A MICROCOMPUTER IMPLEMENTATION OF QUERY-BY-EXAMPLE

by

Li-Ling Chen

B.S., Feng Chia University, 1980

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

Approved by :

RA Mr Brude
Major Professor

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

As the fields of information system have become increasingly complex, there has arisen a need for improved formal data base models and formal languages to support them. Though originally adequate, 'record-at-a-time' technology is no longer sufficient for all users, and the need to perform complex operations on the data base has become essential. This has become particularly true as computers have gotten cheaper and therefore easier to obtain, resulting in an increasing number of small businesses and institutions automating their data storage and retrieval operations. Thus, the age of the non-programmer professional, i.e., the age of end users (e.g. secretaries, engineers, clerks) is emerging. Many of these users, although professionals in their own fields, have neither the time nor the motivation to learn a conventional programming language. Therefore, in order to address the non-programmer community, non-procedural, flexible, and user-friendly query languages are becoming important.

A query language may be defined as "a language suitable for non-programmers and oriented towards retrieval of data with fast response." [TAG81]. Since 1970 many such languages have emerged; they have ranged from informal natural languages and formally structured English to formal two-dimensional and graphical languages.

One such formal language is Query-By-Example (QBE) [DAT75,ZLO74,ZLO77,ZLO78] which provides the user with the ability to query, update, define, and manage a relational data base. The first implementation of QBE was done by IBM on the VM/370, and has since been used for such applications as the management of library files, correspondence files, expense accounts, and budgeting. The philosophy of QBE is to minimize what the user must know for both getting started and the number of concepts that the user subsequently has to learn. The results of various psychological studies [THO75] of the language show that it requires less than three hours of instruction for non-programmers to acquire skill in making fairly complicated queries.

1.2 Purpose

The purpose of the effort in preparing this report is twofold: 1) to implement QBE on an IBM compatible microcomputer, the Zenith-150, and 2) to lay the groundwork for utilization of QBE in a computer science class as an auxiliary teaching device. Since dBase II is capable of solving almost any data processing problem it was chosen as the programming language tool. Because of time, manpower, and the inherent limitations of dBaseII, only the QBE operation of retrieval is dealt with in this report.

1.3 Organization of The Report

The remainder of this report is organized into three main chapters: a review of related literature, a description of the algorithm, and a conclusion. The literature review introduces

the relational languages and provides an overview of QBE and research related to it; useful terminology is also defined. The chapter dealing with the algorithm briefly introduces the dBase II programming language and gives a detailed description of the algrorithm. Also included are two examples which are used to demonstrate the query process. The final chapter concludes the report and supplies a discussion of the system's limitations and its future possibilities for improvement.

There are, five appendices. Appendix A provides the table skeleton syntax, a description of the general retrieval rules, and the user's manual for operation of the QBE system. Appendix B gives a brief introduction to the dB compiler. The differences between the IBM's QBE and the QBE designed for the microcomputer are summarized in Appendix C. The way in which program modules are connected together is given in Appendix D, and a source listing of the QBE implementation is given in Appendix E.

CHAPTER 2

REVIEW OF LITERATURE

2.1 The Relational Language

The introduction of the relational model [COD70] in the early 1970's triggered the design of powerful yet easy-to-use relational languages. These languages can be divided into two major classes:

1. Relational algebra-based languages, where queries are expressed by applying specialized operators to relations.
2. Relational calculus-based languages, where queries describe the desired result by giving the conditions which the result tuples have to meet.

QBE can be thought of as graphic version of a relational calculus language which provides users with the ability to manipulate data stored in a data base system. The user sees the data presented in tables with rows and columns of information. All operations (queries) performed on the data are defined using tables, rows, and columns. (An alternative way to represent a query in QBE is to use a linear syntax, which is a one-dimensional string representation; this approach, however, is not considered in this report.)

The following sections present an overview of QBE and research related to it. First, relevant terminology will be presented and defined.

2.2 Terminology and Definitions

This section defines the terminology used throughout the remainder of this report.

2.2.1 Tables Primarily, a table consists of rows and columns of information called data elements. A row is a set of logically associated data elements, one element from each table column. For example, data element JONES with associated data element 18456 (employee number), computer (department name), M (sex), and 30 (age), may constitute one row of a table. A given column will have a name, and contain similar data elements. For example, EMPLOYEE NAME could be the name for a column containing data elements JONES, ADAMS, and SMITH.

2.2.2 Table Skeleton A table skeleton is an empty two-dimensional representation of a table. QBE presents a table skeleton to the user so that the desired processing of the data contained in the data base can be specified. The skeleton contains four distinct types of areas, as shown in Figure 2.1:

table name area	column name area	...	....	column name area
row operator area	data entry area	...	...	data entry area

Figure 2.1 A Table Skeleton

The name of the table is given in the upper left box. A column name area contains the name of a column in that table; some or all of the columns in a table may or may not be represented

depending upon the desires of the user. The row operator area specifies an operation that can be used upon a row. Finally, the data entry area contains column operators and/or entries which define and qualify the processing to take place.

2.2.3 Example Element An example element is used to:

1) associate (i.e.,link) data among different tables or different rows in the same table,
2) specify conditions, or
3) map data from one table to another.

No example element exists in the data base -- it is merely an example of a possible answer to a query. An example element always begins with a leading underline, with the remainder being any combination of letters and numbers.

2.2.4 Constant Element In contrast to an example element, a constant element is used to specify selection criteria; that is, it is used to qualify or limit the results to values matching the constant elements. When used, one must specify the constant exactly as it appears in the actual data.

2.2.5 Condition Box A condition box is used to express one or more desired conditions that are difficult to express in the table. Like a table skeleton, it is a two-dimensional table, containing the condition(s) to be applied to the operations appearing in the table skeleton. Example elements are used to link the condition with the corresponding table skeleton column. When the condition box contains more than one condition, all conditions must be met before the corresponding table data is

accepted as part of the solution.

2.2.6 Query A query specifies the processing to be performed on the QBE data. This processing can involve the definition, retrieval, or modification of data. Any query can contain table skeleton and condition box components, and may utilize more than one skeleton. When multiple skeletons are used, example elements are needed to link the tables for processing purposes.

2.3 Overview of IBM's Version of QBE

Query-By-Example is a high level, non-procedural data base language which provides the end user with a simplified and consolidated interface for querying, updating, defining and managing the data base. It is based on two fundamental factors. First, programming is done within two-dimensional skeleton tables. This is accomplished by filling in the appropriate table cells with an example of the solution. Second, a distinction is made between an example element and a constant element; an example element is used to link data between different tables or rows in the same table, to specify conditions, or to map data from one table to another. A constant element, on the other hand, is used to specify the criteria for selecting data that matches the constant element. Given these two basic factors, the user can express a wide variety of queries.

The major features of the QBE language as defined by IBM, will be presented using a number of examples which are built on the following data base:

EMP(NAME, SAL, MGR, DEPT)

- The EMP table specifies the name, salary, manager, and department of each employee.

SALES(DEPT, ITEM)

- The SALES table is a listing of the items sold by departments.

SUPPLY(SUPPLIER,ITEM)

- The SUPPLY table is a listing of the items provided by suppliers.

A sample of the above data base is shown in Figure 2.2

EMP	NAME	SAL	MGR	DEPT
	JOHN	8000	DAVID	HOUSEHOLD
	ANDERSON	6000	MURPHY	TOY
	MORGAN	10000	LEE	COSMETICS
	LEWIS	12000	LOW	STATIONERY
	NELSON	6000	MURPHY	TOY
	HOFFMAN	16000	MORGAN	COSMETICS
	LOW	7000	MORGAN	COSMETICS
	MURPHY	8000	DAVID	HOUSEHOLD
	DAVID	12000	HOFFMAN	STATIONERY
	HENRY	9000	DAVID	TOY

SALES	DEPT	ITEM
	STATIONERY	DISH
	HOUSEHOLD	PEN
	STATIONERY	PENCIL
	COSMETICS	LIPSTICK
	TOY	PEN
	TOY	PENCIL
	TOY	INK
	COSMETICS	PERFUME
	STATIONERY	INK
	HOUSEHOLD	DISH
	STATIONERY	PEN
	HARDWARE	INK

SUPPLY	ITEM	SUPPLIER
	PEN	PARKER
	PENCIL	BIC
	INK	PARKER
	PERFUME	REVLON
	INK	BIC
	DISH	DUPONT
	LIPSTICK	REVLON
	DISH	BIC
	PEN	REVLON
	PENCIL	PARKER

Figure 2.2 A Data Base Sample

2.3.1 Sample Queries

Q1. Print the names of all employees whose manager is David.

The user fills in the EMP table in the following manner.

EMP	NAME	MGR
	P._LEWIS	DAVID

Explanation. The "P." (Print) indicates the target of the query, i.e., the values that are to appear in the result. Since the query is concerned with the manager DAVID, DAVID is a constant element and is therefore not preceded by an underline. On the other hand _LEWIS, which is preceded by an underline, is the example element and is entered as an example of a possible solution. Actually, Lewis may not necessarily be an element of the data base and can be replaced with _SMITH, _MARY, or a variable _N without altering the meaning of the query.

Later on it shall be shown that example elements are used to establish links between two or more rows in the same table or different tables. Where no links are necessary, one can entirely omit the example element; so, for the above query, P. under the NAME column would have been sufficient.

For the sample data base, the answer to Q1 is:

NAME
JOHN
MURPHY
HENRY

Q2. Print the names, salaries, and managers of employees in the Toy department.

EMP	NAME	SAL	MGR	DEPT
	P._LOW	P._10000	P._MORGAN	TOY

ANSWER:

NAME	SAL	MGR
ANDERSON	6000	MURPHY
NELSON	6000	MURPHY
HENRY	9000	DAVID

Explanation. Here the multiple output was achieved by placing P. before the example elements in the NAME, SAL, and MGR columns. The only constant element is TOY. Since the example elements _LOW, _10000, and _DAVID are not used for linkage purposes, one could just write the function P., as before leaving blank spaces in place of the example elements.

Q3. Find the department(s) that sells an item(s) supplied by BIC.

SALES	DEPT	ITEM
	P._TOY	_INK

SUPPLY	ITEM	SUPPLIER
	_INK	BIC

ANSWER:

DEPT
STATIONERY
TOY
HOUSEHOLD
HARDWARE

Explanation. The significance of the example element is illustrated in this query. The same example element _INK must be

used in both tables. This query may be paraphrased as: print out all of the departments that sell _INK (as an example) such that _INK is supplied by BIC.

In Query-By-Example, the AND and OR operations are expressed implicitly. We ANDed conditions together either by writing more than two entries in the same row, or by linking different rows with the same example element. Queries 4 and 5 demonstrate the AND and OR operations as used implicitly.

Q4. Print the names of employees whose salary is between $10000 and $15000, provided it is not $13000.

EMP	NAME	SAL
	P._JOHN	> 10000
	_JOHN	< 15000
	_JOHN	<> 13000

ANSWER:

NAME
LEWIS
DAVID

Explanation. The use of the same example element _JOHN in all three rows implies that these three conditions are ANDed on to the employee _JOHN. The output is the intersection of the three sets of answers. One has the option of using any of the following relational operators: >, >= , <, <=, and <> (not equal). If no operator is used as a prefix, equality is implied.

Q5. Print the names of employees whose salary is either $10000, $13000, or $16000.

EMP	NAME	SAL
	P._JOHN	10000
	P._LEWIS	13000
	P._HENRY	16000

ANSWER:

NAME
MORGAN
HOFFMAN

Explanation. Here different example elements are used in each row, so that the three lines express independent queries. The output is the union of the three sets of answers.

There are seven built-in functions in the Query-By-Example language namely SUM., ALL. (i.e., include duplicates), CNT. (count), UNQ. (unique), AVG. (average), MAX. (maximum), and MIN. (minimum). The function UNQ. can be attached to the function CNT., SUM., or AVG. Thus SUM.UNQ. means sum only the unique values. Queries Q6 and Q7 illustrate the use of the built-in functions.

Q6. Print the total salaries of the employees in the Toy department.

EMP	SAL	DEPT
	P.SUM.ALL._S	TOY

ANSWER:

SAL SUM
21000

Explanation. All._S is defined as the set of all salaries matching the Toy department. The aggregate operator SUM. sums this set and the P. prints the result. ALL. does not automatically eliminate duplicates since it creates a multiset (a set that retains duplicates).

Q7. Count the total number of departments in the SALES table.

SALES	DEPT
	P.CNT.UNQ._J

ANSWER:

DEPT	CNT
5	

Explanation. Since there are duplicate departments in the DEPT column, the function UNQ. is attached to eliminate duplicates. Here again, one can use P.CNT.UNQ., omitting the example element _J.

In Query-By-Example there are two two-dimensional objects. The first is the two-dimensional table skeleton that has been described. The second is the condition box, which is a box with the heading CONDITIONS, and is used when one or more desired conditions are difficult to express in the tables. A blank condition box may be displayed at any time the user desires.

Q8. Print the names of the employees whose salaries are greater than the sum of those of John and Low.

EMP	NAME	SAL
	P.	_S1
	JOHN	_S2
	LOW	_S3

CONDITIONS
_S1 > (_S2 + _S3)

ANSWER:

NAME
HOFFMAN

Explanation. This simple condition could have been expressed by replacing _S1 by > (_S2 + _S3) in the first row of the EMP table. An equality operator in a condition box should not be confused with an assignment statement. An assignment statement implies a procedure, and QBE is a non-procedural language. Thus, assignment statements are not allowed; that is, an expression like _M = _M + 1 is always false. Different expressions in a condition box are entered on separate lines, but all must hold simultaneously, i.e., all conditions in a condition box are ANDed together.

2.3.2 Modification

To modify the data base, only three essential operators are necessary: I., D., and U. (standing for "insert", "delete", and "update" respectively). With the same query syntax and the above three operators the user can perform complicated modifications on the data base. Deleting John's record from a table, for example,

is accomplished by entering D. against that row as follows:

EMP	NAME	SAL	MGR	DEPT
D.	JOHN			

Updating a group of records is achieved by means of a query expression. For example, if we wish to replace manager DAVID by LEWIS in all relevant employee records, we state:

EMP	NAME	MGR
	_N	DAVID
	_N	U. LEWIS

This is termed a 'query dependent update', since the system must first query the data base to find all the employees under the manager named DAVID and then update the manager name to LEWIS.

The point to be made here is that with the introduction of very little syntax, the user has the expressive query power to modify the data base. Furthermore, since the output of a query is itself a relation , the user can directly edit that output by placing D., I., or U. before the attributes to be modified.

2.3.3 Table Creation

To interactively create tables, the user mimics the way he would create tables manually, that is, starting with a blank skeleton of table and then inserting the table name and the column headings. In addition, to complete the definition of a table, the user must use the system's keywords such as DOMAIN, KEY, LENGTH, and TYPE. A domain is a QBE entity which specifies value classes for one or more table columns. These value classes

are then used for describing the characteristics of the data for the table columns. A key is an optional specification. When the values in this table column are to be used to uniquely identify this row or record from any others in the table, the keyword KEY is specified (K means key, NK means non-key).

With these keywords, then, one could define a table called EMP follows:

I. EMP I.	NAME	SAL	MGR	DEPT
TYPE I.	CHAR	FIXED	CHAR	CHAR
LENGTH I.	20	8	20	12
KEY I.	K	NK	NK	NK
DOMAIN I.	NAMES	MONEY	NAMES	DEPTS

(Placing I. in the beginning of the row is a shorthand notation for prefixing every entry of that row with I.)

After a table is created, the user has the ability to update any of the existing definitions in the same manner as a record is updated. In addition to new base tables, the user can create a 'snapshot', a table whose data is mapped from already existing ones.

2.4 Brief Review of Related Research

SBA [DEJ77] (The System for Business Automation) and OBE [ZL082] (Office-By-Example) are two IBM products which are premised on the same data base concepts as QBE. An SBA program is composed of a collection of Query-By-Example transactions over the tables of more complex data objects such as forms, charts, and reports. Further, the program uses examples for specification of program invocation, parameters, triggers, and user interaction. A business system consists of a collection of

programs and additional representations for user roles, organization, business functions, and communications. So, QBE is actually a subset of the SBA programming language.

Office By Example is a two-dimensional language, which attempts to mimic manual procedures of business and office systems. It is a superset and natural extension of the QBE while containing features from SBA.

In closing, then, the language for OBE can be used to combine and integrate aspects of word processing (including editing and formatting), data processing, report writing, graphics, and electronic mail. With such a language, end users are able to specify and store complex OBE programs, thus developing their own applications. OBE has been widely used in such applications as finance, government, manufacturing, and construction, among others.

CHAPTER 3

THE ALGORITHM FOR UTILIZATION OF QBE

3.1 Objective

This chapter describes an implementation algorithm for a subset of Query-By-Example. Query-By-Example was designed to provide a conceptually simple, easy-to-use tool for the retrieval of information from a relational data base. The algorithm will be described with reference to dBase II.

A query composed in QBE is entered in parallel; that is, the order in which the various rows and columns are specified is not necessarily related to the order in which the query is processed. The algorithm uses a recursive serialization process to analyze the query to indentify a processing order. This algorithm is well-suited for implementation on the Zenith-150 microcomputer.

Before the algorithm is presented, the data base programming language dBase II will be introduced in the following section.

3.2 Programming Language - dBase II

dBase II [ASH81], a product of Ashton-Tate, is a relational data base management system that operates in two modes. In the first or immediate mode, the user types a command and presses the enter or carriage return key, and dBase II acts immediately to accomplish the requested function; that is, an interactive manipulation of the data base takes place. The function might involve creating data base structures, updating a file, or displaying data contained in one or more records. This immediate

mode provides the quickest way to do simple, non-repetitive data base operations.

The other mode corresponds to the execute or indirect mode of a programming language, and requires the use of command files. A command file is a sequence of dBase II instructions (including all of those which may be used in the immediate mode) stored as a disk file. The instructions are prepared in advance with an editor provided as part of the dBase II system or with a separate text editor, and stored as a disk file. All programming operations such as decision-making, looping, sorting, searching, selecting, displaying, and data manipulation can be used with dBase II. It also provides "set" commands and macro substitutions. Set commands change the configuration of dBase II, while macro substitution allows an instruction to be replaced by a defined sequence of instructions. The following lists three basic programming structures and how they are implemented in dBase II.

1. Decision Decisions are made in dBase II with the form IF/ELSE/ENDIF. For a simple choice, one may drop the ELSE and use :

```
IF  condition
        S1
        [S2]
         .
         .
         .

ENDIF
```

(where Si is any valid statement)

The "condition" can be a series of expressions (up to a

maximum of 254 characters) that can be logically evaluated as being either true or false. The logical operators are used to tie them together (e.g., *cond1* .AND. cond2 .OR cond3).

In this simple form, if all the conditions are met, the computer will perform the statements given between the IF and the ENDIF sequentially, and then proceed to the next statement following the ENDIF. If the conditions are not met, the computer skips to the first statement following the ENDIF.

If multiple alternatives are needed, one may express it in this manner:

```
IF  condition1
        S1
ELSE
   IF  condition2
         S2
   ELSE
      IF  condition3
            S3
      ELSE
               .
               .
               .
      ENDIF3
   ENDIF2
ENDIF1
```

This nested structuring can be used when a choice among alternatives has to be made. The computer does the first statement (S1) if condition1 is evaluated to be true, otherwise the computer will skip to the statement following the ELSE, and continuously evaluate the other conditions in the same manner until one of the conditions is met or none of them are met. An alternative way to express multiple choice in dBase II is using DO CASE.

```
DO CASE
   CASE expression1
         S1
   CASE expression2
         S2
      .
      .
      .
   [OTHERWISE]
         Sn
ENDCASE
```

Here, an expression may contain anything and the series of CASEs need not have a tight relationship.

DO CASE is a structured procedure; dBase II will examine the expression in the individual CASEs and the first one that is true will have the statement after it executed. When dBase II reaches the next phrase beginning with a "CASE" it will exit to the ENDCASE. This means that if more than one CASE is true, only the first one will be executed.

There is no limit to the number of CASE phrases that a DO CASE may contain. The OTHERWISE phrase is optional. If the otherwise clause is present and none of the CASEs is true, then the Sn in the OTHERWISE clause will be executed. If there is no OTHERWISE clause and none of the CASEs is true, then the DO CASE will be exited with none of the Si statements executed.

2.Repetition Repetition is handled in dBase II with the DO WHILE construction:

```
DO WHILE  conditions
          S1
          [S2]
          .
          .
ENDDO
```

3.Procedure The ability to create standard procedures in a language greatly simplifies programming. In BASIC these procedures are called subroutines, while in PASCAL and PL/1 they are called procedures. In dBase II they are command files that can be called by a program. An example of calling command files might be:

```
IF  command = 'P.'
    DO  print
ELSE
   IF  command = 'D.'
       DO  deletion
   ELSE
      IF command = 'U.'
         DO  update
      ENDIF
   ENDIF
ENDIF
```

Notice that print, deletion, and update are three other command files which are specified elsewhere in the dBase II program.

3.3 Description of algorithm

A query is processed by an algorithm which is divided into four phases (Figure 3.1). In phase one, each query is stored into a file called SFILEi (where i is the number of the table). In phase two, each SFILE is scanned to match up each entry with an appropriate table identification and column heading. Output is produced consisting of an unordered collection of attribute-value pairs. In phase three, the output from phase two is treated as a tree, which is traversed to identify a precise sequence in which the data base should be interrogated. A set of data base instructions are placed on a last-in-first-out stack called the CODE stack. Finally, phase four processes the CODE

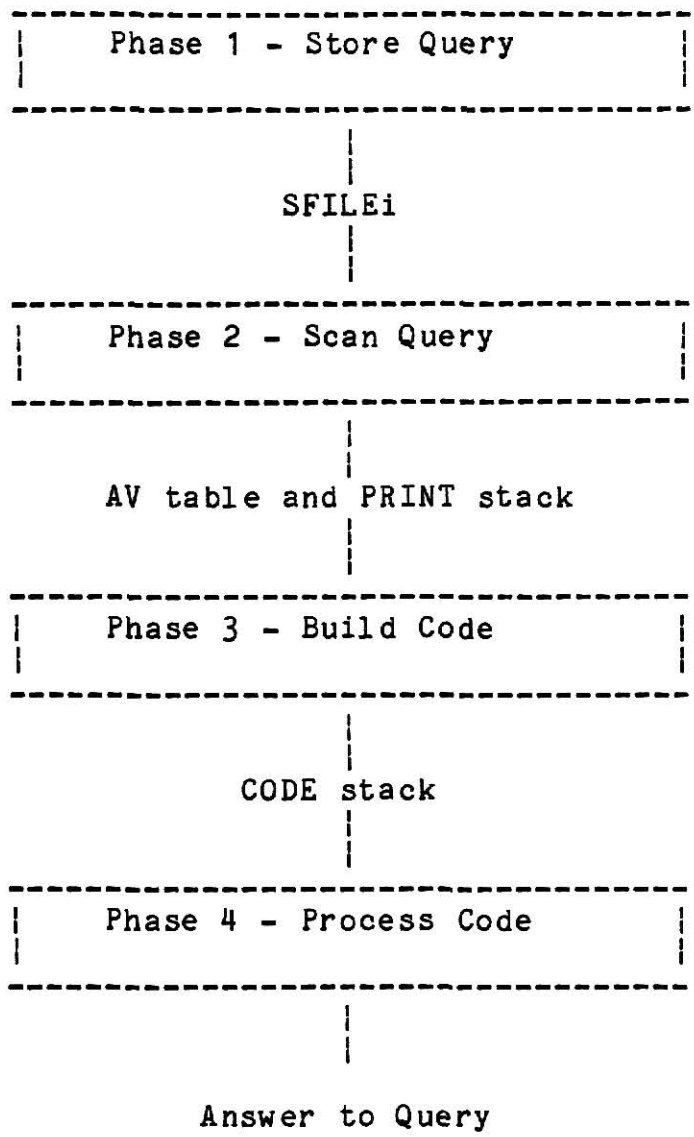

Figure 3.1 - Four Phases of Query Processing

stack by converting each instruction into a dBase II command, and retrieves the information from the data base. All the retrieved information will then be pushed onto another stack called a SAVE stack. When phase four finds the CODE stack empty, the information remaining on the SAVE stack is displayed as the answer to the query.

The following four subsections will discuss each of the four phases in detail; this is followed by some examples of how different queries are processed by the four phases.

3.3.1 Phase One - Store query

Prior to entering a query, the user is shown the necessary table skeletons on the screen of the CRT. The user then types a query into the appropriate columns of one or more tables. When the user finishes entering the query and signals the system by pressing the letter 'R' (run), phase one begins. This phase scans the screen and stores the query into a file called SFILEi(or two files maximum, if the user has indicated such). At the same time, the system will check each table or relation name to see if that particular one exists in the data base. If a table name is not found, the system will show an error message and ask the user to enter another query.

