

A USER-ORIENTED TRANSACTION DEFINITION FACILITY
FOR A RELATIONAL DATABASE SYSTEM

by

C. STEVEN ROUSH

B. S., Kansas State University, Manhattan, Kansas, 1972

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements of the degree

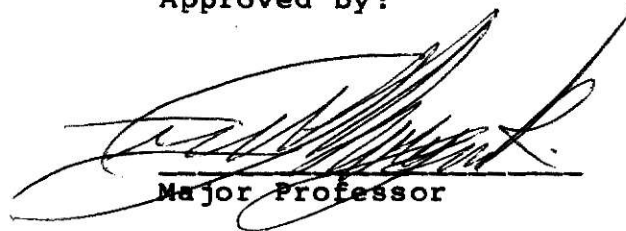
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1979

Approved by:



Major Professor

ACKNOWLEDGEMENTS

I want to thank Dr. Fred Maryanski for keeping my work on track and NCR, Inc. for funding this project.

Spec. Coll.

LD

2668

.R4

1979

R58

C.2

TABLE OF CONTENTS

1	INTRODUCTION.	1
2	THE DESIGN.	6
3	IMPLEMENTATION AND TESTING.	17
4	EVALUATION AND CONCLUSIONS.	32
REFERENCES.		37
BIBLIOGRAPHY.		38
APPENDICIES		
SYNTAX OF THE OUTPUT OF TDF		39
EXAMPLES OF INTERACTION WITH TDF		
	AND THE RESULTING OUTPUT.	41
A LISTING OF TDF.		47
LIST OF FIGURES		
A TRANSACTION PRESENTED AS A HEIRARCHY.		4
SNAP SYSTEM FLOWCHART		5
A DATABASE OF TWO RELATIONS		11
THE DATA STRUCTURES		20
THE PROGRAM STRUCTURE		25

1. INTRODUCTION

Software has become the major obstacle preventing users from reaching their automation goals. According to Boehm's projections, software now accounts for approximately 80% of every dollar invested in computing, or about 20 billion dollars annually[1]. Further, both Boehm[2] and Myers[3] quote projected software growth figures of 15 to 25% annually. It is not clear how these growth rates can be attained.

Myers states that productivity increases will only average 3 to 7% during the period from 1955 to 1985[4], and Boehm estimates that during the ten years before 1985 improved productivity and increased numbers of programmers together can only account for a software growth rate of 12 to 17%[5]. Mills, however makes these estimates seem optimistic. He notes that software maintenance already requires about 75% of the total software effort, and that "unless radical methods are found, maintenance will go even higher in its demands and will very nearly stifle development"[6].

There are other problems with current software production techniques which must also be noted. Software has a reputation for being delivered late and being unreliable when it is delivered[7]. Finally, because current programmer productivity appears to be between 600 and 6000 instructions per man-year[8], it is very difficult for software systems to keep pace with the changing needs of

the users.

For these reasons, a large effort is being made to help reduce the problems of software development. Structured design, structured programming, new language design, program proving, project management techniques and database management systems have all evolved with the intention of improving the quality of software. This report will discuss work done in the database management area that may eventually help reduce the effort required to create and maintain a body of software, thus helping to enhance the utility of the computer.

Specifically, the report discusses the design and implementation of a subsystem of a user-oriented database management system which is under development. The goal of this prototype system, which is named SNAP, is to allow transactions, both retrieval and maintenance, to be defined by people who are not computer professionals, and entire systems of transactions to be defined by these same people with professional assistance required only during the process of identifying the interrelationships between the data items in the system. Though far from complete, the prototype database system does currently allow a trained professional to define transactions which would probably satisfy many of the user requirements to retrieve (display) their data.

The subsystem described by this report is named the Transaction Definition Facility (TDF). Its goal is to

interact with the user to define his transactions in terms which do not require knowledge of the internal structure of the database (the schema), and to produce programs to actually perform those transactions.

SNAP's design has been based on two assumptions. The first is that users of a relational database in third normal form (3NF) require less knowledge of the database schema to successfully perform transactions than users of databases which have not been so rigorously defined. Instead, the database management system can rely on the axioms which are known to apply to relations in 3NF to supplement a user's commands. The second assumption is that a user's database transactions are heirarchical in nature. That is, that transactions are composed of 'lines' which represent various levels of detail or nesting, but those levels of detail never partially overlap. This is analagous to the concept of blocks in a block structured language. Figure 1. presents an example of a three level transaction.

Figure 1. A transaction presented as a heirarchy.

```

For each division in the company,
  list its name
      phone-no
      mail-location
and
For each department in that division,
  list its name
      phone-no
      mail-location
and
For each employee in that department,
  list his/her name
      salary
      job-title

```

From these assumptions a technique has been derived to create a schema in 3NF without requiring knowledge of database systems, except when identifying the relationships between data items. The process involves putting the data items involved in each transaction into a hierarchy, with each line in the transaction being a node, and identifying the keys or determinants of each line. From this information, the system can develop the 3NF schema.

The schema creation system has been previously implemented[9] and produces input to the Transaction Definition Facility. Figure 2 presents the system flowchart for the present SNAP system, including TDF.

The following chapters will discuss the design, implementation, and testing of TDF; will present several examples of its output; and will offer conclusions about the use of the system and possible future work.

**THIS BOOK
CONTAINS
NUMEROUS PAGES
WITH DIAGRAMS
THAT ARE CROOKED
COMPARED TO THE
REST OF THE
INFORMATION ON
THE PAGE.**

**THIS IS AS
RECEIVED FROM
CUSTOMER.**

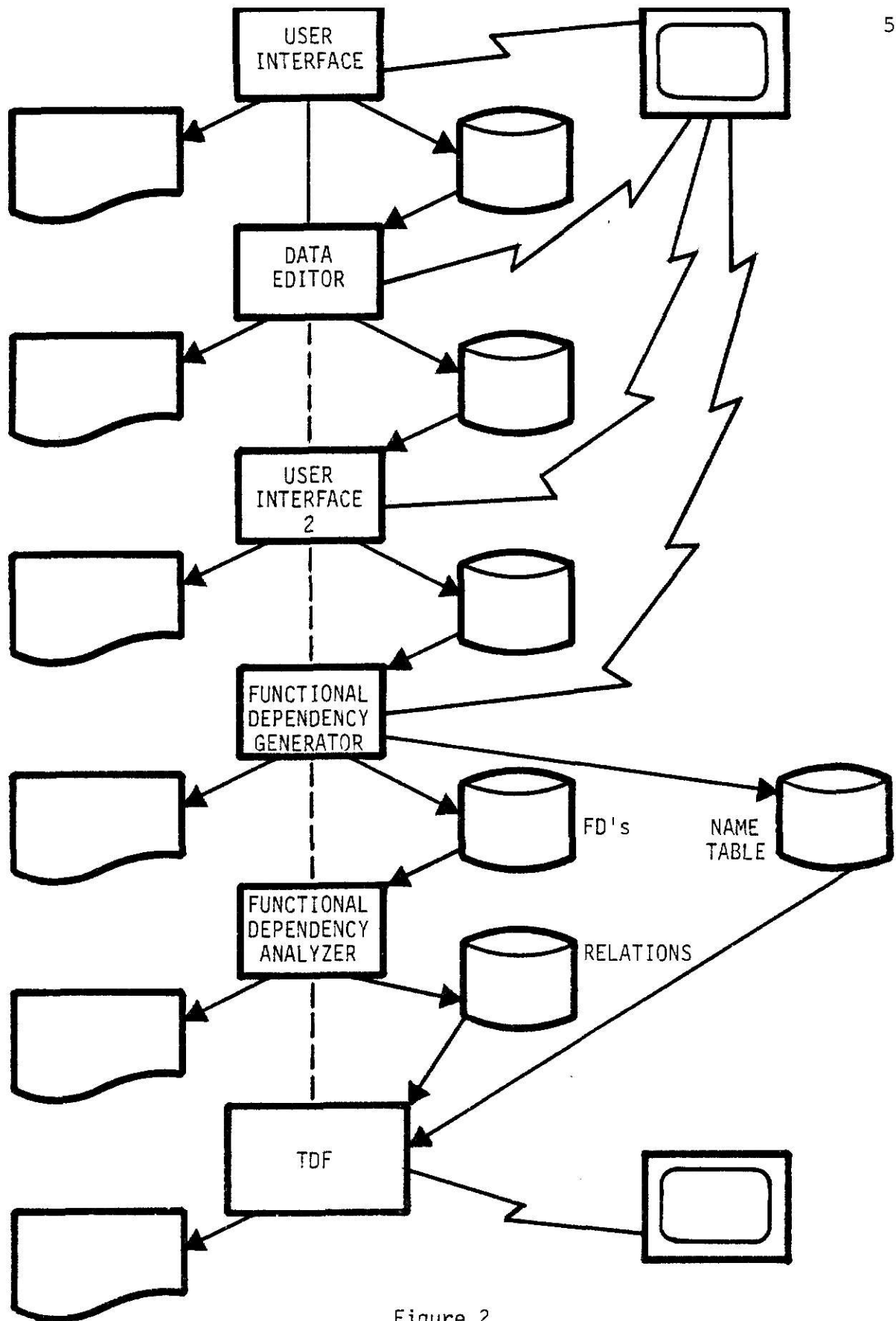


Figure 2

SNAP System Flowchart

2. THE DESIGN

The design phase of any programming project is usually the most important to the project's success. This chapter will discuss the techniques used to design TDF and will present the results of this design.

The goal of the Transaction Definition Facility is to allow the user to define his transactions without help from computing professionals or knowledge of a computer language. The problems involved in attaining this goal were split informally into two groups: those which primarily involved human engineering questions and those which required the development of algorithms. It was agreed that this implementation would not address the human engineering problems but would concentrate on trying to develop solutions for the second class of problems.

It was also agreed that the magnitude of these problems would inhibit the investigation of all the problems in the time available. Therefore, it was decided to try to design and implement the system in stages.

There was no specification of output, input, or internal processes by the Grantors or the project manager, as long as the previously discussed goals and restrictions were met. Therefore, one of the first tasks was to decide on a model for the output. The model had to be complete, it had to cover most or all of the transactions possible. This resulted in a decision to base the output on an existing

relational language, instead of trying to design and prove a new language. This decision was enforced by the ease of creating correct syntax for an existing language. SEQUEL 2 was chosen for the model because its syntax and semantics were well suited to the needs of the system[10].

The next step was to perform a selective design. This consisted of a top-down design which was not complete in either breadth or depth. Instead, a pruning technique was used to reduce design time. The first design level defined four mutually exclusive functions which could apply to a transaction. They are the basic database functions add, delete, update, and retrieve. It was decided to pursue only the retrieval function, because it seemed to present the most problems in the area of database navigation, and it was felt that most of the problems encountered in designing the other three functions would also apply to retrieval. It must be noted that for this purpose, retrieval does not include formatting the retrieved data. It was decided that the problems of formatting could be handled after the retrieval was defined and were of a human engineering nature.

Using the SEQUEL model, retrieval was further divided into five non-exclusive processes which would be performed on each line. Collectively they would produce one SEQUEL statement for the line. These five are defined as follows:

- * Identification of data elements to be displayed
- * Identification of formulae to derive temporary items
- * Identification of sorting information
- * Identification of data selection conditions
- * Determination as to whether duplicate tuples should be displayed

While these processes are sufficient to define SEQUEL statements, additional processes are needed to meet the goal of allowing users to define their own transactions. As stated earlier, the primary method of reaching this goal has been to allow the user to define transactions without having knowledge of the schema. All specification of transactions is done without identifying the relations involved. The following processes were defined to allow this capability:

- * Identification of a primary relation
- * Identification of navigation paths
- * Identification of source relations of all elements involved

The next step in the design process was to specify the desired coverage for TDF and to identify a subset of SEQUEL sufficient to attain that coverage. The two are dependent on each other and on the set of concepts to be presented to the user as his tools to define transactions. Appendix 1

presents the syntax of the output of TDF, which is not a proper subset of SEQUEL as some modifications have been made.

While some concepts, like sorting and identifying displayed elements, are easy to understand and to present to the user, two concepts were identified which do not seem to fit easily into the TDF/SNAP design. They are:

Navigation paths based on general joins

If navigation paths can be assumed to stem only from natural joins, given one operand, the other operand and the operator can be identified without user interaction. Otherwise, the user would have to specify both operands and the operator, which would seem to conflict with the goals of the system. It appears that few real-life retrievals are affected by this assumption.

Relationships between tuples of the same relation.

The original version of the existing system could not handle intra-relation joins, but attempts are being made to correct this. These relationships are handled as special cases by SEQUEL, but are even harder to deal with if the user is unable to explicitly discuss entity-attribute relationships. Unfortunately, this situation is common in real-life: managers manage managers, and parts are made from other parts.

IDENTIFICATION OF A PRIMARY RELATION

To facilitate identifying navigation paths and the potential of creating duplicate tuples, the concept of a primary relation was developed. It is based on the belief that every element in a line could be covered or 'reached' by starting with the keys to a single 'primary' relation. This can be equated to the idea of every tree having a single root. The process of identifying the primary relation requires no interaction with the user, but relies entirely on the user's earlier interaction with the Functional Dependency Generator(FDGEN), the relations derived from this interaction, and the axioms that apply to relations in 3NF. During testing of the TDF it was discovered that some situations could actually require multiple primary relations. This does not seem to limit the potential usefulness of SNAP or TDF because the primary relation concept is not essential to TDF but merely facilitates other processes, additionally it appears that the primary relation algorithm can be easily modified to deal with multiple primary relations. Unfortunately, time did not allow these changes to be made to the system.

IDENTIFICATION OF DATA ELEMENTS TO BE DISPLAYED

Because the original lines of each transaction included data elements which were not to be displayed,

but were involved in computations, data selection, or navigation; it is necessary to identify those that are to be displayed. This is done by listing all of the elements involved in the line and asking the user to identify those which are to be displayed.

IDENTIFICATION OF NAVIGATION PATHS

A navigation path is a relationship between data elements in two tuples, of the same or different relations, which can be represented by a join operation. For example, DEPT-NAME would be the basis for a navigation path between the 'Department' relation and the 'Employee' relation given in Figure 3.

Figure 3. A database of two relations.

Department:

<u>DEPT-NAME</u>	<u>PHONE-NO</u>	<u>DIVISION-NAME</u>
Personnel	2331	General Sys.
Finance	7105	Administration
Data Pro.	4655	General Sys.

Employee:

<u>EMP-NUMBER</u>	<u>EMP-NAME</u>	<u>DEPT-NAME</u>
862468	Codd	Finance
594786	Holmes	Personnel
121342	Smith	Finance

The process of identifying navigation paths appears to be one of the keys to creating a truly user-oriented system. In keeping with the original goal of insulating the user from knowledge of the schema, it was decided to try to identify navigation paths internally, without user involvement, except for cases where multiple paths between two relations were possible. In those situations, correct paths would be identified by the user in terms of the data elements involved, instead of the relations.

There are two basic relationship types which can require a navigation path to be established.

1. If the data elements listed in the line are not all in the same tuple, paths must be created to link the relations containing the elements. This includes elements to be displayed, elements involved in calculations and elements involved in data selection.
2. If a line is not at the highest level of the report, there is a relationship between the line and its most recent predecessor at a higher level which must be indicated thru a navigation path.

IDENTIFICATION OF DERIVATION OF TEMPORARY (DERIVED) ITEMS

Data items which are not to be stored in the database are identified in one of the existing programs in the system,

FDGEN, and this information is passed on to TDF. Because defining a method of specifying calculations is felt to be a human engineering problem, TDF merely asks the user to specify those data elements which are involved in the calculation, without specifying any operations.

Two types of temporary data were identified, and they have to be treated slightly differently. One is the common calculation of the form:

A := function (B, C, D, ...), where each element is
atomic

At present, TDF will handle this type by requesting that each item needed to derive a given temporary element be identified.

The second case will be described informally as grouping-type computations. Totals, averages, maximums, and counts all fit in this class. There are two types of data items involved in computing a grouping-type data item:

- 1) The item(s) to grouped (totaled, counted, etc)
- 2) The item(s) needed to specify when to 'break' the calculation

There are two problems related to defining grouping-type elements. The first is the human engineering problem of asking the user to accept this type of calculation and to specify each item to be grouped and it's 'break' or 'control' items. The second stems from the fact that the elements being grouped do

not occur at the same level in the hierarchy as the temporary item which they are used to compute. Though this problem is recognized, and changes are being made to correctly handle it throughout the system, it is not correctly handled by TDF.

IDENTIFICATION OF SORTING INFORMATION

Sorting information is gathered by asking the user if the elements actually displayed should be presented in a sorted order. If the answer is affirmative, these elements are listed and the user is asked to indicate sort keys in major to minor order, noting whether in ascending or descending order.

IDENTIFICATION OF DATA SELECTION CONDITIONS

To allow users to define both queries and reports, it must be possible to add data selection criteria to the transaction. To do this, the user is asked if the system should retrieve every instance of the elements listed, or if it should first apply additional data selection criteria. If additional criteria are requested, they must be of the same form as conditions or booleans of conventional languages. There were two problems in this process, the first is the human engineering problem of specifying compound conditions in an easy to understand

manner. Currently TDF requires all conditions in a compound condition to be AND'ed.

The second problem involves specifying the source of the elements in the selection criteria. While one operand of the condition must involve element(s) in the line, the other operand may come from anywhere, and may change value at any time. In other words, the second operand may come from a terminal, a temporary or permanent relation, or its value may be a constant and it may vary with arbitrary frequency. Though this is an important issue, it is not central to the prototype system. Therefore in the prototype the second operand is to always come from a terminal, and at the frequency of once per execution of that SEQUEL statement.

