The Usage of Query-By-Example in a Microcomputer Environment

By

Shun-Jane Lin

B.S., Chung-Hsing University, Taiwan, R.O.C., 1983

-----------------------------------------------

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:

[Signature]
Major Professor
# TABLE OF CONTENTS

**Chapter 1. Introduction** ............................................. 1
  1.0 Overview ....................................................... 1
  1.1 Relational Query Language ..................................... 1
  1.2 The Features of QBE ........................................... 2
  1.3 Motivation for Porting QBE .................................... 3
  1.4 Report Organization ........................................... 4

**Chapter 2. Review of Literature** ................................. 5
  2.0 Introduction ................................................... 5
  2.1 The QBE Language ............................................. 5
    2.1.1 Screen Object ........................................... 6
    2.1.2 The Components of QBE Language ........................ 7
  2.2 The Facilities of QBE ........................................ 10
    2.2.1 Retrieval Operations .................................... 11
    2.2.2 Insertion, Deletion, and Update operations .......... 15
    2.2.3 Table Operations ........................................ 17
  2.3 History of QBE ............................................... 19
  2.4 Extensions of QBE ............................................ 19

**Chapter 3. System Implementation** .............................. 22
  3.0 Introduction ................................................... 22
  3.1 dBASE-II ....................................................... 22
  3.2 The QBE Translator ............................................ 24
  3.3 Parsing a Simple Query ...................................... 33
  3.4 Limitations .................................................... 35

**Chapter 4. Conclusion** ............................................ 35

**References** .......................................................... 40

**Appendices**

A. User Manual ..................................................... 45
B. Logging Queries .................................................. 65
C. The comparison of Two Version of QBE ......................... 67
D. The Hierarchical Diagram for the QBE system ............... 69
E. Program Listing .................................................. 77
THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE. THIS IS AS RECEIVED FROM CUSTOMER.
List of Figures

Figure 2.1 General Table ............................................. 6
Figure 2.2 Condition Table .......................................... 7
Figure 2.3 The Operations of QBE Language ....................... 9
Figure 2.4 Simple Retrieval ......................................... 11
Figure 2.5 Simple Retrieval with Ordering ....................... 11
Figure 2.6 Simple Retrieval with Multiple Prints ............... 12
Figure 2.7 Retrieval of Table Name ................................. 12
Figure 2.8 Retrieval of Column Heading ......................... 12
Figure 2.9 Qualified Retrieval using Links ....................... 13
Figure 2.10 Retrieval Using the Condition Box ................. 13
Figure 2.11 Retrieval of Collected Output from Multiple Tables 14
Figure 2.12 Simple Retrieval Using a Built-in Function ....... 14
Figure 2.13 Simple Insertion ........................................ 15
Figure 2.14 Simple Deletion ......................................... 16
Figure 2.15 Simple Update .......................................... 16
Figure 2.16 Query-Dependent Insertion ........................... 16
Figure 2.17 Query-Dependent Deletion ............................ 16
Figure 2.18 Query-Dependent Update ............................... 17
Figure 2.19 Table Creation ......................................... 18
Figure 2.20 Table Deletion ......................................... 20
Figure 3.1 The Process Diagram of QBE Translator .............. 24
Figure 3.2 The Process of Parsing The Table Name Entry ....... 26
Figure 3.3 An Example Query ...................................... 27
Figure 3.4 Sfile1 .................................................... 28
ACKNOWLEDGEMENTS

I wish to thank Dr. Richard McBride for his help and guidance in completing this project. And a special thanks to my parents for love, encouragement and financial support.
Chapter 1
Introduction

1.0 OVERVIEW

Query-By-Example (QBE) is a high level, non-procedural database language which provides users with facilities to query, update, and create a database. All requests from a user are specified in QBE by filling in two-dimensional skeleton tables on the screen; this technique is known as two-dimensional programming. This unique and novel method allows the end user to express, with very few entries, the equivalent of a lengthy application program in a conventional language.

This report contains a description of an implementation of QBE in a microcomputer environment. QBE's language and facilities are also described through illustrative examples.

1.1 RELATIONAL QUERY LANGUAGE

The relational data base model was introduced by E. F. Codd in 1970 [COD70]. The central element of this model, a 'relation', can easily be mapped into a user's mental frame. Various high level data manipulation languages have been used to support this model in order to make these languages easy to use. Basically these languages can be divided into two major classes:

1. Languages based on relational algebra; and
2. Languages based on relational calculus which can be divided further into tuple relational calculus and domain relational calculus. The domain relational calculus language differs from the tuple relational calculus in that its variables range over domains rather than relations [ULL83].
In an algebraic language, the queries are expressed by applying specialized operators to relations. However, in a calculus-based language, the queries describe a desired set of tuples by specifying a predicate that the tuples must satisfy. Although both language classes are considered to be high level, it is often said that relational calculus-based languages are of a higher level than the algebraic languages. This is due to the fact that a program in a algebra-based language specifies the execution order of operations while a program in the calculus leaves it to the compiler or interpreter to determine the most efficient order of evaluation [ULL83].

A list of some existing languages, most of which are partially implemented or in use, is given for the reader's reference:

PRTV -- based on relational algebra [TOD76];
ALPHA -- based on relational calculus [COD71];
SQL2 SQUARE -- based on a mixture of relational calculus and algebra [CHA76, BOY73];
QUEL -- based on relational tuple calculus [STO75];
Query-By-Example -- based on domain relational calculus;
CUPID -- based on relational algebra [MCD75]; and
FORAL -- based on the binary relational model [SEN77].

1.2 THE FEATURES OF QBE

As mentioned in the previous section, QBE is a non-procedural language. The advantage of a non-procedural language is that a user need not possess the knowledge, required to specify a sequence of operations to be performed on the database,
in order to obtain a desired result. The QBE user is only required to have that knowledge of the data base structure which is needed to specify the data that is desired from the system. The translation of non-procedural QBE statements into a series of procedural language statements is done by an interpreter in the underlying system.

The syntax of QBE is very simple, yet it permits a wide variety of complex transactions to be expressed. The basic principle behind QBE is the simplicity of its user interface. This interface allows information to be specified by the user concerning desired information contained in the data base. The simplicity is provided by displaying a skeleton table in which user can enter a query by providing an example of a possible answer. To use QBE, a user only needs to have knowledge of the simple concepts of an example element, i.e. a variable, and a constant element. Further details of Query-by-Example concepts will be introduced in Chapter 2 and Appendix A through a collection of illustrations of queries and their answers.

1.3 MOTIVATION FOR PORTING QBE

Computers have become cheaper and widely available in the last decade. Because of this, interactive computing is in use in many businesses, and home computers have become commonplace. The people who own or use computers are often not professional programmers, so there is a need for a powerful tool which is easy to use and requires a minimum of user training. QBE is an appropriate system that meets such requirement since it is non-procedural and has a simple user interface.
QBE is available from IBM as an installed user program running on the VM/CMS operating system [IBM78]. QBE has been implemented on an Apple II microcomputer [COM80]; however, no details of this implementation were available to aid in this project. The version of QBE that was developed at KSU utilizes the facilities of dBASE-II [ASH81]. The user's queries are translated into a series of dBASE-II command statements in order to access the information in the data base.

The dBASE-II software can run on CP/M, MS-DOS, or Cromix operating systems. Since the Zenith 150 computer systems run MS-DOS and are widely available on campus, it was chosen as the microcomputer to which QBE would be ported. However, because of the characteristics of the Zenith 150, the implemented QBE system also runs on any IBM compatible computer system.

1.4 REPORT ORGANIZATION

The intent of this project is to implement Query-by-Example for use on a microcomputer. Since the implementation of QBE was a cooperative effort, this report concentrates on those facilities in QBE that were implemented by the author, namely, the update, delete, and insert facilities.

The purpose of this project and the general features of QBE have been discussed in this chapter. Chapter 2 provides a closer look at QBE's language and facilities. Details of how QBE statements are translated and executed using the facilities of dBASE-II are presented in chapter 3. Finally, chapter 4 summarizes the results of this part of the project and discusses possible enhancements.
Chapter 2
Review of Literature

2.0 INTRODUCTION

Before going into the implementation details of the QBE project, an overview of the QBE language is provided in this chapter. The basic components or objects of the language are discussed. Also, the facilities of QBE are introduced through a set of examples which illustrate the operations of data retrieval and manipulation. Since this chapter only briefly introduces the language, more details concerning the use of QBE on the Zenith 150 microcomputer are presented in the user manual (Appendix A). The history of QBE and languages based on it are also discussed in this chapter.

2.1 THE QBE LANGUAGE AS PROPOSED BY IBM

IBM's version of QBE is a language with a two-dimensional, graphic input format. A query in QBE is expressed through the use of a skeleton table. Two questions might be asked by a user who wants to start using the system. The first one is - what kinds of tables are provided by the system? The second question is - what can be put in each type of table? In order to answer these questions, this section describes tables or screen objects, and the elements of the QBE language.
2.1.1 Screen Objects

There are three types of screen objects in the QBE language: general tables, result tables and condition boxes. Each type of screen object has a purpose which is described below.

A general table is used in the operations of retrieval, change and creation of a relation in the data base. Unless a user desires to create a new table, each table must be associated with an existing relation in the data base. Any general table may contain four types of entries (see Figure 2.1): a table name entry, field name entries, row operator entries and data field entries. The table name entry is the place for a user to enter the name of the table that is to be dealt with. Each field name entry specifies a field name for a table (field names can also be generated automatically by the system). A row operator entry gives the QBE operation to be performed. There are four row operators provided by QBE: print(P.), delete(D.), insert(I.), and update(U.). A data field entry specifies a portion of the user's query, i.e., the data field entries describe what data are to be manipulated.

<table>
<thead>
<tr>
<th>table name entry</th>
<th>field name entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>row operator entry</td>
<td>data field entry</td>
</tr>
</tbody>
</table>

Figure 2.1 General Table

A result table is used to specify a format when the user
wants to print out or display a report. A result table also can be used when the user wants to store a new table whose contents are derived from existing tables. The data types of the fields of the new table are inherited from the relation which the data is taken. A result table may contain the same four types of entries which are used in a general table.

A condition box is used to specify conditions for complex query operations. The condition box only contains one type of entry, namely, a condition-entry (see Figure 2.2). A condition entry consists of elements (example elements or constant elements which are explained shortly) and relational operators. The evaluation of a condition should result in a boolean value. Each condition entry is specified on a separate line and results of all these entries are logically ANDed together; that is, the query will operate only if all of the conditions are true.

```
<table>
<thead>
<tr>
<th>CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition-entry</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

Figure 2.2 Condition Box

2.1.2 The Components of QBE Language

The QBE language consists of elements, operators, and expressions.

Elements are used to define and qualify data. There are two types of elements used in the QBE language: constant elements and example elements. Constant elements are used in specifying selection criteria; that is, to qualify or limit the results to
values which match a relation on the constant element. A constant element requires no special notation.

Example elements are used to: link data among different tables or different rows in the same table, specify conditions, and map data from one table to another. Example elements do not exist in the data base; they merely represent an example of a possible answer to a query. Example elements must begin with a prefixed underscore and be followed by one or more letters (e.g., _A, _RED).

QBE operators are used to define processing and to specify conditions. The types of operators provided by the QBE language as follows:

- System operators which define processing;
- Built-in functions, which determine the average, maximum, sum, count, minimum, and uniqueness (which eliminates duplicates) of some data items;
- Relational operators which permit comparisons to be made and conditions to be defined;
- Arithmetic operations;
- Logical operators which link conditions together.

All of the operators available in QBE are explained in Figure 2.3.
<table>
<thead>
<tr>
<th>Type</th>
<th>Operators</th>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>System operators</td>
<td>P.</td>
<td>print</td>
</tr>
<tr>
<td></td>
<td>I.</td>
<td>Insert</td>
</tr>
<tr>
<td></td>
<td>U.</td>
<td>update</td>
</tr>
<tr>
<td></td>
<td>D.</td>
<td>delete</td>
</tr>
<tr>
<td></td>
<td>AO.</td>
<td>in ascending order (used with print operator)</td>
</tr>
<tr>
<td></td>
<td>DO.</td>
<td>in descending order (used with print operator)</td>
</tr>
<tr>
<td>Built-in functions</td>
<td>CNT.</td>
<td>count data item</td>
</tr>
<tr>
<td></td>
<td>SUM.</td>
<td>sum data item</td>
</tr>
<tr>
<td></td>
<td>MAX.</td>
<td>find the maximum value</td>
</tr>
<tr>
<td></td>
<td>MIN.</td>
<td>find the minimum value</td>
</tr>
<tr>
<td></td>
<td>AVG.</td>
<td>calculate the average</td>
</tr>
<tr>
<td></td>
<td>UNQ.</td>
<td>eliminate (all but one) identical data items</td>
</tr>
<tr>
<td>Relational comparisons</td>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td></td>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td></td>
<td>&lt;&gt;</td>
<td>not equal to</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>Logical</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and</td>
</tr>
</tbody>
</table>

| Figure 2.3  The operators of QBE language. |
Expressions are constructed out of a combination of elements and operators. They are used to qualify or limit the data that the user wants to deal with. There are two types of expressions which are allowed in QBE language:

. Arithmetic expressions, which combine example and constant elements by using arithmetic operators; and
. Logical expressions, which combine example and constant elements by using logical operators.

2.2 THE FACILITIES OF QBE

There are four system operators, P., I., U., D., available in QBE. These four operators allow a user to create, query, and update a data base. 'P.' stands for print; it is the operator used to retrieve information from the data base. The 'I.' (insert) operator allows the information to be added to the data base. 'U.' is the operator which is used to update or change information values. 'D.' stands for deletion; it is the operator used to delete or drop information from the data base.

The use of the print operator does not have any side effects upon the data base. Whenever an insert, delete, or update operator is used in a table skeleton, a corresponding change is made in the data base. In the following text, several examples using these operations are given in order to illustrate the basic concepts concerning the usage of these operators. A detailed syntactic and semantic description of QBE language is covered in the user manual (Appendix A).
2.2.1 Retrieval Operations

An example of a simple retrieval is shown in Figure 2.4. The user fills in the name of the table in the table name field; in this case, the name is PART. Next, the user may either fill in the column heading or let the system generate them automatically. The user next expresses a query by entering the print operator(P.) and an example element (which is prefixed with underscore) to print out all PART colors.

<table>
<thead>
<tr>
<th>PART</th>
<th>PNAME</th>
<th>COLOR</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>P._C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.4 Simple Retrieval

To retrieve the data with ordering. The user can enter the print operator along with a system operator(AO. or DO.). The system prints out the data depending upon the request. The example shown in Figure 2.5 prints out all part names in alphabetical order. 'AO.' stands for ascending order, and 'DO' stands for descending order.

<table>
<thead>
<tr>
<th>PART</th>
<th>PNAME</th>
<th>COLOR</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.AO._C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.5 Simple Retrieval with Ordering

An example of a simple retrieval with multiple fields being printed is shown in Figure 2.6. The user can print out the entire PART table by entering the print operator(P.) in the first row
operator field.

<table>
<thead>
<tr>
<th>PART</th>
<th>PNAME</th>
<th>COLOR</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.6 Simple Retrieval with Multiple Prints

To list the available table names in the data base, the user enters a print operator and an example element in the table name field (see Figure 2.7).

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

Figure 2.7 Retrieval of Table Name

To see the column headings of a table, the user first fills in the table name to be worked with. This is followed by print operator. The example in Figure 2.8 shows how QBE can be used to print out the attributes of the PART table.

<table>
<thead>
<tr>
<th>PART P</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.8 Retrieval of Column Heading

Qualified retrieval using links is shown in the example in Figure 2.9. In order to print the names of all the red parts made by the manufacturer YKK, the user uses two tables, the PART table and the MAKE table. The same example element is used in both tables, indicating that in order for a part to meet the criteria
specified by this query it must be red, and that same item should also be made by manufacturer YKK. Only if these condition are met simultaneously does the item qualify as a solution.

<table>
<thead>
<tr>
<th>PART</th>
<th>PNAME</th>
<th>COLOR</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.</td>
<td>_P</td>
<td>RED</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MAKE</th>
<th>MNAME</th>
<th>PNAME</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>YKK</td>
<td>_P</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.9 Qualified Retrieval using Links

A condition box can be used to express one or more desired conditions that would be difficult to express in the tables. An example is shown in Figure 2.10, in which the names of all parts whose weight is heavier than the total weight of bolt and nut, but also lighter than 10 pounds are printed. The user can use example elements (in this case, _W1, _W2, _W3) and specify conditions by using example elements combined with relational operators in a condition box.

<table>
<thead>
<tr>
<th>PART</th>
<th>PNAME</th>
<th>COLOR</th>
<th>WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.</td>
<td>_W1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOLT</td>
<td>_W2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUT</td>
<td>_W3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CONDITION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_W1 &gt; _W2 + _W3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_W1 &lt; 10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.10 Retrieval using the Condition Box
The user can combine together output that was retrieved from multiple tables. This is done by making use of a result table. An example of this is shown in Figure 2.11, where each manufacturer along with its corresponding buyer is printed out. Since the output is a new table, the user must use a third table. The third table is filled in with examples mapped from two existing tables which satisfy the stipulation of the query.

```
<table>
<thead>
<tr>
<th>BUY</th>
<th>PNAME</th>
<th>BUYER</th>
</tr>
</thead>
<tbody>
<tr>
<td>_P</td>
<td>_BUY</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MAKE</th>
<th>MNAME</th>
<th>PNAME</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>_N</td>
<td>_P</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RESULT</th>
<th>MNAME</th>
<th>BUYER</th>
</tr>
</thead>
<tbody>
<tr>
<td>P._N</td>
<td>P._BUY</td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 2.11 Retrieval of Collected Output from Multiple Tables

QBE provides a set of built-in functions (as discussed in the previous section). These built-in functions can be used in specifying the desired answer. For example, the user can find the total number of manufacturers by entering a print operator along with a built-in function name (see Figure 2.12).

```
<table>
<thead>
<tr>
<th>MAKE</th>
<th>MNAME</th>
<th>PNAME</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.CNT.UNQ.ALL.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 2.12 Simple Retrieval using a Built-in Function
2.2.2 Insertion, Deletion, and Update Operations

Insertions(I.), deletions(D.), and updates(U.) are done in the same style as the query operation. The following examples are categorized into two parts: simple insertions, deletions, and updates (the term 'simple' applies to operations that involve constant elements only), and those that are query-dependent i.e., certain information has to be extracted from the database in order to accomplish a query.

One tuple may be added into an existing table by placing an insert operator(I.) in the row operator field, and the information in each column for the table. Figure 2.13 demonstrates how to insert part name 'WASHER' (that has a color of 'RED', a weight of 5, and is made in the city of 'LONDON') into the PART table. The user specifies the new tuples as in the example shown in Figure 2.13.

<table>
<thead>
<tr>
<th>PART</th>
<th>PNAME</th>
<th>COLOR</th>
<th>WEIGHT</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>WASHER</td>
<td>RED</td>
<td>5</td>
<td>LONDON</td>
</tr>
</tbody>
</table>

Figure 2.13 Simple Insertion

The user may delete information from a table by entering the delete operator(D.) in the row operator field and the condition for this operation. The example shown in Figure 2.14, causes QBE to delete all of the information about parts made in the city 'PARIS'.

15
<table>
<thead>
<tr>
<th>PART</th>
<th>PNAME</th>
<th>COLOR</th>
<th>WEIGHT</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PARIS</td>
</tr>
</tbody>
</table>

Figure 2.14 Simple Deletion

The user can modify the information in a relation by entering the update operator (U.) followed by a new value in the column that is to be changed. As the result of the example shown in Figure 2.15, QBE updates the color of all parts named bulb to be blue.

<table>
<thead>
<tr>
<th>PART</th>
<th>PNAME</th>
<th>COLOR</th>
<th>WEIGHT</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BULB</td>
<td>_COLOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>U.BLUE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.15 Simple Update

The following is an example of a query-dependent insertion. A user can add tuples whose values are derived from other tuples in the database. The example shown in Figure 2.16 causes QBE to find the colors of bulb, and then insert the colors with the information for all tuples with weight '4' and the part name 'BOX' into the PART table.

<table>
<thead>
<tr>
<th>PART</th>
<th>PNAME</th>
<th>COLOR</th>
<th>WEIGHT</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>BOX</td>
<td>_COLOR</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BULB</td>
<td>_COLOR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.16 Query-Dependent Insertion

In the example shown below, QBE determines the number of products manufactured by YKK. QBE then deletes all of the
information about any other company which manufactures more products than YKK.

\[
\begin{array}{|c|c|c|c|}
\hline
\text{MAKE} & \text{MNAME} & \text{PNAME} & \text{QTY} \\
\hline
\text{D.} & & >_Q & \\
\text{D.} & \text{YKK} & >_Q & \\
\hline
\end{array}
\]

Figure 2.17 Query-Dependent Deletion

Sometimes it is desirable to update an entry with a value based upon its previous value. Suppose the weight of part name 'washer' must be increased by 5. Figure 2.18 demonstrates how this can be accomplished; QBE will find the old value of the part WASHER and increase it by 5. Finally, the weight of the part WASHER is updated to the new value.

\[
\begin{array}{|c|c|c|c|}
\hline
\text{PART} & \text{PNAME} & \text{COLOR} & \text{WEIGHT} & \text{CITY} \\
\hline
\text{WASHER} & & \text{WT} & \\
\text{WASHER} & & \text{U.WT+5} & \\
\hline
\end{array}
\]

Figure 2.18 Query-Dependent Update

2.2.3 Table Operation

In the QBE language, the creation of a table is done in the same style as the previous operations.

Suppose it is necessary to create a new table with table name STU and column headings: NAME, ADDR, GPA, DEPT (see Figure 2.19). Starting with a blank skeleton on the screen, the user fills in the headings by inserting the field names. For each field name, the following row attributes have to be specified:

- **TYPE** specifies the data entry type, such as float, char, fixed, etc.,
LENGTH specifies the length of that field,

KEY specifies the fields that are considered to form the primary key of that table,

DOMAIN specifies the name of the underlying domain of the field, i.e., the value set from which the field's data elements are drawn,

INVERSION specifies whether or not an index is to be built based on the field (QBE assumes that every field is part of the key and that every field is to be indexed, unless informed to the contrary).

<table>
<thead>
<tr>
<th>I.</th>
<th>STU I.</th>
<th>NAME</th>
<th>ADDR</th>
<th>GPA</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>I.</td>
<td>CHAR</td>
<td>CHAR</td>
<td>FLOAT</td>
<td>CHAR</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>I.</td>
<td>NAMES</td>
<td>ADDS</td>
<td>GPAS</td>
<td>DEPTS</td>
</tr>
<tr>
<td>LENGTH</td>
<td>I.</td>
<td>20</td>
<td>30</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>INVERSION</td>
<td>I.</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

Figure 2.19 Table Creation

The operator U. can be used to rename a table name. For example, to update the table name from STU TO STU1, the user just enters 'STU U.STU1' in the table name field.

The user can remove a table from the data base by entering in the table name field the delete operator followed by the name of the table to be removed. For example, to remove the table named PART from the data base, the user just enters 'D. PART' into the table name field (as in the following example).
2.3 HISTORY OF QBE

Initially, QBE was intended as a query language for relational data bases. Since then it has been developed and extended to form a complete data base manipulation language with facilitites to update, and create a data base. Because most of the operations used for querying the data base are also employed in updating and creating a data base, the name of QBE language was not altered.

The first implementation of Query-by-Example was done by K. E. Niebuhr and S. E. Smith[NI77]. Afterward, M. M. Zloof developed it into a commercial product at the IBM Yorktown Heights Research Laboratory, and the product was released in 1978.

2.4 EXTENSION OF QBE

Since QBE was introduced, two programming languages based on QBE have been developed for business and office automation. Two extended versions of QBE are the System for Business Automation (SBA)[ZL77] and Office Procedures by Example (OBE) [ZL81], [ZL82].

SBA is a system in which business applications are described
and executed on a computer. The user of SBA views an application as manipulation of information in two-dimensional pictures of tables, business forms and reports on a terminal. A user can gradually automate an application by giving "examples" to the system of how to manipulate the information.

The basic data object of QBE is a table. QBE uses the examples (specified by the user) through the use of tables to describe a complete data base management system. Each QBE transaction is a non-procedural specification of a complex data base query or modification. An SBA program is composed of a collection of QBE transactions over tables or more complex data object such as forms, charts, and reports. In addition, the program uses the example provided by the user for specification of program invocations, parameters, triggers, and user interactions.

OBE is also a two-dimensional language and system that is an attempt to mimic manual procedures of business and office systems. OBE combines and unifies aspects of word processing (including editing and formatting), data processing, report writing, graphics and electronic mail. Because of this combination, OBE supplies more general data objects, such as letters, forms, reports, charts and graphs, than QBE whose fundamental data object is the table. With such a language, end users are able to specify and store complex programs, thus developing their own applications.

One important feature provided by both languages is that the user can express various trigger conditions. This provides the system with an ability to act automatically when a specified
condition is met. The trigger feature allows the user to automate many routine business procedures and devote the time saved to more advanced tasks.

Both of these languages use QBE as their data base management system, so that users can easily retrieve the data from the data base and map it into the body of objects. The programming style of OBE and SBA are essentially the same as that of QBE, and so the users of QBE require very little additional training to be able to use OBE or SBA.
3.0 INTRODUCTION

This chapter discusses the implementation of update, insert, and delete facilities. In this chapter, the term 'QBE' will be used to refer to the version of QBE which was implemented on the microcomputers at Kansas State University. Because there are many procedures involved in the implementation of QBE, the implementation procedures are not discussed in detail in this chapter. However, the interested reader may refer to Appendix C for the internal documentation. The facilities of dBASE-II, which support QBE, are briefly discussed in this chapter. Also, the limitations of QBE are described at the end of this chapter. A user's manual (Appendix A) contains some examples which illustrate the specification of the QBE queries and their results.

3.1 dBASE-II

dBASE-II[ASH81] is a data base management tool that allows easy manipulation of small and medium-sized databases using English-like commands. dBASE-II provides users with very powerful commands to manipulate the data base. These commands can be categorized into three types: create, update, and print.

1) Create: the user can create a data file with a maximum of 63,535 records. Each record may contain up to 1000 characters and 32 fields.

2) Update: the INSERT, DELETE, REPLACE, and EDIT commands can
be employed to update the information in the data base.

(3) Print: the user can use the commands, DISPLAY or LIST, to show the information in the data base either on screen or paper. dBASE-II also allows the user to create a report form, where the content of the report is defined by the user.

These commands are used to manipulate the data in the data base. The data base consists of data files in dBASE-II which are called data base files and which have file name extensions of '.dbf'. A dBASE-II data base file consists of a structure record and data record. A structure record is essentially a map of the data record format. The structure can hold up to 32 different entries. Each entry in the structure refers to a field of data in the data records. The QBE user's data is stored in data base files.

It is inconvenient for the user to go through the same sequence of commands each time the user needs to employ that sequence. For the user's convenience, dBASE-II provides a method of saving a sequence of frequently used commands. This is accomplished by creating a command file which contains a sequence of dBASE command statements. A user may also create loops (using the DO WHILE statement) and make conditional jumps (using the IF statement) in a dBASE command file. dBASE-II starts from the top of the command file and processes the commands one at a time until either a RETURN statement or the end of the command file is encountered. The translator of QBE is coded using command files.

dBASE-II is quite a powerful system. To learn more about the system, the reader may refer to the user manual and reference
manual published by Ashton-Tate[ASH81].

3.2 THE QBE TRANSLATOR

This section describes how users' queries utilizing the operators I., U., and D. are carried out by the QBE translator. Basically, there are six stages involved in the translation process (see Figure 3.1). The details of each step are discussed in the following text. The final step, converting a user's query into a dBASE command, is not discussed separately, because it is already incorporated in each of the steps.

