMES.
SET STACK-INDEX TO TOP-STACK.
IF LOOK IS NOT EQUAL TO 7
DISPLAY "YOU SPelled IT WRONG**TRY AGAIN**PLEASE.",
POSITION 20
DISPLAY "PRESS NEW LINE", POSITION 20
ACCEPT REPLY
IF REPLY IS NOT EQUAL TO "KSHTKT" DISPLAY ""
GO TO GET-NAME.
PERFORM CHECK-DNAME L-S-COUNT TIMES.
SET STACK-INDEX TO TOP-STACK.
IF LOOK IS EQUAL TO 8
DISPLAY "DUPLICATE DETERMINANT ENTRY--ENTER A DETERMINANT"
POSITION 20
DISPLAY "NAME ONLY ONCE PER UNIQUE GROUP, PPRESS NEW LINE"
POSITION 20
ACCEPT REPLY
IF REPLY IS NOT EQUAL TO "KDFQIJ" DISPLAY " "
GO TO GET-NAME.

LOAD HOLDING STACK
SET STACK-INDEX UP BY 1.
MOVE REPLY TO ATTR(STACK-INDEX).
ADD 1 TO L-S-COUNT.
SUBTRACT TOPP FROM KNT GIVING NUMBERS.
SET ANINDEX TO TOPP.
PERFORM SET-KEY NUMBERS TIMES.
SET ANINDEX TO TOPP.
GO TO GET-NAME.
NO MORE DETERMINANTS IDENTIFIED IN THAT GROUP.
OUT-OF-NAME.
SET NEXT-LOC TO ANINDEX.
SET ANINDEX TO TOPP.
IF NEXT-LOC IS NOT EQUAL TO TOPP OR L-S-COUNT IS EQUAL TO 0
DISPLAY "A DETERMINANT WAS NOT IDENTIFIED FOR THE LAST GROUP--GROUP IS DESCARDED."
LINE 5, POSITIONS 5
ACCEPT REPLY.
IF REPLY IS NOT EQUAL TO "WLDHUK"
DISPLAY " ", ERASE
ADD 1 TO LEVEL-HOLD
GO TO SECOND-LOOP.