IDENTIFICATION OF WHETHER DUPLICATE TUPLES SHOULD BE DISPLAYED

If the displayed items do not include all the keys of the primary relation, it is possible to display duplicate tuples. Therefore, the system asks the user if duplicates should be displayed, or only unique tuples.

CONCLUSION

As currently designed, TDF allows a user to define a significant subset of possible retrieval transactions

without ever having to reference any schema-dependent information. Five simple interactive processes are sufficient to perform the transaction definition, as long as the database is in 3NF. Additionally, there are no known problems which would prevent the design from being extended to cover the maintenance functions, in addition to retrieval.

TDF was not designed using the design methodologies which are currently popular. Though a top-down process was used, it was not completed. Instead, the implementation effort was begun as soon as the basic processes were identified. There are several reasons for this:

- * This was a research project, its goals were open-ended.
- * There were inflexible time constraints
- * The implementor is biased against long design efforts if the goal is not well defined

Most of the problems with the design were in the area of finding a primary relation. Though the problems involving primary relations appear to be understood and partially corrected, there is no proof of the concept.

3. IMPLEMENTATION AND TESTING

It is hard to put faith in untested ideas. Similarly, it is hard to put faith in untested designs. Because the design of TDF could not rely on previous work; design, implementation, and testing were performed iteratively, in an attempt to guide the design effort. This chapter will discuss both implementation and testing of TDF.

ENVIRONMENT

TDF was implemented in COBOL on an NCR 8230 computer, under the IMOS III operating system. This version of COBOL is a large subset of the full language with extensions for string and boolean handling[11]. Because both of these extensions were used, it would not be a simple task to port this program to another machine, though it probably could be done without too much effort if the second machine's version of COBOL allowed use of subroutines in some language which could provide the boolean and string facilities.

The program is approximately 1850 lines long and was implemented in five weeks. This effort was aided by the following environmental factors:

- * the hardware was almost never down
- * there was little contention for the machine by other users
- * the compiler and operating system were reliable, and unchanging

- * the on-line environment promoted quick detection and correction of errors
- * the author was experienced in COBOL

while these factors hindered it:

- * several compiler errors were encountered
- * the IMOS editor seemed awkward to use
- * IMOS lacked desired facilities (eg. no parameterization of control strings)
- * the COBOL compiler was deficient in several areas (eg. no cross reference)
- * COBOL is unsuited to the application

THE IMPLEMENTATION TECHNIQUE

Though the process of staged design did cause problems and probably more than 500 lines of working code had to be discarded or altered to meet changing needs, the implementation effort did benefit from this technique. Testing began during the first week and continued in parallel with design and coding. This allowed code to be tested while still 'fresh' and debugging code to be installed incrementally. Most of this debugging code was left in the program and can be turned on or off by the user.

Additionally, the testing process kept the design process on track by early identification of problems and guidance as to solutions. Finally, it not only kept the

implementor appraised as to the exact state of the project, but also provided quick gratification and therefore encouragement.

THE PROGRAM

THE DATA STRUCTURES

The original set of data structures seemed to suit the original design quite well. However, as soon as the design was extended to handle temporary data items, this was no longer true. Existing data structures were modified and new ones created, but the result is unfortunately much less manageable or comprehensible. Much of the problem was caused by the iterative design/implementation technique, but COBOL's limitations in this area must also be recognized. A third factor was the undue concern for efficiency by the implementor which resulted in such design-dependent data structures.

The following data structures which are important to the implementation are presented in Figure 4.

DATA-DICTIONARY

The purpose of the data dictionary is to link data items to relations. It contains a entry for each data

item in the data base and is built from the relation file. Each entry contains:

- * The data item's name
- * A bit string indicating the relations to which it is key
- * A bit string indicating the relations within which it found as a non-key

TEMPORARY TABLE

The temporary table is used to store the names of all temporary (derived) data items in the transaction. It is used with the data dictionary to guarantee that unidentified data items cannot be referenced.

RELATION-TABLE

The purpose of the relation table is to link relations to their data items. This is accomplished by linking together several data structures. The relation table contains one entry per relation. Each entry contains a pointer to a string of pointers in the relation pool, which in turn point to the data dictionary. Each entry also contains:

- * A count of items in the key
- * A count of items not in the key
- * Several temporary fields which are used by other processes for line-dependent information

RELATION-POOL

The purpose of the relation pool is to finish the link between relations and their constituent data items. Each relation is represented by a string of entries within the pool, one entry per data item. Entries for a relation's keys precede the entries for its non-keys. Each entry contains a pointer to the corresponding data dictionary entry.

LINE-TABLE

The line table is very similar to the relation table, except each entry represents one line in the current transaction. Each entry points to the transaction pool which represents all data items in a transaction. Additionally, it contains:

- * A level number, indicating the line's level in the transaction
- * An occurrence number, indicating the occurrence of that level
- * A count of candidate keys of the line

TRANSACTION-POOL

The transaction pool is similar to the relation pool. It has one entry per data item in each line in the transaction. Each entry contains:

- * A number which indicates if the item is a possible determinant of the line, or if the item is a

temporary item

- * A pointer to a data dictionary entry or an entry in the temporary table, depending on whether it is a temporary item or not

LINE-ELEMENT-TABLE

The line item table is the central data structure in the actual definition of the transaction at the line level. As each step of the process is performed, the system gathers more information about the line. This information is then transformed into the output of the system, in the form of executable commands. There is a table entry for each item in the data base which is necessary to create the line. Each entry contains:

- * A pointer to the item's name, in either the data dictionary or the temporary table
- * A flag which indicates if the data item is a determinant of the line
- * A flag to indicate data items added to the line by the system
- * A flag to indicate data items to be displayed
- * A flag to indicate data items to be retrieved
- * A flag to indicate temporary data items
- * The identifier of the data item's source relation
- * The identifier of any second relation which is involved in a natural join, based on this data item

THE PROCESSES

The interactive processes identified in the design are basically quite simple in function and in implementation. Each presents one or two questions and stores the answers, with no manipulation of the answer of any consequence. The non-interactive processes are much more complex. These processes are really intended to replace human interaction by deriving the answers to schema-related questions from other sources. In some circumstances, this requires extensive work, whether it is being performed by a human or the machine.

Figure 5 displays the structure of the program.

GET-RELATIONS-BUILD-DATA-DICTIONARY

This process builds the data dictionary, relation table, and relation pool from the relations file built by the FDANALYZER. This is basically an initialization step which requires no interaction and is not key to the achievement of the program's goals. The hashing technique used in building the data dictionary is not sophisticated but has not caused any problems during testing. The other two data structures are built in a sequential manner.

HANDLE-TRANSACTIONS

This process reads the file created by the FDGENERATOR which identifies the report name, each line

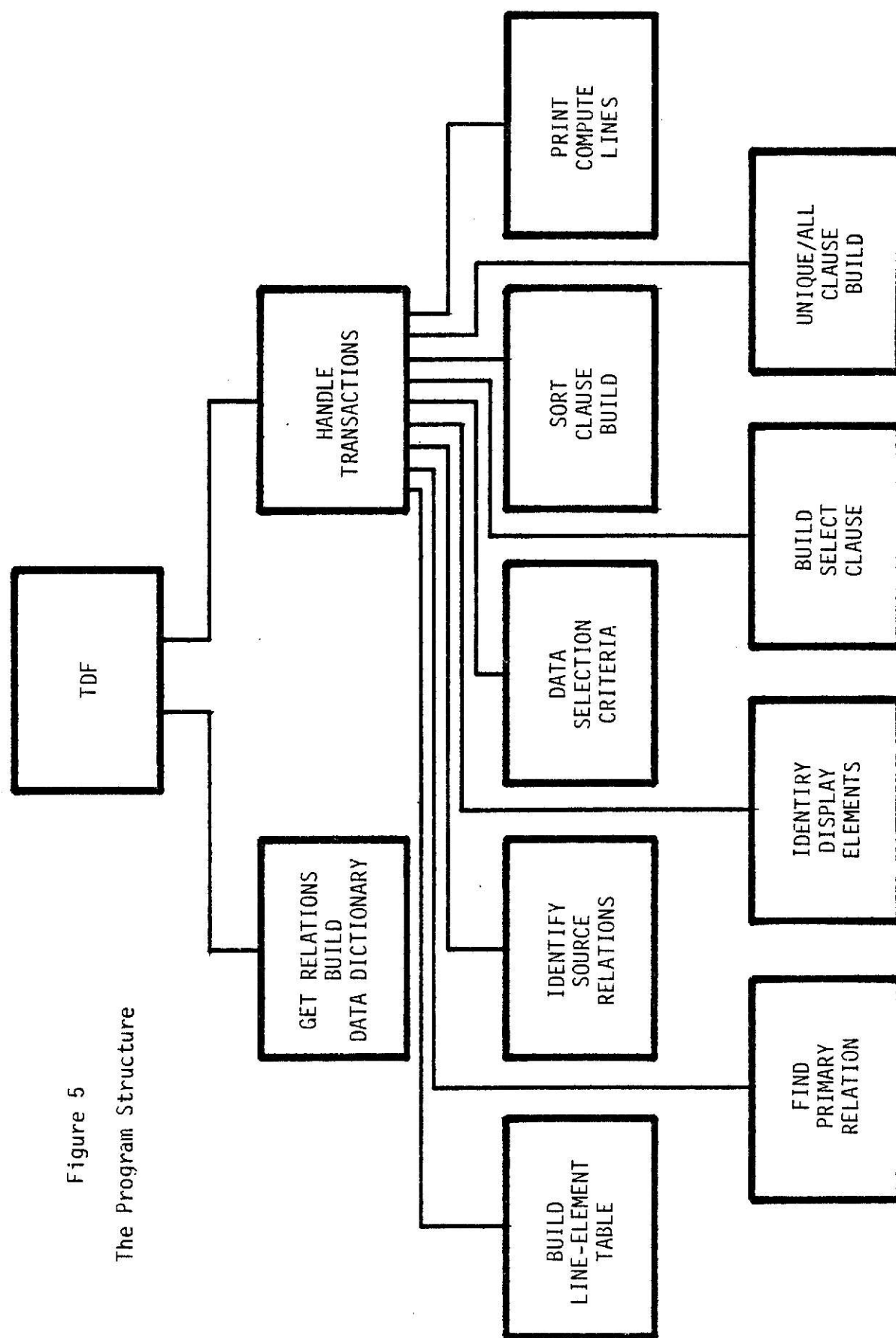


Figure 5
The Program Structure

in the report, the levels of the lines, the data items in each line, and whether each item is a temporary item or a possible determinant of the line. The line table and transaction pool are built from this information and the DEFINE-NEXT-LINE process is performed for each line.

DEFINE-NEXT-LINE

The only duty of this process is to invoke the other processes which perform the tasks needed to define a transaction. Each execution of this process is independent of the results of all others and results in one output statement.

BUILD-LINE-ELEMENT-TABLE

The building of the line item table is started by this process. All of the items involved are identified, temporary items are flagged and their inputs listed, and items are added to establish needed links to higher level lines in the transaction.

FIND-PRIMARY-RELATION

This is the longest and most complex routine in TDF. Its task is to find the primary relation for a transaction. The primary relation is later used to determine navigation paths. The basic steps involved in identifying the primary relation are as follows:

- * Determine a reasonable set of candidate determinants

of the line

- * Eliminate items not marked as possible determinants by the user
- * Eliminate items not key to any relation
- * Identify items which only appear in relations as keys
- * Eliminate items which never appear in a concatenated key with the above items
- * Determine a set of candidate relations
 - * Start with every relation with any candidate determinant in its key
 - * Eliminate those relations without fully covered keys
 - * Eliminate any candidate relation which can be covered by another
- * The remaining relation is the primary relation

The algorithm will stop at several intermediate points if only one candidate relation remains. As noted earlier, it was discovered that in some circumstances no single relation can cover the rest. It appears that only minor changes would be required to handle the situation of multiple primary relations.

IDENTIFY-SOURCE-RELATIONS

TDF must take responsibility for identifying the source relation of every item involved in the line. Once a primary relation has been identified, This can be done easily, except in those instances where an item appears in multiple relations in a non-key role and the keys of two or more of those relations can be covered from the items originally listed as being in the line.

At present, this situation cannot be handled by TDF; however it appears to be uncommon, at least in testing.

The following algorithm is used:

```

Repeat for each data item involved in the line
  If item is in primary relation
    Use primary relation
  Else
    If item appears in only one relation as non-key
      Use that relation
    Else
      Move all relations in which it appears as
        non-key to candidate list
      Repeat for each candidate
        If all keys of that relation are not in
          the line
          Eliminate relation from candidacy
        Endif
      Endloop
      If only one candidate is left
        Use that relation
      Else
        Error, source relation is indeterminable
      Endif
    Endif
  Endif
Endloop

```

**SENTENCES IN THE
LAST PARAGRAPH OF
PAGE 29 APPEAR TO
BE CUT OF ON THE
RIGHT SIDE BUT
THEY ARE NOT.**

**THIS IS AS RECEIVED
FROM THE
CUSTOMER.**

IDENTIFY-DISPLAY-ELEMENTS

The items in the line which are actually to be displayed are identified by asking the user about each item individually. This information is then stored in the line item table.

DATA-SELECTION-CRITERIA

Data selection criteria are identified interactively. At present, the user only identifies one operand and the operator for each condition. The operand identified must come from the set of items listed in the line, while the other operand is restricted to being a value input from a terminal during execution of the transaction.

BUILD-SELECT-CLAUSE

This process builds the SELECT clause from data gathered by the previous processes and stored in the line item table. It is primarily an exercise in string handling and requires no user involvement.

SORT-CLAUSE-BUILD

The sort clause is built by asking the user to identify any sort keys in major to minor sequence and to note whether each should be presented in ascending or

descending order.

UNIQUE-ALL-CLAUSE-BUILD

If the entire key of the primary relation is not to be displayed, the user is asked if duplicate lines should be retrieved.

PRINT-COMPUTE-LINES

A set of clauses identifying each temporary item and the items from which they are derived, is added to the other clauses already output. This requires printing lines which were built earlier and stored.

TESTING

The testing effort required that input files be supplied from the FDGEN and FDANALYZER programs. Because these programs were not modified to produce these files until after testing of TDF was to begin the input files had to be hand built. This was not a significant problem and did not delay the implementation.

Testing started as soon as there was enough code to build the data dictionary and continued as each routine was added or changed. Every test involved the entire program,

not just the most recently changed routine, helping to guarantee that all the routines continued to work as the design was enlarged or modified.

CONCLUSIONS

Data structures and processes together determine a program. The data structures used by TDF became a hinderance by the end of the implementation. If TDF is ever enhanced or modified, new data structures should be considered.

One of the major goals of this project was to show that algorithms to identify navigation paths and source relations could be developed. The implementation effort did show this, in addition to verifying that the model chosen was attainable, at least for a restricted environment.

4. EVALUATION AND CONCLUSIONS

The goal of this effort was to design and implement a transaction definition facility for the SNAP system to allow users to define database transactions without requiring knowledge of the structure of the database. Within the constrained environment of database retrievals only, TDF does handle a wide variety of transactions.

There are two important criteria that should be used to evaluate TDF:

- * How easy is it to understand and use
- * How well it covers all possible transactions

Neither of these questions can be answered at present.

USABILITY

Though TDF seems very simplistic in its interaction, the designer is the poorest judge of such matters. To better evaluate its usability, experiments like those described by Shneiderman[12] and Reisner[13] should be conducted. Until such experiments are made, the examples in Appendix 2 will have to stand as the only testimony to TDF's usability.

COVERAGE

Without a method of classifying the functions required to cover all transactons, it is difficult to identify or describe the limitations of TDF. However in the following areas, limitations have been recognized and can be

discussed:

* Grouping-type calculations

Totals, counts, maximums, etc are not handled by TDF. It appears that this limitation can be overcome, though possibly in an awkward manner.

* Data selection criteria

TDF does not allow general conditional expressions to be specified as it does not use the 'OR' operator. Though this operator can be easily added, there are human engineering considerations as to how to give this capability to the user.

* Generality of calculations

TDF does not allow calculations to be completely defined, as it only accepts operands, not operators. When this facility is completed, some provision must be made for conditional operators within expressions. In other words, it must allow the equivalent of the following:

```
IF X = A
    THEN Y := B;
    ELSE Y := C;
```

While the previously discussed problems dealt only with TDF, there are other problems with the system as a whole:

1. The user is required to identify all transactions and all the data items involved in each transaction before starting to use SNAP.

2. There is no way to store commonly used calculations in the system
3. There is no way to modify or display the definition of a transaction
4. The system does not allow the user to refer to objects (relations) only data items (attributes)
5. The system does not let the user check his actions by presenting a paraphrased version of the transaction definition.

The following proposals for a new version of SNAP would eliminate or reduce the problems listed above:

1. Transactions would be split into two categories
 - a. Data retrieval only
 - b. Maintenance - add, update or delete transactions
2. Transactions would be completely defined, one at a time
3. Maintenance transactions would have to be defined before retrieval transactions which accessed those elements
4. Relationships between data items would not have to be defined for retrieval transactions
5. Transaction definition would be an iterative process, with the system helping the user to identify all elements involved in stages
6. The user would be asked to name certain relations

after they had been defined by the system

7. During the process of defining a transaction, a paraphrased version would be presented to the user by the system
8. The only output of the definition process would be a log of the questions and answers used to define a transaction.
9. Transactions would be executed by translating the log for that transaction into an executable program and checking that program for correctness before executing
10. Definitions of temporary data items could be defined once and reused each time a transaction needed that calculation
11. Transactions could be defined at any time, as long as previous restrictions were maintained

While this proposal still is far short of defining a complete application system, it does try to build a framework for a truly usable environment.