3.3.2 Phase Two - Scan Query

In phase two, the system scans each SFILE from phase one and builds an internal table which contains six columns called, respectively, ATTR (attribute), VALUE, PREFIX, BUILTFUN (built-in function), OPERATOR, and TABLEID. This table is called the

attribute-value or AV table. Each row of the AV table contains information about a single field in the query which the user has specified. The field entry itself appears in the VALUE column; the name of the column in the query is placed in the ATTR column; the PREFIX column holds the command (such as P.); any built-in function (such as SUM., AVG.) goes into the BUILTFUN column; the relational operator (such as >, <, =) appears in the OPERATOR column; and a numeric identifier for the table in the query appears in the TABLEID column. In addition to being placed in the AV table as it is being built, every field with a print statement (a line containing a P.) is placed on a stack called the PRINT stack.

3.3.3 Phase Three - Build Code

The purpose of phase three is to process the information contained in the AV table in order to form the CODE stack. Phase three, given in flow chart form in Figure 3.2, is summarized as follows:

1. Scan the PRINT stack. If an attribute appears more than once in the PRINT stack, then set the union bit of that associated attribute to true.
2. If the user employed a condition box then an attempt is made to match up each condition statement with the appropriate VALUE column in the AV table.
3. If there are arithmetic expressions in the VALUE column of the AV table, then a result is calculated for each arithmetic expression, and the result, together with the corresponding VALUE column of the AV table, is pushed onto the CODE stack.

4. If there are example elements existing in the VALUE column of the AV table, then search the AV table again to match up another VALUE with the same example element. Next,

   a. If one is found, then check the TABLEID. If the former example element's TABLEID is different from the latter, then set variable JOIN to true. If one is not found, then check if the PREFIX has the value P., and if it does, delete the whole row from the AV table and go to step d; otherwise, display an error message to user, and exit.

   b. Move to the row that precedes the current one and check the value in the VALUE column. If the value is a constant push all the information onto the CODE stack and delete the whole row from the AV table; otherwise move to the row that is two lines below the current one and repeat this same procedure of checking for a constant.

   c. Delete the two rows having the same example element.

   d. Repeat step 4 until no example element is found.

5. Scan the AV table again. If i is not empty, push the remaining information onto the CODE stack.

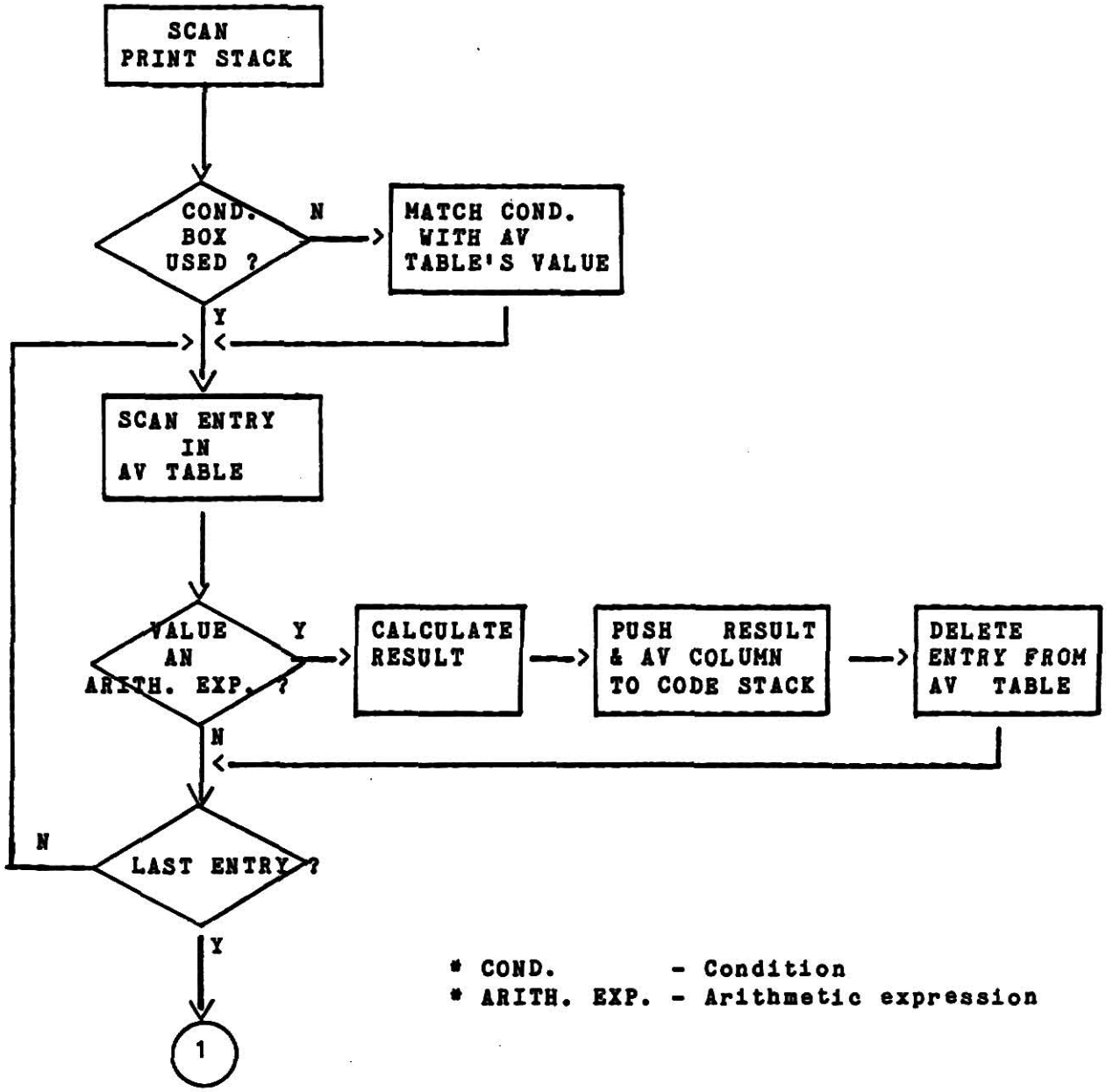

Figure 3.2 Flowchart for Phase Three

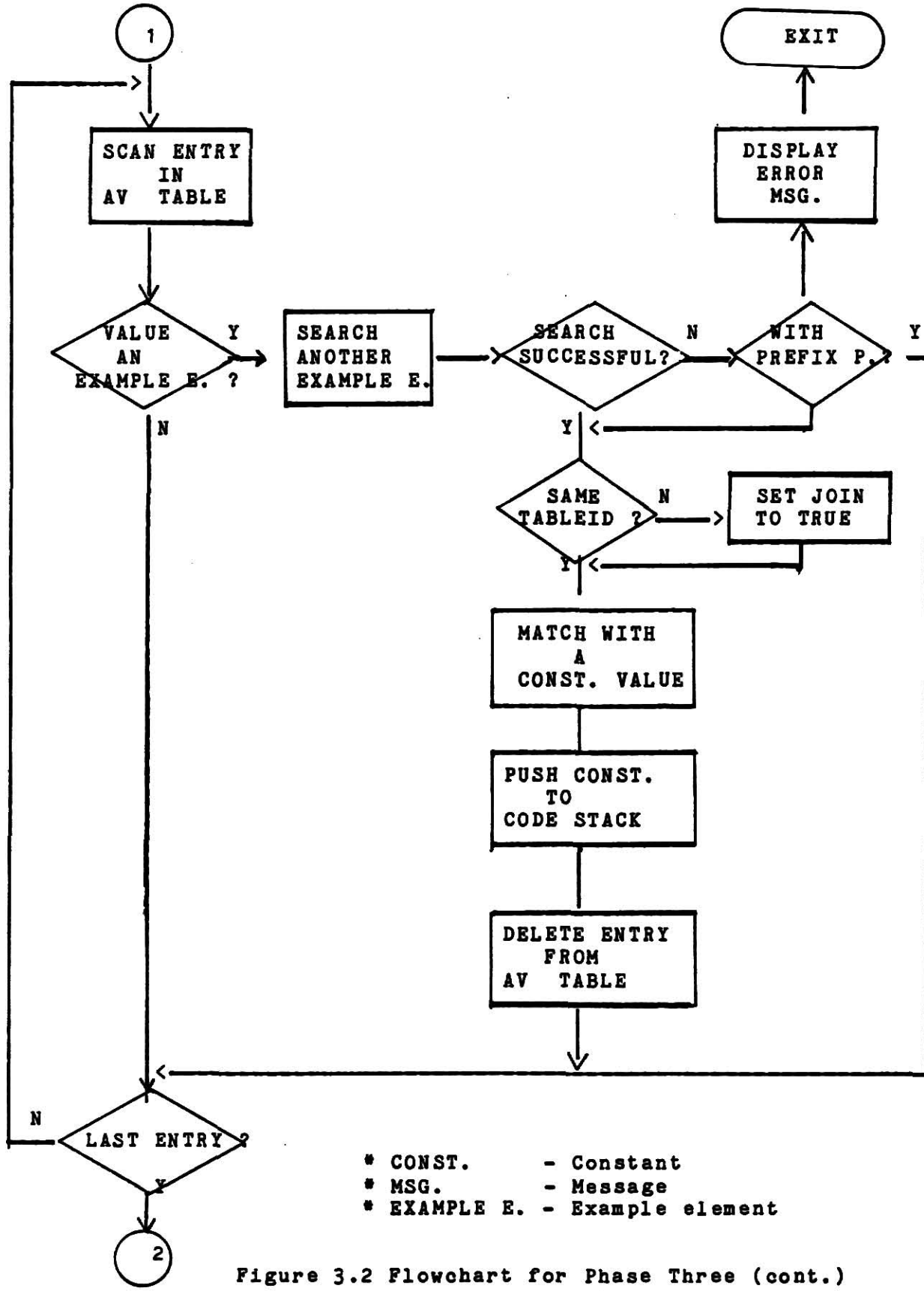

Figure 3.2 Flowchart for Phase Three (cont.)

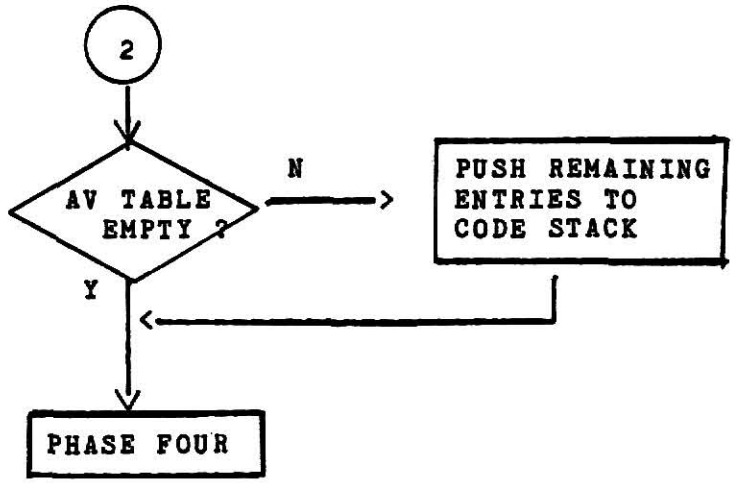

Figure 3.2 Flowchart for Phase Three (cont.)

3.3.4 Phase Four - Process Code

The purpose of phase four is to sequentially process each item on the CODE stack by translating each instruction into a dBase II command and retrieving information from the data base. This phase uses an additional stack called SAVE1 (or two if two tables are used, thus using SAVE1 and SAVE2), and also one temporary storage file called BUFFER1 (or two if two tables are used, thus using BUFFER1 and BUFFER2). The final answer to the query will be pushed onto another stack called SAVE and then printed out.

A flow chart for phase four is shown in Figure 3.3. The summary of this phase is as follows:

1. If the CODE stack is empty, then go to step 6.; otherwise continue.
2. Pop one entry from the CODE stack and translate it (by means of USE and COPY operations) into dBase II commands.
3. Retrieve information from the data base and push it into the temporary storage file BUFFER1.
4. If the SAVE1 stack is empty, then copy all of the information in BUFFER1 to SAVE1. Otherwise, check to see if the union bit is true or not. If it is true, then combine (by means of a JOIN operation) all the information in both BUFFER1 and SAVE1 and push the result into SAVE1; otherwise intersect the information in BUFFER1 and SAVE1, and push the result into SAVE1. (If the user has entered two tables, there would be two SAVE files -- SAVE1 and SAVE2 and two temporary storage files --BUFFER1 and BUFFER2. In such a case, BUFFER2 would become incorporated into

SAVE2.)

5. Go back to step 1.

6. If two tables were used, check if variable JOIN is true. If it is, join the information in SAVE1 and SAVE2 and push the result onto the SAVE stack.

7. If only one table was used, copy SAVE1 into SAVE.

8. Print the result from the SAVE stack and exit.

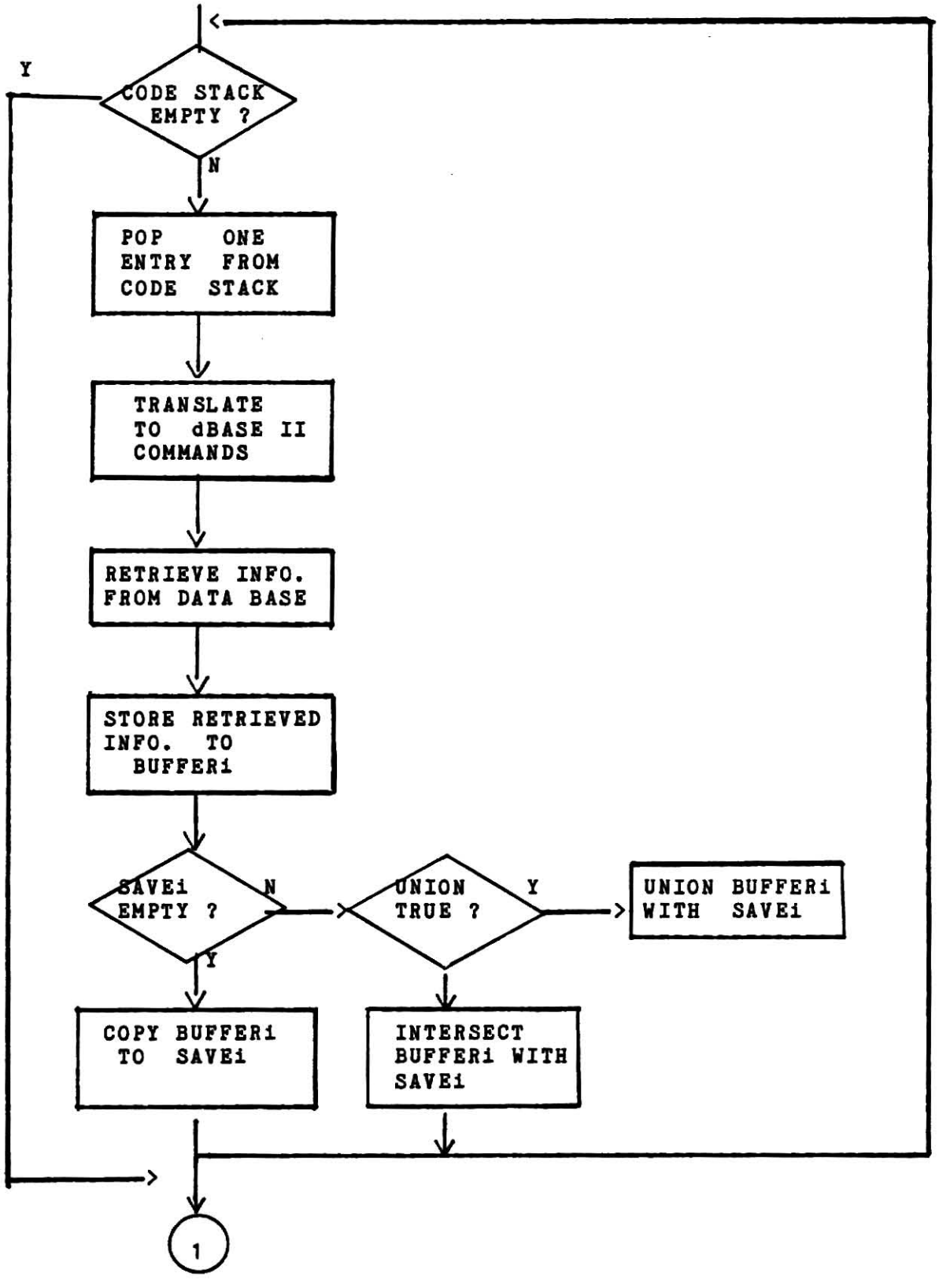

Figure 3.3 Flowchart for Phase Four

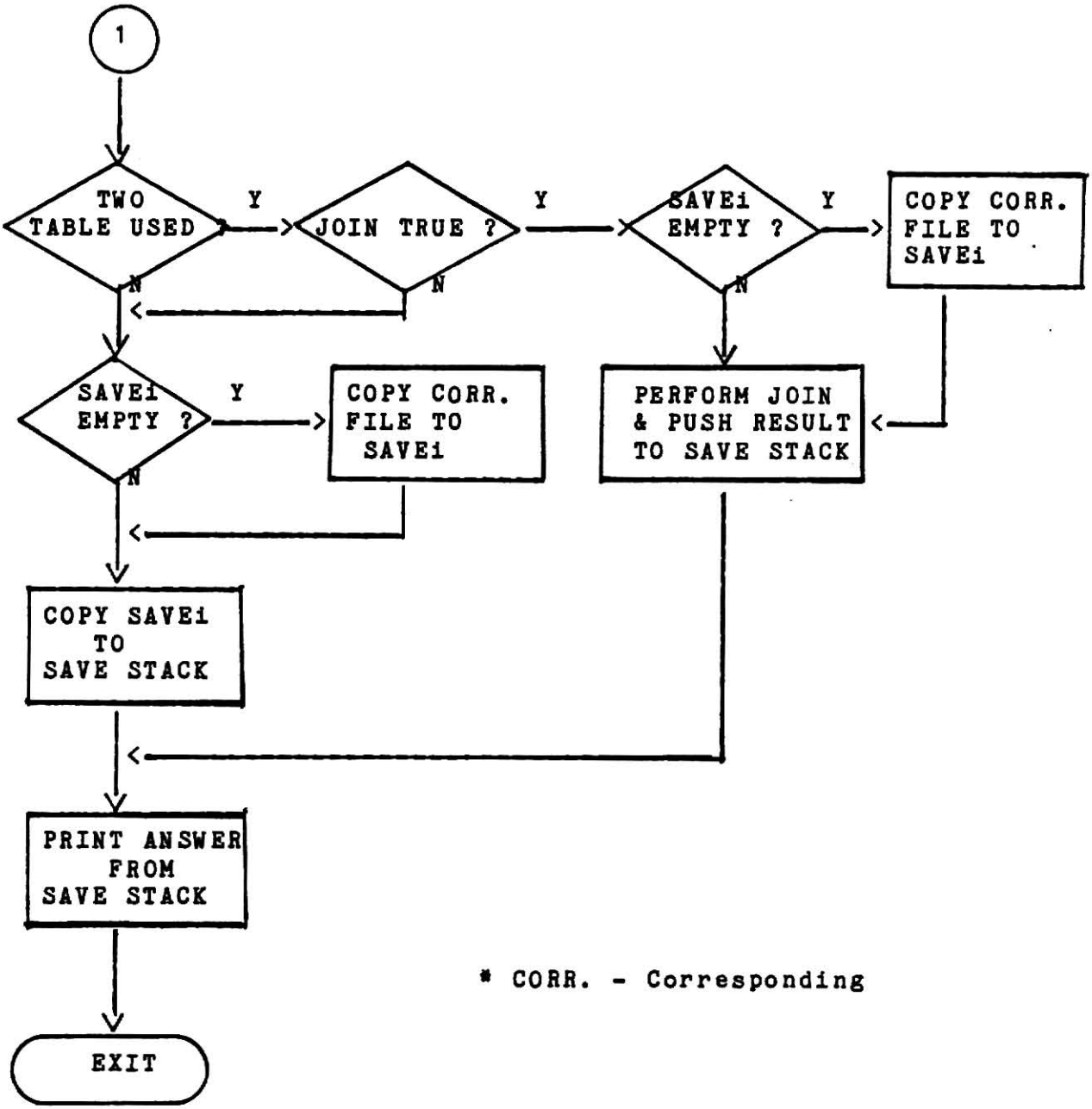

Figure 3.3 Flowchart for Phase Four (Cont.)

3.4 Sample Operation

In order to clarify the operation of the algorithm, two examples will be followed through all four phases. The queries and the explanations are premised on the data base shown in Figure 2.2 (see page 8).

Example 1. Find the department(s) that sells an item(s) supplied by BIC.

In Query-By-Example, this could be entered as follows.

SALES	DEPT	ITEM
	P._TOY	_INK

SUPPLY	ITEM	SUPPLIER
	_INK	BIC

After the query is entered phase one stores it into two sfiles -- SFILE1 and SFILE2. In phase two, the system creates the following AV table by syntactically processing the display, and pushes the attribute DEPT with the print (P.) statement onto the PRINT stack.

ATTR	VALUE	PREFIX	OPERATOR	BUILTFUN	TABLEID
DEPT	_TOY	P.			1
ITEM	_INK				1
ITEM	_INK				2
SUPPLIER	BIC				2

This AV table is then used as input for the next phase, phase three. In the third phase, the PRINT stack is scanned, and since no attribute appears more then once no union bit is set. Next, processing begins by scanning the AV table, which in this case finds the example element _TOY in the first row. _TOY is

scanned for in the remainder of the AV table but it is not found. Thus, the first row is deleted from the AV table (for clarity, the number of the rows in this example remains unchanged). Next, the system finds the example element _INK. _INK is then sought in the VALUE column of the AV table and a match occurs with the third row. The TABLEID in the second and third rows will then be checked. Since they are different, the variable JOIN is set to true. Next, the VALUE column in the second row is checked, and since it is not a constant element, the system then jumps to the fourth row and checks the VALUE column again. This time the value is a constant, so all the information will be pushed onto the CODE stack. Finally, the last three rows in the AV table are deleted and the AV table is now empty.

After phase three is finished, the CODE stack will contain the following information:

ATTR	VALUE	OPERATOR	BUILTFUN	TABLEID	UNION
SUPPLIER	BIC			2	F

Phase four begins processing by using the information in the CODE stack to form dBase II commands, which are USE SUPPLY and COPY TO BUFFER2 FOR SUPPLIER = "BIC". The SUPPLY information is then searched for SUPPLIER = "BIC", and the items PENCIL, INK, and DISH are pushed onto BUFFER2. Since the SAVE2 stack is still empty, no union or intersection processing can be performed, and BUFFER2 is copied onto SAVE2. The CODE stack is now empty, and the JOIN variable is tested and found to be true. Now the system joins SAVE1 and SAVE2 into one stack called SAVE, but since the stack SAVE1 contains no information, all the information in the

SALES table is first copied onto SAVE1, and then the joining of the SAVE1 and SAVE2 files is performed. Finally, the items STATIONERY, STATIONERY, TOY, TOY, STATIONERY, HOUSEHOLD, and HARDWARE on the SAVE stack are displayed as the answer to the query. Notice here that STATIONERY and TOY are duplicate items. This is because the system does not automatically eliminate the duplicates from the answer unless the user indicates that it is to do so. (by placing UNQ. in front of the _TOY under the DEPT column.)

Example 2. Print the names of employees whose salaries are greater than the sum of those of JOHN and LOW.

EMP	NAME	SAL
	P.	_S1
	JOHN	_S2
	LOW	_S3

CONDITIONS
_S1 > (_S2 + _S3)

In phase one the system stores the query into two files called SFILE1 and CONDITION. Phase two creates the AV table containing the following information, and also pushes the attribute NAME onto the PRINT stack.

ATTR	VALUE	PREFIX	OPERATOR	BUILTFUN	TABLEID
NAME		P.			1
SAL	_S1				1
NAME	JOHN				1
SAL	_S2				1
NAME	LOW				1
SAL	_S3				1

In phase three, the PRINT stack is scanned and since no attribute appears more than once no union bit is set to true. Next, the condition statement in the condition box is scanned and used to replace the _S1 in the VALUE column of the AV table. After the replacement, the AV table becomes:

ATTR	VALUE	PREFIX	OPERATOR	BUILTFUN	TABLEID
NAME		P.			1
SAL	(_S2 + _S3)		>		1
NAME	JOHN				1
SAL	_S2				1
NAME	LOW				1
SAL	_S3				1

The system then scans the AV table and finds the arithmetic expression (_S2 + _S3) in the second row. It calculates the result for the arithmetic expression in the following manner. First, the example element _S2 is sought in the AV table and a match is found in the fourth row; the system then moves to the third row, and finds the value JOHN. JOHN's salary of 8000 is then retrieved. Next, the example element _S3 is sought and a match is found in the sixth row; the value LOW in the fifth row is then searched, and LOW's salary of 6000 is retrieved. Finally the values 8000 and 6000 are summed and the result, together with the information in the second row of the AV table, are pushed onto the CODE stack. After calculating the result for this arithmetic expression, the last five rows in the AV table are deleted. Now the first row is the only one left in the AV table. In this row the VALUE column contains nothing. Although the PREFIX column has a P. in it, the system ignores it and deletes it from the AV table. The AV table is now empty.

