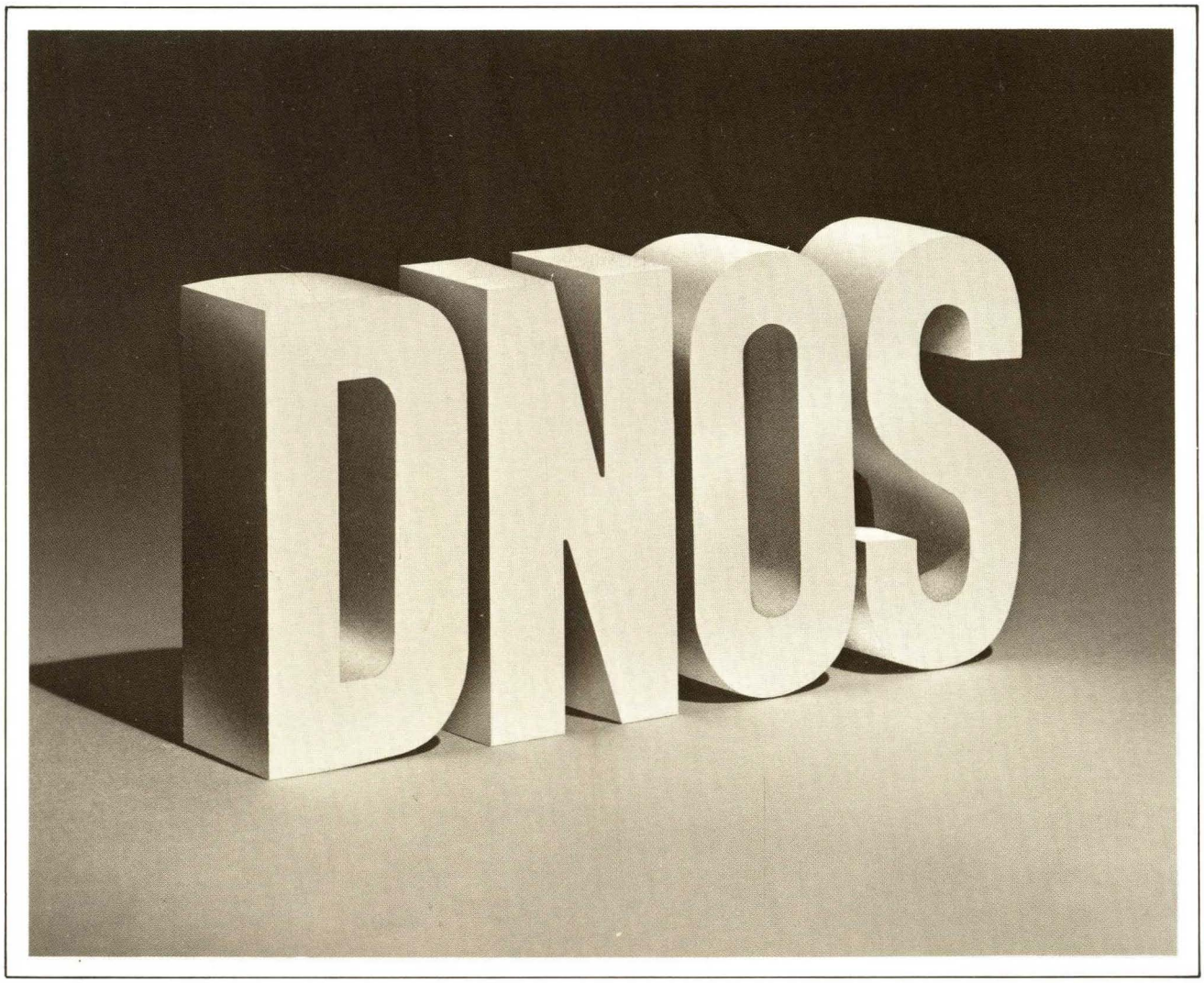
DNOS Query-990 User's Guide



Part No. 2276554-9701 *A
15 July 1982

TEXAS INSTRUMENTS
INCORPORATED

MANUAL REVISION HISTORY

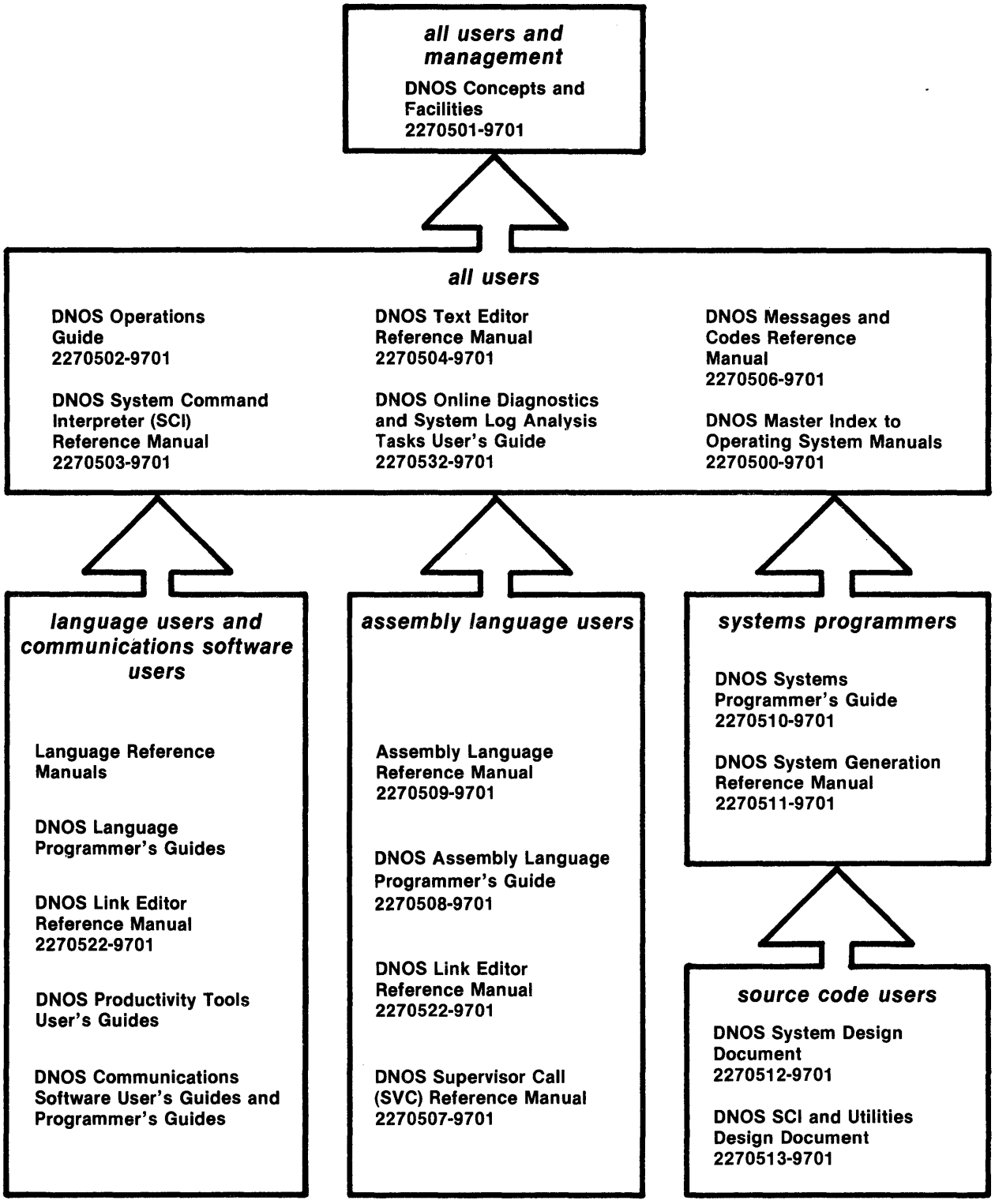
Query-990 User's Guide (2276554-9701)

Original Issue . . . . . 1 August 1981
Revision . . . . . 15 July 1982

The total number of pages in this publication is 196.

DNOS
Distributed Network Operating System Software Manuals

The manuals supporting DNOS are arranged in this diagram according to the type of user. The manuals most beneficial to your needs are those contained in the block identified as your user group and in all the blocks above that set.



2280078

DNOS

Distributed Network Operating System Software Manuals Summary

Concepts and Facilities
Presents an overview of DNOS with topics grouped into functions of the operating system. All new users (or evaluators) of DNOS should read this manual.

Operations Guide
Provides the information necessary to perform daily tasks at a TI 990 Computer installation using DNOS. Step-by-step procedures are presented for such tasks as operating peripherals, initializing the system, backing up the system, and manipulating disk files.

System Command Interpreter (SCI) Reference Manual
Describes how to use SCI in both interactive and batch jobs. Command procedures and primitives are described, followed by a detailed presentation of all SCI commands in alphabetical order for easy reference.

Text Editor Reference Manual
Shows how to use the Text Editor interactively on DNOS and includes a detailed description of each of the editing commands and function keys.

Messages and Codes Reference Manual
Lists the error messages, informative messages, and error codes reported by DNOS.

Online Diagnostics and System Log Analysis Tasks User's Guide
Provides the information necessary to execute the online diagnostic tasks and the system log analysis tasks and to interpret the results.

Master Index to Operating System Manuals
Contains a composite index to topics in the DNOS operating system manuals.

Programmer's Guides and Reference Manuals for Languages
Each programmer's guide describes one of the languages supported by DNOS (for example, assembly language, Pascal, COBOL). Each guide covers operating system information relevant to the use of that language in the DNOS environment. The details of the language itself, including language syntax and programming considerations, are in the language reference manual.

Link Editor Reference Manual
Describes how to use the Link Editor on DNOS to combine separately generated object modules to form a single linked output.

User's Guides for Productivity Tools
Each user's guide describes one of the productivity tools (for example, TIFORM, Query-990, DBMS-990, Sort/Merge) supported by DNOS. Each guide explains the function of the processor, its features, and its interface requirements.

User's Guides and Programmer's Guides for Communications Software
Describe the features, functions, and use of the communications software available for execution under DNOS. For example, there is a user's guide for the DNOS 3780/2780 Emulator software package.

Supervisor Call (SVC) Reference Manual
Presents detailed information about each DNOS supervisor call and general information about DNOS services.

Systems Programmer's Guide
Discusses the DNOS nucleus and subsystems at a conceptual and functional level and describes how to modify the system for a specific application environment.

System Generation Reference Manual
Contains the information needed to perform system generation, including pregeneration requirements, generation procedures, and information about postgeneration results.

System Design Document
Contains the information needed to understand the functioning of the system when using a source kit. This includes descriptions of the subsystems in detail, naming and coding conventions, module cross-references, data structure details, and information not found in other manuals.

SCI and Utilities Design Document
Presents design information about SCI and the DNOS utilities.

Preface

This manual is intended for programmers, managers, operations personnel, and users of Query-990.

This manual is organized into the following sections and appendixes:

Section

1 General Description — Briefly describes the major features of Query-990.

2 Basic Concepts of Query-990 — Provides an introduction to the language, elements, and syntax of Query-990.

3 Functions — Discusses the four available functions and the special clauses associated with them.

4 Clauses — Describes the Query-990 clauses and explains report formatting and change data constants.

5 Optimization — Discusses how to optimize Query-990.

6 Program Language Interface Subroutines — Describes the subroutines used in application programs to execute Query.

7 The Guided Query Utility — Describes the Guided Query utility and familiarizes users with the Query language.

8 Error Messages — Explains the error messages associated with the Query processor and the Guided Query utility.

Appendix

A Query-990 Syntax — Provides the syntax definitions for the elements of the Query language.

B Calculation Data Types — Describes the data types used for Query calculations and in DBMS-990 files.

C Alternate Collating Sequences — Describes the alternate collating sequences for the Germany/Austria and the Sweden/Finland character sets.

D DDL Listings for Example Files — Lists the DDL descriptions of files used for examples in this manual.

E Example Query Application — Provides examples of interactive data-retrieval applications.

The following documents contain additional information related to operating Query-990 under the DNOS operating system:

Title	Part Number
Model 990 Computer DNOS Data Base Management System Programmer's Guide	2272058-9701
Model 990 Computer DNOS Data Base Administrator User's Guide	2272059-9701
Model 990 Computer DNOS Operations Guide	2270502-9701
Model 990 Computer Data Dictionary User's Guide	2276582-9701

Contents

4 — Clauses

5 — Optimization

6 — Program Language Interface Subroutines

7 — Guided Query Utility

8 — Error Messages

Appendixes

Index

Illustrations

Tables

1

General Description

1.1 INTRODUCTION

This manual discusses the conceptual and functional characteristics of Query-990. It contains both user information and reference material needed to understand and operate Query-990.

This section introduces Query-990. Topics covered include Query-990 capabilities and environments.

1.2 QUERY-990 CAPABILITIES

Query-990 is a powerful retrieval system that allows you to access, modify, and display information contained in a file. Query-990 uses the English language in simple, logical statements to generate reports. Query statements can replace application programs that produce reports.

Both experienced programmers and users without programming knowledge can use Query-990. You can use Query-990 as a stand-alone utility, or the application interface subroutines can execute Query-990 from an application program.

1.3 ENVIRONMENTS

The combination of Query-990 and the Data Dictionary (DD-990) provides access to key indexed files (KIFs), relative record (random access) files, and sequential files. The combination of Query-990 and DBMS-990 provides access to data base files. Combining all three systems results in an efficient and versatile self-documenting data management system.

1.4 DBMS-990 AND DD-990

Prior to detailed discussion of DBMS-990 and Data Dictionary, it is necessary to establish a common definition for DBMS-990, key indexed, relative record, and sequential files. The following terminology is used to refer to a file and its components:

- FILE — A collection of records
- RECORD — A collection of all lines (regardless of type) with primary keys having the same value
- LINE — An occurrence of a line type
- KEY — A field whose value can be used to access like-valued lines directly

- GROUP — A collection of fields within a line
- FIELD — A named element in a line

Query-990 requires either DBMS-990 or DD-990 (or both) in order to operate. DBMS-990 accesses DBMS-990 files, and DD-990 accesses key indexed, relative record, and sequential files. DD-990 can also enhance DBMS-990.

DBMS-990 is the data base management system that executes on the Texas Instruments Model 990 Computer. DBMS-990 is a data manager, allowing you to establish the contents, grouping, relationships, and security of all data elements within a DBMS-990 file. DBMS-990 is easy to use and provides a logical viewpoint of the data. Normal physical constraints such as access method, record size, block size, and relative field position should not concern you.

DBMS-990 is a mechanism for organizing, storing, updating, and retrieving data through mass-storage devices. The 990 computer's mass-storage media is disk, which facilitates the use of random-access techniques.

1.4.1 DBMS-990 File Structures

The highest-level element in DBMS-990 is the file (Figure 1-1). A file is composed of records, each of which has a unique primary key value. Each record is composed of one or more lines, and each line is composed of fields. The DDL processor in DBMS-990 defines the file structurally. File and field IDs are four characters long; the first character must be a letter, but the other three characters can be either letters or numbers. Line IDs are two characters long, in the range of AA through ZZ and 01 through 99.

Lines have a unique order within a record. You establish this order when you insert the lines. The 01 line type is a special line type. If you define a 01 line for a file, that line can occur only once per record, and it must be the first line added and the last line deleted. Other line types can have zero or more occurrences in a particular record. Also, a given line type (other than 01) can occur in different records a different number of times.

To create a record, add a line to the DBMS-990 file, specifying a new primary key value. You can then add additional lines to the record. When the last line in a record is deleted, the record is deleted.

1.4.2 Data Dictionary (DD-990)

DD-990 is a data manager providing data base descriptions for key indexed, relative record, and sequential files.