ADD 1 TO LEVEL-HOLD.
CONCATENATE GROUP DETERMINANTS WHICH BELONG AND BUILD THE
FUNCTIONAL DEPENDENCY TABLE.
SET TOP-STACK TO STACK-INDEX.
704* 007040 PERFORM LOAD-CAT-STACK.
705* 007050
706* 007060 SET STACK-INDEX TO TOP-STACK.
707* 007070 PERFORM BUILD-F-D-TABLE.
708* 007080 GO TO SECOND-LOOP.
709* 007090 NEXT-PROC.
710* 007100 SET F-D-INDEX TO 1.
711* 007110 PERFORM LOAD-TABLE UNTIL LIST(F-D-INDEX) IS EQUAL TO "XXXXXXXXX".
712* 007120 LAST-PROC.
713* 007130 EXIT.
714* 007140 CHECK-D-NAME.
715* 007150 IF REPLY IS EQUAL TO ATTR(STACK-INDEX) MOVE 6 TO LOOK.
716* 007160 SET STACK-INDEX DOWN BY 1.
717* 007170 CHECK-NAME.
718* 007180 IF REPLY IS EQUAL TO ATTR(STACK-INDEX) MOVE 7 TO LOOK.
719* 007190 SET STACK-INDEX UP BY 1.
720* 007200 LOAD-TABLE.
721* 007210 WRITE F-D-TABLE-ELF FROM LIST(F-D-INDEX).
722* 007220 SET F-D-INDEX UP BY 1.
723* 007230 THE END OF DETERM-IDENTIFY.
724* 007240 MAKE-NULL.
725* 007250 MOVE SPACES TO ATTR(STACK-INDEX).
726* 007260 SET STACK-INDEX UP BY 1.
727* 007270 CLEAN-HEAD.
728* 007280 MOVE SPACES TO LIST(F-D-INDEX).
729* 007290 SET F-D-INDEX UP BY 1.
730* 007300 MOVES-NAME(S-INDEX).
731* 007310 SAV-CLEAR.
732* 007320 MOVE SPACES TO S-NAME(S-INDEX).
733* 007330 MOVE 00 TO SLEV(S-INDEX).
734* 007340 SET S-INDEX UP BY 1.
735* 007350 CAT-CLEAR.
736* 007360 MOVE SPACES TO D-NAME(CAT-INDEX).
737* 007370 MOVE 00 TO LEV(CAT-INDEX).
738* 007380 SET CAT-INDEX UP BY 1.
739* 007390 LOAD-CAT-STACK.
740* 007400 UNSTRING KEY1 INTO FIRSTN1, FIRSTN2.
741 007410  MOVE FIRSTN1 TO KEY2.
742 007420  IF KEY2 IS GREATER THAN LAST-LEV
743 007430    PERFORM PUSH-STACK L-S-COUNT TIMES.
744 007440  IF KEY2 IS EQUAL TO 1
745 007450    PERFORM PUSH-STACK L-S-COUNT TIMES.
746 007460  IF KEY2 IS EQUAL TO LAST-LEV AND LAST-LEV IS NOT EQUAL TO 1
747 007470    PERFORM POP-STACK THRU ENDREGIN3.
748 007480  IF KEY2 IS EQUAL TO LAST-LEV AND LAST-LEV IS NOT EQUAL TO 1
749 007490    PERFORM PUSH-STACK L-S-COUNT TIMES.
750 007500  IF KEY2 IS LESS THAN LAST-LEV
751 007510    PERFORM POP-STACK THRU ENDREGIN3.
752 007520  IF KEY2 IS LESS THAN LAST-LEV.
753 007530    PERFORM PUSH-STACK L-S-COUNT TIMES.
754 007540  MOVE KEY2 TO LAST-LEV.
755 007550
756 007560
757 007570  PUSH-STACK.
758 007580    SET CAT-INDEX UP BY 1.
759 007590  MOVE ATTR(STACK-INDEX) TO D-NAME(CAT-INDEX).
760 007600  MOVE KEY1 TO LEV(CAT-INDEX).
761 007610    IF CAT-INDEX IS GREATER THAN 1 SET NUMBER4 TO CAT-INDEX
762 007620    SUBTRACT 1 FROM NUMBER4
763 007630    MOVE D-NAME(CAT-INDEX) TO MATCH
764 007640    MOVE 1 TO LOOK
765 007650    SET CAT-INDEX TO 1
766 007660    PERFORM CAT-UNION NUMBER4 TIMES
767 007670    IF LOOK IS EQUAL TO 5
768 007680    MOVE SPACES TO D-NAME(CAT-INDEX)
769 007690    MOVE 00 TO LEV(CAT-INDEX)
770 007700    MOVE 1 TO LOOK
771 007710    SET CAT-INDEX DOWN BY 1.
772 007720    SET STACK-INDEX DOWN BY 1.
773 007730  POP-STACK.
774 007740  REGIN3.
775 007750  IF CAT-INDEX IS LESS THAN 1 GO TO ENDREGIN3.
776 007760  MOVE LFV(CAT-INDEX) TO KEY3.
777 007770  UNSTRING KEY3 INTO KEY4, KEY5.
778* 007780  MOVE KEY4 TO HOLD-LEVEL.
779* 007790  IF HOLD-LEVEL IS NOT LESS THAN KEY2
780* 007800  MOVE SPACES TO D-NAME(CAT-INDEX)
781* 007810  MOVE 00 TO LEVEL(CAT-INDEX)
782* 007820  SET CAT-INDEX DOWN BY 1
783* 007830  GO TO BEGIN3.
784* 007840
785* 007850  ENDBEGIN3.
786* 007860  EXIT.
787* 007870
788* 007880  CAT-UNION.
789* 007890  IF MATCH IS EQUAL TO D-NAME(CAT-INDEX) MOVE 5 TO LOOK.
790* 007900  SET CAT-INDEX UP BY 1.
791* 007910  SET THE DETERMINANT FLAG TO YES IF IDENTIFIED
792* 007920  SET-KEY.
793* 007930  SET ANINDEX UP BY 1.
794* 007940  IF DETERM-FLAG(ANINDEX) IS NOT EQUAL TO 3 AND
795* 007950  NAME(ANINDEX) IS EQUAL TO ATTR(STACK-INDEX) MOVE 1 TO
796* 007960  DETERM-FLAG(ANINDEX).
797* 007970  BUILD-F-D-TABLE.
798* 007980  MOVE "0" TO STAT-HOLD.
799* 007990  SET NUMBER4 TO CAT-INDEX.
800* 008000  SET NUMBER3 TO STACK-INDEX.
801* 008010  SUBTRACT L-S-COUNT FROM NUMBER3.
802* 008020  PERFORM DELETE THRU LETE-DE.
803* 008030  SUBTRACT L-S-COUNT FROM NUMBER3 GIVING HOLD-NO.
804* 008040  IF NUMBER4 IS GREATER THAN 1 AND
805* 008050  HOLD-NO IS NOT GREATER THAN 0 PERFORM SPEC-CASE,
806* 008060  IF LAST-LEV IS GREATER THAN 1 PERFORM SPEC-CASE2 THRU
807* 008070  CASE2END.
808* 008080  MOVE LAST-LEV TO SAVE-LEVEL.
809* 008090  SET F-D-INDEX UP BY 1.
810* 008100  MOVE NUMBER4 TO MOVERT.
811* 008110  MOVE MOVERT TO LIST(F-D-INDEX).
812* 008120  SET CAT-INDEX TO 1.
813* 008130  PERFORM LOAD1 NUMBER4 TIMES.
814* 008140  SET CAT-INDEX TO NUMBER4.
SET F-D-INDEX UP BY 1.