```
+-------------------------+              
| PARSE THE              |
| TABLE NAME ENTRY       |
+-------------------------+              |
|                        |
+-------------------------+              
| STORE THE USER'S QUERY |
+-------------------------+              
|                        |
+-------------------------+              
| PARSE THE FIELD NAME   |
| ENTRIES                |
+-------------------------+              
|                        |
+-------------------------+              
| PARSE THE ROW OPERATOR |
| ENTRIES                |
+-------------------------+              
|                        |
+-------------------------+              
| PARSE THE DATA FIELD   |
| ENTRIES                |
+-------------------------+              
|                        |
+-------------------------+              
| CONVERT THE USER'S     |
| QUERY INTO A dBASE     |
| COMMAND                |
```

Figure 3.1 The Process Diagram of QBE Translator
Stage 1. Parse the table name entry

In the microprocessor version of QBE, a user is always asked to fill in the table name and any accompanying operation to be performed on the entire table first. After a user fills in the table name entry, several actions are performed:

1) a check is made to see if there are any of the allowable table operators, such as I.I., D., and U., present in that entry.

2) the file name is checked to ensure that it is valid.

If the operator I.I. is in the entry, it means that the user wants to create a new table. The system then displays a two-dimensional table with five heading columns: the file name which was specified by the user, the field name, the field type, the field length, and the field decimal (which is used to specify the number of digits after the decimal point). For each field in the table, the user has to enter the field name to be contained in the table and the information about that field. The new table is created based on the information that is entered by the user. After the new table is created, the dBASE-II input facility is used to accept input data from the user.

If the operator D. is present in the entry, it means that the user wants to delete an existing table from the data base. To prevent the user from accidentally damaging the data base, a message is displayed on the screen in order to confirm the user's request before the delete operation actually takes place.

If the operator U. is present in the entry, it means that the user wants to rename a table. In this case, the system
prompts the user for a new name, which replaces the old table name.

If none of the above operators are present in the table name entry, it means that the user is requesting a table skeleton for the named table. Before the table is displayed, the file name entered by the user is checked to see if it already exists in the database. If the file name is not already in existence, then the user is asked to re-enter, otherwise the process goes on to stage 2. Figure 3.2 provides a process diagram of stage 1.

![Diagram of process](image)

**Figure 3.2** The Process of Parsing the Table Name Entry
Stage 2. Store a user's query

A user's query is stored in data base files (.dbf) for further parsing. There are four data base files (.dbf) which are used to store the user's queries: sfile1, sfile2, result, and condition. 'sfile1' and 'sfile2' are used to store the portions of the query which appear in the two general tables. General tables are QBE tables which allow thirteen or fewer characters in each column. The information contained in the first general table which is shown on the screen is stored in 'sfile1', and any information in a second general table is stored in 'sfile2'. The information in 'sfile1' and 'sfile2' is the actual data for the insert operation or the selection criteria for the delete or update operation. 'result' is used to store the part of a query that was put in a result table. Any information appearing in a condition box is stored in 'condition'.

A sample query is shown in Figure 3.3, and Figures 3.4 and 3.5 shows how the information in general tables and condition boxes is stored in data base files (.dbf). The method used to save the piece of a query shown in a result table is the same as that used for a general table.

The query in Figure 3.3 is stored in two data base (.dbf) files. The information from the general table is stored in sfile1.dbf (see Figure 3.4) and the information from the condition box is stored in condition.dbf (see Figure 3.5).
ENTER FILE NAME (AND COMMAND IF ANY): EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>PHONE NO</th>
<th>DEPT</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>STEVE</td>
<td>10 ST.</td>
<td>5370416</td>
<td>HARDWARE</td>
<td>_G</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TOY</td>
<td>_S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>_G = SUM.UNQ._S</td>
</tr>
</tbody>
</table>

Figure 3.3 The Example Query

<table>
<thead>
<tr>
<th>REC</th>
<th>FLD:</th>
<th>NAME</th>
<th>ROW1</th>
<th>ROW2</th>
<th>ROW3</th>
<th>TYPE</th>
<th>LEN</th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EMP</td>
<td>I.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NAME</td>
<td>STEVE</td>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADDRESS</td>
<td>10 ST.</td>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PHONE NO</td>
<td>5370416</td>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>DEPT</td>
<td>HARDWARE</td>
<td>_S</td>
<td></td>
<td></td>
<td>C</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SALARY</td>
<td>_G</td>
<td>TOY</td>
<td></td>
<td></td>
<td>N</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.4 Sfile1 (.dbf)

<table>
<thead>
<tr>
<th>REC</th>
<th>CONDITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>_G = SUM.UNQ._S</td>
</tr>
</tbody>
</table>

Figure 3.5 Condition (.dbf)

Stage 3. Parse the field name entries

Before parsing a user's field name entry, each entry is checked to see if it has a corresponding field in an actual file in the data base. If it does not, the query processing is terminated and a message is displayed on the screen.
In this stage, the field name entries are parsed to see if the user desires to expand an existing table. If a field name is prefixed with an insert operator, it means that a table expansion operation is requested. If a table expansion is not requested, then the translator jumps to stage 4 (to parse the row operator entries). In the case of a table expansion, a two-dimensional table (as shown in query 6 of Appendix A) is then displayed. The user must fill in the new field with its type, length, and the number of digits to the right of the decimal point. The information entered in the table is used to expand the file that is specified by the user.

Stage 4. Parse user's row operator entry and process row operation

The command rows are parsed to determine which row operation, insert or delete, is desired. If there is no command found in the row operator area, the process goes on to stage 5 (to parse the data field entry).

Although delete(D.) and insert(I.) are very different function operators, the QBE translator uses the same method to carry out both of the operations.

If a row operator is specified, then the operation is based upon the information contained in the data field. The data field entry becomes the information which is added into the table for the insert operation or specifies criteria for the deletion. Therefore, the major work of this stage is to interpret the data field entry in order to perform the required operation and obtain the desired result. Because the processing at this stage
is complicated, an algorithm (Figure 3.6) is provided to aid the reader. The description of this algorithm explains the conditional branches (which are numbered for clarity) and is general in nature. More specific details of the code used to implement this stage can be found in Appendix B.

(1) DO WHILE NOT EOF
(2) IF BUILT-IN FUNCTION
    INVOKE A ROUTINE TO EVALUATE A DESIRED VALUE
ELSE
(3) IF ARITHMETIC EXPRESSION
    INVOKE A ROUTINE TO CALCULATE THE DESIRED RESULT
ELSE
(4) IF LOGICAL EXPRESSION
    CONVERT THE EXPRESSION TO dBASE-II ACCEPTABLE COMMAND STATEMENT
ELSE
(5) IF EXAMPLE ELEMENT
    INVOKE A ROUTINE THAT QUERY THE DATA BASE TO GET DESIRED VALUE
ELSE (CONSTANT ELEMENT)
(6) DATA ENTRY CHECKING
    ENDDO
ENDIF
ENDIF
ENDIF
CONCATENATE THE VALUE FROM ABOVE EXECUTION INTO A STMT FETCH NEXT DATA FIELD ENTRY
ENDDO

Figure 3.6 The Algorithm of Stage 4

(1) DO WHILE NOT EOF

As mentioned in stage 2, a user's query is stored in a database file (.dbf). Therefore, when EOF is encountered, it marks the end of the user's data entry.

(2) IF BUILT-IN FUNCTION

The built-in functions can be: CNT.ALL., MAX.ALL., MIN.ALL., AVG.ALL., SUM.ALL., CNT.UNQ., SUM.UNQ., or AVG.UNQ. After determining which function is desired as an operator, a specific
routine is invoked to obtain the resulting value.

In another case, if an example element or arithmetic operator is specified in conjunction with a built-in function, the processing is not completed until an answer is provided by the data base or the arithmetic expression is evaluated.

(3) IF ARITHMETIC EXPRESSION

A routine is invoked to evaluate the expression. If an example element is specified along with an arithmetic operator, the processing is not finished until a qualified value is obtained by querying the data base.

(4) IF LOGICAL EXPRESSION

The logical operators are '|' (or) and '&' (and). The notation of '|' and '&' is not accepted by the dBASE-II compiler. Therefore, a routine is invoked to convert the expression into a dBASE-II command statement, changing '|' to .OR. and '&' to to '>1000 .AND. < 12000'.

(5) IF EXAMPLE ELEMENT

A routine is invoked in order to query the data base. This routine checks the condition that limits the data first, then it uses the condition to find the qualified data in the data base.

For example, to carry out the query shown in Figure 3.7, the parser finds an example element (in this case, _S) from a row which has operator 'I.' in the row operator field. Then, the parser finds an identical element from another row which has no
row operator in it, in order to obtain the condition associated with the example element (NAME = "HENRY" is the condition in this example). Finally, the parser uses the condition to obtain a qualified value from the database, and then duplicates it for the employee Nancy.

<table>
<thead>
<tr>
<th>EMP</th>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>NANCY</td>
<td>_S</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HENRY</td>
<td>_S</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.7 An Example Query

(6) CONSTANT ELEMENT

A routine is executed to check if the user's entries are correct. For example, if the data type of a field is numeric, then the data entry is checked to see if the user enters a character string instead of digits.

Whenever an entry is parsed, it is checked to see if it violates a syntax rule. If a syntax error is detected, the process is terminated immediately, and an error message is displayed on the screen.

After the execution of one of the above steps, the result of the execution is converted into the dBASE command statement, which performs the given delete or insert operation.

Stage 5. Parse the data field entries

The only operation that is processed in this stage is the update operation. Basically, the method of processing the update operation is the same as that for processing the insert or delete operation. The only difference is that the update operator is a column operator, rather than a row operator. Therefore, the
field that a user wants to update has to be found and converted into a dBASE command statement. Then, the same procedure, as shown in stage 4, is used.

For example, to carry out the query shown Figure 3.8, the parser first finds the entry which begins with an update operator (U.). Then, this entry is converted into a dBASE command statement (which in this case is 'replace salary with 10000'). Finally, the parser processes the entries in the same row (using the method shown in Figure 3.6). These entries become the condition governing the operation (in this case, the condition is \text{NAME = "NANCY"}).

\begin{verbatim}
EMP | NAME | SALARY | MGR |
------------------------
| NANCY | U.10000 |

Figure 3.8 the Example Query
\end{verbatim}

3.3 PARSING A SIMPLE QUERY

This section shows how a query is parsed to perform a desired operation on a database.

\text{ENTER FILE NAME (AND COMMAND IF ANY): EMP}

\begin{verbatim}
| NAME  | SALARY | MGR  | DEPT |
-------------------------------
I. | JONES  | SUM.UNQ._G | | TOY |
|    | _G    |        |     |

Figure 3.9 A Simple Example
\end{verbatim}

In the example of Figure 3.9, the field name entry is parsed by stage 1 to see if a table name is entered accompanied by an
operator. Since there is no operator specified with the table
name EMP, a table skeleton is displayed. The information, entered
by the user, in the general skeleton table is then stored by
stage 2 into the 'sfile' data base file. Figure 3.10 shows how
this information is stored.

<table>
<thead>
<tr>
<th>REC</th>
<th>FIELD:NAME</th>
<th>ROW1</th>
<th>ROW2</th>
<th>ROW3</th>
<th>TYPE</th>
<th>LEN</th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EMP</td>
<td>I.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NAME</td>
<td>JONES</td>
<td></td>
<td></td>
<td>C</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SALARY</td>
<td>SUM.UNQ._G</td>
<td>_G</td>
<td></td>
<td>N</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>MGR</td>
<td></td>
<td>TOY</td>
<td></td>
<td>C</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>DEPT</td>
<td></td>
<td></td>
<td></td>
<td>C</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.10 Sfile1.dbf

After the information in the general table is stored, the
field name of the EMP table is parsed by stage 3. Since the field
names are entered correctly and no insert operator is present,
the process goes on to the next stage to parse the row operator
to entries. When the row operator entries are parsed, the insert
operation is found in the first row. This means the information
in the data fields provide the actual data to be added to the EMP
table. Therefore, the information in the first row is then parsed
column by column by stage 4. The NAME column contains a constant
element 'JONES' which is checked to ensure that its data type
matches its declared type. Since the data is entered correctly,
the parser then checks the SALARY column. The SALARY column has
a built-in function accompanied by an example element '_G'. The
parser first finds another row which has no row operator and has
an identical example element in the SALARY column (which, in this
case, is the second row). Next, the selection criteria specified
with the example element in the second row is used to determine
the desired value for the built-in operation. In this example, the sum of salary without duplicates in the toy department is calculated. The MGR and DEPT columns must be parsed after the SALARY column to obtain the desired result. Since the columns in the first row are empty, the parsing is terminated with the information JONES and the value resulting from evaluating the built-in function is inserted into the EMP table. Since only one insert, delete, or update operation can appear in a query, stage 5, which is used to parse an update operation, does not occur.

3.4 LIMITATIONS

There are several limitations that have been imposed during the development of QBE on a microcomputer. These are listed below:

(1) This version of QBE runs on a personal computer. Since personal computers are limited in their memory capacity and in the access time to the storage devices, it takes QBE a long time to complete a query.

(2) The procedural language provided by dBASE-II was intended for the storage and execution of some frequently used command sequences, not for complex programming. Therefore, because of the restrictions of this language, it is impossible to program a scrollable screen.

(3) Because of the last reason, the users' interface tables are designed to be fixed skeleton tables. For example, a general table has six columns and each column can contain up to ten characters. This means that only six fields can be contained in
one table.

(4) Since a user can only open two files simultaneously in dBASE-II, QBE allows a user to use at most two general tables for a query.

(5) dBASE-II is a relational data base management system which allows duplicate tuples in a data base. Therefore, QBE does not require users to specify the key for a relation.
Chapter 4
Conclusion

This report has presented QBE language and its facilities. The reasons why QBE is a valuable tool for the end users were also discussed. The main purpose of this project was to implement QBE on a microprocessor. This provides students, who are learning about QBE, with an actual system to work with. By using this system, they can use the concepts from the text book to formulate their own queries and learn more about QBE.

The microprocessor version of QBE provides all of the facilities of QBE. These are insert, delete, retrieve, and update. There are some defects which prevent the system from being as powerful as it could be. These defects and the possible improvements are now discussed.

The first thing that may bother users is the long execution time of an operation. Normally, it takes roughly five minutes to complete a simple operation, and between eight and ten minutes to complete a complex operation. This delay is due to the fact that dBASE-II is interpreted. This problem may be reduced by using the product, dB compiler, sold by WordTech Systems, Inc.. Such a compiler allows the legal translation of dBASE-II programs into executable code. A compiled program runs faster in all areas, including calculations, logical operations and file access. However, some features of dBASE-II are not supported by this compiler. For example, a user may not macro-substitute a command file name (i.e., in DO &FILENAME) or use commands such as 'SET
TALK ON/OFF', 'CREATE <file>'. Therefore, certain commands used to implement QBE would have to be changed in order to use the dB compiler.

The microprocessor version of QBE supplies users with general tables, result tables, and condition boxes, but the size of these tables is fixed. The reason that the microprocessor version of QBE does not supply dynamic table sizes is that dBASE-II does not provide as many facilities to control the screen as other programming languages. Therefore, it is impossible to program a scrollable screen with dBASE's limited facilities.

Two methods may be used to improve the screen handling problem. First, a portion of the user's interface could be written in assembly code instead of dBASE-II. The dBASE-II system supports assembly interface calls by using commands such as peek, poke, load, and call. These commands allow the process to jump from the execution of dBASE-II to the execution of assembly code. The process returns to dBASE-II and resumes the execution of dBASE-II when the end of assembly code is reached.

A second method to improve the system response is to utilize dBASE-III[ASH84] rather than dBASE-II to improve screen handling. dBASE-III is also primarily a data base management system like dBASE-II, but dBASE-III provides more facilities than dBASE-II. One of its more useful facilities is that dBASE-III can interface with other software. This is done by using a 'run' command which executes a designated .COM or .EXE file. The run command can be embedded in a dBASE-III program, and it causes the execution of a designated file and returns control to dBASE-III only after that file has completed execution. The run facility allows programmers
to employ a more powerful programming language, such as Turbo Pascal or Basic, in conjunction with a data base management system. Thus QBE could be implemented in a more efficient way.

Since QBE is a user friendly language, the manner in which a user's query is specified should be as flexible as possible. Therefore, it is very difficult to limit the syntactic and semantic specification of QBE. It is possible that there are some features of QBE which are unintentionally not implemented in this version of QBE. In order to allow future extension of QBE's ability, the microprocessor version of QBE provides a facility to store all users' queries which do not work along with their accompanying error messages. This information is stored in a file named 'review'. The future implementor may look at the information provided in the review file and improve a future version of QBE by including features which were not implemented in the current version of QBE.
REFERENCES


[CHE86] Chen, Li-Ling "A Microcomputer implementation of Query-by-Example", a Master Report, Kansas State University, 1986


APPENDICIES
Appendix A
User Manual

This manual uses a tutorial approach to lead the user through the various operations and options available in this version of QBE. To use the QBE system, two disks are required: a system disk and a personal disk which has been formatted in advance. The first time QBE is used, the file 'colfile.dbf' must be copied from the system disk to the user's disk. Once the colfile.dbf file is placed on a user's disk, QBE is ready to use. To start the system, put the system disk in drive A and a data disk in drive B, and type 'DBASE'. When the dBASE-II prompt appears on the screen, type 'DO QBE'. The system then prompts for the current day. Once the current day is entered, the user may start to formulate query. The following examples and their answers show how to create a table, and how to change its contents.

(1) Creating a table

The following steps show how to create a table named EMP, where each record in EMP contains these four fields:

Name : which contains up to fifteen characters,
Salary: which contains up to ten digits,
Mgr : which contains up to fifteen characters,
Dept : which contains up to ten characters.

First, QBE prompts the user with the message "ENTER FILE NAME (AND COMMAND IF ANY):". To create the table named EMP, the user has to enter 'I. EMP I.'. I. is the insert operator, which must appear before and after the name (EMP) of the table to be
must appear before and after the name (EMP) of the table to be created. The table name must start with a letter and may be up to eight alphanumeric characters in length.

After the command has been executed, a table is displayed as shown below.

```
EMP | FIELD:NAME | FIELD:TYPE | FIELD: LEN | FIELD:DEC
----|------------|------------|------------|------------
    |            |            |            |            |
```

The user is then asked to fill in the field names that are going to be contained in the EMP table, along with the information about each field. A field name can be up to ten alphanumeric characters long. The data type of the field is specified by a single letter, C for character, and N for numeric. A field length can be of any length up to fifty-eight characters. If the field is numeric, the number of places after the decimal point may be specified (QBE defaults to a value of zero). The decimal point also takes up a space. An example of the record description follows:

```
EMP | FIELD:NAME | FIELD:TYPE | FIELD: LEN | FIELD:DEC
----|------------|------------|------------|------------
    | NAME       | C          | 10         |            |
    | SALARY     | N          | 10         |            |
    | MGR        | C          | 10         |            |
    | DEPT       | C          | 10         |            |
```

After the user has finished entering the information, <return> must be pressed twice. Then, QBE responds with a message "DO YOU WANT TO MODIFY ANYTHING?". If all of the information has been entered correctly, then N should be entered, otherwise Y should be entered. QBE prompts with messages that lead the user through any desired modification.
QBE is case sensitive, therefore all of data for the database has to be entered in upper case, otherwise the query will not be executed as the user expects.

(2) Updating a table name

In order to update a table name from EMP to EMP1, simply enter 'U. EMP' in response to the system prompt "ENTER FILE NAME (AND COMMAND IF ANY)". The system prompts the user for a new name. If the new name that the user enters already exists in the database, a message is displayed on the screen.

(3) Deleting an existing table

To delete the existing table EMP, enter 'D.EMP' in response to the system prompt. To prevent losing a table accidentally, a message is displayed to ask if the user really wants to drop the table. If the answer is positive, the deletion takes place, otherwise QBE aborts the request.

(4) Changing the content of the table

In order to change the information in an existing table, the user has to go through the following procedures. When the system prompts for the table name "ENTER FILE NAME (AND COMMAND IF ANY) :

: ", the table name to be worked with has to be supplied. The system then responds with a general table. The user can use the table to formulate the action desired from the system (this is covered, in greater detail, later). After the user finishes filling in the table, the system asks if another table is needed. Another general table, result table, or condition box may be chosen then. QBE provides two general tables, one result table,
and as many condition boxes as necessary. If no additional table is needed, then enter 'N'. The system then starts processing the user's request. The user's request may be terminated if there are any syntax errors. If the user's request completes, a message "DO YOU WANT TO QUIT OR CONTINUE? " is displayed on the screen. The user should enter 'Y' to continue with the user session, or 'N' to quit the QBE system.

The following examples illustrate ways to specify an insert, update, and delete operations. Let us assume that there are two existing tables, named EMP and SALES, with the following information in the data base. All of the examples operates upon these two tables. Also, all of the queries are numbered for clarity.

<table>
<thead>
<tr>
<th>EMP</th>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JONES</td>
<td>7000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td></td>
<td>ANDERSON</td>
<td>5000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>MORGAN</td>
<td>10000</td>
<td>LEE</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>LEWIS</td>
<td>12000</td>
<td>LONG</td>
<td>STATIONERY</td>
</tr>
<tr>
<td></td>
<td>NELSON</td>
<td>6000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>HOFFMAN</td>
<td>14000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>LONG</td>
<td>7000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>MURPHY</td>
<td>8000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td></td>
<td>SMITH</td>
<td>12000</td>
<td>HOFFMAN</td>
<td>STATIONERY</td>
</tr>
<tr>
<td></td>
<td>HENRY</td>
<td>9000</td>
<td>SMITH</td>
<td>TOY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SALES</th>
<th>DEPT</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STATIONERY</td>
<td>DISH</td>
</tr>
<tr>
<td></td>
<td>HOUSEHOLD</td>
<td>PEN</td>
</tr>
<tr>
<td></td>
<td>STATIONERY</td>
<td>PENCIL</td>
</tr>
<tr>
<td></td>
<td>COSMETICS</td>
<td>LIPSTICK</td>
</tr>
<tr>
<td></td>
<td>TOY</td>
<td>PEN</td>
</tr>
<tr>
<td></td>
<td>TOY</td>
<td>PENCIL</td>
</tr>
<tr>
<td></td>
<td>HARDWARE</td>
<td>INK</td>
</tr>
<tr>
<td></td>
<td>STATIONERY</td>
<td>PEN</td>
</tr>
<tr>
<td></td>
<td>COSMETICS</td>
<td>PERFUME</td>
</tr>
</tbody>
</table>
(4.1) Insert Operation Examples

Query 1 is an example of a simple Insertion. In a simple insertion, a user adds a whole row to a table. A user may specify a unique value as the key of the row. The key uniquely identifies the row in order to make future querying easier.

Query 1: Insert a new employee into the EMP table; The employee is named BETTY and her salary is $14,000; her manager and department are to be added later (see query 7).

Note: The field names that are to be worked with must be supplied. In this case, the field names are NAME, and SALARY. The MGR and DEPT fields may also be filled in or left blank.

ENTER FILE NAME (AND COMMAND IF ANY): EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>-------</td>
<td>--------</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>I.</td>
<td>BETTY</td>
<td>14000</td>
<td></td>
</tr>
<tr>
<td>I.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this case, Betty is the key of the row. This is based on the assumption that there are no duplicate names in the database. The key may be used to retrieve, update, or delete this row.

*************************************************************************
* QUERY 1. ANSWER       *
*************************************************************************
<table>
<thead>
<tr>
<th>EMP</th>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JONES</td>
<td>7000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td></td>
<td>ANDERSON</td>
<td>5000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>MORGAN</td>
<td>10000</td>
<td>LEE</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>LEWIS</td>
<td>12000</td>
<td>LONG</td>
<td>STATIONERY</td>
</tr>
<tr>
<td></td>
<td>NELSON</td>
<td>6000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>HOFFMAN</td>
<td>14000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>LONG</td>
<td>7000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>MURPHY</td>
<td>8000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td></td>
<td>SMITH</td>
<td>12000</td>
<td>HOFFMAN</td>
<td>STATIONERY</td>
</tr>
<tr>
<td></td>
<td>HENRY</td>
<td>9000</td>
<td>SMITH</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>BETTY</td>
<td>14000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The length of each column in a table is restricted to ten characters, but QBE provides an easy way to insert data whose length is longer than ten characters.

Query 1.1: Insert a tuple in an easy way.

For example, if the user wants to insert a new employee, Steven Johnson, and his manager is Jimmy McDonald, and his salary is $15,000. An operator(C.) may be entered in the first column of the data field of the table. This is shown in the following query.

ENTER FILE NAME (AND COMMAND IF ANY): EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

QBE responds with a table to accept the data.

NAME: Steven Johnson

SALARY: 15000

MGR: Jimmy McDonald

DEPT:
Query 2 illustrates insertion using built-in functions. There are eight built-in functions provided by QBE:

\begin{itemize}
\item CNT.ALL. : counts the number of items,
\item SUM.ALL. : adds the items together,
\item AVG.ALL. : finds the average of the items,
\item MAX.ALL. : find the largest value,
\item MIN.ALL. : find the smallest value,
\item CNT.UNQ. : counts the number of items with the exception of duplicates,
\item SUM.UNQ. : add the items together without including duplicates,
\item AVG.UNQ. : evaluate the average of the items without duplicates.
\end{itemize}

Query 2: Insert a new employee into the EMP table; the employee's name is SCOTT and his salary is the average salary in the EMP table.

\begin{verbatim}
ENTER FILE NAME (AND COMMAND IF ANY): EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCOTT</td>
<td>AVG.ALL.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

***************************
* QUERY 2. ANSWER *
***************************
\end{verbatim}
Query 4 and 5 are examples of query-dependent insertion. With a query-dependent insertion, QBE must first query the database to obtain desired information, when it has found the required information, the insertion take place. If QBE cannot find the required information, a message "THERE ARE NO RECORDS WHICH MEET THE CONDITION -- " is displayed on the screen. The query-dependent insertion is a single record insertion. It only adds one tuple into the data base at a time. If there is more than one data item which fulfills the condition, QBE always chooses the first one which occurs.

Query 4: Insert a new employee into the EMP table; the employee is named Mary and her salary is the same as Lewis's salary; her manager and department are to be added later.

ENTER FILE NAME (AND COMMAND IF ANY): EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARY</td>
<td>_S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEWIS</td>
<td>_S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the preceding example, the example element '_S' is used to link two rows. This means if Lewis's salary is the value, _S, then Mary's salary takes on the same value. QBE must first find
Lewis's salary (in this case, $12000) and then duplicate it for the new employee, Mary.

<table>
<thead>
<tr>
<th>EMP</th>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>JONES</td>
<td>7000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
<td></td>
</tr>
<tr>
<td>ANDERSON</td>
<td>5000</td>
<td>MURPHY</td>
<td>TOY</td>
<td></td>
</tr>
<tr>
<td>MORGAN</td>
<td>10000</td>
<td>LEE</td>
<td>COSMETICS</td>
<td></td>
</tr>
<tr>
<td>LEWIS</td>
<td>12000</td>
<td>LONG</td>
<td>STATIONERY</td>
<td></td>
</tr>
<tr>
<td>NELSON</td>
<td>6000</td>
<td>MURPHY</td>
<td>TOY</td>
<td></td>
</tr>
<tr>
<td>HOFFMAN</td>
<td>14000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
<td></td>
</tr>
<tr>
<td>LONG</td>
<td>7000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
<td></td>
</tr>
<tr>
<td>MURPHY</td>
<td>8000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
<td></td>
</tr>
<tr>
<td>SMITH</td>
<td>12000</td>
<td>HOFFMAN</td>
<td>STATIONERY</td>
<td></td>
</tr>
<tr>
<td>HENRY</td>
<td>9000</td>
<td>SMITH</td>
<td>TOY</td>
<td></td>
</tr>
<tr>
<td>MARY</td>
<td>12000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Because of the limited length of each column in a general table, a condition box may be used to enter a constant or example element which is more than ten characters long. To illustrate this, two equivalent ways of performing the same query are given.

ENTER FILE NAME (AND COMMAND IF ANY): EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>JONES</td>
<td>_S</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>_G</td>
<td></td>
<td>TOY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>_S = SUM.UNQ._G</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

The query above is the same as:
ENTER FILE NAME (AND COMMAND IF ANY): EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>JONES</td>
<td>SUM.UNQ._G</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_G</td>
<td></td>
<td>TOY</td>
<td></td>
</tr>
</tbody>
</table>

The QBE system provides you with a set of arithmetic operators: '+' (addition), '-' (subtraction), '*' (multiplication), and '/' (division).

Query 5: Insert a new employee into the EMP table; the employee is named Richard and his salary is double of that Henry's salary, and his manager is Jones, and he works in the toy department.

ENTER FILE NAME (AND COMMAND IF ANY): EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>RICHARD</td>
<td>_S * 2</td>
<td>JONES</td>
<td>TOY</td>
</tr>
<tr>
<td>HENRY</td>
<td>_S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

******************************************************************************
* QUERY 5. ANSWER *
******************************************************************************

EMP | NAME | SALARY | MGR | DEPT            |
-----|------|--------|-----|----------------|
| JONES | 7000 | SMITH  | HOUSEHOLD |
| ANDERSON | 5000 | MURPHY | TOY     |
| MORGAN | 10000 | LEE    | COSMETICS |
| LEWIS  | 12000 | LONG   | STATIONERY|
| NELSON | 6000  | MURPHY | TOY     |
| HOFFMAN | 14000 | MORGAN | COSMETICS |
| LONG   | 7000  | MORGAN | COSMETICS |
| MURPHY | 8000  | SMITH  | HOUSEHOLD |
| SMITH  | 12000 | HOFFMAN| STATIONERY|
| HENRY  | 9000  | SMITH  | TOY     |
| RICHARD | 18000 | JONES  | TOY     |
Query 6: Table Expansion

More fields can be added to an existing table by using the insert operator. For example, to add one more field into the EMP table, enter an insert operator(I.) followed by the new field name. The system responds with a table to accept the information about the new field.

ENTER FILE NAME (AND COMMAND IF ANY): EMP

<table>
<thead>
<tr>
<th>I. PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>-----------</td>
</tr>
</tbody>
</table>

The system responds with a table, which the user fills in, as follows:

<table>
<thead>
<tr>
<th>EMP</th>
<th>FIELD:NAME</th>
<th>FIELD:TYPE</th>
<th>FIELD: LEN</th>
<th>FIELD: DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHONE</td>
<td>C</td>
<td>12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After the execution of the operation, the EMP table contains five fields: name, salary, mgr, dept, and phone. The data type of the phone field is a character string which may be up to twelve characters. Remember, one table can only contain a maximum of six fields. If an attempt is made to add a seventh field to a table, the request is rejected and an error message is displayed on the screen.

Query 7 shows a multiple insertion using multiple tables.

Query 7: Copy all the data in the EMP table to another EMP1 table.
To copy the information from the EMP table into the EMP1 table. Be sure that the structure of EMP1 table (the field name and the data type) is declared, and is identical to that of the EMP table. Notice that each field requires a different example element name and corresponding fields in the EMP and EMP1 table need identical example element names.

<table>
<thead>
<tr>
<th>EMP1</th>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>JONES</td>
<td>7000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
<td></td>
</tr>
<tr>
<td>ANDERSON</td>
<td>5000</td>
<td>MURPHY</td>
<td>TOY</td>
<td></td>
</tr>
<tr>
<td>MORGAN</td>
<td>10000</td>
<td>LEE</td>
<td>COSMETICS</td>
<td></td>
</tr>
<tr>
<td>LEWIS</td>
<td>12000</td>
<td>LONG</td>
<td>STATIONERY</td>
<td></td>
</tr>
<tr>
<td>NELSON</td>
<td>6000</td>
<td>MURPHY</td>
<td>TOY</td>
<td></td>
</tr>
<tr>
<td>HOFFMAN</td>
<td>14000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
<td></td>
</tr>
<tr>
<td>LONG</td>
<td>7000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
<td></td>
</tr>
<tr>
<td>MURPHY</td>
<td>8000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
<td></td>
</tr>
<tr>
<td>SMITH</td>
<td>12000</td>
<td>HOFFMAN</td>
<td>STATIONERY</td>
<td></td>
</tr>
<tr>
<td>HENRY</td>
<td>9000</td>
<td>SMITH</td>
<td>TOY</td>
<td></td>
</tr>
</tbody>
</table>

(4.2) Update Operations Examples

The update operator(U.) is used to modify the information in
the data base. It can be used to replace null values in a certain record, e.g., to add in Betty's manager or department, which are null values after the execution of Query 1. Note: the update operator (U.) has to be placed in the column that is to be updated, it cannot be used as a row operator. Also, only one column at a time can be updated.

Query 8: Update Betty's manager to Smith.

**ENTER FILE NAME (AND COMMAND IF ANY): EMP**

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>BETTY</td>
<td></td>
<td>U.SMITH</td>
<td></td>
</tr>
</tbody>
</table>

I trust you have found this to your satisfaction.

**ENTER FILE NAME (AND COMMAND IF ANY): EMP**

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>JONES</td>
<td>7000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td>ANDERSON</td>
<td>5000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td>MORGAN</td>
<td>10000</td>
<td>LEE</td>
<td>COSMETICS</td>
</tr>
<tr>
<td>LEWIS</td>
<td>12000</td>
<td>LONG</td>
<td>STATIONERY</td>
</tr>
<tr>
<td>NELSON</td>
<td>6000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td>HOFFMAN</td>
<td>14000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td>LONG</td>
<td>7000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td>MURPHY</td>
<td>8000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td>SMITH</td>
<td>12000</td>
<td>HOFFMAN</td>
<td>STATIONERY</td>
</tr>
<tr>
<td>BETTY</td>
<td>14000</td>
<td>SMITH</td>
<td></td>
</tr>
</tbody>
</table>

Query 9: Update Murphy's salary to 13,000

**ENTER FILE NAME (AND COMMAND IF ANY): EMP**

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MURPHY</td>
<td>U.13000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMP1</td>
<td>NAME</td>
<td>SALARY</td>
<td>MGR</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>JONES</td>
<td>7000</td>
<td>SMITH</td>
<td></td>
</tr>
<tr>
<td>ANDERSON</td>
<td>5000</td>
<td>MURPHY</td>
<td></td>
</tr>
<tr>
<td>MORGAN</td>
<td>10000</td>
<td>LEE</td>
<td></td>
</tr>
<tr>
<td>LEWIS</td>
<td>12000</td>
<td>LONG</td>
<td></td>
</tr>
<tr>
<td>NELSON</td>
<td>6000</td>
<td>MURPHY</td>
<td></td>
</tr>
<tr>
<td>HOFFMAN</td>
<td>14000</td>
<td>MORGAN</td>
<td></td>
</tr>
<tr>
<td>LONG</td>
<td>7000</td>
<td>MORGAN</td>
<td></td>
</tr>
<tr>
<td>MURPHY</td>
<td>13000</td>
<td>SMITH</td>
<td></td>
</tr>
<tr>
<td>SMITH</td>
<td>12000</td>
<td>HOFFMAN</td>
<td></td>
</tr>
</tbody>
</table>

Queries 10-12 are examples of query-dependent updates.

Query 10: Makes James the manager of all employees who work in a department that sells pens.

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>U.JAMES</td>
<td>_D</td>
</tr>
</tbody>
</table>
```

ENTER FILE NAME (AND COMMAND IF ANY): SALES

```
<table>
<thead>
<tr>
<th>DEPT</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>_D</td>
<td>PEN</td>
</tr>
</tbody>
</table>
```

Note: The field name and the data type which are linked by the example element must be the same. In this case, the data type of dept in the EMP table is the same as that of dept in SALES table.
EMP1 | NAME    | SALARY | MGR   | DEPT    |
-----|---------|--------|-------|---------|
    | JONES   | 7000   | JAMES | HOUSEHOLD |
    | ANDERSON| 5000   | JAMES | TOY     |
    | MORGAN  | 10000  | LEE   | COSMETICS |
    | LEWIS   | 12000  | LONG  | STATIONERY |
    | NELSON  | 6000   | JAMES | TOY     |
    | HOFFMAN | 14000  | MORGAN| COSMETICS |
    | LONG    | 7000   | MORGAN| COSMETICS |
    | MURPHY  | 13000  | JAMES | HOUSEHOLD |
    | SMITH   | 12000  | JAMES | STATIONERY |

Query 11: Double the salaries of the employees in the toy department.

ENTER FILE NAME (AND COMMAND IF ANY): EMP

| NAME | SALARY | MGR | DEPT |
-----|--------|-----|------|
    | _JONES | U._S*2 |       |
    | _JONES | _S  | TOY  |

Another way of constructing this same query is now given.

ENTER FILE NAME (AND COMMAND IF ANY): EMP

| NAME | SALARY | MGR | DEPT |
-----|--------|-----|------|
    | U._S*2 | _S  | TOY  |

The second row retrieves the old salary value, then the salary expression in the first row determines the value after the update. This operation can be paraphrased as follows: retrieve a record that has TOY as a department, find the salary _S and update that salary to 2 times _S.

*****************************************************************************
* QUERY 11. ANSWER *
*****************************************************************************

59
<table>
<thead>
<tr>
<th>EMP1</th>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JONES</td>
<td>7000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td></td>
<td>ANDERSON</td>
<td>10000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>MORGAN</td>
<td>10000</td>
<td>LEE</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>LEWIS</td>
<td>12000</td>
<td>LONG</td>
<td>STATIONERY</td>
</tr>
<tr>
<td></td>
<td>NELSON</td>
<td>12000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>HOFFMAN</td>
<td>14000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>LONG</td>
<td>7000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>MURPHY</td>
<td>13000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td></td>
<td>SMITH</td>
<td>12000</td>
<td>HOFFMAN</td>
<td>STATIONERY</td>
</tr>
</tbody>
</table>

Query 12: Make Harold's salary the same as the highest salary in the cosmetics department.

ENTER FILE NAME (AND COMMAND IF ANY): EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEWIS</td>
<td>_G</td>
<td>_S</td>
<td>COSMETICS</td>
</tr>
</tbody>
</table>

**CONDITIONS**

|-- G = MAX.ALL._S

---------------------------

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

* QUERY 12. ANSWER *

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

<table>
<thead>
<tr>
<th>EMP1</th>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JONES</td>
<td>7000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td></td>
<td>ANDERSON</td>
<td>10000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>MORGAN</td>
<td>10000</td>
<td>LEE</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>LEWIS</td>
<td>14000</td>
<td>LONG</td>
<td>STATIONERY</td>
</tr>
<tr>
<td></td>
<td>NELSON</td>
<td>6000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>HOFFMAN</td>
<td>14000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>LONG</td>
<td>7000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>MURPHY</td>
<td>13000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td></td>
<td>SMITH</td>
<td>12000</td>
<td>HOFFMAN</td>
<td>STATIONERY</td>
</tr>
</tbody>
</table>
(4.3) Delete Operations Examples

Queries 13 and 14 are examples of simple deletions. A simple deletion is similar to simple insertion in that it can operate on a whole row at one time.

Query 13: Delete all information about Jones from the EMP table.

ENTER FILE NAME (AND COMMAND IF ANY): EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>D. JONES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The delete operation deletes all the information that satisfies the condition specified. For example, if there is more than one employee named JONES, they are all deleted from the database.

***************
# QUERY 13. ANSWER #
***************

<table>
<thead>
<tr>
<th>EMP1</th>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ANDERSON</td>
<td>5000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>MORGAN</td>
<td>10000</td>
<td>LEE</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>LEWIS</td>
<td>12000</td>
<td>LONG</td>
<td>STATIONERY</td>
</tr>
<tr>
<td></td>
<td>NELSON</td>
<td>6000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>HOFFMAN</td>
<td>14000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>LONG</td>
<td>7000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>MURPHY</td>
<td>13000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td></td>
<td>SMITH</td>
<td>12000</td>
<td>HOFFMAN</td>
<td>STATIONERY</td>
</tr>
</tbody>
</table>

Query 14: Delete all information about all employees in the toy department.
**QUERY 14. ANSWER**

<table>
<thead>
<tr>
<th>EMP</th>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>JONES</td>
<td>7000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
<td></td>
</tr>
<tr>
<td>MORGAN</td>
<td>10000</td>
<td>LEE</td>
<td>COSMETICS</td>
<td></td>
</tr>
<tr>
<td>LEWIS</td>
<td>12000</td>
<td>LONG</td>
<td>STATIONERY</td>
<td></td>
</tr>
<tr>
<td>HOFFMAN</td>
<td>14000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
<td></td>
</tr>
<tr>
<td>LONG</td>
<td>7000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
<td></td>
</tr>
<tr>
<td>MURPHY</td>
<td>8000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
<td></td>
</tr>
<tr>
<td>SMITH</td>
<td>12000</td>
<td>HOFFMAN</td>
<td>STATIONERY</td>
<td></td>
</tr>
</tbody>
</table>

Queries 15 and 16 illustrate how to perform query-dependent deletion. With a query-dependent delete, the deletion cannot take place until a query has been answered.

In the example of query 15, a relational operator is used. QBE allows the user to specify the condition using relational operators, such as '>', '<' (less than), '>=' (greater than or equal to), '<=' (less than or equal to), '<>' (not equal to), and '=' (equal to).

Query 15: Delete all employees who earn more than Morgan in the cosmetics department.

**ENTER FILE NAME (AND COMMAND IF ANY): EMP**

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.</td>
<td>&gt; _S</td>
<td></td>
<td>COSMETICS</td>
</tr>
<tr>
<td>MORGAN</td>
<td>_S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Query 15. Answer

<table>
<thead>
<tr>
<th>EMP1</th>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JONES</td>
<td>7000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td></td>
<td>ANDERSON</td>
<td>5000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>MORGAN</td>
<td>10000</td>
<td>LEE</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>LEWIS</td>
<td>12000</td>
<td>LONG</td>
<td>STATIONERY</td>
</tr>
<tr>
<td></td>
<td>NELSON</td>
<td>6000</td>
<td>MURPHY</td>
<td>TOY</td>
</tr>
<tr>
<td></td>
<td>LONG</td>
<td>7000</td>
<td>MORGAN</td>
<td>COSMETICS</td>
</tr>
<tr>
<td></td>
<td>MURPHY</td>
<td>13000</td>
<td>SMITH</td>
<td>HOUSEHOLD</td>
</tr>
<tr>
<td></td>
<td>SMITH</td>
<td>12000</td>
<td>HOFFMAN</td>
<td>STATIONERY</td>
</tr>
</tbody>
</table>

Query 16: Delete all employees who earn less than $10,000 and work in a department that sells lipstick.

**ENTER FILE NAME (AND COMMAND IF ANY): EMP**

<table>
<thead>
<tr>
<th></th>
<th>NAME</th>
<th>SALARY</th>
<th>MGR</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.</td>
<td>&gt; 10000</td>
<td></td>
<td>_D</td>
<td></td>
</tr>
</tbody>
</table>

**ENTER FILE NAME (AND COMMAND IF ANY): SALES**

<table>
<thead>
<tr>
<th>DEPT</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>_D</td>
<td>LIPSTICK</td>
</tr>
</tbody>
</table>

### Query 16. Answer
EMP | NAME  | SALARY | MGR  | DEPT  
-------------------------------
| JONES  | 7000  | SMITH | HOUSEHOLD |
| ANDERSON | 5000  | MURPHY | TOY |
| MORGAN | 10000 | LEE  | COSMETICS |
| LEWIS  | 12000 | LONG | STATIONERY |
| NELSON | 6000  | MURPHY | TOY |
| HOFFMAN | 14000 | MORGAN | COSMETICS |
| LONG   | 7000  | MORGAN | COSMETICS |
| MURPHY | 8000  | SMITH | HOUSEHOLD |
| SMITH  | 12000 | HOFFMAN | STATIONERY |
| HENRY  | 9000  | SMITH | TOY |

Query 17: Delete all existing rows in the EMP table.

| NAME  | SALARY | MGR  | DEPT  |
-------------------------------
| D.    |       |      |      |

***************
* QUERY 17. ANSWER - *
***************

EMP1 | NAME  | SALARY | MGR  | DEPT  
-------------------------------
Appendix B
Logging Queries

The 'REVIEW' file permits any users' queries that did not work to be saved, along with their associated error message. This file provides the QBE administrator with information about the features which were expected by the users, but which were not implemented in this version of QBE. This section discusses how to access the 'REVIEW' file and how user's queries are stored in the 'REVIEW' file.

The 'REVIEW' file is stored on the user's disk. You can use the 'edit' facility of dBASE-II to view the file. For example, the query shown below is returned with an error message '_S NOT FOUND'.

ENTER FILE NAME (AND COMMAND IF ANY) : EMP

<table>
<thead>
<tr>
<th>NAME</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>D.</td>
<td>&gt;_S</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The query is stored in the 'REVIEW' file by using three records. If you issue the command dBASE-II 'EDIT1'(assuming this is the first query), you will find the query is stored in the following manner:

RECORD 00001
FLD:NAME : EMP
ROW1 : D.
ROW2 :
ROW3 :
MESSAGES : _S NOT FOUND
RECORD 00002
FLD:NAME : NAME
ROW1 :
ROW2 :
ROW3 :
MESSAGES :
RECORD 00003
FLD:NAME : SALARY
ROW1 : >_S
ROW2 :
ROW3 :
MESSAGES : END OF QUERY

The first record of a query in the 'REVIEW' file is used to store the table name, the row operator, and the error message. In this case, FLD:NAME in the first record is the table to be dealt with, and D. is the row operator corresponding to the table skeleton. The error message is always stored in the first record of a query. For the remaining records of a query, each record corresponds to a column in the skeleton table. In this case, the second record and third record represent the two columns in the skeleton table. Record 00002 contains column heading NAME. Since there is no data entry for the NAME column, the field row1, row2, and row3 are blank. Record 00003 contains the data entry ' > _S' in row1. To separate queries, the delimiter 'END OF QUERY' is placed in the message field of the last record of each query.
Appendix C

The Comparison of Two Versions of QBE

This appendix lists the differences between IBM's version of QBE and the microcomputer version of QBE:

(1) The message 'PLEASE ENTER THE FILE NAME (AND COMMAND IF ANY):' is displayed to prompt the user to enter the table name to be dealt with. Thus, in the microcomputer version, the name of a general table appears before the skeleton table. In IBM's version, the table name entry is the first column of the first row of the skeleton table.

(2) The size of the general tables and result tables is fixed in the microcomputer version of QBE. Therefore, each table can contain six columns where each column is twelve characters wide. In IBM's version, any table can contain up to ninety-nine columns and each column can be up to thirty-two characters wide.

(3) The name of a table is up to eight characters in length and the field names of a table cannot be more than ten characters in length for the microcomputer version of QBE. In IBM's version, each table name and field name can be up to thirty-two characters in length.

(4) In the microcomputer version of QBE, the maximum length of a data item can be up to fifty-eight characters. While, in IBM's version, a field can be from one to three thousand and two hundred characters. This is due to the fact that the microcomputer version of QBE does not provide a scrollable screen while IBM's version does.
(5) In the microcomputer version of QBE, the data items in a record are displayed in a linear form. While IBM's version displays them in a tabular form.

(6) In IBM's QBE, the users can use as many skeleton tables as they need to express a query. The microcomputer version of QBE only allows the user to employ two general tables, one result table, and as many condition boxes as needed for a query.

(7) When the user formulates a query, the names of the attributes of the table to be dealt with have to be specified by the user in the microcomputer version of QBE. However, in IBM version, they can be generated by the system.

(8) To create a new table with the IBM's QBE, the user has to specify information like key, domain, type, length and inversion for each field to be contained in the new table. In the microcomputer version of QBE, the user only specifies the field type, the field length, and the decimal place.

(9) In IBM's QBE, each table has a primary key. The key of a table is used to avoid duplicate data in a data base. No key is specified in the microcomputer version, therefore, redundant tuples are allowed.