1.4.2.1 Data Dictionary File Structures. Query-990 allows DD-990 access to key indexed, relative record, and sequential files. Only a subset of the Query syntax is allowed for sequential and relative record files. You need not specify in the Query statement which file type is being accessed.

Key Indexed Files (KIFs). The structure of a KIF is almost identical to that of a data base file. A record in a KIF is all records with primary keys having the same value. If duplicates are not allowed for primary keys, a KIF is a collection of records containing one line each. A key is anything defined to the operating system as a key when the KIF is created. Therefore, KIFs may have both primary and secondary keys.

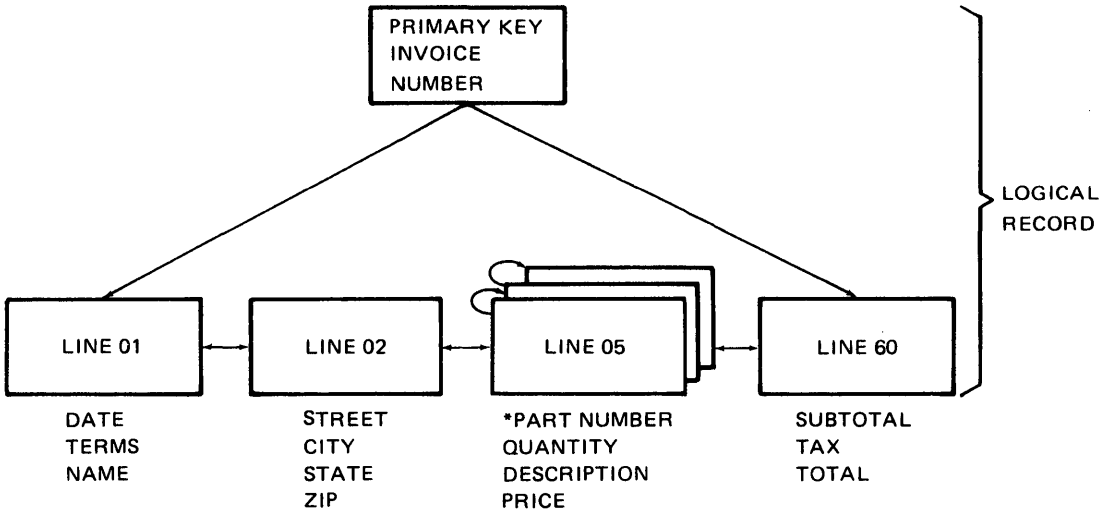


Figure 1-1. DBMS-990 File Structure

Relative Records Files. In a relative record file, each record is also a data base record. Each record contains one line. The primary key for each record is the record number the operating system associates with each record. Relative records do not have secondary keys.

When you are using Query-990 with relative record files, a restriction applies to the POSITION clause. Records are positioned only by the value of their record number. Therefore, the POSITION clause in the INSERT function is not allowed. Also, the DELETE and DELETE RECORD statements are not legal for relative record files.

Sequential Files. Sequential files are treated much like relative record files. Each line in the file is a record. Each record contains only one line. The primary key is the record number. Sequential files do not have secondary keys.

The only restriction when using Query-990 with sequential files occurs in the POSITION clause and the DELETE function. Since operating systems do not provide any delete capability for sequential files, the DELETE and DELETE RECORD statements in Query-990 are not allowed.

Since you can add records to the end of the file, you cannot specify the POSITION clause on the INSERT function.

1.5 EXAMPLES

The examples in this manual are based on the files PAY1, CUST, ITEM and SOFL. These files are included on the Query installation disk. The example information is duplicated in two files; one is a DBMS-990 file for users with only DBMS-990, the other is a KIF for users with only DD-990. Users with both products can run the examples on either file. The DBMS-990 files are located in the directory DNQRYOBJ.TESTFILE.DBMS. The KIFs are located in the directory DNQRYOBJ.TESTFILE.DD. A dictionary containing the descriptions of the KIF files is located at DNQRYOBJ.TESTFILE.DD.DICT. Appendix D contains the DDL declarations for these files.

The restriction that no two IDs in a dictionary can be the same has caused slight DDL changes for the KIF files CUST and SOFL. In CUST, the ADDR group ID is changed to ADRS. In SOFL, the ITEM field ID is changed to SITM.

NOTE

The Query examples shown in this manual were all executed using DBMS-990 files. Since KIFs order data differently than data base files, the output of examples using KIFs might differ from the examples illustrated in this manual.

In order to use examples with KIFs, the following figures should be changed:

Figure 4-1	The field name ITEM should be changed to SITM
Figure 4-2	The field name ITEM should be changed to SITM
Figure 4-8	The field name ITEM should be changed to SITM
Figure 4-20	The field name ITEM should be changed to SITM
Figure 4-21	The field name ITEM should be changed to SITM
Figure 4-28	The field name ITEM should be changed to SITM (do not change the file name ITEM in the FROM clause)
Figure 4-30	The field name ITEM should be changed to SITM (do not change the file name ITEM in the FROM clause)
Figure 4-31	The field name ITEM should be changed to SITM
Figure 4-32	The field name ITEM should be changed to SITM
Figure 4-33	The field name ITEM should be changed to SITM
Figure 4-34	The field name ITEM should be changed to SITM (do not change the file name ITEM in the FROM clause)
Figure 4-35	The field name ITEM should be changed to SITM (do not change the file name ITEM in the FROM clause)
Figure 4-36	The field name ITEM should be changed to SITM (do not change the file name ITEM in the FROM clause)

CAUTION

It is possible to assign both a data base and KIF with the same four character ID.

2

Basic Concepts of Query-990

2.1 INTRODUCTION

This section explains the Query language and the elements and syntax of Query statements. This section also introduces the Guided Query utility as a training tool for learning the syntax of Query statements. Finally, execution of the Query processor is explained.

2.2 THE QUERY LANGUAGE

The Query language is a nonprocedural language that resembles English. A Query statement consists of one or more lines of instructions. You can easily read a Query statement and understand the instructions to be executed. An example Query statement is as follows:

```
LIST EMPLOYEE-INFO FROM PAYROLL-FILE WHERE EMPLOYEE-NUMBER EQ 65
```

Query statements consist of functions and clauses. The function is the operation that the Query statement performs. Each statement must include only one function. Clauses are conditions or modifications to the function. Each statement must include one FROM clause and as many additional clauses as necessary.

In the preceding example, LIST is the function, and FROM and WHERE are clauses. The function and clauses in a Query statement can be positioned in any order. (However, the order of the clauses sometimes affects the output; see Section 4 for more information.) For example, the following is equivalent to the preceding Query statement:

```
WHERE EMPLOYEE-NUMBER EQ 65 FROM PAYROLL-FILE LIST EMPLOYEE-INFO
```

2.2.1 Functions

Query-990 can perform any of the following functions:

- LIST — Produces reports or generates data for further processing
- UPDATE — Modifies information that is currently stored in a file
- INSERT — Adds new lines to a file
- DELETE — Deletes one or more lines from a record, an entire record, or an entire file

A Query statement must include only one of the functions LIST, UPDATE, INSERT, or DELETE. Consequently, to both list and update a file, you must use two separate statements.

2.2.2 Clauses

There are two types of clauses: record-level and line-level. Record-level clauses apply to the entire Query statement and act in a global manner, affecting all report lines or modification lines. They should be specified after the FROM clause or before the function.

Line-level clauses affect only the single report line with which they are specified. They must be specified after the list of fields and literals in a report line and before the semicolon.

Several of the clauses are as follows:

- FROM — Identifies which file the Query statement will process.
- WHERE — Specifies test conditions that identify which records or lines in the file will be listed or modified.
- SORT — Orders output data based on the two levels of sorting (line-level and record-level). This clause can order lines within a record or order records in a file.
- TRACE — Allows you to see a listing that shows changes to the file before the file has actually been modified. The use of this clause is recommended with the DELETE function.
- BY — Controls the order in which the Query processor reads the data in the file.
- DEFINE — Specifies calculations on fields and allows the use of the calculations as report elements or operands in a condition.
- BREAK — Allows control break processing on totals, counts, and duplicate values.
- UNIQUE — Indicates a single occurrence of a specified line type per record. Fields within a UNIQUE line type can act as primary keys when used in report lines. Secondary keys within that line can be optimized.
- LINKED BY — Defines the relationship between files when using more than one file in a Query statement.

2.2.3 Punctuation

The punctuation conventions for Query statements are as follows:

Double quotes Single quotes	(" ") (' ')	Data surrounded by single or double quotes is printed or displayed exactly as written. This type of data is referred to as either a literal or a literal constant.
Exclamation point	(!)	The Query processor treats the exclamation point on a line as a comment. (Comments are a recommended form of internal documentation for the Query statement.)
Semicolon	(;)	Report lines used to group data for listing or conditioning are separated by semicolons.

The example in Figure 2-1 shows a Query using punctuation.

Query Statement:

```
! This comment is included for internal documentation

LIST "EMPLOYEE NAME:     " MNAM;
     ´        SALARY:     ´ MRAT;
     ´   PAY PERIOD:     ´ MPYP " DAYS";
FROM PAY1
```

Query Output:

```
EMPLOYEE NAME:     LI, KIM
        SALARY:      230.00
   PAY PERIOD:     15 DAYS
EMPLOYEE NAME:     PASCHAL, JIMMY
        SALARY:     2500.00
   PAY PERIOD:     30 DAYS
EMPLOYEE NAME:     MEREDITH, JOHN
        SALARY:      900.00
   PAY PERIOD:     30 DAYS
EMPLOYEE NAME:     HOWELL, JOHN
        SALARY:      375.00
   PAY PERIOD:      6 DAYS
EMPLOYEE NAME:     BROWN, WILLIE
        SALARY:      215.00
   PAY PERIOD:      5 DAYS
EMPLOYEE NAME:     STEPHENS, JANET
        SALARY:      385.00
   PAY PERIOD:      7 DAYS
EMPLOYEE NAME:     ABLE, CHARLIE
        SALARY:     1950.00
   PAY PERIOD:     30 DAYS
EMPLOYEE NAME:     HAYNES, BILL
        SALARY:      750.00
   PAY PERIOD:     15 DAYS
EMPLOYEE NAME:     PARKS, FRED
        SALARY:      558.00
   PAY PERIOD:      7 DAYS
```

Figure 2-1. Punctuation Example

2.3 QUERY STATEMENT ELEMENTS

Query statements use file elements (such as PAY1), aliases (such as PAYROLL-FILE), variables, reserved words (such as SORTED BY), and constants to select and define the output or content of the report. The following paragraphs describe each of these items.

2.3.1 File Elements

You can specify file elements such as field, group, line, and file IDs in a Query statement. The file ID must match the Data Definition Language (DDL) ID specified when the file was created unless you have defined an alias for that file name. An example of a Query statement using several file elements is as follows:

```
FROM PAY1     LIST     MNUM     ADDR     CU
```

PAY1 is the file, MNUM a field, ADDR a group, and CU a line. Appendix D lists file specifications for all examples in this manual.

2.3.2 Alias Feature

Aliases are substitute names for field, group, line, and file IDs. These longer substitute names are easier to remember than the short names they replace. An alias name can be up to 20 characters long for DBMS-990. Alternate names of up to 30 characters are used for DD-990. It must start with an alphabetic character and can include letters, numbers, dollar signs ($), underscores (_), or dashes (-). An example of a Query statement using aliases is as follows:

```
LIST EMPLOYEE-NUMBER   EMPLOYEE-NAME   FROM PAY1
WHERE SALARY$ = 700
```

The three names EMPLOYEE-NUMBER, EMPLOYEE-NAME, and SALARY$ are aliases. The actual field names cannot be greater than four characters. The *Model 990 Computer DNOS Data Base Administrator User's Guide* contains further discussion of aliases. The *Model 990 Computer Data Dictionary User's Guide* contains further discussion of alternate names.

2.3.3 Variables

Variables are the user-defined names that are specified in the DEFINE clause (Section 4) and are used for calculation. They follow the same naming conventions as aliases. If the same name is used for a DEFINE variable and an alias, the name is assumed to specify the DEFINE variable. If the same name is used for a DDL element and an alias, it is assumed to specify the alias. In the following example of a Query statement using variables, assume A and B are elements in the file PAY1:

```
DEFINE C:CN/4.2 = A + B;
LIST C   FROM PAY1
```

2.3.4 Reserved Words

The following reserved words cannot be used as user-assigned variables, aliases, or DDL IDs:

AFTER	FALSE	NULL
ALL	FIRST	OFF
AND	FOOTING	ON
ANY	FROM	ONLY
AVERAGE	GE	OR
BEFORE	GT	PAGE
BREAK	HEADER	RECORD
BY	IN	SKIP
CONTENTS	INSERT	SPACE
COUNT	KEY	TAB
DEFAULT	LE	TOTAL
DEFINE	LINKED	TRACE
DELETE	LIST	TRUE
EQ	LT	UNIQUE
EVERY	NE	UPDATE
EXCLUSIVE	NO	WHERE

2.3.5 Constants

The three types of constants are as follows:

- Literals — Composed of a character string enclosed by single or double quotes. Both single or double quotes can be mixed in the same Query statement. However, each literal must begin and end with either single or double quotes.

- Numeric constants — Used in conditions, calculations, and content lists. Numeric constants can begin with a number, plus sign (+), minus sign (−), or period (.). Decimal points can be included.

- Change data constants — Allow you to supply a series of values for one variable while the Query statement is executing. This applies only to CONTENTS clauses, WHERE conditions, DEFINE clauses, and report line literals. See Section 4 for more information.

2.3.6 Special Constants

NULL, TRUE, and FALSE are special constants. When a NULL constant is specified, the constant value is automatically defined to be of the same length and data type as the field being compared. The NULL constant contains binary zeros. TRUE and FALSE constants are valid only for fields of type Boolean and result in a constant (two bytes long) that contains either binary 1 (for TRUE) or 0 (for FALSE).

2.4 QUERY SYNTAX

Certain symbols are used to clarify statement definitions.

Symbol	Description
::=	Used in writing definitions; means "is defined to be"
[ ]	Encloses entities that are optional
|	Indicates alternatives (e.g., A | B | C means A or B or C)
{ }	Encloses one or more entities from which you must select at least one
. . .	Indicates the position at which a previous item may be repeated as required
_	Underlined keywords and symbols must appear exactly as shown

A Query statement has the following basic syntax:

[DEFINE-clause] function-clause FROM-clause [WHERE-clause]

Only the FROM clause and one of the functions are always required. You can include additional clauses as needed. Although Query-990 is a free-format language, the order of the clause in the Query statement will affect the output.

2.5 GUIDED QUERY

The Guided Query utility allows you to gather data from a file without having extensive knowledge of Query-990. Although it does not have all the capabilities of Query-990, Guided Query is excellent for training. The following exercise will help you become familiar with writing and executing a Query statement. Guided Query can perform only the LIST function.

NOTE

Refer to the *Model 990 Computer DNOS Data Base Administrator User's Guide*, part number 2272059-9701, or refer to the *Model 990 Computer Data Dictionary User's Guide*, part number 2276582-9701 for instructions on starting the Data Base manager.

The following example shows each step of building a Guided Query statement. In the first column, the screens are numbered to coincide with the screen numbers you will see on the video display terminal (VDT). In the second column, enter input as written. The SKIP instruction does not refer to the SKIP control key; it means enter the word SKIP. Your password is the password assigned when Query-990 was installed. After entering the input, press the control key indicated in the last column.

In the following exercise, use the F4 key to return to previous screens if you need to change your input.

To begin this exercise, enter the input GQUERY as shown below and then follow the example.