CONCLUSION

A facility to define retrieval transactions for a

relational database system has been defined and implemented. This facility (TDF) allows users to interactively define transactions without knowledge of the database's schema, or even the database model used. During the interaction, the user must supply only five types of information:

- * Which items are to be displayed
- * How to calculate derived data items
- * What criteria to use to restrict the volume of data retrieved
- * How to sort retrieved data
- * Whether duplicate tuples should be retrieved

All other information is supplied by TDF.

If this work can be extended as expected, it may well be possible to build full systems which can be used without programming or computer knowledge. But even if such systems cannot be developed quickly, the ability to provide these same capabilities to computer programmers would help eliminate many of the current problems in software production and increase programmer productivity and software quality.

REFERENCES

1. Boehm, B. W., "Software Engineering", IEEE Transactions on Computers, Vol. C-25, No. 12, Dec. 1976, page 1227.
2. Boehm, page 1226.
3. Myers, W., "The Need for Software Engineering", Computer, Vol. 11, No. 2, Feb. 1978, page 12.
4. Myers, page 13.
5. Boehm, page 1226.
6. Mills, H. D., "Software Development", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, Dec. 1976, page 267.
7. Zelkowitz, M. V., "perspectives on Software Engineering", Computing Surveys, Vol. 10, No. 2, June 1978, page 197.
8. Brooks, F. P., "The Mythical Man-Month", Datamation, Vol. 20, No. 12, Dec. 1974, page 50.
9. Maryanski, F. J., et al., "Automatic Generation of Third Normal Form Relations", Technical Report No. CS 77-21, Computer Science Department, Kansas State University, Manhattan, Kansas, 1977.
10. Chamberlin, D. D., et al, "SEQUEL 2, A Unified Approach to Data Definition, Manipulation, and Control", IBM Journal of Research and Development, Vol. 4, No. 6, Nov. 1976, pp. 560-575.
11. NCR Corporation, NCR IMOS COBOL Student Text, EP-9823-00, Dayton, Ohio, 1978.
12. Shneiderman, B., "Improving the Human Factors Aspect of Database Interactions", ACM Transactions on Database Systems, Vol. 3, No. 4, Dec. 1978, pp. 417-439.
13. Reisner, P., "Use of Psychological Experimentation as an aid to Development of a Query Language", IEEE Trans. on Software Engineering, Vol. SE-3, No. 3, May 1977, pp. 218-229.

BIBLIOGRAPHY

- Boehm, B. W., "Software Engineering", IEEE Transactions on Computers, Vol. C-25, No. 12, Dec. 1976, pp. 1226-1241.
- Brooks, F. P., "The Mythical Man-Month", Datamation, Vol. 20, No. 12, Dec. 1974, pp. 44-52.
- Chamberlin, D. D., et al, "SEQUEL 2, A Unified Approach to Data Definition, Manipulation, and Control", IBM Journal of Research and Development, Vol. 4, No. 6, Nov. 1976, pp. 560-575.
- Maryanski, F. J., et al., "Automatic Generation of Third Normal Form Relations", Technical Report No. CS 77-21, Computer Science Department, Kansas State University, Manhattan, Kansas, 1977.
- Mills, H. D., "Software Development", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, Dec. 1976, pp. 265-273.
- Myers, W., "The Need for Software Engineering", Computer, Vol. 11, No. 2, Feb. 1978, pp. 12-26.
- NCR Corporation, NCR IMOS COBOL Student Text, EP-9823-00, Dayton, Ohio, 1978.
- Reisner, P., "Use of Psychological Experimentation as an Aid to Development of a Query Language", IEEE Trans. on Software Engineering, Vol. SE-3, No. 3, May 1977, pp. 218-229.
- Shneiderman, B., "Improving the Human Factors Aspect of Database Interactions", ACM Transactions on Database Systems, Vol. 3, No. 4, Dec. 1978, pp. 417-439.
- Zelkowitz, M. V., "Perspectives on Software Engineering", Computing Surveys, Vol. 10, No. 2, June 1978, pp. 197-216.

APPENDIX 1
SYNTAX OF THE OUTPUT OF TDF


```

statement      ::= [accept-list] query
query          ::= query-block [ORDER BY
                    ord-spec-list][UNIQUE]
query-block    ::= select-clause FROM from-list [WHERE
                    boolean]
select-clause  ::= SELECT (ALL) select-expr-list
                    ! SELECT (NEXT) select-expr-list
select-expr-list ::= var-name
                    ! select-expr-list var-name
from-list      ::= relation-name
                    ! from-list relation-name
boolean        ::= predicate
                    ! predicate AND boolean
predicate      ::= var-name compar-op parm-name
                    ! var-name compar-op current-name
compar-op      ::= =
                    ! <
                    ! >
                    ! <>
var-name       ::= db-name
                    ! temp-name
relation-name  ::= R !! number
parm-name      ::= PARAMETER-number
current-name   ::= CURRENT (var-name)
db-name        ::= relation-name.name
temp-name      ::= name
accept-list    ::= accept-clause
                    ! accept-list accept-clause
accept-clause  ::= ACCEPT parm-name FROM CRT

```

APPENDIX 2
EXAMPLES OF INTERACTION WITH TDF
AND THE RESULTING OUTPUT

The Relations:

(Keys are marked by '***')

RELATION NO. 01

DIVNAME ***

DIV-MGR

RELATION NO. 02

JOBNO ***

JOBNAME

SKILL-GRP

SAL-GROUP

INS-CODE

RELATION NO. 03

INS-CODE ***

INS

COVER-HOSP

COVER-DENT

COVER-LIFE

COVER-MMED

RELATION NO. 04

EMPNO ***

EMPNAME

SAL

DEPTNAME

JOBNO

RELATION NO. 05

DEPTNAME ***

DEPT-MGR

DIVNAME

Transaction No. 1

The Problem Statement:

Print the name and salary of every employee with a salary greater than the value input from the user's terminal.

The Original Input to USERINTERFACE:

Report name: R2

Line 1: EMPNO EMPNAME SAL

The Log of User Interaction with TDF:

TDF: STARTING NEW TRANSACTION (REPORT)

TDF: TRANSACTION-NAME: R2

```

TDF:      EMPNO      EMPNAME      SAL
TDF:      WHICH DO YOU WANT TO DISPLAY
TDF:      Y - YES, THIS ITEM
TDF:      N - NOT, THIS ITEM
TDF:      A - ALL OF THE REST OF THE ITEMS
TDF:      S - STOP, NONE OF THE REST OF THE ITEMS
TDF:      EMPNO
USER:      N
TDF:      EMPNAME
USER:      A
TDF:      DO YOU WANT THIS TRANSACTION TO INVOLVE EVERY
TDF:      OCCURRENCE OF THE DATA FOR THIS LINE? (Y/N)
TDF:      ('N' IF THERE IS ADDITIONAL SELECTION CRITERIA)
USER:      N
TDF:      OF THESE ELEMENTS IN THE LINE:
TDF:      EMPNO      EMPNAME      SAL
TDF:      WHICH IS INVOLVED IN THIS CONDITION?
USER:      SAL
TDF:      PLEASE ENTER CONDITION OPERATOR (=, <, >, ETC)
USER:      >
TDF:      FOR NOW, THE 'RIGHT-HAND SIDE' WILL COME FROM
TDF:      THE CRT
TDF:      OF THESE ELEMENTS IN THE LINE:
TDF:      EMPNO      EMPNAME      SAL
TDF:      WHICH IS INVOLVED IN THIS CONDITION?
USER:
TDF:      EMPNAME      SAL
TDF:      DO YOU WANT THIS LINE SORTED? (Y/N)
USER:      Y
TDF:      ENTER SORT KEYS MAJOR TO MINOR
TDF:      EMPNAME
USER:      Y
TDF:      ASCENDING OR DESCENDING? (A/D)
USER:      A
TDF:      ARE THERE ANY MORE SORT KEYS? (Y/N)
USER:      N
TDF:      DO YOU WANT TO DISPLAY ALL VALID RECORDS,
TDF:      OR JUST THOSE THAT ARE UNIQUE? (A/U)
USER:      A

```

The SEQUEL Output by TDF:

```

                TRANSACTION NAME: R2
ACCEPT PARAMETER-01 FROM CRT
SELECT (ALL)
    R04.EMPNAME
    R04.SAL
FROM
    R04
WHERE
    R04.SAL > PARAMETER-01
ORDERED BY
    R04.EMPNAME ASCENDING

```

Transaction No. 2

The Problem Statement:

Print the name and manager of every division, and for each division list the names and managers of every department in that division, and for every department, list the name and salary of every employee in that department with a jobname equal to the value input by the user at his terminal. Present the division data ordered by division name, the department data by department name, and the employee data by employee name.

The Original Input to USERINTERFACE:

Report name: EMPLOYEES
 Line 1: DIVNAME PHONE-NO MAIL-STOP
 Line 2: DEPTNAME MAIL-STOP PHONE-NO
 Line 3: EMPNO EMPNAME SAL JOBNAME

The Log of User Interaction with TDF:

TDF: STARTING NEW TRANSACTION (REPORT)
 TDF: TRANSACTION-NAME: EMPLOYEES
 TDF: DIVNAME DIV-MGR
 TDF: WHICH DO YOU WANT TO DISPLAY
 TDF: Y - YES, THIS ITEM
 TDF: N - NOT, THIS ITEM
 TDF: A - ALL OF THE REST OF THE ITEMS
 TDF: S - STOP, NONE OF THE REST OF THE ITEMS
 TDF: DIVNAME
 USER: A
 TDF: DO YOU WANT THIS TRANSACTION TO INVOLVE EVERY
 TDF: OCCURRENCE OF THE DATA FOR THIS LINE? (Y/N)
 TDF: ('N' IF THERE IS ADDITIONAL SELECTION CRITERIA)
 USER: Y
 TDF: DIVNAME DIV-MGR
 TDF: DO YOU WANT THIS LINE SORTED? (Y/N)
 USER: Y
 TDF: ENTER SORT KEYS MAJOR TO MINOR
 TDF: DIVNAME
 USER: Y
 TDF: ASCENDING OR DESCENDING? (A/D)
 USER: A
 TDF: ARE THERE ANY MORE SORT KEYS? (Y/N)
 USER: N

TDF: DEPTNAME DEPT-MGR
 TDF: WHICH DO YOU WANT TO DISPLAY
 TDF: Y - YES, THIS ITEM
 TDF: N - NOT, THIS ITEM
 TDF: A - ALL OF THE REST OF THE ITEMS
 TDF: S - STOP, NONE OF THE REST OF THE ITEMS
 TDF: DEPTNAME
 USER: A
 TDF: DO YOU WANT THIS TRANSACTION TO INVOLVE EVERY
 TDF: OCCURRENCE OF THE DATA FOR THIS LINE? (Y/N)
 TDF: ('N' IF THERE IS ADDITIONAL SELECTION CRITERIA)
 USER: Y
 TDF: DEPTNAME DEPT-MGR
 TDF: DO YOU WANT THIS LINE SORTED? (Y/N)
 USER: Y
 TDF: ENTER SORT KEYS MAJOR TO MINOR
 TDF: DEPTNAME
 USER: Y
 TDF: ASCENDING OR DESCENDING? (A/D)
 USER: A
 TDF: ARE THERE ANY MORE SORT KEYS? (Y/N)
 USER: N
 TDF: EMPNO EMPNAME SAL JOBNAME
 TDF: WHICH DO YOU WANT TO DISPLAY
 TDF: Y - YES, THIS ITEM
 TDF: N - NOT, THIS ITEM
 TDF: A - ALL OF THE REST OF THE ITEMS
 TDF: S - STOP, NONE OF THE REST OF THE ITEMS
 TDF: EMPNO
 USER: N
 TDF: EMPNAME
 USER: A
 TDF: DO YOU WANT THIS TRANSACTION TO INVOLVE EVERY
 TDF: OCCURRENCE OF THE DATA FOR THIS LINE? (Y/N)
 TDF: ('N' IF THERE IS ADDITIONAL SELECTION CRITERIA)
 USER: N
 TDF: OF THESE ELEMENTS IN THE LINE:
 TDF: EMPNO EMPNAME SAL JOBNAME
 TDF: WHICH IS INVOLVED IN THIS CONDITION?
 USER: JOBNAM
 TDF: NOT IN THE LINE, TRY AGAIN
 USER: JOBNAM
 TDF: PLEASE ENTER CONDITION OPERATOR (=, <, >, ETC)
 USER: =
 TDF: FOR NOW, THE 'RIGHT-HAND SIDE' WILL COME FROM
 TDF: THE CRT
 TDF: OF THESE ELEMENTS IN THE LINE:
 TDF: EMPNO EMPNAME SAL JOBNAME
 TDF: WHICH IS INVOLVED IN THIS CONDITION?
 USER:
 TDF: EMPNAME SAL JOBNAME
 TDF: DO YOU WANT THIS LINE SORTED? (Y/N)
 USER: Y
 TDF: ENTER SORT KEYS MAJOR TO MINOR

TDF: EMPNAME
 USER: Y
 TDF: ASCENDING OR DESCENDING? (A/D)
 USER: A
 TDF: ARE THERE ANY MORE SORT KEYS? (Y/N)
 USER: N
 TDF: DO YOU WANT TO DISPLAY ALL VALID RECORDS,
 TDF: OR JUST THOSE THAT ARE UNIQUE? (A/U)
 USER: A

The SEQUEL Produced by TDF:

```

                                TRANSACTION-NAME:EMPLOYEES
SELECT (NEXT)
    R01.DIVNAME
    R01.DIV-MGR
FROM
    R01
ORDERED BY
    R01.DIVNAME  ASCENDING
SELECT (NEXT)
    R05.DEPTNAME
    R05.DEPT-MGR
    R05.DIVNAME
FROM
    R05
WHERE
    R05.DIVNAME = CURRENT(R01.DIVNAME)
ORDERED BY
    R05.DEPTNAME  ASCENDING
ACCEPT PARAMETER-01  FROM CRT
SELECT (ALL)
    R04.EMPNAME
    R04.SAL
    R02.JOBNAME
    R04.DEPTNAME
FROM
    R02    R04
WHERE
    R02.JOBNO = R04.JOBNO
    AND  R04.DEPTNAME = CURRENT(R05.DEPTNAME)
    AND  R02.JOBNAME  =  PARAMETER-01
ORDERED BY
    R04.EMPNAME  ASCENDING
  
```

APPENDIX 3
A LISTING OF TDF


```

IDENTIFICATION DIVISION.
PROGRAM-ID. TDF.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. NCR-IMOS.
OBJECT-COMPUTER. NCR-IMOS.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT PRINT-FILE ASSIGN TO PRINTER
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL.
    SELECT NAMETABLE
        ASSIGN DISC.
    SELECT RELATION-FILE
        ASSIGN DISC.
DATA DIVISION.
FILE SECTION.
FD  PRINT-FILE    BLOCK CONTAINS 1 RECORDS
                    LABEL RECORDS ARE OMITTED.
01  PRINT-REC          PIC X(132).
FD  NAMETABLE
    BLOCK CONTAINS 34 RECORDS
    LABEL RECORD STANDARD.
01  NAMETABLE-REC      PIC X(15).
FD  RELATION-FILE
    BLOCK CONTAINS 51 RECORDS
    LABEL RECORD STANDARD.
01  RELATION-REC       PIC X(10).
WORKING-STORAGE SECTION.
01  DATA-Dictionary.
    02  DD-MAX          PIC 999 COMP    VALUE 255.
    02  DDX-HOLD        PIC 999 COMP.
    02  DD-ENTRY        OCCURS 255 INDEXED DDX.
        03  DD-NAME      PIC X(10).
        03  DD-HASH-ST   PIC 9 COMP.
        03  DD-RELAT-KEY PIC 1(56) BIT.
        03  DD-RELAT-NON PIC 1(56) BIT.
        03  DD-COVERED   PIC X.
01  RELATION-TABLE.
    02  REL-MAX          PIC 99 COMP    VALUE 56.
    02  REL-LAST        PIC 99 COMP.
    02  RELX-HOLD        INDEX.
    02  RELATION-ENTRY  OCCURS 56 INDEXED RELX.
        03  REL-1ST      INDEX.
        03  REL-LHS      PIC 99 COMP.
        03  REL-RHS      PIC 99 COMP.
        03  REL-USED     PIC 99.
        03  REL-CANDIDATE PIC X.
        03  REL-NAVIGATION-COUNTER PIC 99.
        03  FILLER       PIC X.
***** NOTE: THIS FILLER USED TO FIX COMPILER PROBLEM.
01  RELATION-POOL.
    02  RPX-HOLD        PIC 9999 COMP.
    02  RP-ELEM-ID      INDEX OCCURS 1200

```

```

                                INDEXED RPX.
01  REL-REC.
    02  SIZE-REC.
        03  ELEM-LHS                PIC 99.
        03  ELEM-RHS                PIC 99.
        03  FILLER                  PIC X(6).
    02  ELEM-REC                    REDEFINES SIZE-REC.
        03  ELEM-NAME                PIC X(10).
    02  ELEM-REC-R                  REDEFINES SIZE-REC.
        03  ELEM-CHAR1              PIC X.
        03  FILER                   PIC X(9).
01  TR-POOL.
    02  TRX-HOLD                    INDEX.
    02  TR-ENTRY                    OCCURS 200   INDEXED TRX.
        03  TR-ID                    INDEX.
        03  TR-KEY-FLAG              PIC X.
        03  TR-FLAG                  PIC X.
01  LINE-TABLE.
    02  LNX-HOLD                    PIC 99   COMP.
    02  LINE-ENTRY                  OCCURS 10   INDEXED LNX.
        03  LINE-LINK.
            04  LINE-LEVEL            PIC X.
            04  LINE-OCCUR            PIC X.
            04  FILLER                PIC X.
        03  LINE-1ST                 INDEX.
        03  LINE-STOP                 INDEX.
        03  LINE-NO-DETERMINANTS      PIC 99   COMP.
        03  LINE-PRIMARY-RELATION     PIC 99.
        03  FILLER                   PIC XX.
*   THIS FILLER IS USED TO AVOID COMPILER BUG
01  TRANS-REC.
    02  TRANS-NAME                  PIC X(10).
    02  TRANS-LEVEL                 PIC 9.
    02  TRANS-LINK                  PIC XXX.
    02  TRANS-KEY-FLAG              PIC X.
01  LINE-ELEMENTS-TABLE.
    02  LEX-HOLD                    PIC 99   COMP.
    02  LE-MAX                      PIC 99   COMP      VALUE 70.
    02  LE-END-IN-DD                PIC 99   COMP.
    02  LE-LAST                     PIC 99   COMP.
    02  LE-ENTRIES                  OCCURS 70   INDEXED LEX.
        03  LE-ID                    INDEX.
        03  LE-KEY-FLAG              PIC X.
        03  LE-ORIGIONAL             PIC X.
        03  LE-FUNCTION              PIC X.
        03  LE-PROCESS               PIC X.
        03  LE-SORT                  PIC X.
        03  LE-TEMPORARY             PIC X.
        03  LE-RELATION              PIC 99   COMP.
        03  LE-NAV-RELATION          PIC 99   COMP.
01  WHERE-STACK.
    02  WS-MAX                      PIC 99   COMP      VALUE 99.
    02  WS-LAST                     PIC 99   COMP.
    02  WHERE-STACK-ENTRIES         OCCURS 99   INDEXED WSX.