After the third phase is finished the CODE stack contains the following information:

ATTR	VALUE	OPERATOR	BUILTFUN	TABLEID	UNION
SAL	14000	>		1	F

In phase four, this information is translated into the dBase II commands USE EMP and COPY TO BUFFER1 FOR SAL > 14000. The EMP information is then searched for SAL greater than 14000 and the employee's name HOFFMAN is pushed into the BUFFER1 stack and then copied into SAVE1. Since the code stack is now empty, the variable JOIN is tested and is found to be false. Therefore, the single item HOFFMAN on the SAVE1 is copied into stack SAVE and then displayed as the answer to the query.

3.5 Summary

This chapter has described an algorithm for translating queries in QBE into dBase II commands. Four phases of query processing were described: query storage, query scanning, code generation and execution.

Since the four phases are complex, it may be helpful to briefly restate the major steps in each phase. Phase one stores the query into a file(s) called SFILEi. Each of the table names is also validated during this phase.

In the second phase, the query is processed by scanning the SFILE(s). The AV table is then created and each row of the AV table contains information about a single field which the user has specified in the query.

Phase three generates code. It treats the AV table as a

binary tree and traverses it in order to create an intermediate data structure called the CODE stack.

The last phase analyzes each entry in the CODE stack in order to direct the information retrieval process and format the output for the user.

CHAPTER 4

CONCLUSION

This report has presented an overview of the Query-By-Example language, and the details of how QBE queries could be hosted by a dBase II system. The unique features of this language are summarized as follows:

1. The user is presented with a graphic representation of a table, thereby providing an image to aid in comprehension. Data manipulation is done with this table.
2. The syntax is kept to a minimum and the format is simplified, so that the user can easily understand and learn it.
3. The user can formulate a query in any sequence desired, since the sequence of filling in the table and the rows within the tables is immaterial. Thus, the thought processes of the individual are easily captured.
4. The use of a table skeleton and the example and constant elements allows the user to divide the query into segments, making it highly non-procedural. In contrast, most other languages require the user to first specify the information to be outputed and then structure the query accordingly.

The algorithm described in chapter 3 has been implemented on the Zenith-150 microcomputer, and the system has been tested using different sets of data to confirm that it performs correctly.

4.1 Limitations

At this point, it seems appropriate to mention what appears

to be the main limitations of the implementation.

1. A user cannot formulate a query with a cyclical structure (i.e., multiple link pathways to the same point). To do this would require a different algorithm and a larger amount of storage.

2. The number of columns per table were restricted to six because the system does not supply a scroll function.

3. Column width is restricted to 12 characters. If the user wants to enter a statment containing more than 12 characters, then a condition box must be employed.

4. The number of table skeletons that a query can contain is restricted to two, so that the user cannot link more than two tables at a time.

5. The query cannot be modified if a mistake is found in it, the user must re-type the entire query.

6. The average processing time of each query is about 5 minutes, a long time to wait. This long processing delay is due to the interpreting of large amount of codes and the frequent simultaneous accessing to two files. The processing time might be improved by using the dB Complier which is a product of WordTech Systems, Inc. A brief introduction to the dB complier is given in Appendix B.

4.2 Future Improvements

Corresponding to the above limitations are some possible future improvements.

1. Allow the user to formulate more complicated queries, including queries of a cyclical nature.

2. Expand the table size.

3. Introduce a scroll function.

4. Supply a modification function for query mistakes.

5. Reduce query processing time.

4.3 Project Conclusions

It has been shown that a data base query system utilizing QBE is feasible in the microcomputer environment. Although some limitations are present, a useful and cost-effective implementation has been created. The implementation has been produced in a rather short amount of time and should be used as a prototype and a guide for future implementation of QBE.

BIBLIOGRAPHY

[ASH81] dBase II User Manual. California: Ashto-Tate Inc, 1981.

[DAT75] Date, C. J. An Introduction to Data Base Systems. California: Addison-Wesley, 1975.

[DEJ77] deJong, S. P. and M. M. Zloof. "The System for Business Automation (SBA) : Programming Language," Communications of the ACM, Vol. 20, No. 6, 1977, pp. 385-396.

[LIN86] Jane, Lin. "The Usage of QBE On A Microcomputer," Kansas State University, 1986.

[TAG81] Tagg, M. Roger. "Query Languages for Some Current DBMS," in Databases. Ed. S. M. Deen and P. Hammersley. London: Pentach Press, 1981, pp. 99-118.

[THO75] Thomas, J. C. and J. D. Gould. "A Psychological Study of Query by Example," Proc. National Computer Conference, AFIPS press, Vol. 44, 1975, pp. 439-445.

[ZLO75] Zloof, M. M. "Query By Example," Proc. National Conference, AFIPS press, Vol. 44, 1975, pp. 431-438.

[ZLO77] Zloof, M. M. "Query-By-Example: A Data Base Language." IBM Systems Journal, Vol. 16, No. 4, 1977, pp. 324-342.

[ZL078] Zloof, M. M. "Design Aspects of The Query-By-Example Data Base Management Language." in Databases: Improving Usability and Responsiveness. Ed. Ben Shneiderman. New York: Academic Press, 1978, pp. 29-51.

[ZL082] Zloof, M. M. "Office-by-Example: A Business Language that Unifies Data and Word Processing and Electronic Mail," IBM System Journal, Vol. 21, No. 3, 1982, pp. 272-304.

APPENDIX A

THE ZENITH-150 QBE USER'S MANUAL

Table of Contents

A.1 Introduction

Query-By-Example is a query language that allows for easy manipulation of data. With the QBE system one can create, update, and query a data base. This manual only deals with the query operation.

The language has a minimal syntax and a graphic representation of the user's data processing requests is used. The interactive language facilities are designed for simple formulation of requests.

QBE is a "user-friendly" query system. However, to get the most out of it, please take the time to read the instructions before you begin using it.

A.2 Getting Started

To start the sytem, one must follow three steps:

1.Insert the system disk into the upper (or right) floppy disk drive and turn on the power to boot up the system.

2.Insert the working disk into the lower (or left) floppy disk drive.

3.Type DBASE QBE (upper or lower case letters) and press RETURN. The system will then display the messages, WELCOME TO THE QBE QUERY SYSTEM and PLEASE ENTER TODAY'S DATE (MM/DD/YY): / / . After entering the date, you are ready to formulate the query by using the language components, discussed in the next section.

A.3 Language Components

The Query-By-Example contains three components: elements, operators, and expressions. The following describes each component in detail.

Elements

QBE provides two types of elements: a constant element and an example element. A constant element is used to specify the criteria for selection; that is, to qualify or limit the results to values matching the constant element. An example element is used to link data between different tables or rows in the same table, to specify condition(s), or to map data from one table to another. An example element always begins with an underline, followed by a digit(s) or letter(s).

Operators

In QBE, operators are used to define processing and specify conditions. The following lists the type of operators and their meaning.

TYPE	OPERATOR	MEANING
System	P.	Print, display
	AO.	Sort in ascending order
	DO.	Sort in descending order
Built-in Functions	SUM.	Sum
	CNT.	Count
	AVG.	Average
	MAX.	Find the maximum
	MIN.	Find the minimum
	ALL.	Use the duplicate values
	UNQ.	Use the unique values only
Arithmetic	+	Addition
	-	Subtraction
	*	Multiplication
	/	Division
Comparison	=	Equal to
	<>	Not, or not equal to
	>	Greater than
	<	Less than
	>=	Greater than or equal to
	<=	Less than or equal to

TYPE	OPERATOR	MEANING
Logical	|	Or
	&	And

Expressions

An expression is composed of elements and operators, and is used to qualify or limit the data. The QBE algorithm allows for two types of expressions:

1. Arithmetic An arithmetic expression can contain constant elements and/or example elements with any of the arithmetic operators. Elements in the arithmetic expression can be parenthesized in order to prioritize their evaluation. To determine the order of evaluation of the operators, the following precedence is used: *, /, + , -. Two examples of QBE arithmetic expressions are: (_S1 * _S2) / 2 and _S1 + _S2.

2. Logical A logical expression can only be used in the condition box. A logical expression is evaluated from left to right. Two examples of valid logical expressions are: _S1 = (>1000 | <3000) and _S1 = (1000 & 2000).

A.4 Syntax

When you are ready to enter the query, the system will first show you the message ENTER TABLE NAME (AND COMMAND IF ANY). This asks you to enter the name of the table that you are going to manipulate. To retrieve table names or column headings of a specific table, or both the table names and the corresponding column names, you need to supply the system operator P. with the table name. The syntax for each of the above retrieval cases is

as follows:

1. To retrieve all the table names, enter simply P. or something such as P._TAB, where _TAB is an example element. In this case, names of example elements are arbitrary with respect to their use and meaning.

2. To retrieve all of the column headings for a specific table, enter <table name> P. The table name should be the name that you created on a prior occassion.

3. To retrieve table names and their corresponding column headings, type P._TAB P. or simply P. P. .

If you choose any one of the above cases, the system will not show you a table skelton (a blank table), since the table is not needed in these three cases.

row operator field [P.]	column field name	...	...	...	...	column field name
	data entry field [P.]					

[AO.]
[DO.]
[SUM. {ALL. UNQ.}]
[AVG. {ALL. UNQ.}]
[MAX. {ALL. UNQ.}]
[MIN. {ALL. UNQ.}]
[=]
[<>]
[>]
[<]
[>=]
[<=]

* [ ] Brackets indicate optional information; you can omit any or all of the items inside the brackes.

* { } Braces indicate required information; you must select one of the items inside the braces.

Figure A-1. Retrieval Syntax

To retrieve information from a table or tables, enter the table name (you will then see a blank table skeleton displayed on the screen) and construct the query according to the general syntax as shown in Figure A-1.

If an expression is too long to fit into one data entry field you must use the condition box (Figure A-2). The condition box contains only one type of entry, an arithmetic or a logical expression, as previously discussed.

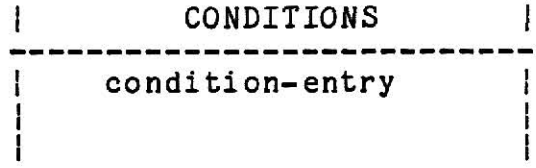

Figure A-2. Condition Box

Each condition-entry is specified on a separate line and all lines are ANDed together.

If you want to print information from different tables then you need to use the result table. This result table can also be used if you want to create a snapshot (i.e., a new table for the output). You will need to give the system a unique table name for creating a snapshot. The syntax for a result table is shown in Figure A-3.

RESULT TABLE ENTER NEW TABLE NAME (IF RESULT NEED TO BE SAVED):

	column field name	...	...	...	...	column field name
row operator field	data entry field					
[P.	<ee>]*					
	[P. <ee>]*					

* <ee> - Example Element

* P. can be placed in the row operator field with the example element appearing in the data entry field, or both P. and the example element can appear together in the data entry field. These two different ways can not be mixed during any given query, but yield the same results.

Figure A-3. Result Table Syntax

A.5 General Retrieval Rules

The following general rules apply to the use of the Query-By-Example language:

1. The print (P.) operator can only appear in the first table skeleton or in the result table.
2. If the value in a column must match a value or range of values known to exist in the data base, type in a constant element (with or without a comparison operator).
3. If the value in a column must be specified but its exact value in the data base is unknown, type in an arbitrary example element.
4. If the value in a column must be linked with a value in another row, column, or table, the same example element must be used in all parts of the link.
5. System operators and built-in functions should end with a period.
6. Any built-in function which is used should be qualified by the system operators ALL. or UNQ.
7. No cyclical structure is allowed. That is, the same example elements cannot be used to link to two different columns, whether in the same or different tables, unless the row in which one of the example elements appears contains a constant element.
8. If the result table is used for output purposes, the column names should match those in the stored tables.
9. If the output information comes from different tables, the result table must be used.
10. The name given to the snapshot (i.e., a saved result table)

must be unique within the data base.

11. Query-By-Example is not case sensitive. That is, it makes no difference whether you use uppercase or lowercase letters. However, if you enter the query using lowercase letters, the system will automatically convert it to uppercase.

A.6 Sample Session

Now, let's find out how you can perform a simple query in the QBE system. Once you understand this basic operating procedure, you should have no problem conducting more complex queries.

Again, this sample is based on the data base shown in Figure 2.2 (see page 8).

Suppose you want to print all the employees' names and salaries in the Toy department. The following will show you how to enter the query step by step.

Step 1. When you boot up the system and type DBASE QBE, the system will then display the messages as shown below.

```
******** WELCOME TO THE QBE QUERY SYSTEM ********

PLEASE ENTER TODAY'S DATE (MM/DD/YY):  /  /
```

Step 2. After you enter today's date and press RETURN, the message 'ENTER TABLE NAME (AND COMMAND IF ANY): ' will be displayed. Type EMP in the blanks which follow the colon, and then press RETURN.

```
ENTER TABLE NAME(AND COMMAND IF ANY):  EMP
```

Step 2. After about thirty seconds, the system completes validation of the table name that you just entered, and a table skeleton is displayed. Enter the column headings NAME,SAL, and DEPT into columns 2,3,and 4 respectively, and then type P._N, P._S, and TOY, respectively, under the column headings. Since the values of NAME and SAL are unknown, example elements _N and _S are used. You can omit the example elements in this case. The main concern in this query is the Toy department, so TOY should be entered as a constant element. Now your screen should look like the sample shown below:

NAME	SAL	DEPT		
P. _N	P. _S	TOY		

Step 3. After you have finished entering the query, and pressing the RETURN key enough times so that the cousor is out of the table, the system asks you to wait for a minute while it stores the query. After that, a message will be displayed on the screen, saying '(R)un (E)rase (Q)uit, PLEASE CHOOSE A LETTER ===>'. If you want the system to process the query, type in R. If you have found some mistakes in your query, enter E to let the system erase the query before you re-type it. If you want to quit, then enter Q for exit.

If R was entered, the system then displays the message PLEASE WAIT FOR A WHILE. This message will stay on the screen until the system finishes processing your query. Then, the message "DO YOU NEED HARD COPY (Y/N)?" will appear. If you need a hard copy, type in Y and turn on the printer, otherwise type in

N. Next, the system will display one employee's record at a time as the answer, as shown below.

```
RECODR #     1

NAME: JOHN
SAL : 8000
DEPT: HOUSEHOLD

Hit <RETURN> to continue or Q to quit
```

If you want to see the next employee's record, simply hit RETURN until the last employee record is reached. Then, the system will show messages YOUR QUERY HAS BEEN DONE, DO YOU WANT TO (C)ONTINUE OR (Q)UIT ====>. You can then type C for continue, or Q for exit.

Now that you have seen how to conduct a simple query, try some of the other examples described in chapter 2 for additional practice.

A.7 Query Maintenance

If the system responds to a query with an error message, but you believe that there is nothing wrong with the query, then the system allows you to save that query for later review. This feature permits potential bugs or limitations of this implementation to be archived.

The file which contains all the error queries is called REVIEW, and is stored in the working disk. To look at the REVIEW file, one must perform the following steps:

1.Insert the system disk into the upper (right) floppy disk drive and insert the working disk containing the REVIEW file into the lower floppy disk drive.

2.Turn on the power to boot up the system.

3.Type in DBASE.

4.Type in USE B:REVIEW when a period shows on the screen.

5.Type in EDIT <n>, where n is the record number.

Each archived query will be stored in the REVIEW file, and possibly may be referred to by multiple record numbers, depending on the amount of storage required by the query. Now let's see how each error query is stored in the REVIEW file.

Suppose the query shown below has been returned with the error message 'Column SALARY does not exist in the file EMP'.

	NAME	SALARY			
P.					

For this example, we will have this query stored in the REVIEW file with three record numbers. If you issue EDIT 1 (supposing the query is stored in first record) and use ^C (control C) to view other records, you would find the query stored in the following manner:

```
RECORD  00001

FLD:NAME   : EMP
ROW1       : P.
ROW2       :
ROW3       :
MESSAGES   : Column SALARY does not exist in the table EMP


RECORD 00002

FLD:NAME   : NAME
ROW1       : _N
ROW2       :
ROW3       :
MESSAGES   :
```

```
RECORD 0003

FLD:NAME   : SALARY
ROW1       : _S
ROW2       :
ROW3       :
MESSAGES   : END OF QUERY
```

FLD:NAME in the first record is the actual table name and P. in ROW1 is the row operator corresponding to the table skeleton. The error message is always put together with the first record of the query.

Each of the remaining records (i.e., 2 and 3) in the REVIEW file corresponds to a column in the table skeleton. For example, record 2 contains the column heading NAME and a data entry _N. To separate queries, the indicator END OF QUERY is placed in the message field of the last record for each query.

If you continuously store error queries into the REVIEW file without erasing some, the REVIEW file may grow so large that it will eventually crash your working disk. Therefore, be sure to erase the error queries that you have reviewed.

A.8 Error Messages

The following are error messages used by QBE:

<built-in function> IS NOT FOLLOWED BY BLANK OR EXAMPLE ELEMENT

EXAMPLE ELEMENT DOES NOT HAVE A CONSTANT VALUE TO MATCH IT

EXAMPLE ELEMENT HAS CONFLICTING COLUMN HEADING
The example element is linked to different column headings.

EXAMPLE ELEMENT IS NOT MATCHED
Cannot find another element to match the example element. (Used for linkage purposes.)

EXAMPLE ELEMENT UNDEFINED
An example element appears in the query, but the system does not know for what purpose because of insufficient information.

COLUMN <column name> IS NOT A NUMERICAL TYPE, CANNOT DO COMPARISON

COLUMN <column name> IS NOT A NUMERICAL TYPE, CANNOT DO CALCULATION

COLUMN <column name> IS NOT EXISTING IN TABLE <table name>
The column name is not part of the table.

COLUMN <column name> OF THE RESULT TABLE IS NOT FOUND IN OTHER TABLES
The column heading that is placed in the result table does not match the column heading having a print statement in other table skeletons.

INCOMPLETE EXPRESSION

INCOMPLETE QUERY
The table is empty.

INSUFFICIENT DATA ENTRY IN RESULT TABLE

INVALID ARITHMETIC EXPRESSION

INVALID COMPARISON OPERATOR IN LOGICAL EXPRESSION

INVALID CONDITION EXPRESSION

INVALID LOGICAL OPERATOR IN <expression>

INVALID QUERY

INVALID SYMBOL
The data entry contains some notations which are unknown to the system.

INVALID SYSTEM OPERATOR

MISSING ALL. OR UNQ. IN BUILT-IN FUNCTION

MISSING COLUMN NAME
A data entry does not have a corresponding column heading.

MISSING COLUMN NAME IN THE RESULT TABLE
The data entry in the result table does not have a corresponding column.

MISSING LOGICAL OPERATOR OR USING INVALID LOGICAL OPERATOR IN <expression>

MISSING P. OPERATOR IN RESULT TABLE

NO BLANK CAN BE EMBEDDED INTO EXAMPLE ELEMENT

NO RECORD AVAILABLE, CANNOT PROCEED <built-in function>
No record exists relevant to your query.

NUMBER CAN NOT BE ENDED WITH A DECIMAL POINT

P. SHOULD BE FOLLOWED BY AN EXAMPLE ELEMENT
In retrieval of table names and/or their corresponding column heading, the P. operator can only be followed either by an example element or blanks.

SORRY, THE SYSTEM CAN ONLY SORT ONE COLUMN
You have entered the system's operator AO. or DO. more than once.

SYNTAX ERROR
Check each data entry, you may not have ended the system's operator (such as P.) or built-in function with a period.

SYNTAX ERROR IN RESULT TABLE
The data entries in the result table should be all example elements.

SYSTEM OPERATOR NOT SPECIFIED

TABLE <table name> DOES NOT EXIST

THE RESULT OF THE ARITHMETIC EXPRESSION IS OVERFLOW
The result of the arithmetic expression is too large to fit into the column specified.

TOO MANY DATA ENTRIES WITH BUILT-IN FUNCTION

TOO MANY P. OPERATORS IN RESULT TABLE

TOO MANY SYSTEM OPERATORS
You have entered too many system's operators in the row operator area.

TOO MANY VALUES OR NO VALUE FOR EXAMPLE ELEMENT
Too many values match the example element, or no values match it.

APPENDIX B

A BRIEF INTRODUCTION TO THE dB COMPILER

The dB complier is the first compiler for dBase II allowing for the legal translation of dBase II programs into code that will execute successfully without the dBase II system.

The dB complier compiles the dBaseII command files and produces an independently executable program that occupies 20 - 30 percent less storage space than an uncompiled application and dBase II combined. It also speeds up the execution time in calculations, logical operations, and file access. However, some of the dBase II commands are not supported by the dB Compiler. These are listed as follows: APPEND*, BROWSE, CHANGE, CREATE*, DISPLAY/LIST STATUS, EDIT, HELP, INSERT, MODIFY COMMAND, MODIFY STRUCTURE, SET CARRY ON/OFF, SET DEBUG ON/OFF, SET ECHO ON/OFF, SET SCREEN* ON/OFF, SET STEP ON/OFF, SET TALK ON* and some macro-substitutes*.

* asterisk indicates the command is used in the implementation program.

APPENDIX C

A COMPARISON OF TWO QBES

This Appendix summarizes the differences between the IBM's QBE and the implemented Zenith-150 microcomputer version of QBE as follows:

FEATURE	IBM QBE	MICROCOMPUTER QBE
Table name	placed in the upper left corner of the table skeleton	placed above and to the right of the table skeleton
Column names	entered either by user or system	entered by user
Number of columns per table	99	6
Column width	32 characters	12 characters
Number of tables available per query	no limit	2 general tables, 1 result table, any number of condition boxes
Horizontal and vertical screen scrolling function	supplied	not available
Grouping function	supplied	not available
Cyclical structure	allowed	not allowed
Output	displayed in table form	displayed in linear form

APPENDIX D

PROGRAM ORGANIZATION

The implemented QBE program contains a main routine which serves as a driver, along with several subroutines. The following diagram shows the hierarchical structure of the major subroutines in the program.

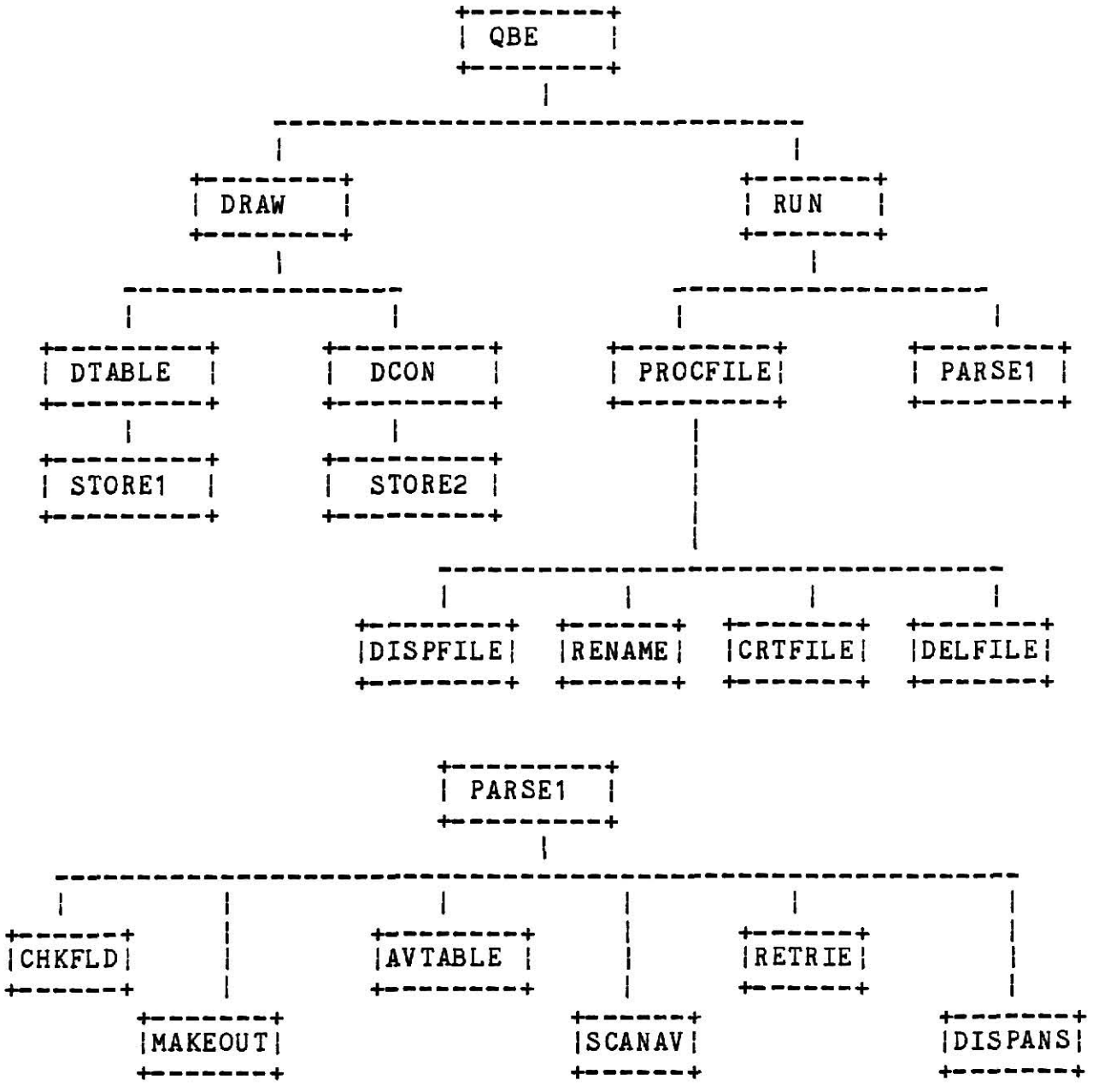

Figure C.1 Program's Hierarchical Diagram

This diagram reflects the nature of calls which take place between the program's modules. If there is a direct path from one module to another at a lower level, then the module at the lower level is directly invoked by the other to perform a function. Since dBase II does not supply parameter passing mechanisms when a subroutine is invoked, all the variables are used as global variables, so each subroutine is constructed without a parameter list and invoked without an argument list.