(10) The update operator is required to be a column operator in the microcomputer version of QBE, but it can be either column operator or a row operator in the IBM version of QBE.
Appendix D

The Hierarchical Diagrams for the QBE system

This appendix presents the hierarchical diagrams of the modules that implement the update, delete, and insert operation. Each rectangle represents a named module. If the name of a module is prefixed by an asterisk, this means that the named module has its structure, in turn, detailed in another hierarchical diagram. Since this implementation is part of a joint project, some of the modules are connected with the portion that were implemented by Li-Ling Chen. These modules are RUN, PARSE1, and AVTABLE. The reader may refer to her report [CHE86] to see how these modules are interconnected to the procedures developed by her.
**ARThDep command file**

This program retrieves the information from the database and returns the value to its calling program. This program is called by ARThOPE command file.

```plaintext
STORE T TO GO
DO WHILE GO
  IF K > LEN(COLUMN)
    STORE F TO GO
    LOOP
  ELSE
    IF $(COLUMN,K,1) = '*' .OR. $(COLUMN,K,1) = '-' .OR. $(COLUMN,K,1) = '/' .OR. $(COLUMN,K,1) = '+' .OR. $(COLUMN,K,1) = '(' .OR. $(COLUMN,K,1) = ')
      STORE F TO GO
      LOOP
    ELSE
      STORE K+1 TO K
    ENDDIF
  ENDDIF (* k > len(column) *)
ENDDO (* GO *)
STORE K TO STORAGE
STORE $(COLUMN,1,K-1) TO VARIABLE
RELEASE GO
STORE COLUMN TO MCOL
IF BULD:IN
  DO BULDARTH
  IF ERROR
    RETURN
  ENDDIF (* error *)
  STORE MCOL TO COLUMN
  STORE TOTAL TO CONTENT
  STORE STORAGE TO K
  RELEASE STORAGE , MCOL
ELSE
  DO FINDROW
  IF ERROR
    RETURN
  ENDDIF (* error *)
  store field:name to f:name
  DO REPLACE
  STORE F:NAME TO FIELD:NAME
  STORE MCOL TO COLUMN
  RELEASE MCOL
  IF ERROR
    RETURN
  ENDDIF (* error *)
USE B:&CRTFILE
```

77
SET EXACT ON
LOCATE FOR &I:QUERY
IF EOF .AND. .NOT. &I:QUERY
   STORE T TO ERROR
   STORE TRIM(&I:QUERY) TO MSG
   STORE ' condition &MSG does not exist ' TO ERROR:MSG
RETURN
ENDIF (* eof.and. &i:query *)
STORE &FIELD:NAME TO CONTENT
STORE STORAGE TO K
SET EXACT OFF
release storage
USE B:FILE
ENDIF (* build:in *)
RETURN

******************************************************************************
** ARTHOPER command file ******************************************************
* This program handles a column of a query in which arithmetic *
* operators are involved. This program evaluates the expression* *
* and returns the result to the calling program. This program is *
* called by SCAN command file.
******************************************************************************

STORE 0 TO INDEX
DO PCHECK
   IF ERROR
      RETURN
   ENDIF (* error *)
RELEASE TEMP
STORE ' ' TO MVALUE
STORE T TO KEEP:GOING
DO WHILE KEEP:GOING
   STORE F TO DEPENDENT
   STORE 1 TO K
   STORE F TO STOP
   DO WHILE .NOT. STOP
      IF K > LEN(COLUMN)
         STORE T TO STOP
      LOOP
ELSE
   IF $(COLUMN,K,1) = '_'
      DO ARTHDEF
         IF ERROR
            RETURN
         ENDIF (* error *)
         STORE T TO STOP
         STORE T TO DEPENDENT
      LOOP
      ENDIF (* $(column,1,1) = '_' *)
   IF $(COLUMN,K,1)='+' OR $(COLUMN,K,1)='-' OR $(COLUMN,K,1)='/' OR $(COLUMN,K,1)='*'
      STORE T TO STOP
   ENDIF
   LOOP
ELSE
IF $(COLUMN,K,1) >= '0' .AND. $(COLUMN,K,1) <= '9'
STORE K+1 TO K
ELSE
STORE T TO ERROR
STORE " *** ERROR, ONLY NUMBER CAN BE USED IN "+ ARITHMETIC OPERATION" TO ERROR:MSG
RETURN
ENDIF
ENDIF
ENDDO (* len(column) ................. *)
IF K = 1 .AND. (.NOT. ( $(COLUMN,1,1) >= '0' .AND. $(COLUMN,1,1) <= '9') )
STORE $(COLUMN,1,1) TO CONTENT
IF CONTENT = '*' .OR. CONTENT = '/' .OR. CONTENT = '-' .OR. CONTENT = '+'
STORE MVALUE+ CONTENT TO MVALUE
IF LEN(COLUMN) = 1 .AND. ( CONTENT = '*' .OR. CONTENT = '/' .OR. CONTENT = '-' .OR. CONTENT = '+')
STORE T TO ERROR
STORE " *** ERROR, INVALID QUERY " TO ERROR:MSG
RETURN
ELSE
STORE $(COLUMN,2) TO COLUMN
ENDIF (* len(column) = 1 *)
ENDIF
ELSE
STORE 1+INDEX TO INDEX
STORE 'M'+STR(INDEX,1) TO M
IF DEPENDENT
STORE CONTENT TO &M
release content
ELSE
STORE VAL($(COLUMN,1,K-1)) TO &M
ENDIF (* dependent *)
STORE MVALUE+ '&M' TO MVALUE
IF LEN(COLUMN) = K-1
STORE F TO KEEP:GOING
ELSE
STORE $(COLUMN,K) TO COLUMN
ENDIF (* len(column) = k-1 *)
ENDIF (* K = 1 *)
IF KEEP:GOING
DO WHILE LEN(COLUMN) >= 1 .AND. $(COLUMN,1,1) = '1'; .AND. KEEP:GOING
IF $(COLUMN,1,1) = '
STORE VALUE + ' TO VALUE
IF LEN(COLUMN) = 1
STORE F TO KEEP:GOING
ELSE
STORE $(COLUMN,2) TO COLUMN
ENDIF (* len(column) = 1 *)
ENDIF (* $(COLUMN,1,1) = '1' *)
ENDDO
ENDDO(*)
STORE&MVALUE TO TOTAL
STORE 1 TO INDEX
DO WHILE INDEX < INDEX
    STORE 'M'+STR(INDEX,1) TO M
    RELEASE &M
    STORE INDEX+1 TO INDEX
ENDDO(*)
release m, index, index, value
RETURN

AVG COMMAND FILE
This program calculates the average or the sum of an data item. This program is invoked by SCAN command file.

GO BOTTOM
STORE # TO BOTTOM
IF UNIQUE
    GO TOP
STORE 0 TO CNT
DO WHILE .NOT. EOF
    IF .NOT.*
        STORE &VAR:AVG TO TEMP
        STORE CNT+1 TO CNT
        STORE # TO REC:NO
        SET EXACT ON
        LOCATE FOR &VAR:AVG = TEMP
        CONTINUE
        DO WHILE .NOT. EOF
            IF .NOT.*
                DELETE
                ENDF(*.not.* *)
                CONTINUE
            ENDDO(*)
            IF REC:NO = BOTTOM
                SKIP
            ELSE
                STORE REC:NO+1 TO REC:NO
                GO REC: NO
            ENDDO
            SET EXACT OFF
            ELSE
                SKIP
            ENDF(*.not.* *)
            ENDDO(*)
    ELSE
        GO BOTTOM
        STORE # TO CNT
        GO TOP
        ENDF(* AVG:UNQ *)
        SUM &VAR:AVG TO TOTAL
        IF CMD = 'AVG'
            STORE TOTAL/CNT TO TOTAL
ENDIF
RECALL ALL
RELEASE VAR:AVG,AVG:UNQ,REC:NO,REC
RETURN

******************************************************************************* AVTABLE command file ****************************

* This program, which is called by PARSE1 command file, scans the user's query and separates each data entry into six categories.

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

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:SFFILE
    STORE 'FILE' + STR(I,1) TO FILE
    STORE TRIM(FIELD: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,MBUILTDFUN
        STORE TYPE TO MTYPE
        STORE LEN TO MLEN
        STORE DEC TO MDEC
        STORE TRIM(FIELD: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.'
                    RELEASE R:EXIST,UNION,JOIN,BUILDFUN,UNPURGED,MPREFIX,
                    MVALUE,MOPERATOR,MBUILTDFUN,MAX:REC,MTYPE,MLEN,MDEC
                    RELEASE PARSE,PURGED
                    IF C:EXIST
                        DO SCANCMD
IF ERROR
   RETURN
ENDIF
ENDIF
STORE T TO FINISHED
DO PROCUPDATE
IF FINISHED
   RETURN
ENDIF
ELSE
   IF $(PARSER,1,1) = ' _'
      IF $(PARSER,1,2) = ' '
         STORE '*** ERROR - no blank can be' + ;
         'imbedded between variable name and underline' ; 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.;
               OR. $(PARSER,1,4) = 'UNIQUE'
               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 - ';
                        'invalid symbol' TO ERROR:MSG
                        STORE T TO ERROR
                  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,;
TABLEID WITH STR(I,1)
REPLACE BUILTFUN WITH MBUILTFUN
ENDIF
ENDIF
SELECT PRIMARY
ENDDO
ENDDO
RELEASE MTYPE,FIELD:NAME,MVALUE,MPREFIX,;
MBUILTFUN,MOPERATOR,I,J,K,MAX:REC
RELEASE PARSER,FILE,SPFILE,ROW:NO,BLANK
SELECT PRIMARY
USE
SELECT SECONDARY
USE
RETURN

*************** BULDARTH command file ********************
* This program invokes the proper built-in function subroutine *
* and returns the result which is obtained from evaluating a  *
* built-in function to its calling program - ARTHDEP command  *
* file.                                                 *
****************************************************************

DO FINDROW
IF ERROR
    RETURN
ENDIF (* error *)
STORE FIELD:NAME TO F:NAME
DO REPLACE
IF ERROR
    RETURN
ENDIF
STORE F:NAME TO FIELD:NAME
RELEASE F:NAME
USE B:&CRTFILE
DO CASE
CASE CMD = 'AVG'
    STORE FIELD:NAME TO VAR:AVG
DO GETAVG
CASE CMD = 'CNT'
STORE FIELD:NAME TO VAR:CNT
DO GETCNT
CASE CMD = 'SUM'
STORE FIELD:NAME TO VAR:SUM
DO GETSUM
CASE CMD = 'MAX'
STORE FIELD:NAME TO VAR:MAX
DO GETMAX
CASE CMD = 'MIN'
STORE FIELD:NAME TO VAR:MIN
DO GETMIN
OTHERWISE
STORE T TO ERROR
STORE '*** ERROR , INVALID BUILD-IN FUNCTION' TO ERROR:MSG
RETURN
ENDCASE /* CASE */
RETURN

*************** BULDFUN command file ****************************
* This program handles the update query with built-in function.  *
* This program is called by CONUPD command file.               *
**************************************************************

STORE ROW TO T:ROW
DO SCAN
IF ERROR
  RETURN
ENDIF /* error */
STORE '' TO CONTENT
IF FLD:DEC > 0
  STORE STR(TOTAL,FLD:LEN,FLD:DEC) TO CONTENT
ELSE
  STORE STR(TOTAL,FLD:LEN) TO CONTENT
ENDIF /* dec > 0 */
STORE " &FIELD:NAME WITH &CONTENT " TO STRING
STORE T:ROW TO ROW
RELEASE TOTAL,T:ROW
USE
SELECT PRIMARY
USE B:&FILE
GO RECO REC:NUM
RELEASE REC:NUM
RETURN

*********************** CHKFLD command file ********************
* This program checks all field names of a specified file   *
* to see if they are all existing in the file. This program  *
* is called by PARSE1 command file.                        *
**************************************************************
STORE 0 TO I
DO WHILE I < FILE:NUM
  STORE I + 1 TO I
84
STORE 'SFILE' + STR(I,1) TO SFILE
USE B:SFILF
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
    ENDF
  ENDF
SKIP
ENDDO
ENDDDO
STORE 0 TO I
DO WHILE I < FILE:NUM
  STORE I + 1 TO I
  STORE 'SFILE' + STR(I,1) TO SFILE
  USE B:SFILF
  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:SFILF
    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 '*** ERROR - SYNTAX ERROR' TO ERROR:MSG
        END "COMMAND <> 'I'
          STORE '*** ERROR - INVALID COMMAND ' TO;
          ERROR:MSG
          STORE T TO ERROR
        ELSE
          DO INSFILD
          STORE T TO FINISHED
        ENDF
      ENDF
    ENDDO
LOOP
ELSE
    STORE TRIM(FIELD:NAM) TO FIELD:NAM
    SELECT SECONDARY
    USE B:FIELDS INDEX B:FLDNX
    FIND &FIELD:NAM
    IF # = 0
        STORE "*** ERROR - field &field:nam is not"+;
            " existing in file &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

*************** CONCATENATE command file ***************
* This program concatenates the condition portion of a query *
* into a dBASE-II command statement. This program is invoked *
* by CONUDP command file. ***************

GO RECO 2
STORE ' ' TO I:QUERY
DO WHILE NOT. EOF
    IF &ROW:NO = ' ' .OR. FLD:NAM = FIELD:NAM
        SKIP
    LOOP
ELSE
    IF '"$(&ROW:NO) .OR. '/"$(&ROW:NO) .OR. '+"$(&ROW:NO);
        .OR. '-'.$(&ROW:NO)
        IF TYPE = 'C'
            STORE T TO ERROR
            STORE " *** ERROR , ARITHMETIC OPERATORS ONLY CAN"+;
"BE USED IN NUMERICAL TYPE" TO ERROR:MSG
ENDIF (* type = 'c'. *)
STORE TRIM(FLD:NAME) TO FIELD
STORE &ROW:NO TO COLUMN
STORE T TO BULD:IN
DO ARTHOPER
IF ERROR
RETURN
ENDIF
STORE '.AND.' + '&FIELD = TOTAL ' + I:QUERY TO I:QUERY
RELEASE FIELD
ELSE
IF '|'$(&ROW:NO) .OR. '$('(&ROW:NO)
DO LOGQUERY
IF ERROR
RETURN
ENDIF (* error *)
STORE '.AND.' + TEMP + I:QUERY TO I:QUERY
RELEASE TEMP
ELSE
STORE TRIM(&ROW:NO) TO COLUMN
DO TYPECHK
IF ERROR
RETURN
ENDIF (* error *)
IF TYPE = 'N'
STORE TRIM(FLD:NAME) TO FIELD
STORE '.AND.' + '&FIELD = COLUMN' + I:QUERY TO I:QUERY
ELSE
STORE TRIM(FLD:NAME) TO FIELD
STORE '.AND.' + '&FIELD = "' + COLUMN + "' + I:QUERY;
TO I:QUERY
ENDIF
ELSE
ENDIF (*
ENDIF
ENDIF
RELEASE FIELD
SKIP
ENDDO

*****************************************************************************
CONUPD command file ****************************
* This program processes the update query which only uses one *
* table and also is a single record update. This program is *
* invoked by UPDATE command file.
*****************************************************************************

SET EXACT ON
LOCATE FOR FLD:NAME = FIELD:NAME
SET EXACT OFF
STORE TRIM(&ROW:NO) TO COLUMN
STORE $(COLUMN,3) TO COLUMN
STORE #('.',COLUMN) TO FOUND:AT
STORE T TO POINT
IF FOUND:AT > 0
IF $(COLUMN,FOUND:AT-1,1)='012345678'
STORE T TO POINT
ELSE
STORE F TO POINT
ENDIF
IF FOUND:AT = 4 .AND. .NOT. POINT
STORE FLD:NAME TO FILE:NAME
STORE $(COLUMN,1,3) TO CMD
IF CMD = 'SUM' .OR. CMD = 'CNT' .OR. CMD = 'MIN' .OR. CMD = 'AVG' .OR. CMD = 'MAX'
IF TYPE = 'C'
STORE T TO ERROR
STORE '*** ERROR, &FIELD:NAME is NOT NUMERICAL ';
+ 'TYPE' TO ERROR:MSG
RETURN
ENDIF (* type = 'c' *)
STORE F TO POINT
STORE # TO REC:NUM
DO SCAN
IF ERROR
RETURN
ENDIF (* error *)
USE B:FILE
GO RECO REC:NUM
IF DEC > 0
STORE STR(TOTAL,LEN,DEC) TO VALUE
ELSE
STORE STR(TOTAL,LEN) TO VALUE
ENDIF (* dec > 0 *)
STORE '&FIELD:NAME with &VALUE' TO STRING
RELEASE VALUE,REC:NUM
ELSE
IF .NOT. POINT
STORE T TO ERROR
STORE '*** ERROR, invalid built-in function &COLUMN';
TO ERROR:MSG
RETURN
ENDIF (* .not. point *)
ENDIF(* cmd = 'sum' .or. cmd = 'cnt' .. *)[1]
ELSE
IF .NOT. POINT
STORE T TO ERROR
STORE '*** ERROR, invalid built-in function &COLUMN';
TO ERROR:MSG
RETURN
ENDIF (* .not. point *)
ENDIF (* found:at = 0 *)
ENDIF (* *)
IF POINT
IF '+'$COLUMN .OR. '-'$COLUMN .OR. '*'$COLUMN .OR. '/'$COLUMN
IF TYPE <> 'N'
STORE T TO ERROR
STORE " &FLD:NAME is not numeric type " TO ERROR:MSG
RETURN
ENDIF (* type = 'n' *)
IF BLNK
   DO GETVAR
   DO PROCONTENT
   IF ERROR
      RETURN
   ENDIF (* error *)
   STORE ROW TO ROW: NO
ENDIF (* error *)
ELSE
   STORE F TO BULD: IN
   STORE # TO REC: NUM
   DO ARTHOPERATOR
   IF ERROR
      RETURN
   ENDIF (* error *)
   USE B: &FILE
   GO RECO REC: NUM
   IF DEC > 0
      STORE STR(TOTAL, LEN, DEC) TO CONTENT
   ELSE
      STORE STR(TOTAL, LEN) TO CONTENT
   ENDIF (* dec > 0 *)
   STORE '&FIELD: NAME WITH &CONTENT' TO STRING
   ENDIF (* blank *)
ELSE
   IF '"'$COLUMN
   IF ERROR
      RETURN
   ENDIF (* error *)
   STORE COLUMN TO VARIABLE
   STORE # TO REC: NUM
   STORE LEN TO FLD: LEN
   STORE DEC TO FLD: DEC
   DO QURYDEP
   IF ERROR
      RETURN
   ENDIF (* error *)
   IF TYPE = 'N'
      STORE '&FIELD: NAME WITH &CONTENT' TO STRING
   ELSE
      STORE '&FIELD: NAME WITH &CONTENT' TO STRING
   ENDIF (* type = 'N' *)
ELSE
   IF TYPE = 'N'
      DO TYPECHK
      IF ERROR
         RETURN
      ENDIF (* error *)
      STORE '&FIELD: NAME WITH &COLUMN' TO STRING
   ELSE
      DO TYPECHK
      IF ERROR
         RETURN
     ENDIF (* error *)
     STORE '&FIELD: NAME WITH &COLUMN' TO STRING
   ENDIF
RETURN
ENDIF (* error *)
STORE '&FIELD:NAME WITH "&COLUMN"' TO STRING
ENDIF (* type = 'n' *)
ENDIF (*
ENDIF
ENDIF
IF .NOT. BLNK
DO CONCATENATE
IF ERROR
RETURN
ENDIF (* error *)
ENDIF (* not blnk *)
USE B:&FILE:NAME
SET EXACT ON
IF I:QUERY = ' '
   REPLACE ALL &STRING
ELSE
   STORE $(I:QUERY,6) TO I:QUERY
   REPLACE ALL &STRING FOR &I:QUERY
ENDIF (* i:query = ' ' *)
SET EXACT OFF
USE
STORE T TO FINISH
RETURN

*************** CRTFILE command file **********************
* This program generates a table which has four column
* headings: field: name, field:type, field:len, and field:dec. *
* The purpose of this skeleton table is to accept information *
* about new fields that are desired to be contained in the new *
* table.

ERASE
SET TALK OFF
SET COLON OFF
USE FILES
COPY TO TEMP STRUCTURE EXTENDED
USE TEMP
DELETE ALL
PACK
@ 2,1 SAY " PLEASE FILL IN THE FIELD:NAME AND STRUCTURE "
STORE 1 TO COUNTER
@ 4,2 SAY FNAME
@ 4,11 SAY " | FIELD:NAME | FIELD:TYPE |"+;
   "FIELD:LEN | FIELD:DEC |"
@ 5,1 SAY "-----------------------------+"+
     "-----------------------------|
STORE 6 TO ROW
STORE T TO FLAG
DO WHILE FLAG .AND. COUNTER < 7
   STORE T TO DRAWBAR
   @ ROW,1 SAY COUNTER
   STORE 13 TO COL

90
DO WHILE DRAWBAR
   @ ROW, COL SAY " |"
STORE COL+13 TO COL
IF COL > 70
   STORE F TO DRAWBAR
ENDIF
ENDDO(#DRAWBAR*)
STORE ' ' TO FLDNAME
STORE ' ' TO FLDTYPE
STORE ' ' TO FLDDDEC
@ ROW, 15 GET FLDNAME
READ
IF FLDNAME = ' '
   STORE F TO FLAG
   STORE F TO MORE
LOOP
ENDIF
STORE T TO OK
DO WHILE OK
   @ ROW, 32 GET FLDTYPE
READ
   IF (FLDTYPE) <> 'C' .AND. (FLDTYPE) <> 'N'
      @ 23, 1 SAY " **** INVALID TYPE ,";+
      " PLEASE RE-ENTER *******"
   ENDIF
   STORE 1 TO CNT
   DO WHILE CNT < 20
      STORE CNT+1 TO CNT
   ENDDO
   RELEASE CNT
   STORE ' ' TO FLDTYPE
   @ 23, 1
ELSE
   STORE F TO OK
ENDIF (* CHECK FLDTYPE *)
ENDDO (* OK *)
STORE F TO OK
DO WHILE NOT. OK
STORE ' ' TO FLDDLLEN
@ ROW, 45 GET FLDDLLEN PICTURE '99'
READ
   IF VAL(FLDDLLEN) <= 60
      STORE T TO OK
   ENDDIF
ENDDO
IF (FLDTYPE) <> 'C'
STORE T TO OK
DO WHILE OK
   @ ROW, 55 GET FLDDDEC PICTU '99'
READ
   STORE FLDDDEC TO COLUMN
   RELEASE COLUMN, TYPE ,DEC
   IF VAL(FLDDDEC) < 0 .OR. VAL(FLDDDEC) > VAL(FLDDLLEN)
      @ 23, 1 SAY " *** BAD FIELD DEIMAL ,";+
      " PLEASE RE-ENTER ***"
   ENDIF
   STORE 1 TO CNT
   -
DO WHILE CNT < 20
    STORE CNT+1 TO CNT
    ENDDO
    @ 23,1
    RELEASE CNT
    STORE ' ' TO FLDDDEC
ELSE
    STORE F TO OK
ENDIF
ENDIF
ENDDO (* OK *)
RELEASE OK
ENDDO
COUNTER + 1 TO COUNTER
APPEND BLANK
REPLACE FIELD:NAME WITH I(FLDNAME),;
    FIELD:TYPE WITH I(FLDTYPE),FIELD:LEN WITH;
    VAL(FLDLEN), FIELD:DEC WITH VAL(FLDDEC)
    STORE ROW + 1 TO ROW
ENDDO (* FLAG *)
IF COUNTER > 7
    @ 23,1 SAY " ***** OUT OF RANGE *****"
    STORE F TO MORE
ENDIF (* COUNTER > 7 *)
RELEASE FLAG,FLDNAME,FLDLEN,FLDTYPE,FLDDEC
STORE COUNTER -1 TO COUNTER
DO GETDATA
RETURN

************** CHKFUN command file ***************
* This program is a subroutine of the replace program which *
* is used to handle the query which has a built-in function *
* involved.
******************************************************************

IF TYPE = 'C'
    STORE T TO ERROR
    STORE '*** ERROR, BUILD-IN FUNCTION ONLY USE IN'+;
    'NUMBERICAL TYPE' TO ERROR:MSG
RETURN
ENDIF (* type = 'c' *)
IF MVAL = 1
    STORE $(COLUMN,4) TO COLUMN
    STORE F TO PREFIX
ELSE
    STORE $(COLUMN,MVAL-4) TO FRT:PART
    STORE $(COLUMN,MVAL+1) TO COLUMN
    STORE T TO PREFIX
    SET EXACT ON
    IF FRT:PART <> '<' .OR. FRT:PART <> '<' .OR.;
       FRT:PART <> '<=' .OR. FRT:PART <> '<=' .OR.;
       FRT:PART <> '>.'
    STORE T TO ERROR
    STORE '*** ERROR, INVALID RELATIONAL'+;
'OPERATOR &FRT:PART' TO ERROR:MSG
RETURN
ENDIF (* .f rt:part <> '<<'... *)
SET EXACT OFF
ENDIF (* mval = 1 *)
DO SCAN
IF ERROR
RETURN
ENDIF
IF DEC > 0
STORE STR(TOTAL, LEN, DEC) TO CONTENT
ELSE
STORE STR(TOTAL, LEN) TO CONTENT
ENDIF (* dec > 0 *)
IF PREFIX
STORE '&FLD:NAME'-FRT:PART+CONTENT TO CONTENT
ELSE
STORE '&FLD:NAME'-='+CONTENT TO CONTENT
ENDIF (* prefix *)
RELEASE PREFIX, TOTAL, COLUMN, MVAL, FRT:PART
RETURN

*****************************************************************************
* CHKFILE command file ******************************************************
* This program, which is called by SCANFN command file, checks*  
* a new fiel 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 exceeds 8 character ';
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 - imbedded 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

93
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

*************** CLEARFILE command file **********************
* This program deletes all the files that are used as temporary *
* storage files. This program is called by RUN command file. *
*************** CLEARFILE command file **********************

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

*************** CHKOTHER command file **********************
* This program checks if the user enters more than one type of *
* of operator in a query. This program is called by PARSE1 *
* command file. 
*************** CHKOTHER command file **********************

IF $(ROW1,1,1)$ OPERATOR
  IF $(ROW1,2,1) <> SYMBOL
    STORE '*** ERROR - missing '.' To ERROR:MSG
    STORE T TO ERROR
  ELSE
    IF $(ROW2,1,2) <> ' ' .OR. $(ROW3,1,2) <> ' '
      STORE '***ERROR - too many operator' TO ERROR:MSG
      STORE T TO ERROR
    ELSE
      STORE 1 TO NUM
    ENDIF
  ELSE
    IF $(ROW2,1,1)$ OPERATOR
IF $(ROW2,2,1) <> SYMBOL
  STORE '*** SYNTAX ERROR' TO ERROR:MSG
  STORE T TO ERROR
ELSE
  IF $(ROW1,1,2) <> ' ' .OR. $(ROW3,1,2) <> ' '
    STORE '***ERROR - too many operator' TO ERROR:MSG
    STORE T TO ERROR
  ENDIF
ELSE
  STORE 2 TO NUM
ENDIF
ELSE
IF $(ROW3,1,1) $ OPERATOR
  IF $(ROW3,2,1) <> SYMBOL
    STORE '*** SYNTAX ERROR' TO ERROR:MSG
    STORE T TO ERROR
  ELSE
    IF $(ROW1,1,2) <> ' ' .OR. $(ROW2,1,2) <> ' '
      STORE '***ERROR - too many operator' TO ERROR:MSG
      STORE T TO ERROR
    ELSE
      STORE 3 TO NUM
    ENDIF
  ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
RETURN

************************************************************************
CNT COMMAND FILE 
************************************************************************
* This program counts the number of data item. This program is *
* is called by SCAN command file. 
************************************************************************
GO BOTTOM
STORE # TO BOTTOM
GO TOP
STORE 0 TO TOTAL
IF UNIQUE
  DO WHILE .NOT. EOF
    IF *
      SKIP
    ELSE
      STORE TOTAL + 1 TO TOTAL
      STORE &VAR:CNT TO TEMP
      STORE # TO REC:NO
      LOCATE FOR &VAR:CNT = TEMP
      CONTINUE
    DO WHILE .NOT. EOF
      IF .NOT. *
        DELETE
    ENDIF
CONTINUE
ENDDO(* .not. EOF *)
IF REC: NO = BOTTOM
   SKIP
ELSE
   STORE REC: NO + 1 TO REC: NO
   GO REC: NO
ENDIF
ENDDO (* .not. eof *)
RELEASE REC, REC: NO, TEMP, VAR: CNT
ELSE
   GO BOTTOM
   STORE # TO TOTAL
   GO TOP
ENDIF (* if avg: unq *)
RELEASE VAR: CNT
RECALL ALL

******************************************************************************
* This program will interact with users and ask them what *
* kinds of table general or result or condition table)they *
* need, then display the table as users' request. This program* *
* is called by QBE command file.                                  **********

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

96
STORE T TO OK
ENDIF
ENDDO (* ok *)
RELEASE OK
IF ANSWER = 'Y'
IF FILE:NUM < 2
IF R:EXIST
STORE '(G)ENERAL TABLE (C)ONDITON 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)ONDITON 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)ONDITON BOX (E)XIT, CHOOSE A LETTER ==='';
TO QUESTION
STORE "ANSWER = 'C' .OR. ANSWER = 'E'" TO CORRECT
ELSE
STORE "ANSWER = 'C' .OR. ANSWER = 'E'" TO CORRECT
STORE '(R)ESULT TABLE (C)ONDITON BOX, (E)XIT, '+;
' CHOOSE A LETTER ==='' TO QUESTION
STORE "ANSWER = 'R' .OR. ANSWER = 'C' .OR. '' +;
 " ANSWER = 'E'" TO CORRECT
ENDIF (* result:exit *)
ENDIF (* 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 >= maxrow *)
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
*************** DOCOM command file ***************
* This program invokes the proper subroutine for processing *
* the delete or update operation. This program is invoked *
* by PARSE1 command file.