Screen	Input	Control Key
	GQUERY	RETURN
	(Your Password)	RETURN (if security is installed)
1.	PAY1	F3
2.	Y	RETURN
	SKIP	RETURN
	" EMPLOYEE ADDRESSES"	RETURN
	SKIP	F3
3.	01	F3
4.	2	RETURN
	3	RETURN
	4	RETURN
	5	RETURN
	6	F3
5.		RETURN
		RETURN
		RETURN
		RETURN
		RETURN
6.	Y	RETURN
	Y	F3
7.	N	F3
10.	N	F3
11.	Y	F3
12.	EVERY	RETURN
	MNAM	RETURN
	NE	RETURN
		RETURN
	"PASCHAL, JIMMY"	F3
13.	N	F3
14.	Y	F3
15.	01	F3
16.	2	F3
17.		F3
18.		F3
	<directory pathname>.TESTIN	RETURN
		RETURN
		CMD
		CMD
		RETURN
	<directory pathname>.TESTOUT	RETURN
		CMD
SF	<directory pathname>.TESTOUT	(to see the Guided Query results)

At the end of the Guided Query session, you can display and then save the resulting Query statement. The Query processor can then execute the saved statement.

NOTE

While the Guided Query is an aid to the beginner, it is not intended to be the primary mode of operation for Query sessions.

Figure 2-2 shows the results of the Guided Query exercise example.

Query Statement:

```
LIST
MNAM MSTR
MCTY MSTT MZIP
HEADER
;
BY KEY BY LIST
FROM PAY1
SORTED BY MNAM
HEADER
SKIP
"          EMPLOYEE ADDRESSES"
SKIP
WHERE
ANY MNAM NE "PASCHAL, JIMMY"
```

Query Output:

```
          EMPLOYEE ADDRESSES

MNAM                 MSTR                 MCTY           MSTT  MZIP
ABLE, CHARLIE        2800 SKYWAY          ASPERMONT      MO    32145
MNAM                 MSTR                 MCTY           MSTT  MZIP
BROWN, WILLIE        600 W 55TH           NEW YORK       NY    88889
MNAM                 MSTR                 MCTY           MSTT  MZIP
HAYNES, BILL         500 LAIRD            DEL CURTO      TX    85269
MNAM                 MSTR                 MCTY           MSTT  MZIP
HOWELL, JOHN         555 RIO GRANDE       GRANGER        TX    78787
MNAM                 MSTR                 MCTY           MSTT  MZIP
LI, KIM              3800 TONKAWA TRAIL   BROOKSIDE      MO    22222
MNAM                 MSTR                 MCTY           MSTT  MZIP
MEREDITH, JOHN       98 N. LAMAR          GOLIAD         TX    89898
MNAM                 MSTR                 MCTY           MSTT  MZIP
PARKS, FRED          200 NEW YORK AVE.    RUSK           NY    78998
MNAM                 MSTR                 MCTY           MSTT  MZIP
STEPHENS, JANET      56 PURNAM DR         ECHO           TX    87989
```

Figure 2-2. Guided Query Example

2.6 EXECUTING THE QUERY PROCESSOR

Complete processing of a Query statement involves compiling, loading, and executing. Query-990 gives you the option of executing the Query processor (that is, performing the entire process) or only compiling the statement for later execution from an application program. You can perform either operation in foreground mode or in a batch stream.

To execute the operation in foreground, enter the appropriate command (QUERY or QCOMPILE) at a terminal that is ready to execute in foreground mode. Execute the operation in a batch stream in the same manner as for other System Command Interpreter (SCI) commands executed in batch mode; in the batch command file, write the command followed by the required keywords and their values.

2.6.1 QUERY Command

The Query processor is executed through the SCI command processor by entering the following:

```
QUERY
```

The following prompts appear:

```
QUERY-990      <VERSION L.V.R     YY.DDD>
                          PASSWORD:   (if security is installed )
          INPUT STATEMENT PATHNAME:
        OUTPUT STATEMENT PATHNAME:
          REPORT/TRACE ACCESS NAME:
       DEFAULT REPORT PARAMETERS:   YES
```

PASSWORD

This prompt appears only if your system includes security. In response, enter a valid password that has appropriate access to the files to be used in the Query statement. If data base alias names are to be used, the password must also have access to the alias file. Usually, the data administrator for your system assigns these passwords. Refer to the *Model 990 Computer DNOS Data Base Administrator User's Guide*, part number 2272059-9701 for more information on assigning passwords.

INPUT STATEMENT PATHNAME

If the Query statement to be executed has been created previously through the Text Editor or the Query Editor and you wish to execute or modify the statement, enter the file pathname that indicates where the statement is stored. To enter a new Query statement, respond to this prompt by pressing the TAB key or the NEW LINE/RETURN key and leaving the prompt blank.

OUTPUT STATEMENT PATHNAME

This prompt has two functions. It identifies the file that stores the results of the Query edit session, and it executes the Query Editor. If you respond with a pathname or DUMY, the Query Editor executes. Use DUMY when you want to edit the statement but you do not want to save the results of the Query edit. If you do not respond to this prompt, the Editor executes only if you also enter no response to the prompt INPUT STATEMENT PATHNAME.

REPORT/TRACE ACCESS NAME

This prompt requests the file to which the results of executing the Query statement will be sent. To specify that the results appear on the VDT, press the RETURN key. If output is unformatted, a file name must be specified.

DEFAULT REPORT PARAMETERS

To enter the default response (YES), press the NEW LINE/RETURN key or the TAB key. When executing a modification function, use the default. However, when executing a LIST function, you might need to reset the report parameters; in this case, enter NO. A response of NO causes the following screen to appear:

```
REPORT PARAMETERS
          REPORT/UNFORMATTED?:  R
      NUMBER OF LINES PER PAGE:  60
NUMBER OF COLUMNS PER LINE:  80
            LIST QUERY TEXT?:  YES
       CHANGE DATA PATHNAME:
```

NOTE

Refer to paragraph 4.4 for a discussion of report output.

REPORT/UNFORMATTED

This parameter determines whether output will be formatted for readability or left in its original form. This is significant for binary, integer, real, computational, or packed data. To obtain output in report form, enter R. To obtain output that can be used as input for some other program or processor, enter U; as a result, the data remains in its original format.

NUMBER OF LINES PER PAGE

In response to this prompt, specify the page length for reports produced in report format. Any value entered is ignored if the report is unformatted.

NUMBER OF COLUMNS PER LINE

In response to this prompt, enter any number from 1 through 132 to specify the number of columns per line. This value applies only to report-formatted queries; it has no effect when the Query output is unformatted.

LIST QUERY TEXT?

A YES response to this prompt lists the Query statement text on page 1 of the output. Any syntax errors are listed with the text. Output from executing the Query statement begins on page 2 of the report. Syntax errors are listed only when the Query text is also listed. If executing a Query statement produces the message FATAL ERRORS IN QUERY STATEMENT, reexecute the statement with this option set to YES to determine the nature of the errors.

With unformatted output, the Query statement is listed at the beginning of the file, but pages are not numbered. Generally, list the Query text for unformatted output only while syntax errors still occur.

CHANGE DATA PATHNAME

A Query statement can contain change data constants (described in Section 4), the values of which are stored in a separate file. If you are using change data constants, enter the pathname of the file that contains their values.

2.6.2 QCOMPILE Command

Use QCOMPILE to compile a Query statement and to store the object code in a file for use within an application program. The application calls the subroutine QINIT to load the object.

Execute the stand-alone compiler by entering the command as follows:

```
QCOMPILE
```

The following prompts appear:

```
QUERY-990 STANDALONE COMPILER  <VERSION L.V.R     YY.DDD>
                     PASSWORD:  (if security is installed)
       INPUT STATEMENT PATHNAME:
      OUTPUT STATEMENT PATHNAME:
                OBJECT PATHNAME:
               LISTING PATHNAME:
  DEFAULT REPORT PARAMETERS:  YES
```

PASSWORD

This prompt appears only if your system includes security. In response, enter a valid password that has appropriate access to the files to be used in the Query statement. If data base alias names are to be used, the password must also have access to the alias file. Usually, the data administrator for your system assigns these passwords. Refer to the *Model 990 Computer DNOS Data Base Administrator User's Guide*, part number 2272059-9701 for more information on assigning passwords.

INPUT STATEMENT PATHNAME

If the Query statement to be executed has been created previously through the Text Editor or the Query Editor and you wish to execute or modify the statement, enter the file pathname that indicates where the statement is stored. To enter a new Query statement, respond to this prompt by pressing the TAB key or the NEW LINE/RETURN key.

OUTPUT STATEMENT PATHNAME

This prompt has two functions. It both identifies the file that will store the results of the Query edit session and signals that the Query Editor is to be invoked. If you respond with a pathname or DUMY, the Query Editor executes. Use DUMY when you want to edit the statement but do not want to save the results. If you do not respond to this prompt, the editor is invoked only if you enter no response to the prompt INPUT STATEMENT PATHNAME.

OBJECT PATHNAME

This prompt requires the name of the file in which the statement object code is to be stored.

LISTING PATHNAME

This prompt requests the name of the file to which the compiler sends the compilation listing. The file includes any compiler error messages and a listing of the Query statement.

DEFAULT REPORT PARAMETERS

To enter the default response (YES), press the NEW LINE/RETURN key or the TAB key. When executing a modification function, use the default. However, when executing a LIST function, you might need to reset the report parameters; in this case, enter NO. Responding NO causes the following screen to appear:

```
REPORT PARAMETERS
                REPORT/UNFORMATTED:  R
      NUMBER OF LINES PER PAGE:      60
    NUMBER OF COLUMNS PER LINE:      80
                LIST QUERY TEXT?:    YES
             CHANGE DATA PATHNAME:
```

REPORT/UNFORMATTED

This parameter determines whether output will be formatted for readability or left in its original form. This is significant for binary, integer, real, computational, or packed data. To obtain output in report form, enter R. To obtain output that can be used as input for some other program or processor, enter U; as a result, the data remains in its original format.

NUMBER OF LINES PER PAGE

In response to this prompt, specify the page length for reports produced in report format. Any value entered is ignored if the report is unformatted.

NUMBER OF COLUMNS PER LINE

In response to this prompt, enter any number from 1 through 132 to specify the number of columns per line. This value applies only to report-formatted queries; it has no effect when the Query output is unformatted.

LIST QUERY TEXT?

A YES response to this prompt lists the Query statement text on page 1 of the output. Any syntax errors are listed with the text. Output from executing the Query statement begins on page 2 of the report. Syntax errors are listed only when the Query text is also listed. If executing a Query statement produces the message FATAL ERRORS IN QUERY STATEMENT, reexecute the statement with this option set to YES to determine the nature of the errors.

With unformatted output, the Query statement is listed at the beginning of the file, but pages are not numbered. Generally, list the Query text for unformatted output only while syntax errors still occur.

CHANGE DATA PATHNAME

A Query statement can contain change data constants, the values of which are stored in a separate file. If you are using change data constants, enter the pathname of the file that contains their values.

2.6.3 Query Editor

The Query Editor is activated if the response to the prompt INPUT STATEMENT PATHNAME is blank or the response to OUTPUT STATEMENT PATHNAME is not blank when the Query processor or compiler is initialized. The words ENTER YOUR QUERY appear at the top of the screen. If you entered a response for INPUT STATEMENT PATHNAME, the contents of the input statement file appear on the VDT followed by the end-of-file indicator (*EOF). Otherwise, the *EOF appears on the second line of the screen, and the cursor is positioned at the top of the screen in the left-hand corner.

Use the following keys on the VDT keyboard to compose or edit a Query statement:

Key	Description
Up arrow (↑)	Moves cursor up.
Down arrow (↓)	Moves cursor down.
Left arrow (←)	Moves cursor to the left.
Right arrow (→)	Moves cursor to the right.
DEL CHAR	Deletes character.
INS CHAR	Inserts character.
ERASE INPUT	Deletes line.
Blank gray	Inserts a line.
SKIP	Erases characters at and to the right of the cursor on the current line (uppercase SKIP is the TAB key).
TAB and FIELD back	Uses the tab stops set for the system Text Editor.
Roll up (F1) and Roll down (F2)	The F1 key scrolls the screen forward (toward the end of the file), and the F2 key scrolls the screen backwards. The value specified in the Modify Roll (MR) SCI command determines the number of lines scrolled.
F4	Copies the preceding line from the current cursor position to the end of the line.
RETURN	The cursor moves to the first character position of the next line. If the cursor is on the last line of the screen when you press this key, a new line is inserted at the bottom of the screen and the cursor moves to that line.

Pressing either the SEND/ENTER key or the HELP/CMD key causes the following to appear:

DO YOU WANT TO ABORT(A), EXECUTE(E), OR CONTINUE EDITING(C)?

Respond by entering A, C, or E and pressing the RETURN or ENTER key. Entering A aborts the Query Editor and does not save the results of the edit. Entering E executes the Query statement. Entering C allows you to continue the editing process.

3

Functions

3.1 INTRODUCTION

A function is the operation that the Query statement performs. Each Query statement performs one of the following functions:

- LIST — Produces reports or generates data for further processing
- UPDATE — Modifies information currently stored in a file
- INSERT — Adds new lines to a file
- DELETE — Deletes one or more lines, one or more records, or every record in a file

The syntax for a function statement is as follows:

function-clause :: = { LIST-clause | INSERT-clause | UPDATE-clause | DELETE-clause }

Clauses expand the capabilities of each function. This section discusses several clauses with each function. Clauses are discussed in greater detail in Section 4.

3.2 LIST FUNCTION

The format of the LIST function is as follows:

LIST-clause :: = LIST report-line [; report-line]. . .[;]

When the specified function is LIST, you must specify the contents of each report line to be listed. A report line can be composed of any or all of the following:

- Entire lines
- Groups or fields from one or more lines
- The results of calculations (such as totals and averages)
- Literals

3.2.1 Report Line Elements

A report line is a unique group of data that can be listed or conditioned. You can specify lines, groups, or fields in a file as the elements of a report line. If you specify a line or group, it is broken down into its component fields in the report output. These fields are listed in the order in which they were specified in the DDL; the following rules apply:

- If the order is BY LIST, each report line must be composed of fields from the same line type and/or the primary key.
- If the order is BY KEY or BY KEY BY LIST, each report line can be composed of fields from any line, but the grouping for output follows a fixed traversal rule.

In general, if line type A is specified along with other line types and line type A occurs more often than the others, some of the data from line type A will not be listed. Consequently, you should use caution grouping report line fields from different line types unless this order is clearly understood and appropriate.

A report line specification includes file elements, literals, and formatting information. The output for a report line consists of fields, literals, and spaces; the line length ranges from 1 to 480. The output for a report line may require only one line or many lines. For instance, if a report specifies 270 characters of data, literals, and spacing and the page width is 80, the report line uses 4 output lines. When the Query statement is to produce unformatted output, no spacing occurs. When the Query statement is to produce a formatted report, two blanks are automatically inserted between fields. No blanks are inserted between literals or between a field and a literal. In report format, if a field spans two lines of output and its output length is shorter than the page width, the field is adjusted so that it starts on a new line.

3.2.2 Syntax

The syntax for a report line is as follows:

```
report-line ::= report-line-element [[,] report-line-element] ...
                                    [HEADER-clause] [WHERE-clause] [SORT-clause]

                                  { field-type [option] [THRU-clause]
                                  | line-type [option]
                                  | key-type [option]
                                  | variable [option]
report-line element ::=           | BREAK-clause
                                  | X length
                                  | SPACE length
                                  | TAB digit [digit]
                                  | string
                                  | change-data }
```

3.3 INSERT FUNCTION

The INSERT function adds new lines or new information to a file. The basic format of the INSERT function is as follows:

INSERT-clause ::= INSERT [trace-indicator] insert-line [; insert-line]. . .[;]

insert-line ::= line-type [position-clause] CONTENTS-clause [WHERE-clause]

When using the INSERT function, you need not insert values for all fields in a line type. Only the fields specified in the Query statement are added. An example of a Query statement using the INSERT function is as follows:

```
INSERT 01      CONTENTS MNUM = 4444, MNAM = 'JONES, BILL'
FROM PAY1
```

You must specify both the line identifier and the CONTENTS clause for every inserted line. The fields listed in the CONTENTS clause must be in the line type specified as the line ID. Lines are inserted in the order in which you specify them. Therefore, in data base files you must specify a 01 line type first if the INSERT is creating a new record.