IF HOLD-NO IS NOT GREATER THAN 0 MOVE 1 TO HOLD-NO
ADD 1 TO HOW-MANY
MOVE HOW-MANY TO NUM-PART
STRING COLOR, NUM-PART DELIMITED BY SIZE INTO SPEC-SY
MOVE HOLD-NO TO MOVERT
MOVE MOVERT TO LIST(F-D-INDEX)
SET F-D-INDEX UP BY 1
MOVE SPEC-SY TO LIST(F-D-INDEX)
MOVE 0 TO HOLD-NO
SET FIND TO F-D-INDEX
ELSE
MOVE HOLD-NO TO MOVERT
MOVE MOVERT TO LIST(F-D-INDEX)
SET TOP-STACK TO STACK-INDEX
SET STACK-INDEX TO 1
PERFORM LOAD2 TOP-STACK TIMES
SET FIND TO F-D-INDEX.

SPECIAL CASE OPTION SELF IDENTIFYING DETERMINANT CHANGE ROUTINE

DISPLAY "SPECIAL SITUATION", LINE 5, POSITION 31, ERASE.
DISPLAY "THE FOLLOWING DATA-NAME-ITEMS HAVE BEEN", LINE 7,
POSITION 20.
DISPLAY "CONCATENATED TO FORM A DETERMINANT", THE", LINE 8,
POSITION 20.
DISPLAY "DETERMINANT AS STRUCTURED IS SELF-IDENTIFYING", LINE
9, POSITION 20.
MOVE 12 TO LINE-NO.
MOVE 0 TO DET-COUNT.
SET CAT-INDEX TO 1.
PERFORM DISP-DET NUMBER4 TIMES.
SET CAT-INDEX TO NUMBER4.
ADD 2 TO LINE-NO.
DISPLAY "IF THE DATA-NAME-ITEMS MARKED * UNIQUELY IDENTIFIES"
LINE LINE-NO, POSITION 20.
852# 098520 ADD 1 TO LINE-NO.
853# 098530 DISPLAY "ANY OF THE OTHER LISTED ITEMS **ENTER YES** IF NO--"
854# 098540 LINE LINE-NO, POSITION 20.
855# 098550 ADD 1 TO LINE-NO.
856# 098560 DISPLAY "THE RELATION IS ASSUMED CORRECT--PRESS NEWLINE.",
857# 098570 LINE LINE-NO, POSITION 20.
858# 098580 ACCEPT REPLY, POSITION 20.
859# 098590 IF REPLY IS EQUAL TO "Y" OR
860# 098600 REPLY IS EQUAL TO "YES" SET STACK-INDEX TO 0
861# 098610 PERFORM NEXT-Q
862# 098620 DISPLAY " ", ERASE.
863# 098630
864# 098640
865# 098650
866# 098660 EDIT OPTION TO ALLOW USER TO MOVE DETERMINANTS FROM LEFT
867# 098670 SIDE OF A RELATION TO THE RIGHT SIDE AS A DATA-NAME-ITEM.
868# 098680 SPEC-CASE2.
869# 098690 IF SAVL-LEVEL IS EQUAL TO LAST-LEV AND LAST-LEV IS NOT
870# 098700 EQUAL TO 1 MOVE "S" TO STAT-HOLD.
871# 098710 IF STAT-HOLD IS EQUAL TO "S" PERFORM SAV-MOVE THRU S-MEND
872# 098720 SET CAT-INDEX TO 0
873# 098730 MOVE 0 TO KOUNTY
874# 098740 MOVE NUMFR4 TO KNTTY
875# 098750 PERFORM NEW-UNION-MAIN NUMBER4 TIMES
876# 098760 MOVE KNTTY TO NUMBER4
877# 098770 SET CAT-INDEX TO 1.
878# 098780 IF NUMFR4 IS LESS THAN 2 SET S-INDEX TO 1
879# 098790 SET CAT-INDEX TO 1
880# 098800 PERFORM SAV-LAST
881# 098810 SET S-INDEX TO 1
882# 098820 SET CAT-INDEX TO 1.
883# 098830 MOVE 1 TO SAVETOP
884# 098840 GO TO CASE2END.
885# 098850 MOVE 12 TO LINE-NO.
886# 098860 MOVE 0 TO DET-COUNT.
887# 098870 SET CAT-INDEX TO 1.
888# 098880 DISPLAY "ARE ANY OF THE FOLLOWING DETERMINANTS UNIQUELY"
869* 00890
  LINE 8, POSITION 20,
  "IDENTIFIED BY THE DETERMINANTS MARKED WITH THE"
870* 00890
891* 00891
  LINE 9, POSITION 20,
  "Y", ENTER YES ** OTHERWISE PRESS NEWLINE."
892* 00892
893* 00893
  LINE 10, POSITION 20,
894* 00894
  PERFORM DISP-DET NUMBER4 TIMES.
895* 00895
  SET CAT-INDEX TO 1.
896* 00896
  ADD 2 TO LINE-NO.
897* 00897
  ACCEPT REPLY, LINE LINE-NO, POSITION 20.
898* 00898
  IF REPLY IS EQUAL TO "Y" OR
899* 00899
     REPLY IS EQUAL TO "YES"
900* 00900
  ELSE
901* 00901
  SET S-INDEX TO 1
902* 00902
  PERFORM SAV-LAST NUMBER4 TIMES
903* 00903
  SET S-INDEX DOWN BY 1
904* 00904
  SET SAVETOP TO S-INDEX
905* 00905
  PERFORM NEXT-GUES
906* 00906
907* 00907
  ELSE
908* 00908
  PERFORM SAV-LAST NUMBER4 TIMES
909* 00909
  SET S-INDEX DOWN BY 1
910* 00910
  SET CAT-INDEX DOWN BY 1
911* 00911
  SET SAVETOP TO S-INDEX
912* 00912
  DISPLAY " ", ERASE.
913* 00913
  CASE2END.
914* 00914
  EXIT.
915* 00915
916* 00916
917* 00917
  PACK-IT-D.
918* 00918
  MOVE 1 TO KNIT.
919* 00919
  ADD 1 TO NUMBER.
920* 00920
  PERFORM MOVE-DATER L-S-COUNT TIMES.
921* 00921
  MOVE KEY2 TO LAST-LEV.
922* 00922
  SET NUMBER4 TO CAT-INDEX.
923* 00923
  SUBTRACT L-S-COUNT FROM DET-COUNT.
924* 00924
  ADD DET-COUNT TO HOLD-NO.
925* 00925
0926  009260 PACK-IT-DOWN.
0927  009270  MOVE 1 TO KNIT.
0928  009280  ADD 1 TO NUMBER4.
0929  009290  PERFORM MOVE-DFT2 L-S-COUNT TIMES.
0930  009300  MOVE 1 TO LAST-LEV.
0931  009310  SET NUMBER4 TO CAT-INDEX.
0932  009320  SUBTRACT L-S-COUNT FROM DET-COUNT.
0933  009330  ADD DET-COUNT TO HOLD-NO.
0934  009340  DISPLAY THE CONCATENATED DETERMINANT.
0935  009350  DISP-DET.
0936  009360  DISPLAY D-NAME(CAT-INDEX) ; LINE LINE-NO; POSITION 35.
0937  009370  IF LEV(CAT-INDEX) IS EQUAL TO KEY1
0938  009380  DISPLAY "**", LINE LINE-NO; POSITION 33.
0939  009390  SET CAT-INDEX UP BY 1.
0940  009400  ADD 1 TO LINE-NO.
0941  009410  ADD 1 TO DET-COUNT.
0942  009420  BUILD THE NEW RELATION FROM THE CONCATENATED DETERMINANT.
0943  009430  R-NEW-RELATION.
0944  009440  SET CAT-INDEX TO 1.
0945  009450  SUBTRACT L-S-COUNT FROM NUMBER4.
0946  009460  PERFORM L-TO-R NUMBER4 TIMES.
0947  009470  R-NEW-POCMF.
0948  009480  EXIT.
0949  009490  L-TO-R.
0950  009500  SET STACK-INDEX UP BY 1.
0951  009510  MOVE D-NAME(CAT-INDEX) TO ATTH(STACK-INDEX).
0952  009520  SET CAT-INDEX UP BY 1.
0953  009530  LOAD THE LEFT SIDE DATA INTO THE FDTABLE.
0954  009540  LOAD.
0955  009550  SET F-D-INDEX UP BY 1.
0956  009560  MOVE D-NAME(CAT-INDEX) TO LIST(F-D-INDEX).
0957  009570  SET CAT-INDEX UP BY 1.
0958  009580  DELETE IDENTICAL NAME ITEMS FROM THE Stack LS ITEMS IN RT SIDE.
0959  009590  DF-LEFT.
0960  009600  PERFORM REMOVE L-S-COUNT TIMES.
0961  009610  LETE-RT.
FILL THE STACK WITH BLANKS.