A brief description of each major subroutine shown in the diagram is given as follows. Program modules which are not included can be found in another report [LIN86].

QBE

The QBE routine displays greeting messages and controls the initial use of the QBE query system.

DRAW

The DRAW routine interacts with the user to determine what kind of table (general, result, or condtion table) the user needs.

RUN

The RUN routine processes the user's query, and, after finishing, continues the interaction with the user who decides at this point whether to continue with another query or not.

DTABLE

The DTABLE routine displays either a general table or a result table on the screen to allow the user to enter the query.

DCON

The DCON routine displays a condition box on the screen to let the user enter the condition statement(s).

PROCFILE

The PROCFILE routine processes queries which are concerned with file management. Examples of the query functions that are supported include: displaying all the file names existing in the user's disk, displaying the structure of a specific file, deleting a file, and creating a new file.

PARSE1

The PARSE1 routine parses the user's query so that information from the data base can be retrieved.

STORE1

The STORE1 routine stores the user's query (entered in the general or result table) into SFILEs.

STORE2

The STORE2 procedure stores the condition statements (entered in the condition box) into a file.

DISPFILE

The DISPFILE routine displays answers for the query which concern either displaying file names existing in the user's disk, or the structure of a specific file.

RENAME

The RENAME routine (which is not included in the program listing) renames an existing file.

CRTFILE

The CRTFILE routine (which is not included in the program listing) creates a new file and inserts it into the data base.

DELFILE

The DELFILE routine (which is not included in the program listing) deletes an existing file from the data base.

CHKFLD

The CHKFLD routine checks all the field names of a specified file to find if they are all existing in the file.

MAKEOUT

The MAKEOUT routine (which is not existing in the program listing) is a general name for two subroutines. One is MAKEOUT1, which scans the general table and stores all field names having a print statement onto the PRINT stack. The other subroutine is MAKEOUT2, which scans the result table and stores all field names having a print statment onto the PRINT stack.

AVTABLE

The AVTABLE routine scans the user's query and separates each data entry into six categories: attributes, value, prefix, operator, built-in function and tableid. They are then stored into an avtable (attribute and variable table).

SCANAV

The SCANAV routine (which is not existing in the program listing) is a general name for three subroutine -- SCANAV1, SCANAV2, and SCANAV3. SCANAV1 scans the avtable and checks if there are any

values with arithmetic operators. SCANAV2 scans the avtable and checks if any value contains the logical operators 'AND' or 'OR'. SCANAV3 scans the avtable and stores all the remaining informatioon onto the CODE stack.

RETRIE

The RETRIE routine (which is not existing in the program listing) is a general name for two subroutines -- RETRIE1 and RETRIE2. Both subroutines retrieve the information from the data base and push the information onto the SAVE stack.

DISPANS

The DISPANS routine displays the result of a query either on the screen or on a hard copy.