```

```

      03 WHERE-STACK-REL      INDEX.
01  TEMP-TABLE.
      02 TT-MAX              PIC 99  COMP      VALUE 40.
      02 TT-LAST            PIC 99  COMP.
      02 TT-ENTRY           OCCURS 40  INDEXED TTX.
      03 TT-NAME            PIC X(10).
01  ADDED-ELEMENT-LIST.
      02 AD-MAX              PIC 99  COMP      VALUE 20.
      02 AD-LAST            PIC 99  COMP.
      02 AD-ENTRY           OCCURS 20  INDEXED ADX.
      03 ADDED-ID           INDEX.
01  DISPLAY-LINE-DATA.
      02 DISP-LINE-ENTRY     OCCURS 6.
      03 FILLER              PIC X(3).
      03 DISP-LINE-NAME     PIC X(10).
01  PR-DD1.
      02 PR-DDX              PIC ZZZ.
      02 FILLER              PIC X          VALUE SPACES.
      02 PR-DD-HASH-ST       PIC 9.
      02 FILLER              PIC X          VALUE SPACES.
      02 PR-DD-NAME          PIC X(12).
      02 PR-DD-KEY           PIC X(56).
      02 FILLER              PIC X(2)      VALUE SPACES.
      02 PR-DD-NON          PIC X(56).
01  HEAD1.
      02 HEAD-DATE           PIC X(8).
      02 FILLER              PIC X(40)     VALUE SPACES.
      02 FILLER              PIC X(50)
                                VALUE "TRANSACTION DEFINITION FACILITY".
      02 FILLER              PIC X(26)     VALUE SPACES.
      02 FILLER              PIC X(5)      VALUE "PAGE ".
      02 HEAD-PAGE-NO        PIC ZZZ.
01  HEAD2.
      02 FILLER              PIC X(132)    VALUE SPACES.
01  HEAD-DD1.
      02 FILLER              PIC X(40)
                                VALUE " ID H NAME          KEY RELATIONS".
      02 FILLER              PIC X(36)     VALUE SPACES.
      02 FILLER              PIC X(18)
                                VALUE "NON-KEY RELATIONS".
*
      RESERVED FOR FUTURE GOODIES
01  COMPUTE-POOL            PIC X(250).
01  WHERE-POOL              PIC X(250).
01  ANS.
      02 ANS5.
          03 ANS4.
              04 ANS3.
                  05 ANS2.
                      06 ANS1          PIC X.
                      06 FILLER        PIC X.
                  05 FILLER          PIC X.
              04 FILLER          PIC X.
          03 FILLER          PIC X.
      02 FILLER          PIC X(5).

```

01	WORK-DATE.		
	02 WORK-YR	PIC 99.	
	02 WORK-MO	PIC 99.	
	02 WORK-DA	PIC 99.	
01	DD-HASH-NAME.		
	02 DD-HASH-N1	PIC 9(12)	COMP-5.
	02 DD-HASH-N2	PIC 9(12)	COMP-5.
77	PAGE-NO	PIC 999.	
01	BC-CONVERT-B	PIC 1(56)	BIT.
01	BC-CONVERT-C.		
	02 ONE-CHAR	PIC X	OCCURS 56 INDEXED OX.
01	COUNTERS-BOOLS-SUBSCRIPTS-ETC.		
	02 ONE-BIT	PIC 1(56)	BIT.
	02 BC-CONVERT-WORK	PIC 1(56)	BIT.
	02 BOOL-PRIMARY	PIC 1(56)	BIT.
	02 BOOL-RELATION	PIC 1(56)	BIT.
	02 BOOL-CANDIDATES	PIC 1(56)	BIT.
	02 B-AND	PIC 1(56)	BIT.
	02 B-OR	PIC 1(56)	BIT.
	02 BOOL3	PIC 1(56)	BIT.
	02 BOOL-ALL-1	PIC 1(56)	BIT
	VALUE		
	B"11".		
	02 B-WORK0	PIC 1(56)	BIT.
	02 B-WORK1	PIC 1(56)	BIT.
	02 B-WORK2	PIC 1(56)	BIT.
	02 RELATION-BOOL	PIC 1(56)	BIT
			VALUE B"10000000".
	02 FUNCTION	PIC X(10).	
	02 BLANKS	PIC X(132)	VALUE SPACES.
	02 LNMAX	PIC 999	COMP.
	02 PRINT-LINE	PIC X(132).	
	02 NO-OF	PIC 99	COMP.
	02 LAST-LEVEL	PIC X.	
	02 LHS	PIC 99	COMP.
	02 RHS	PIC 99	COMP.
	02 DOM-CTR	PIC 9999	COMP.
	02 DEBUG-RTNS-SW	PIC X.	
	02 DD-SLOW	PIC X	VALUE "N".
	02 TR-SLOW	PIC X	VALUE "N".
	02 PR-SLOW	PIC X	VALUE "N".
	02 DI-SLOW	PIC X	VALUE "N".
	02 DD-PRINT-SW	PIC X	VALUE "N".
	02 R-PRINT-SW	PIC X	VALUE "N".
	02 TRANS-PRINT-SW	PIC X	VALUE "N".
	02 MSG	PIC X(79).	
	02 GROUP-FUNCTION	PIC X(7).	
	02 GROUP-ELEMENT	PIC X(10).	
	02 GROUP-CONTROL	PIC X(10).	
	02 LINES-LEFT	PIC 99	COMP.
	02 EOF-REL	PIC X.	
	02 DUMMY	PIC X.	
	02 TEMP-STRING	PIC X(80).	

02	SEQL-LINE	PIC X(80).
02	FULL-NAME	PIC X(30).
02	A-D	PIC X(16).
02	FN1	PIC X(30).
02	FN2	PIC X(30).
02	FROM-CLAUSE	PIC X(130).
02	WHERE-CLAUSE	PIC X(130).
02	FROM-NEXT	PIC 999 COMP.
02	WHERE-POINTER	PIC 999 COMP.
02	EOF-TRANS	PIC X.
02	PRIMARY-RELATION	PIC 99 COMP.
02	LEFT-HAND-SIDE	PIC X.
02	REL-NO	PIC 99.
02	NUM	PIC 9(14) COMP.
02	QUOTIENT	PIC 9(14) COMP.
02	REMAINS	PIC 9(14) COMP.
02	DIV	PIC 9(14) COMP.
02	DISPLAY-NUM	PIC 9(14).
02	NO-DETERMINANTS	PIC 99.
02	CANDIDATES-LEFT	PIC 99.
02	COM-PTR	PIC 9999 COMP.
02	WHERE-PTR	PIC 9999 COMP.
02	PTR-LAST	PIC 9999 COMP.
02	ELEMENT-COUNT	PIC 99.
02	LAST-CANDIDATE	PIC 99 COMP.
02	CANDIDATES-NO-OF-KEYS	PIC 99.
02	REL-SS	PIC 99 COMP.
02	SSL	PIC 999 COMP.
02	PARM	PIC 99.
02	MORE-RESTRICTIONS	PIC X.
02	ALL-IN-LINE	PIC X.
02	DD-HASHED-OK	PIC X.
02	DD-HASHED-IN-ERR	PIC X.
02	DD-HASHED-BAD	PIC 99.
02	DD-HASH-TEMP	PIC 999.
02	DD-HASH-DIST	PIC 9999 COMP VALUE ZERO.
02	TRANSACTION-NAME	PIC X(10).
02	TEST-NAME	PIC X(10).
02	NEXT-LINE	PIC 99 COMP.
02	ELEM-ID	PIC 999.
02	FOUND-SOURCE	PIC X.
02	FOUND-IT	PIC X.
02	DEAD-END	PIC X.
02	FOUND-NEXT	PIC X.
02	ABSENT	PIC X.
02	REPEAT	PIC X.
02	COVERED-KEYS	PIC X.
02	CANDIDATE-REACHED	PIC X.
02	AND-SW	PIC X.
02	PRINT-WHERE	PIC X.
02	GOOD-CANDIDATE	PIC X.
02	GOT-KEY	PIC X.
02	GOOD-RESPONSE	PIC X.

```

02 LAST-RESPONSE          PIC X.
02 MORE-SORT-KEYS         PIC X.
02 DISP-ALL-KEYS         PIC X.
02 ALL-UNIQUE            PIC X.
02 TEMPORARY-NAME        PIC X(10).
02 ELEMENT-NAME          PIC X(10).
01 CONSTANTS.
02 A-YES                 PIC X          VALUE "Y".
02 A-NO                  PIC X          VALUE "N".
02 C-DISPLAY            PIC X          VALUE "D".
02 C-NAVIGATION         PIC X          VALUE "N".
02 BOOL-1               PIC 1(8)      BIT
                                VALUE B"10000000".

PROCEDURE DIVISION.
THE-BEGINNING.
  OPEN OUTPUT
    PRINT-FILE
      INPUT
        NAMETABLE
        RELATION-FILE.
  DISPLAY " ANY DEBUGGING THIS TIME?      (Y/N) "
  ACCEPT DEBUG-RTNS-SW.
  IF DEBUG-RTNS-SW = A-YES
    DISPLAY " SLOW DOWN DATA DICTIONARY ?  (Y/N) "
    ACCEPT DD-SLOW
    DISPLAY " SLOW DOWN TRANSACTION RTNS?  (Y/N) "
    ACCEPT TR-SLOW
    DISPLAY " SLOW DOWN PRIMARY RELATION?  (Y/N) "
    ACCEPT PR-SLOW
    DISPLAY " SLOW DOWN DISPLAY?          (Y/N) "
    ACCEPT DI-SLOW
    DISPLAY "PRINT DATA DICTIONARY?      (Y/N) "
    ACCEPT DD-PRINT-SW
    DISPLAY "PRINT RELATIONS?            (Y/N) "
    ACCEPT R-PRINT-SW
    DISPLAY "PRINT TRANSACTIONS?         (Y/N) "
    ACCEPT TRANS-PRINT-SW.
  ACCEPT WORK-DATE FROM DATE.
  STRING WORK-MO "/" WORK-DA "/" WORK-YR
    DELIMITED SIZE
    INTO HEAD-DATE.
  MOVE 1 TO PAGE-NO.
  MOVE 0 TO LINES-LEFT.
  MOVE 1 TO NEXT-LINE.
  PERFORM INIT-DATA-DICT
    VARYING DDX FROM 1 BY 1 UNTIL DDX > DD-MAX.
  PERFORM DD-HASHED-INIT.
  MOVE A-NO TO EOF-REL.
  SET RPX TO 1.
  PERFORM GET-REL-REC.
  PERFORM GET-RELATIONS-BUILD-DATA-DICT
    VARYING RELX FROM 1 BY 1 UNTIL
      RELX > REL-MAX OR EOF-REL = A-YES.
  SET RELX DOWN BY 1.

```

```

SET REL-LAST TO RELX.
IF DD-PRINT-SW = A-YES
    PERFORM PRINT-DATA-DICT.
*
*       START ON 'NAME TABLE'
*
MOVE A-NO TO EOF-TRANS
MOVE SPACES TO TRANS-REC.
PERFORM GET-TRANS-REC.
PERFORM HANDLE-TRANSACTIONS
    UNTIL EOF-TRANS = A-YES.
DISPLAY "::::::::::::::::::::::::::::::::::::::::::::::::::"
    "      GOODBY".
CLOSE
    PRINT-FILE
    NAMETABLE
    RELATION-FILE.
STOP RUN.
INIT-DATA-DICT.
    MOVE SPACES TO DD-NAME(DDX).
    MOVE ZERO   TO DD-HASH-ST(DDX).
    MOVE ZERO TO DD-RELAT-KEY(DDX).
    MOVE ZERO TO DD-RELAT-NON(DDX).
    MOVE A-NO TO DD-COVERED(DDX).
GET-RELATIONS-BUILD-DATA-DICT.
    IF DD-SLOW = A-YES
        DISPLAY "STARTED NEXT RELATION".
    IF R-PRINT-SW = A-YES
        MOVE 2 TO NO-OF
        SET REL-NO TO RELX
        STRING "      RELATION NO. " REL-NO BLANKS
            DELIMITED SIZE
            INTO PRINT-LINE
        PERFORM PRINT-RTN.
    SET REL-1ST(RELX) TO RPX.
    MOVE ELEM-LHS TO REL-LHS(RELX).
    MOVE ELEM-LHS TO LHS.
    MOVE ELEM-RHS TO REL-RHS(RELX).
    MOVE ELEM-RHS TO RHS.
    MOVE A-YES TO LEFT-HAND-SIDE.
    PERFORM HANDLE-DOMAIN
        VARYING DOM-CTR FROM 1 BY 1
        UNTIL DOM-CTR > LHS.
    MOVE A-NO TO LEFT-HAND-SIDE.
*       "....STARTING RIGHT HAND SIDE      NON-KEYS"
    PERFORM HANDLE-DOMAIN
        VARYING DOM-CTR FROM 1 BY 1
        UNTIL DOM-CTR > RHS.
    SHIFT RELATION-BOOL RIGHT 1.
    PERFORM GET-REL-REC.
*       GETTING NEXT 'SIZE-REC'
HANDLE-DOMAIN.
    MOVE DOM-CTR TO DISPLAY-NUM.
    PERFORM GET-REL-REC.