SET TOP-STACK TO STACK-INDEX.

MOVE ATTR(STACK-INDEX) TO THIS-ATTR.

SET STACK-INDEX TO NUMBER3.

PERFORM CHECK-LIST NUMBER3 TIMES.

SET STACK-INDEX TO TOP-STACK.

MOVE SPACES TO ATTR(STACK-INDEX).

SET STACK-INDEX DOWN BY 1.

CHECK-LIST.

IF ATTR(STACK-INDEX) IS EQUAL TO THIS-ATTR

MOVE SPACES TO ATTR(STACK-INDEX).

SET STACK-INDEX DOWN BY 1.

SAV-MOVE.

SET S-INDEX TO 1.

IF SLEV(S-INDEX) IS NOT EQUAL TO 00

PERFORM TRANSFER-DET SAVETOP TIMES.

SET S-INDEX TO 1.

PERFORM SAVCLEAR SAVETOP TIMES.

MOVE 0 TO SAVETOP.

SET S-INDEX TO 1.

SET CAT-INDEX TO NUMBER4.

S-MEND.

EXIT.

TRANSFER-DET.

SET CAT-INDEX TO NUMBER4.

MOVE SLEV(S-INDEX) TO KEY3.

UNSTRING KEY3 INTO KEY4; KEY5.

MOVE KEY4 TO HOL-LEV.