DO CASE
  CASE COMMAND = 'I.'
    IF C:EXIST
      DO SCANCND
    ENDF (* c:exist *)
    DO PROCINSERT
  CASE COMMAND = 'D.'
    IF FILE:NUM = 1
      DO DROPALL
    IF DROP
      RETURN
    ENDF
    ENDF
    IF C:EXIST
      DO SCANCND
    ENDF (* c:exist *)
    DO DELETION
ENDCASE
RETURN

*************** DTABLE command file ***************
* This program displays either a general table or a result *
* table on the screen to allow the user to enter the query. *
* This program is invoked by DRAW command file.

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 FILE 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
    ENDF (* command <> ' ' *)
    ELSE
      @ 23, 1 SAY ERROR:MSG
      STORE 0 TO TIMER
      DO WHILE TIMER <= 50
58


STORE TIMER + 1 TO TIMER
ENDDO (* timer <= 50 *)
@ 23,1
ENDDO (* ok *)
RELEASE TIMER, ERROR:MSG
ENDDO (* not ok *)
STORE FILE:NUM + 1 TO FILE:NUM
ENDDO (* 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 FILE 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
ENDDO
DO CHKFILE
IF .NOT. OK
@ 23,1 SAY ERROR:MSG
STORE 0 TO TIMER
DO WHILE TIMER <= 50
STORE TIMER + 1 TO TIMER
ENDDO (* timer <= 50 *)
@ 23,1
RELEASE ERROR:MSG
LOOP
ELSE
STORE T TO SNAPSHOT
STORE FNAME TO SNAP:NAME
ENDDO
ENDIF
ENDDO
ENDDO (* answer = 'R' *)
IF MAXROW = 6
STORE ROW + 1 TO ROW
ELSE
STORE ROW + 1 TO ROW
STORE ROW + 2 TO ROW
ENDDO (* 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 <> 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
      ENDF (* col:count = 1 *)
      @ ROW,COL GET &PARSER PICTURE '!!!!!!!!!!!!!!'
      ENDF (* row:count <> 1 or col:count <> 1 *)
    IF COL:COUNT = 1
      STORE COL + 2 TO COL
    ELSE
      STORE COL + 12 TO COL
      ENDF (* col:count = 1 *)
    IF COL:COUNT <> MAXCOL
      @ ROW,COL SAY '|
      ENDF (* col:count <> maxcol *)
    STORE COL + 1 TO COL
    ENDDO (* col:count <= maxcol *)
    STORE ROW:COUNT + 1 TO ROW:COUNT
    ENDF (* row = maxrow - 3 *)
    STORE ROW + 1 TO ROW
    STORE 0 TO COL, COL:COUNT
  ENDDO (* row < maxrow *)
READ
DO STORE1

************** DELETION command file ***********************
* This program processes the queries which cause a delete operation to an existing file. This program converts the user's query to a dBASE-II command statement in order to perform operation on the data base. This program is invoked by DOCOM command file. ***********************

USE B:SF1
GO TOP
STORE 1 TO I
STORE F TO PROCESS
STORE ' ' TO MY:QUERY
DO WHILE I <= 3
  STORE 'ROW' + STR(I,1) TO ROW:NO
  IF .NOT. (&ROW:NO = 'D' .OR. &ROW:NO = ' ')
    STORE I TO ERROR
    STORE " *** ERROR, YOU CAN ONLY HAVE ONE QUERY" +;
RETURN
ENDIF (* i <= 3 *)
STORE I+1 TO I
IF I = 4
IF FILE:NUM = 2
IF .NOT. PROCESS
USE B:SFFILE2
STORE T TO PROCESS
STORE 1 TO I
ELSE
USE B:SFFILE1
ENDIF
ENDIF (* file:no > 1 *)
ENDIF (* i = 4 *)
ENDDO (* i <= 3 *)
STORE O TO I
STORE 'SFFILE1' TO FILE
GO TOP
DO WHILE I < 3
USE B:&FILE
STORE I+1 TO I
STORE 'ROW'+STR(I,1) TO ROW:NO
STORE FLD:NAME TO FILE:NAME
IF &ROW:NO = 'D.'
SKIP
DO WHILE .NOT. EOF
IF &ROW:NO = ' '
SKIP
LOOP
ELSE
STORE &ROW:NO TO COLUMN
IF '/'+$COLUMN .OR. '*'+$COLUMN .OR. '-'+$COLUMN .OR. '+'+$COLUMN
STORE T TO ERROR
STORE "*** ERROR, ARITHMETIC OPERATOR AND"+;
" BUILT-IN FUNCTION ARE NOT ALLOWED IN"+;
"DELETION QUERY" TO ERROR:MSG
RETURN
ELSE
STORE @$('.' ,COLUMN) TO FOUND:AT
IF FOUND:AT <> 0
IF $(COLUMN,FOUND:AT-1,1) <= '0' .AND.; $(COLUMN,FOUND:AT-1,1) >= '9'
STORE T TO ERROR
STORE "*** ERROR, BUILT-IN FUNCTIONS"+;
" ARE NOT ALLOWED IN DELETION QUERY ";
TO ERROR:MSG
RETURN
ENDIF (* found:at <> 0 *)
ENDIF
release found:at
IF ']'+$COLUMN .OR. '&'$COLUMN
DO LOGQUERY
STORE TEMP TO PART
ELSE
  IF "_'.$ COLUMN
    IF @(".'", COLUMN) = 1
      STORE COLUMN TO PART
    ELSE
      STORE @(".'", COLUMN) TO FOUND:AT
      STORE T TO E
      STORE $(COLUMN,1,FOUND:AT-1) TO SIGN
      IF SIGN <> '>'; .AND. SIGN <> '<'; .AND.;
        SIGN <> '='; .AND. SIGN <> '<='; .AND.;
        SIGN <> '<' '>
      STORE T TO ERROR
      STORE "*** ERROR, INVALID QUERY";
      TO ERROR: MSG
      RETURN
    ELSE
      STORE LEN TO FLD:LEN
      STORE DEC TO FLD:DEC
      STORE $(COLUMN,FOUND:AT) TO PART
    ENDIF
  ENDF
  DO QUERYINDEP
  IF ERROR
    RETURN
  ENDF (* error *)
ELSE
  STORE &ROW: NO TO PART
  STORE '"' TO TEMP
  DO PQQUERY
  IF ERROR
    RETURN
  ENDF (* error *)
  STORE TEMP TO PART
  release temp
  ENDF (* '._ column *)
  ENDF (* '|".$ column .or. ' &".$ column *)
  SKIP
  ENDF (* &row: no = "' *)
  STORE MY: QUERY+".' .AND. ' + PART TO MY: QUERY
ENDDO (* . NOT. EOF *)
STORE $(MY: QUERY,7) TO MY: QUERY
USE B: &FILE: NAME
SET EXACT ON
LOCATE FOR (&MY: QUERY)
IF EOF . AND. . NOT. (&MY: QUERY)
  STORE T TO ERROR
  STORE " NOTHING HAS BEEN DELETED, SINCE NO RECORDS";
  " MEET THE CONDITION OF QUERY" TO ERROR: MSG
  RETURN
ENDF (* eof . and. . not. (&my: query))
LOCATE FOR (&MY: QUERY)
DO WHILE . NOT. EOF
  DELETE . NOT. EOF
  CONTINUE
ENDDO (* .not. eof *)
PCK
ENDIF (* i < 3 *)
SET EXACT OFF
ENDDO (* *)

************* DROPALL command file ******************
* This program checks if the user wants to erase all the *
* information in a existing file. This program is invoked by *
* DOCOM command file.

USE B:SPFILE1
GO TOP
IF ROW1 = 'D.' .AND. ROW2 = '' .AND. ROW3 = ''
  SKIP
  DO WHILE .NOT. EOF
    IF ROW1 = '' .AND. ROW2 = '' .AND. ROW3 = ''
      SKIP
    ELSE
      USE
      STORE F TO DROP
      RETURN
    ENDF
  ENDDO
STORE T TO DROP
STORE F TO OK
@ 23,1
DO WHILE .NOT. OK
  STORE '' TO ANS
  @ 23,1 SAY ' ARE YOU SURE ? ' GET ANS PICTURE 'I'
  READ
  IF ANS = 'Y'
    USE B:&FILE:NAME
    DELETE ALL
    PACK
    RETURN
  ELSE
    IF ANS = 'N'
      RETURN
    ENDF
  ENDF
ENDDO
ENDDO
ENDDF

*************** FINDROW COMMAND FILE **********************
* This program is invoked when the the user's query is query- *
* dependent. This program locates the process pointer to the *
* the row which has condition for an example element. This *
* program is invoked by QURYDEP, UPDEPEND, and ARTHDEP command *
* file.

**********
STORE O TO J
USE B:SF1E1
STORE 'SF1E1' TO FL
STORE FLD:NAME TO CRTFILE
STORE F TO FOUND
STORE F TO PROCESS
SET EXACT ON
DO WHILE J < 3 .AND. .NOT. FOUND
  STORE J+1 TO J
  STORE 'ROW'+STR(J,1) TO ROW
  IF &ROW = 'I.' .OR. (FL = FILE .AND. J = 1)
    LOOP
  ENDF (* &row *)
  LOCATE FOR FLD:NAME = FIELD:NAME
  IF &ROW = VARIABLE
    STORE T TO FOUND
  LOOP
  ENDF (* &row = variable *)
mb
  IF J = 3 .AND. .NOT. FOUND .AND. FILE:NUM > 1;
    .AND. .NOT. PROCESS
    USE B:SF1E2
    STORE 'SF1E2' TO FL
    STORE FLD:NAME TO CRTFILE
    STORE O TO J
    STORE T TO PROCESS
  ENDF
ENDDO (* j < 3 *)
SET EXACT OFF
IF .NOT. FOUND
  STORE T TO ERROR
  STORE " &VARIABLE CAN NOT BE FOUND IN YOUR QUERY " ;
  TO ERROR:MSG
  RETURN
ENDF (* .not. found *)
release found,process
RETURN

*********************** GETAVG COMMAND FILE ***********************
* This program calculates the average of a data item in a *
* certain scope which is specified by the user. This program is *
* is invoked by BULDARTH command file.  
*********************** GETAVG COMMAND FILE ***********************

SET EXACT ON
STORE O TO TOTAL
STORE O TO CNT
IF I:QUERY = '
  STORE T TO ERROR
  STORE ' *** ERROR, NO CONDITION HAS BEEN SPECIFIED';
  TO ERROR:MSG
  RETURN
ENDF (* i:query*)
LOCATE FOR (&I:QUERY)
IF EOF .AND. .NOT. (&I:QUERY)
  STORE T TO ERROR
STORE "** ERROR, There are no records which meet the' +;
' condition of query" TO ERROR:MSG
RETURN
ENDIF(* @ef .and. .not. (@i:query) *)
IF UNIQUE
STORE T TO GO
LOCATE FOR (@I:QUERY)
DO WHILE GO
  DO WHILE * .AND. .NOT. EOF
    CONTINUE
  ENDDO (* * *)
  IF EOF .AND. .NOT. (@I:QUERY)
    RECALL ALL
    STORE TOTAL/CNT TO TOTAL
    RELEASE GO,CONTENT
    RETURN
  ENDF(* e=of *)
  STORE &FIELD:NAME+TOTAL TO TOTAL
  STORE &FIELD:NAME TO CONTENT
  STORE CNT+1 TO CNT
  DELETE
  DO WHILE .NOT. EOF
    IF .NOT. *
      IF &FIELD:NAME = CONTENT
        DELETE
        ENDF (* &frt:col = content *)
      CONTINUE
      ELSE
        CONTINUE
        ENDF(* .not. * * *)
      ENDDO (* .not. eof *)
    GO TOP
  CONTINUE
  ENDDO (* FLAG *)
ELSE (* if unique = f *)
  LOCATE FOR ( &I:QUERY )
  DO WHILE .NOT. EOF
    STORE TOTAL+&FIELD:NAME TO TOTAL
    STORE CNT+1 TO CNT
    CONTINUE
  ENDDO (*. not. eof *)
  STORE TOTAL/CNT TO TOTAL
ENDIF (* unique *)
SET EXACT OFF
RELEASE I:QUERY,CNT
RETURN

*********************** GETCNT command file ***********************
* This program counts the number of data items in the certain    *
* scope which is specified by the user. This program is invoked *
* by BULDARTH command file.                                     *
*********************** BULDARTH command file ***********************
STORE 0 TO TOTAL
SET EXACT ON
IF I:QUERY = "'
STORE T TO ERROR
STORE "*** ERROR, NO CONDITION HAS BEEN SPECIFIED" TO ERROR:MSG
ENDIF (*i:query = "'
LOCATE FOR (&I:QUERY)
IF EOF .AND. .NOT. (&I:QUERY)
    STORE '*** ERROR, There are no records which meet the '+'
    'condition of query' TO ERROR:MSG
STORE T TO ERROR
RETURN
ENDIF(*EOF .AND. .NOT (&I:QUERY) *)
IF UNIQUE
STORE T TO GO
DO WHILE GO
    LOCATE FOR (&I:QUERY)
    DO WHILE *.AND. .NOT. EOF
        CONTINUE
    ENDDO (* * *)
    IF EOF
        RECALL ALL
        RELEASE GO,CONTENT,I:QUERY
        RETURN
    ENDIF (*eof *)
    STORE &FIELD:NAME TO CONTENT
    STORE TOTAL+1 TO TOTAL
    DELETE
    DO WHILE .NOT. EOF
        CONTINUE
        IF .NOT. * 
            IF &FIELD:NAME = CONTENT
                DELETE
            ENDIF (* &f:col = content *)
        ENDIF (* .not. * *)
    ENDDO (* .not. eof *)
    GO TOP
    ENDDO (* FLAG * )
ELSE (* if unique = f * )
    LOCATE FOR ( &I:QUERY )
    DO WHILE .NOT. EOF
        STORE 1+TOTAL TO TOTAL
        CONTINUE
    ENDDO (* .not. eof *)
ENDIF (* unique *)
SET EXACT OFF
RELEASE MY:QUERY
RETURN

*************************************************************************** GETDATA command file***************************************************************************
* This program is invoked by 'crtfile'. This program will allow * user to correct the data entry that (s)he did, and also allows* a user to input data for the new table created.
***************************************************************************

SET TALK OFF
STORE T TO FLAG
STORE ' ' TO ANS
DO WHILE FLAG
  @ 23,1 SAY " DO YOU WANT TO MODIFY ANYTHING ? Y/N " GET ANS
  READ
  IF 1(ANS) <> 'Y' .AND. 1(ANS) <> 'N'
    @ 23,1 SAY " PLEASE ENTER Y OR N , TRY AGAIN !
    STORE 1 TO CNT
    DO WHILE CNT < 20
      STORE 1+CNT TO CNT
    ENDDO
  STORE ' ' TO ANS
ENDIF
STORE F TO FLAG
ENDDO (* FLAG *)
RELEASE FLAG,CNT
IF 1(ANS) = 'Y'
  DO MODATA
ENDIF
CREATE B:&FNAME FROM B:TEMP
USE
DELETE FILE B:TEMP
USE B:COLFILE
APPEND BLANK
REPLACE NAME WITH FNAME
USE B:&FNAME
SET TALK ON
SET TALK OFF
STORE T TO FLAG
DO WHILE FLAG
  STORE ' ' TO ANS
  @ 23,1 SAY "
  @ 23,1 SAY " DO YOU WANT TO INPUT DATA, Y/N ? " GET ANS
  READ
  IF 1(ANS) <> 'Y' .AND. 1(ANS) <> 'N'
    LOOP
ENDIF
STORE F TO FLAG
ENDDO
IF 1(ANS) = 'Y'
  USE B:&FNAME
  APPEND
ENDIF
RETURN
******************************************************************************
* This program finds the largest data item in a certain scope. *
* This program is called by BULDARTH command file. *
******************************************************************************
IF I:QUERY = " 
  STORE T TO ERROR
  STORE ' *** ERROR, NO CONDITION HAS BEEN SPECIFIED!' TO ERROR:MSG
  RETURN
ENDIF (* i:query *)
LOCATE FOR (&I:QUERY)
IF EOF .AND. .NOT. &I:QUERY
STORE T TO ERROR
STORE ' *** ERROR, There are no records which meet the'+
condition of &I:query ' TO ERROR:MSG
RETURN
ENDIF (* eof .and. .not. &i:query *)
STORE &FIELD:NAME TO TOTAL
DO WHILE .NOT. EOF
    CONTINUE
    IF TOTAL < &FIELD:NAME
        STORE &FIELD:NAME TO TOTAL
    ELSE
        SKIP
    ENDDO (* total < &nam *)
ENDDO (* .NOT. EOF *)
RETURN

******************************************************************************
** GETMIN command file ****************************************************
* This program finds the smallest data item in a certain scope. *
* This program is called by BULDARTH command file. *
******************************************************************************

SET EXACT ON
IF &I:QUERY = ' '
STORE T TO ERROR
STORE ' *** ERROR, NO CONDITION HAS BEEN SPECIFIED' TO ERROR:MSG
RETURN
ENDIF (* &i:query *)
LOCATE FOR (&I:QUERY)
IF EOF .AND. .NOT. &I:QUERY
STORE T TO ERROR
STORE ' *** ERROR, There are no records which meet the condition '+
&I:query ' TO ERROR:MSG
RETURN
ENDIF (* eof .and. .not. &i:query *)
STORE &FIELD:NAME TO TOTAL
DO WHILE .NOT. EOF
    CONTINUE
    IF TOTAL > &FIELD:NAME
        STORE &FIELD:NAME TO TOTAL
    ELSE
        SKIP
    ENDDO (* value > &nam *)
ENDDO (* .not. eof *)
SET EXACT OFF
RETURN

******************************************************************************
** INSFLD command file ******************************************************
* This program processes the queries which want to insert fields *
* to an existing file. This program is called by CHKFLD command *
* file. *
******************************************************************************

USE
USE B:SF1
GO TOP

108
STORE FLD:NAME TO FNAME
SKIP
STORE 0 TO NUMBER
DO WHILE .NOT. EOF
    IF $(FLD:NAME,1,2) = 'I.'
       STORE 1+NUMBER TO NUMBER
       STORE 'FLD'+'STR(NUMBER,1) TO FIELD
       STORE TRIM($((FLD:NAME,3)) TO &FIELD
    ENDF (* $(&:fld:name,1,2) = 'i.' *)
    SKIP
ENDDO (* .not. eof * )
DO TEXPAND
RETURN

*************** LOGQUERY COMMAND FILE ***********************
* This program changes a logical expression from the QBE pattern *
* to a dBASE-II command statement. For example, changing the *
* '1000 || 2000' to '1000 .or. 2000'. This program is invoked *
* by REPLACE, DELETION, AND CONCATENATE command file. *
*************** LOGQUERY COMMAND FILE ***********************

DO PCHECK
IF ERROR
    RETURN
ENDIF (* error * )
STORE T TO TRUE
DO WHILE TRUE
    STORE LEN(COLUMN) TO STOP
    IF "|"$COLUMN)
       STORE @('|',COLUMN) TO FOUND:AT
       IF FOUND:AT = 1 .OR. FOUND:AT = STOP
          STORE "*** ERROR,INVALID QUERY IN &COMMAND EXPRESSION" TO ERROR:MSG
          STORE T TO ERROR
          RETURN
       ELSE
          STORE $((COLUMN,1,FOUND:AT-1) TO FIRST
          STORE $((COLUMN,FOUND:AT+1) TO SECOND
          STORE FIRST\"." .OR. " +SECOND TO COLUMN
          ENDF (* found:at = 1 .or. found:at = stop * )
       ELSE
          IF ' &"$COLUMN
             STORE @('&',FRT:COL) TO FOUND:AT
             STORE @('&',COLUMN) TO FOUND:AT
             IF FOUND:AT = 1 .OR. FOUND:AT = STOP
                STORE " *** ERROR, INVALID QUERY IN &COMMAND: +; 
                " EXPRESSION" TO ERROR:MSG
                STORE T TO ERROR
                RETURN
             ELSE
                STORE $((COLUMN,1,FOUND:AT-1) TO FIRST
                STORE $((COLUMN,FOUND:AT+1) TO SECOND
                STORE FIRST + ".AND. " + SECOND TO COLUMN

109
ENDIF(* found:at = 1 .or. found:at = stop *)
ELSE
    STORE F TO TRUE
ENDIF(* '&'$(column) *)
ENDIF(* '!$(column) *)
ENDDO(* true *)
STORE T TO TRUE
STORE F TO CORRECT
STORE ' ' TO TEMP
DO WHILE TRUE
    DO WHILE LEN(COLUMN) >= 1 .AND. $(COLUMN,1,1) = '('.OR.;
    $(COLUMN,1,1) = '
    IF $(COLUMN,1,1) = '
        STORE TEMP+('('TO TEMP
    ENDIF(* $(column,1,1) = '(' *)
    IF LEN(COLUMN) = 1 .AND. $(COLUMN,1,1) = '
        STORE F TO TRUE
        IF .NOT. CORRECT
            STORE T TO ERROR
            STORE ** ERROR, INVALID QUERY ** TO ERROR:MSG
            RETURN
        ENDIF(* .not. correct *)
    ELSE
        STORE $(COLUMN,2) TO COLUMN
        ENDIF(* len(column) *)
    ENDDO(* len(column) >= 1 .and. $(column,1,1) = '[' ... *)
    IF .NOT. TRUE
        RETURN
    ENDIF(* .not. true *)
    STORE 1 TO J
    STORE LEN(COLUMN) TO WORD:LEN
    STORE T TO KEEP:GOING
    DO WHILE J <= WORD:LEN .AND. KEEP:GOING
        IF $(COLUMN,J,1) = '.' .OR. $(COLUMN,J,1) = ')
            STORE F TO KEEP:GOING
        ELSE
            STORE J+1 TO J
        ENDIF(* $(column,j,1) <> '. ' .or. $(column,j,1) <> ')') *
    ENDDO(* j <= length .and. keep:going *)
    RELEASE KEEP:GOING
    IF J-1 > 1
        IF J > WORD:LEN .AND. $(COLUMN,J-1,1) = '.' .OR.;
            $(COLUMN,J-1,1) = '
        STORE $(COLUMN,1,J-1) TO PART
        STORE $(COLUMN,J) TO COLUMN
    ELSE
        IF J < WORD:LEN
            STORE $(COLUMN,1,J-1) TO PART
            STORE $(COLUMN,J) TO COLUMN
        ELSE
            STORE $(COLUMN,1,J-1) TO PART
            STORE F TO TRUE
        ENDIF(* j < length *)
    ENDIF(* j=length .and.($(column,j,1) = '.' .or.... *)
    IF '¬'*$PART