```

```

IF ELEM-CHAR1 NOT = ":"
    PERFORM BUILD-ELEMENT-ENTRY
ELSE
    SUBTRACT 1 FROM REL-RHS(RELX).
BUILD-ELEMENT-ENTRY.
PERFORM DD-HASH-TABLE-CHECK-BUILD.
SET RP-ELEM-ID(RPX) TO DDX.
IF LEFT-HAND-SIDE = A-YES
*   "LEFT...KEY"
    COMPUTE DD-RELAT-KEY(DDX) =
        DD-RELAT-KEY(DDX) OR RELATION-BOOL
ELSE
*   "RIGHT...NON-KEY"
    COMPUTE DD-RELAT-NON(DDX) =
        DD-RELAT-NON(DDX) OR RELATION-BOOL.
IF R-PRINT-SW = A-YES
    IF LEFT-HAND-SIDE = A-YES
        STRING DD-NAME(DDX) " ****" BLANKS
            DELIMITED SIZE
            INTO PRINT-LINE
        PERFORM PRINT-RTN
    ELSE
        MOVE DD-NAME(DDX) TO PRINT-LINE
        PERFORM PRINT-RTN.
SET RPX UP BY 1.
IF DD-SLOW = A-YES    ACCEPT DUMMY.
GET-REL-REC.
READ RELATION-FILE INTO REL-REC
    AT END MOVE A-YES TO EOF-REL.
IF DD-SLOW = A-YES
    DISPLAY "REL-REC =" REL-REC.
DD-HASH-TABLE-CHECK-BUILD.
IF DD-SLOW = A-YES
    DISPLAY "STARTING DD-HASH BUILD".
MOVE ELEM-NAME TO DD-HASH-NAME.
PERFORM DD-HASH-COMPUTATION.
MOVE A-YES TO DD-HASHED-OK.
PERFORM DD-HASH-1ST-TRY.
IF DD-HASHED-OK = A-NO
    PERFORM DD-HASH-RETRY
        UNTIL DD-HASHED-OK = A-YES
    IF DDX = DD-MAX
        PERFORM DD-HASH-ERROR.
DD-HASH-1ST-TRY.
IF DD-SLOW = A-YES
    DISPLAY "DD-HASH....1ST TRY".
IF DD-HASH-ST(DDX) = ZERO
    MOVE DD-HASH-NAME TO DD-NAME(DDX)
    MOVE 1 TO DD-HASH-ST(DDX)
ELSE IF DD-NAME(DDX) NOT = DD-HASH-NAME
    MOVE 2 TO DD-HASH-ST(DDX)
    MOVE A-NO TO DD-HASHED-OK.
DD-HASH-RETRY.
IF DD-SLOW = A-YES

```



```

        DISPLAY "DD-HASH....RETRY".
    SET DDX UP BY 1.
    IF DD-SLOW = A-YES
        DISPLAY
            "NOW COMPARING:" DD-HASH-NAME DD-NAME(DDX).
    IF DD-HASH-ST(DDX) = ZERO
        MOVE A-YES TO DD-HASHED-OK
        MOVE DD-HASH-NAME TO DD-NAME(DDX)
        MOVE 1 TO DD-HASH-ST(DDX)
        ADD 1 TO DD-HASHED-BAD
        ADD DD-HASH-TEMP TO DD-HASH-DIST
        MOVE 1 TO DD-HASH-TEMP
    ELSE IF DD-NAME(DDX) = DD-HASH-NAME
        MOVE A-YES TO DD-HASHED-OK
    ELSE
        ADD 1 TO DD-HASH-TEMP.
*
DD-HASHED-INIT.
    MOVE 231 TO DIV    DISPLAY-NUM.
    MOVE ZERO TO DD-HASH-DIST.
    MOVE 1 TO DD-HASH-TEMP.
    IF DEBUG-RTNS-SW = A-YES
        DISPLAY "DIV = "   LINE 0   POSITION 0   ERASE
            DISPLAY-NUM      LINE 0   POSITION 7
            "DO YOU WANT TO CHANGE IT" LINE 0 POSITION 20
        ACCEPT ANS1
        IF ANS1 = "Y"
            DISPLAY "ENTER DIV"
            ACCEPT DISPLAY-NUM
            MOVE DISPLAY-NUM TO DIV.
DD-HASH-COMPUTATION.
    IF DD-SLOW = A-YES
        DISPLAY "START DD-HASH COMPUTATION".
    ADD DD-HASH-N1, DD-HASH-N2 GIVING NUM.
    DIVIDE NUM BY DIV GIVING QUOTIENT
        REMAINDER REMAINS.
    ADD 1 TO REMAINS.
    IF DD-SLOW = A-YES OR TR-SLOW = A-YES
        MOVE REMAINS TO DISPLAY-NUM
        DISPLAY "DDX = "   DISPLAY-NUM.
    SET DDX TO REMAINS.
DD-HASH-IT.
    MOVE A-NO TO DD-HASHED-OK.
    MOVE A-NO TO DD-HASHED-IN-ERR.
    PERFORM DD-HASH-COMPUTATION.
    PERFORM DD-HASH-SEARCH
        UNTIL DD-HASHED-OK = A-YES
            OR DD-HASHED-IN-ERR = A-YES.
DD-HASH-SEARCH.
    IF DD-HASH-ST(DDX) = ZERO
        MOVE A-YES TO DD-HASHED-IN-ERR
    ELSE
        IF DD-NAME(DDX) = DD-HASH-NAME
            MOVE A-YES TO DD-HASHED-OK

```

```

        ELSE
            IF DD-HASH-ST(DDX) = 2
                SET DDX UP BY 1
            ELSE
                MOVE A-YES TO DD-HASHED-IN-ERR.
DD-HASH-ERROR.
    DISPLAY "DD-HASHING ERROR ... NAME =", DD-HASH-NAME.
    PERFORM PRINT-DATA-DICT.
    ACCEPT DUMMY.
PRINT-DATA-DICT.
    MOVE HEAD-DD1 TO HEAD2.
    MOVE 0 TO LINES-LEFT.
    PERFORM PRINT-DD-LINE
        VARYING DDX FROM 1 BY 1
        UNTIL DDX > DD-MAX.
    MOVE SPACES TO HEAD2.
    MOVE 0 TO LINES-LEFT.
PRINT-DD-LINE.
    IF DD-HASH-ST(DDX) NOT = ZERO
        SET ELEM-ID TO DDX
        MOVE ELEM-ID TO PR-DDX
        MOVE DD-NAME(DDX) TO PR-DD-NAME
        MOVE DD-HASH-ST(DDX) TO PR-DD-HASH-ST
        MOVE DD-RELAT-NON(DDX) TO BC-CONVERT-B
        PERFORM BOOL-TO-CHAR
        MOVE BC-CONVERT-C TO PR-DD-NON
        MOVE DD-RELAT-KEY(DDX) TO BC-CONVERT-B
        PERFORM BOOL-TO-CHAR
        MOVE BC-CONVERT-C TO PR-DD-KEY
        MOVE PR-DD1 TO PRINT-LINE
        PERFORM PRINT-RTN.
BOOL-TO-CHAR.
    MOVE SPACES TO BC-CONVERT-C
    PERFORM CONVERT-1
        VARYING OX FROM 1 BY 1 UNTIL OX > REL-LAST.
CONVERT-1.
    COMPUTE BC-CONVERT-WORK = BC-CONVERT-B AND BOOL-1.
    IF BC-CONVERT-WORK = ZERO
        MOVE "." TO ONE-CHAR(OX)
    ELSE
        MOVE "1" TO ONE-CHAR(OX).
    SHIFT BC-CONVERT-B LEFT 1.
PRINT-RTN.
    IF LINES-LEFT = 0
        IF PAGE-NO > 1
            WRITE PRINT-REC FROM BLANKS
                AFTER ADVANCING PAGE
            PERFORM HEADING-RTN
        ELSE
            PERFORM HEADING-RTN.
    WRITE PRINT-REC FROM PRINT-LINE
        AFTER ADVANCING NO-OF LINES.
    SUBTRACT NO-OF FROM LINES-LEFT.
    MOVE 1 TO NO-OF.

```

HEADING-RTN.

MOVE PAGE-NO TO HEAD-PAGE-NO.

WRITE PRINT-REC FROM HEAD1 AFTER ADVANCING 1.

WRITE PRINT-REC FROM HEAD2 AFTER ADVANCING 1.

MOVE 2 TO NO-OF.

MOVE 56 TO LINES-LEFT.

ADD 1 TO PAGE-NO.

```

*
*   PROCESS THE NEXT TRANSACTION
*
HANDLE-TRANSACTIONS.
  DISPLAY "      STARTING NEW TRANSACTION      (REPORT) "
  MOVE TRANS-NAME TO TRANSACTION-NAME.
  MOVE SPACES TO TEMP-STRING.
  STRING "          TRANSACTION-NAME:      "
        TRANSACTION-NAME
        DELIMITED SIZE
        INTO TEMP-STRING.
  DISPLAY TEMP-STRING.
  MOVE TEMP-STRING TO PRINT-LINE.
  MOVE 2 TO NO-OF.
  PERFORM PRINT-RTN.
  SET TRX      TO 1.
  MOVE ZERO TO TT-LAST.
  PERFORM GET-TRANS-REC.
  PERFORM BUILD-TRANS-TABLE
        VARYING LNX FROM 1 BY 1
        UNTIL TRANS-LEVEL = ZERO OR EOF-TRANS = A-YES.
  SET LNMAX TO LNX.
  SUBTRACT 1 FROM LNMAX.
  IF TR-SLOW = A-YES
        MOVE LNMAX TO DISPLAY-NUM
        DISPLAY " LNMAX = " DISPLAY-NUM.
  PERFORM TRANS-DEF-START.
BUILD-TRANS-TABLE.
  IF TR-SLOW = A-YES
        DISPLAY "      HEY!!      LINK HAS CHANGED.....".
  MOVE TRANS-LINK TO LINE-LINK(LNX).
  SET LINE-1ST(LNX) TO TRX.
  SET DISPLAY-NUM TO TRX
  IF TR-SLOW = A-YES
        DISPLAY "LINE-1ST() = " DISPLAY-NUM.
  MOVE ZERO TO NO-DETERMINANTS.
  PERFORM BUILD-TRANS-ENTRY
        UNTIL TRANS-LINK NOT = LINE-LINK(LNX)
        OR TRANS-LEVEL = ZERO OR EOF-TRANS = A-YES.
  SET LINE-STOP(LNX) TO TRX.
  MOVE NO-DETERMINANTS TO LINE-NO-DETERMINANTS(LNX).
  IF TR-SLOW = A-YES
        SET DISPLAY-NUM TO TRX
        DISPLAY "LINE-STOP() =" DISPLAY-NUM
        DISPLAY "# OF DETERMINANTS =" NO-DETERMINANTS.
BUILD-TRANS-ENTRY.
  IF TRANS-KEY-FLAG = "3"
        MOVE TRANS-NAME TO TEMPORARY-NAME
        PERFORM TEMP-TABLE-ENTRY-RTN
        SET TR-ID(TRX) TO TTX
  ELSE
        MOVE TRANS-NAME TO DD-HASH-NAME
        PERFORM DD-HASH-IT
        IF DD-HASHED-IN-ERR = A-YES

```

```

        PERFORM DD-HASH-ERROR
    ELSE
        SET TR-ID(TRX) TO DDX.
    IF TR-SLOW = A-YES
        SET DISPLAY-NUM TO TRX
        DISPLAY "TRX ="          DISPLAY-NUM
        DISPLAY "NAME=" TRANS-NAME.
    MOVE TRANS-KEY-FLAG TO TR-KEY-FLAG(TRX).
    IF TRANS-KEY-FLAG = "1"
        ADD 1 TO NO-DETERMINANTS.
    SET TRX UP BY 1.
    PERFORM GET-TRANS-REC.
GET-TRANS-REC.
    IF TRANS-PRINT-SW = A-YES
        STRING "                " TRANS-REC  BLANKS
            DELIMITED SIZE
            INTO PRINT-LINE
        PERFORM PRINT-RTN.
    READ NAMETABLE INTO TRANS-REC
        AT END MOVE A-YES TO EOF-TRANS.
    IF TR-SLOW = A-YES
        DISPLAY " TRANS-REC = " TRANS-REC
        ACCEPT DUMMY.
TEMP-TABLE-ENTRY-RTN.
    ADD 1 TO TT-LAST.
    SET TTX TO TT-LAST.
    MOVE TEMPORARY-NAME TO TT-NAME(TTX).
    IF TR-SLOW = A-YES
        STRING "..TEMPORARY DATA..  *DATA NAME = "
            TEMPORARY-NAME  BLANKS
            DELIMITED SIZE
            INTO MSG
        DISPLAY MSG.
RETRIEVE-ELEMENT-DATA.
    IF LE-TEMPORARY(LEX) = "3"
        MOVE TT-NAME(TTX) TO TEMPORARY-NAME.

```

```

*
*   START TO DEFINE THIS TRANSACTION
*
TRANS-DEF-START.
*   DISPLAY "WHAT KIND OF TRANSACTION IS THIS?"
*   "DISPLAY, UPDATE, DELETE, OR ADD".
*   ACCEPT ANS
*   IF ANS2 = "DI"
*       PERFORM DISPLAY-DEF
*   ELSE
*       DISPLAY "SORRY, NOT YET IMPLEMENTED".
*
*   PERFORM DISPLAY-DEF.
*
*
DISPLAY-DEF.
    MOVE "DISPLAY"      TO FUNCTION.
    IF DI-SLOW = A-YES
        DISPLAY " * * * * * "
        "DISPLAY"
        " * * * * * ".
    SET TRX TO 1
    PERFORM DEFINE-NEXT-LINE
        VARYING LNX FROM 1 BY 1
        UNTIL LNX > LNMAX.
*
*   DEFINE NEXT LINE (WHICH DISPLAYED, RESTRICTION,ETC)
*   PROCESS DISPLAY TRANSACTIONS
*   (THE ONLY TYPE PRESENTLY HANDLED)
*
DEFINE-NEXT-LINE.
    MOVE 1 TO COM-PTR.
    PERFORM BUILD-LINE-ELEM-TABLE.
    PERFORM FIND-PRIMARY-RELATION.
    PERFORM IDENTIFY-SOURCE-RELATIONS.
    PERFORM IDENTIFY-DISPLAY-ELEMENTS.
    PERFORM DATA-SELECTION-CRITERIA.
    PERFORM BUILD-SELECT-CLAUSE.
    PERFORM SORT-CLAUSE-BUILD.
    PERFORM UNIQUE-ALL-CLAUSE-BUILD.
    PERFORM PRINT-COMPUTE-LINES.
    MOVE 3 TO NO-OF.
BUILD-LINE-ELEM-TABLE.
    SET LEX TO 1.
    SET TRX TO LINE-1ST(LNX)
    PERFORM BUILD-LINE-DD-ELEMENTS
        VARYING TRX FROM TRX BY 1
        UNTIL TRX = LINE-STOP(LNX).
    SET TRX TO LINE-1ST(LNX)
    IF DI-SLOW = A-YES
        DISPLAY "      END OF NON-TEMPORARY ORIGINALS "
        SET DISPLAY-NUM TO LEX
        DISPLAY " LEX ="  DISPLAY-NUM.

```

```

PERFORM BUILD-LINE-ADDED-ELEMENTS
    VARYING TRX FROM TRX BY 1
    UNTIL TRX = LINE-STOP(LNX).
PERFORM ADD-MULTI-LEVEL-KEYS.
SET LEX DOWN BY 1.
SET LE-END-IN-DD TO LEX.
IF DI-SLOW = A-YES
    DISPLAY "      END OF NON-TEMPORARY NON-ORIGIONALS"
    SET DISPLAY-NUM TO LEX
    DISPLAY " LEX ="  DISPLAY-NUM.
SET LEX UP BY 1.
SET TRX TO LINE-1ST(LNX)
PERFORM BUILD-LINE-TEMP-ELEMENTS
    VARYING TRX FROM TRX BY 1
    UNTIL TRX = LINE-STOP(LNX).
SET LEX DOWN BY 1.
SET LE-LAST TO LEX.
IF DI-SLOW = A-YES
    DISPLAY "      END OF TEMPORARIES"
    SET DISPLAY-NUM TO LEX
    DISPLAY " LEX ="  DISPLAY-NUM.
BUILD-LINE-DD-ELEMENTS.
    IF TR-KEY-FLAG(TRX) NOT = "3"
        PERFORM BUILD-LINE-ORIG-ELEMENTS
        IF DI-SLOW = A-YES
            SET DDX TO TR-ID(TRX)
            DISPLAY " LINE ELEMENT ID FOR" DD-NAME(DDX).
BUILD-LINE-ORIG-ELEMENTS.
    SET LE-ID(LEX) TO TR-ID(TRX).
    IF TR-KEY-FLAG(TRX) = "1"
        MOVE A-YES TO LE-KEY-FLAG(LEX)
    ELSE
        MOVE A-NO TO LE-KEY-FLAG(LEX).
    IF TR-KEY-FLAG(TRX) = "3"
        MOVE A-YES TO LE-TEMPORARY(LEX)
    ELSE
        MOVE A-NO TO LE-TEMPORARY(LEX).
    MOVE A-NO TO LE-PROCESS(LEX).
    MOVE A-NO TO LE-FUNCTION(LEX).
    MOVE A-YES TO LE-ORIGIONAL(LEX).
    MOVE A-NO TO LE-SORT(LEX).
    MOVE ZERO TO LE-NAV-RELATION(LEX).
    SET LEX UP BY 1.
BUILD-LINE-TEMP-ELEMENTS.
    IF TR-KEY-FLAG(TRX) = "3"
        PERFORM BUILD-LINE-ORIG-ELEMENTS.
BUILD-LINE-ADDED-ELEMENTS.
    IF TR-KEY-FLAG(TRX) = "3"
        PERFORM HANDLE-TEMP-DATA
        PERFORM BUILD-LINE-ADDED-ELEMENTS-1
            VARYING ADX FROM 1 BY 1
            UNTIL ADX > AD-LAST.
BUILD-LINE-ADDED-ELEMENTS-1.
    SET LE-ID(LEX) TO ADDED-ID(ADX).

```

```

MOVE A-NO TO LE-KEY-FLAG(LEX).
MOVE A-NO TO LE-TEMPORARY(LEX).
MOVE A-NO TO LE-PROCESS(LEX).
MOVE A-NO TO LE-FUNCTION(LEX).
MOVE A-NO TO LE-ORIGINAL(LEX).
MOVE A-NO TO LE-SORT(LEX).
MOVE ZERO TO LE-NAV-RELATION(LEX).
IF DI-SLOW = A-YES
    SET DDX TO LE-ID(LEX)
    DISPLAY " LINE ELEMENT ID FOR" DD-NAME(DDX).
SET LEX UP BY 1.
FOLLOW-NAVIGATION-PATHS.
IF PR-SLOW = A-YES
    DISPLAY
        "< NAVIGATION PATHS TO FIND PRIMARY RELATION >".
SET WSX TO WS-LAST.
SUBTRACT 1 FROM WS-LAST.
SET RELX TO WHERE-STACK-REL(WSX).
IF REL-USED(RELX) = 1
    MOVE 2 TO REL-USED(RELX)
    SET RPX TO REL-1ST(RELX)
    MOVE A-NO TO DEAD-END
    PERFORM FOLLOW-A-PATH
        VARYING REL-SS FROM 1 BY 1
        UNTIL REL-SS > REL-LHS(RELX)
        OR DEAD-END = A-YES.
FOLLOW-A-PATH.
SET DDX TO RP-ELEM-ID(RPX).
IF PR-SLOW = A-YES
    SET REL-NO TO RELX
    STRING " STARTING FROM R" REL-NO " NAME: "
        DD-NAME(DDX) BLANKS
        DELIMITED SIZE
        INTO MSG
    DISPLAY MSG.
PERFORM BUILD-A-PATH.
SET RPX UP BY 1.
NOT-IN-PRIMARY-RELATION.
SUBTRACT 1 FROM LINE-NO-DETERMINANTS(LNX)
MOVE "0" TO LE-KEY-FLAG(LEX).
*
*
* ADD ELEMENTS TO THIS LINE TO ESTABLISH NAVIGATION TO
* THE KEYS OF PRECEEDING LOWER-LEVEL LINES
*
*
ADD-MULTI-LEVEL-KEYS.
IF TR-SLOW = A-YES AND LINE-LEVEL(LNX) NOT = "1"
    DISPLAY "GOING TO ADD KEYS FROM PREVIOUS LINE".
IF LINE-LEVEL(LNX) NOT = "1"
    SET LNX-HOLD TO LNX
    PERFORM ADD-NEXT-LEVEL-KEYS
        UNTIL LINE-LEVEL(LNX) = "1"
    SET LNX TO LNX-HOLD.