IF HOL-LEV IS LESS THAN KEY2 PERFORM MAKE-ROOM NUMBER4 TIMES
1000* 010000  SET CAT-INDEX UP BY 1
1001* 010010  ADD 1 TO NUMBER4
1002* 010020  MOVE S-NAME(S-INDEX) TO D-NAME(CAT-INDEX)
1003* 010030  MOVE SLEV(S-INDEX) TO LEV(CAT-INDEX)
1004* 010040  MOVE SPACES TO S-NAME(S-INDEX)
1005* 010050  MOVE 60 TO SLEV(S-INDEX)
1006* 010060  SET S-INDEX UP BY 1
1007* 010070  ELSE SET S-INDEX UP BY 1.
1008* 010080  MAKE-ROOM.
1009* 010090  MOVE D-NAME(CAT-INDEX) TO ATTRIBUTE.
1010* 010100  MOVE LEV(CAT-INDEX) TO LEV-HOLD.
1011* 010110  SET CAT-INDEX UP BY 1.
1012* 010120  MOVE ATTRIBUTE TO D-NAME(CAT-INDEX).
1013* 010130  MOVE LEV-HOLD TO LEV(CAT-INDEX).
1014* 010140  SET CAT-INDEX DOWN BY 2.
1015* 010150  NEXT-QUEST.
1016* 010160  DISPLAY "ARE ALL UNMARKED DETERMINANTS IDENTIFIED BY"
1017* 010170  "YES or "NO," OR ELSE PERFORM SEL-ITEMS THRU SEL-END.
1018* 010180  POSITION 20.
1019* 010190  "THE MARKED DETERMINANTS?"; POSITION 20,
1020* 010200  "ENTER YES ** OTHERWISE PRESS NEWLINE."
1021* 010210  "YES; OTHERWISE PRESS NEWLINE."
1022* 010220  "ARE ALL UNMARKED DETERMINANTS IDENTIFIED BY"
1023* 010230  "YES or "NO," OR ELSE PERFORM SEL-ITEMS THRU SEL-END.
1024* 010240  ACCEPT REPLY, POSITION 20.
1025* 010250  IF REPLY IS EQUAL TO "YES" OR
1026* 010260  "YES; OTHERWISE PRESS NEWLINE."
1027* 010270  "ARE ALL UNMARKED DETERMINANTS IDENTIFIED BY"
1028* 010280  "YES or "NO," OR ELSE PERFORM SEL-ITEMS THRU SEL-END.
1029* 010290  DISPLAY "YES; OTHERWISE PRESS NEWLINE."
1030* 010300  "ARE ALL UNMARKED DETERMINANTS IDENTIFIED BY"
1031* 010310  "YES or "NO," OR ELSE PERFORM SEL-ITEMS THRU SEL-END.
1032* 010320  DISPLAY "NO; OTHERWISE PRESS NEWLINE."
1033* 010330  "ARE ALL UNMARKED DETERMINANTS IDENTIFIED BY"
1034* 010340  "YES or "NO," OR ELSE PERFORM SEL-ITEMS THRU SEL-END.
1035* 010350  SAV-LAST.
1036* 010360  MOVE D-NAME(CAT-INDEX) TO S-NAME(S-INDEX).
1037* 010370  MOVE LEV(CAT-INDEX) TO SLEV(S-INDEX).
1037  010370  SET CAT-INDEX UP BY 1.
1038  010380  SET S-INDEX UP BY 1.
1039  010390
1040  010400
1041  010410
1042  010420 NEXT-Q.
1043  010430  DISPLAY "ARE ALL UNMARKED DATA-NAME-ITEMS IDENTIFIED"
1044  010440  POSITION 20.
1045  010450  "ARE THE MARKED DATA-NAME-ITEMS?", POSITION 20.
1046  010460  "ENTER YES ** OTHERWISE PRESS NEWLINE.", POSITION 20.
1047  010470  ACCEPT REPLY, POSITION 20.
1048  010480  IF REPLY IS EQUAL TO "Y" OR
1049  010490  "REPLY IS EQUAL TO "YE" OR
1050  010500  "REPLY IS EQUAL TO "YES" DISPLAY " ", ERASE
1051  010510  PERFORM R-NEW-RELATION THRU R-NDONE.
1052  010520  PERFORM PACK-IT-DOWN.
1053  010530  ELSE MOVE 1 TO LAST-LEV.
1054  010540  PERFORM SEL-ITEMS THRU SEL-END.
1055  010550  DISPLAY " ", ERASE.
1056  010560
1057  010570
1058  010580
1059  010590 MOVE-DETER.
1060  010600  SET CAT-INDEX TO NUMBER4.
1061  010610  MOVE D-NAME(CAT-INDEX) TO ATTRIBUTE.
1062  010620  MOVE LEV(CAT-INDEX) TO LEV-HOLD.
1063  010630  MOVE SPACES TO D-NAME(CAT-INDEX).
1064  010640  MOVE 00 TO LEV(CAT-INDEX).
1065  010650  SET CAT-INDEX TO KNTT.