110
DO QUERYINDEP
IF ERROR
   RETURN
ENDDIF (* error *)
ELSE
   DO QUERY
      STORE PART+TEMP TO PART
   ENDDIF (* ' ' 'part ' ' ' *)
   IF ERROR
      RETURN
   ENDDIF
ELSE
   STORE T TO ERROR
   STORE " *** ERROR , INVALID QUERY " TO ERROR:MSG
   RETURN
ENDDIF (* )
IF TRUE
   IF $(COLUMN,1,1) = ' ' '
      IF LEN(COLUMN) >= 2
         DO WHILE LEN(COLUMN) >= 2 .AND. $(COLUMN,1,1) = ' ' '
            STORE TEMP + ' ' ' TO TEMP
            STORE $(COLUMN,2) TO COLUMN
         ENDDO(* len(column) >= 2 .and. $(column,1,1) = ' ' ' )
         IF $(COLUMN,1,1) = ' ' '
            STORE TEMP + ' ' ' TO TEMP
         ENDDIF
      ELSE
         STORE TEMP + ' ' ' TO TEMP
         STORE F TO TRUE
      ENDDIF (* $(column,1,1) = ' ] ' *)
   ENDDIF (* $(column,1,1) = ' ] ' *)
   IF LEN(COLUMN) > 2
      IF $(COLUMN,1,2) = ' .0 '
         STORE TEMP + ' .0 . ' TO TEMP
         IF LEN(COLUMN) = 4
            STORE T TO ERROR
            STORE " *** ERROR , INVALID QUERY " TO ERROR:MSG
            RETURN
         ELSE
            STORE $(COLUMN,5) TO COLUMN
         ENDDIF (* len(column) = 4 *)
      ELSE
         STORE TEMP + ' .AND. ' TO TEMP
         IF LEN(COLUMN) = 5
            STORE T TO ERROR
            STORE " *** ERROR , INVALID QUERY " TO ERROR:MSG
            RETURN
         ELSE
            STORE $(COLUMN,6) TO COLUMN
         ENDDIF
      ENDDIF (* $(column,1,2) = ' .0 ' *)
   ENDDIF (* true *)
ENDDO (* true *)
RELEASE TRUE , COLUMN, CORRECT
USE
STORE '' TO QUERY
SELECT PRIMARY
USE B:&FILE
SELECT SECONDARY
USE B:&FL
SELECT PRIMARY
GO RECO 2
DO WHILE .NOT. EOF
  IF &ROW: NO = '' .OR. FLD: NAME = FIELD: NAME
    IF FLD: NAME = FIELD: NAME
      IF */ '' $&ROW: NO . OR. '/'' $&ROW: NO . OR. '-'' $&ROW: NO . OR. ;
        '+'' $&ROW: NO
      IF TYPE = 'C'
        STORE T TO ERROR
        STORE '*** ERROR, &FIELD: NAME IS NOT NUMERICAL' +
        ' TYPE' TO ERROR: MSG
        RETURN
      ENDIF (* type = 'c' *)
    STORE F TO BULD: IN
    STORE ROW TO T: ROW
    STORE TRIM($(&ROW: NO,3)) TO COLUMN
    STORE # TO REC
    DO ARTHOPER
    IF ERROR
      RETURN
    ENDIF (* error * )
    STORE T: ROW TO ROW
    SELE PRIM
    GO RECO REC
    IF DEC > 0
      STORE STR(TOTAL, LEN, DEC) TO VALUE
    ELSE
      STORE STR(TOTAL, LEN) TO VALUE
      ENDIF (* dec > 0 * )
      STORE '&FIELD: NAME WITH &VALUE' TO STRING
    ELSE
      STORE TRIM($(&ROW: NO,3)) TO COLUMN
      STORE @('.',COLUMN) TO FOUND: AT
      IF FOUND: AT > 0
        IF FOUND: AT = 4
          STORE $(COLUMN,1,3) TO CMD
          IF CMD = 'SUM' .OR. CMD = 'CNT'. OR. CMD =
             'AVG'. OR. CMD = 'MIN' . OR. CMD = 'MAX'
             IF TYPE = 'C'
               STORE T TO ERROR
               STORE ' *** ERROR, &FIELD: NAME IS NOT';
             ELSE
' NUMERICAL TYPE' TO ERROR:MSG
RETURN
ENDIF (* type = 'c' *)
STORE # TO REC:NUM
STORE LEN TO FLD:LEN
STORE DEC TO FLD:DEC
DO BULDFUN
IF ERROR
RETURN
ENDIF (* error *)
RELEASE FLD:DEC, FLD:LEN, CMD
ENDIF
ENDIF (* found:at = 4 *)
ELSE
IF '_':&ROW:NO
STORE # TO REC
DO UPDEPEND
IF ERROR
RETURN
ENDIF (* error *)
SELE PRIM
USE B:&FILE
GO RECO REC
SELE SECO
USE B:&FL
SELE PRIM
ELSE
STORE TRIM(&ROW:NO, 3)) TO CONTENT
IF TYPE = 'N'
STORE '&FIELD:NAME WITH &CONTENT' TO STRING
ELSE
STORE '&FIELD:NAME WITH "&CONTENT"';
TO STRING
ENDIF (* TYPE = 'N' *)
ENDIF (* found:at = 0 *)
ENDIF (* found:at > 0 *)
ENDIF (* found:at = field:name *)
ELSE
STORE FLD:NAME TO F:NAME
STORE TRIM(&ROW:NO) TO VARIABLE
SELECT SECONDARY
SET EXACT ON
LOCATE FOR FLD:NAME = F:NAME
SET EXACT OFF
IF EOF .AND. FLD:NAME <> F:NAME
STORE T TO ERROR
STORE " *** ERROR, FIELD &F:NAME CAN NOT BE FOUND"+
" IN &FL " TO ERROR:MSG
RETURN
ENDIF (* eof .and. fld:name <> f:name *)
IF VARIABLE = TRIM(&ROW)
STORE '.AND."&F:NAME = P.&F:NAME"+ QUERY TO QUERY
ELSE
STORE T TO ERROR

113
STORE " *** ERROR, &VARIABLE NOT FOUND " TO ERROR:MSG
RETURN
ENDIF (* variable = trim(&row) *)
SELECT PRIMARY
ENDIF
SKIP
ENDDO (* .not. eof *)
STORE $(QUERY,6) TO QUERY
SELECT SECONDARY
GO TOP
STORE FLD:NAME TO FL:NAME
DO REPLACE
IF ERROR
RETURN
ENDIF (* error *)
USE
SELECT PRIMARY
GO TOP
STORE FLD:NAME TO FILE:NAME
USE
SELECT PRIMARY
USE B:FL:NAME
SELECT SECONDARY
USE B:FILE:NAME
SELECT PRIMARY
SET EXACT ON
LOCATE FOR &I:QUERY
IF EOF .AND. .NOT. &I:QUERY
  STORE T TO ERROR
  STORE " *** ERROR, There are no records which meet ";
    the condition of &query" TO ERROR:MSG
RETURN
ENDIF (* eof .and. .not.&i:query *)
DO WHILE .NOT. EOF
  SELECT SECONDARY
  REPLACE ALL &STRING FOR &QUERY
  SELECT PRIMARY
  CONTINUE
ENDDO (* not. eof *)
SET EXACT OFF
SELECT SECONDARY
USE
SELECT PRIMARY
USE
RETURN

**************************************************************************
* MODATA command file**************************************************************************
* This program is invoked after the user finishes entering the *
* fields that are added to an existing file. This program *
* allows the user to modify the information that he just *
* entered. This program is called by CRTFILE command file.  *
**************************************************************************

SET TALK OFF
STORE T TO FLAG
DO WHILE FLAG
  STORE T TO MORE
  DO WHILE MORE
    STORE ' ' TO NUMBER
    SET COLON ON
    @ 23,1 SAY "PLEASE ENTER THE FIELD NUMBER THAT YOU"+;
    " WANT TO MODIFY ==> " GET NUMBER
    READ
    SET COLON OFF
    STORE F TO MORE
    STORE VAL(NUMBER) TO NUM
    @ 23,1
    IF NUM > COUNTER .OR. NUM <= 0
      @ 23,1 SAY "** BAD ENTRY, PLEASE RE-ENTER **" +
      STORE 1 TO CNT
      DO WHILE CNT < 20
        STORE CNT+1 TO CNT
      ENDDO
    ENDDO
    STORE T TO MORE
  ENDDO
  ENDDO (* MORE *)
  @ 23,1 SAY
  RELEASE NUMBER
  GOTO NUM
  STORE ' ' TO FLDNAME
  STORE ' ' TO FLDTYPE
  STORE ' ' TO FLDLEN
  STORE ' ' TO FLDDEC
  @ 22,1 SAY "CHANGE " + FLDNAME + "TO " GET FLDNAME
  READ
  IF FLDNAME <> ' '
    REPLACE FLDNAME WITH FLDNAME
  ENDDF
  @ 22,35 SAY "CHANGE " + FLDTYPE + " TO " GET FLDTYPE
  READ
  IF FLDTYPE <> ' ' .AND. (!FLDTYPE) = 'C' .OR. (!FLDTYPE) = 'N')
    REPLACE FLDTYPE WITH FLDTYPE
  ENDDF
  @ 23,1 SAY "CHANGE " + STR(FLDLEN,3) +;
  " TO " GET FLDLEN ;
  PICTURE '999'
  READ
  IF FLDLEN <> ' '
    REPLACE FLDLEN WITH VAL(FLDLEN)
  ENDDF
  IF FLDTYPE = 'N'
    @ 23,35 SAY "CHANGE " + STR(FLDDEC,2) +;
    " TO " GET FLDDEC ;
    PICTURE '999'
  READ
  IF FLDDEC <> ' '
    REPLACE FLDDEC WITH VAL(FLDDDEC)
  ENDDF
ENDDF
@ 22,1
@ 23,1
STORE T TO YES
DO WHILE YES
   STORE ' ' TO ANS
   @ 23,1 SAY " DO YOU STILL WANT TO MODIFY ANYTHING, Y/N ?";
   GET ANS
   READ
   IF I(ANS) <> 'Y' .AND. I(ANS) <> 'N'
      LOOP
   ENDF
   STORE F TO YES
ENDDO
IF I(ANS) = 'N'
   STORE F TO FLAG
ENDF
ENDDO
RELEASE FLDNAME,FLDTYPE,FLDDEC,FLDLLEN,ANS,FLAG,MORE,YES,
RETURN

************************************************************************ MAX COMMAND FILE *****************************
* This program finds the largest value of a data item. This program is called by SCAN command file.
************************************************************************

GO TOP
STORE T TO FLAG
DO WHILE FLAG
   IF .NOT. *
      STORE &VAR:MAX TO TOTAL
      STORE F TO FLAG
   ENDF
   SKIP
ENDDO
RELEASE FLAG
GO TOP
DO WHILE .NOT. EOF
   IF .NOT. *
      IF TOTAL < &VAR:MAX
          STORE &VAR:MAX TO TOTAL
          ENDF (* maxi > var:max *)
      ENDF (* .not. eof *)
      SKIP
   ENDDO (* not eof *)
RETURN

************************************************************************ MIN command file *****************************
* This program finds the smallest value of a data item. This program is invoked by SCAN command file.
************************************************************************

GO TOP
STORE T TO FLAG
DO WHILE FLAG
   IF .NOT. *
      STORE &VAR:MIN TO TOTAL
     (SKIP
   ENDF
   SKIP
ENDDO (* not eof *)
RETURN
STORE F TO FLAG
ENDIF
SKIP
ENDDO
RELEASE FLAG
GO TOP
DO WHILE .NOT. EOF
   IF .NOT.*
      IF TOTAL > &VAR:MIN
         STORE &VAR:MIN TO TOTAL
       ENDF (* MIN > VAR:MIN *)
      ENDF (* .NOT. EOF *)
   SKIP
ENDDO (* .NOT. EOF *)
RETURN

************ MULTISERT command file ************
* This program handles the queries which are multiple records *
* and multiple table insertion. This program is called by *
* PROCINSERT command file.  

************
SELECT PRIMARY
USE B:&FILE
STORE FLD:NAME TO FILE:NAME
STORE 0 TO ROW
GO RECO 2
DO WHILE .NOT. EOF
   IF %(%ROW:No,1,1) = '-'
      STORE TRIM(%ROW:No) TO VARIABLE
      STORE FLD:NAME TO FIELD:NAME
   ELSE
      IF %(%ROW:No,1,1) = '
      SKIP
      LOOP
   ELSE
      STORE T TO ERROR
      STORE TRIM(%ROW:No) TO MSG
      STORE "*** ERROR, INVALID VARIABLE &MSG IN YOUR QUERY" TO ERROR:MSG
      RETURN
   ENDF (* %(%row:No,1,1) = '-' *)
   ENDF (* %(%row:num,1,1) = ' ' *)
SELECT SECONDARY
SET EXACT ON
IF FILE = 'SFILE1'
   USE B:SFII2
   STORE FLD:NAME TO FL:NAME
ELSE
   USE B:SFII1
   STORE FLD:NAME TO FL:NAME
ENDIF (* file = 'sfill1' *)
LOCATE FOR FLD:NAME = FIELD:NAME
IF EOF .AND. FLD:NAME <> FIELD:NAME
   STORE T TO ERROR
STORE "*** ERROR, field &FIELD:NAME can't be found in" +
" &fl:name " TO ERROR:MSG
RETURN
ENDIF (* eof .and. fld:name <> field:name *)
SET EXACT OFF
IF ROW = 0
STORE 0 TO K
DO WHILE K < 3 .AND. ROW = 0
STORE K+1 TO K
STORE 'ROW' + STR(K, 1) TO WHERE
IF &WHERE = 'I.'
  LOOP
ELSE
  SET EXACT ON
  LOCATE FOR FLD:NAME = FIELD:NAME
  SET EXACT OFF
  IF &WHERE = VARIABLE
    STORE K TO ROW
    LOOP
  ENDIF (* &where = variable *)
  STORE K+1 TO K
ENDIF (* &WHERE = 'I.' *)
ENDDO (* k < 3 .and. row > 0 *)
IF ROW = 0
STORE T TO ERROR
STORE "*** ERROR, &VARIABLE NOT FOUND " TO ERROR:MSG
RETURN
ENDIF (* row = 0 *)
release k
STORE FIELD:NAME TO MY:QUERY
ELSE
IF &WHERE = VARIABLE
STORE FIELD:NAME + ', ' + MY:QUERY TO MY:QUERY
ELSE
STORE T TO ERROR
STORE "*** ERROR, &VARIABLE NOT FOUND " TO ERROR:MSG
RETURN
ENDIF (* &where = variable *)
ENDDO (* row = 0 *)
SELECT PRIMARY
SKIP
ENDDO
SELECT SECONDARY
GO TOP
STORE ' ' TO ROW
STORE WHERE TO ROW
DO REPLACE
IF ERROR
  RETURN
ENDIF (* error *)
USE
SELECT PRIMARY
USE
USE B:&FL:NAME
IF I:QUERY = ''
**PROCONTENT command file ******************
* This program is invoked when one of the columns of user's *
* query is using arithmetic operators. This program changes *
* the column to a string that can be process by dBASEIII. This *
* program is called by SUPUPDATE command file.  

STORE TRIM($(&ROW: NO,3)) TO COLUMN
DO PCHECK
IF ERROR
  RETURN
ENDIF (* error *)
STORE T TO GO
STORE O TO INDEX
STORE FIELD: NAME - ' WITH ' TO STRING
DO WHILE GO
  STORE F TO DOWN
  STORE 1 TO K
  DO WHILE . NOT. DOWN
    IF K > LEN(COLUMN)
      STORE T TO DOWN
    ELSE
      IF $(COLUMN,K,1) = '(' . OR. $(COLUMN,K,1) = ')
        . OR.
        $(COLUMN,K,1) = '*' . OR. $(COLUMN,K,1) = '/'
        . OR.
        $(COLUMN,K,1) = '-'
      . OR. $(COLUMN,K,1) = '+'
      STORE T TO DOWN
    ELSE
      STORE 1+K TO K
    ENDIF
  ENDDO (* k = len(column *)
ENDDO (* . not. down *)
RELEASE DOWN
IF K-1 > 1
  STORE $(COLUMN,1,K-1) TO CONTENT
  IF '-' CONTENT
    IF (&ROW) = CONTENT
      STORE STRING+FIELD:NAME TO STRING
    ELSE
      STORE T TO ERROR
      STORE " ** ERROR, INVALID QUERY IN &ROW" TO;
      ERROR: MSG
      RETURN
    ENDIF (* (&row) = content *)
  ENDIF
ELSE
  STORE 1 TO POSITION
  DO WHILE POSITION <= K - 1
    IF $(CONTENT, POSITION, 1) >= '0'.AND.
       $(CONTENT, POSITION, 1) <= '9'
      STORE 1+POSITION TO POSITION
    ELSE
      STORE T TO ERROR
      STORE " *** ERROR, NOT NUMERICAL TYPE "
      TO ERROR:MSG
      RETURN
    ENDIF
  ENDDO (* position <= k-1 *)
  STORE 1+INDEX TO INDEX
  STORE 'M'+STR(INDEX, 1) TO M
  STORE STRING + '&M' TO STRING
  STORE VAL(CONTENT) TO &M
ENDIF (*
IF K > LEN(COLUMN)
  STORE F TO GO
ELSE
  STORE $(COLUMN, K) TO COLUMN
ENDIF (* k > len(column) *)
ELSE
  STORE $(COLUMN, 1, 1) TO CONTENT
  IF '!'$(CONTENT .OR. '/$(CONTENT .OR. '+'$(CONTENT .OR.
     '!'$(CONTENT .OR. '('$(CONTENT .OR. '!')'$(CONTENT
  STORE STRING=CONTENT TO STRING
  IF LEN(COLUMN) = 1
    STORE F TO GO
  ELSE
    STORE $(COLUMN, 2) TO COLUMN
  ENDIF (* len = 1 *)
ELSE
  IF CONTENT >= '0' .AND. CONTENT <= '9'
    STORE 1+INDEX TO INDEX
    STORE 'M'+STR(INDEX, 1) TO M
    STORE VAL(CONTENT) TO &M
    STORE STRING+'&M' TO STRING
    IF LEN(COLUMN) = 1
      STORE F TO GO
    ELSE
      STORE $(COLUMN, 2) TO COLUMN
    ENDIF (* len(column) = 1 *)
  ELSE
    STORE T TO ERROR
    STORE " *** ERROR, NOT NUMERICAL TYPE "
    TO ERROR:MSG
    RETURN
  ENDIF (* content >= '0' .and. content <= '9' *)
ENDIF (* content >= '0' .and. content <= '9' *)
ENDIF (*
ENDDO
******************** PROCUPDATE command file ********************
* This program scans user's query and locates pointers to the row that has the update operator. This program is called by AVTABLE command file.

STORE 0 TO I
STORE F TO ACCOMPLISH
STORE 'SFFILE1' TO FILE
USE
DO WHILE I < 3
  STORE I+1 TO I
  STORE 'ROW'+STR(I,1) TO ROW:0
  USE B:FILE
  STORE F TO STOP
  DO WHILE .NOT. EOF .AND. .NOT. STOP
    STORE @('.',&ROW:0) TO FOUND:AT
    IF FOUND:AT = 2 .AND. $(&ROW:0,1,1) = 'U'
      STORE FLD:NAME TO FIELD:NAME
      DO UPDATE
      IF ERROR
        RETURN
      ENDIF (* error *)
      STORE T TO STOP
  ELSE
    SKIP
  ENDIF (* found:at = 2 .and. $(&row:0,1,1) = 'u' *)
  ENDDO (* .not. eof .and. .not. stop *)
IF I = 3 .AND. FILE:NUM > 1 .AND. .NOT. ACCOMPLISH
STORE 'SFFILE2' TO FILE
STORE T TO ACCOMPLISH
ENDIF

******************** PQUERY COMMAND FILE ********************
* This program is used to check a column of query which is a constant example and see if it is a valid value. This program is called by REPLACE, ARTHOPER, AND DELETION command file.

STORE LEN(PART) TO WORD:LEN
IF $(PART,1,1) = '=' .OR. $(PART,1,1) = '>' .OR. $(PART,1,1) = '<' .OR. $(PART,1,1) = '0' .AND. $(PART,1,1) <= '9') .OR. $(PART,1,1) >='A' .AND. $(PART,1,1) <= 'Z'
IF $(PART,1,1) = '>' .OR. $(PART,1,1) = '<'
IF $(PART,2,1) = '=' .OR. $(PART,2,1) = '>'
IF $(PART,1,2) = '=' .OR. $(PART,1,2) = '<'
STORE 3 TO LOCATION
STORE T TO BLANK
DO WHILE BLANK .AND. LOCATION <= WORD:LEN
  IF $(PART,LOCATION,1) = ' '
    STORE LOCATION + 1 TO LOCATION
  ELSE
    STORE F TO BLANK
  ENDF
ENDIF (* blank .and. location *)
IF LOCATION > WORD:LEN
STORE T TO ERROR
STORE " *** SYNTAX ERROR, NOTHING AFTER OPERATOR " ;
TO ERROR:MSG
RETURN
ENDIF(* location > length *)
RELEASE BLANK
ELSE
STORE T TO ERROR
ENDIF
ELSE
IF $(PART,2,1) = ' ' .OR. $(PART,2,1) >= '0' .AND. ;
$(PART,2,1) <= '9') .OR. $(PART,2,1) >= 'A' .AND. ;
$(PART,2,1) <= 'Z')
IF $(PART,2,1) = ' ' 
STORE 2 TO LOCATION
ELSE
STORE 2 TO LOCATION
STORE T TO BLANK
DO WHILE BLANK .AND. LOCATION <= WORD:LEN
IF $(PART,LOCATION,1) = ' '
STORE LOCATION+1 TO LOCATION
ELSE
STORE F TO BLANK
ENDIF(* $(part,location,1) = ' ' *)
ENDIF(* $(part,2,1) = ' ' *)
ELSE
STORE T TO ERROR
STORE " *** ERROR, ILLEGAL SYMBOL IN YOUR QUERY ";
TO ERROR:MSG
RETURN
ENDIF(* $(part,2,1) = ' ' .OR. $(part,2,1) >= '0' .. *)
ENDIF(* $(part,2,1) = ' ' .OR. $(part,2,1) = '>' .. *)
ELSE
IF $(PART,1,1) = ' ='
STORE 2 TO LOCATION
STORE T TO BLANK
DO WHILE BLANK .AND. LOCATION <= WORD:LEN
IF $(PART,LOCATION,1) = ' '
STORE LOCATION+1 TO LOCATION
ELSE
STORE F TO BLANK
ENDIF(* $(part,location,1) = ' ')
ENDIF(* $(part,2,1) = ' ' *)
ENDIF(* $(part,1,1) = ' >' .OR. $(part,1,1) = ' >' *)
ELSE
STORE T TO ERROR
ENDIF(* end of the if statement *)
IF ERROR
    STORE "*** ERROR, INVALID OPERATION OR SYMBOL IN YOUR QUERY";
    TO ERROR:MSG
    RETURN
ENDIF (* error *)
STORE TRIM(FLD:NAME) TO FIELD:NAME
IF TYPE = 'N'
    IF LOCATION = 1
        STORE T TO VARIABLE
    ELSE
        STORE F TO VARIABLE
    ENDIF(* location = 1 *)
STORE LEN(PART) TO WORD:LEN
DO WHILE LOCATION <= WORD:LEN
    IF ($(PART,LOCATION,1) >= '0' .AND. $(PART,LOCATION,1); <= '9') .OR. $(PART,LOCATION,1) = ' '
        STORE LOCATION+1 TO LOCATION
    ELSE
        STORE T TO ERROR
        STORE "*** ERROR, INVALID TYPE" TO ERROR:MSG
        RETURN
    ENDIF
ENDDO (* location <= length *)
IF VARIABLE
    STORE TEMP+'&FIELD:NAME = &PART' TO TEMP
ELSE
    STORE TEMP+'&FIELD:NAME &PART' TO TEMP
ENDIF (* variable *)
ELSE
    STORE $(PART,1,LOCATION-1) TO FIRST
    STORE $(PART,LOCATION) TO SECOND
    STORE LEN(SECOND) TO WORD:LEN
    STORE 1 TO K
    DO WHILE K <= WORD:LEN
        IF ($(SECOND,K,1) >= '0' .AND. $(SECOND,K,1) <= '9')
        .OR. ($(SECOND,K,1) >= 'A' .AND. $(SECOND,K,1) <= 'Z')
        .OR. $(SECOND,K,1) = ' '
            STORE K+1 TO K
        ELSE
            STORE T TO ERROR
            STORE "*** ERROR, ILLEGAL TYPE IN YOUR QUERY";
            TO ERROR:MSG
            RETURN
        ENDIF (* k <= length *)
    ENDDO (* k <= length *)
RELEASE K , WORD:LEN
IF LOCATION = 1
    STORE TEMP + FIELD:NAME + '="" + SECOND + '"" TO TEMP
ELSE
    STORE TEMP+FIELD:NAME+FIRST+'""+SECOND+'"" TO TEMP
ENDIF (* location = 1 *)
RELEASE FIRST,SECOND
ENDIF
* ENDIF (*
ENDIF (*
ENDIF (*

123
RETURN

*************** PARSE1 command file ***************
* This program, which is called by RUN command file, parses the *
* user's query to determine what kind of operation that the user*
* desires.
*
*******(read the rest of the code)*******/

@ 23,1 SAY 'RUNNING, PLEASE WAIT A MINUTE!'
STORE F TO ERROR,FINISHED,OUTPUT,UNION,JOIN,BUILDFUN
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 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 command' TO ERROR:MSG
STORE T TO ERROR
RETURN
ENDIF
ENDIF
ENDIF
IF R:EXIST
IF OUTPUT
STORE '*** ERROR - too many P. operator' TO ERROR:MSG
STORE T TO ERROR
RETURN
ELSE
DO MAKEOUT2
IF ERROR
RETURN
ENDIF
ENDIF
ENDIF
DO AVTABLE
IF FINISHED
RETURN
ENDIF
IF .NOT. OUTPUT
STORE '*** ERROR - command 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 O TO I
DO WHILE I < FILE:NUM
STORE I + 1 TO I
STORE 'FILE' + STR(I,1) TO FILE
STORE &FILE TO FILE
STORE 'SFILE' + STR(I,1) TO SFILE
USE B:&FILE
COPY TO B:&SFILE
ENDDO
USE
IF C:EXIST
DO SCANCOND
ENDIF
DO SCANAV1
IF ERROR
RETURN
ENDIF
IF MAX:AV > 0
DO SCANAV2
IF ERROR2
RETURN
ENDIF
125
ENDIF
IF MAX:AV > 0
  DO SCANAV3
  IF ERROR
    RETURN
  ENDDIF
ENDIF
DO RETRIE1
  IF ERROR
    RETURN
  ENDDIF
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
  ENDDIF
ENDIF
DO DISPANS
RETURN