3.3.1 CONTENTS Clause

The INSERT function requires the use of the CONTENTS clause. The CONTENTS clause serves a purpose similar to the report line in the LIST function. You specify both the field IDs to be affected and their new values. The format of the CONTENTS clause is as follows:

CONTENTS-clause ::= CONTENTS { field-type / key-type } = { constant / variable / change-data }

[[,] { field-type / key-type } = { constant / variable / change-data }] ...

3.3.2 POSITION Clause

The POSITION clause identifies the place where the new line will be inserted in every qualified record of the file. The format of the POSITION clause is as follows:

position-clause ::= { BEFORE / AFTER } { FIRST line-type [WHERE-clause] / KEY }

The POSITION clause is not used for relative record or sequential files.

3.3.3 BEFORE, AFTER, and FIRST Features

BEFORE KEY inserts a new line before any existing line in the record. AFTER KEY inserts the line after any existing lines in the record.

You can specify a positioning line type to insert the new line after or before every occurrence of the positioning line type in a record. For example, if you specify a WHERE condition, the new line is inserted before or after every occurrence of the positioning line type that meets the WHERE condition. If you include the word FIRST, the line is inserted before or after only the first occurrence in a record of the positioning line type that meets the WHERE condition. If you do not specify a POSITION clause, AFTER KEY is assumed. The positioning clause affects only the single insertion line with which it is associated. If multiple lines are being inserted, those that require more than the default positioning must have their own POSITION clause. The default for the POSITION clause is the last line in the record.

Some examples of the INSERT function using the POSITION clause are as follows:

INSERT 03 CONTENTS FLD1 = 1 ; FROM PAY1
One 03 line is inserted after every line in every record in the file.

INSERT 03 AFTER 02 CONTENTS FLD = 1 ; FROM PAY1
One 03 line is inserted after every 02 line in the file.

INSERT 03 AFTER 02 WHERE FLD2 GT 10 CONTENTS FLD1 = 1 FROM PAY1
One 03 line is inserted after every 02 line with FLD2 greater than 10. If more than one 02 line meets this condition in a record, a 03 line is inserted after each 02 line that meets the condition.

INSERT 03 AFTER FIRST 02 WHERE FLD2 GT 10 CONTENTS FLD1 = 1 FROM PAY1
One 03 line is inserted after the first 02 line in each record with FLD2 greater than 10. Only one 03 is inserted per record.

3.3.4 Identifying the Primary Key

To insert a 01 line in a data base file or any line type in a relative record file, specify the primary key; otherwise, an error condition results. If you do not specify conditions or primary key values for data base files, the line is inserted in every record in the file. If you specify a record-level condition, the line is inserted only in the records that meet that condition. If the primary key is a KIF group key, the individual field names within the group key must be specified, not the group key name.

You can identify the primary key in any of the following three ways:

- Include the primary key in the CONTENTS clause for every line to be inserted, not just the first line or the 01 line.
- Specify the primary key in the condition for the POSITION clause by testing the equality of the primary key to a value. Again, you must specify the primary key for each separate line to be inserted. This does not apply to key indexed, relative record, or sequential files.
- Specify the primary key in the record-level condition by testing the primary key's equality to a value or values. This affects all lines to be inserted, and the single record-level condition is sufficient to identify the primary key for all line types. To specify a group key in this manner, set each element of the group key equal to a value "AND".

NOTE

If the primary key is a KIF group key, the individual field names within the group key must be specified, not the group key name.

3.4 UPDATE FUNCTION

The UPDATE function modifies information stored in a file. The UPDATE function requires the CONTENTS clause and has the following format:

UPDATE-clause :: = UPDATE [trace-indicator] modification-line
[; modification-line]. . . [;]

modification-line :: = line-type [position-clause] CONTENTS-clause [WHERE-clause]

The fields specified in the CONTENTS clause must be in the line type specified by the line ID. The WHERE condition can test only fields from that line and the primary key. This condition is a line-level condition and applies only to the preceding single update line. UPDATE functions can have line-level conditions associated with each line ID specified and a single record-level condition.

UPDATE does not automatically modify secondary keys. To modify secondary keys, first delete the line and then reinsert it with the new values. Rewriting of secondary keys is not allowed.

Some examples of the UPDATE function using the CONTENTS clause are as follows:

UPDATE 01 CONTENTS MNAM = 'JONES, MARY' FROM PAY1 WHERE MNUM = 12345
Updates the name of employee #12345, changing it from SMITH, MARY to JONES, MARY.

UPDATE CU CONTENTS MLOC = 'DOWNTOWN' WHERE MLOC = "MT. VIEW"; FROM PAY1
All employees at the MT. VIEW site are being relocated to the DOWNTOWN site. This Query changes all current job locations accordingly.

3.5 DELETE FUNCTION

The DELETE function deletes one or more lines or entire records. The DELETE RECORD function deletes the entire file. In the DELETE function, you need specify only the line type identifiers. The format of the DELETE function is as follows:

DELETE-clause :: = DELETE [trace-indicator] { RECORD / delete-line [; delete-line]. . . } [;]

delete-line :: = line type [WHERE-clause]

The WHERE clause is a line-level condition and applies only to the single delete line with which it is specified. The line-level condition can test only fields from the delete line type or the primary key. You can specify a single record-level condition.

Lines are deleted in the order in which you specify them. Therefore, if you are deleting a 01 line type (that is, the entire record is being deleted), specify the 01 line type last. This is necessary because a 01 line must be the last line deleted in a record.

Some examples of the DELETE function using the WHERE clause are as follows:

DELETE RECORD FROM PAY1 WHERE EVERY MSTT NE "TEXAS"
All employees who do not live in Texas have all information about them deleted.

DELETE ED WHERE DEGR = 'AA' FROM PAY1
Delete all education information on Associate of Arts degrees from all employee files.

DELETE PE WHERE JOBT = 'SALESMAN' FROM PAY1 WHERE MNUM = 55555.
A PE line was mistakenly entered for employee 55555; this Query deletes that line.

If you specify DELETE RECORD, all qualified records are deleted. Note that the DELETE RECORD FROM XXXX deletes all data from file XXXX. Line-level conditions do not apply to DELETE RECORD. You can specify a single record-level condition. The DELETE function is not legal for relative record or sequential files.

3.5.1 TRACE Clause

The DELETE RECORD deletes an entire file. To be sure that this is what you want, use the TRACE clause to view the results of DELETE RECORD on the file. TRACE allows you to see the results without actually making changes to the file. See Section 4 for more information.

4

Clauses

4.1 INTRODUCTION

This section contains detailed information about Query-990 clauses and explains report formatting and change data constants.

Several of the clauses are as follows:

- FROM — Identifies which file the Query statement will process.
- WHERE — Specifies test conditions that identify which records or lines in the file will be listed or modified.
- SORT — Orders output data based on the two levels of sorting (line-level and record-level). This clause can order lines within a record or order records in a file.
- TRACE — Allows you to see a listing that shows changes to the file before the file has actually been modified. The use of this clause is recommended with the DELETE function.
- BY — Controls the order in which the Query processor reads the data in the file.
- DEFINE — Specifies calculations on fields and allows the use of the calculations as report elements or operands in a condition.
- BREAK — Allows control break processing on totals, counts, and duplicate values.
- UNIQUE — Indicates a single occurrence of a specified line type per record. Fields within a UNIQUE line type can act as primary keys when used in report lines. Secondary keys within that line can be optimized.
- LINKED BY — Defines the relationship between files when using more than one file in a Query statement.

There are two types of clauses: record-level and line-level. Record-level clauses apply to the entire Query statement and act in a global manner, affecting all report lines or modification lines. They should be specified after the FROM clause or before the function.

Line-level clauses affect only the single report line with which they are specified. They must be specified after the list of fields and literals in a report line before the FROM clause.

Clauses are record-level, line-level, or both, depending on where the clauses are specified in the Query statement. A list of the different kinds of clauses is as follows:

Record-Level	Line-Level	Both
FROM	CONTENTS	WHERE
LINKED BY	TRACE	SORT
UNIQUE	BREAK	HEADER
DEFINE	THRU	FOOTING
BY		

4.2 FROM CLAUSE

The FROM clause identifies which files the Query processor will use to modify or build the report. If the function is INSERT, DELETE, or UPDATE, you can specify only one file. You can specify any number of files for the LIST function; however, if you specify more than one file, you must include the LINKED BY clause. Specify the main headings and footings, record-level sorting information, and record-level conditions after the FROM clause.

4.2.1 Syntax

The basic syntax of the FROM clause for the LIST function is as follows:

```
FROM-clause :: = FROM file-name [EXCLUSIVE] [file-name [EXCLUSIVE]] . . .
                   [LINKED-BY-clause] [SORT-clause] [HEADER-clause]
                     [BY-clause] [UNIQUE-clause]
```

The file-name is the logical DBMS-990 or DD-990 file name to be accessed. If the keyword EXCLUSIVE is included after a file name, the file will be opened with exclusive access. This can be used to ensure that no updates are being made to the file while it is being accessed by Query-990.

4.3 HEADER AND FOOTING CLAUSE

You can specify headings and footings of several types. Main headings and footings appear at the top or bottom of each page. Report line headings and footings precede or follow a report line when either the record or the report line type has changed. Default report line headings are available to automatically list DDL or alias names above the columns that contain data for that field.

4.3.1 Syntax

The basic syntax of a HEADER or FOOTING clause is as follows:

```
HEADER-clause ::= [NO HEADER] [HEADER [header-element]]
                                [FOOTING [header-element]]

                  { PAGE [digit] [digit] }
HEADER-element ::= { SKIP [digit] [digit] }
                  { DEFAULT              }
                  { header-literal       }

header-literal ::= " [literal-element]... " { " [literal-element]... " }
                                            { ' [literal-element]... ' }

                  { ascii-character }
literal-element ::= { ^SYSTIME      }
                  { ^SYSDATE      }
                  { ^PAGENUM      }
```

You can specify any number of HEADER and FOOTING clauses. You must specify report line headings before the semicolon that terminates specification of the associated report line. Specify main headings and footings after the FROM clause. Literals, paging, and skipping of lines occur in the order in which you specify them within the clause. Heading and footing literals should not be longer than the width of a page. If they are longer than the page width, the Query statement executes, but the literal is right-truncated after the page width and a warning message appears.

4.3.2 Main Headings and Footings

Main headings and footings appear at the top and bottom of each page of output. You can specify them in a HEADER or FOOTING clause that follows the FROM clause. The number of main headings and footings cannot exceed the number of lines per page. Figure 4-1 shows a Query using main headings.

Query Statement:

```
LIST 'SALES ORDER NUMBER: ' SONM
     'QUANTITY ORDERED: ' QUAN
WHERE ITEM = 555
FROM SOFL
BY KEY BY LIST
HEADER  '  ORDER INFORMATION FOR ITEM NUMBER 555'
FOOTING ' *** ITEM NUMBER 555 = GREEN JEANS'
```

Query Output:

```
  ORDER INFORMATION FOR ITEM NUMBER 555
SALES ORDER NUMBER:  100   QUANTITY ORDERED:     2
SALES ORDER NUMBER:   75   QUANTITY ORDERED:     2
SALES ORDER NUMBER:   50   QUANTITY ORDERED:   101
```

Figure 4-1. Main Headings Example

4.3.3 Report Line Headings and Footings

Specify report line headings and footings within the report line to which they apply. You can use SKIP and PAGE in a report line HEADER or FOOTING clause. Report line headings for a specific report line are printed when the following hold true:

- The next output line has the report line type of those headings.
- The previous output line had a different report line type or was in a different record.

Report line footings are printed when the following hold true:

- The last output line printed has the same report line type of those footings.
- The next output line has a different report line type or is in a different record.

Figure 4-2 shows a Query using report line header and footing and introduces the PAGE option.

Query Statement:

```
LIST 'INVOICE NUMBER: ' SONM
     HEADER '***** SALES ORDER - ^SYSDATE **************************' SKIP;

     'ITEM NUMBER: ' ITEM   '  QUANITY ORDERED: ' QUAN
     FOOTING SKIP 2
     SORTED BY ITEM;

FROM SOFL
NO HEADER
FOOTING  '---------------------------------------- ^PAGENUM --'
SORTED BY SONM
BY KEY BY LIST
```

Query Output:

```
***** SALES ORDER - 06/02/82 ***************************

INVOICE NUMBER:  50
ITEM NUMBER: 555   QUANITY ORDERED:  101
ITEM NUMBER: 777   QUANITY ORDERED:    5


***** SALES ORDER - 06/02/82 ***************************

INVOICE NUMBER:  75
ITEM NUMBER: 111   QUANITY ORDERED:    3
ITEM NUMBER: 333   QUANITY ORDERED:    1
ITEM NUMBER: 555   QUANITY ORDERED:    2
ITEM NUMBER: 777   QUANITY ORDERED:    4


***** SALES ORDER - 06/02/82 ***************************

INVOICE NUMBER: 100
ITEM NUMBER: 333   QUANITY ORDERED:    1
ITEM NUMBER: 555   QUANITY ORDERED:    2


***** SALES ORDER - 06/02/82 ***************************

INVOICE NUMBER: 300
ITEM NUMBER: 777   QUANITY ORDERED:    5


---------------------------------------- PAGE   1 --
```

Figure 4-2. Report Line Heading Example

Each report line has a special default report line literal. The Query processor automatically composes this literal, which consists of the DDL ID of all fields in the report line above the columns in which the fields appear. If you include the word HEADER or FOOTING with no literal, PAGE command, or SKIP command, the default heading is assumed. You can also specify the default literal by using the word DEFAULT in the HEADER or FOOTING clause. Only one default heading or footing is built for a report line.

Figure 4-3 shows a Query using the default report line header.

Query Statement:

```
LIST 'EMPLOYEE NAME: ' MNAM
     HEADER  '*************************************************************';
     ED HEADER DEFAULT
     FOOTING '*************************************************************';
FROM PAY1 SORTED BY DEGR BY KEY BY LIST
```

Query Output:

```
************************************************************
EMPLOYEE NAME: MEREDITH, JOHN
DEGR  YEAR  COLL                  GPA
AA    1968  PLUMBERS SCHOOL       2.5
************************************************************
************************************************************
EMPLOYEE NAME: HOWELL, JOHN
************************************************************
EMPLOYEE NAME: BROWN, WILLIE
************************************************************
EMPLOYEE NAME: STEPHENS, JANET
DEGR  YEAR  COLL                  GPA
BA    1975  JOURNALISM SCHOOL     2.9
************************************************************
************************************************************
EMPLOYEE NAME: ABLE, CHARLIE
DEGR  YEAR  COLL                  GPA
BA    1971  DETECTIVE COLLEGE     3.9
************************************************************
************************************************************
EMPLOYEE NAME: PARKS, FRED
DEGR  YEAR  COLL                  GPA
BA    1966  SALES COLLEGE         3.6
************************************************************
************************************************************
EMPLOYEE NAME: LI, KIM
DEGR  YEAR  COLL                  GPA
BS    1977  PROGRAMMING SCHOOL    3.9
MS    1979  GRADUATE SCHOOL       3.7
************************************************************
************************************************************
EMPLOYEE NAME: PASCHAL, JIMMY
DEGR  YEAR  COLL                  GPA
MA    1946  MT. VIEW COLLEGE      3.1
BBA   1941  BUSINESS COLLEGE      2.4
************************************************************
************************************************************
EMPLOYEE NAME: HAYNES, BILL
DEGR  YEAR  COLL                  GPA
MA    1976  SUPERVISOR COLLEGE    3.0
************************************************************
```

Figure 4-3. Default Heading Example

4.3.4 Special Heading Constants

The three special constants that can be included in any heading or footing literal are ^PAGENUM, ^SYSDATE, and ^SYSTIME. These constants are replaced as follows with the page number, current date, and current time when the heading or footing is printed:

- ^PAGENUM is replaced with PAGE nnn, where nnn is the current page number. Page numbering is automatic and begins with page 1 if the Query statement is not listed; otherwise, it begins with page 2.
- ^SYSDATE is replaced with a date in the form MM/DD/YY, where MM is the month, DD is the day of the month, and YY is the last two digits of the year.
- ^SYSTIME is replaced with the time in the form HH:MM:SS, where HH is the hour in the range of 0 through 23, MM is the minutes, and SS is the seconds.