```



```

      IF TR-SLOW = A-YES
        DISPLAY "ADDED KEYS OF PREVIOUS PRIMARY RELATION".
      ADD-NEXT-LEVEL-KEYS.
      MOVE LINE-LEVEL(LNX) TO LAST-LEVEL.
      PERFORM FIND-NEXT-LEVEL
        VARYING LNX FROM LNX BY -1
        UNTIL LINE-LEVEL(LNX) < LAST-LEVEL.
      SET RELX TO LINE-PRIMARY-RELATION(LNX).
      SET REL-NO TO RELX.
      SET RPX TO REL-1ST(RELX).
      PERFORM ADD-NEXT-KEY-IF-ABSENT REL-LHS(RELX) TIMES.
      FIND-NEXT-LEVEL.
*      THIS IS A NULL PARAGRAPH, BUT DON'T DELETE IT
      ADD-NEXT-KEY-IF-ABSENT.
      MOVE A-YES TO ABSENT.
      SET LEX-HOLD TO LEX.
      PERFORM LOOK-FOR-IN-LINE
        VARYING LEX FROM 1 BY 1
        UNTIL LEX = LEX-HOLD OR ABSENT = A-NO.
      SET LEX TO LEX-HOLD.
      IF ABSENT = A-YES
        PERFORM ADD-NEXT-KEY.
      SET RPX UP BY 1.
      LOOK-FOR-IN-LINE.
      IF LE-ID(LEX) = RP-ELEM-ID(RPX)
        MOVE REL-NO TO LE-NAV-RELATION(LEX)
        MOVE A-NO TO ABSENT.
      ADD-NEXT-KEY.
      SET LE-ID(LEX) TO RP-ELEM-ID(RPX).
      MOVE A-NO TO LE-TEMPORARY(LEX).
      MOVE A-YES TO LE-PROCESS(LEX).
      MOVE A-NO TO LE-FUNCTION(LEX).
      MOVE A-NO TO LE-ORIGIONAL(LEX).
      MOVE A-NO TO LE-SORT(LEX).
      MOVE REL-NO TO LE-NAV-RELATION(LEX).
      IF DI-SLOW = A-YES
        SET DDX TO LE-ID(LEX)
        DISPLAY " LINE ELEMENT ID FOR" DD-NAME(DDX).
      SET LEX UP BY 1.
      SET RPX UP BY 1.

```

```

*
*   PROCESS TEMPORARY (COMPUTED) DATA
*
HANDLE-TEMP-DATA.
    MOVE ZERO TO AD-LAST.
    SET ADX TO 1.
    SET ADX DOWN BY 1.
    SET TTX TO TR-ID(TRX).
    STRING      "COMPUTE " TT-NAME(TTX) " FROM "
                DELIMITED SIZE
                INTO COMPUTE-POOL
                POINTER COM-PTR.
*   STRING " IS <" DELIMITED SIZE
*   TT-NAME(TTX) DELIMITED SPACES
*   "> A GROUPING-TYPE ELEMENT" BLANKS
*   DELIMITED SIZE
*   INTO MSG
*   DISPLAY MSG.
*   DISPLAY "(IE, IS IT A TOTAL, MAX, MIN, COUNT OR AVG)?"
*   ACCEPT ANS1.
*   IF ANS1 = A-YES
*       PERFORM HANDLE-TEMP-GROUPING-TYPE
*   ELSE
*       PERFORM HANDLE-TEMP-CALC.
    STRING ""
                DELIMITED SIZE
                INTO COMPUTE-POOL
                POINTER COM-PTR.
HANDLE-TEMP-GROUPING-TYPE.
    DISPLAY "PLEASE ENTER THE TYPE."
    ACCEPT GROUP-FUNCTION
    STRING HAT IS IT A " DELIMITED SIZE
                GROUP-FUNCTION DELIMITED SPACES
                " OF" BLANKS DELIMITED SIZE
                INTO MSG
    DISPLAY MSG
    MOVE A-NO TO GOOD-RESPONSE.
    PERFORM ACCEPT-ELEMENT-NAME
        UNTIL GOOD-RESPONSE = A-YES
    MOVE DD-HASH-NAME TO GROUP-ELEMENT
    STRING DD-HASH-NAME " "
                DELIMITED SIZE
                INTO COMPUTE-POOL
                POINTER COM-PTR.
    SET ADX TO 1
    SET ADDED-ID(ADX) TO DDX
    DISPLAY "WHICH ELEMENT IS THE CONTROL?"
    DISPLAY "(IE, WHICH ELEMENT SHOULD CAUSE A BREAK?)".
    MOVE A-NO TO GOOD-RESPONSE
    PERFORM ACCEPT-ELEMENT-NAME
        UNTIL GOOD-RESPONSE = A-YES
        OR DD-HASH-NAME = SPACES
    IF DD-HASH-NAME NOT = SPACES
        MOVE DD-HASH-NAME TO GROUP-CONTROL

```

```

        STRING DD-HASH-NAME " "
            DELIMITED SIZE
            INTO COMPUTE-POOL
            POINTER COM-PTR
        SET ADX TO 2
        SET ADDED-ID(ADX) TO DDX.
HANDLE-TEMP-CALC.
    DISPLAY "PLEASE ENTER NAMES OF EACH ELEMENT INVOLVED"
    STRING " IN COMPUTING <" DELIMITED SIZE
        TT-NAME(TTX) DELIMITED SPACES
        ">" BLANKS DELIMITED SIZE
    INTO MSG
    DISPLAY MSG.
    MOVE A-NO TO LAST-RESPONSE.
    PERFORM ACCEPT-ALL-ELEMENT-NAMES
        UNTIL LAST-RESPONSE = A-YES.
ACCEPT-ALL-ELEMENT-NAMES.
    MOVE A-NO TO GOOD-RESPONSE
    PERFORM ACCEPT-ELEMENT-NAME
        UNTIL GOOD-RESPONSE = A-YES.
    IF DD-HASH-NAME NOT = SPACES
        STRING DD-HASH-NAME " "
            DELIMITED SIZE
            INTO COMPUTE-POOL
            POINTER COM-PTR
        SET ADX UP BY 1
        SET ADDED-ID(ADX) TO DDX.
ACCEPT-ELEMENT-NAME.
    ACCEPT DD-HASH-NAME
    IF DD-HASH-NAME = SPACES
        MOVE A-YES TO LAST-RESPONSE
        MOVE A-YES TO GOOD-RESPONSE
    ELSE
        PERFORM DD-HASH-IT
        IF DD-HASHED-OK = A-NO
            DISPLAY "NOT IN DATABASE"
        ELSE
            SET TRX-HOLD TO TRX
            SET TRX TO LINE-1ST(LNX)
            PERFORM CHECK-LINE
                VARYING TRX FROM TRX BY 1
                UNTIL TRX = LINE-STOP(LNX)
                OR GOOD-RESPONSE = A-YES
            SET TRX TO TRX-HOLD
            IF GOOD-RESPONSE = A-NO
                DISPLAY "NOT IN THE LINE, TRY AGAIN".
CHECK-LINE.
    IF TR-KEY-FLAG(TRX) NOT = "3"
        SET DDX TO TR-ID(TRX)
        IF DD-HASH-NAME = DD-NAME(DDX)
            MOVE A-YES TO GOOD-RESPONSE.

```

```

IDENTIFY-SOURCE-RELATIONS.
  PERFORM IDENTIFY-NEXT-SOURCE
    VARYING LEX FROM 1 BY 1
    UNTIL LEX > LE-END-IN-DD.
IDENTIFY-NEXT-SOURCE.
  IF DI-SLOW = A-YES
    DISPLAY " STARTING TO BUILD FULL-NAMES"
    SET DISPLAY-NUM TO LEX
    DISPLAY "LEX =" DISPLAY-NUM
    PERFORM GET-LINE-ELEM-NAME
    DISPLAY "ELEMENT-NAME ="
      ELEMENT-NAME
    ACCEPT DUMMY.
  SET DDX TO LE-ID(LEX)
  COMPUTE B-WORK0 = BOOL-PRIMARY AND DD-RELAT-KEY(DDX)
  IF B-WORK0 = ZERO
    PERFORM SFW-1
  ELSE
    MOVE PRIMARY-RELATION TO REL-NO
    MOVE REL-NO TO LE-RELATION(LEX).

```

```

IDENTIFY-DISPLAY-ELEMENTS.
    PERFORM LINE-DISPLAY.
    DISPLAY "WHICH DO YOU WANT TO DISPLAY"
        "    Y - YES, THIS ITEM"
        "    N - NOT, THIS ITEM"
        "    A - ALL OF THE REST OF THE ITEMS"
        "    S - STOP, NONE OF THE REST OF THE ITEMS".
    ADD 1 TO NEXT-LINE.
    MOVE SPACES TO ANS.
    PERFORM ELEMENT-QUESTION
        VARYING LEX FROM 1 BY 1
        UNTIL LEX > LE-LAST.
    MOVE SPACES TO ANS1.
LINE-DISPLAY.
    SET LEX TO 1.
    IF DI-SLOW = A-YES
        MOVE LE-LAST TO DISPLAY-NUM
        DISPLAY "STARTING DISPLAY LIST..... LE-LAST ="
            DISPLAY-NUM
        ACCEPT DUMMY.
    PERFORM ONE-LINE
        UNTIL LEX > LE-LAST.
ONE-LINE.
    MOVE SPACES TO DISPLAY-LINE-DATA.
    MOVE 1 TO SS1.
    PERFORM LINE-SETUP
        UNTIL SS1 > 6 OR LEX > LE-LAST.
    DISPLAY DISPLAY-LINE-DATA.
LINE-SETUP.
    PERFORM GET-LINE-ELEM-NAME
    IF LE-ORIGIONAL(LEX) = A-YES
        MOVE ELEMENT-NAME TO DISP-LINE-NAME (SS1)
        ADD 1 TO SS1.
    IF TR-SLOW = A-YES
        SET DISPLAY-NUM TO LEX
        DISPLAY "LEX ="          DISPLAY-NUM
        DISPLAY "ELEMENT-NAME =" ELEMENT-NAME.
    SET LEX UP BY 1.
ELEMENT-QUESTION.
    IF DI-SLOW = A-YES
        DISPLAY "  QUESTION TIME!!!!  "
        SET DISPLAY-NUM TO LEX
        DISPLAY "LEX ="          DISPLAY-NUM
        PERFORM GET-LINE-ELEM-NAME
        DISPLAY "ELEMENT-NAME =" ELEMENT-NAME
        ACCEPT DUMMY.
    PERFORM GET-LINE-ELEM-NAME
    IF ANS1 NOT = "S" AND ANS1 NOT = "A"
        DISPLAY ELEMENT-NAME
        ACCEPT ANS1.
    IF ANS1 = "Y" OR ANS1 = "A"
        MOVE A-YES TO LE-FUNCTION(LEX)
        MOVE A-YES TO LE-PROCESS(LEX).

```

```

*
*   FIND THE RELATION WHICH COVERS ALL OTHER RELATIONS
*   INVOLVED IN THE LINE
*
FIND-PRIMARY-RELATION.
  IF PR-SLOW = A-YES
    DISPLAY " "
    " PRIMARY RELATION ".
  MOVE A-NO TO FOUND-IT.
  PERFORM FIND-PRIMARY-0.
  IF FOUND-IT = A-NO
    PERFORM FIND-PRIMARY-3.
  MOVE PRIMARY-RELATION TO LINE-PRIMARY-RELATION(LNX).
  IF PR-SLOW = A-YES
    MOVE BOOL-PRIMARY TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " BOOL-PRIMARY =" BC-CONVERT-C
    MOVE PRIMARY-RELATION TO DISPLAY-NUM
    DISPLAY "!!!!!!!!!!!! FANFARE !!!!!!!!!!!!!!"
    "THE PRIMARY RELATION IS....."
    DISPLAY-NUM
    ACCEPT DUMMY.
FIND-PRIMARY-0.
  MOVE BOOL-ALL-1 TO B-AND.
  MOVE ZEROS TO B-OR
  MOVE ZERO TO ONE-BIT.
  PERFORM FIND-PRIMARY-1
    VARYING LEX FROM 1 BY 1
    UNTIL LEX > LE-END-IN-DD.
  IF PR-SLOW = A-YES
    MOVE B-AND TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " B-AND =" BC-CONVERT-C
    MOVE B-OR TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " B-OR =" BC-CONVERT-C.
*
*   IF B-AND NOT = ZERO, THE PRIMARY RELATION IS
*   REPRESENTED BY ONE OF THOSE BITS, AND IT CAN
*   BE DISCOVERED WITHOUT FOLLOWING
*   NAVIGATION ROUTES.
*
  IF B-AND NOT = ZERO
    MOVE BOOL-1 TO ONE-BIT
    PERFORM FIND-PRIMARY-2
      VARYING SS1 FROM 1 BY 1
      UNTIL SS1 > REL-LAST OR FOUND-IT = A-YES.
FIND-PRIMARY-1.
  IF PR-SLOW = A-YES
    DISPLAY " COMPUTING B-AND & B-OR".
  SET DDX TO LE-ID(LEX).
  IF PR-SLOW = A-YES
    SET DISPLAY-NUM TO DDX
    DISPLAY "DDX =" DISPLAY-NUM DD-NAME(DDX).

```

```

*
*      NOW ELIMINATE SUPERFLUOUS DETERMINATES, THOSE
*      WHICH ARE NOT IN THE KEY OF ANY RELATION.
*
IF LE-KEY-FLAG(LEX) = A-YES AND
    DD-RELAT-KEY(DDX) = ZERO
    PERFORM NOT-IN-PRIMARY-RELATION.
*
*      FIND THE COVERAGE OF THOSE REAL DETERMINATES AND
*      OTHERS INVOLVED IN TRANSITIVE DEPENDENCIES.
*
IF LE-KEY-FLAG(LEX) = A-YES
    COMPUTE B-AND = B-AND AND DD-RELAT-KEY(DDX)
    COMPUTE B-OR = B-OR OR DD-RELAT-KEY(DDX).
IF PR-SLOW = A-YES
    MOVE DD-RELAT-KEY(DDX) TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " DD-RELAT-KEY(DDX) =" BC-CONVERT-C
    MOVE B-AND TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " B-AND =" BC-CONVERT-C
    MOVE B-OR TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " B-OR =" BC-CONVERT-C
    ACCEPT DUMMY.
FIND-PRIMARY-2.
*
*      STARTING FROM A LIST OF CANDIDATE RELATIONS, EACH
*      WITH A SET OF KEYS WHICH INCLUDES ALL OF THE ACTUAL
*      DETERMINANTS OF THE LINE, THE PRIMARY RELATION CAN
*      BE FOUND BY SIMPLY COMPARING # OF KEYS TO THE # OF
*      DETERMINANTS OF THE LINE.
*
COMPUTE B-WORK0 = ONE-BIT AND B-AND.
IF B-WORK0 NOT = ZERO
    SET RELX TO SS1
    IF REL-LHS(RELX) = LINE-NO-DETERMINANTS(LNX)
        MOVE A-YES TO FOUND-IT
        MOVE SS1 TO PRIMARY-RELATION
        MOVE B-WORK0 TO BOOL-PRIMARY.
SHIFT ONE-BIT RIGHT 1.
IF PR-SLOW = A-YES
    DISPLAY "LOOKING FOR THE RIGHT # DETERMINANTS"
    MOVE B-WORK0 TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " B-WORK0 =" BC-CONVERT-C
    MOVE B-AND TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " B-AND =" BC-CONVERT-C
    MOVE REL-LHS(RELX) TO DISPLAY-NUM
    DISPLAY "REL-LHS =" DISPLAY-NUM
    MOVE LINE-NO-DETERMINANTS(LNX) TO DISPLAY-NUM
    DISPLAY "LINE-NO-DETERM. =" DISPLAY-NUM
    DISPLAY "FOUND-IT =" FOUND-IT