APPENDIX E

PROGRAM LISTING

```
*************************************************************************
*                              AVTABLE.PRG                              *
* This procedure, which is called by procedure PARSE1.prg, scans*
* the user's query and separates each data entry into six       *
* categories:  attribute, value,prefix,operator, built-in       *
* function and tableid.  They are then stored into an avtable   *
* (attribute and variable table).                                *
*************************************************************************

IF .NOT. OUTPUT
   USE PRNSTACK
   COPY TO B:PRNSTACK
ENDIF
USE AVTABLE
COPY TO B:AVTABLE
STORE 0 TO I
DO WHILE I < FILE:NUM
   STORE I + 1 TO I
   STORE 'SFILE' + STR(I,1) TO SFILE
   SELECT PRIMARY
   USE B:&SFILE
   STORE 'FILE' + STR(I,1) TO FILE
   STORE TRIM(FLD:NAME) TO &FILE
   GOTO BOTTOM
   STORE # TO MAX:REC
   STORE 0 TO J
   DO WHILE J < 3
      STORE J + 1 TO J
      STORE 'ROW' + STR(J,1) TO ROW:NO
      STORE 1 TO K
      DO WHILE K < MAX:REC
         STORE K + 1 TO K
         GOTO K
         STORE F TO BLANK
         STORE ' ' TO MPREFIX,MVALUE,MOPERATOR,MBUILTFUN
         STORE TYPE TO MTYPE
         STORE LEN TO MLEN
         STORE DEC TO MDEC
         STORE TRIM(FLD:NAME) TO FIELD:NAME
         STORE &ROW:NO TO PARSER
         IF PARSER = ' '
               LOOP
         ELSE
           IF $(PARSER,1,2) = 'P.'
              DO PARSE2
           ELSE
             IF $(PARSER,1,2) = 'U.'
                DO PROCUPDATE
```

```
      ELSE
        IF $(PARSER,1,1) = '_'
           IF $(PARSER,1,2) = ' '
              STORE '*** ERROR - NO BLANK CAN BE'+;
              'EMBEDDED INTO EXAMPLE ELEMENT';
              TO ERROR:MSG
              STORE T TO ERROR
           ELSE
              STORE TRIM(PARSER) TO MVALUE
           ENDIF
        ELSE
          IF $(PARSER,1,1) = '('
             STORE TRIM(PARSER) TO MVALUE
          ELSE
           IF $(PARSER,1,1) = '>' .OR. $(PARSER,1,1) = '<'
               DO  PARSE3
           ELSE
              IF $(PARSER,1,4) = 'ALL.' .OR. ;
                 $(PARSER,1,4) = 'UNQ.'
                 DO PARSE4
              ELSE
                IF $(PARSER,1,4) = 'SUM.' .OR.;
                $(PARSER,1,4) = 'CNT.'.OR.;
                $(PARSER,1,4) = 'AVG.' .OR.;
                $(PARSER,1,4) = 'MAX.'.OR. ;
                $(PARSER,1,4) = 'MIN.'
                   DO  PARSE5
                ELSE
                  IF $(PARSER,1,3) = 'AO.' .OR. ;
                     $(PARSER,1,3) = 'DO.'
                       DO  PARSE6
                  ELSE
                    IF $(PARSER,1,1)$'0123456789'
                       STORE TRIM(PARSER) TO MVALUE
                    ELSE
                      IF $(PARSER,1,1)$'ABCDEFGHIJKLMNOP';
                         .OR. $(PARSER,1,1)$'QRSTUVWXYZ'
                         STORE TRIM(PARSER) TO MVALUE
                      ELSE
                        STORE '***SYNTAX ERROR - INVALID';
                              ' SYMBOL' TO ERROR:MSG
                        STORE T TO ERROR
                      ENDIF
                    ENDIF
                  ENDIF
                ENDIF
              ENDIF
           ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
ENDIF
IF ERROR
   RETURN
```

```
         ELSE
           IF .NOT. BLANK
              SELECT SECONDARY
              USE B:AVTABLE
              APPEND BLANK
              REPLACE ATTR WITH FIELD:NAME, VALUE WITH MVALUE;
              TYPE WITH MTYPE,LEN WITH MLEN, DEC WITH MDEC;
              PREFIX WITH MPREFIX,OPERATOR WITH MOPERATOR;
              REPLACE TABLEID WITH STR(I,1);
                      BUILTFUN WITH MBUILTFUN
           ENDIF
         ENDIF
         SELECT PRIMARY
      ENDDO
   ENDDO
ENDDO
RELEASE MTYPE,FIELD:NAME,MVALUE,MPREFIX,MBUILTFUN,MOPERATOR,I,J;
RELEASE K, MAX:REC,PARSER,FILE,SFILE,ROW:NO,BLANK
SELECT PRIMARY
USE
SELECT SECONDARY
USE
RETURN


*************************************************************************
*                              BUILTANS.PRG                             *
*  This  procedure, which is called by procedure DISPANS.prg,           *
*  displays  the result for a query which  is related to the            *
*  built-in function.                                                   *
*************************************************************************

ERASE
IF PRINTING = 'Y'
   SET PRINT ON
   SET FORMAT TO PRINT
ENDIF
@ 2,3 SAY 'The answer to your query is shown as follows:'
USE B:BUILTFUN
COPY STRUCTURE EXTENDED TO B:STRU
USE B:STRU
STORE FIELD:NAME TO FLD:NAME
STORE '  ' + '&FLD:NAME' + '  ' TO FLD:NAME
STORE FIELD:LEN + 4 TO FLD:LEN
@ 5,20 SAY '|'
@ 5,21 SAY FLD:NAME
STORE FLD:LEN + 21 TO COL
@ 5,COL SAY '|'
STORE FLD:LEN + 2 TO FLD:LEN
STORE 0 TO I
STORE 19 TO COL
DO WHILE I < FLD:LEN
   STORE I + 1 TO I
   STORE COL + 1 TO COL
   IF I = 1 .OR. I = FLD:LEN
```

```
      @ 6,COL SAY '+'
   ELSE
      @ 6,COL SAY '-'
   ENDIF
ENDDO
STORE FLD:LEN - 2 TO FLD:LEN
STORE 6 TO ROW
USE B:BUILTFUN
DO WHILE .NOT. EOF
   STORE ROW + 1 TO ROW
   @ ROW,20 SAY '|'
   @ ROW,23 SAY &FLD:NAME
   STORE FLD:LEN + 20 TO COL
   @ ROW,COL + 1 SAY '|'
   @ ROW,COL + 3 SAY FUNCTION
   SKIP
ENDDO
IF PRINTING = 'Y'
   SET FORMAT TO SCREEN
   SET PRINT OFF
ENDIF
@ 22,1 SAY 'YOUR QUERY HAS BEEN DONE'
RETURN


*************************************************************************
*                              BUILTFUN.PRG                             *
* This procedure, which is called by procedure RETRIE1.prg,             *
* processes all the system supported built-in functions: AO.            *
* (sort field into ascending order), DO. (sort file into                *
* descending order),MIN. (find the minimum value) MAX. (find            *
* the maximum value) SUM. (summation), AVG. (average)  and CNT.*
* (count the record numbers of a file).                                 *
*************************************************************************

USE B:CODESTACK
STORE TRIM(ATTR) TO MATTR,OATTR
STORE TRIM(BUILTFUN) TO MBUILTFUN
IF $(MBUILTFUN,1,3) = 'AO.'
   IF MAX:CODE > 1
      STORE '***ERROR - SORRY,THE SYSTEM CAN ONLY SORT ONE'+;
            ' COLUMN' TO ERROR:MSG
      STORE T TO ERROR
   ELSE
      USE B:SAVE
      SORT ON &MATTR TO B:BUFFER1 ASCENDING
      USE B:BUFFER1
      COPY TO B:SAVE
   ENDIF
   RETURN
ELSE
   IF $(MBUILTFUN,1,3) = 'DO.'
      IF MAX:CODE > 1
        STORE '*** ERROR - SORRY, THE SYSTEM CAN ONLY SORT ONE'+;
              ' COLUMN' TO ERROR:MSG
```

```
      STORE T TO ERROR
   ELSE
      USE B:SAVE
      SORT ON &MATTR TO B:BUFFER1 DESCENDING
      USE B:BUFFER1
      COPY TO B:SAVE
   ENDIF
   RETURN
ELSE
   IF LEN(MBUILTFUN) = 4
      IF $(MBUILTFUN,1,4) = 'UNQ.'
         DO UNIQUE
         USE B:SAVE1
         COPY TO B:SAVE
      ENDIF
      RETURN
   ELSE
      STORE T TO BUILDFUN
      USE B:SAVE
      COPY STRUCTURE EXTENDED TO B:STRU
      USE B:STRU
      LOCATE FOR FIELD:NAME = '&MATTR'
      STORE FIELD:LEN + 3 TO FLD:LEN
      STORE FIELD:DEC TO FLD:DEC
      USE OUTSTRU
      COPY TO B:STRU
      USE B:STRU
      APPEND BLANK
      REPLACE FIELD:NAME WITH MATTR, FIELD:TYPE WITH 'N';
              FIELD:LEN WITH FLD:LEN, FIELD:DEC WITH FLD:DEC
      APPEND BLANK
      REPLACE FIELD:NAME WITH 'FUNCTION', FIELD:TYPE WITH 'C';
              FIELD:LEN WITH 8
      CREATE B:BUILTFUN FROM B:STRU
      DO WHILE MAX:CODE > 0
         USE B:CODESTACK
         GOTO  MAX:CODE
         STORE TRIM(ATTR) TO MATTR
         STORE TRIM(BUILTFUN) TO MBUILTFUN
         IF MATTR = OATTR
            STORE 'SAVE' TO FILE
            STORE @('UNQ.',MBUILTFUN) TO FOUND
            IF FOUND > 0
               USE B:SAVE
               GOTO BOTTOM
               IF # > 1
                  DO UNIQUE
                  STORE 'SAVE1' TO FILE
               ENDIF
            ENDIF
            IF $(MBUILTFUN,1,4) = 'SUM.' .OR.;
               $(MBUILTFUN,1,4) = 'AVG.'
               DO SUMORAVG
            ELSE
               IF $(MBUILTFUN,1,4) = 'CNT.'
```

```
                  DO COUNT
               ELSE
                  IF $(MBUILTFUN,1,4) = 'MAX.' .OR.;
                     $(MBUILTFUN,1,4) = 'MIN.'
                      DO MAXORMIN
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
      ELSE
         STORE '*** ERROR - TOO MANY DATA ENTRIES WITH' +;
               ' BUILT-IN FUNCTION' TO ERROR:MSG
         STORE T TO ERROR
         RETURN
      ENDIF
      IF ERROR
         RETURN
      ENDIF
      USE B:BUILTFUN
      APPEND BLANK
      REPLACE &MATTR WITH RESULT, FUNCTION WITH MBUILTFUN
      STORE MAX:CODE - 1 TO MAX:CODE
   ENDDO
   RELEASE FLD:LEN, FLD:DEC
ENDIF
ENDIF
RELEASE MATTR, MBUILTFUN, OATTR
RETURN


*************************************************************************
*                          CALCULATE.PRG                                *
* This procedure, which is called by procedure SCANAV1.prg,             *
* parses an arithmetic expression to see if it is valid or not.         *
* If the expression is valid, the procedure will then calculate         *
* the result.                                                           *
*************************************************************************

USE PSTACK
COPY TO B:PSTACK
USE VSTACK
COPY TO B:VSTACK
DO WHILE $(EXPRESSION,1,1) <> ' '
   IF $(EXPRESSION,1,1) = '('
      STORE '(' TO MVAL
      DO PUSHP
      DO PUSHV
      STORE $(EXPRESSION,2) TO EXPRESSION
   ELSE
      IF $(EXPRESSION,1,1) = '+' .OR. $(EXPRESSION,1,1) = '-' ;
        .OR. $(EXPRESSION,1,1) = '/' .OR. $(EXPRESSION,1,1) = '*'
            STORE $(EXPRESSION,1,1) TO MVAL
            DO PUSHV
            STORE $(EXPRESSION,2) TO EXPRESSION
      ELSE
```

```
            IF $(EXPRESSION,1,1)$'0123456789'
                  DO CONSTANT
                  IF ERROR
                     RETURN
                  ENDIF
                  DO PUSHV
            ELSE
               IF $(EXPRESSION,1,1) = '_'
                  IF NO:VALUE
                     STORE '*** ERROR - INVALID ARITHMETIC'+;
                           ' EXPRESSION' TO ERROR:MSG
                     STORE T TO ERROR
                     RETURN
                  ELSE
                     DO VARIABLE
                     IF ERROR
                        RETURN
                     ENDIF
                     DO PUSHV
                  ENDIF
               ELSE
                  IF $(EXPRESSION,1,1) = ')'
                     STORE ')' TO MVAL
                     DO POPP
                     IF ERROR
                        RETURN
                     ENDIF
                     DO PUSHV
                     STORE $(EXPRESSION,2) TO EXPRESSION
                  ELSE
                     STORE '*** ERROR - INVALID ARITHMETIC'+;
                           ' EXPRESSION' TO ERROR:MSG
                     STORE T TO ERROR
                     RETURN
                  ENDIF (*expression = ')'*)
               ENDIF (*expression = '_' *)
            ENDIF(* expression $'0123456789 *)
         ENDIF (* expression = '+'or '-' or '/' or '*'*)
      ENDIF(* expression = '(' *)
      IF $(EXPRESSION,1,1) = ' '
         STORE EXPRESSION TO UNPURGED
         DO PURGEBLANK
         STORE PURGED TO EXPRESSION
      ENDIF
   ENDDO (*while expression <> ' ' *)
   USE B:PSTACK
   GOTO BOTTOM
   IF # > 0
      STORE 'ERROR - INVALID ARITHMETIC EXPRESSION' TO ERROR:MSG
      STORE T TO ERROR
      RETURN
   ELSE
      STORE ' ' TO EXP
      USE B:VSTACK
      DO WHILE .NOT. EOF
```

```
      STORE EXP + TRIM(VALUE) TO EXP
      SKIP
   ENDDO
   IF NO:VALUE
      DO REPLACE1
   ELSE
      STORE &EXP TO MRESULT
      STORE '9' TO BASE
      STORE 1 TO LEN
      DO WHILE MRESULT - VAL(BASE) >= 1
         STORE '9' + '&BASE' TO BASE
         STORE LEN + 1 TO LEN
      ENDDO
      STORE LEN + MDEC + 1 TO LEN
      STORE STR(MRESULT,LEN,MDEC) TO RESULT
      USE B:AVTABLE
      GOTO I
      REPLACE VALUE WITH RESULT
      RELEASE MRESULT,BASE,LEN,RESULT,EXP
   ENDIF (* no:value *)
ENDIF (* # > 0 *)


*********************************************************************
*                          CHKFILE.PRG                              *
* This procedure, which is called by procedure SCANFN.prg,          *
* checks a new file name to see if it is valid or not.              *
*********************************************************************

STORE F TO OK
IF LEN(FNAME) = 1 .AND. FNAME = ' '
   STORE '*** ERROR - MISSING FILE NAME' TO ERROR:MSG
   RETURN
ELSE
   IF LEN(FNAME) > 8
      STORE  '*** ERROR - FILE NAME CANNOT EXCEEDS 8 CHARACTERS';
             TO ERROR:MSG
      RETURN
   ELSE
      IF FILE ('B:&FNAME')
         STORE '*** ERROR - &FNAME file already exit ';
               TO ERROR:MSG
         RETURN
      ELSE
         IF  @(' ',FNAME) <> 0
            STORE  '*** ERROR - EMBEDDED BLANKS ARE NOT'+;
                   ' PERMITTED IN THE FILE NAME' TO  ERROR:MSG
            RETURN
         ELSE
            IF $(FNAME,1,1) < 'A' .OR. $(FNAME,1,1) > 'Z'
                STORE '*** ERROR - FILE NAME SHOULD BEGIN WITH'+;
                      ' A LETTER' TO ERROR:MSG
                RETURN
            ELSE
              STORE LEN(FNAME) TO STR:LEN
```

```
                STORE 0 TO I
                DO WHILE I < STR:LEN
                   STORE I + 1 TO I
                   IF ($(FNAME,I,1) < 'A' .OR. $(FNAME,I,1) > 'Z');
                  .AND. ($(FNAME,I,1) < '0' .OR. $(FNAME,I,1) > '9')
                         STORE  '*** ERROR - FILE NAME CAN ONLY'+;
                                ' CONSIST OF LETTER OR DIGIT';
                                TO ERROR:MSG
                         RETURN
                   ENDIF (* invalid symbol in file name *)
                ENDDO (* i < str:len *)
                RELEASE I, STR:LEN
             ENDIF (* file name not begin with letter*)
          ENDIF (* file name consist of a space*)
       ENDIF (* file already exist *)
    ENDIF (*file name too long *)
ENDIF (*missing file name *)
STORE T TO OK
RETURN


*************************************************************************
*                              CHKFLD.PRG                               *
* This procedure, which is called by procedure PARSE1.prg,              *
* checks all field names of a specified file to see if they are         *
* all existing in the file or not.                                      *
*************************************************************************

STORE 0 TO I
DO WHILE I < FILE:NUM
   STORE I + 1 TO I
   STORE 'SFILE' + STR(I,1) TO SFILE
   USE B:&SFILE
   GOTO 2
   DO WHILE .NOT. EOF
      IF FLD:NAME = ' '
         IF ROW1 = ' ' .AND. ROW2 = ' ' .AND. ROW3 = ' '
            DELETE
            PACK
         ELSE
            STORE '*** ERROR - MISSING FIELD NAME' TO ERROR:MSG
            STORE T TO ERROR
            RETURN
         ENDIF
      ENDIF
      SKIP
   ENDDO (*while not eof*)
ENDDO (* while i < file:num *)
STORE 0 TO I
DO WHILE I < FILE:NUM
   STORE I + 1 TO I
   STORE 'SFILE' + STR(I,1) TO SFILE
   USE B:&SFILE
   GOTO BOTTOM
   IF # <= 1
```

```
   STORE '*** ERROR - INCOMPLETE QUERY' TO ERROR:MSG
   STORE T TO ERROR
   RETURN
ELSE
   GOTO 1
   STORE FLD:NAME TO FNAME
   USE B:&FNAME
   COPY STRUCTURE EXTENDED TO B:FIELDS
   USE B:FIELDS
   INDEX ON FIELD:NAME TO B:FLDNX
   SELECT PRIMARY
   USE B:&SFILE
   GOTO 2
   DO WHILE (.NOT. EOF) .AND. (.NOT. ERROR) .AND. ;
            (.NOT. FINISHED)
      STORE FLD:NAME TO  FIELD:NAM
      STORE @('.',FIELD:NAM) TO FOUND
      IF FOUND <> 0
         IF FOUND <> 2
            STORE '*** ERROR - SYNTAX ERROR' TO ERROR:MSG
         ELSE
            STORE $(FIELD:NAM,1,1) TO COMMAND
            IF COMMAND <> 'I'
              STORE '*** ERROR - INVALID COMMAND' TO ERROR:MSG
              STORE T TO ERROR
            ELSE
               DO INSERTION
               STORE T TO FINISHED
            ENDIF
         ENDIF
         LOOP
      ELSE
         STORE TRIM(FIELD:NAM) TO FIELD:NAM
         SELECT SECONDARY
         USE B:FIELDS INDEX B:FLDNX
         FIND &FIELD:NAM
         IF # = 0
            STORE "*** ERROR - COLUMN &field:nam IS NOT "+;
                  "EXISTING IN TABLE &fname" TO ERROR:MSG
            STORE T TO ERROR
            LOOP
         ELSE
            SELECT PRIMARY
            REPLACE P.LEN WITH S.FIELD:LEN;
            P.TYPE WITH S.FIELD:TYPE, P.DEC WITH S.FIELD:DEC
         ENDIF
      ENDIF
      SKIP
   ENDDO  (* not eof  and not error and not finished *)
   SELECT SECONDARY
   USE
   SELECT PRIMARY
   USE
   IF ERROR .OR. FINISHED
      USE
```

```
         DELETE FILE B:FIELDS
         DELETE FILE B:FLDNX.NDX
         RETURN
      ENDIF (* error .or. finished *)
   ENDIF (* # <= 1 *)
ENDDO (* i < file:num *)
RELEASE FNAME,SFILE,FIELD:NAM,FOUND,MAX:REC
DELETE FILE B:FIELDS
DELETE FILE B:FLDNX.NDX
RETURN


*************************************************************************
*                              CLEARFILE.PRG                            *
* This procedure, which is called by procedure RUN.prg and              *
* QBE.prg, deletes all the files that are used as temporary             *
* storage files.                                                        *
*************************************************************************

USE FILES
DO WHILE .NOT. EOF
   STORE TRIM(FNAME) TO MFNAME
   IF FILE('B:&MFNAME')
      DELETE FILE B:&MFNAME
   ENDIF
   SKIP
ENDDO
RETURN


*************************************************************************
*                              CONSTANT.PRG                             *
* This procedure, which is called by procedure CALCULATE.prg,           *
* checks if a constant value is valid or not.                           *
*************************************************************************

STORE $(EXPRESSION,1,1) TO CONSTANT
STORE $(EXPRESSION,2) TO EXPRESSION
DO WHILE $(EXPRESSION,1,1)$'0123456789.'
   STORE CONSTANT + $(EXPRESSION,1,1) TO CONSTANT
   STORE $(EXPRESSION,2) TO EXPRESSION
ENDDO
IF $(CONSTANT,LEN(CONSTANT) ,1) = '.'
   STORE "*** ERROR - NUMBER CANNOT BE ENDED WITH DECIMAL" +;
         " POINT" TO ERROR:MSG
   STORE T TO ERROR
   RETURN
ELSE
   STORE CONSTANT TO MVAL
ENDIF
RELEASE CONSTANT
RETURN
```

```
*************************************************************
*                          COUNT.PRG                        *
* This procedure, which is called by procedure BUILTFUN.prg,*
* counts the record numbers of a file.                      *
*************************************************************

USE B:&FILE
GOTO BOTTOM
IF # = 0
   STORE 0 TO RESULT
ELSE
   COUNT TO RESULT
ENDIF
RETURN


*************************************************************
*                          DCON.PRG                         *
* This procedure, which is called by procedure DRAW.prg,    *
* displays a condition table on the screen to let the user enter*
* the condition statement(s).                               *
*************************************************************

STORE T TO C:EXIST
IF MAXROW = 6
   STORE ROW + 1 TO ROW
ELSE
   STORE ROW + 2 TO ROW
ENDIF (* maxrow = 6*)
STORE 0 TO ROW:COUNT
STORE 15 TO COL
@ ROW,COL SAY '1      C  O  N  D  I  T  I  O  N  S     1'
STORE ROW + 1 TO ROW
DO WHILE ROW <= MAXROW
   IF ROW = MAXROW - 3
       @ ROW, COL SAY '+--------------------------------------+'
   ELSE
     STORE ROW:COUNT + 1 TO ROW:COUNT
     STORE 1 TO COL:COUNT
     STORE 'ROW'+STR(ROW:COUNT,1)+':COL'+STR(COL:COUNT,1) ;
           TO PARSER
     STORE '                                          ' TO &PARSER
     @ ROW,COL SAY '|'
     @ ROW,COL + 1 GET &PARSER;
                PICTURE '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
     @ ROW,COL + 40 SAY '|'
  ENDIF (* row = maxrow - 3*)
  STORE ROW + 1 TO ROW
ENDDO (* row <= maxrow *)
READ
DO STORE2
RETURN
```

```
*********************************************************************
*                            DISPANS.PRG                            *
* This procedure,which is called by procedure PARSE1.prg,           *
* displays the result of a query either on the screen or on a       *
* hard copy.                                                        *
*********************************************************************

ERASE
IF .NOT. BUILDFUN
   USE B:SAVE
   GOTO  BOTTOM
   IF # = 0
      @ 1,10 SAY '*** NO RECORD IS QUALIFIED TO YOUR QUERY ***'
      STORE T TO NO:REC
      STORE 'NO RECORD IS QUALIFIED TO YOUR QUERY' TO ERROR:MSG
      RETURN
   ENDIF
   USE
ENDIF
STORE F TO OK
DO WHILE .NOT. OK
   STORE ' ' TO PRINTING
   @ 1,10
   @ 1,10 SAY 'Do you want hard copy (Y/N)?' GET PRINTING ;
     PICTURE '!'
   READ
   IF PRINTING <> 'Y' .AND. PRINTING <> 'N'
         LOOP
   ELSE
         STORE T TO OK
   ENDIF (* printing <> Y or N *)
ENDDO (* not ok *)
STORE 'N' TO TABLE
IF PRINTING = 'Y'
   STORE F TO OK
   DO WHILE .NOT. OK
      STORE ' ' TO TABLE
      @ 2,10
      @ 2,10 SAY 'Report in table format? (Y/N)' GET TABLE;
        PICTURE '!'
      READ
      IF !(TABLE) <> 'Y' .AND. !(TABLE) <> 'N'
            LOOP
      ELSE
            STORE T TO OK
      ENDIF (* table <> 'Y' and 'N'*)
   ENDDO (* ok *)
   ERASE
   SET TALK ON
   @ 1,15 SAY '*** BE SURE YOUR PRINTER IS ON ***' + CHR(7)
   @ 2,15
   ACCEPT '                Hit <RETURN> when ready' TO ACTION
   RELEASE ACTION
   SET TALK OFF
ENDIF (*printing = Y *)
```

```
ERASE
@ 1,5 SAY 'WAIT !'
IF BUILDFUN
   DO BUILTANS
ELSE
   IF TABLE = 'Y'
      DO TABFORM
   ELSE
      DO NTABFORM
   ENDIF
ENDIF


*********************************************************************
*                          DISPFILE.PRG                             *
* This procedure, which is called by procedure PROCFILE.prg,        *
* displays answers which concern file names existing in the         *
* user's disk, or the structure of a specific file.                 *
*********************************************************************

ERASE
STORE F TO OK
DO WHILE .NOT. OK
   STORE ' ' TO PRINTING
   @ 1,10
   @ 1,10 SAY 'Do you want hard copy (Y/N)?' GET PRINTING ;
               PICTURE '!'
   READ
   IF PRINTING <> 'Y' .AND. PRINTING <> 'N'
         LOOP
   ELSE
         STORE T TO OK
   ENDIF (* printing <> Y or N *)
ENDDO (* not ok *)
IF PRINTING = 'Y'
   SET PRINT ON
ENDIF
ERASE
SET EXACT ON
DO CASE
    CASE COMMAND = 'P'
      *** display all the file names that are created by user ***
      ? 'The following file(s) is (are) created by you:'
      ?
      USE B:COLFILES
      DISP ALL OFF

    CASE COMMAND =  ',P'
      *** display the structure of a file name which is ***
      ***  specified by user ***
      ? 'The structure of file &fname is shown as following:'
      ?
      USE B:&FNAME
      DISP STRUCTURE
      RELEASE FNAME
```

```
    CASE COMMAND = 'P,P'
      *** display all the file name and their corresponding ***
      *** strutures ***
      ? 'All the files and their corresponding structures are'
      ?? ' shown as following:'
      SELECT PRIMARY
      USE B:COLFILES
      DO WHILE .NOT. EOF
         STORE FNAME TO FILE:NAME
         SELECT SECONDARY
         USE B:&FILE:NAME
         ?
         ?
         DISP STRUCTURE
         IF !(PRINTING) = 'Y'
             SET PRINT OFF
         ENDIF (*printing = Y *)
         ?
         ?
         ACCEPT 'Hit <RETURN> to continue or Q to quit' TO ACTION
         ERASE
         IF !(ACTION) = 'Q'
             SELECT PRIMARY
             USE
             SELECT SECONDARY
             USE
             RETURN
         ELSE
             IF !(PRINTING) = 'Y'
                 SET PRINT ON
             ENDIF (* printing = Y *)
         ENDIF (*action = Q *)
         SELECT PRIMARY
         SKIP
      ENDDO (* not eof *)
      SELECT PRIMARY
      USE
      SELECT SECONDARY
      USE
      RELEASE FILE:NAME, ACTION
ENDCASE
SET PRINT OFF
@ 22,1 SAY 'YOUR QUERY HAS BEEN DONE '
SET EXACT OFF
RETURN
```

```
*****************************************************************
*                            DRAW.PRG                           *
*  This procedure, which is called by procedure QBE.prg,        *
*  interactes with the user to see what kind of table (general, *
*  result or condition table) the user needs.                   *
*****************************************************************


SET TALK OFF
ERASE
STORE 6 TO MAXROW
STORE 1 TO ROW
STORE 0 TO FILE:NUM
STORE F TO R:EXIST
STORE F TO C:EXIST
STORE 'G' TO ANSWER
STORE T TO MORE
DO DTABLE
USE CONDITION
COPY TO B:CONDITION
@ 23, 1
DO WHILE MORE
   STORE F TO OK
   DO WHILE .NOT. OK
      STORE ' ' TO ANSWER
      @ 23,0 SAY 'DO YOU NEED ANOTHER TABLE (Y/N)?' GET ANSWER ;
             PICTURE '!'
      READ
      STORE 0 TO TIMER
      DO WHILE TIMER <= 5
          STORE TIMER + 1 TO TIMER
      ENDDO (* timer <= 5 *)
      @ 23,0
      IF ANSWER <> 'Y' .AND. ANSWER <> 'N'
            LOOP
      ELSE
            STORE T TO OK
      ENDIF
   ENDDO (* ok *)
   RELEASE OK
   IF ANSWER = 'Y'
     IF FILE:NUM < 2
       IF R:EXIST
         STORE '(G)ENERAL TABLE  (C)ONDITION BOX,  (E)XIT ,'+;