*************************** PROCINSERT command file ***************************
* This program processes the queries which are the insert operation and then invokes a proper subroutine depends on the characteristic of the query, such as single record insertion or multiple records and tables insertion. This program is invoked by DOCOM command file. *

USE B:SFILE1
GO TOP
STORE FLD:NAME TO FILE:NAME
STORE ' ' TO MY:QUERY
STORE 1 TO I
STORE F TO PROCESS
DO WHILE I <= 3
  STORE 'ROW'+STR(I,1) TO ROW:NO
  IF .NOT. (.&ROW:NO = 'I.' .OR. &ROW:NO = '' )
    STORE T TO ERROR
    STORE " *** ERROR, YOU CAN ONLY HAVE ONE QUERY EACH TIME ";
    TO ERROR:MSG
    RETURN
  ENDDIF (* .not. (.&row:no = 'i.' .or. &row:no = '' )
  STORE 1+I TO I
  IF I= 4
    IF FILE:NUM = 2 .AND. .NOT. PROCESS
      USE B:SFILE2
      STORE T TO PROCESS
      STORE 1 TO I
    ELSE
USE B:SF1
   ENDF (* file:num = 2 .and. .not. process *)
   ENDF (* i = 4 *)
ENDDO (* * i <= 3 *)

*** check if user wants to have single record insertion ***

STORE 0 TO I
STORE 'SF1' TO FILE
USE B:&FILE
GO TOP
IF ROW1 = 'I.'
   SKIP
   IF ROW1 = 'C.'
      DO SRECORD
      RETURN
   ENDF (* row1 = 'c.' *)
ELSE
   USE
ENDF (* row1 = 'i.' *)

**** process user's query ****

STORE 'SF1' TO FILE
STORE F TO ACCOMPLISH
STORE T TO MULTIPLE
STORE 0 TO I
DO WHILE I < 3
   USE B:&FILE
   STORE 1+I TO I
   STORE 'ROW'+STR(I,1) TO ROW:NO
   GO TOP
   IF &ROW:NO = ''
      STORE I+1 TO I
   LOOP
ELSE
   GO RECORD 2
   DO WHILE .NOT. EOF
      IF .NOT. '_$(&ROW:NO)
         STORE F TO MULTIPLE
      ENDF (* .not. '_$(&row:no) *)
      SKIP
   ENDDO (* .not. eof *)
ENDF (* &row:no = ' ' *)
IF MULTIPLE
   USE
   IF FILE:NUM = 1
      STORE T TO ERROR
      STORE " *** ERROR, INVALID QUERY " TO ERROR:MSG
      RETURN
   ENDF (* file:num = 1 *)
   DO MULTINSERT
   IF ERROR
      RETURN
   ENDF (* error *)
ELSE
USE
DO STABLE
IF ERROR
RETURN
ENDIF
ENDIF
STORE T TO MULTIPLE
IF I = 3 .AND. .NOT. ACCOMPLISH .AND. FILE:NUM > 1
STORE 'SPFILE2' TO FILE
STORE T TO ACCOMPLISH
STORE 0 TO INDX
STORE T TO MULTIPLE
ENDIF (* indx = 3 .and. .not. accomplish .and. file: num > 1 *)
ENDDO (* i < 3 *)

********************************************************************** PROCFILE command file **********************************************************************
* This program will process the user's query which is concerned *
* only with file's management, such as displaying all *
* the files names existing in the user's disk, displaying the *
* structure of a specific file, deleting a file, and creating *
* a new file. This program is called by RUN command 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 'I'
READ
DO CASE
CASE 1(ANS) = 'Y'
DELETE FILE B: &FNNAME
ERASE
? ' &FNNAME IS DELETED '
RELEASE ANS,FNNAME
RETURN
CASE 1(ANS) = 'N'
ERASE
ENDCASE
ENDDO (* COMMAND *)
SET EXACT OFF

128
*************** PURGEBLANK command file *******************
* This program, which is invoked by several procedures,    *
* all the unnecessary blanks from a string.               *
**********************************************************

STORE LEN(UNPURGED) TO MAX:LEN
STORE LEN(QUNEURGED) TO MAX:LEN
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

*************** QBE command file ***********************
* This program is used to implement the query language Query-By-
* Example. The program will display table skeletons to let *
* users enter their queries and then display the answer on the*
* screen as well as on the hard copy.                     *
**********************************************************

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
         @ ROW,5 SAY 'PLEASE ENTER TODAY'S DATE (MM/DD/YY):'
         GET DAT PICTURE "99/99/99"
      ENDW
STORE "FIELD"+STR(NUMB,1) TO FIELD:NUMB
STORE &FIELD:NUMB TO NAM
& ROW, 13 GET &FLD:NUMB
READ
STORE LEN(&FLD:NUMB) TO LEN
STORE T TO OK
STORE &FIELD:NUMB TO RECORD
DO CASE
  CASE TYPE(&RECORD) = 'N'
    STORE 1 TO I
    DO WHILE I <= LEN
      IF $(&FLD:NUMB,1,1) < '0' .OR. ;
        $(&FLD:NUMB,1,1) > '9'
        STORE F TO OK
        @ 23,1 SAY "**** INVALID TYPE ****"
        STORE LEN TO I
        STORE $(BLANK,1,&FLD:LEN) TO &FLD:NUMB
    ENDF
    STORE I+1 TO I
  ENDDO (* I <= LEN*)
  IF .NOT. OK
    LOOP
  ENDF (* OK=F *)
  REPLACE &NAM WITH VAL(&FLD:NUMB)
  CASE TYPE(&RECORD) = 'C'
    REPLACE &NAM WITH !(&FLD:NUMB)
ENDCASE (* *)
STORE F TO OK
STORE NUMB+1 TO NUMB
STORE ROW+1 TO ROW
ENDDO (* NOT OK *)
ENDDO(* ROW < MAXROW *)
SET COLON OFF
RELEASE ROW, MAX:ROW, NUMB, LEN, OK, BLANK
STORE T TO OK
DO WHILE OK
  STORE ' ' TO ANS
  @ 23,1 SAY " DO YOU HAVE MORE INPUT DATA ? (Y/N)";
  GET ANS PICT '1'
  READ
  IF ANS <> "Y" .AND. ANS <> "N"
    LOOP
  ELSE
    IF ANS = "N"
      STORE F TO OK
      STORE F TO MORE
  ENDF
ENDIF
ENDDO
STORE F TO OK
ENDDO (* OK *)
ENDDO (* MORE *)
RETURN
STORE I TO CURRENT
STORE FILE:NAME TO FNAME
STORE @ TO RCD:NO
STORE FLD:NAME TO F:NAME
STORE TYPE TO FLD:TYPE
STORE F TO FOUND
STORE F TO STOP
STORE F TO PROCESS
DO WHILE .NOT. FOUND .AND. .NOT. STOP
   GO TOP
   STORE 0 TO J
   DO WHILE J < 3 .AND. .NOT. FOUND
      STORE J+1 TO J
      STORE 'ROW' + STR(J,1) TO ROW:NO
      IF &ROW:NO = ' '
         LOCATE FOR FLD:NAME = '&F:NAME'
      IF EOF .AND. FLD:NAME <> '&FIELD:NAME'
         LOOP
      ELSE
         IF &ROW:NO = PART
            STORE T TO PROCESS
            DO REPLC
               IF ERROR
                  RETURN
               ENDIF
            STORE T TO FOUND
            LOOP
         ENDIF (* row:no = part *)
      ENDIF
      ELSE (*) &row:no = ' ' *)
      ENDDO (*) J < 3 *)
   IF FOUND
      LOOP
   ENDDO (* found *)
   IF FILE:NUM > 1 .AND. .NOT. PROCESS
      USE B:FILE2
   STORE 0 TO J
   STORE T TO PROCESS
   GO TOP
   STORE FLD:NAME TO FNAME
   ELSE
      STORE T TO STOP
   ENDDF (* file:no > 1 .AND. .NOT. PROCESS *)
   ENDDO (*)
   IF .NOT. FOUND
STORE T TO ERROR
STORE "*** ERROR, INCOMPLETE QUERY " TO ERROR:MSG
RETURN
ENDIF (* error *)
USE B:&FNAME
GO BOTTOM
STORE # TO REC:NO
GO TOP
SET EXACT ON
LOCATE FOR (&I:QUERY)
IF EOF
  IF .NOT. &I:QUERY
    STORE " *** ERROR, NO RECORD MEETS ALL CONDITION OF THE"+;
    " QUERY" TO ERROR:MSG
    STORE T TO ERROR
    RETURN
  ENDIF (* .not. &my:query *)
ENDIF (* # = rec:no *)
STORE ' ' TO TEMP
LOCATE FOR (&I:QUERY)
DO WHILE .NOT. EOF
  IF FLD:TYPE = 'N'
    STORE &F:NAME TO CONTENT
    IF FLD:DEC > 0
      STORE STR(CONTENT,FLD:LEN,FLD:DEC) TO MVALUE
    ELSE
      STORE STR(CONTENT,FLD:LEN) TO MVALUE
    ENDIF
    IF E
      STORE TEMP-'.' OR '.'-F:NAME-SIGN-MVALUE TO TEMP
    ELSE
      STORE TEMP-'.' OR '.'- 'F:NAME = &MVALUE' TO TEMP
    ENDIF
  ELSE
    STORE TRIM(&F:NAME) TO CONTENT
    STORE TEMP-'.' OR '.'- F:NAME - " = " - CONTENT - "" TO TEMP
  ENDIF (* type = 'n' *)
  CONTINUE
ENDDO (* .not. eof *)
SET EXACT OFF
USE B:SPFILE1
STORE '('+$(TEMP,5)-')' TO PART
RELEASE TEMP
STORE 'ROW'+STR(CURRENT,1) TO ROW:NO
GO RCD:NO
STORE CURRENT TO I
RETURN

*****************************************************************************
* This program retrieves information from data base and returns *
* the information to the calling program. This program is called*  
* by STABLE command file.                                    *
*****************************************************************************

STORE FIELD:NAME TO F:NAME
DO FINDROW
IF ERROR
  RETURN
ENDIF (* error *)

DO REPLACE
IF ERROR
  RETURN
ENDIF (* error *)
IF i:QUERY = ""
  STORE T TO ERROR
  STORE " NO CONDITION HAS BEEN SPECIFIED FOR &VARIABLE ";
  TO ERROR:MSG
  RETURN
ENDIF (* i:query = ' ' *)
STORE F:NAME TO FIELD:NAME
USE B:&CRTFILE
SET EXACT ON
LOCATE FOR (&I:QUERY)
IF EOF .AND. .NOT. (&I:QUERY)
  STORE T TO ERROR
  STORE '*** ERROR, There is no records which meet the';
  ' condition of &i:query' TO ERROR:MSG
  RETURN
ENDIF (* eof .and. .not. (&i:query) *)
STORE &FIELD:NAME TO CONTENT
SET EXACT OFF
USE B:&FILE
GO REC:NUM
IF TYPE = 'N'
  IF DEC > 0
    REPLACE &ROW:NO WITH STR(CONTENT,FLD:LEN,DECIMAL)
    STORE ' ' TO CONTENT
    STORE TRIM(&ROW:NO) TO CONTENT
  ELSE
    REPLACE &ROW:NO WITH STR(CONTENT,FLD:LEN)
    STORE ' ' TO CONTENT
    STORE TRIM(&ROW:NO) TO CONTENT
  ENDIF (* decimal > 0 *)
ELSE
  REPLACE &ROW:NO WITH CONTENT
ENDIF (* type = 'n' *)
RETURN

*************************************************************************
* This program, which is called by QBE command file, 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

133
DO PARSER
IF ERROR
   @ 23,1
   @ 23,1 SAY ERROR:MSG
   ACCEPT 'Hit <RETURN> to continue ' TO ACTION
   RELEASE ACTION
   ERASE
ENDIF (* error *)
ENDIF (* file:only *)
CLEAR
STORE T TO FLAG
DO WHILE FLAG
   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
   DO CASE
      CASE !(ANS) = 'Q'
         ERASE
         @ 10,30 SAY 'G O O D B Y E !'
         @ 22,0 QUIT
      CASE !(ANS) = 'C'
         ERASE
         CLEAR
         STORE T TO MORE
         RETURN
      OTHERWISE
         LOOP
   ENDCASE
   ENDDO (* flag *)

*************** RENAME COMMAND FILE ********************************
* This program renames the old file name to new file name. This *
* program is invoked by PROCFILE command file.                  *

*************** RENAME COMMAND FILE ********************************

SET COLON ON
STORE ' ' TO NEW:NAME
@ 23,1 SAY " PLEASE ENTER THE NEW FILE NAME THAT YOU WANT TO"+;
    " REPLACE '&FNAME'" GET NEW:NAME
READ
SET COLON OFF
@ 23,1
STORE 1 TO J
STORE TRIM(NEW:NAME) TO NEW:NAME
STORE LEN(NEW:NAME) TO NAME:LEN
DO WHILE $(NEW:NAME,J,1) = ' ' .AND. J <= NAME:LEN
    STORE J+1 TO J
ENDDO (* $(fname,j,1) = ' ' .and. j <= name:len *)
IF J = NAME:LEN .AND. $(NEW:NAME,J,1) = ' ' 
    STORE F TO OK
    STORE " *** ERROR, MISSING NEW NAME " TO ERROR:MSG
    RETURN
ELSE
    STORE $(NEW:NAME,J) TO NEW:NAME
ENDIF (* j =name:ilen .and. $(fname,j,1) = ' ' *)
STORE FNAME TO OLD:NAME
STORE NEW:NAME TO FNAME
RELEASE NEW:NAME
IF FILE('B:&FNAME')
    @ 23,1
    STORE T TO KEEP:GOING
DO WHILE KEEP:GOING
    STORE ' ' TO ANSWER
    @ 23,1 SAY " &FNAME is already exist, do you want to"+
    " overwrite it, y/n?" GET ANSWER PICT 'I'
    READ
    IF ANSWER = 'Y' .OR. ANSWER = 'N'
        STORE F TO KEEP:GOING
    ENDIF (* answer = 'y' .and. answer = 'n' *)
ENDDO (* keep:going *)
    @ 23,1
    IF ANSWER = 'N'
        RETURN
    ENDIF (* answer *)
ENDIF (* file('b:&fname') *)
RENAME B:&OLD:NAME TO B:&FNAME
RETURN

****************************** REPLACE command file ******************************
* This program converts the condition for an example element *
* to a dBASE-II command statement. This program is invoked by *
* MULTINSERT, QURYDEP, ARTHDEP, AND UPDEPEND command file. *

STORE ' ' TO I:QUERY
GO RECORD 2
DO WHILE .NOT. EOF
    IF '_'&ROW .OR. &ROW = ' '
        SKIP
    ELSE
        STORE @(',',&ROW) TO FOUND:AT
        STORE TRIM(&ROW) TO COLUM
        IF FOUND:AT >= 4
            STORE FOUND:AT - 3 TO MVAL
            IF $(COLUMN,MVAL,3) = 'SUM' .OR. $(COLUMN,MVAL,3) = ;
                'CUT' .OR. $(COLUMN,MVAL,3) = 'AVG' .OR. ;
                $(COLUMN,MVAL,3) = 'MAX' .OR. $(COLUMN,MVAL,3) = 'MIN'
            STORE FILE TO T:FILE
            STORE # TO REC:NUM
            DO CHKFUN

135
IF ERROR
RETURN
ENDIF (* error *)
STORE I:QUERY+.AND.'+CONTENT TO I:QUERY
STORE T:FILE TO FILE
USE B:&FILE
GO RECO REC:NUM
RELEASE REC:NUM,T:FILE
SKIP
LOOP
ENDIF
ENDIF (* found:at > 0 .and. found:at > = 4 *)
IF '+'$(&ROW) .OR. '*'$(&ROW) .OR. '/'$(&ROW) .OR.
'-'$(&ROW)
STORE T TO ERROR
STORE "*** ERROR, ARITHMETIC OPERATIONS ARE NOT"+;
" ALLOWED" TO ERROR:MSG
RETURN
ELSE
IF '!$(&ROW) .OR. '&$(&ROW)
STORE TRIM(&ROW) TO COLUMN
DO LOGQUERY
STORE I:QUERY -= .AND.'-TEMP TO I:QUERY
IF LEN(I:QUERY) > 130
STORE T TO ERROR
STORE " *** ERROR, CONDITION STRING IS TOO LONG";
TO ERROR:MSG
RETURN
ENDIF (* len(i:query) > 130 *)
ELSE
STORE &ROW TO PART
STORE ' ' TO TEMP
DO PQQUERY
STORE I:QUERY-'.'AND.'-TEMP TO I:QUERY
IF LEN(I:QUERY) > 130
STORE T TO ERROR
STORE " *** ERROR, CONDITION STRING IS TOO LONG";
TO ERROR:MSG
RETURN
ENDIF (* len(i:query) > 130 *)
ENDIF (* '!$(&row .or. '&$(&row) *)
SKIP
ENDIF (* '+'$(&row) .or. '*'$(&row) .or. ... *)
ENDIF (* '_$(&row *)
ENDDO (* .not. eof *)
IF .NOT. (I:QUERY = ' ')
STORE $(I:QUERY,7) TO I:QUERY
ENDIF (* .not. i:query = ' ' *)
RETURN

*************** REPLC COMMAND FILE ****************************
* This program gets the condition for an example element. This *
* program is called by QUERYIND command file. *******
**STORE** ' ' TO I:QUERY
**GO** RECORD 2
**DO** WHILE .NOT. EOF
   **IF** '_'$('&ROW: NO .OR. &ROW: NO = ' ')
      **SKIP**
   **ELSE**
      **IF** '+'$('&ROW: NO .OR. ' '*$('&ROW: NO .OR. '/$('&ROW: NO);
         .OR. ' '-'$('&ROW: NO)
         **STORE** T TO ERROR
         **STORE** " *** ERROR, INVALID QUERY " TO ERROR: MSG
         **RETURN**
   **ELSE**
      **IF** '! $('&ROW: NO .OR. ' &$('&ROW: NO)
         **DO** LOGQUERY
         **STORE** I:QUERY - '. AND.'-TEMP TO I:QUERY
         **IF** LEN(I:QUERY) > 130
            **STORE** T TO ERROR
            **STORE** " *** ERROR, condition string is too long ";
            TO ERROR: MSG
            **RETURN**
         **ENDIF** (* len(i:query *)
      **ELSE**
         **STORE** &ROW: NO TO PART
         **STORE** ' ' TO TEMP
         **DO** QUERY
         **STORE** I:QUERY - '. AND.'-TEMP TO I:QUERY
         **IF** LEN(I:QUERY) > 130
            **STORE** T TO ERROR
            **STORE** " *** ERROR, condition string is too long ";
            TO ERROR: MSG
            **RETURN**
         **ENDIF** (* len(i:query) > 130 *)
      **ENDIF** (* '!'$('&ROW: NO .OR. ' &$('&ROW: NO) *)
      **SKIP**
      **ENDIF** (* '+'$('&ROW: NO .OR. ' '*$('&ROW: NO .OR. ... *)
   **ENDIF** (* '_'$('&ROW: NO *)
**ENDDO** (* .not. eof *)
**STORE** $(I:QUERY, 7) TO I:QUERY
**RETURN**

******************************************************************************
* This program scans the query entered by the user and *
* determines which built-in operation is desired. Then, it *
* makes a proper subroutine call. This program is called by *
* STABLE, and BULDFUN command file.*
******************************************************************************

**GO** TOP
**STORE** FLD: NAME TO FILE: NAME
**STORE** $(COLUMN, 5) TO COLUMN
**STORE** $( ' ', COLUMN) TO FOUND: AT
**IF** FOUND: AT < > 4
   **STORE** T TO ERROR
   **STORE** " *** ERROR, INVALID QUERY " TO ERROR: MSG
**RETURN**
ELSE
  IF LEN(COLUMN) = 4
    IF $(COLUMN,1,3) = 'ALL'
      STORE F TO UNIQUE
    ELSE
      IF $(COLUMN,1,3) = 'UNQ'
        STORE T TO UNIQUE
        IF CMD = 'MIN' .OR. CMD = 'MAX'
          STORE T TO ERROR
          STORE " *** ERROR, INVALID QUERY ";
          TO ERROR:MSG
          RETURN
        ENDF ( * cmd = 'min' .or. cmd = 'max' * )
      ELSE
        STORE T TO ERROR
        STORE " *** ERROR, INVALID QUERY " TO ERROR:MSG
        RETURN
      ENDF ( * $column,1,3) = 'unq' * )
    ENDF ( * $(column,1,3) = 'all' * )
  USE B:&FILE:NAME
  DO CASE
    CASE CMD = 'AVG'
      STORE FIELD:NAME TO VAR:AVG
      DO AVG
    CASE CMD = 'SUM'
      IF UNIQUE
        STORE FIELD:NAME TO VAR:AVG
        DO AVG
      ELSE
        SUM &FIELD:NAME TO TOTAL
      ENDF ( * unique * )
    CASE CMD = 'CNT'
      STORE FIELD:NAME TO VAR:CNT
      DO CNT
    CASE CMD = 'MIN'
      STORE FIELD:NAME TO VAR:MIN
      DO MIN
    CASE CMD = 'MAX'
      STORE FIELD:NAME TO VAR:MAX
      DO MAX
    OTHERWISE
      STORE T TO ERROR
      STORE " *** ERROR, INVALID BUILT-IN FUNCTION ";
      TO ERROR:MSG
      RETURN
  ENDCASE
ELSE
  IF $(COLUMN,1,3) = 'ALL' .AND. '_'$COLUMN
    STORE F TO UNIQUE
  ELSE
    IF $(COLUMN,1,3) = 'UNQ' .AND. '_'$COLUMN
      STORE T TO UNIQUE
      IF CMD = 'MIN' .OR. CMD = 'MAX'
        STORE T TO ERROR
        STORE " *** ERROR, INVALID BUILT-IN FUNCTION ";
      ELSE
        STORE T TO UNIQUE
      ENDF ( * cmd = 'min' .or. cmd = 'max' * )
  ENDF ( * $(column,1,3) = 'all' * )
ENDIF ( * $column,1,3) = 'unq' * )

TO ERROR:MSG
RETURN
ENDIF (* cmd = 'min' .or. cmd = 'max' *)
ELSE
STORE T TO ERROR
STORE " *** ERROR, INVALID BUILT-IN FUNCTION";
TO ERROR:MSG
RETURN
ENDIF
ENDIF
STORE $(COLUMN,5) TO COLUMN
STORE T TO BULD:IN
DO ARTHOPER
IF ERROR
RETURN
ENDIF (* error *)
ENDIF (*
return

***************************************************************************** SCANCND command file*****************************************************************************
* This program is invoked by the DOCOM program when a condition box is used by the user. Basically the program checks if the * variables in condition box are also in general table, if so, * the variable in the general table is replaced with the value * which is specified in the condition box.
*****************************************************************************
USE B:CONDITION
DO WHILE .NOT. EOF
  GO TOP
  STORE TRIM(CONDITION) TO COND:STR
  DELETE
  PACK
  IF $(COND:STR,1,1) <> '_'
    STORE T TO ERROR
    STORE " *** ERROR, INVALID QUERY " TO ERROR:MSG
  RETURN
ELSE
  STORE F TO OK
  STORE 1 TO K
  DO WHILE .NOT. OK
    IF $(COND:STR,K,1)='.' .OR. $(COND:STR,K,1) = '<'. .OR.;
       $(COND:STR,K,1) = '>'
      STORE T TO OK
      STORE TRIM($(COND:STR,1,K-1)) TO VARIABLE
      IF $(COND:STR,K,1) = '='
      STORE K+1 TO K
      STORE T TO Y
      DO WHILE K <= LEN(COND:STR) .AND. Y
        IF $(COND:STR,K,1) = '
          STORE K+1 TO K
        ELSE
          STORE F TO Y
      ENDIF (* $(cond: str, k, 1) = ' ' *)
ENDDO (* k <= len(part) .and. y *)
RELEASE Y
ENDIF (* $(cond:str,k,1) = '=' *)
ELSE
STORE K+1 TO K
ENDIF (* $(cond:str,k,1) <> '=' .or. ...... *)
ENDDO (* .not. ok *)
RELEASE OK
STORE $(COND:STR,K) TO MICRO
IF $(MICRO,1,1) = '
STORE MICRO TO UNPURGED
DO PURGEbla
STORE PURGED TO MICRO
ENDIF
STORE 1 TO I
STORE F TO STOP
STORE 'SFILE1' TO FILE
STORE F TO PROCESS
DO WHILE .NOT. STOP
USE B:FILE
STORE 0 TO J
DO WHILE J < 3
GO RECORD 2
STORE J+1 TO J
STORE 'ROW'+STR(J,1) TO ROW:NO
DO WHILE .NOT. EOF
IF &ROW:NO = '
  SKIP
ENDIF (* &row:no = ' '*
STORE &ROW:NO TO CONTENT
STORE $(VARIABLE,CONTENT) TO FOUND:AT
IF FOUND:AT = 0
  SKIP
ELSE
STORE 1 TO K
DO WHILE K+FOUND:AT-1 <= LEN(CONTENT)
  IF $(CONTENT,FOUND:AT,K) = VARIABLE
    STORE T TO FOUND
    IF FOUND:AT = 1 .AND. (FOUND:AT+K-1) = LEN(CONTENT)
      REPLACE &ROW:NO WITH MICRO
    ELSE
      IF FOUND:AT = 1 .AND. (FOUND:AT+K-1)>
        LEN(CONTENT)
      STORE $(CONTENT,(FOUND:AT+1)) TO SECOND
      STORE MICRO+SECOND TO CONTENT
      REPLACE &ROW:NO WITH CONTENT
    RELEASE SECOND
    ELSE
      IF FOUND:AT > 1 .AND. ;
        (FOUND:AT+K-1) = LEN(CONTENT)
      STORE $(CONTENT,1,FOUND:AT-1)
      TO FIRST
ENDIF
STORE FIRST+MICRO TO CONTENT
REPLACE &ROW:NO WITH CONTENT
RELEASE FIRST
ELSE
IF FOUND:AT > 1 .AND. ;
FOUND:AT+K-1 > LEN(CONTENT)
STORE $(CONTENT,FOUND:AT-1);
TO FIRST
STORE $(CONTENT,FOUND:AT+1);
TO SECOND
STORE FIRST+MICRO+SECOND;
TO CONTENT
REPLACE &ROW:NO WITH CONTENT
ENDIF
ENDIF
ENDIF
ENDIF(* found:at = 1 .and. ... *)
ENDIF
STORE K+1 TO K
ENDDO (* j+found:at < len(&fret:col) *)
SKIP
ENDIF (* found:at = 0 *)
ENDDO (* .not. eof *)
ENDDO(* j< 3 *)
IF FILE:NUM > 1 .AND. .NOT. PROCESS
STORE 'SFIL2' TO FILE
STORE T TO PROCESS
STORE 0 TO J
ELSE
STORE T TO STOP
ENDIF (* file:num > 1 *)
ENDDO (* .not. stop *)
release file,stop, process
ENDIF (* $(cond:str,1,1) <> ' ' *)
USE B:CONDITION
ENDDO (* .not. eof *)
RELEASE PURGED,UNPURGED,COND:STR
RETURN