Figure 4-4 shows a Query using a main heading, the SKIP option, and special heading constants.

Query Statement:

```
LIST MNAM MJOB MCTY
FROM PAY1 SORTED BY MNAM BY KEY BY LIST
HEADER SKIP 2
'                MONTHLY EMPLOYEE REPORT' SKIP
'                 PREPARED ON  ^SYSDATE'   SKIP 2
'EMPLOYEE NAME          JOB TITLE  CITY STATIONED'
'-------------          ---------  --------------' SKIP
FOOTING '* COPY TO MR. SMITH, PERSONNEL DIRECTOR'
```

Query Output:

```
                MONTHLY EMPLOYEE REPORT

                 PREPARED ON MM/DD/YY

EMPLOYEE NAME          JOB TITLE  CITY STATIONED
-------------          ---------  --------------

ABLE, CHARLIE          DETECTIVE   ASPERMONT
BROWN, WILLIE          SHOEMAKER   NEW YORK
HAYNES, BILL           PROD SUPV   DEL CURTO
HOWELL, JOHN           MECHANIC    GRANGER
LI, KIM                PROGRAMMER  BROOKSIDE
MEREDITH, JOHN         PLUMBER     GOLIAD
PARKS, FRED            SALESMAN    RUSK
PASCHAL, JIMMY         VICE PRES   LIBERTY HILL
STEPHENS, JANET        REPORTER    ECHO

* COPY TO MR. SMITH, PERSONNEL DIRECTOR
```

Figure 4-4. Special Heading Example

4.3.5 System Heading and NO HEADER Clause

A system heading with the following format is automatically printed at the top of every page:

QUERY L.V.R. YY.DDD QUERY-990 MM/DD/YY HH:MM:SS ΛPAGE nnn

where:

L.V.R identifies the version of the Query processor.
YY.DDD is the Julian release date for that version.
MM/DD/YY indicates the month, day, and year (referenced as ΛSYSDATE).
HH:MM:SS indicates the hour, minutes, and seconds (referenced as ΛSYSTIME).
ΛPAGE nnn indicates the page number (referenced as ΛPAGENUM).

You can suppress the system heading by specifying NO HEADER after the FROM clause. Since NO HEADER applies only to the system header, you can still define other main headings and footings.

Figure 4-5 shows a Query using the NO HEADER clause.

4.4 REPORT OUTPUT

You can obtain two types of output when using the LIST function: formatted report and unformatted. You select the type of output when the Query-990 processor is initiated in the REPORT PARAMETERS screen.

- Formatted report output is intended to be read by users. If the output is in report format, Query-990 converts all data to a readable format with carriage controls included to make the report print correctly.

- Unformatted data serves as input to another program or utility. If output is unformatted, Query-990 does not convert the data, and headings and footings are not allowed. Each report line produces one record in the output file, with no carriage control included.

4.4.1 Formatting

You control the formatting of a report line by changing the output length of a field, specifying tab stops, and specifying spacing between fields or literals.

To change the output length of a field, enter a colon and the new output length after the field ID. The data types with their default lengths are as follows:

Data Type	Default Length
CH	Number of characters defined
CN, AN	Number of digits plus 1 if decimal point is needed
CS, AS, PK	Number of digits plus 1 for sign and plus 1 for decimal point, if needed
IS	5 characters
ID, RS	10 characters
RD	18 characters
LG	5 characters (TRUE or FALSE)

Query Statement:

```
LIST MNAM
     HEADER  '*****************************';
     MSTR;
     MCTY ', ' MSTT '  ' MZIP
     FOOTING '*****************************';
FROM PAY1 SORTED BY MNAM BY KEY BY LIST
NO HEADER HEADER 'MAILING LABELS FOR EMPLOYEE MAIL' SKIP 2
```

Query Output:

```
MAILING LABELS FOR EMPLOYEE MAIL


*****************************
ABLE, CHARLIE
2800 SKYWAY
ASPERMONT       , MO  32145
*****************************
*****************************
BROWN, WILLIE
600 W 55TH
NEW YORK        , NY  88889
*****************************
*****************************
HAYNES, BILL
500 LAIRD
DEL CURTO       , TX  85269
*****************************
*****************************
HOWELL, JOHN
555 RIO GRANDE
GRANGER         , TX  78787
*****************************
*****************************
LI, KIM
3800 TONKAWA TRAIL
BROOKSIDE       , MO  22222
*****************************
*****************************
MEREDITH, JOHN
98 N. LAMAR
GOLIAD          , TX  89898
*****************************
*****************************
PARKS, FRED
200 NEW YORK AVE.
RUSK            , NY  78998
*****************************
*****************************
PASCHAL, JIMMY
1000 ACORN OAKS
LIBERTY HILL    , MO  79666
*****************************
*****************************
STEPHENS, JANET
56 PURNAM DR
ECHO            , TX  87989
*****************************
```

Figure 4-5. NO HEADER Clause Example

If you specify an output length greater than the default length, the field is blank filled to the right. If you specify an output length less than the default length, all data is truncated to the right.

4.4.2 TAB

To specify a tab stop, enter the word TAB followed by a column number. Specify tab stops in increasing values. The field that follows the tab stop specification begins in the column specified in the tab stop. Fields that follow are positioned in the normal manner after the end of the field or literal preceding them, with a default value of two spaces between fields. If a report line requires more than one line of output, you can specify tab stops on lines other than the first line.

4.4.3 SPACE

To specify spacing other than the default between fields and literals, enter the word SPACE followed by a number. The specified number of blanks are written in the current position in the report line. For example, if you write SPACE 0 between two field names, no spaces are written between the fields; if you write SPACE 30, 30 spaces are written between them. In the initial release of Query-990, the user indicated spacing by specifying the number of spaces, followed by an X. Although the current release supports this method (for compatibility), use of SPACE is encouraged for its readability.

Figure 4-6 shows a Query using SPACE, TAB, and output lengths.

Query Statement:

```
LIST MNAM:15 TAB 19 JOBT:11 SPACE 5 COMP:20 TAB 60 PSAL
FROM PAY1 SORTED BY MNAM BY KEY BY LIST
NO HEADER HEADER
'                PREVIOUS POSITIONS HELD BY OUR EMPLOYEES' SKIP 2
'EMPLOYEE NAME     JOB TITLE         COMPANY           PREVIOUS SALARY'
'-------------     ---------         -------           ---------------'
```

Query Output:

```
                    PREVIOUS POSITIONS HELD BY OUR EMPLOYEES

EMPLOYEE NAME      JOB TITLE         COMPANY           PREVIOUS SALARY
-------------      ---------         -------           ---------------
ABLE, CHARLIE      POLICEMAN      POLICE DEPT              1700.00
HAYNES, BILL       SALESMAN       EQUIPMENT MFG.           1500.00
HOWELL, JOHN       MECHANIC       GREASE MONKEY LTD         270.00
MEREDITH, JOHN     JOURNEYMAN     H & H COMPANY             750.00
PARKS, FRED        SALESMAN       DEEP HOLE SALES INC       379.50
PASCHAL, JIMMY     SALESMAN       EQUIPMENT MFG.           1500.00
STEPHENS, JANET    COPY EDITOR    JOURNALISM SCHOOL         150.00
```

Figure 4-6. Space, Tab, and Formatting Example

4.4.4 PAGE and SKIP

You can use the PAGE command to skip a specified number of pages. To specify the number, enter the following in any report line heading or footing clause:

PAGE n

where:

n is the number of pages to skip.

If you do not specify a number, one page is skipped. Any main footings are printed before the next page begins, and any main headings are printed at the top of the next page. PAGE is not allowed in a main heading or footing.

Use the SKIP command in any heading or footing clause to skip a specified number of lines. If you do not specify a number, one line is skipped. The format of the SKIP command is as follows:

SKIP n

where:

n is the number of lines to skip.

4.4.5 Literals

You can include any literal in a report line. Literals appear in the order in which you specify them. If you specify a literal between two fields, two spaces are not inserted between the values for each field. You can use change data constants as literals in the report line. They are treated as character data, and you must specify their length.

4.5 WHERE CLAUSE

The WHERE clause specifies test conditions that identify which records or lines in the file will be listed or modified. Test conditions are Boolean expressions consisting of fields, DEFINE variables, constants, relational operators, the Boolean operators AND and OR, and parentheses to control the order of evaluation. The two types of test conditions are record-level conditions and line-level conditions. The syntax for the WHERE clause is as follows:

WHERE-clause :: = WHERE condition

condition :: = { simple-condition | complex-condition }

A record-level condition tests all of the information in a record and determines whether to print any data from the record. You can specify only one record-level condition in a Query statement. Each record in the file must pass the record-level condition before any of the lines in that record can be listed or modified. Record-level tests can compare fields from different line types. Because a line type can occur any number of times in a record, record-level conditions require that you specify the quantifier ANY or EVERY before each field name tested. If ANY is specified, only one occurrence of the field in the record must meet the condition; if EVERY is specified, all occurrences of the field in the record must meet the condition. A condition must have at least one ANY or EVERY or test only the primary key for Query-990 to recognize it as the record-level condition.

Figure 4-7 shows a Query using a record-level condition on a primary key with the WHERE clause.

Query Statement:

```
LIST TAB 11 MSTR;
     TAB 11 MCTY ', ' MSTT;
     TAB 17 MZIP;
FROM PAY1 WHERE MNUM = 55555 BY KEY BY LIST
NO HEADER HEADER 'ADDRESS OF EMPLOYEE NUMBER 55555:' SKIP 2
```

Query Output:

```
ADDRESS OF EMPLOYEE NUMBER 55555:

          1000 ACORN OAKS
          LIBERTY HILL    , MO
                79666
```

Figure 4-7. Record-Level Condition with WHERE Clause

A line-level condition tests only the data being used for a single report line to determine whether the line is to be used. You can specify line-level conditions for every single report or modification line in the Query statement. If a record is qualified for output or modification by the record-level condition or if no record-level condition exists, each line-level test must also be met before the report or modification line with which the test is associated will be listed or modified. Line-level tests in modification functions can test only fields in the line with which they are associated. Line-level conditions in the LIST function can test fields from several line types, but such conditions follow rules specified for multiline report lines.

Figure 4-8 shows a Query using the line-level condition with the WHERE clause.

Query Statement:

```
LIST 'SALES ORDER NUMBER:  ' SONM
     ' <-------->     '
     'QUANTITY ORDERED:  ' QUAN
WHERE ITEM = 555 SORTED BY QUAN
FROM SOFL BY KEY BY LIST
```

Query Output:

```
SALES ORDER NUMBER:  100    <-------->    QUANTITY ORDERED:     2
SALES ORDER NUMBER:   75    <-------->    QUANTITY ORDERED:     2
SALES ORDER NUMBER:   50    <-------->    QUANTITY ORDERED:   101
```

Figure 4-8. Line-Level Condition with WHERE Clause

Query-990 allows you to specify a WHERE clause inside each report line specification. This condition applies only to the single report line specified. If several report lines have the same line type, each report line can have its own individual line-level conditions. Query statements containing line-level conditions and formatted in the Query-990 1.0 release should be reformatted so that the line-level conditions are included with the report line to which they apply. The meaning of the condition is clearer in the current format.

4.5.1 Syntax

A WHERE clause can define either simple conditions or complex conditions formed from simple conditions connected by the logical operators AND and OR.

4.5.1.1 Simple Conditions. A simple condition has the following form:

simple-condition ::= [(] [quantifier] op1 rel-op [quantifier] op2 [)]

Figure 4-9 shows a Query using a simple condition.

```
LIST MNAM MDEP FROM PAY1 BY KEY BY LIST
WHERE MNUM = 55555
```

Figure 4-9. Simple Condition Example

The relational operators have two forms: a two-letter mnemonic and a character symbol. Since the Query processor considers the two forms equivalent, they can be mixed in a single Query statement. Table 4-1 lists and explains the relational operators.

Table 4-1. Relational Operators

Symbol	Mnemonic	Meaning
=	EQ	Equal to
<>	NE	Not equal to
>	GT	Greater than
<	LT	Less than
>=	GE	Greater than or equal to
<=	LE	Less than or equal to

Figure 4-10 shows a Query using relational operators.

```
LIST MNAM MRAT WHERE MRAT GT 1000
FROM PAY1 BY KEY BY LIST
```

Figure 4-10. Relational Operators Example

Use numeric constants only for numeric data types and literals for character data. You can enclose numeric constants in quotation marks. A numeric constant must be compatible with the type of field to which it is being compared; otherwise, a conversion error results. If two fields are being compared, they must have exactly the same format.

4.5.1.2 String Operators. For character fields, certain string operators are available. You can compare the beginning characters of a field by specifying a literal with the required beginning characters followed by an ellipsis (. . .). Compare the ending characters of a field by specifying a literal preceded by an ellipsis. You can use any of the relational operators. For example, to determine whether a NAME field 20 characters long begins with 5 characters greater than or equal to "SMITH", the test condition is as follows:

NAME GE "SMITH"...

To specify a search to determine whether a field contains a certain string, compare testing for equality between a field and a literal that is surrounded by ellipses. For example, to determine whether an address field (ADDR) 50 characters long contains "CHICAGO", the test condition is as follows:

ADDR = ..."CHICAGO"...

Figure 4-11 shows a Query using string operators.

Query Statement:

```
LIST TAB 4 ITMN TAB 18 DESC WHERE DESC = ... 'RING'...
FROM ITEM BY KEY BY LIST
HEADER 'ITEM NUMBER       DESCRIPTION'
       '-----------       -----------'
```

Query Output:

```
ITEM NUMBER       DESCRIPTION
-----------       -----------
   777            RED HERRING
   333            GOLDEN RING
```

Figure 4-11. String Operators Example

4.5.1.3 Complex Conditions. Complex conditions are formed from simple conditions connected by AND and OR. The basic syntax of a complex condition is as follows.

complex-condition ::= [(] simple-condition [log-op complex-condition] [)]

Notice that the right-hand side of this definition is a complex condition itself. This means that any number of simple conditions can be connected together.

The result of a complex condition using AND is true only if both conditions are true. The result of a complex condition using OR is true if one of the conditions is true.

Figure 4-12 shows a Query using a complex condition using the logical operators AND and OR and parentheses.

Query Statement:

```
LIST MNAM MCTY MSTT
     WHERE (MCTY = 'GOLIAD' OR MCTY = 'ECHO') AND
           MSTT = 'TX'
FROM PAY1 BY KEY BY LIST
HEADER 'EMPLOYEE NAME      CITY STATIONED       STATE'
       '-------------      --------------       -----'
```

Query Output:

```
EMPLOYEE NAME      CITY STATIONED       STATE
-------------      --------------       -----
MEREDITH, JOHN         GOLIAD            TX
STEPHENS, JANET        ECHO              TX
```

Figure 4-12. Complex Condition with Logical Operators Example

The components of a condition are evaluated in the following order:

1. All simple conditions are evaluated.
2. All AND operations are performed from left to right.
3. All OR operations are performed from left to right.