1066  010660  MOVE ATTRIBUTE TO D-NAME(CAT-INDEX).
1067  010670  MOVE LEV-HOLD TO LEV(CAT-INDEX).
1068  010680  AND 1 TO NUMBER4.
1069  010690  ADD 1 TO KNTT.
1070  010700
1071  010710 MOVE-DETT.
1072  010720  SET CAT-INDEX TO NUMBER4.
1073  010730  MOVE D-NAME(CAT-INDEX) TO ATTRIBUTE.
1074 U10740 move spaces to d-name(cat-index).
1075 U10750 move 00 to lev(cat-index).
1076 U10760 st cat-index to knit.
1077 U10770 move attribute to d-name(cat-index).
1078 U10780 move 11 to lev(cat-index).
1079 U10790 add 1 to number4.
1080 U10800 add 1 to knit.
1081 U10810
1082 U10820
1083 U10830 sel-items.
1084 U10840 display "list those data-name-items which are identified."
1085 U10850 position 20.
1086 U10860 display "enter xx to signal no more entries to be made."
1087 U10870 position 20.
1088 U10880 sel-next.
1089 U10890 accept reply: position 20.
1090 U10900 if reply is equal to "xx" or
1091 U10910 reply is equal to "xxx" go to sel-end.
1092 U10920 reply is equal to "xxx" go to sel-end.
1093 U10930 perform move-det-pt thru m-d-end.
1094 U10940 go to sel-next.
1095 U10950 sel-end.
1096 U10960 exit.
1097 U10970
1098 U10980
1099 U10990 move-det-pt.
1100 U11000 move 4 to look.
1101 U11010 set cat-index to 1.
1102 U11020 perform search-stack number4 times.
1103 U11030 if look is equal to 4
1104 U11040 display "you spelled it wrong**try again**please."
1105 U11050 position 20.
1106 U11060 "press newline": position 20
1107 U11070 accept reply
1108 U11080 if reply is not equal to "usheroief" display ""
1109 U11090 else next sentence
1110 U11100 else set cat-index to 1
1111*  u11110  MOVE 1 TO KNIT
1112*  u11120  ADD 3; L-S-COUNT TO NUMBER4
1113*  u11130  PERFORM COMPACT THRU PACK-IT NUMBER4 TIMES
1114*  u11140  SUBTRACT 4* L-S-COUNT FROM NUMBER4
1115*  u11150  ADD 1 TO HOLD-NO
1116*  u11160  SET CAT-INDEX DOWN BY 1.
1117*  u11170  M-O-REND.
1118*  u11180  EXIT.
1119*  u11190
1120*  u11200
1121*  u11210  SEARCH-STACK.
1122*  u11220  IF REPLY IS EQUAL TO D-NAME(CAT-INDEX)
1123*  u11230  MOVE 1 TO LOCK
1124*  u11240  SET STACK-INDEX UP BY 1
1125*  u11250  MOVE REPLY TO ATTR STACK-INDEX)
1126*  u11260  MOVE SPACES TO D-NAME(CAT-INDEX)
1127*  u11270  MOVE 00 TO LEV(CAT-INDEX)
1128*  u11280  SET CAT-INDEX UP BY 1
1129*  u11290  ELSE SET CAT-INDEX UP BY 1.
1130*  u11300
1131*  u11310  NEW-UNION-MAIN.
1132*  u11320  SET CAT-INDEX UP BY 1.
1133*  u11330  MOVE D-NAME(CAT-INDEX) TO MATCH.
1134*  u11340  MOVE 1 TO LOCK.
1135*  u11350  SUBTRACT 1 FROM KNITTY GIVING THAT-MANY.
1136*  u11360  PERFORM NEW-UNION THAT-MANY TIMES.
1137*  u11370  IF LOCK IS EQUAL TO 3
1138*  u11380  SET CAT-INDEX TO KNIT
1139*  u11390  MOVE SPACES TO D-NAME(CAT-INDEX)
1140*  u11400  MOVE 00 TO LEV(CAT-INDEX)
1141*  u11410  ADD 2 TO KNITTY
1142*  u11420  PERFORM COMPACT THRU PACK-IT KNITTY TIMES
1143*  u11430  SUBTRACT 3 FROM KNITTY
1144*  u11440  ADD 1 TO KOUNTY
1145*  u11450  SET CAT-INDEX TO KOUNTY
1146*  u11460  ELSE SET CAT-INDEX DOWN BY 1.
NEW-UNION.