               'CHOOSE A LETTER' TO QUESTION
         STORE "ANSWER = 'G'.OR. ANSWER = 'C'.OR. ANSWER = 'E'";
                TO CORRECT
       ELSE
         STORE '(G)ENERAL TABLE  (R)ESULT TABLE  (C)ONDITION '+;
               ' BOX  (E)XIT, CHOOSE A LETTER ===>' TO QUESTION
         STORE "ANSWER = 'G' .OR. ANSWER = 'R' .OR."+;
               " ANSWER = 'C' .OR. ANSWER = 'E'" TO CORRECT
       ENDIF  (*result:exit*)
     ELSE
       IF R:EXIST
```

```
        STORE '(C)ONDITION BOX  (E)XIT, CHOOSE A LETTER ===>';
              TO QUESTION
        STORE "ANSWER = 'C' .OR. ANSWER = 'E'" TO CORRECT
      ELSE
        STORE '(R)ESULT TABLE (C)ONDITION BOX, (E)XIT,CHOOSE'+;
              ' A LETTER  ===>' TO QUESTION
        STORE "ANSWER = 'R' .OR. ANSWER = 'C' .OR." +;
              " ANSWER = 'E'" TO CORRECT
      ENDIF  (* result:exit *)
    ENDIF (* file:num <= 2 *)
    DO GETANS
    RELEASE QUESTION, CORRECT
    @ 23,1
    IF ROW >= 22
       *** screen is full ***
       ERASE
       STORE 6 TO MAXROW
       STORE 1 TO ROW
    ELSE
       STORE MAXROW + 8 TO MAXROW
       STORE ROW + 1 TO ROW
    ENDIF (* row >= 22 *)
    IF ANSWER = 'G' .OR. ANSWER = 'R'
        DO DTABLE
    ELSE
        IF ANSWER = 'C'
           DO DCON
        ENDIF
    ENDIF(* answer = 'G' .or. 'R'*)
  ELSE
    RELEASE MAXROW,MAXCOL,ROW,COL
    STORE F TO MORE
  ENDIF (*answer = 'y' *)
ENDDO (* more *)
RETURN


*************************************************************************
*                              DTABLE.PRG                               *
* This procedure,which is called by procedure DRAW.prg, displays*
* either a general table or a result table on the screen to     *
* allow the user to enter the query.                            *
*************************************************************************

SET TALK OFF
STORE 0 TO COL
IF ANSWER = 'G'
   STORE F TO OK
   DO WHILE .NOT. OK
      STORE '                ' TO FILE:NAME
      SET COLON ON
      @ ROW,COL SAY 'ENTER TABLE NAME (AND COMMAND IF ANY)';
        GET FILE:NAME PICTURE '!!!!!!!!!!!!'
      READ
      SET COLON OFF
```

```
      DO SCANFN
      @ 23, 1
      IF OK
         IF FILE:ONLY
            STORE F TO MORE
            RETURN
         ENDIF (* command <> ' ' *)
      ELSE
         @ 23,1 SAY ERROR:MSG
         STORE 0 TO TIMER
         DO WHILE TIMER <= 100
            STORE TIMER + 1 TO TIMER
         ENDDO (* while timer <= 100*)
         @ 23,1
         RELEASE TIMER, ERROR:MSG
      ENDIF (* ok *)
   ENDDO (* not ok *)
   STORE FILE:NUM + 1 TO FILE:NUM
ENDIF (* answer = 'G'*)
IF ANSWER = 'R'
   STORE  T TO R:EXIST
   @ ROW, COL SAY 'RESULT TABLE'
   STORE F TO OK
   DO WHILE .NOT. OK
      STORE '         ' TO FNAME
      SET COLON  ON
      @ ROW,COL+17 SAY 'ENTER A NEW TABLE NAME (IF RESULT NEED'+;
                       ' TO BE SAVED)' GET FNAME  PICTURE '!!!!!!!!'
      SET COLON OFF
      READ
      IF FNAME = '        '
         STORE F TO SNAPSHOT
         STORE T TO OK
         LOOP
      ELSE
         STORE TRIM(FNAME) TO FNAME
         IF $(FNAME,1,1) = ' '
            STORE FNAME TO UNPURGED
            DO PURGEBLANK
            STORE PURGED TO FNAME
         ENDIF
         DO CHKFILE
         IF .NOT. OK
            @ 23,1 SAY ERROR:MSG
            STORE 0 TO TIMER
            DO WHILE TIMER <= 100
               STORE TIMER + 1 TO TIMER
            ENDDO (* timer <= 100 *)
            @ 23,1
            RELEASE ERROR:MSG
            LOOP
         ELSE
            STORE T TO SNAPSHOT
            STORE FNAME TO SNAP:NAME
         ENDIF(* not ok *)
```

```
      ENDIF (*fname = '         '*)
   ENDDO (* not ok*)
ENDIF (* answer = 'R'*)
IF MAXROW = 6
   STORE ROW + 1 TO ROW
ELSE
   STORE ROW + 2 TO ROW
ENDIF (* maxrow = 6*)
STORE 1 TO ROW:COUNT
STORE 0 TO COL:COUNT
STORE 7 TO MAXCOL
DO WHILE ROW <= MAXROW
   IF ROW = MAXROW - 3
      @ ROW,COL SAY '--+----------+----------+----------'+;
                    '+----------+----------+----------'
   ELSE
      DO WHILE COL:COUNT < MAXCOL
        STORE COL:COUNT + 1 TO COL:COUNT
        IF ROW:COUNT <> 1 .OR. COL:COUNT <> 1
           STORE 'ROW'+STR(ROW:COUNT,1)+':COL'+STR(COL:COUNT,1);
           TO PARSER
           IF COL:COUNT = 1
               STORE '  ' TO &PARSER
           ELSE
               STORE '            ' TO &PARSER
           ENDIF (* col:count = 1 *)
           @ ROW,COL GET &PARSER  PICTURE '!!!!!!!!!!!!'
        ENDIF (* row:count <> 1 or col:count <> 1*)

        IF COL:COUNT = 1
           STORE COL + 2 TO COL
        ELSE
           STORE COL + 12 TO COL
        ENDIF (* col:count = 1 *)

        IF COL:COUNT <> MAXCOL
            @ ROW,COL SAY '|'
        ENDIF (* col:count <> maxcol *)
        STORE COL + 1 TO COL
      ENDDO (* col:count < maxcol *)
      STORE ROW:COUNT + 1 TO ROW:COUNT
   ENDIF (* row = maxrow - 3 *)
   STORE ROW + 1 TO ROW
   STORE 0 TO COL, COL:COUNT
ENDDO (* row < maxrow *)
READ
DO STORE1
```

```
*************************************************************************
*                                GETANS.PRG                             *
* This procedure, which is called by procedure DRAW.prg,                *
* obtains the user's answer from the keyboard.                          *
*************************************************************************

STORE F TO OK
DO WHILE .NOT. OK
   STORE ' ' TO ANSWER
   @ 23,0  SAY  QUESTION GET ANSWER   PICTURE '!'
   READ
   STORE 0 TO TIMER
   DO WHILE TIMER <= 10
      STORE TIMER + 1 TO TIMER
   ENDDO (* timer <= 5 *)
   @ 23,0
   IF &CORRECT
      STORE T TO OK
   ELSE
      LOOP
   ENDIF
ENDDO (* not ok *)
RETURN
```

```
*************************************************************************
*                              MAKEOUT1.PRG                             *
* This procedure, which is called by procedure PARSE1.prg, scans*
* the general table and stores all the field names with a print *
* statement (i.e., P.) onto the PRINT stack.                            *
*************************************************************************

USE PRNSTACK
COPY TO B:PRNSTACK
SELECT PRIMARY
USE B:SFILE1
GOTO 2
DO WHILE .NOT. EOF
   SELECT SECONDARY
   USE B:PRNSTACK
   APPEND BLANK
   REPLACE S.ATTR  WITH P.FLD:NAME
   SELECT PRIMARY
   SKIP
ENDDO
SELECT PRIMARY
USE
SELECT SECONDARY
USE
```

```
*********************************************************************
*                          MAKEOUT2.PRG                             *
* This procedure, which is called by procedure PARSE1.prg, scans*
* the result table and stores all the field names with a print  *
* statement (i.e., P.) onto the PRINT stack.                        *
*********************************************************************

USE PRNSTACK
COPY TO B:PRNSTACK
SELECT PRIMARY
USE B:RESULT
GOTO 2
DO WHILE .NOT. EOF
   IF FLD:NAME = ' '
      IF ROW1 = ' ' .AND. ROW2 = ' ' .AND. ROW3 = ' '
         DELETE
         PACK
      ELSE
         STORE '***ERROR - MISSING COLUMN NAME IN THE RESULT';
               ' TABLE' TO ERROR:MSG
         STORE T TO ERROR
         RETURN
      ENDIF (* row1 = ' ' and row2 = ' ' and row3 = ' ' *)
   ENDIF (* fld:name = ' ' *)
   SKIP
ENDDO (* while not eof *)
GOTO BOTTOM
IF # <= 1
   STORE '***ERROR - INSUFFICIENT DATA ENTRY IN RESULT TABLE';
         TO ERROR:MSG
   STORE T TO ERROR
   RETURN
ELSE
   GOTO 1
   STORE F TO P:ALL
   DO WHILE .NOT. EOF
      IF $(ROW1,1,1) = 'P' .OR. $(ROW2,1,1) = 'P' .OR. ;
         $(ROW3,1,1) = 'P'
         STORE 'P' TO OPERATOR
         STORE '.' TO SYMBOL
         DO CHKOTHER
         IF ERROR
            RETURN
         ENDIF
         IF # = 1
            STORE T TO P:ALL, OUTPUT
         ELSE
            IF P:ALL
               STORE '*** ERROR - TOO MANY P. OPERATORS IN'+;
                     ' RESULT TABLE' TO ERROR:MSG
               STORE T TO ERROR
               RETURN
            ELSE
               STORE T TO OUTPUT
               STORE TRIM(FLD:NAME) TO FIELD:NAME
```

```
            DO SEARCH
            IF ERROR
               RETURN
            ENDIF
            STORE 'ROW' + STR(NUM,1) TO ROW:NO
            STORE $(&ROW:NO,3) TO VARIABLE
            IF $(VARIABLE,1,1) = ' '
               STORE VARIABLE TO UNPURGED
               DO PURGEBLANK
               STOR PURGED TO VARIABLE
            ENDIF
            IF $(VARIABLE,1,1) <> '_'
               STORE '***ERROR - SYNTAX ERROR IN RESULT'+;
                     ' TABLE' TO ERROR:MSG
               STORE T TO ERROR
               RETURN
            ELSE
               SELECT SECONDARY
               USE B:PRNSTACK
               APPEND BLANK
               REPLACE S.ATTR WITH P.FLD:NAME;
                       S.VARIABLE WITH VARIABLE
            ENDIF (*variable <> '_' *)
         ENDIF (*p: all *)
      ENDIF(*  # = 1 *)
   ELSE
      IF # <> 1
        IF .NOT. P:ALL
           STORE '***ERROR - MISSING P. OPERATOR IN RESULT'+;
                 ' TABLE' TO ERROR:MSG
           STORE T TO ERROR
           RETURN
        ELSE
           STORE TRIM(FLD:NAME) TO FIELD:NAME
           DO SEARCH
           IF ERROR
              RETURN
           ENDIF
           STORE 0 TO I
           DO WHILE I < 3
              STORE I + 1 TO I
              STORE 'ROW' + STR(I,1) TO ROW:NO
              IF &ROW:NO <> ' '
                 IF $(&ROW:NO,1,1) = '_'
                    SELECT SECONDARY
                    USE B:PRNSTACK
                    APPEND BLANK
                    REPLACE S.ATTR WITH P.FLD:NAME;
                            S.VARIABLE WITH &ROW:NO
                    STORE 3 TO I
                    LOOP
                 ELSE
                    STORE '***ERROR - SYNTAX ERROR';
                                                  TO ERROR:MSG
                    STORE T TO ERROR
```

```
                        RETURN
                     ENDIF
                  ENDIF
               ENDDO
            ENDIF
            SELECT SECONDARY
            USE
         ENDIF
      ENDIF
      SELECT PRIMARY
      SKIP
   ENDDO
ENDIF
SELECT PRIMARY
USE
SELECT SECONDARY
USE
RELEASE ROW:NO,OPERATOR,SYMBOL,NUM,P:ALL



*************************************************************************
*                              MAXORMIN.PRG                             *
* This procedure, which is called by procedure BUILTFUN.prg,            *
* finds the maximum or minimum value of a  numerical field.             *
*************************************************************************

USE B:&FILE
GOTO BOTTOM
IF # = 0
   STORE '***ERROR - NO RECORD AVAILABLE, CANNOT PROCEED'+;
         ' &mbuiltfun' TO ERROR:MSG
   STORE T TO ERROR
   RETURN
ELSE
   GOTO TOP
   STORE &MATTR TO TARGET
   IF $(MBUILTFUN,1,4) = 'MAX.'
      STORE '>' TO OPERATOR
   ELSE
      STORE '<' TO OPERATOR
   ENDIF
   DO WHILE .NOT. EOF
      IF &MATTR &OPERATOR TARGET
         STORE &MATTR TO TARGET
      ENDIF
      SKIP
   ENDDO (*not eof*)
   STORE TARGET TO RESULT
ENDIF(* # = 0 *)
RELEASE TARGET,OPERATOR
RETURN
```

```
*****************************************************************
*                          NTABFORM.PRG                         *
* This procedure, which is called by procedures DISPANS.prg and *
* TABFORM.prg, displays the result of a query in linear format  *
* on the screen.                                                *
*****************************************************************

ERASE
IF PRINTING = 'Y'
   SET PRINT ON
ENDIF
?
? 'The answer to your query is shown as follows:'.
SELECT PRIMARY
USE B:SAVE
COPY STRUCTURE EXTENDED TO B:STRU
STORE 0 TO I
*** display each record's content ***
DO WHILE .NOT. EOF
   STORE I + 1 TO I
   STORE STR(I,5) TO NUM
   ?
   ?
   ? 'RECORD#  &NUM'
   ?

   *** display each field's content ***
   SELECT SECONDARY
   USE B:STRU
   GOTO BOTTOM
   STORE # TO MAX:REC
   STORE 0 TO J
   DO WHILE J < MAX:REC
      STORE J + 1 TO J
      GOTO J
      STORE FIELD:NAME TO FLD:NAME
      *** check if the field length is more than 65 characters***
      *** if so display rest content to the next line          ***
      STORE 65 TO MAX:LEN
      IF FIELD:TYPE <> 'N'
         SELECT PRIMARY
         STORE TRIM(&FLD:NAME) TO CONTENT
         STORE LEN(CONTENT) TO C:LEN
         IF C:LEN > MAX:LEN
            STORE $(CONTENT,1,MAX:LEN) TO SUB:CONT
            ? '&FLD:NAME : &SUB:CONT'
            STORE $(CONTENT,MAX:LEN + 1) TO SUB:CONT
            STORE LEN(SUB:CONT) TO C:LEN
            STORE (C:LEN / MAX:LEN) TO LINE:NEED
            STORE INT(LINE:NEED) TO LINE:NEED
            STORE C:LEN - LINE:NEED * MAX:LEN TO REMAINDER
            IF REMAINDER > 0
               STORE LINE:NEED + 1 TO LINE:NEED
            ENDIF (* remainder > 0 *)
            STORE 1 TO BEGIN
```

```
            STORE 0 TO K
            DO WHILE K < LINE:NEED
               STORE $(SUB:CONT,BEGIN,MAX:LEN) TO SUB:CONT
               ?? '                &SUB:CONT'
               STORE K + 1 TO K
               STORE MAX:LEN + BEGIN TO BEGIN
            ENDDO (* k < line:need *)
         ELSE
            ? '&FLD:NAME : &CONTENT'
         ENDIF (* c:len > max:len *)
      ELSE
         STORE FIELD:LEN TO FLD:LEN
         STORE FIELD:DEC TO FLD:DEC
         SELECT PRIMARY
         ? '&FLD:NAME : '+ STR(&FLD:NAME,FLD:LEN,FLD:DEC)
      ENDIF (* p.type <> 'N' *)
      SELECT SECONDARY
      SKIP
   ENDDO (* j <= no *)
   IF !(PRINTING) = 'Y'
       SET PRINT OFF
   ENDIF
   ?
   ACCEPT 'Hit <RETURN> to continue or Q to quit' TO ACTION
   IF !(ACTION) = 'Q'
       SELECT PRIMARY
       USE
       SELECT SECONDARY
       USE
       RETURN
   ELSE
       ERASE
   ENDIF (* action = 'Q'*)
   IF !(PRINTING) = 'Y'
      SET PRINT ON
   ENDIF
   SELECT PRIMARY
   SKIP
ENDDO
SELECT PRIMARY
USE
SELECT SECONDARY
USE
SET PRINT OFF
@ 22,1 SAY 'YOUR QUERY HAS BEEN DONE'
RETURN
```

```
*****************************************************************
*                          PARSE1.PRG                           *
*   This procedure, which is called by procedure RUN.prg, parses*
*   the user's query so that information from the data base     *
*   can be retrieved.                                           *
*****************************************************************

@ 23,1 SAY 'RUNNING, PLEASE WAIT FOR A WHILE !'
STORE F TO ERROR,FINISHED,OUTPUT,UNION,JOIN,BUILDFUN,NO:REC
DO CHKFLD
IF ERROR .OR. FINISHED
   RETURN
ENDIF
USE B:SFILE1
IF $(ROW1,1,1)$'ID' .OR. $(ROW2,1,1) $ 'ID' .OR. ;
   $(ROW3,1,1) $ 'ID'
   STORE 'ID' TO OPERATOR
   STORE '.' TO SYMBOL
   DO CHKOTHER
   IF ERROR
      RETURN
   ELSE
      STORE 'ROW' + STR(NUM,1) TO ROW:NO
      STORE $(&ROW:NO,1,2) TO COMMAND
   ENDIF
   RELEASE OPERATOR,SYMBOL,NUM
   DO CONSQUERY
   DO DOCOM
   RETURN
ELSE
   IF $(ROW1,1,1) = 'P' .OR. $(ROW2,1,1) = 'P' .OR. ;
      $(ROW3,1,1) = 'P'
      STORE 'P' TO OPERATOR
      STORE '.' TO SYMBOL
      DO CHKOTHER
      RELEASE OPERATOR,SYMBOL,NUM
      IF ERROR
         RETURN
      ENDIF
      STORE T TO OUTPUT
      DO MAKEOUT1
   ELSE
      IF $(ROW1,1,1) = ' ' .OR. $(ROW2,1,1) = ' ' .OR. ;
         $(ROW3,1,1) = ' '
         STORE ' ' TO OPERATOR
         STORE ' ' TO SYMBOL
         DO CHKOTHER
         IF ERROR
            RETURN
         ENDIF
         RELEASE OPERATOR,SYMBOL,NUM
      ELSE
         STORE '***ERROR - INVALID SYSTEM OPERATOR' TO ERROR:MSG
         STORE T TO ERROR
         RETURN
```

```
      ENDIF
   ENDIF
ENDIF
IF R:EXIST
   IF OUTPUT
      STORE '*** ERROR - TOO MANY SYSTEM OPERATORS' TO ERROR:MSG
      STORE T TO ERROR
      RETURN
   ELSE
      DO MAKEOUT2
      IF ERROR
         RETURN
      ENDIF
   ENDIF
ENDIF
DO AVTABLE
IF .NOT. OUTPUT
   STORE '*** ERROR - SYSTEM OPERATOR NOT SPECIFIED' TO ERROR:MSG
   STORE T TO ERROR
ENDIF
IF ERROR
   RETURN
ENDIF
DO SCANPRN
USE B:AVTABLE
GOTO BOTTOM
STORE # TO MAX:AV
IF MAX:AV > 0
   USE CODESTACK
   COPY TO B:CODESTACK
   STORE 0 TO I
   DO WHILE I < FILE:NUM
      STORE I + 1 TO I
      STORE 'FILE' + STR(I,1) TO FILE
      STORE &FILE TO FILE
      STORE 'TEMP' + STR(I,1) TO TEMP
      USE B:&FILE
      COPY TO B:&TEMP
   ENDDO
   USE
   IF C:EXIST
      DO SCANCOND
   ENDIF
   DO SCANAV1
   IF ERROR
      RETURN
   ENDIF
   IF MAX:AV > 0
      DO SCANAV2
      IF ERROR
         RETURN
      ENDIF
   ENDIF
   IF MAX:AV > 0
      DO SCANAV3
```

```
      IF ERROR
         RETURN
      ENDIF
   ENDIF
   DO RETRIE1
   IF ERROR
      RETURN
   ENDIF
ELSE
   USE B:&FILE1
   COPY FIELD &OUT:FLDS TO B:SAVE
ENDIF
IF R:EXIST
   IF SNAPSHOT
      USE B:SAVE
      COPY TO B:&SNAP:NAME
   ENDIF
ENDIF
DO DISPANS
RETURN


*************************************************************************
*                              PARSE2.PRG                               *
* This procedure, which is called by procedure AVTABLE.prg,             *
* parses the string which follows a P. operator.                        *
*************************************************************************

IF FILE:NUM = 1 .AND. K = 1
   IF OUTPUT
      STORE '*** ERROR- TOO MANY SYSTEM OPERATORS' TO ERROR:MSG
      STORE T TO ERROR
      RETURN
   ENDIF
ENDIF
STORE T TO OUTPUT
SELECT SECONDARY
USE B:PRNSTACK
APPEND BLANK
REPLACE ATTR WITH FIELD:NAME
SELECT SECONDARY
USE
STORE $(PARSER,1,2) TO MPREFIX
STORE $(PARSER,3) TO PARSER
IF $(PARSER,1,1) = ' '
   STORE PARSER TO UNPURGED
   DO PURGEBLANK
   STORE PURGED TO PARSER
ENDIF
IF PARSER = ' '
   STORE T TO BLANK
   RETURN
ELSE
  IF $(PARSER,1,1) = '_'
     IF $(PARSER,1,2) = ' '
```

```
        STORE '*** ERROR - NO BLANK CAN BE EMBEDDED INTO'+;
              ' EXAMPLE ELEMENT' TO ERROR:MSG
        STORE T TO ERROR
      ELSE
        STORE TRIM(PARSER) TO MVALUE
      ENDIF
    ELSE
      IF $(PARSER,1,1) = '('
        STORE TRIM(PARSER) TO MVALUE
      ELSE
        IF $(PARSER,1,1) = '>' .OR. $(PARSER,1,1) = '<'
          DO  PARSE3
        ELSE
          IF $(PARSER,1,4) = 'ALL.' .OR. $(PARSER,1,4) = 'UNQ.'
            DO PARSE4
          ELSE
            IF $(PARSER,1,4) = 'SUM.' .OR. $(PARSER,1,4) = 'CNT.';
            .OR. $(PARSER,1,4) = 'AVG.'.OR. $(PARSER,1,4) = 'MAX.';
            .OR. $(PARSER,1,4) = 'MIN.'
                  DO  PARSE5
            ELSE
              IF $(PARSER,1,3) = 'AO.' .OR. $(PARSER,1,3) = 'DO.'
                DO  PARSE6
              ELSE
                IF $(PARSER,1,1)$'0123456789'
                   STORE TRIM(PARSER) TO MVALUE
                ELSE
                  IF $(PARSER,1,1)$'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
                     STORE TRIM(PARSER) TO MVALUE
                  ELSE
                     STORE '***SYNTAX ERROR ' TO ERROR:MSG
                    STORE T TO ERROR
                  ENDIF
                ENDIF
               ENDIF
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
ENDIF

*************************************************************************
*                              PARSE3.PRG                               *
*  This  procedure, which is called by procedures AVTABLE.prg           *
*  or PARSE2.prg, parses the string having  binary operator  >,         *
*  <, >=, <=, or  <>.                                                   *
*************************************************************************

IF MTYPE <> 'N'
   STORE '*** ERROR - COLUMN &field:name IS NOT A NUMERICAL '+;
         'TYPE, CANNOT DO COMPARISION' TO ERROR:MSG
   STORE T TO ERROR
   RETURN
ELSE
```

```
  IF $(PARSER,2,1) = '='
     STORE $(PARSER,1,2) TO MOPERATOR
     STORE $(PARSER,3) TO PARSER
  ELSE
    IF $(PARSER,1,1) = '<' .AND. $(PARSER,2,1) = '>'
       STORE $(PARSER,1,2) TO MOPERATOR
       STORE $(PARSER,3) TO PARSER
    ELSE
       STORE $(PARSER,1,1) TO MOPERATOR
       STORE $(PARSER,2) TO PARSER
    ENDIF
  ENDIF
ENDIF
IF $(PARSER,1,1) = ' '
   STORE PARSER TO UNPURGED
   DO PURGEBLANK
   STORE PURGED TO PARSER
ENDIF
IF $(PARSER,1,1) <> ' '
   IF $(PARSER,1,1)$'_(0123456789'
       STORE TRIM(PARSER) TO MVALUE
   ELSE
      STORE '*** ERROR - SYNTAX ERROR ' TO ERROR:MSG
      STORE T TO ERROR
   ENDIF
ELSE
   STORE '***ERROR - INCOMPLETE EXPRESSION' TO ERROR:MSG
   STORE T TO ERROR
ENDIF
RETURN


*************************************************************************
*                              PARSE4.PRG                               *
* This procedure, which is called by procedures AVTABLE.prg or          *
* PARSE2.prg, parses the string having UNQ. or ALL. operator.           *
*************************************************************************

IF $(PARSER,1,4) = 'UNQ.'
   STORE 'UNQ.' TO MBUILTFUN
ENDIF
STORE $(PARSER,5) TO PARSER
IF $(PARSER,1,1) = ' '
   STORE PARSER TO UNPURGED
   DO PURGEBLANK
   STORE PURGED TO PARSER
ENDIF
IF $(PARSER,1,1) = '_' .OR. $(PARSER,1,1) = ' '
   STORE TRIM(PARSER) TO MVALUE
ELSE
   STORE '*** ERROR - INVALID QUERY'  TO ERROR:MSG
   STORE T TO ERROR
ENDIF
RETURN
```

```
*****************************************************************
*                          PARSE5.PRG                           *
* This procedure, which is called by procedures AVTABLE.prg or  *
* PARSE2.prg, parses the string having a built-in function      *
* (MAX., MIN., CNT., SUM., AVG.).                              *
*****************************************************************

IF $(PARSER,1,4) <> 'CNT.'
   IF TYPE <> 'N'
      STORE '*** ERROR - COLUMN &field:name IS NOT A NUMERCIAL '+;
            'TYPE, CANNOT DO CALCULATION' TO ERROR:MSG
      STORE T TO ERROR
      RETURN
   ENDIF
ENDIF
STORE $(PARSER,1,4) TO MBUILTFUN
STORE $(PARSER,5) TO PARSER
IF $(PARSER,1,1) = ' '
   STORE PARSER TO UNPURGED
   DO PURGEBLANK
   STORE PURGED TO PARSER
ENDIF
IF $(PARSER,1,4) = 'ALL.' .OR. $(PARSER,1,4) = 'UNQ.'
   STORE MBUILTFUN + $(PARSER,1,4) TO MBUILTFUN
ELSE
  STORE '*** ERROR - MISSING ALL. OR UNQ. IN BUILT-IN FUNCTION'+;
        ' &mbuiltfun' TO ERROR:MSG
   STORE T TO ERROR
   RETURN
ENDIF
STORE $(PARSER,5) TO PARSER
IF $(PARSER,1,1) = ' '
   STORE PARSER TO UNPURGED
   DO PURGEBLANK
   STORE PURGED TO PARSER
ENDIF
IF $(PARSER,1,1) = ' ' .OR. $(PARSER,1,1) = '_'
   STORE TRIM(PARSER) TO MVALUE
ELSE
   STORE '***ERROR- INVALID QUERY' TO ERROR:MSG
   STORE T TO ERROR
ENDIF
RETURN

*****************************************************************
*                          PARSE6.PRG                           *
* This procedure, which is called by procedures AVTABLE.prg or  *
* PARSE2.prg, parses a string having  AO. or  DO. built-in      *
* functions.                                                    *
*****************************************************************

STORE $(PARSER,1,3) TO MBUILTFUN
STORE $(PARSER,4) TO PARSER
IF $(PARSER,1,1) = ' '
   STORE PARSER TO UNPURGED
```

```
   DO PURGEBLANK
   STORE PURGED TO PARSER
ENDIF
IF $(PARSER,1,1) = ' ' .OR. $(PARSER,1,1) = '_'
   STORE TRIM(PARSER) TO MVALUE
ELSE
   STORE '***ERROR - INVALID QUERY' TO ERROR:MSG
   STORE T TO ERROR
ENDIF
RELEASE PURGED,UNPURGED
RETURN


*********************************************************************
*                              POPP.PRG                             *
* This procedure, which is called by procedure CALCULATE.prg,       *
* pops a left parenthesis from a stack if a right parenthesis is    *
* matched.                                                          *
*********************************************************************

USE B:PSTACK
GOTO BOTTOM
IF # > 0
   DELETE
   PACK
ELSE
   STORE '*** ERROR - INVALID ARITHMETIC EXPRESSION';
         TO ERROR:MSG
   STORE T TO ERROR
ENDIF
RETURN

*********************************************************************
*                           PROCFILE.PRG                            *
* This procedure, which is called by proecedure RUN.prg,            *
* processes queries concerned with the file management, such as     *
* displaying all the file names existing in the user's disk,        *
* displaying the structure of a specific file, deleting a file      *
* and creating a file.                                              *
*********************************************************************

SET EXACT ON
DO CASE
     CASE @('P',COMMAND) <>  0
         *** retrieve  data base file name or file structure ***
             DO DISPFILE
     CASE COMMAND = 'U'
         *** rename a data base file ***
         DO  RENAME
     CASE COMMAND = 'I,I'
         *** create a new data base file ***
         @ 23,1 SAY " JUST A SECOND, YOUR QUERY IS PROCESSING "
         DO CRTFILE
     CASE COMMAND = 'D'
         *** delete an existing data base file ****
```

```
         STORE ' ' TO ANS
         @ 23,1 SAY " ARE YOU SURE THAT YOU WANT TO DROP THE"+;
                    " TABLE ? Y/N " GET ANS PICTURE '!'
         READ
         DO CASE
            CASE !(ANS) = 'Y'
                 DELETE FILE B:&FNAME
                 ERASE
                 ? ' &FNAME IS DELETED '
                 RELEASE ANS,FNAME
                 RETURN
            CASE !(ANS) = 'N'
                 ERASE
         ENDCASE
ENDCASE (* COMMAND *)
SET EXACT OFF
RETURN


*************************************************************
*                          PURGEBLANK.PRG                   *