You can use parentheses to change the order of evaluation. The condition inside the innermost parentheses is evaluated first, effectively removing them. Then, the expression in the next innermost parentheses is evaluated, until all parentheses are removed.

4.5.2 Record-Level Conditions

You can test a record by using a record-level condition. Only one record-level condition is allowed per Query statement. The record-level condition must be specified in the main WHERE clause. Record-level conditions are not allowed when the function is LIST and the sequence is BY LIST.

The record-level condition can test any field(s) within a record. Record-level conditions require the use of quantifiers unless only the primary key is tested. Conditions that test only the primary key are assumed to be record-level conditions if they are in the main WHERE clause. Otherwise, the condition will be treated as line-level and applied to only one of the report lines.

Figure 4-13 shows a Query using the record-level condition with a primary key.

Query Statement:

```
LIST 'EMPLOYEE RECORD FOR:   ' MNAM;
     TAB 81
     '               ADDRESS:  ' MCTY MSTT MZIP;
     '             JOB TITLE:  ' MJOB;
     '            DEPARTMENT:  ' MDEP;
     '          RATE OF PAY:   ' MRAT;
     TAB 81
     '   PREVIOUS JOB TITLE:   ' JOBT;
     ' COMPANY EMPLOYED BY:    ' COMP;
FROM PAY1
WHERE MNUM = 55555
```

Query Output:

```
EMPLOYEE RECORD FOR:  PASCHAL, JIMMY

              ADDRESS: LIBERTY HILL      MO  79666
            JOB TITLE: VICE PRES
           DEPARTMENT: SALES
         RATE OF PAY:   2500.00

  PREVIOUS JOB TITLE: SALESMAN
COMPANY EMPLOYED BY: EQUIPMENT MFG.
```

Figure 4-13. Record-Level Example

4.5.3 EVERY and ANY Quantifiers

Quantifiers are EVERY and ANY. When the word EVERY precedes a field name, every occurrence of that field in the record must meet the specified condition. Otherwise, the condition is false. When the word ANY precedes a field name, only one occurrence of the field must meet the condition for the condition to be true. You must use at least one quantifier in a condition that tests fields other than the primary key in order to identify it as a record-level condition. If one quantifier is used, any fields not preceded by a quantifier are treated as if they are preceded by ANY. Quantifiers cannot precede constants.

When both operands of the simple condition are fields, the following cases are possible:

- ANY field1 operator ANY field2 — Any occurrence of field1 must satisfy the relational operator for any field2 in the record.
- EVERY field1 operator EVERY field2 — Every occurrence of field1 in the record must satisfy the relational operator for every occurrence of field2 in the record.
- ANY field1 operator EVERY field2 — Any occurrence of field1 in the record must satisfy the relational operator for every occurrence of field2 in the record.
- EVERY field1 operator ANY field2 — Every occurrence of field1 in the record must satisfy the relational operator for some occurrence of field2 in the record.

If a record-level condition exists, it is tested first, before any of the lines in the record are considered for output or modification. If the condition is false, the record is skipped. If the condition is true, individual lines in the record must also meet their line-level condition, if one exists, before they are modified or output.

Figure 4-14 shows a Query using ANY and EVERY quantifiers.

Query Statement:

```
LIST 'SALESMAN:   ' MNAM
     '<------>    '
     'SALES FOR LAST MONTH:  ' MSLS;
FROM PAY1 WHERE ANY MSLS GT 0 BY KEY BY LIST
NO HEADER HEADER
'                              SALESMAN REPORT'  SKIP 2
```

Query Output:

```
                              SALESMAN REPORT


SALESMAN:   LI, KIM          <------> SALES FOR LAST MONTH:   5000.000
SALESMAN:   BROWN, WILLIE    <------> SALES FOR LAST MONTH:    150.000
SALESMAN:   PARKS, FRED      <------> SALES FOR LAST MONTH:    300.000
```

Figure 4-14. Record-Level Condition Example

4.5.4 Line-Level Conditions

You can specify line-level conditions in a WHERE clause included with a report line or modification line specification. If a record has passed the record-level condition or no record-level condition exists, each set of data for each report line or modification line must meet a line-level condition, if one exists. If the function is LIST and the sequence is BY KEY or BY KEY BY LIST, fields from several line types can be tested in the same line-level condition. Otherwise, line-level conditions can test fields only from one line type and/or the primary key. Quantifiers cannot be used in line-level conditions.

4.6 SORT CLAUSE

The SORT clause orders output data based on the values of fields in the file. The two levels of sorting are record level and line level. Record-level sorting orders retrieved records on the basis of field values and key values within the record. Line-level sorting orders lines within a record or report based on the values within the line.

To specify a record-level sort use a SORT clause after the FROM clause. To specify a line-level sort use a SORT clause within a report line specification. Both a record-level and line-level sort can be specified in a single Query statement if the statement is sequenced BY KEY BY LIST.

4.6.1 Syntax

The syntax for the SORT clause is as follows:

```
SORT-clause ::= SORTED BY { field-type } [order-indicator]
                          { key-type   }
                          { variable   }
                [ { field-type } [order-indicator] ] ...
                { key-type   }
                { variable   }

order-indicator ::= { : A          }
                    { : ASCENDING  }
                    { : D          }
                    { : DESCENDING }
```

where:

:A is equivalent to :ASCENDING.
:D is equivalent to :DESCENDING.

The order indicators :A and :D specify whether a particular field is to be sorted in ascending or descending order. The indicator is not required; if it is omitted, the field is sorted in ascending order. The order indicator applies only to the field preceding it. You can specify more than one sort field, and the sort order can be mixed (that is, some fields ascending and some descending). The sort fields are used as sort keys in the order in which they are specified.

4.6.2 Record-Level Sort

Indicate a record-level sort by placing the SORT clause after the FROM clause. This type of sorting is meaningful only for Query statements ordered BY KEY or BY KEY BY LIST. All fields in a record are valid sort fields, as are DEFINE variables. If fields are specified from a line type that is not unique, the values in the first occurrence of that line type in each record are used as the sort value. If a specified line type does not occur in a record, the value of the sort key for that record is binary zeros.

Note that a record is composed of all of the lines associated with a primary key. (When records are sorted by fields that are not included in the report, the ordering scheme is not always obvious.) The order by which individual lines within the record are printed is not affected by the record-level SORT clause.

Figure 4-15 shows a Query using a record-level SORT clause.

Query Statement:

```
LIST TAB 6 MSSN TAB 28 MRAT TAB 45 MDDT
FROM PAY1 SORTED BY MSSN BY KEY BY LIST
HEADER
'SOCIAL SECURITY NUMBER        SALARY        DEDUCTIONS'
'----------------------        ------        ----------'  SKIP
```

Query Output:

```
SOCIAL SECURITY NUMBER        SALARY        DEDUCTIONS
----------------------        ------        ----------

      485298742                900.00          99.00
      487265478                750.00          79.88
      653878954                215.00          30.56
      654789622                385.00          58.30
      852106987               1950.00          99.50
      852369741                558.00          50.00
      852417931                230.00          55.50
      875247964               2500.00          87.20
      888894566                375.00          46.00
```

Figure 4-15. Record-Level SORT

Figure 4-16 shows a Query using a record-level SORT clause on two fields.

Query Statement:

```
LIST TAB 3 MSTT TAB 20 MNAM
FROM PAY1 SORTED BY MSTT MNAM BY KEY BY LIST
HEADER 'STATE                    NAME'
       '-----                    ----' SKIP
```

Query Output:

```
STATE                    NAME
-----                    ----

  MO                ABLE, CHARLIE
  MO                LI, KIM
  MO                PASCHAL, JIMMY
  NY                BROWN, WILLIE
  NY                PARKS, FRED
  TX                HAYNES, BILL
  TX                HOWELL, JOHN
  TX                MEREDITH, JOHN
  TX                STEPHENS, JANET
```

Figure 4-16. Record-Level SORT on Two Fields

4.6.3 Line-Level SORT

Indicate a line-level sort by including a SORT clause after all of the report line elements but before the semicolon. This type of sort is legal only when the output sequence is BY KEY BY LIST or BY LIST. You can use any number of fields in the specified line type as sort fields. These fields need not be listed as output fields to be used as sort fields. If the Query is sequenced BY LIST, the entire report line output is sorted. If the Query is sequenced BY KEY BY LIST, report lines are sorted separately within each record for each line-level sort. Figure 4-17 shows a Query using the line-level SORT.

Query Statement:

```
LIST 'EMPLOYEE NAME: ' MNAM
     TAB 81 '-------------------------------------'
     HEADER SKIP 2 'EDUCATION INFORMATION';
     ED SORTED BY DEGR
     HEADER 'DEGR YEAR      COLLEGE           GPA';
FROM PAY1
BY KEY BY LIST
WHERE ANY GPA > 3.0
```

Query Output:

```
EDUCATION INFORMATION
EMPLOYEE NAME: LI, KIM
-------------------------------------
DEGR YEAR      COLLEGE           GPA
BS   1977  PROGRAMMING SCHOOL    3.9
MS   1979  GRADUATE SCHOOL       3.7


EDUCATION INFORMATION
EMPLOYEE NAME: PASCHAL, JIMMY
-------------------------------------
DEGR YEAR      COLLEGE           GPA
BBA  1941  BUSINESS COLLEGE      2.4
MA   1946  MT. VIEW COLLEGE      3.1


EDUCATION INFORMATION
EMPLOYEE NAME: ABLE, CHARLIE
-------------------------------------
DEGR YEAR      COLLEGE           GPA
BA   1971  DETECTIVE COLLEGE     3.9


EDUCATION INFORMATION
EMPLOYEE NAME: PARKS, FRED
-------------------------------------
DEGR YEAR      COLLEGE           GPA
BA   1966  SALES COLLEGE        3.6
```

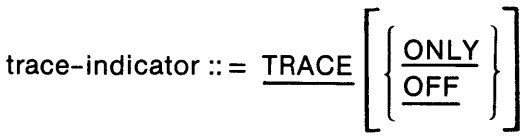
Figure 4-17. Line-Level SORT BY KEY BY LIST

4.7 TRACE CLAUSE

Executing an untested Query statement that performs an INSERT, UPDATE, or DELETE function can cause unwanted changes to the file due to logical or typing errors. The TRACE clause shows you a listing of the changes made to the file before the file is modified. This clause is particularly important when you are using the DELETE function.

4.7.1 Syntax

The syntax of the TRACE clause is as follows:



The TRACE clause follows the word INSERT, DELETE, or UPDATE in the Query statement. Specify TRACE ONLY to check the modifications to be made to the file. TRACE OFF does not perform the trace for the file and a listing file is not created. If neither TRACE ONLY nor TRACE OFF is specified, TRACE ON is assumed. TRACE ON performs both the trace and the modifications to the file. If this is not what was intended, the results can be undesirable. Generally, TRACE ONLY and TRACE OFF should be specified. For each change made to the file, a TRACE line will be written with the following format:

```
LINE   LOC1        LOC2        key      field1   field2. . . fieldn
 xx    xxxxxxxxx   xxxxxxxxx   xxxxxx   xxxxxx   xxxxxx     xxxxxx
```

Enter TRACE OFF to cancel the TRACE clause.

Figure 4-18 shows a Query that uses the DELETE function with the TRACE clause.

Query Statement:

```
DELETE TRACE ONLY ED FROM PAY1
```

Query Output:

```
LINE  LOC1      LOC2      MNUM   DEGR  YEAR   COLL                 GPA
 ED   00000E40 00000B40   895203  BS    1977   PROGRAMMING SCHOOL   3.9

LINE  LOC1      LOC2      MNUM   DEGR  YEAR   COLL                 GPA
 ED   00000B40   ****     895203  MS    1979   GRADUATE SCHOOL      3.7

LINE  LOC1      LOC2      MNUM   DEGR  YEAR   COLL                 GPA
 ED   00000BA0 00000C00    55555  MA    1946   MT. VIEW COLLEGE     3.1

LINE  LOC1      LOC2      MNUM   DEGR  YEAR   COLL                 GPA
 ED   000011A0 00000C60    55555  BBA   1941   BUSINESS COLLEGE     2.4

LINE  LOC1      LOC2      MNUM   DEGR  YEAR   COLL                 GPA
 ED   00000960 00000AE0  632566  AA    1968   PLUMBERS SCHOOL      2.5

LINE  LOC1      LOC2      MNUM   DEGR  YEAR   COLL                 GPA
 ED   00000420 00000480  963285  BA    1975   JOURNALISM SCHOOL    2.9

LINE  LOC1      LOC2      MNUM   DEGR  YEAR   COLL                 GPA
 ED   000001E0 00000240  997335  BA    1971   DETECTIVE COLLEGE    3.9

LINE  LOC1      LOC2      MNUM   DEGR  YEAR   COLL                 GPA
 ED   000007E0 00001200  458795  MA    1976   SUPERVISOR COLLEGE   3.0

LINE  LOC1      LOC2      MNUM   DEGR  YEAR   COLL                 GPA
 ED   00000600 00000660   89745  BA    1966   SALES COLLEGE        3.6
```

Figure 4-18. DELETE Function Used with TRACE Clause

4.8 BY CLAUSE

The BY clause controls the order in which the Query statement reads and gathers the file; consequently, it also controls the output sequence order. The three possible orders are BY KEY BY LIST, BY KEY, and BY LIST. The BY clause syntax is as follows:

BY-clause ::= { BY KEY BY LIST | BY KEY | BY LIST }

For most Query statements, the best order is BY KEY BY LIST. Use BY KEY when you have imposed a meaningful order on the lines in a record. Use BY LIST to gain speed or efficiency or when record boundaries should be ignored. BY KEY sequencing follows the order in the file and keeps together all of the data for a single primary key. BY LIST sequencing refers to the order of the report lines in the Query statement. Therefore, BY KEY orders output according to the order of the data in the file; BY LIST orders data according to the order of the report lines in the Query statement; and BY KEY BY LIST combines the two methods.

4.8.1 BY KEY BY LIST

Query reads a file one record at a time. All data requested from each record is listed together. The report lines are output in the order in which you have listed them. The Query processor begins with the first report line. Starting at the beginning of the line, the processor reads this line and then every occurrence of the same line type, building a new report line. The processor outputs the line if it meets any line-level test and then proceeds to the next report line, and so on, until all report lines have been printed.

BY KEY BY LIST recognizes that lines have been grouped into records. However, it does not make use of the order of the different line types within the record. BY KEY BY LIST is useful when the lines in a record have no special order or when the report line order supersedes the file order.

Record-level conditions and record-level sorting are allowed when the output sequence is BY KEY BY LIST. When you use BY KEY BY LIST, optimum data access occurs when a record-level condition tests the primary key for equality to a specified value(s), provided the record-level condition does not contain an OR clause with another type of condition. A line-level condition of this type also optimizes access if you have only one report line.

You can build report lines from more than one line when the output sequence is BY KEY BY LIST. Lines that have not been designated as UNIQUE within the record are associated with each other on a one-to-one basis for each report line, as follows: If a report line references line types a, b, and c, the first report line for a given record will be built from the first occurrence of line type a, the first occurrence of line type b, and the first occurrence of line type c; the second output line for the report line will be built from the second occurrences of all of its member lines, and so on, until one of the member line types has no more occurrences.

If one of the line types used in a report line has no occurrence in a record, that report line will not be listed for that record, even if the other components exist. The line-level condition associated with a report line can also test fields from more than one line type. In fact, it can test fields from line types not used in the report line. However, a one-to-one correspondence exists between the report line and the line types (as described for output elements in the preceding paragraph).

4.8.2 BY KEY

All data requested from each record is listed together. When reading the lines in a record, the Query processor looks for any line type listed in a report line. When it finds an appropriate line type, the processor tests the report lines that use that line type to see if a line-level condition exists; if the lines meet the condition, the processor outputs them.