SET CAT-INDEX UP BY 1.

IF MATCH IS EQUAL TO D-NAME(CAT-INDEX) AND MATCH IS NOT EQUAL TO SPACES MOVE 3 TO LOOK

SET KNIT TO CAT-INDEX.

COMPACT.

IF D-NAME(CAT-INDFX) IS NOT EQUAL TO SPACES

MOVE D-NAME(CAT-INDEX) TO ATTRIBUTE

MOVE LEV(CAT-INDEX) TO LEV-HOLD

MOVE SPACES TO D-NAME(CAT-INDEX)

MOVE 00 TO LEV(CAT-INDEX)

SET CAT-INDEX TO KNIT

MOVE ATTRIBITE TO D-NAME(CAT-INDEX)

MOVE LEV-HOLD TO LEV(CAT-INDEX)

ADD 1 TO KNIT

SET CAT-INDEX UP BY 1

ELSE SET CAT-INDEX UP BY 1.

PACK-IT.

EXIT.

LOAD THE RT SIDE DATA ITEMS INTO THE FDTABLE AND BLANK THE STACK.

LOAD2.

IF ATTR(STACK-INDEX) IS NOT EQUAL TO " "

SET F-D-INDEX UP BY 1

MOVE ATTR(STACK-INDEX) TO LIST(F-D-INDEX)

MOVE SPACES TO ATTR(STACK-INDEX)

SET STACK-INDEX UP BY 1

ELSE SET STACK-INDEX UP BY 1.