*************** SCANFN command file ***************

This program, which is called by DTABLE command file, parses *
* the file name and command which are entered by the user and *
* checks if they are valid. If invalid, the program displays *
* an error message and asks 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 PUGEBLANK
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 ***
  IF .NOT. FILE('B:&FNAME')
    STORE '*** ERROR - file &fname does not exist' TO ;
    ERROR:MSG
    STORE F TO OK
  ENDIF
  RELEASE COMMAND
  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 command' TO ERROR:MSG
    STORE F TO OK
    RETURN
  ELSE
    IF FNAME = ' '
      STORE '*** ERROR - missing file name' TO ERROR:MSG
      STORE F TO OK
      RETURN
    ELSE
      IF .NOT. FILE('B:&FNAME')
        STORE '*** ERROR - file &FNAME does not exist';
        TO ERROR:MSG
        STORE F TO OK
        RETURN
      ENDIF (* not file exist *)
    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 - file name should be a variable '+';
        ' name' 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

143
GO RECORD 2
STORE 0 TO NO
DO WHILE .NOT. EOF
   STORE 1+NO TO NO
   STORE 'FIELD'+STR(NO,1) TO FIELD:NO
   STORE 'FLD:LEN'+STR(NO,1) TO FLD:LEN
   STORE 'DEC'+STR(NO,1) TO MDEC
   STORE FLD:NAME TO &FIELD:NO
   STORE LEN TO &FLD:LEN
   STORE DEC TO MDEC
   SKIP
ENDDO (* .not. eof *)
USE B:\file:name
STORE T TO MORE
DO WHILE MORE
   ERASE
   STORE " " TO BLANK
   STORE 1 TO NUMB
   DO WHILE NUMB <= NO
      STORE 'FLD'+STR(NUMB,1) TO FLD:NUMB
      STORE 'FLD:LEN'+STR(NUMB,1) TO FLD:LEN
      STORE $(BLANK,1,&FLD:LEN) TO &FLD:NUMB
      STORE NUMB + 1 TO NUMB
   ENDDO
   @ 3,5 SAY " PLEASE INSERT DATA IN THE FOLLOWING TABLE "
   @ 7,1 SAY "+-----------------------------+;
      "-----------------------------+
   STORE 8 TO ROW
   STORE 1 TO NUMB
   SET COLON ON
   STORE 8+NO TO MAX:ROW
   DO WHILE ROW < MAX:ROW
      STORE 'FIELD'+STR(NUMB,1) TO FIELD:NUMB
      @row,1 SAY "|" + &FIELD:NUMB
      @row,72 SAY "|"
      STORE NUMB + 1 TO NUMB
      STORE ROW+1 TO ROW
   ENDDO
   @ MAX:ROW,1 SAY "+-----------------------------+;
      "-----------------------------+
   STORE 8 TO ROW
   APPEND BLANK
   STORE 1 TO NUMB
   DO WHILE ROW < MAX:ROW
      STORE F TO OK
      DO WHILE .NOT. OK .AND. NUMB <= NO
         @ 23,1
         STORE "FLD"+STR(NUMB,1) TO FLD:NUMB
STORE "FIELD"+STR(NUMB, 1) TO FIELD:NUMB
STORE &FIELD:NUMB TO NAM
@ ROW, 13 GET &FLD:NUMB
READ
STORE LEN(&FLD:NUMB) TO LEN
STORE T TO OK
STORE &FIELD:NUMB TO RECORD
DO CASE
  CASE TYPE(&RECORD) = 'N'
    STORE 1 TO I
    DO WHILE I <= LEN
      IF $(&FLD:NUMB, 1, 1) < '0' .OR.;
      $(&FLD:NUMB, 1, 1) > '9'
      STORE F TO OK
      @ 23, 1 SAY " *** INVALID TYPE ***"
      STORE LEN TO I
      STORE $(BLANK, 1, &FLD:LEN) TO &FLD:NUMB
    ENDF
    STORE I+1 TO I
    ENDDO (* I <= LEN*)
    IF .NOT. OK
      LOOP
    ENDF (* OK=F *)
    REPLACE &NAM WITH VAL(&FLD:NUMB)
  CASE TYPE(&RECORD) = 'C'
    REPLACE &NAM WITH I(&FLD:NUMB)
  ENDCASE (* * *)
STORE F TO OK
STORE NUMB+1 TO NUMB
STORE ROW+1 TO ROW
ENDDO (* NOT OK *)
ENDDO (* ROW < MAXROW *)
SET COLON OFF
RELEASE ROW, MAX:ROW, NUMB, LEN, OK, BLANK
STORE T TO OK
DO WHILE OK
  STORE ' ' TO ANS
  @ 23, 1 SAY " DO YOU HAVE MORE INPUT DATA ? (Y/N)"
  GET ANS PICT '1'
  READ
  IF ANS <> "Y" .AND. ANS <> "N"
    LOOP
  ELSE
    IF ANS = "N"
      STORE F TO OK
      STORE F TO MORE
    ENDF
    ENDF
STORE F TO OK
ENDDO (* OK *)
ENDDO (* MORE *)
RETURN
*** STABLE command file ***
* This program processes the single record insert operation. *
* This program is invoked by PROCINSERT command file. *
***
USE B:\FILE
GO RECORD 2
DO WHILE .NOT. EOF
  IF &ROW:NO = ' ' 
    SKIP 
  ELSE 
    STORE FLD:NAME TO FIELD:NAME
    STORE LEN TO FLD:LEN
    STORE DEC TO DECIMAL
    STORE # TO REC:NUM
    IF ' '<$(&ROW:NO) .OR. '>'$(&ROW:NO) .OR. '|'$(&ROW:NO); .OR. '&'$(&ROW:NO)
      STORE T TO ERROR
      STORE " *** ERROR, INVALID QUERY" TO ERROR:MSG
      RETURN
    ELSE 
      STORE @('.',&ROW:NO) TO FOUND:AT
      IF FOUND:AT <> 0 .AND. ( .NOT. ; $(&ROW:NO,FOUND:AT-1,1)$'0123456789')
        IF FOUND:AT = 4 
          IF TYPE = 'C'
            STORE T TO ERROR
            STORE " *** ERROR, built-in functions only";
            " use in numerical type " TO ERROR:MSG
            RETURN
          ENDIF (* type = 'c' *)
          STORE $(&ROW:NO,1,3) TO CMD
          STORE TRIM(&ROW:NO) TO COLUMN
          DO SCAN
          IF ERROR 
            RETURN
          ENDIF (* error *)
          USE B:\FILE
          GO REC:NUM
          IF DECIMAL > 0
            REPLACE &ROW:NO WITH STR(TOTAL, FLD:LEN, DECIMAL)
          ELSE 
            REPLACE &ROW:NO WITH STR(TOTAL, FLD:LEN)
          ENDIF (* decimal > 0 *)
          RELEASE REC:NUM, TOTAL
          SKIP 
          LOOP
        ENDIF(* found:at = 4 *)
        ENDIF (* found:at <> 0 *)
    IF ' '<$(&ROW:NO) .OR. '('*$(&ROW:NO) .OR. '>('$(&ROW:NO); .OR. '+'$(&ROW:NO)
      IF TYPE = 'C'
        STORE T TO ERROR
        STORE &ROW:NO TO MSG
        STORE " *** ERROR, &MSG is not numerical type "; 
        TO ERROR:MSG

146
RETURN
ENDIF (* type = 'c' *)
STORE TRIM(&ROW: NO) TO COLUMN
STORE F TO BUILD: IN
DO ARTHOPERTOR
IF ERROR
RETURN
ENDIF (* error *)
USE B: &FILE
GO REC: NUM
IF DECMAL > 0
REPLACE &ROW: NO WITH STR(TOTAL, FLD: LEN, DECMAL)
ELSE
REPLACE &ROW: NO WITH STR(TOTAL, FLD: LEN)
ENDIF (* decimal > 0 *)
ELSE
IF ' _ '$(&ROW: NO)
IF $(&ROW: NO, 1, 1) = ' _ '
STORE TRIM(&ROW: NO) TO VARIABLE
DO QRYDEP
IF ERROR
RETURN
ENDIF (* error *)
ELSE
STORE T TO ERROR
STORE &ROW: NO TO MSG
STORE " *** ERROR, INVALID QUERY IN &MSG" ;
TO ERROR: MSG
RETURN
ENDIF (* $(&row: no, 1, 1) = ' - ' *)
ELSE
STORE TRIM(&ROW: NO) TO COLUMN
DO TYPECHK
IF ERROR
RETURN
ENDIF (* error *)
ENDIF (* ' ' | ' '$(&row: no) .or. ' ' &$(&row: no) *)
ENDIF
ENDIF (* &row: no = ' ' *)
SKIP
ENDIF (* &row: no = ' ' *)
ENDDO (* .not. eof *)
USE
SELECT PRIMARY
USE B: &FILE
SELECT SECONDARY
USE B: &FILE: NAME
APPEND BLANK
SELECT PRIMARY
GO RECORD 2
DO WHILE . NOT. EOF
STORE FLD: NAME TO FIELD: NAME
STORE &ROW: NO TO CONTENT
SELECT SECONDARY
IF TYPE = 'N'

147
REPLACE &FIELD:NAME WITH VAL(CONTENT)
ELSE
  REPLACE &FIELD:NAME WITH CONTENT
ENDIF (* type(field:name) = 'n' *)
SELECT PRIMARY
SKIP
ENDDO (* .not. eof *)
select secondary
use
select primary
use
*************** STORE1 command file ***************
# This program, which is called by DTABLE command file, stores *
# the user's query entered in the general table in 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 PURGE:BLANK
    USE SFILE
    COPY TO B:RESULT
    USE B:RESULT
    APPEND BLANK
    REPLACE FLD:NAME WITH '&PURGED'
  ENDF (* 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 FPARSER
  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 command file ************************
* This program, which is called by DCON command file, stores *
* the user's condition statements to 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 (* &parser <> ' ' *)
RELEASE &PARSER
STORE ROW:COUNT + 1 TO ROW:COUNT
ENDDO (* row:count <= max:row *)
RELEASE ROW:COUNT, MAX:ROW, PARSER
RETURN

*********************** SUPDATE command file ***********************
* This program handles the update operation queries which only *
* use one table. It scans the query and converts the query to *
* the dBASE command. This program is invoked by UPDATE command *
* file. *

GO RECO 2

149
DO WHILE .NOT. EOF
  IF &ROW:NO = ' ' .OR. FLD:NAME = FIELD:NAME
    IF FLD:NAME = FIELD:NAME
      STORE TRIM($(&ROW:NO,3)) TO COLUMN
      STORE @('.','COLUMN') TO FOUND:AT
      IF FOUND:AT > 0
        IF FOUND:AT = 4
          STORE $(COLUMN,1,3) TO CMD
          IF CMD = 'SUM' .OR. CMD = 'CNT' .OR. CMD = 'MIN';
            .OR. CMD = 'AVG' .OR. CMD = 'MAX'
            IF TYPE = 'C'
              STORE T TO ERROR
              STORE ' *** ERROR, &FIELD:NAME IS NOT '+';
                'NUMERICAL TYPE' TO ERROR:MSG
              RETURN
            ENDDIF (* type = 'c' *)
          STORE # TO REC:NUM
          STORE LEN TO FLD:LEN
          STORE DEC TO FLD:DEC
          DO BULDFUN
          IF ERROR
            RETURN
          ENDDIF (* error *)
          RELEASE FLD:LEN, FLD:DEC, REC:NUM
          ENDDIF(* cmd = 'sum' .or. cmd = 'cnt' .. *)
        ENDDIF (* found:at = 0 *)
      ELSE
        IF '*'COLUMN .OR. '/'COLUMN .OR. '-'COLUMN .OR. 
          '+'COLUMN
          IF TYPE = 'C'
            STORE T TO ERROR
            STORE TRIM(FIELD:NAME) TO MSG
            STORE ' *** ERROR, &MSG IS NOT NUMBERICAL '+';
              'TYPE' TO ERROR:MSG
            RETURN
          ELSE
            DO PROCONTENT
            IF ERROR
              RETURN
            ENDDIF (* error *)
          ENDDIF
        ELSE
          IF '_'$&ROW:NO
            DO UPDEPEND
            IF ERROR
              RETURN
            ENDDIF (* error *)
            USE B:&FILE
            GO RECO REC:NUM
            RELEASE REC:NUM
          ELSE
            STORE TRIM(&ROW:NO,3) TO CONTENT
            STORE '&FIELD:NAME WITH &CONTENT' TO STRING
            ENDDIF (* '_'$row:no *)
          ENDDIF (* found: at > 0 *)
        ENDDIF
      ENDIF
    ENDIF
ENDIF ENDIF (* fld:name = field:name *) SKIP ELSE IF .NOT. TRIM(&ROW:NO) = TRIM(&ROW) STORE T TO ERROR STORE TRIM(&ROW:NO) TO MSG STORE " *** ERROR, &MSG NOT FOUND " TO ERROR:MSG RETURN ENDIF (* .not. trim SKIP ENDIF ENDDO DO REPLACE IF ERROR RETURN ENDIF USE B:&FILE:NAME SET EXACT ON IF I:QUERY = ' ' REPLACE ALL &STRING ELSE REPLACE ALL &STRING FOR &I:QUERY ENDIF (* i:query = ' ' *) SET EXACT OFF USE RETURN

****************************************************************************** TEXPAND command file *********************************************** * This program displays a table which has five column heading * * and field names that user intends to expand. This displayed * * table is used to accept the information form the new field. * * This is invoked by INSFLD command file. *********************************************** ERASE SET TALK OFF SET COLON OFF USE B:&NAME COPY TO B:TEMP STRUCTURE EXTENDED USE B:TEMP GO BOTTOM STORE # TO B IF B+NUMBER > 7 @ 23,1 SAY " *** OUT OF RANGE ***,","+; " YOU CAN HAVE ONLY SIX FIELDS IN EACH FILE " RETURN ENDIF GO TOP @ 2,1 SAY " PLEASE FILL IN STRUCTURES " @ 4,2 SAY FNAME

151
@ 4,11 SAY " | FIELD:NAME | FIELD:TYPE |"+;
" FIELD:LEN | FIELD:DEC |"
@ 5,1 SAY "---------------------+---------------------"+;
"---------------------|"
STORE 6 TO ROW
STORE T TO FLAG
STORE 1 TO COUNTER
STORE ROW + NUMBER TO MAXROW
DO WHILE ROW < MAXROW
    STORE T TO DRAWBAR
    STORE 13 TO COL
    DO WHILE DRAWBAR
        IF COL = 13
            STORE 'FLD'+STR(COUNTER,1) TO FIELD
            @ ROW,COL SAY '||'+ &FIELD
        ELSE
            @ ROW,COL SAY '||'
        ENDIF
    STORE COL+13 TO COL
    IF COL > 70
        STORE F TO DRAWBAR
    ENDIF
ENDDO(*DRAWBAR*)
STORE ' ' TO FLDTYPE
STORE ' ' TO FLDDEC
STORE T TO OK
DO WHILE OK
    @ ROW,32 GET FLDTYPE
    READ
    IF !(FLDTYPE)<'C' .AND. !(FLDTYPE) <> 'N'
        @ 23,1 SAY "***** INVALID TYPE, PLEASE"+;
        " RE-ENTER ********
    STORE 1 TO CNT
    DO WHILE CNT < 20
        STORE CNT+1 TO CNT
    ENDDO
    RELEASE CNT
    STORE ' ' TO FLDTYPE
    @ 23,1
    ELSE
        STORE F TO OK
    ENDIF (* CHECK FLDTYPE *)
ENDDO (* ok *)
STORE F TO OK
DO WHILE .NOT. OK
    STORE ' ' TO FLDLEN
    @ ROW,45 GET FLDLEN PICT '99'
    READ
    IF VAL(FLDLEN) <= 60
        STORE T TO OK
    ENDDIF
ENDDO
RELEASE OK
IF !(FLDTYPE) <> 'C'
    STORE T TO OK

152
DO WHILE OK
  @ ROW,55 GET FLDDEC PICT '99'
  READ
  IF VAL(FLDDEC) < 0
    @ 23,1 SAY " *** BAD FIELD DECIMAL,"+;
    " PLEASE RE-ENTER ***"
    STORE 1 TO CNT
    DO WHILE CNT < 20
      STORE CNT+1 TO CNT
    ENDDO
    @ 23,1
    RELEASE CNT
    STORE ' ' TO FLDDEC
  ELSE
    STORE F TO OK
  ENDDIF
  ENDDO (* OK *)
  RELEASE OK
ENDIF

APPEND BLANK
REPLACE FIELD:NAME WITH &FIELD,FIELD:TYPE WITH FLTYPE,
  FIELD:LEN WITH VAL(FLDLEN),FIELD:DEC WITH VAL(FLDDEC)
  STORE F TO FLAG
  STORE COUNTER+1 TO COUNTER
  STORE ROW+1 TO ROW
ENDDO (* ROW < MAXROW *)
RENAME B:@FNAME TO B;NEW
  @ 23,1 SAY " JUST A SECOND, YOUR QUERY IS PROCESSING "
CREATE B:@FNAME FROM B;TEMP
USE B:@FNAME
APPEND FROM B;NEW
DELETE FILE B;NEW,B;TEMP
RETURN

*************** TYPECHK command file **************
* This program checks the constant entered by use to see if the *
* type match with the declared type. This program is invoked by *
* CONCATENATE command file. ***************

IF TYPE = 'N'
  STORE 0 TO FOUND:AT
  IF DEC > 0
    STORE @('.',COLUMN) TO FOUND:AT
    IF FOUND:AT > 0
      STORE $(COLUMN, 1,FOUND:AT-1) TO FIRST:PART
      IF LEN(FIRST:PART) > (FLD:LEN-(DEC+1))
        STORE T TO ERROR
        STORE " *** ERROR, &COLUMN is too large ";
        TO ERROR:MSG
        RETURN
      ENDF (* len(first:part) > (fld:len-(decimal+1)) *)
    ENDF (* found:at > 0 *)
  ENDF (* decimal *)
  STORE 1 TO K

153
DO WHILE K <= LEN(COLUMN)
  IF K = FOUND:AT
    STORE K+1 TO K
  ELSE
    IF NOT. ($(COLUMN,K,1) >= '0' .AND. $(COLUMN,K,1) <= '9')
      STORE T TO ERROR
      STORE " *** ERROR, invalid type in &COLUMN" ;
      TO ERROR:MSG
      RETURN
    ELSE
      STORE K+1 TO K
    ENDIF
  ENDDO (* k = found:at *)
ENDIF (* k < len(column) *)
ELSE
  STORE 1 TO K
ENDIF
RELEASE K,FOUND:AT
RETURN

***************************************************************************
* UPDEPEND command file *****************************************************
* This program handles the query which needs to retrieve the *
* data from the data base for the update operation. This *
* program is invoked by MUPDATE command file. *
***************************************************************************

STORE # TO REC:NUM
STORE TYPE TO FLD:TYPE
STORE DEC TO FLD:DEC
STORE LEN TO FLD:LEN
STORE TRIM($(ROW:NO,3)) TO VARIABLE
SET EXACT ON
LOCATE FOR FLD:NAM$ = FIELD:NAM$
SET EXACT OFF
IF TRIM(ROW) = VARIABLE
  STORE T TO ERROR
  STORE ' You did not update anything' TO ERROR:MSG
  RETURN
ELSE
  STORE # TO REC:NUM
  STORE ROW TO T:ROW
  STORE FL TO T:FL
  DO FINDROW
    IF ERROR
      RETURN
    ENDDIF (* error *)
    STORE FIELD:NAM$ TO F:NAM$
    DO REPLACE
      STORE F:NAM$ TO FIELD:NAM$
      IF ERROR
        RETURN
      ENDDIF (* error *)
      USE B:FL
      GO TOP
STORE FLD:NAME TO FL:NAME
IF I:QUERY = ' '
STORE T TO ERROR
STORE ' *** ERROR, You did not enter any condition';
' in &row' TO ERROR:MSG
RETURN
ELSE
USE B:FL:NAME
SET EXACT ON
LOCATE FOR &I:QUERY
IF EOF .AND. .NOT. &I:QUERY
STORE T TO ERROR
STORE ' *** ERROR, There is no records which meet the';
' condition of &i:query' TO ERROR:MSG
RETURN
ELSE
STORE &FIELD:NAME TO CONTENT
IF FLD:TYPE = 'C'
STORE '&FIELD:NAME WITH "&CONTENT" ' TO STRING
ELSE
IF FLD:DEC > 0
STORE STR(CONTENT,FLD:LEN,FLD:DEC) TO VALUE
ELSE
STORE STR(CONTENT,FLD:LEN) TO VALUE
ENDIF (* fld:dec > 0 *)
STORE '&FIELD:NAME WITH &VALUE' TO STRING
ENDIF (* type = 'C' *)
ENDIF (* eof.and. .not. &i:query *)
SET EXACT OFF
ENDIF (*
USE
STORE T:ROW TO ROW
STORE T:FL TO FL
RELEASE T:ROW ,T:FL,FLD:TYPE ,FLD:LEN,FLD:DEC
RETURN

************************ UPDATE command file ************************
* This program scans the user's query to see which types of *
* update operation is desired. If the query is multiple table *
* and records update, then the mupdate command file is invoked,*
* otherwise the conupd command file is invoked. This program *
* is called by PROCUPDATE command file.

GO RECO 2
STORE T TO MULTIPLE
STORE T TO BLNK
DO WHILE .NOT. EOF
STORE &ROW:NO TO COLUMN
IF COLUMN = ' ' .OR. FLD:NAME = FIELD:NAME
   SKIP
ELSE
   STORE F TO BLNK
   IF '$_'$(COLUMN)
IF .NOT. $(COLUMN,1,1) = '_'
   STORE T TO ERROR
   STORE " *** ERROR, invalid query in &COLUMN " ;
   TO ERROR:MSG
   RETURN
ENDIF (* .not. $(column,1,1) = '_' *)
IF .NOT. MULTIPLE
   STORE T TO ERROR
   STORE '*** ERROR, INVALID QUERY' TO ERROR:MSG
   RETURN
ENDIF (* .not. multiple *)
ELSE
   STORE F TO MULTIPLE
ENDIF (* '_'$column*)
SKIP
ENDIF (* column = ' ' .or. fld:name = field:name *)
ENDDO (* .not. eof *)

IF BLINK
   STORE F TO MULTIPLE
ENDIF (* blink *)
IF MULTIPLE
   GO RECO 2
   STORE F TO OK
   DO WHILE .NOT. OK
      IF &ROW:NO = ' ' .OR. FLD:NAME = FIELD:NAME
         SKIP
      ELSE
         STORE FIELD:NAME TO F:NAME
         STORE FLD:NAME TO FIELD:NAME
         STORE TRIM(&ROW:NO) TO VARIABLE
         DO FINDROW
            IF ERROR
               RETURN
            ENDF (* error *)
            STORE F:NAME TO FIELD:NAME
            RELEASE F:NAME
            IF FL= FILE
               DO SUPDATE
               RETURN
            ELSE
               DO MUPDATE
               RETURN
            ENDF (* fl=file *)
         ENDF (* &row:no = ' ' .or. fld:name = field:name *)
      ENDDO (* .not. ok *)
ELSE
   DO CONUPD
ENDIF (* multiple *)
RETURN
The Usage of Query-By-Example in a Microcomputer Environment

By

Shun-Jane Lin

B.S., Chung-Hsing University, Taiwan, R.O.C. 1983

-------------------------------

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
Query-by-Example (QBE) is a high level, non-procedural database language which provides users with facilities to create, query, and update a database. All users' requests are specified in QBE by filling in two-dimensional skeleton tables on the screen; this technique is known as two-dimensional programming. This unique and novel method allows the end users to express, with very few entries, the equivalent of a lengthy application program in a conventional language.

The purpose of this project is to implement QBE on a microcomputer. The dBASE-II facilities were used to implement QBE. Since the implementation of QBE is a cooperative effort, this report concentrates on the insert, update, and delete operations of the QBE implementation.

A discussion of how a user request is translated into dBASE-II command statements is given. The translation involves: storing the user's query, parsing the QBE request, and converting it into a dBASE-II command statement.