Record-level conditions are legal when the sequence is BY KEY. The Query processor optimizes the Query when a record-level condition tests the primary key for equality to a specified value(s), provided the record condition does not contain an OR clause with another type of condition. A line-level condition of this type also optimizes access if you have only one report line. BY KEY allows record-level sorts.

Use BY KEY when you have added lines under a key in a meaningful order. The order of the lines defines a structure within the record. For example, assume that you have created a customer file that contains a line, CC, to include information about customer complaints (such as the date of the complaint, the name of the person who handled the complaint, and the current disposition of the complaint). The file also contains a complaint description (CD) line which contains text that describes the nature of the complaint. Each CC line might require several CD lines. The CD lines immediately follow the associated CC line. For example, CC, CD, CD, CD, CC, CD, CC, CD, CD represents three complaints. The first requires three description lines, the second requires one description line, and the third requires two description lines. Use BY KEY to list the CC and CD lines in the order in which they appear in the record (that is, to avoid listing the CC lines together and then the CD lines together).

Figure 4-19 shows a Query using BY KEY BY LIST with the desired results.

Query Statement:

```
LIST TAB 81
     'EMPLOYEE NAME: ' MNAM;
     '       SALARY: ' MRAT;
     'DEGREE EARNED: ' DEGR  SORTED BY DEGR;
FROM PAY1
SORTED BY MNAM
BY KEY BY LIST
```

Query Output:

```
EMPLOYEE NAME: ABLE, CHARLIE
       SALARY:  1950.00
DEGREE EARNED: BA

EMPLOYEE NAME: BROWN, WILLIE
       SALARY:   215.00

EMPLOYEE NAME: HAYNES, BILL
       SALARY:   750.00
DEGREE EARNED: MA

EMPLOYEE NAME: HOWELL, JOHN
       SALARY:   375.00

EMPLOYEE NAME: LI, KIM
       SALARY:   230.00
DEGREE EARNED: BS
DEGREE EARNED: MS

EMPLOYEE NAME: MEREDITH, JOHN
       SALARY:   900.00
DEGREE EARNED: AA

EMPLOYEE NAME: PARKS, FRED
       SALARY:   558.00
DEGREE EARNED: BA

EMPLOYEE NAME: PASCHAL, JIMMY
       SALARY:  2500.00
DEGREE EARNED: BBA
DEGREE EARNED: MA

EMPLOYEE NAME: STEPHENS, JANET
       SALARY:   385.00
DEGREE EARNED: BA
```

Figure 4-19. BY KEY BY LIST Example

Figure 4-20 shows a Query using BY KEY BY LIST with unwanted results.

Query Statement:

```
LIST '***' ' SALES ORDER NUMBER: ' SONM '***'
     HEADER SKIP 2
     FOOTING SKIP;
     '       ITEM NUMBER --- ' ITEM;
     '  QUANTITY DESIRED ---' QUAN;
FROM SOFL SORTED BY SONM BY KEY BY LIST
```

Query Output:

```
*** SALES ORDER NUMBER:   50   ***

       ITEM NUMBER --- 555
       ITEM NUMBER --- 777
  QUANTITY DESIRED --- 101
  QUANTITY DESIRED ---   5


*** SALES ORDER NUMBER:   75   ***

       ITEM NUMBER --- 333
       ITEM NUMBER --- 555
       ITEM NUMBER --- 111
       ITEM NUMBER --- 777
  QUANTITY DESIRED ---   1
  QUANTITY DESIRED ---   2
  QUANTITY DESIRED ---   3
  QUANTITY DESIRED ---   4


*** SALES ORDER NUMBER:  100   ***

       ITEM NUMBER --- 333
       ITEM NUMBER --- 555
  QUANTITY DESIRED ---   1
  QUANTITY DESIRED ---   2


*** SALES ORDER NUMBER:  300   ***

       ITEM NUMBER --- 777
  QUANTITY DESIRED ---   5
```

Figure 4-20. BY KEY BY LIST with Unwanted Results

NOTE

Figure 4-21 illustrates the correct results for the Query using BY KEY.

If you specify the BY KEY sequence in a report line built from several nonunique line types, the lines in a record are grouped together to build each line of output. Users control the number and position of each line in a record when they insert lines. The Query processor reads through the lines for a record, looking for line types used in report lines and building the report lines as it finds the specified line types. When a report line is complete, it is written and reinitialized to empty. If a report line contains information from line type XX and Query reads another XX line, the second set of XX data is written over the first. If a record does not include all of the line types specified in a report line, that report line will not be written for that record. A line-level condition can also test fields from several line types or from a line type different from the line types in the report line. The tested and reported fields are associated in the same way that different lines in a report line are associated.

Query can only write data from one occurrence of each line type used in a report line or line-level condition. For example, assume that you need to build a report line from AA, BB, and CC lines and that the order of the lines under one key is as follows:

AA1 - BB1 - AA2 - CC1 - AA3 - BB2 - CC2 - AA4 - BB3 - CC3

The following report lines result from the Query statement LIST AA BB CC; BY KEY:

AA2 BB1 CC1
AA3 BB2 CC2
AA4 BB3 CC3

The data from AA1 has been written over by AA2. To include AA1 in the report, you must specify AA, BB, and CC as different report lines, as follows:

LIST AA; BB; CC;

The resulting report is as follows:

AA1
BB1
AA2
CC1

The report continues in this manner, in the same order as the lines exist in the record.

Figure 4-21 shows a Query using BY KEY with the desired results.

Query Statement:

```
LIST '***' ' SALES ORDER NUMBER: ' SONM '***'
     HEADER SKIP 2
     FOOTING SKIP;
     '       ITEM NUMBER --- ' ITEM;
     '  QUANTITY DESIRED ---' QUAN;
FROM SOFL SORTED BY SONM BY KEY
```

Query Output:

```
*** SALES ORDER NUMBER:   50   ***

       ITEM NUMBER --- 555
  QUANTITY DESIRED --- 101
       ITEM NUMBER --- 777
  QUANTITY DESIRED ---   5


*** SALES ORDER NUMBER:   75   ***

       ITEM NUMBER --- 333
  QUANTITY DESIRED ---   1
       ITEM NUMBER --- 555
  QUANTITY DESIRED ---   2
       ITEM NUMBER --- 111
  QUANTITY DESIRED ---   3
       ITEM NUMBER --- 777
  QUANTITY DESIRED ---   4


*** SALES ORDER NUMBER:  100   ***

       ITEM NUMBER --- 333
  QUANTITY DESIRED ---   1
       ITEM NUMBER --- 555
  QUANTITY DESIRED ---   2


*** SALES ORDER NUMBER:  300   ***

       ITEM NUMBER --- 777
  QUANTITY DESIRED ---   5
```

Figure 4-21. BY KEY Example

4.8.3 BY LIST

BY LIST does not organize lines into records when building the report. Instead, it lists all of the data for a single report line together. The first report line is listed first, then the second, and so on, until all report lines are listed.

Report lines must be built from the same line type and/or the primary key. Record-level conditions and record-level sorting are not allowed.

For a relatively full file, BY LIST executes faster than BY KEY or BY KEY BY LIST. However, if a file is almost empty, BY LIST might be significantly slower than BY KEY and BY KEY BY LIST since it causes Query to read numerous empty records. Using BY LIST optimizes access to the data for line-level conditions that test the primary key or any secondary key for equality with a value(s), provided the line-level conditions do not contain an OR clause with a condition that does not involve a key.

4.9 DEFINE CLAUSE

The DEFINE clause specifies calculations on fields and allows the use of the calculations as report elements or operands in a condition.

4.9.1 Syntax

The syntax of the DEFINE clause is as follows:

DEFINE-clause ::= DEFINE [variable : type = define-expression [;]] ...

define-expression ::= [(] subexpression [operator define-expression] [)]

Variable names must follow the syntax rules for aliases. If a variable is used as an operand in a DEFINE expression, its definition must precede its use as an operand. Each variable must have its data type defined. The data types are those defined in the DDL specification of fields. Data types have the following syntax:

type-code / digits . decimal-places

The following data types are legal for calculation: IS, ID, RS, RD, CN, CS, AN, AS, and PK. For an explanation of these data types, see Appendix B.

Figure 4-22 shows a Query using BY LIST with the desired results.

Query Statement:

```
LIST MNUM MRAT SORTED BY MRAT: DESCENDING
     HEADER 'EMPLOYEES RANKED BY SALARY';
     MNUM MCOM SORTED BY MCOM: DESCENDING
     HEADER 'EMPLOYEES RANKED BY COMMISSION';
     MNUM MSLS SORTED BY MSLS: DESCENDING
     HEADER 'EMPLOYEES RANKED BY SALES';
FROM PAY1 BY LIST
```

Query Output:

```
EMPLOYEES RANKED BY SALARY
 55555    2500.00
997335    1950.00
632566     900.00
458795     750.00
 89745     558.00
963285     385.00
 50005     375.00
895203     230.00
441887     215.00
EMPLOYEES RANKED BY COMMISSION
 55555   .300
 89745   .200
441887   .200
895203   .100
632566   .000
458795   .000
997335   .000
963285   .000
 50005   .000
EMPLOYEES RANKED BY SALES
895203       5000.000
 89745        300.000
441887        150.000
997335           .000
963285           .000
632566           .000
 50005           .000
458795           .000
 55555      -1000.000
```

Figure 4-22. BY LIST Example

Figure 4-23 shows a Query using the BY LIST with unwanted results.

Query Statement:

```
LIST 'EMPLOYEE NAME:       ' MNAM;
     '      DEGREE EARNED:      ' DEGR;
     '      COLLEGE ATTENDED:  ' COLL;
FROM PAY1 BY LIST
```

Query Output:

```
EMPLOYEE NAME:       PARKS, FRED
EMPLOYEE NAME:       HAYNES, BILL
EMPLOYEE NAME:       ABLE, CHARLIE
EMPLOYEE NAME:       STEPHENS, JANET
EMPLOYEE NAME:       BROWN, WILLIE
EMPLOYEE NAME:       HOWELL, JOHN
EMPLOYEE NAME:       LI, KIM
EMPLOYEE NAME:       MEREDITH, JOHN
EMPLOYEE NAME:       PASCHAL, JIMMY
      DEGREE EARNED:      BA
      DEGREE EARNED:      BA
      DEGREE EARNED:      BA
      DEGREE EARNED:      MA
      DEGREE EARNED:      AA
      DEGREE EARNED:      MS
      DEGREE EARNED:      MA
      DEGREE EARNED:      BS
      DEGREE EARNED:      BBA
      COLLEGE ATTENDED:  DETECTIVE COLLEGE
      COLLEGE ATTENDED:  JOURNALISM SCHOOL
      COLLEGE ATTENDED:  SALES COLLEGE
      COLLEGE ATTENDED:  SUPERVISOR COLLEGE
      COLLEGE ATTENDED:  PLUMBERS SCHOOL
      COLLEGE ATTENDED:  GRADUATE SCHOOL
      COLLEGE ATTENDED:  MT. VIEW COLLEGE
      COLLEGE ATTENDED:  PROGRAMMING SCHOOL
      COLLEGE ATTENDED:  BUSINESS COLLEGE
```

Figure 4-23. BY LIST with Unwanted Results

Figure 4-24 shows a Query using the DEFINE clause with different data types.

Query Statement:

```
DEFINE RAISE : CN/7.2 = .20 * MRAT;
       NEW-SALARY : AN/7.2 = MRAT + RAISE;
LIST MNAM:14 TAB 20 MRAT TAB 34 RAISE TAB 50 NEW-SALARY;
FROM PAY1
BY KEY BY LIST
WHERE ANY MPYP = 30
NO HEADER HEADER
  '       YEARLY RAISE REPORT FOR MONTHLY PAID EMPLOYEES' SKIP 2
  'EMPLOYEE NAME      OLD SALARY  AMOUNT OF RAISE   NEW SALARY'
  '-------------      ----------  ---------------   ----------'
```

Query Output:

```
        YEARLY RAISE REPORT FOR MONTHLY PAID EMPLOYEES

EMPLOYEE NAME       OLD SALARY  AMOUNT OF RAISE   NEW SALARY
-------------       ----------  ---------------   ----------
PASCHAL, JIMMY       2500.00        500.00         3000.00
MEREDITH, JOHN        900.00        180.00         1080.00
ABLE, CHARLIE        1950.00        390.00         2340.00
```

Figure 4-24. DEFINE Clause Example

4.9.2 Where Variables Can Be Used

Variables specified in the DEFINE clause can be used in the LIST function anywhere that field IDs can be used. The variables can be output, sorted, or tested. In modification functions, they can be used as test variables or on the right side of an equal sign in the CONTENTS clause.

4.9.3 DEFINE Expression

A DEFINE expression uses the operators +, −, *, and /. Totals, counts, record totals, and record counts are also allowed. You can use parentheses to change the order of evaluation. Expressions are evaluated according to the following rules, in the order shown:

1. Innermost parentheses are evaluated first.
2. TOTAL, COUNT, RECORD TOTAL, and RECORD COUNT are performed left to right.
3. Multiply and divide (* and /) are performed left to right.
4. Addition and subtraction (+ and −) are performed left to right.

The syntax for a DEFINE expression is as follows:

define expression ::= [(] subexpression [operator define-expression] [)]

A subexpression is defined as follows:

subexpression ::= [(] { [RECORD] TOTAL field-type | [RECORD] COUNT field-type | field-type | variable | change-data } [)]

Notice that the right side of the definition of a DEFINE expression is itself a DEFINE expression. Accordingly, DEFINE expressions can be built up to be as complex as is necessary.

Figure 4-25 shows a Query using the DEFINE expression with COUNT.

Query Statement:

```
DEFINE ADJUSTED-SALARY : CN/7.2 = 30 / MPYP * MRAT;
       OVER-1000 : CN/2.0 = COUNT MNAM;
LIST TAB 9 MNAM TAB 42 ADJUSTED-SALARY;
     TAB 161 'NUMBER OF EMPLOYEES THAT MAKE OVER $1000.00 A MONTH: '
     OVER-1000;
FROM PAY1 BY KEY BY LIST WHERE ANY ADJUSTED-SALARY GT 1000
NO HEADER HEADER
'                  EMPLOYEES IN THE 1000 CLUB' SKIP 2
'        EMPLOYEE NAME                    MONTHLY SALARY'
'        -------------                    --------------'
```

Query Output:

```
                    EMPLOYEES IN THE 1000 CLUB

        EMPLOYEE NAME                    MONTHLY SALARY
        -------------                    --------------
        PASCHAL, JIMMY                      2500.00
        HOWELL, JOHN                        1875.00
        BROWN, WILLIE                       1290.00
        STEPHENS, JANET                     1647.80
        ABLE, CHARLIE                       1950.00
        HAYNES, BILL                        1500.00
        PARKS, FRED                         2388.24


NUMBER OF EMPLOYEES THAT MAKE OVER $1000.00 A MONTH:  7
```

Figure 4-25. DEFINE Expression Example Using COUNT

Figure 4-26 shows a Query using the DEFINE expression with TOTAL.