* This procedure, which is called by several precedures, deletes*
* all the unnecessary blanks from a string.                 *
*************************************************************

STORE LEN(UNPURGED) TO MAX:LEN
STORE 1 TO MI
DO WHILE $(UNPURGED,MI,1) = ' ' .AND. MI < MAX:LEN
     STORE  MI + 1  TO  MI
ENDDO
IF $(UNPURGED,MI,1) = ' '
   STORE ' ' TO  PURGED
ELSE
   STORE $(UNPURGED,MI) TO PURGED
ENDIF
RELEASE MI, MAX:LEN
RETURN


*************************************************************
*                            PUSHP.PRG                      *
* This procedure, which is called by proecedure CALCULATE.prg,  *
* pushes a left parenthesis onto a stack.                   *
*************************************************************

USE B:PSTACK
APPEND BLANK
REPLACE PREN WITH MVAL
USE
RETURN
```

```
*************************************************************
*                          PUSHV.PRG                        *
* This procedure, which is called by procedure CALCULATE.prg,*
* pushes all the symbols and values of an arithmetic expression *
* onto a stack.                                             *
*************************************************************

USE B:VSTACK
APPEND BLANK
REPLACE VALUE WITH MVAL
USE
RELEASE MVAL
RETURN


*************************************************************
*                          QBE.PRG                          *
* This procedure displays greeting messages and controls the *
* initial use of the QBE query system.                      *
*************************************************************

CLEAR
SET COLON OFF
SET COLOR TO 11,10,10
SET TALK OFF
ERASE
STORE T TO START
@ 3,5 SAY "********** WELCOME TO QBE QUERY SYSTEM **********"
STORE 6 TO ROW
DO WHILE START
   STORE '        ' TO DAT
   @ ROW,5  SAY "PLEASE ENTER TODAY'S DATE (MM/DD/YY):";
            GET  DAT PICTURE "99/99/99"
   READ
   STORE $(DAT,1,2) TO MO
   STORE $(DAT,4,2) TO DAY
   IF MO < '01' .OR. MO > '12' .OR. DAY < '01' .OR. DAY > '31'
         STORE ROW + 1 TO ROW
         STORE 0 TO TIMER
         @ ROW,5 SAY 'INVALID DATE, PLEASE TRY AGAIN'
         STORE ROW + 1 TO ROW
         DO WHILE TIMER < 5
             STORE TIMER + 1 TO TIMER
         ENDDO
         RELEASE DAT, MO, DAY, TIMER
         LOOP
   ENDIF (* invalid date *)
   STORE ROW + 1 TO ROW
   SET DATE TO &DAT
   STORE F TO START
ENDDO (* start *)
RELEASE START,DAT, MO, DAY
STORE ' ' TO ACTION
@ ROW, 5 SAY 'Please insert your disk to drive B'
ACCEPT '     and press RETURN when ready' TO ACTION
```

```
RELEASE ROW,ACTION

*** show up the table skeleton ***
STORE T TO MORE
DO WHILE MORE
    DO DRAW
    STORE F TO OK
    DO WHILE .NOT.OK
       STORE ' ' TO ANSWER
       @ 23,0 SAY '(R)UN, (E)RASE,(Q)UIT'+;
         ', PLEASE CHOOSE A LETTER ===>' GET ANSWER  PICTURE '!'
       READ
       STORE 1 TO TIMER
       DO WHILE TIMER <= 10
            STORE TIMER + 1 TO TIMER
       ENDDO (* timer <= 10 *)
       IF ANSWER <> 'R'.AND. ANSWER <> 'E'.AND. ANSWER <> 'Q'
            LOOP
       ELSE
            STORE T TO OK
       ENDIF
    ENDDO (* ok *)
    RELEASE OK
    DO CASE
        CASE ANSWER = 'R'
             @ 23,0
             DO RUN
        CASE ANSWER = 'E'
             STORE T TO MORE
             ERASE
        CASE ANSWER = 'Q'
             DO CLEARFILE
             ERASE
             @ 10,30 SAY 'G O O D B Y E !'
             @ 22,0
             QUIT
   ENDCASE
ENDDO (* more *)



*********************************************************************
*                            REPLACE1.PRG                           *
* This procedure, which is called by procedure CALCULATE.prg,       *
* replaces the arithmetic expression with its result.               *
*********************************************************************

STORE 'TEMP' + $(MTABLEID,1,1) TO TEMP
USE B:&TEMP
DO WHILE .NOT. EOF
   STORE &EXP TO RESULT
   STORE '9' TO BASE
   STORE 1 TO LEN
   DO WHILE RESULT - VAL(BASE) >= 1
      STORE '9' +'&BASE' TO BASE
```

```
      STORE LEN + 1 TO LEN
   ENDDO
   STORE LEN + MDEC + 1 TO LEN
   IF LEN > MLEN
      STORE '*** ERROR - THE RESULT OF THE ARITHMETIC '+;
            'EXPRESSION IS OVERFLOW' TO ERROR:MSG
      STORE T TO ERROR
      RETURN
   ELSE
      REPLACE &OATTR WITH RESULT
   ENDIF
   SKIP
ENDDO
RELEASE TEMP, RESULT, BASE, LEN, EXP
USE B:AVTABLE
GOTO I
DELETE
PACK
STORE I - 1 TO I
STORE MAX:AV - 1 TO MAX:AV
RETURN


*************************************************************
*                          RETRIE1.PRG                      *
* This procedure, which is called by procedure PARSE1.prg,  *
* retrieves the information from the data base and pushes the *
* information onto the SAVE stack.                          *
*************************************************************

USE B:CODESTACK
GOTO BOTTOM
STORE # TO MAX:CODE
USE
STORE 'TEMP1' TO FILE1
STORE 'TEMP2' TO FILE2
IF MAX:CODE > 0
   DO RETRIE2
   IF ERROR
      RETURN
   ENDIF
ENDIF
IF JOIN
   SELECT PRIMARY
   USE B:&FILE1
   SELECT SECONDARY
   USE B:&FILE2
   JOIN TO B:BUFFER1 FOR P.&FIELD = S.&FIELD
   SELECT PRIMARY
   USE
   SELECT SECONDARY
   USE
   USE B:BUFFER1
   COPY FIELD &OUT:FLDS TO B:SAVE
ELSE
```

```
   USE B:&FILE1
   COPY FIELD &OUT:FLDS TO B:SAVE
ENDIF
USE B:CODESTACK
GOTO BOTTOM
STORE # TO MAX:CODE
IF MAX:CODE > 0
   DO BUILTFUN
   IF ERROR
      RETURN
   ENDIF
ENDIF
RELEASE MAX:CODE,FILE1,FILE2
RETURN


*************************************************************************
*                              RETRIE2.PRG                              *
* This  procedure, which is called by procedure RETRIE1.prg,            *
* retrieves information from the data base and pushes the               *
* information onto the SAVE stack.                                      *
*************************************************************************

DO WHILE MAX:CODE > 0
   USE B:CODESTACK
   GOTO MAX:CODE
   STORE TRIM(ATTR) TO MATTR
   STORE TRIM(OPERATOR) TO MOPERATOR
   STORE TRIM(VALUE) TO MVALUE
   STORE TRIM(BUILTFUN) TO MBUILTFUN
   STORE TABLEID TO MTABLEID
   STORE UNION TO MUNION
   STORE 'TEMP' + $(MTABLEID,1,1) TO TEMP
   STORE 'BUFFER' + $(MTABLEID,1,1) TO BUFFER
   STORE 'SAVE' + $(MTABLEID,1,1) TO SAVE
   IF $(MVALUE,1,1) = ' '
      STORE MVALUE TO UNPURGED
      DO PURGEBLANK
      STORE PURGED TO MVALUE
   ENDIF
   IF MVALUE = ' '
      IF MOPERATOR <> ' '
         STORE 'SAVE1' TO FILE1
         STORE 'SAVE2' TO FILE2
         IF FILE('B:&BUFFER')
            USE B:&BUFFER
            GOTO BOTTOM
            IF # = 1
               STORE &MATTR TO VALUE
               COPY STRUCTURE EXTENDED TO B:STRU
               USE B:STRU
               LOCATE FOR FIELD:NAME = '&MATTR'
               STORE FIELD:TYPE TO FLD:TYPE
               STORE FIELD:LEN  TO FLD:LEN
               STORE FIELD:DEC  TO FLD:DEC
```

```
            IF FLD:TYPE = 'C'
               STORE "***ERROR - COLUMN &mattr IS NOT A "+;
                     "NUMERICAL TYPE CANNOT DO COMPARISION";
                     TO ERROR:MSG
               STORE T TO ERROR
               RETURN
            ELSE
               STORE STR(VALUE,FLD:LEN,FLD:DEC) TO MVAL
               STORE '&MATTR' + ' ' + '&MOPERATOR' + ' ' + ;
                     '&MVAL' TO CONDITION
            ENDIF
            RELEASE FLD:TYPE, FLD:LEN,FLD:DEC,MVAL,VALUE
         ELSE
            STORE '*** ERROR - TOO MANY VALUES OR NO VALUE'+;
                  ' FOR COLUMN &mattr' TO ERROR:MSG
            STORE T TO ERROR
            RETURN
         ENDIF
      ELSE
         STORE '*** ERROR - NO VALUE FOR COLUMN &mattr' ;
               TO ERROR:MSG
         STORE T TO ERROR
         RETURN
      ENDIF
      SET EXACT ON
      IF .NOT. MUNION
         USE B:&SAVE
         GOTO BOTTOM
         IF # > 1
            COPY TO B:&BUFFER FOR &CONDITION
            USE B:&BUFFER
            COPY TO B:&SAVE
         ELSE
            USE B:&TEMP
            COPY TO B:&BUFFER FOR &CONDITION
            USE B:&BUFFER
            COPY TO B:&SAVE
         ENDIF
      ELSE
         USE B:&TEMP
         COPY TO B:&BUFFER FOR &CONDITION
      ENDIF
      SET EXACT OFF
   ENDIF
ELSE
   STORE 'SAVE1' TO FILE1
   STORE 'SAVE2' TO FILE2
   IF MOPERATOR = ' '
      STORE '=' TO MOPERATOR
   ENDIF
   USE B:&TEMP
   IF TYPE (&MATTR) = 'C'
      STORE '&MATTR' + ' ' + '&MOPERATOR' + ' ' + "'&MVALUE'";
            TO CONDITION
   ELSE
```

```
      STORE '&MATTR' + ' ' + '&MOPERATOR' + ' ' + '&MVALUE';
            TO CONDITION
   ENDIF
   SET EXACT ON
   IF .NOT. MUNION
      IF FILE('B:&SAVE')
         USE B:&SAVE
         COPY TO B:&BUFFER FOR &CONDITION
         USE B:&BUFFER
         COPY TO B:&SAVE
      ELSE
         USE B:&TEMP
         COPY TO B:&BUFFER FOR &CONDITION
         USE B:&BUFFER
         COPY TO B:&SAVE
      ENDIF
   ELSE
      USE B:&TEMP
      COPY TO B:&BUFFER FOR &CONDITION
   ENDIF
   SET EXACT OFF
ENDIF
IF MVALUE <> ' ' .OR. MOPERATOR <> ' '
   IF MUNION
      IF .NOT. FILE('B:&SAVE')
         USE B:&BUFFER
         COPY TO B:&SAVE
      ELSE
         USE B:&BUFFER
         COPY STRUCTURE EXTENDED TO B:STRU
         USE B:STRU
         STORE ' ' TO CONDITION,MREPLACE
         DO WHILE .NOT. EOF
            STORE TRIM(FIELD:NAME) TO FLD:NAME
            STORE CONDITION + 'P.&FLD:NAME = S.&FLD:NAME'+;
                  ' .AND.' TO CONDITION
            STORE MREPLACE + 'S.&FLD:NAME WITH P.&FLD:NAME,';
                  TO MREPLACE
            SKIP
         ENDDO
         STORE $(CONDITION,2,LEN(CONDITION) - 7) TO CONDITION
         STORE $(MREPLACE,2,LEN(MREPLACE) - 2) TO MREPLACE
         SELECT PRIMARY
         USE B:&BUFFER
         DO WHILE .NOT. EOF
            SELECT SECONDARY
            USE B:&SAVE
            GOTO BOTTOM
            STORE # TO MAX:REC
            STORE 0 TO J
            STORE F TO FOUND
            DO WHILE  J < MAX:REC .AND. .NOT. FOUND
               STORE J + 1 TO J
               GOTO J
               IF &CONDITION
```

```
                        STORE T TO FOUND
                        LOOP
                     ENDIF
                  ENDDO
                  IF .NOT. FOUND
                     APPEND BLANK
                     REPLACE &MREPLACE
                  ENDIF
                  SELECT PRIMARY
                  SKIP
               ENDDO
               SELECT PRIMARY
               USE
               SELECT SECONDARY
               USE
               RELEASE MREPLACE,FLD:NAME,FOUND,J,MAX:REC
            ENDIF
         ENDIF
      ENDIF
      IF MBUILTFUN = ' '
         USE B:CODESTACK
         GOTO MAX:CODE
         DELETE
         PACK
      ENDIF
      STORE MAX:CODE - 1 TO MAX:CODE
ENDDO
IF JOIN
      IF .NOT. FILE('B:SAVE1')
          USE B:TEMP1
          COPY TO B:SAVE1
      ENDIF
      IF .NOT. FILE ('B:SAVE2')
         USE B:TEMP2
         COPY TO B:SAVE2
      ENDIF
ENDIF
RETURN


*****************************************************************
*                              RUN.PRG                          *
* This procedure, which is called by preocedure QBE.prg,        *
* processes the user's query, and, after finishing, continues   *
* the interactions with the user who decides at this point      *
* whether to continue with another query or not.                *
*****************************************************************

IF FILE:ONLY
    DO PROCFILE
ELSE
    RELEASE FILE:ONLY
    DO PARSE1
    IF ERROR .OR. NO:REC
       @ 23,1
```

```
        @ 23,1 SAY ERROR:MSG
        ACCEPT 'HIT <RETURN> TO CONTINUE ' TO ACTION
        RELEASE ACTION
        ERASE
        STORE F TO OK
        DO WHILE .NOT. OK
           STORE ' ' TO ANS
           @ 23,1 SAY 'DO YOU WANT TO SAVE YOUR QUERY? (Y/N)' ;
                  GET ANS PICTURE '!'
           READ
           IF ANS <> 'Y' .AND. ANS <> 'N'
              LOOP
           ELSE
              STORE T TO OK
           ENDIF
        ENDDO
        IF ANS = 'Y'
           DO SAVEQ
        ENDIF
        ERASE
     ENDIF (* error *)
ENDIF (* file:only *)
CLEAR
STORE F TO OK
DO WHILE  .NOT. OK
   STORE ' ' TO ANS
   @ 23,1 SAY "DO YOU WANT TO (Q)UIT OR (C)ONTINUE ===>";
   GET ANS PICTURE '!'
   READ
   IF ANS <> 'Q' .AND. ANS <> 'C'
      LOOP
   ELSE
      STORE T TO OK
   ENDIF
ENDDO
DO CLEARFILE
ERASE
DO CASE
    CASE ANS = 'Q'
         @ 10,30 SAY 'G O O D B Y E !'
         @ 22,0
         QUIT
    CASE ANS = 'C'
         CLEAR
         STORE T TO MORE
         RETURN
ENDCASE
```

```
*************************************************************************
*                              SAVEQ.PRG                                *
* This procedure, which is called by precodure RUN.prg, saves           *
* the user's query having error messages in a file for later            *
* review.                                                               *
*************************************************************************

IF .NOT. FILE('B:REVIEW')
   USE REVIEW
   COPY TO B:REVIEW
ENDIF
STORE T TO FIRST
STORE 0 TO I
DO WHILE I < FILE:NUM
   STORE I + 1 TO I
   STORE 'SFILE' + STR(I,1) TO SFILE
   SELECT PRIMARY
   USE B:&SFILE
   DO WHILE .NOT. EOF
      SELECT SECONDARY
      USE B:REVIEW
      APPEND BLANK
      REPLACE S.FLD:NAME WITH P.FLD:NAME ,S.ROW1 WITH P.ROW1;
              S.ROW2 WITH P.ROW2, S.ROW3 WITH P.ROW3
      IF I = 1 .AND. FIRST
         REPLACE MESSAGES WITH ERROR:MSG
         STORE F TO FIRST
      ENDIF
      SELECT PRIMARY
      SKIP
   ENDDO (* not eof *)
ENDDO (* i < file:num *)
IF R:EXIST
   SELECT SECONDARY
   APPEND BLANK
   REPLACE FLD:NAME WITH 'RESULT'
   SELECT PRIMARY
   USE B:RESULT
   DO WHILE .NOT. EOF
      SELECT SECONDARY
      USE B:REVIEW
      APPEND BLANK
      REPLACE S.FLD:NAME WITH P.FLD:NAME, S.ROW1 WITH P.ROW1;
              S.ROW2 WITH P.ROW2, S.ROW3 WITH P.ROW3
      SELECT PRIMARY
      SKIP
   ENDDO  (* not eof *)
ENDIF (* r:exist *)
SELECT PRIMARY
USE
SELECT SECONDARY
USE
IF C:EXIST
   STORE T TO FIRST
   USE B:CONDITION
```

```
   GOTO BOTTOM
   STORE # TO MAX:REC
   STORE 0 TO I
   DO WHILE I < MAX:REC
      STORE I + 1 TO I
      USE B:CONDITION
      GOTO I
      STORE CONDITION TO MCONDITION
      USE B:REVIEW
      APPEND BLANK
      IF I = 1 .AND. FIRST
         REPLACE FLD:NAME WITH 'CONDITION'
         STORE F TO FIRST
      ENDIF
      STORE 0 TO J
      DO WHILE J < 3
         USE B:REVIEW
         GOTO BOTTOM
         STORE J + 1 TO J
         STORE 'ROW' + STR(J,1) TO ROW
         REPLACE &ROW WITH MCONDITION
         IF J = 3 .OR. I = MAX:REC
            STORE 3 TO J
            LOOP
         ELSE
            STORE I + 1 TO I
            USE B:CONDITION
            GOTO I
            STORE CONDITION TO MCONDITION
         ENDIF
      ENDDO (* while j < 3 *)
   ENDDO (* i < max:rec *)
ENDIF (* c:exist *)
USE B:REVIEW
GOTO BOTTOM
REPLACE MESSAGES WITH 'END OF QUERY'
SELECT PRIMARY
USE
SELECT SECONDARY
USE
RETURN


*********************************************************************
*                              SCANAV1.PRG                          *
* This procedure, which is called by procedure PARSE1.prg,scans*
* the avtable and checks if there are any value with arithmetic*
* operators. If so, procedure CALCULATE will be invoked. If     *
* not, then return to the calling procedure                     *
*********************************************************************

STORE 0 TO I
DO WHILE I < MAX:AV
   STORE I + 1 TO I
   USE B:AVTABLE
```

```
    GOTO I
    STORE @('+', VALUE) TO PLUS
    STORE @('-', VALUE) TO MINUS
    STORE @('/', VALUE) TO DIVIDE
    STORE @('*', VALUE) TO MULTIPLY
    IF PLUS <> 0 .OR. MINUS <> 0 .OR. DIVIDE <> 0 .OR. ;
       MULTIPLY <> 0
       IF TYPE = 'N'
          STORE TRIM(ATTR) TO MATTR,OATTR
          STORE TABLEID TO MTABLEID
          STORE VALUE TO EXPRESSION
          STORE DEC TO MDEC
          STORE LEN TO MLEN
          STORE F TO NO:VALUE
          DO CALCULATE
          IF ERROR
             RETURN
          ENDIF
          RELEASE MATTR,MTABLEID,EXPRESSION,MDEC,MLEN,NO:VALUE;
                  OATTR
       ELSE
          STORE "*** ERROR - " + TRIM(ATTR) + " IS NOT A "+;
                "NUMERICAL TYPE, CANNOT DO CALCULATION" ;
                TO ERROR:MSG
          STORE T TO ERROR
          RETURN
       ENDIF
    ENDIF
ENDDO
RELEASE PLUS,MINUS,DIVIDE,MULTIPLY,I



*****************************************************************
*                            SCANAV2.PRG                        *
* This procedure, which is called by procedure PARSE1.prg,scans*
* the avtable and checks if any value contains the logical      *
* operators 'AND' or 'OR'.                                      *
*****************************************************************

STORE 0 TO I
DO WHILE I < MAX:AV
   STORE I + 1 TO I
   USE B:AVTABLE
   GOTO I
   IF $(VALUE,1,1) = '('
      STORE @(' AND ', VALUE) TO INVALID1
      STORE @(' OR ', VALUE) TO INVALID2
      IF INVALID1 <> 0 .OR. INVALID2 <> 0
         STORE '*** ERROR - INVALID LOGICAL OPERATOR IN '+;
               TRIM(VALUE) TO ERROR:MSG
         STORE T TO ERROR
         RETURN
      ENDIF
      STORE $(VALUE,2) TO MVALUE
```

```
STORE TRIM(MVALUE) TO MVALUE
STORE ATTR TO MATTR
STORE TABLEID TO MTABLEID
STORE F TO END
STORE T TO START
DO WHILE .NOT. END
   STORE @('|', MVALUE) TO LOGICAL
   IF LOGICAL = 0
      STORE @('&',MVALUE) TO LOGICAL
      IF LOGICAL = 0
         IF START
            STORE '*** ERROR - INVALID CONDITION '+;
                  'EXPRESSION' TO ERROR:MSG
            STORE T TO ERROR
            RETURN
         ENDIF
         STORE @(')', MVALUE) TO LOGICAL
         STORE T TO END
         IF LOGICAL = 0
            STORE  '*** ERROR - MISSING LOGICAL OPERATOR'+;
                   ' OR  USING INVALID LOGICAL OPERATOR'+;
                   ' IN CONDITION EXPRESSION' TO ERROR:MSG
            STORE T TO ERROR
            RETURN
         ENDIF
      ENDIF
   ENDIF
   STORE $(MVALUE,1,LOGICAL - 1) TO EXP
   STORE TRIM(EXP) TO EXP
   STORE $(MVALUE,LOGICAL,1) TO LOGICAL:OP
   IF .NOT. END
      STORE $(MVALUE,LOGICAL + 1) TO MVALUE
   ENDIF
   IF $(EXP,1,1) = ' '
      STORE EXP TO UNPURGED
      DO PURGEBLANK
      STORE PURGED TO EXP
   ENDIF
   IF $(EXP,1,2) = '>=' .OR. $(EXP,1,2) = '<=' .OR. ;
      $(EXP,1,2) = '<>'
      STORE $(EXP,1,2) TO MOPERATOR
      STORE $(EXP,3) TO EXP
   ELSE
      IF $(EXP,1,1) = '>' .OR. $(EXP,1,1) = '<' .OR. ;
         $(EXP,1,1) = '='
         STORE $(EXP,1,1) TO MOPERATOR
         STORE $(EXP,2) TO EXP
      ELSE
         STORE ' ' TO MOPERATOR
      ENDIF
   ENDIF
   IF $(EXP,1,1) = ' '
      STORE EXP TO UNPURGED
      DO PURGEBLANK
      STORE PURGED TO EXP
```

```
            ENDIF
            IF $(EXP,1,1)$'><='
              STORE  '*** ERROR - INVALID CONDITION EXPRESSION ';
                     TO ERROR:MSG
               STORE T TO ERROR
               RETURN
            ENDIF
            IF LOGICAL:OP = '|'
               STORE T TO MUNION
            ELSE
               STORE F TO MUNION
            ENDIF
            USE B:CODESTACK
            APPEND BLANK
            REPLACE ATTR WITH MATTR, VALUE WITH EXP;
                    UNION WITH MUNION, TABLEID WITH MTABLEID
            IF MOPERATOR <> '='
               REPLACE OPERATOR WITH MOPERATOR
            ENDIF
            IF .NOT. END
               IF $(MVALUE,1,1) = ' '
                  STORE MVALUE TO UNPURGED
                  DO PURGEBLANK
                  STORE PURGED TO MVALUE
               ENDIF
            ENDIF
            STORE F TO START
         ENDDO
         RELEASE MATTR, EXP, MUNION, MTABLEID, MOPERATOR, MVALUE
         RELEASE LOGICAL:OP, INVALID1, INVALID2, START,END, LOGICAL
         USE B:AVTABLE
         GOTO I
         DELETE
         PACK
         STORE MAX:AV - 1 TO MAX:AV
         STORE I - 1 TO I
      ENDIF
   ENDDO
   RELEASE I
   RETURN


   ********************************************************************
   *                              SCANAV3.PRG                         *
   * This procedure, which is called by procedure PARSE1.prg,scans*
   * the avtable and stores all the remaining information onto the*
   * CODE stack.                                                      *
   ********************************************************************

   STORE 0 TO I
   DO WHILE I < MAX:AV
      STORE F TO OP:EXIST
      STORE I + 1 TO I
      USE B:AVTABLE
      GOTO I
```

```
STORE VALUE TO MVARIABLE
STORE ATTR TO OMATTR,MATTR
STORE OPERATOR TO MOPERATOR
STORE BUILTFUN TO MBUILTFUN
STORE TABLEID TO MTABLEID
IF $(MVARIABLE,1,1) = '_'
   STORE F TO FOUND
   IF MOPERATOR <> ' '
      STORE  T TO OP:EXIST
      USE B:CODESTACK
      APPEND BLANK
      REPLACE ATTR WITH MATTR, OPERATOR WITH MOPERATOR;
              TABLEID WITH MTABLEID
   ENDIF
   USE
   STORE I TO J
   DO WHILE J < MAX:AV
      STORE J + 1 TO J
      USE B:AVTABLE
      GOTO J
      IF VALUE = MVARIABLE
         IF ATTR = OMATTR
            STORE T TO FOUND
            STORE OPERATOR TO MOPERATOR
            IF TABLEID <> MTABLEID
               STORE T TO JOIN
               STORE TRIM(ATTR) TO FIELD
            ELSE
              USE B:PRNSTACK INDEX B:PRNDX
              STORE TRIM(OMATTR) TO OMATTR
              FIND '&OMATTR'
              IF # <> 0
                 STORE UNION TO MUNION
              ELSE
                 STORE F TO MUNION
              ENDIF
              IF MOPERATOR <> ' '
                 STORE T TO OP:EXIST
                 USE B:CODESTACK
                 APPEND BLANK
                 REPLACE ATTR WITH MATTR;
                 OPERATOR WITH MOPERATOR,TABLEID WITH MTABLEID
              ENDIF
            ENDIF
            USE
            USE B:AVTABLE
            GOTO J
            DELETE
            PACK
            STORE MAX:AV - 1 TO MAX:AV
            STORE J - 1 TO PREVIOUS
            GOTO PREVIOUS
            IF VALUE  <> ' ' .AND. $(VALUE,1,1) <> '_'
               IF TABLEID = MTABLEID
                 STORE ATTR TO MATTR
```

```
                  STORE OPERATOR TO MOPERATOR
                  STORE BUILTFUN TO MBUILTFUN
                  STORE VALUE TO MVALUE
                  DELETE
                  PACK
                  STORE MAX:AV - 1 TO MAX:AV
                  SE B:CODESTACK
                  APPEND BLANK
                  REPLACE ATTR WITH MATTR;
                  OPERATOR WITH MOPERATOR, UNION WITH MUNION;
                  BUILTFUN WITH MBUILTFUN, VALUE WITH MVALUE;
                  REPLACE TABLEID WITH MTABLEID
               ENDIF
            ENDIF
            IF .NOT. FOUND .AND. J < MAX:AV
               GOTO J
               IF VALUE  <> ' ' .AND. $(VALUE,1,1) <> '_'
                  IF TABLEID = MTABLEID
                     STORE ATTR TO MATTR
                     STORE VALUE TO MVALUE
                     STORE OPERATOR TO MOPERATOR
                     STORE BUILTFUN TO MBUILTFUN
                     DELETE
                     PACK
                     STORE MAX:AV - 1 TO MAX:AV
                     USE B:CODESTACK
                     APPEND BLANK
                     REPLACE ATTR WITH MATTR;
                     OPERATOR WITH MOPERATOR,UNION WITH MUNION
                     REPLACE BUILTFUN WITH MBUILTFUN;
                     VALUE WITH MVALUE, TABLEID WITH MTABLEID
                  ENDIF
               ENDIF
            ENDIF
            IF OP:EXIST .AND. .NOT. FOUND
               STORE TRIM(MVARIABLE) TO MVARIABLE
               STORE '*** ERROR - EXAMPLE ELEMENT &mvariable'+;
                     ' DOES NOT HAVE A CONSTANT VALUE TO '+;
                     ' MATCH IT' TO ERROR:MSG
               STORE T TO ERROR
               RETURN
            ENDIF
            STORE J - 1 TO J
         ELSE
            STORE TRIM(MVARIABLE) TO MVARIABLE
            STORE '*** ERROR - EXAMPLE ELEMENT &mvariable'+;
                  ' HAS CONFLICTING COLUMN NAMES' TO ERROR:MSG
            STORE T TO ERROR
            RETURN
         ENDIF
      ENDIF
   ENDDO
   USE
   USE B:AVTABLE
   GOTO I
```

```
IF .NOT. FOUND
   STORE TRIM(MVARIABLE) TO MVARIABLE
   STORE TRIM(ATTR) TO MATTR
   USE B:PRNSTACK INDEX B:PRNDX
   FIND '&MATTR'
   IF # = 0
      STORE '*** ERROR - EXAMPLE ELEMENT &mvariable'+;
            ' UNDEFINED' TO ERROR:MSG
      STORE T TO ERROR
      RETURN
   ELSE
      IF VARIABLE <> ' '
         IF TRIM(VARIABLE) <> TRIM(MVARIABLE)
            STORE '*** ERROR - EXAMPLE ELEMENT &mattr HAS'+;
                  ' CONFILCTING COLUMN NAMES' TO ERROR:MSG
            STORE T TO ERROR
            RETURN
         ENDIF
      ENDIF
      STORE UNION TO MUNION
      IF MUNION
         IF MBUILTFUN <> ' '
            USE B:CODESTACK
            APPEND BLANK
            REPLACE ATTR WITH MATTR, UNION WITH MUNION;
            BUILTFUN WITH MBUILTFUN,TABLEID WITH MTABLEID
         ELSE
            IF I > 1
               USE B:AVTABLE
               STORE I - 1 TO PREVIOUS
               GOTO PREVIOUS
               IF TABLEID = MTABLEID
                  IF VALUE <> ' '.AND. $(VALUE,1,1) <> '_'
                     STORE ATTR TO MATTR