```

```

ACCEPT DUMMY.
FIND-PRIMARY-3.
*
*   START THE PROCESS OF ELIMINATING SUPPOSED
*   DETERMINANTS WHICH ARE ACTUALLY INVOLVED THROUGH
*   TRANSITIVE DEPENDENCIES.
*
PERFORM FIND-PRIMARY-4
    VARYING LEX FROM 1 BY 1
    UNTIL LEX > LE-END-IN-DD.
PERFORM FIND-PRIMARY-5
    VARYING LEX FROM 1 BY 1
    UNTIL LEX > LE-END-IN-DD.
PERFORM FIND-PRIMARY-0.
IF FOUND-IT = A-NO
    MOVE ZERO TO CANDIDATES-LEFT
    MOVE BOOL-1 TO B-WORK0
    PERFORM FIND-PRIMARY-6
        VARYING RELX FROM 1 BY 1
        UNTIL RELX > REL-LAST
    IF CANDIDATES-LEFT > 1
        PERFORM FIND-PRIMARY-9.
IF PR-SLOW = A-YES
    MOVE B-WORK0 TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " B-WORK0 =" BC-CONVERT-C.
FIND-PRIMARY-4.
*
*   ANY SUPPOSED DETERMINANT WHICH DOESN'T APPEAR IN
*   ANY RELATION AS A NON-KEY, MUST INDEED BE PART OF
*   THE ACTUAL DETERMINANT OF THE LINE
*
IF LE-KEY-FLAG(LEX) = A-YES
    PERFORM GET-LINE-ELEM-NAME
    IF DD-RELAT-NON(DDX) = ZERO
        COMPUTE B-OR = B-OR AND DD-RELAT-KEY(DDX).
FIND-PRIMARY-5.
*
*   ELIMINATE THOSE SUPPOSED DETERMINANTS WHICH NEVER
*   APPEAR IN THE KEY OF A RELATION WHICH HAS THOSE
*   ELEMENTS IN ITS KEY WHICH WERE IDENTIFIED AS BEING
*   IN THE ACTUAL DETERMINANT OF THE LINE
*
IF LE-KEY-FLAG(LEX) = A-YES
    PERFORM GET-LINE-ELEM-NAME
    COMPUTE B-WORK0 = B-OR AND DD-RELAT-KEY(DDX)
    IF B-WORK0 = ZERO
        PERFORM NOT-IN-PRIMARY-RELATION.
FIND-PRIMARY-6.
*
*   ELIMINATE THOSE CANDIDATE RELATIONS WHICH INCLUDE
*   IN THEIR KEY, ELEMENTS WHICH AREN'T IN THE LINE'S
*   DETERMINANTS
*

```



```

MOVE A-NO TO REL-CANDIDATE(RELX).
COMPUTE B-WORK1 = B-WORK0 AND B-OR.
IF B-WORK1 NOT = ZERO
    MOVE A-YES TO GOOD-CANDIDATE
    SET RPX TO REL-1ST(RELX)
    PERFORM FIND-PRIMARY-7
        VARYING SS1 FROM 1 BY 1
        UNTIL SS1 > REL-LHS(RELX)
        OR GOOD-CANDIDATE = A-NO
    IF GOOD-CANDIDATE = A-YES
        MOVE A-YES TO REL-CANDIDATE(RELX)
        ADD 1 TO CANDIDATES-LEFT
        IF CANDIDATES-LEFT = 1
            MOVE B-WORK0 TO BOOL-PRIMARY
            SET PRIMARY-RELATION TO RELX.
    SHIFT B-WORK0 RIGHT 1.
FIND-PRIMARY-7.
*
*     LOOK FOR EACH OF THE ELEMENTS OF THIS RELATION'S KEY
*     IN THE LINE'S DETERMINANTS
*
MOVE A-NO TO GOOD-CANDIDATE.
PERFORM FIND-PRIMARY-8
    VARYING LEX FROM 1 BY 1
    UNTIL LEX > LE-END-IN-DD
    OR GOOD-CANDIDATE = A-YES.
SET RPX UP BY 1.
FIND-PRIMARY-8.
*
*     CHECK THIS DETERMINANT AGAINST AN ELEMENT IN A KEY
*
IF LE-KEY-FLAG(LEX) = A-YES
IF LE-ID(LEX) = RP-ELEM-ID(RPX)
    MOVE A-YES TO GOOD-CANDIDATE.
IF PR-SLOW = A-YES
    SET DISPLAY-NUM TO RELX
    SET DDX TO RP-ELEM-ID(RPX)
    STRING "RELATION # = " DISPLAY-NUM
        "    RELATION KEY: " DD-NAME(DDX)
        BLANKS
        DELIMITED SIZE
        INTO MSG
    DISPLAY MSG
    PERFORM GET-LINE-ELEM-NAME
    STRING "    DETERMINANT NAME: " DD-NAME(DDX)
        "    GOOD-CANDIDATE: " GOOD-CANDIDATE
        BLANKS
        DELIMITED SIZE
        INTO MSG
    DISPLAY MSG.
FIND-PRIMARY-9.
*
*     BITE THE BULLET - START TO FOLLOW NAVIGATION PATHS
*

```

```

IF PR-SLOW = A-YES
    DISPLAY "*** NAVIGATION FOR PRIMARY RELATION ***".
MOVE BOOL-1 TO B-WORK0.
MOVE ZERO TO LAST-CANDIDATE.
PERFORM FIND-PRIMARY-11
    VARYING RELX FROM 1 BY 1
    UNTIL RELX > REL-LAST OR CANDIDATES-LEFT = 1.
SET RELX DOWN BY 1.
SET PRIMARY-RELATION TO RELX.
SHIFT B-WORK0 LEFT 1.
MOVE B-WORK0 TO BOOL-PRIMARY.
FIND-PRIMARY-11.
*
*     ONCE PER RELATION
*
IF PR-SLOW = A-YES
    SET REL-NO TO RELX
    STRING ".....STARTING RELATION R" REL-NO
        BLANKS
        DELIMITED SIZE
        INTO MSG
    DISPLAY MSG
    ACCEPT DUMMY.
SHIFT B-WORK0 RIGHT 1.
IF REL-CANDIDATE(RELX) = A-YES
    PERFORM INIT-DD-COVERED
        VARYING DDX FROM 1 BY 1
        UNTIL DDX > DD-MAX
    SET RPX TO REL-1ST(RELX)
    ADD REL-LHS(RELX) REL-RHS(RELX)
        GIVING ELEMENT-COUNT
    PERFORM MARK-ELEMENTS-COVERED
        ELEMENT-COUNT TIMES
    SET RELX-HOLD TO RELX
    PERFORM ZERO-RELATION-TEMPS
        VARYING RELX FROM 1 BY 1
        UNTIL RELX > REL-LAST
    MOVE A-YES TO REPEAT
    PERFORM FIND-PRIMARY-12
        UNTIL REPEAT = A-NO
    SET RELX TO RELX-HOLD.
FIND-PRIMARY-12.
MOVE A-NO TO REPEAT.
PERFORM FIND-PRIMARY-13
    VARYING RELX FROM 1 BY 1
    UNTIL RELX = RELX-HOLD.
SET RELX UP BY 1.
PERFORM FIND-PRIMARY-13
    VARYING RELX FROM RELX BY 1
    UNTIL RELX > REL-LAST.
FIND-PRIMARY-13.
IF REL-USED(RELX) = ZERO
    PERFORM FIND-PRIMARY-13A.
FIND-PRIMARY-13A.

```

```

IF PR-SLOW = A-YES
  DISPLAY "TRYING TO COVER NEXT RELATION"
  SET DISPLAY-NUM TO RELX
  STRING "RELATION  $\frac{1}{2}$  = " DISPLAY-NUM
    "      CANDIDATE = " REL-CANDIDATE(RELX)
    "      CANDIDATES LEFT = " CANDIDATES-LEFT
  BLANKS
  DELIMITED SIZE
  INTO MSG
  DISPLAY MSG.
SET RPX TO REL-1ST(RELX).
MOVE A-YES TO COVERED-KEYS.
PERFORM FIND-PRIMARY-14
  VARYING SS1 FROM 1 BY 1
  UNTIL SS1 > REL-LHS(RELX)
    OR COVERED-KEYS = A-NO.
IF COVERED-KEYS = A-YES
  SET RPX TO REL-1ST(RELX)
  ADD REL-LHS(RELX) REL-RHS(RELX)
  GIVING ELEMENT-COUNT
  PERFORM MARK-ELEMENTS-COVERED
    ELEMENT-COUNT TIMES
  MOVE A-YES TO REPEAT
  MOVE 1 TO REL-USED(RELX)
  IF REL-CANDIDATE(RELX) = A-YES
    MOVE A-NO TO REL-CANDIDATE(RELX)
    SUBTRACT 1 FROM CANDIDATES-LEFT.
IF PR-SLOW = A-YES
  DISPLAY "FINISHED"
  STRING
    "      CANDIDATE = " REL-CANDIDATE(RELX)
    "      CANDIDATES LEFT = " CANDIDATES-LEFT
  BLANKS
  DELIMITED SIZE
  INTO MSG
  DISPLAY MSG.
FIND-PRIMARY-14.
SET DDX TO RP-ELEM-ID(RPX)
IF DD-COVERED(DDX) = A-NO
  MOVE A-NO TO COVERED-KEYS.
SET RPX UP BY 1.
INIT-DD-COVERED.
MOVE A-NO TO DD-COVERED(DDX).
MARK-ELEMENTS-COVERED.
SET DDX TO RP-ELEM-ID(RPX).
MOVE A-YES TO DD-COVERED(DDX).
SET RPX UP BY 1.

```

```

*
*   IDENTIFY ADDITIONAL DATA SELECTION CRITERIA
*   [EG SALARY < 9000 OR JOBNAME = 'PROGRAMMER']
*
DATA-SELECTION-CRITERIA.
    MOVE 1 TO WHERE-PTR
    DISPLAY
        "DO YOU WANT THIS TRANSACTION TO INVOLVE EVERY"
        "OCCURRENCE OF THE DATA FOR THIS LINE? (Y/N) "
        "('N' IF THERE IS SELECTION CRITERIA)".
    ACCEPT ANS1.
    IF ANS1 = A-NO
        MOVE A-YES TO MORE-RESTRICTIONS
        PERFORM GET-NEXT-RESTRICTION
            VARYING PARM FROM 1 BY 1
            UNTIL MORE-RESTRICTIONS = A-NO.
GET-NEXT-RESTRICTION.
    DISPLAY "OF THESE ELEMENTS IN THE LINE:"
    PERFORM LINE-DISPLAY.
    DISPLAY "WHICH IS INVOLVED IN THIS CONDITION?".
    MOVE A-NO TO GOOD-RESPONSE.
    PERFORM ACCEPT-DATA-NAME
        UNTIL GOOD-RESPONSE = A-YES.
    IF TEST-NAME NOT = SPACES
        PERFORM BUILD-CONDITION
    ELSE
        MOVE A-NO TO MORE-RESTRICTIONS.
BUILD-CONDITION.
    DISPLAY
        "PLEASE ENTER CONDITION OPERATOR (=, <, >, ETC)"
    ACCEPT ANS2.
    DISPLAY
        "FOR NOW, THE 'RIGHT-HAND SIDE' WILL COME FROM"
        "THE CRT".
    STRING "ACCEPT PARAMETER-" PARM " FROM CRT" BLANKS
        DELIMITED SIZE
        INTO PRINT-LINE.
    PERFORM PRINT-RTN.
    STRING FULL-NAME DELIMITED SPACE
        " " DELIMITED SIZE
        ANS2 DELIMITED SPACE
        " " DELIMITED SIZE
        "PARAMETER-" PARM " " DELIMITED SIZE
        INTO WHERE-POOL
        POINTER WHERE-PTR.
ACCEPT-DATA-NAME.
    ACCEPT TEST-NAME
    IF TEST-NAME = SPACES
        MOVE A-YES TO LAST-RESPONSE
        MOVE A-YES TO GOOD-RESPONSE
    ELSE
        SET LEX-HOLD TO LEX
        PERFORM CHECK-LE-LINE
            VARYING LEX FROM 1 BY 1

```

```

        UNTIL LEX > LE-LAST
        OR   GOOD-RESPONSE = A-YES
    SET LEX TO LEX-HOLD
    IF GOOD-RESPONSE = A-NO
        DISPLAY "NOT IN THE LINE, TRY AGAIN".
CHECK-LE-LINE.
    PERFORM GET-LINE-ELEM-NAME.
    IF TEST-NAME = ELEMENT-NAME
        MOVE A-YES TO GOOD-RESPONSE.
BUILD-SELECT-CLAUSE.
    IF DI-SLOW = A-YES
        DISPLAY "HOORAY!!!!      SELECT CLAUSE TIME".
*   ACCEPT DUMMY.
    MOVE SPACES TO TEMP-STRING.
    MOVE SPACES TO SEQL-LINE.
    IF LNX < LNMAX
        AND   LINE-LEVEL(LNX) < LINE-LEVEL(LNX + 1)
        MOVE "NEXT)" TO TEMP-STRING
    ELSE
        MOVE "ALL)" TO TEMP-STRING.
*   DISPLAY "TEMP-STRING =" TEMP-STRING.
    STRING "SELECT (" DELIMITED SIZE
        TEMP-STRING DELIMITED SPACE
        INTO SEQL-LINE.
    MOVE SEQL-LINE TO PRINT-LINE
    PERFORM PRINT-RTN.
    MOVE SPACES TO PRINT-LINE.
    IF DI-SLOW = A-YES
        DISPLAY " WAHOO " SEQL-LINE.
    PERFORM ZERO-RELATION-TEMPS
        VARYING RELX FROM 1 BY 1
        UNTIL RELX > REL-LAST.
    PERFORM BUILD-SELECT-ENTRIES
        VARYING LEX FROM 1 BY 1
        UNTIL LEX > LE-END-IN-DD.
    SET RELX TO PRIMARY-RELATION.
    MOVE 2 TO REL-USED(RELX).
    MOVE "FROM " TO PRINT-LINE.
    PERFORM PRINT-RTN.
    MOVE SPACES TO FROM-CLAUSE.
    MOVE 5 TO FROM-NEXT.
    PERFORM BUILD-FROM-CLAUSE
        VARYING RELX FROM 1 BY 1
        UNTIL RELX > REL-LAST.
    MOVE FROM-CLAUSE TO PRINT-LINE.
    PERFORM PRINT-RTN.
    SET WSX TO 1.
    SET WSX DOWN BY 1.
    PERFORM INIT-WHERE-STACK
        VARYING RELX FROM 1 BY 1
        UNTIL RELX > REL-LAST.
    SET WS-LAST TO WSX.
    MOVE A-NO TO AND-SW.
    MOVE SPACES TO WHERE-CLAUSE.

```

```

MOVE 5 TO WHERE-POINTER.
PERFORM BUILD-NAVIGATION-WHERE-CLAUSE
    UNTIL WS-LAST = 0.
PERFORM BUILD-LEVEL-NAV-WHERE-SEGMENTS
    VARYING LEX FROM 1 BY 1
    UNTIL LEX > LE-LAST.
PERFORM PRINT-SELECTION-WHERE-SEGMENTS.
IF DI-SLOW = A-YES
    ACCEPT DUMMY.
ZERO-RELATION-TEMPS.
MOVE ZERO TO REL-USED(RELX).
MOVE ZERO TO REL-NAVIGATION-COUNTER(RELX).
BUILD-SELECT-ENTRIES.
IF DI-SLOW = A-YES AND LE-PROCESS(LEX) = A-YES
    DISPLAY "STARTING 'BUILD-SELECT-ENTRIES'"
    PERFORM GET-LINE-ELEM-NAME
    DISPLAY "ELEMENT-NAME ="
        ELEMENT-NAME.
IF LE-PROCESS(LEX) = A-YES
    AND LE-TEMPORARY(LEX) = A-NO
    SET RELX TO LE-RELATION(LEX)
    MOVE 1 TO REL-USED(RELX)
    SET DDX TO LE-ID(LEX)
    MOVE LE-RELATION(LEX) TO REL-NO
    PERFORM NAME-CONCAT
    STRING " " FULL-NAME DELIMITED SIZE
        INTO PRINT-LINE
    PERFORM PRINT-RTN.
SPW-1.
MOVE DD-RELAT-NON(DDX) TO BOOL-CANDIDATES.
PERFORM ESTABLISH-ORIGIN.
SPW-2.
MOVE BOOL-1 TO B-WORK1.
IF DD-RELAT-NON(DDX) NOT = ZERO
    MOVE DD-RELAT-NON(DDX) TO BOOL-CANDIDATES
ELSE
    PERFORM FIND-PRES-SOURCE
        VARYING SS1 FROM 1 BY 1
        UNTIL SS1 = RELX
    COMPUTE BOOL-CANDIDATES = B-WORK1 EXOR
        DD-RELAT-KEY(DDX).
IF DI-SLOW = A-YES
    SET DISPLAY-NUM TO DDX
    DISPLAY "DDX =" DISPLAY-NUM
    DISPLAY "DD-NAME =" DD-NAME(DDX)
    DISPLAY "REL-NO:" REL-NO
    MOVE BOOL-CANDIDATES TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " BOOL-CANDIDATES =" BC-CONVERT-C
    MOVE DD-RELAT-NON(DDX) TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " DD-RELAT-NON(DDX) =" BC-CONVERT-C
    MOVE DD-RELAT-KEY(DDX) TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR

```

```

        DISPLAY " DD-RELAT-KEY(DDX) ="  BC-CONVERT-C
        MOVE B-WORK1 TO BC-CONVERT-B
        PERFORM BOOL-TO-CHAR
        DISPLAY " B-WORK1 ="  BC-CONVERT-C
        MOVE B-WORK2 TO BC-CONVERT-B
        PERFORM BOOL-TO-CHAR
        DISPLAY " B-WORK2 ="  BC-CONVERT-C
        ACCEPT DUMMY.
    PERFORM ESTABLISH-ORIGIN.
FIND-PRES-SOURCE.
    SHIFT B-WORK1 RIGHT 1.
ESTABLISH-ORIGIN.
    MOVE A-NO TO FOUND-SOURCE.
    MOVE BOOL-1 TO B-WORK1.
    PERFORM ELIMINATE-EXTRANEIOUS-SOURCES
        VARYING RELX FROM 1 BY 1
        UNTIL FOUND-SOURCE = A-YES OR RELX > REL-LAST.
    MOVE BOOL-1 TO B-WORK1.
    PERFORM IDENTIFY-ITS-RELATION
        VARYING RELX FROM 1 BY 1
        UNTIL FOUND-SOURCE = A-YES OR RELX > REL-LAST.
    IF FOUND-SOURCE = A-YES
        PERFORM NAME-CONCAT
        SET RELX TO REL-NO
        MOVE REL-NO TO LE-RELATION(LEX)
    ELSE
        PERFORM SOURCE-INDETERMINABLE.
ELIMINATE-EXTRANEIOUS-SOURCES.
    SET REL-NO TO RELX.
    IF DI-SLOW = A-YES
        SET DISPLAY-NUM TO DDX
        DISPLAY "DDX ="  DISPLAY-NUM
        DISPLAY "DD-NAME ="  DD-NAME(DDX).
    COMPUTE B-WORK2 = BOOL-CANDIDATES AND B-WORK1.
    IF B-WORK2 NOT = ZERO
        COMPUTE B-WORK2 =  BOOL-CANDIDATES EXOR  B-WORK1
        IF B-WORK2 = ZERO
            MOVE A-YES TO FOUND-SOURCE
        ELSE
            PERFORM MULTIPLE-SOURCES.
    SHIFT B-WORK1 RIGHT 1.
IDENTIFY-ITS-RELATION.
    SET REL-NO TO RELX.
    IF DI-SLOW = A-YES
        SET DISPLAY-NUM TO DDX
        DISPLAY "DDX ="  DISPLAY-NUM
        DISPLAY "DD-NAME ="  DD-NAME(DDX).
    COMPUTE B-WORK2 = BOOL-CANDIDATES AND B-WORK1.
    IF B-WORK2 NOT = ZERO
        COMPUTE B-WORK2 =  BOOL-CANDIDATES EXOR  B-WORK1
        IF B-WORK2 = ZERO
            MOVE A-YES TO FOUND-SOURCE.
    SHIFT B-WORK1 RIGHT 1.
NAME-CONCAT.