END OF BUILD-F-D-TABLE

OUTPUT THE F-D-TABLE AS L-S --> R-S.
018550         118550 PRINT-F-D-TABLE.
018560         118560 SET F-D-INDEX TO 1.
018570         118570 MOVE SPACES TO THIS-ATTR.
018580         118580 PERFORM GEMOUT UNTIL THIS-ATTR IS EQUAL TO "XXXXXXXXXX".
018590         118590 THATS-ALL.
018600         118600 EXIT.
018610         118610 GEMOUT.
018620         118620 MOVE SPACES TO PRINT-LINE.
018630         118630 MOVE LIST(F-D-INDEX) TO LEFT-SIDE.
018640         118640 MOVE 1 TO CHAR-COUNT.
018650         118650 PERFORM BUILD-L-S LEFT-SIDE TIMES.
018660         118660 SET F-D-INDEX UP BY 1.
018670         118670 STRING " ** ARROW ** " DELIMITED BY SIZE INTO PRINT-LINE
018680         118680 WITH POINTER CHAR-COUNT.
018690         118690 MOVE LIST(F-D-INDEX) TO RIGHT-SIDE.
018700         118700 MOVE RIGHT-SIDE TO NUMBERS.
018710         118710 ADD LEFT-SIDE + 1 TO NUMBERS.
018720         118720 IF NUMBERS IS GREATER THAN 12
018730         118730 SUBTRACT LEFT-SIDE, 1 FROM 12 GIVING NUMBER4
018740         118740 SUBTRACT NUMBER4 FROM RIGHT-SIDE
018750         118750 MOVE RIGHT-SIDE TO NUMBERS5
018760         118760 MOVE NUMBER4 TO RIGHT-SIDE
018770         118770 PERFORM BUILD-R-S RIGHT-SIDE TIMES
018780         118780 WRITE PRINT-LINE
018790         118790 PERFORM MORE-LINES
018800         118800 ELSE PERFORM BUILD-R-S RIGHT-SIDE TIMES
018810         118810 WRITE PRINT-LINE.
018820         118820 SET F-D-INDEX UP BY 1.
018830         118830 MOVE LIST(F-D-INDEX) TO THIS-ATTR.
018840         118840 SET THAT-MANY TO F-D-INDEX.
018850         118850 IF THAT-MANY IS GREATER THAN FIND MOVE "XXXXXXXXXX" TO
018860         118860 THIS-ATTR.
018870         118870 MORE-LINES.
018880         118880 IF NUMBERS IS LESS THAN 12
1222* u12220    MOVE SPACES TO PRINT-LINE
1223* u12230    MOVE 1 TO CHAR-COUNT
1224* u12240    PERFORM BUILD-R-S NUMBERS TIMES
1225* u12250    WRITE PRINT-LINE
1226* u12260    ELSE DIVIDE 12 INTO NUMBERS GIVING THAT-MANY
1227* u12270    REMAINDER NUMBER4
1228* u12280    MOVE 12 TO RIGHT-SIDE
1229* u12290    PERFORM PRINT-IT THAT-MANY TIMES
1230* u12300    MOVE 1 TO CHAR-COUNT
1231* u12310    MOVE SPACES TO PRINT-LINE
1232* u12320    PERFORM BUILD-R-S NUMBER4 TIMES
1233* u12330    WRITE PRINT-LINE.
1234* u12340    BUILD-L-S.  
1235* u12350    SET F-D-INDEX UP BY 1.
1236* u12360    STRING LIST(F-D-INDEX), BREAK DELIMITED BY SIZE INTO
1237* u12370    PRINT-LINE WITH POINTER CHAR-COUNT.  
1238* u12380
1239* u12390    PRINT-IT.  
1240* u12400    MOVE 1 TO CHAR-COUNT.  
1241* u12410    MOVE SPACES TO PRINT-LINE.  
1242* u12420    PERFORM BUILD-R-S RIGHT-SIDE TIMES.  
1243* u12430    WRITE PRINT-LINE.  
1244* u12440
1245* u12450    BUILD-R-S.  
1246* u12460    SET F-D-INDEX UP BY 1.  
1247* u12470    STRING LIST(F-D-INDEX), BREAK DELIMITED BY SIZE INTO
1248* u12480    PRINT-LINE WITH POINTER CHAR-COUNT.
The Interactive Generation of Functional Dependencies

by

WILLIAM CLEM HUNT

B. G. S., University of Nebraska at Omaha
Omaha, Nebraska, 1972

------------------
AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirement for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1978
Information is a prime commodity in today's diverse world. Its availability is enhanced through the use of Data Base Management Systems (DBMS) which control the automatic storage and retrieval of data. These systems allow users to view data independently of its physical storage configuration.

A type DBMS based on the Relational Data Model allows a user to view data in logical groups. It makes possible the elimination of the representation-dependency from the user's interface, thus simplifying the user's interface requirements. A prototype system to automatically generate Third Normal Form Relations (3NF) has been developed. The system developed produces 3NF Relations in three phases: (1) produces a hierarchical representation of the input data; (2) generates a set of Functional Dependencies; and (3) synthesizes the Functional Dependencies into 3NF Relations.

The report 'The Interactive Generation of Functional Dependencies', describes in detail the algorithm for the interactive generation of Functional Dependencies. The design, implementation, and testing of the algorithm are presented in detail, including a comprehensive walk-through of the tested program. The abstract code and MCR level II COBOL code are provided in appendices. Enhancements to the system which would expand and simplify the user's interactions with the system are discussed.