                     STORE VALUE TO MVALUE
                     STORE OPERATOR TO MOPERATOR
                     STORE BUILTFUN TO MBUILTFUN
                     DELETE
                     PACK
                     STORE MAX:AV - 1 TO MAX:AV
                     USE B:CODESTACK
                     APPEND BLANK
                     REPLACE ATTR WITH MATTR;
                     VALUE WITH MVALUE, UNION WITH MUNION;
                     OPERATOR WITH MOPERATOR
                     REPLACE BUILTFUN WITH MBUILTFUN;
                             TABLEID WITH MTABLEID
                  ENDIF
               ENDIF
            ELSE
               IF I < MAX:AV
                  USE B:AVTABLE
                  STORE I + 1 TO NEXT
                  GOTO NEXT
                  IF TABLEID = MTABLEID
```

```
                              IF VALUE <> ' ' .AND. ;
                                 $(VALUE,1,1) <> '_'
                                 STORE ATTR TO MATTR
                                 STORE VALUE TO MVALUE
                                 STORE OPERATOR TO MOPERATOR
                                 STORE BUILTFUN TO MBUILTFUN
                                 DELETE
                                 PACK
                                 STORE MAX:AV - 1 TO MAX:AV
                                 USE B:CODESTACK
                                 APPEND BLANK
                                 REPLACE ATTR WITH MATTR;
                                 VALUE WITH MVALUE,UNION WITH MUNION
                                 REPLACE OPERATOR WITH MOPERATOR;
                                         BUILTFUN WITH MBUILTFUN;
                                         TABLEID WITH MTABLEID
                              ENDIF
                           ENDIF
                        ENDIF
                     ENDIF
                  ENDIF
               ELSE
                  IF MBUILTFUN <> ' '
                     USE B:CODESTACK
                     APPEND BLANK
                     REPLACE ATTR WITH MATTR, TABLEID WITH MTABLEID;
                             BUILTFUN WITH MBUILTFUN
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
         USE
         USE B:AVTABLE
         GOTO I
         DELETE
         PACK
         STORE MAX:AV -1 TO MAX:AV
         STORE I - 1 TO I
      ENDIF
   ENDDO
   USE
   RELEASE MVARIABLE,MATTR, MOPERATOR,MBUILTFUN,MUNION,MTABLEID
   RELEASE  MVALUE,I,J,FOUND,OP:EXIST, OMATTR
   IF MAX:AV > 0
      SELECT PRIMARY
      USE B:AVTABLE
      DO WHILE .NOT. EOF
         SELECT SECONDARY
         USE B:CODESTACK
         APPEND BLANK
         REPLACE S.ATTR WITH P.ATTR, S.VALUE WITH P.VALUE;
         S.BUILTFUN WITH P.BUILTFUN,  S.OPERATOR WITH P.OPERATOR;
         S.TABLEID WITH P.TABLEID
         SELECT PRIMARY
         DELETE
```

```
      PACK
   ENDDO
ENDIF
SELECT PRIMARY
USE
SELECT SECONDARY
USE
RETURN



*********************************************************************
*                          SCANCOND.PRG                             *
* This procedure, which is called by procedure PARSE1.prg,scans*
* the condition box, and  places each codition statement in the*
* corresponding row of value column of avtable.                     *
*********************************************************************

SELECT PRIMARY
USE B:CONDITION
GOTO BOTTOM
STORE # TO MAX:REC
STORE 0 TO I
DO WHILE I < MAX:REC
   STORE I + 1 TO I
   GOTO I
   STORE CONDITION TO MCONDITION
   IF $(MCONDITION,1,1) = '_'
      IF $(MCONDITION,2,1) <> ' '
         STORE '_'  TO MVARIABLE
         STORE $(MCONDITION,2) TO MCONDITION
         DO WHILE ( $(MCONDITION,1,1)$'0123456789' .OR.;
               $(MCONDITION,1,1)$'ABCDEFGHIJKLMNOPQRSTUVWXYZ') ;
               .AND. LEN(MCONDITION) > 1
            STORE MVARIABLE + $(MCONDITION,1,1) TO MVARIABLE
            STORE $(MCONDITION,2) TO MCONDITION
         ENDDO
         IF LEN(MCONDITION) > 1
            IF $(MCONDITION,1,1) = ' '
               STORE MCONDITION TO UNPURGED
               DO PURGEBLANK
               STORE PURGED TO MCONDITION
            ENDIF
            IF $(MCONDITION,1,1)$'<>='
               STORE $(MCONDITION,1,1) TO MOPERATOR
               IF $(MCONDITION,2,1)$'=>'
                  STORE MOPERATOR + $(MCONDITION,2,1);
                        TO MOPERATOR
                  STORE $(MCONDITION,3) TO MCONDITION
               ELSE
                  IF MOPERATOR = '='
                     STORE ' ' TO MOPERATOR
                  ENDIF
                  STORE $(MCONDITION,2) TO MCONDITION
```

```
            ENDIF
            IF $(MCONDITION,1,1) = ' '
               STORE TRIM(MCONDITION) TO UNPURGED
               DO PURGEBLANK
               STORE PURGED TO MCONDITION
            ENDIF
            IF LEN(MCONDITION) > 1
               STORE MCONDITION TO MVALUE
            ELSE
               IF MCONDITION <> ' '
                  STORE MCONDITION TO MVALUE
               ELSE
                  STORE '***ERROR - INVALID CONDITION '+;
                        'EXPRESSION' TO ERROR:MSG
                  STORE T TO ERROR
                  RETURN
               ENDIF
            ENDIF
         ELSE
            STORE '*** ERROR - INVALID CONDITION EXPRESSION';
                  TO ERROR:MSG
            STORE T TO ERROR
            RETURN
         ENDIF
      ELSE
         STORE '*** ERROR - INVALID CONDITION EXPRESSION';
               TO ERROR:MSG
         STORE T TO ERROR
         RETURN
      ENDIF
   ELSE
      STORE '*** ERROR - NO BLANK CAN BE EMBEDDED INTO'+;
            ' EXAMPLE ELEMENT' TO ERROR:MSG
      STORE T TO ERROR
      RETURN
   ENDIF
ELSE
  STORE '*** ERROR - INVALID CONDITION EXPRESSION' TO ERROR:MSG
  STORE T TO ERROR
  RETURN
ENDIF
SELECT SECONDARY
USE B:AVTABLE
STORE F TO FOUND
DO WHILE .NOT. EOF .AND. .NOT. FOUND
   IF S.VALUE = MVARIABLE
      REPLACE S.VALUE WITH MVALUE, S.OPERATOR WITH MOPERATOR
      STORE T TO FOUND
   ENDIF
   SKIP
ENDDO
IF .NOT. FOUND
    STORE  '*** ERROR - EXAMPLE ELEMENT &mvariable IS NOT'+;
           ' MATCHED' TO ERROR:MSG
    STORE T TO ERROR
```

```
      RETURN
   ENDIF
   SELECT PRIMARY
ENDDO
SELECT PRIMARY
USE
SELECT SECONDARY
USE
RELEASE MVALUE,MVARIABLE,MOPERATOR,MAX:REC,I, MCONDITION,FOUND
RETURN


*********************************************************************
*                              SCANFN.PRG                           *
* This procedure,which is called by procedure DTABLE.prg,parses*
* the file name and command which are entered by the user and  *
* checks if they are valid or not. If invalid, the program will*
* display an error message and ask the user to re-enter.       *
*********************************************************************

@ 23,1 SAY 'CHECKING DATA ENTRY, PLEASE WAIT A SECOND!'
STORE T TO FILE:ONLY
STORE T TO OK
STORE TRIM(FILE:NAME) TO FILE:NAME
STORE FILE:NAME TO UNPURGED
DO PURGEBLANK
STORE  PURGED TO FILE:NAME
IF FILE:NAME = ' '
   STORE "*** ERROR - YOU DIDN'T ENTER ANYTHING" TO ERROR:MSG
   STORE F TO OK
   RETURN
ENDIF (* file:name = ' ' *)

STORE T TO CHECKING
STORE T TO FILE:ONLY
STORE 0 TO HOW:MANY
STORE ' ' TO COMMAND
DO WHILE CHECKING
   STORE @('.', FILE:NAME) TO FOUND:AT
   STORE LEN(FILE:NAME) TO STR:LEN
   IF FOUND:AT <> 0
      STORE HOW:MANY + 1 TO HOW:MANY
      IF FOUND:AT = 2
         IF HOW:MANY = 2
            STORE COMMAND + ',' + $(FILE:NAME, 1,1) TO COMMAND
            STORE ' ' TO FILE:NAME
         ELSE
            STORE COMMAND + $(FILE:NAME, FOUND:AT - 1 ,1);
                  TO COMMAND
            IF STR:LEN > 2
               STORE $(FILE:NAME,3,STR:LEN - 2) TO FILE:NAME
            ELSE
               STORE ' ' TO FILE:NAME
            ENDIF (* str:len > 2 *)
         ENDIF (* how:many = 2 *)
```

```
      ELSE
         IF FOUND:AT = STR:LEN
            STORE COMMAND + ',' + $(FILE:NAME,FOUND:AT - 1,1);
                  TO COMMAND
            STORE  TRIM ($(FILE:NAME,1,STR:LEN - 2)) TO FNAME
            STORE F TO CHECKING
            LOOP
         ELSE
            STORE '*** SYNTAX ERROR ***' TO ERROR:MSG
            STORE F TO OK
            RETURN
         ENDIF (* found:at = str:len *)
      ENDIF (* found:at = 2 *)
   ELSE
      STORE FILE:NAME TO FNAME
      STORE F TO CHECKING
      IF HOW:MANY = 0
         STORE F TO FILE:ONLY
      ENDIF (* how:many = 0 *)
  ENDIF  (*found:at <> 0 *)
ENDDO (* checking *)
RELEASE CHECKING, FOUND:AT, STR:LEN

IF HOW:MANY > 0
     STORE $(COMMAND,2, (LEN(COMMAND) - 1)) TO COMMAND
ENDIF (* how:many > 0 *)
RELEASE HOW:MANY
STORE FNAME TO UNPURGED
DO PURGEBLANK
STORE PURGED TO FNAME
RELEASE UNPURGED,PURGED

IF .NOT. FILE:ONLY
   *** check if file name existing in the user's disk ***
   STORE @(' ',FNAME) TO FOUND
   IF .NOT. FILE('B:&FNAME') .OR. FOUND <> 0
      STORE '*** ERROR - TABLE &fname DOES NOT EXIST' TO ERROR:MSG
      STORE F TO OK
   ENDIF
   RELEASE COMMAND, FOUND
   RETURN
ENDIF (* not file:only *)


SET EXACT ON
IF COMMAND <> 'P' .AND. COMMAND <> 'P,P' .AND. COMMAND <> 'I,I'
    IF COMMAND <> ',P'.AND. COMMAND <> 'U' .AND. COMMAND <> 'D'
         STORE '*** ERROR - INVALID SYSTEM OPERATOR' TO ERROR:MSG
         STORE F TO OK
         RETURN
    ELSE
         IF FNAME = ' '
            STORE '*** ERROR - MISSING TABLE NAME' TO ERROR:MSG
            STORE F TO OK
            RETURN
```

```
         ELSE
            STORE @(' ',FNAME) TO FOUND
            IF .NOT. FILE('B:&FNAME') .OR. FOUND <> 0
               STORE '*** ERROR - TABLE &FNAME DOES NOT EXIST';
                     TO ERROR:MSG
               STORE F TO OK
               RETURN
            ENDIF (* not file exist *)
            RELEASE FOUND
         ENDIF (* missing file name'
   ENDIF (* command <> ',P' and <> 'U' and `D' *)
ELSE
   IF COMMAND = 'I,I'
       DO CHKFILE
   ELSE
       IF $(FNAME,1,1) <> '_' .AND. FNAME <> ' '
         STORE  '*** ERROR - TABLE NAME SHOULD BE AN EXAMPLE'+;
                ' ELEMENT' TO ERROR:MSG
         STORE F TO OK
         RETURN
       ENDIF (* not a variable file name*)
   ENDIF (* command = 'I,I' *)
ENDIF (* command <> 'P'and <> 'I,I' and <> 'P,P' *)
SET EXACT OFF
RETURN



*********************************************************************
*                            SCANPRN.PRG                            *
* This procedure, which is called by procedure PARSE1.prg,scans     *
* the print stack and deletes duplicate field name and orders       *
* the remaining field names into a string.                          *
*********************************************************************

USE B:PRNSTACK
GOTO BOTTOM
STORE # TO MAX:REC
STORE 0 TO I
DO WHILE I < MAX:REC
   STORE I + 1 TO I
   GOTO I
   IF .NOT. *
       STORE TRIM(ATTR) TO MATTR
       LOCATE FOR ATTR = '&MATTR'
       STORE F TO MUNION
       STORE # TO REC:NO
       DO WHILE .NOT. EOF
          CONTINUE
          IF EOF
             STORE # TO LAST
             IF ATTR = '&MATTR'
                IF REC:NO <> LAST
                   STORE T TO MUNION
                   DELETE
```

```
                ENDIF
             ENDIF
          ELSE
             STORE T TO MUNION
             DELETE
          ENDIF
       ENDDO
       IF MUNION
          GOTO REC:NO
          REPLACE UNION WITH T
       ENDIF
   ENDIF
ENDDO
PACK
INDEX ON ATTR TO B:PRNDX
USE
STORE ' ' TO OUT:FLDS
USE B:PRNSTACK
DO WHILE .NOT. EOF
   STORE OUT:FLDS + TRIM(ATTR) + ',' TO OUT:FLDS
   SKIP
ENDDO
STORE $(OUT:FLDS,2,LEN(OUT:FLDS) - 2) TO OUT:FLDS
RELEASE I, MAX:REC, MATTR, LAST, REC:NO, MUNION


*************************************************************************
*                                SEARCH.PRG                             *
* This procedure, which is called by procedure MAKEOUT2.prg,            *
* checks if a field name that  appears in the result table also          *
* exists in the general tables. If not, an error message will be*
* displayed.                                                             *
*************************************************************************

STORE 0 TO I
STORE F TO FOUND
DO WHILE I < FILE:NUM
   STORE I + 1 TO I
   STORE 'SFILE' + STR(I,1) TO SFILE
   SELECT SECONDARY
   USE B:&SFILE
   INDEX ON FLD:NAME TO B:SINDEX
   SET INDEX TO B:SINDEX
   FIND &FIELD:NAME
   IF # <> 0
      STORE T TO FOUND
      STORE FILE:NUM TO I
      LOOP
   ENDIF
   SELECT SECONDARY
   USE
ENDDO
IF .NOT. FOUND
   STORE '***ERROR - COLUUMN ' + TRIM(FIELD:NAME) +;
         ' OF THE RESULT TABLE IS NOT FOUND IN OTHER TABLES';
```

```
          TO ERROR:MSG
   STORE T TO ERROR
ENDIF
RELEASE FOUND,FIELD:NAME,I


*********************************************************************
*                              STORE1.PRG                           *
* This procedure, which is called by procedure DTABLE.prg,stores*
* the user's query entered in the general table into SFILEs.    *
*********************************************************************

@ 23,10 SAY 'PLEASE WAIT A MINUTE!'
IF !(ANSWER) = 'G'
   STORE 'SFILE' + STR(FILE:NUM,1) TO SFILE
   USE SFILE
   COPY TO B:&SFILE
   USE B:&SFILE
   APPEND BLANK
   REPLACE FLD:NAME WITH '&FNAME'
ELSE
   IF !(ANSWER) = 'R'
      STORE TRIM(FNAME) TO FNAME
      STORE FNAME TO UNPURGED
      DO PURGEBLANK
      USE SFILE
      COPY TO B:RESULT
      USE B:RESULT
      APPEND BLANK
      REPLACE FLD:NAME WITH '&PURGED'
   ENDIF (* answer = 'R'*)
ENDIF (* answer = 'G'*)

** connect query **
STORE 0 TO ROW:COUNT
STORE 4 TO MAX:ROW
STORE 7 TO MAX:COL
DO WHILE ROW:COUNT < MAX:ROW
   STORE ROW:COUNT + 1 TO ROW:COUNT
   IF ROW:COUNT = 1
      STORE 1 TO COL:COUNT
   ELSE
      STORE 0 TO COL:COUNT
      GOTO TOP
   ENDIF
   DO WHILE COL:COUNT < MAX:COL
      STORE COL:COUNT + 1 TO COL:COUNT
      STORE 'ROW' + STR(ROW:COUNT,1) + ':COL' + STR(COL:COUNT,1);
            TO PARSER
      IF $(&PARSER,1,1) = ' '
         STORE &PARSER TO UNPURGED
         DO PURGEBLANK
         STORE PURGED TO &PARSER
      ENDIF
      IF ROW:COUNT = 1
```

```
            APPEND BLANK
            REPLACE FLD:NAME WITH &PARSER
        ELSE
            STORE ROW:COUNT -1 TO ROW:NUM
            STORE 'ROW' + STR(ROW:NUM,1) TO ROW:NO
            REPLACE &ROW:NO WITH &PARSER
            SKIP
        ENDIF
        RELEASE &PARSER
    ENDDO (* col:count < max:col *)
ENDDO
RELEASE PARSER, MAX:COL,COL:COUNT,ROW:COUNT,MAX:ROW,ROW:NO
RELEASE ROW:NUM
@ 23,10
RETURN


*********************************************************************
*                              STORE2.PRG                           *
* This procedure, which is called by procedure DCON.prg, stores*
* the user's condition statements into a  file.                     *
*********************************************************************

@ 23,10 SAY 'PLEASE WAIT A MINUTE!'
USE B:CONDITION
STORE 1 TO ROW:COUNT
STORE 3 TO MAX:ROW
DO WHILE ROW:COUNT <= MAX:ROW
    STORE 'ROW' + STR(ROW:COUNT,1) + ':COL1' TO PARSER
    IF $(&PARSER,1,1) = ' '
       STORE &PARSER TO UNPURGED
       DO PURGEBLANK
       STORE PURGED TO &PARSER
    ENDIF (* $(&parser,1,1) = ' ' *)
    IF &PARSER <> ' '
        APPEND BLANK
        REPLACE CONDITION WITH &PARSER
    ENDIF (* &parse <> ' ' *)
    RELEASE &PARSER
    STORE ROW:COUNT + 1 TO ROW:COUNT
ENDDO (* row:count <= max:row *)
RELEASE ROW:COUNT, MAX:ROW, PARSER
RETURN


*********************************************************************
*                             SUMORAVG.PRG                          *
* This procedure, which is called by procedure BUILTFUN.prg,    *
* calculates the total or average value for a numerical field. *
*********************************************************************

USE B:&FILE
GOTO BOTTOM
STORE # TO MAX:REC
```

```
IF MAX:REC = 0
   STORE '***ERROR - NO RECORD AVAILABLE, CANNOT PROCEED '+;
         '&mbuiltfun' TO ERROR:MSG
   STORE T TO ERROR
   RETURN
ELSE
   GOTO TOP
   SUM &MATTR TO RESULT
ENDIF
IF $(MBUILTFUN,1,4) = 'AVG.'
   STORE RESULT / MAX:REC TO RESULT
ENDIF
RELEASE MAX:REC
RETURN


*********************************************************************
*                              TABFORM.PRG                          *
* This procedure, which is called by procedure DISPANS.prg,         *
* displays the result of user's query in table form and prints      *
* it out.                                                           *
*********************************************************************

USE B:SAVE
COPY STRUCTURE EXTENDED TO B:STRU
USE B:STRU
GOTO BOTTOM
STORE # TO REC:NO
GOTO TOP
SUM FIELD:LEN TO TOTAL:LEN
STORE TOTAL:LEN + REC:NO TO TOTAL:LEN
IF TOTAL:LEN > 132
   ERASE
   @ 1,3 SAY "*** TOTAL LENGTH OF EACH RECORD EXCEEDS 132"
   @ 2,3 SAY "CHARACTERS, CANN'T PRINT IN TABLE FORM ***"
   STORE F TO OK
   DO WHILE .NOT. OK
      STORE ' ' TO ANS
      @ 4,3 SAY 'Do you want to (Q)uit or (C)hange to '+;
                'non-table form?' GET ANS PICTURE '!'
      READ
      IF ANS <> 'Q' .AND. ANS <> 'C'
         LOOP
      ELSE
         STORE T TO OK
      ENDIF
   ENDDO
   IF ANS = 'Q'
      RETURN
   ELSE
      USE
      DO NTABFORM
      RETURN
   ENDIF
ELSE
```

```
ERASE
SET FORMAT TO PRINT
SET PRINT ON
@ 1,1 SAY 'The answer to your query is shown as follows:'
STORE 2 TO COL
USE B:STRU
DO WHILE .NOT. EOF
   STORE FIELD:LEN / 2 TO FLD:LEN
   STORE INT(FLD:LEN) TO FLD:LEN
   IF FLD:LEN > 5
      STORE FLD:LEN - 5 TO BEGIN
      STORE COL + BEGIN TO COL
      @ 3,COL SAY FIELD:NAME
      STORE FIELD:LEN - BEGIN TO END
      STORE COL + END TO COL
   ELSE
       STORE FIELD:NAME TO FLD:NAME
       @ 3,COL SAY FLD:NAME
       STORE COL + FIELD:LEN TO COL
   ENDIF
   STORE COL + 1 TO COL
   SKIP
ENDDO
STORE 1 TO COL
USE B:STRU
DO WHILE .NOT. EOF
   STORE FIELD:LEN TO FLD:LEN
   @ 4,COL SAY '+'
   STORE 0 TO I
   DO WHILE I < FLD:LEN
      STORE I + 1 TO I
      STORE COL + 1 TO COL
      @ 4,COL SAY '-'
   ENDDO
   STORE COL + 1 TO COL
   SKIP
ENDDO
@ 4,COL SAY '+'
SELECT PRIMARY
USE B:SAVE
GOTO BOTTOM
STORE # TO MAX:REC
STORE 0 TO I
STORE 4 TO ROW
DO WHILE I < MAX:REC
   STORE I + 1 TO I
   STORE ROW + 1 TO ROW
   GOTO I
   @ ROW,1 SAY '|'
   STORE 2 TO COL
   SELECT SECONDARY
   USE B:STRU
   DO WHILE .NOT. EOF
      STORE FIELD:NAME TO FLD:NAME
      @ ROW, COL SAY P.&FLD:NAME
```

```
            STORE COL + S.FIELD:LEN TO COL
            @ ROW,COL SAY '|'
            STORE COL + 1 TO COL
            SKIP
         ENDDO
         SELECT PRIMARY
         SKIP
      ENDDO
      SELECT PRIMARY
      USE
      SELECT SECONDARY
      USE
      SET PRINT OFF
      SET FORMAT TO SCREEN
      @ 22,1 SAY 'YOUR QUERY HAS BEEN DONE'
   ENDIF
   RETURN


*********************************************************************
*                                UNIQUE.PRG                         *
* This procedure, which is called by procedure BUILTFUN.prg,        *
* deletes the duplicate records of a file.                          *
*********************************************************************

USE B:SAVE
COPY TO B:SAVE1
USE B:SAVE1
GOTO BOTTOM
STORE # TO MAX:REC
STORE 0 TO I
DO WHILE I < MAX:REC
   STORE I + 1 TO I
   GOTO I
   IF TYPE(&MATTR) = 'C'
      STORE TRIM(&MATTR) TO CONTENT
      LOCATE FOR &MATTR = "&CONTENT"
   ELSE
      STORE &MATTR TO CONTENT
      LOCATE FOR &MATTR = CONTENT
   ENDIF
   STORE # TO REC:NO
   DO WHILE .NOT. EOF
      CONTINUE
      IF EOF
         STORE # TO LAST
         STORE F TO MDELETE
         IF TYPE(&MATTR) = 'C'
            IF &MATTR = "&CONTENT'"
               STORE  T TO MDELETE
            ENDIF
         ELSE
            IF &MATTR = CONTENT
               STORE T TO MDELETE
            ENDIF
```

```
         ENDIF
         IF MDELETE
            IF REC:NO <> LAST
               DELETE
               PACK
               STORE MAX:REC - 1 TO MAX:REC
            ENDIF
         ENDIF
      ELSE
         IF # <> REC:NO
            DELETE
            PACK
            STORE MAX:REC - 1 TO MAX:REC
         ENDIF
      ENDIF
   ENDDO
ENDDO
USE
RELEASE I, MAX:REC, LAST, REC:NO,CONTENT,CONDITION,MDELETE


*********************************************************************
*                            VARIABLE.PRG                           *
* This procedure, which is called by procedure CALCULATE.prg,       *
* finds a constant value for an example element in the avtable.     *
*********************************************************************

STORE $(EXPRESSION,1,1) TO VARIABLE
STORE $(EXPRESSION,2) TO EXPRESSION
IF $(EXPRESSION,1,1) = ' '
   STORE '*** ERROR - NO  BLANK CAN BE EMBEDDED INTO EXAMPLE'+;
         ' ELEMENT' TO ERROR:MSG
   STORE T TO ERROR
   RETURN
ELSE
  DO WHILE $(EXPRESSION,1,1)$'0123456789' .OR.
           $(EXPRESSION,1,1)$'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      STORE VARIABLE + $(EXPRESSION,1,1) TO VARIABLE
      STORE $(EXPRESSION,2) TO EXPRESSION
   ENDDO
   USE B:AVTABLE
   STORE F TO FOUND
   DO WHILE .NOT. EOF .AND. .NOT. FOUND
      SET EXACT ON
      IF VALUE = VARIABLE
         STORE TRIM(ATTR) TO MATTR
         IF TYPE <> 'N'
            STORE "*** ERROR - COLUMN &mattr IS NOT A NUMERICAL"+;
                  " TYPE, CANNOT DO CALCULATION" TO ERROR:MSG
            STORE T TO ERROR
            RETURN
         ELSE
            STORE T TO FOUND
            STORE TABLEID TO MTABLEID
            STORE DEC TO VDEC
```

```
STORE LEN TO VLEN
STORE # TO REC:NO
DELETE
PACK
STORE MAX:AV - 1 TO MAX:AV
IF REC:NO  < I
   STORE I - 1 TO I
ENDIF
STORE F TO FIND:VAL
IF REC:NO > 1
   STORE REC:NO - 1 TO PREVIOUS
   GOTO PREVIOUS
   IF TABLEID = MTABLEID
      IF VALUE <> ' ' .AND. $(VALUE,1,1) <> '_';
         .AND. $(VALUE,1,1) <> '('
          STORE T TO FIND:VAL
          IF TYPE = 'C'
             STORE TRIM(ATTR) + '=' + '"' + ;
                   TRIM(VALUE) + '"' TO CONDITION
          ELSE

             STORE TRIM(ATTR) + '=' + TRIM(VALUE) ;
                   TO CONDITION
          ENDIF
          DELETE
          PACK
          IF PREVIOUS < I
             STORE I - 1 TO I
          ENDIF
          STORE MAX:AV - 1 TO MAX:AV
          STORE REC:NO - 1 TO REC:NO
          RELEASE PREVIOUS
          LOOP
      ENDIF
   ENDIF
ENDIF
IF REC:NO <= MAX:AV
   STORE REC:NO TO NEXT
   GOTO NEXT
   IF TABLEID = MTABLEID
      IF VALUE <> ' ' .AND. $(VALUE,1,1) <> '_';
         .AND. $(VALUE,1,1) <> '('
         STORE T TO FIND:VAL
         IF TYPE = 'C'
            STORE TRIM(ATTR) + '=' + '"' + ;
                  TRIM(VALUE) + '"' TO CONDITION
         ELSE
            STORE TRIM(ATTR) + '=' + TRIM(VALUE) ;
                  TO CONDITION
         ENDIF
         DELETE
         PACK
         IF NEXT < I
            STORE I - 1 TO I
         ENDIF
```

```
                        STORE MAX:AV - 1 TO MAX:AV
                        STORE REC:NO - 1 TO REC:NO
                        RELEASE NEXT
                        LOOP
                     ENDIF
                  ENDIF
               ENDIF
            ENDIF
         ENDIF
         SKIP
      ENDDO
      SET EXACT OFF
      IF .NOT. FOUND
         STORE T TO NO:VALUE
         STORE MATTR TO MVAL
      ELSE
         IF .NOT. FIND:VAL
            STORE "*** ERROR - EXAMPLE ELEMENT &variable CAN NOT"+;
                  " FIND A CONSTANT TO MATCH IT" TO ERROR:MSG
            STORE T TO ERROR
            RETURN
         ELSE
            STORE 'TEMP' + $(MTABLEID,1,1) TO TEMP
            USE B:&TEMP
            COPY TO B:BUFFER FOR &CONDITION
            USE B:BUFFER
            GOTO BOTTOM
            IF # = 1
               STORE &MATTR TO MRESULT
               STORE STR(MRESULT,VLEN,VDEC) TO RESULT
               STORE RESULT TO MVAL
               RELEASE MRESULT,RESULT
            ELSE
               STORE "***ERROR - TOO MANY VALUES OR NO VALUE FOR "+;
                     "EXAMPLE ELEMENT" TO ERROR:MSG
               STORE T TO ERROR
            ENDIF
            RELEASE CONDITION,TEMP
         ENDIF
         RELEASE FIND:VAL,VDEC,VLEN,REC:NO
      ENDIF
   ENDIF
   RELEASE VARIABLE, FOUND
   RETURN
```

A MICROCOMPUTER IMPLEMENTATION OF QUERY-BY-EXAMPLE

by

Li-Ling Chen

B.S., Feng Chia University, 1980

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

ABSTRACT

Query-By-Example (QBE) is a relational data base query language that provides the end user with a simplified approach to manipulating data. It is a high-level language that is simple to learn and use while also providing a powerful capability for defining, retrieving, updating, inserting, and deleting data. The main philosophy behind QBE is to keep the number of objects and concepts at a minimum so that a user can begin using the language after just a brief introduction.

Using a series of examples, an overview of QBE is presented first. A description of the programming language dBase II is also given. The prototype implementation of QBE, accompanied by an analysis of the algorithm that interprets queries, is also presented. This algorithm was programmed in dBase II programming language and was implemented on the Zenith-150 microcomputer. The algorithm utilizes four phases of interpreting or processing queries and it relies on dBase II system to carry out actions specified by a QBE query.