```

```

MOVE SPACES TO FULL-NAME.
STRING "R" REL-NO "." DELIMITED SIZE
      DD-NAME(DDX) DELIMITED SPACE
      INTO FULL-NAME.
IF DI-SLOW = A-YES
  DISPLAY " NAME-CONCAT.....Q-NAME =" FULL-NAME
  ACCEPT DUMMY.
MULTIPLE-SOURCES.
  SET RPX-HOLD TO RPX.
  SET DDX-HOLD TO DDX.
  MOVE A-YES TO ALL-IN-LINE.
  SET RPX TO REL-1ST(RELX).
  PERFORM CHECK-KEYS REL-LHS(RELX) TIMES.
  IF ALL-IN-LINE = A-NO
    COMPUTE BOOL-CANDIDATES =
      BOOL-CANDIDATES EXOR B-WORK1.
  SET DDX TO DDX-HOLD.
  SET RPX TO RPX-HOLD.
CHECK-KEYS.
  SET LEX-HOLD TO LEX.
  MOVE A-YES TO ABSENT.
  PERFORM COMPARE-LINE-TO-REL-POOL
    VARYING LEX FROM 1 BY 1
    UNTIL LEX > LE-END-IN-DD OR ABSENT = A-NO.
  IF ABSENT = A-YES
    MOVE A-NO TO ALL-IN-LINE.
  SET LEX TO LEX-HOLD.
  SET RPX UP BY 1.
  IF DI-SLOW = A-YES
    MOVE BOOL-CANDIDATES TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " BOOL-CANDIDATES =" BC-CONVERT-C
    MOVE B-WORK1 TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " B-WORK1 =" BC-CONVERT-C
    ACCEPT DUMMY.
COMPARE-LINE-TO-REL-POOL.
  IF LE-ID(LEX) = RP-ELEM-ID(RPX)
    MOVE A-NO TO ABSENT.
  IF DI-SLOW = A-YES
    SET DDX TO LE-ID(LEX)
    STRING "CHECKING FOR ELEMENT " DD-NAME(DDX)
      "- IN RELATION " REL-NO " - "
      "ABSENT: " ABSENT
      BLANKS
      DELIMITED SIZE
      INTO MSG
    DISPLAY MSG.
SOURCE-INDETERMINABLE.
  DISPLAY "HELP !!!! WE'RE AT INDETERMINABLE-SOURCE".
  SET DISPLAY-NUM TO DDX
  DISPLAY "DDX =" DISPLAY-NUM
  DISPLAY "DD-NAME =" DD-NAME(DDX)
  MOVE BOOL-CANDIDATES TO BC-CONVERT-B

```



```

    PERFORM BOOL-TO-CHAR
    DISPLAY " BOOL-CANDIDATES =" BC-CONVERT-C
    MOVE DD-RELAT-NON(DDX) TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " DD-RELAT-NON(DDX) =" BC-CONVERT-C
    MOVE B-AND TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " B-AND =" BC-CONVERT-C
    MOVE B-WORK1 TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " B-WORK1 =" BC-CONVERT-C
    MOVE B-WORK2 TO BC-CONVERT-B
    PERFORM BOOL-TO-CHAR
    DISPLAY " B-WORK2 =" BC-CONVERT-C
    ACCEPT DUMMY.
    MOVE A-YES TO FOUND-SOURCE.
BUILD-FROM-CLAUSE.
    SET REL-NO TO RELX.
    IF REL-USED(RELX) > ZERO
        STRING "R" REL-NO " " DELIMITED SIZE
        INTO FROM-CLAUSE
        POINTER FROM-NEXT.
INIT-WHERE-STACK.
    IF REL-USED(RELX) = 1
        SET WSX UP BY 1
        SET WHERE-STACK-REL(WSX) TO RELX.
BUILD-NAVIGATION-WHERE-CLAUSE.
    MOVE A-NO TO PRINT-WHERE.
    IF DI-SLOW = A-YES
        DISPLAY "++++++NAVIGATION++++++".
    SET WSX TO WS-LAST.
    SUBTRACT 1 FROM WS-LAST.
    SET RELX TO WHERE-STACK-REL(WSX).
    IF REL-USED(RELX) = 1
        MOVE 2 TO REL-USED(RELX)
        SET RPX TO REL-1ST(RELX)
        PERFORM NAV-FOR-RELATION
            VARYING REL-SS FROM 1 BY 1
            UNTIL REL-SS > REL-LHS(RELX).
*    PERFORM PRINT-WHERE-RTN.
PRINT-WHERE-RTN.
    IF PRINT-WHERE = A-YES
        MOVE WHERE-CLAUSE TO PRINT-LINE
        PERFORM PRINT-RTN
        MOVE SPACES TO WHERE-CLAUSE
        MOVE 5 TO WHERE-POINTER.
NAV-FOR-RELATION.
    SET DDX TO RP-ELEM-ID(RPX).
    IF DI-SLOW = A-YES
        DISPLAY "NAVIGATION KEY = " DD-NAME(DDX).
    SET REL-NO TO RELX.
    PERFORM NAME-CONCAT.
    MOVE FULL-NAME TO FN1.
    PERFORM BUILD-A-PATH.

```

```

MOVE FULL-NAME TO FN2.
PERFORM BUILD-WHERE-SEGMENT.
PERFORM PRINT-WHERE-RTN.
BUILD-WHERE-SEGMENT.
  IF AND-SW = A-YES
    STRING " AND " DELIMITED SIZE
    INTO WHERE-CLAUSE
    POINTER WHERE-POINTER
  ELSE
    MOVE "WHERE" TO PRINT-LINE
    PERFORM PRINT-RTN
    MOVE A-YES TO AND-SW.
  STRING FN1 DELIMITED SPACE
    " = " DELIMITED SIZE
    FN2 DELIMITED SPACE
    " " DELIMITED SIZE
    INTO WHERE-CLAUSE
    POINTER WHERE-POINTER.
  MOVE A-YES TO PRINT-WHERE.
BUILD-A-PATH.
  SET RELX-HOLD TO RELX.
  PERFORM SFW-2.
  IF FOUND-SOURCE = A-YES
    PERFORM BUILD-PATH-LINK
  ELSE
    MOVE A-YES TO DEAD-END.
BUILD-PATH-LINK.
  SET RELX TO REL-NO.
  IF DI-SLOW = A-YES
    DISPLAY " NEW RELATION =" REL-NO
    ACCEPT DUMMY.
  IF REL-USED(RELX) = ZERO
    ADD 1 TO REL-USED(RELX)
    ADD 1 TO WS-LAST
    SET WHERE-STACK-REL(WS-LAST) TO RELX.
  IF REL-CANDIDATE(RELX) = A-YES
    ADD 1 TO REL-NAVIGATION-COUNTER(RELX)
    IF REL-NAVIGATION-COUNTER(RELX) =
      CANDIDATES-NO-OF-KEYS
      MOVE A-YES TO CANDIDATE-REACHED.
  IF PR-SLOW = A-YES
    STRING "RELATION ‡ = " REL-NO
      " ‡ OF KEYS = " CANDIDATES-NO-OF-KEYS
      " CANDIDATE REACHED =" CANDIDATE-REACHED
      BLANKS
      DELIMITED SIZE
      INTO MSG
    DISPLAY MSG.
  SET RELX TO RELX-HOLD.
  SET RPX UP BY 1.
BUILD-LEVEL-NAV-WHERE-SEGMENTS.
  IF LE-NAV-RELATION(LEX) NOT = ZERO
    SET DDX TO LE-ID(LEX)
    MOVE LE-RELATION(LEX) TO REL-NO

```

```

PERFORM NAME-CONCAT
MOVE FULL-NAME TO FN1
MOVE LE-NAV-RELATION(LEX) TO REL-NO
PERFORM NAME-CONCAT
STRING "CURRENT(" DELIMITED SIZE
                FULL-NAME DELIMITED SPACE
                ") " DELIMITED SIZE
                INTO FN2
PERFORM BUILD-WHERE-SEGMENT
PERFORM PRINT-WHERE-RTN.
IF DI-SLOW = A-YES
    DISPLAY "LEVEL NAVIGATION WHERE CLAUSE : "
    WHERE-CLAUSE.
PRINT-SELECTION-WHERE-SEGMENTS.
IF DI-SLOW = A-YES
    DISPLAY WHERE-POOL.
MOVE WHERE-PTR TO PTR-LAST.
MOVE 1 TO WHERE-PTR.
MOVE 5 TO WHERE-POINTER.
PERFORM PRINT-NEXT-WHERE-LINE
    UNTIL WHERE-PTR NOT < PTR-LAST.
PRINT-NEXT-WHERE-LINE.
MOVE SPACES TO WHERE-CLAUSE.
IF AND-SW = A-YES
    STRING " AND " DELIMITED SIZE
    INTO WHERE-CLAUSE
    POINTER WHERE-POINTER
ELSE
    MOVE "WHERE" TO PRINT-LINE
    PERFORM PRINT-RTN
    MOVE A-YES TO AND-SW.
MOVE SPACES TO TEMP-STRING.
UNSTRING WHERE-POOL
    DELIMITED BY "*"
    INTO TEMP-STRING
    POINTER WHERE-PTR.
STRING TEMP-STRING " "
    DELIMITED SIZE
    INTO WHERE-CLAUSE
    POINTER WHERE-POINTER.
MOVE A-YES TO PRINT-WHERE.
PERFORM PRINT-WHERE-RTN.
SORT-CLAUSE-BUILD.
PERFORM SORT-LINE-DISPLAY.
DISPLAY "DO YOU WANT THIS LINE SORTED? (Y/N)"
ACCEPT ANS1
IF ANS1 = A-YES
    MOVE A-YES TO MORE-SORT-KEYS
    MOVE "ORDERED BY" TO PRINT-LINE
    PERFORM PRINT-RTN
    DISPLAY " ENTER SORT KEYS MAJOR TO MINOR"
    PERFORM GET-SORT-KEY
        VARYING SS1 FROM 1 BY 1
        UNTIL MORE-SORT-KEYS = A-NO.

```

```

SORT-LINE-DISPLAY.
  SET LEX TO 1.
  IF DI-SLOW = A-YES
    MOVE LE-LAST TO DISPLAY-NUM
    DISPLAY "STARTING DISPLAY LIST..... LE-LAST ="
      DISPLAY-NUM
    ACCEPT DUMMY.
  PERFORM ONE-SORT-LINE
    UNTIL LEX > LE-LAST.
ONE-SORT-LINE.
  MOVE SPACES TO DISPLAY-LINE-DATA.
  MOVE 1 TO SS1.
  PERFORM SORT-LINE-SETUP
    UNTIL SS1 > 6 OR LEX > LE-LAST.
  DISPLAY DISPLAY-LINE-DATA.
SORT-LINE-SETUP.
  PERFORM GET-LINE-ELEM-NAME
  IF LE-ORIGINAL(LEX) = A-YES
    AND LE-FUNCTION(LEX) = A-YES
    MOVE ELEMENT-NAME TO DISP-LINE-NAME (SS1)
    ADD 1 TO SS1.
  IF TR-SLOW = A-YES
    SET DISPLAY-NUM TO LEX
    DISPLAY "LEX ="          DISPLAY-NUM
    DISPLAY "ELEMENT-NAME =" ELEMENT-NAME.
  SET LEX UP BY 1.
GET-SORT-KEY.
  MOVE SPACES TO PRINT-LINE.
  MOVE A-NO TO GOT-KEY.
  SET LEX TO 1.
  SET LEX DOWN BY 1.
  PERFORM SORT-DISP
    UNTIL GOT-KEY = A-YES.
  MOVE A-YES TO LE-SORT(LEX).
  DISPLAY "ASCENDING OR DESCENDING? (A/D)"
  ACCEPT ANS1
  IF ANS1 = "A"
    MOVE " ASCENDING" TO A-D
  ELSE
    MOVE " DESCENDING" TO A-D.
  PERFORM GET-LINE-ELEM-NAME
  IF LE-TEMPORARY(LEX) = A-NO
    MOVE LE-RELATION(LEX) TO REL-NO
  STRING "      R" REL-NO      "." DELIMITED SIZE
  ELEMENT-NAME DELIMITED SPACES
  A-D DELIMITED SIZE
  INTO PRINT-LINE
  ELSE
    STRING "      " DELIMITED SIZE
    ELEMENT-NAME DELIMITED SPACES
    A-D DELIMITED SIZE
    INTO PRINT-LINE.
  PERFORM PRINT-RTN.
  DISPLAY "ARE THERE ANY MORE SORT KEYS? (Y/N)"

```

```

ACCEPT MORE-SORT-KEYS.
SORT-DISP.
  SET LEX UP BY 1.
  IF LE-PROCESS(LEX) = A-YES AND LE-SORT(LEX) = A-NO
    PERFORM GET-LINE-ELEM-NAME
    DISPLAY ELEMENT-NAME
    ACCEPT GOT-KEY.
UNIQUE-ALL-CLAUSE-BUILD.
  MOVE A-YES TO DISP-ALL-KEYS.
  PERFORM DISPLAY-KEYS-TEST
    VARYING LEX FROM 1 BY 1
    UNTIL LEX > LE-END-IN-DD
      OR DISP-ALL-KEYS = A-NO.
  IF DISP-ALL-KEYS = A-NO
    DISPLAY "DO YOU WANT TO DISPLAY ALL VALID RECORDS,"
      " OR JUST THOSE THAT ARE UNIQUE? (A/U)"
    ACCEPT ALL-UNIQUE
    IF ALL-UNIQUE = "U"
      MOVE "UNIQUE" TO PRINT-LINE
      PERFORM PRINT-RTN.
DISPLAY-KEYS-TEST.
  IF LE-PROCESS(LEX) NOT = A-YES
    PERFORM GET-LINE-ELEM-NAME
    COMPUTE B-WORK0 = BOOL-PRIMARY
      AND DD-RELAT-KEY(DDX)
    IF B-WORK0 NOT = ZERO
      MOVE A-NO TO DISP-ALL-KEYS.
GET-LINE-ELEM-NAME.
  IF LE-TEMPORARY(LEX) = A-YES
    SET TTX TO LE-ID(LEX)
    MOVE TT-NAME(TTX) TO ELEMENT-NAME
    MOVE TT-NAME(TTX) TO FULL-NAME
  ELSE
    SET DDX TO LE-ID(LEX)
    MOVE DD-NAME(DDX) TO ELEMENT-NAME
    MOVE LE-RELATION(LEX) TO REL-NO
    PERFORM NAME-CONCAT.
PRINT-COMPUTE-LINES.
  IF DI-SLOW = A-YES
    DISPLAY COMPUTE-POOL.
  MOVE COM-PTR TO PTR-LAST.
  MOVE 1 TO COM-PTR.
  PERFORM PRINT-NEXT-COMPUTE-LINE
    UNTIL COM-PTR NOT < PTR-LAST.
PRINT-NEXT-COMPUTE-LINE.
  MOVE SPACES TO PRINT-LINE.
  UNSTRING COMPUTE-POOL
    DELIMITED BY "*"
    INTO PRINT-LINE
    POINTER COM-PTR.
  PERFORM PRINT-RTN.

```

A USER-ORIENTED TRANSACTION DEFINITION FACILITY
FOR A RELATIONAL DATABASE SYSTEM

by

C. STEVEN ROUSH

B. S., Kansas State University, Manhattan, Kansas, 1972

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements of the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1979

To continue the development of a prototype relational database system, a transaction definition subsystem will be designed and implemented. This subsystem will allow users of the database system to define database transactions without knowledge of the schema of the database or the database model used.

The subsystem will not be complete, but will only address the processes necessary for data retrieval and will not pursue the functions needed to perform maintenance of the database. The retrieval function will be interactively defined in terms of five processes:

1. Identification of those data items to be retrieved.
2. Identification of methods needed to derive all data items which are not stored in the database.
3. Identification of data retrieval criteria.
4. Identification of all sorting requirements.
5. Determination as to whether duplicate lines of data (tuples) should be retrieved.

The subsystem will be implemented in COBOL and will generate a version of SEQUEL as its output.