Query Statement:

```
DEFINE ADJUSTED-SALARY : CN/7.2 = 30 / MPYP * MRAT;
       MONTHLY-SALARY-EXPENSE : CN/8.2 = TOTAL ADJUSTED-SALARY;
LIST TAB 7 MNAM TAB 55 ADJUSTED-SALARY;
     TAB 55 '--------'
     TAB 87 'TOTAL SALARY EXPENSE FOR THIS MONTH:'
     TAB 134 MONTHLY-SALARY-EXPENSE;
FROM PAY1 BY KEY BY LIST
NO HEADER HEADER
'                          MONTHLY SALARY REPORT' SKIP
'                           PREPARED ON ^ SYSDATE' SKIP 2
'       EMPLOYEE NAME                                        SALARY'
'       -------------                                        ------'
```

Query Output:

```
                          MONTHLY SALARY REPORT

                           PREPARED ON MM/DD/YY


        EMPLOYEE NAME                                        SALARY
        -------------                                        ------
      LI, KIM                                                 460.00
      PASCHAL, JIMMY                                         2500.00
      MEREDITH, JOHN                                          900.00
      HOWELL, JOHN                                           1875.00
      BROWN, WILLIE                                          1290.00
      STEPHENS, JANET                                        1647.80
      ABLE, CHARLIE                                          1950.00
      HAYNES, BILL                                           1500.00
      PARKS, FRED                                            2388.24
                                                            ---------
      TOTAL SALARY EXPENSE FOR THIS MONTH:                  14511.04
```

Figure 4-26. DEFINE Expression Example Using TOTAL

4.9.4 Mixed Mode Arithmetic

You can mix COBOL data types (CS, CN, AS, AN, IS, and PK) in the same expression. You can also mix FORTRAN or Pascal data types (IS, ID, RS, and RD). However, you cannot mix COBOL types with FORTRAN/Pascal types. For example, an expression cannot mix operands RD and CN.

Before the calculation is performed, constants in the DEFINE expression are converted to the same type as the result.

4.9.5 Totals and Counts

Totals and counts are legal only for the LIST function. The two types of totaling and counting are as follows:

- When you specify the word TOTAL or COUNT for a field or expression, a total or count of the field or expression is accumulated over the entire file.
- When you precede TOTAL or COUNT by the word RECORD, the total or count is performed on a record basis only and the value is reinitialized to zero when a new record is being read.

Figure 4-27 shows a Query using the DEFINE clause with RECORD COUNT.

Query Statement:

```
DEFINE NUMBER-DEGREES : CN/1.0 = RECORD COUNT DEGR;
LIST 'DEGREE INFORMATION ON: ' MNAM;
     TAB 2 DEGR TAB 18 COLL
     HEADER SKIP 'DEGREE                COLLEGE'
                 '------                -------';
     TAB 81 'TOTAL NUMBER OF DDEGREES EARNED: ' NUMBER-DEGREES;
FROM PAY1 WHERE ANY MNAM = 'LI, KIM'
NO HEADER BY KEY BY LIST
```

Query Output:

```
DEGREE INFORMATION ON: LI, KIM

DEGREE                COLLEGE
------                -------
 BS                PROGRAMMING SCHOOL
 MS                GRADUATE SCHOOL

TOTAL NUMBER OF DDEGREES EARNED: 2
```

Figure 4-27. DEFINE Clause with RECORD COUNT

Figure 4-28 shows a Query using the DEFINE clause with RECORD TOTAL.

Query Statement:

```
DEFINE TOTAL-FOR-ITEM : CN/8.2 = QUAN * UPRC;
       INVOICE-TOTAL : CN/9.2 = RECORD TOTAL TOTAL-FOR-ITEM;
LIST 'SALES ORDER NUMBER: ' SONM
     HEADER '*****************************************************';
     TAB 5 ITEM TAB 17 QUAN TAB 28 UPRC TAB 38 TOTAL-FOR-ITEM
     HEADER SKIP 'ITEM NUMBER   QUANTITY   UNIT PRICE   TOTAL PRICE'
                 '-----------   --------   ----------   -----------';
     TAB 81 'TOTAL AMOUNT OF INVOICE: ' INVOICE-TOTAL
     FOOTING '*****************************************************';
FROM SOFL ITEM
LINKED BY ITEM = ITMN
BY KEY BY LIST
```

Query Output:

```
*****************************************************
SALES ORDER NUMBER: 300

ITEM NUMBER   QUANTITY   UNIT PRICE   TOTAL PRICE
-----------   --------   ----------   -----------
    777            5          500         2.50

TOTAL AMOUNT OF INVOICE:         2.50
*****************************************************
*****************************************************
SALES ORDER NUMBER: 100

ITEM NUMBER   QUANTITY   UNIT PRICE   TOTAL PRICE
-----------   --------   ----------   -----------
    333            1       100.000      100.00
    555            2        15.000       30.00

TOTAL AMOUNT OF INVOICE:       130.00
*****************************************************
*****************************************************
SALES ORDER NUMBER:  75

ITEM NUMBER   QUANTITY   UNIT PRICE   TOTAL PRICE
-----------   --------   ----------   -----------
    333            1       100.000      100.00
    555            2        15.000       30.00
    111            3         1.500        4.50
    777            4          .500        2.00

TOTAL AMOUNT OF INVOICE:       136.50
*****************************************************
*****************************************************
SALES ORDER NUMBER:  50

ITEM NUMBER   QUANTITY   UNIT PRICE   TOTAL PRICE
-----------   --------   ----------   -----------
    555          101        15.000     1515.00
    777            5          .500        2.50

TOTAL AMOUNT OF INVOICE:      1517.50
*****************************************************
```

Figure 4-28. DEFINE Clause with RECORD TOTAL

If a report line is composed only of literals and DEFINE variables that result from TOTAL or COUNT and/or calculations performed with TOTAL, COUNT, and constants, that report line is printed only once at the end of the report. If the fields of a report line include any RECORD TOTAL or RECORD COUNT operations but no fields or results of calculations performed directly on a field, the report line is printed at the end of processing for each record.

If a field or a calculation performed on a field is included in a report line, that report line is printed every time a qualified occurrence of the field would normally be printed; this holds true whether or not the report line includes TOTAL, COUNT, RECORD TOTAL, and RECORD COUNT fields. A value for the current result of expressions on TOTAL, COUNT, RECORD TOTAL, and RECORD COUNT is printed for the current report line. BREAK TOTAL or BREAK COUNT keeps calculating the total on count total until the break field(s) for a report line changes. The value is then printed and the total or count is cleared. In this manner, running totals, counts, subtotals, and subcounts can be printed. Running counts can be used to produce line numbers within a record or for the entire report. Also, you can determine an average for a field or DEFINE variable by specifying the calculation TOTAL fieldx/COUNT fieldx.

4.10 BREAK CLAUSE

The BREAK clause allows control break processing on totals, counts, and duplicate values. This type of processing uses data that is ordered on certain fields and needs to detect when the values of these fields change. Changing heading lines, starting a new page, and printing or clearing new totals or counts are examples of such field changes. BREAK is allowed only with the LIST function. BREAK can be used with multiple files.

There are two places in a Query statement that BREAK is specified to produce reports using control breaks. A BREAK clause is included in break-controlled report lines to designate the fields whose values will be tested to detect the break condition. The syntax of the BREAK clause is as follows:

BREAK-clause ::= BREAK { ON / BEFORE } field-type [field-type]. . .

Control break reports typically should include a record-level SORT clause that sorts the report on all BREAK fields. If this is not the case, the fields should already be in the required sorted order.

Control break reports include three types of lines: heading lines, detail lines, and summary lines. Heading lines contain data, headings, paging, and spacing that should be printed only when a particular field changes data. Use the BREAK ON clause to indicate these fields. This will cause them to be printed out only when the BREAK field(s) value changes. Detail lines have no associated BREAK clause and are printed as normal report lines. Summary lines include BREAK TOTAL and BREAK COUNT define variables and should be indicated by specifying a BREAK clause BREAK BEFORE. This will cause report lines to be printed with the value that was current immediately before the BREAK condition is detected and then cleared.

Figure 4-29 shows a Query using the BREAK clause with BEFORE, ON, and TOTAL.

Query Statement:

```
DEFINE TOTAL-STATE-SALARY: CN/7.2 = BREAK TOTAL MRAT;
LIST TAB 34 '-------'
     TAB 81 'TOTAL SALARY EXPENSE FOR STATE:  ' TOTAL-STATE-SALARY
     BREAK BEFORE MSTT;
     'STATE = ' MSTT HEADER SKIP
     FOOTING 'EMPLOYEE NAME                    SALARY'
             '-------------                    ------'
     BREAK ON MSTT;
     MNAM TAB 33 MRAT;
FROM PAY1
SORTED BY MSTT
BY KEY BY LIST
```

Query Output:

```
STATE = MO
EMPLOYEE NAME                    SALARY
-------------                    ------
PASCHAL, JIMMY                  2500.00
ABLE, CHARLIE                   1950.00
LI, KIM                          230.00
                                -------
TOTAL SALARY EXPENSE FOR STATE:  4680.00

STATE = NY
EMPLOYEE NAME                    SALARY
-------------                    ------
BROWN, WILLIE                    215.00
PARKS, FRED                      558.00
                                -------
TOTAL SALARY EXPENSE FOR STATE:   773.00

STATE = TX
EMPLOYEE NAME                    SALARY
-------------                    ------
STEPHENS, JANET                  385.00
HAYNES, BILL                     750.00
MEREDITH, JOHN                   900.00
HOWELL, JOHN                     375.00
                                -------
TOTAL SALARY EXPENSE FOR STATE:  2410.00
```

Figure 4-29. BREAK Clause Example Using BEFORE, ON, and TOTAL

The BREAK clause is used with TOTAL and COUNT operators in the DEFINE clause.

Specifying BREAK TOTAL or BREAK COUNT in a DEFINE variable definition causes that DEFINE variable to be printed and cleared when a break condition is detected in the report line in which it is included.

Figure 4-30 shows a Query using the BREAK clause with TOTAL, COUNT, BEFORE, and ON.

Query Statement:

```
DEFINE TOTAL-FOR-ITEM : CN/9.2 = QUAN * UPRC;
       INVOICE-TOTAL : CN/9.2 = RECORD TOTAL TOTAL-FOR-ITEM;
       TOTAL-DUE : CN/9.2 = BREAK TOTAL TOTAL-FOR-ITEM;
LIST TAB 14 'TOTAL DUE --->' TOTAL-DUE BREAK BEFORE BILL
     HEADER '                                    -------';
     'CUSTOMER:   ' NAME BREAK ON BILL
       HEADER SKIP 2 '****************************************'
       FOOTING SKIP;
     '     INVOICE NUMBER - ' SONM;
     TAB 28 INVOICE-TOTAL ' (INVOICE TOTAL)';
FROM SOFL ITEM CUST
LINKED BY ITEM = ITMN BILL = CUSN
UNIQUE BL
SORTED BY BILL
```

Query Output:

```
****************************************
CUSTOMER:  ED JONES

     INVOICE NUMBER - 100
                              130.00 (INVOICE TOTAL)
     INVOICE NUMBER -  50
                             1517.50 (INVOICE TOTAL)
                             -------
             TOTAL DUE --->  1647.50


****************************************
CUSTOMER:  PAT SMITH

     INVOICE NUMBER -  75
                              136.50 (INVOICE TOTAL)
                             -------
             TOTAL DUE --->   136.50


****************************************
CUSTOMER:  HARRY ABLE

     INVOICE NUMBER - 300
                                2.50 (INVOICE TOTAL)
                             -------
             TOTAL DUE --->     2.50
```

Figure 4-30. BREAK Clause Example

4.11 UNIQUE CLAUSE

The UNIQUE clause tells the Query processor that there is only one occurrence of the specified line type per record. This allows that line type to act like an 01 line. Fields in a UNIQUE line type can act as primary keys when used in report lines and secondary keys within that line can be optimized. The syntax for the UNIQUE clause is as follows:

UNIQUE-clause :: = UNIQUE line-type [line-type]. . .

Figure 4-31 shows a Query using LIST showing the results without the UNIQUE clause.

Query Statement:

```
       ITEMS PURCHASED THIS MONTH

CUSTOMER NUMBER    ITEM NUMBER    QUANTITY
---------------    -----------    --------
       5               777            5
       10              333            1
       3               333            1
       10              555          101
```

Query Output:

```
LIST TAB 8 BILL TAB 23 ITEM TAB 35 QUAN
FROM SOFL
BY KEY BY LIST
NO HEADER HEADER '       ITEMS PURCHASED THIS MONTH' SKIP
                 'CUSTOMER NUMBER    ITEM NUMBER    QUANTITY'
                 '---------------    -----------    --------'
```

Figure 4-31. Record-Level Without UNIQUE Example

Figure 4-32 shows the same Query using the UNIQUE clause. Notice that the restriction is removed that requires an occurrence of each line type in a report line before that report line can be output. The value of the field BILL (line type BL) is repeated in a new output line with each additional occurrence of the fields ITEM and QUAN (line type 03) within a record.

Query Statement:

```
LIST TAB 8 BILL TAB 23 ITEM TAB 35 QUAN
FROM SOFL
UNIQUE BL
BY KEY BY LIST
NO HEADER HEADER '       ITEMS PURCHASED THIS MONTH' SKIP
                 'CUSTOMER NUMBER   ITEM NUMBER   QUANTITY'
                 '---------------   -----------   --------'
```

Query Output:

```
       ITEMS PURCHASED THIS MONTH

CUSTOMER NUMBER   ITEM NUMBER   QUANTITY
---------------   -----------   --------
       5              777           5
       10             333           1
       10             555           2
       3              333           1
       3              555           2
       3              111           3
       3              777           4
       10             555         101
       10             777           5
```

Figure 4-32. UNIQUE Clause

Figure 4-33 shows Query using the UNIQUE clause and a record-level condition that tests a secondary key. Query-990 will optimize access to the file by using BILL, a secondary key.

Query Statement:

```
LIST TAB 8 BILL TAB 23 ITEM TAB 35 QUAN
FROM SOFL
UNIQUE BL
WHERE ANY BILL = 5
BY KEY BY LIST
NO HEADER HEADER '       ITEMS PURCHASED THIS MONTH' SKIP
                 'CUSTOMER NUMBER   ITEM NUMBER   QUANTITY'
                 '---------------   -----------   --------'
```

Query Output:

```
       ITEMS PURCHASED THIS MONTH

CUSTOMER NUMBER   ITEM NUMBER   QUANTITY
---------------   -----------   --------
       5              777           5
```

Figure 4-33. UNIQUE Example

The UNIQUE clause should follow the FROM clause.

4.12 LINKED BY CLAUSE

The LINKED BY clause is used when the Query processor must access fields from two or more files. When two fields from different files are linked, a logical hierarchy is formed with the first file specified in the FROM clause being the top-level file. The field linked to in the lower-level file must be a primary or secondary key. Multifile queries cannot be sequenced BY LIST.

4.12.1 Syntax

The LINKED BY clause has the following syntax:

LINKED-BY-clause ::= LINKED BY { field-type / concat-field } = key-type
[, { field-type / concat-field } = key-type] ...

Each specification of field ID = key ID specifies an access path between the file that contains the field ID and the file that contains the key ID. These must be two different files. When more than one field in the file is required for the key value in the linked-to file, the caret (^) is used to specify concatenation of the linking fields. For example, if a primary key EKEY from a file PAYR is composed of the fields SSN and NAME in the EMP file, the linkage is specified by "LINKED BY SSN^NAME = EKEY".

Figure 4-34 shows a Query using the LINKED BY clause.

Query Statement:

```
LIST SONM ITEM ITMN DESC
FROM SOFL ITEM LINKED BY ITEM = ITMN
NO HEADER
SORTED BY SONM BY KEY BY LIST
```

Query Output:

```
 50     555   555   GREEN JEANS
 50     777   777   RED HERRING
 75     333   333   GOLDEN RING
 75     555   555   GREEN JEANS
 75     111   111   PURPLE WIDGET
 75     777   777   RED HERRING
100     333   333   GOLDEN RING
100     555   555   GREEN JEANS
300     777   777   RED HERRING
```

Figure 4-34. LINKED BY Example

Figure 4-35 shows a Query using the LINKED BY clause.

Query Statement:

```
LIST 'SALES ORDER NUMBER: ' SONM TAB 81
     HEADER '*************************************************';
     ITEM TAB 15 QUAN TAB 30 DESC
     HEADER 'ITEM           QUANTITY       DESCRIPTION'
            '----           --------       -----------'
     FOOTING '*************************************************';
FROM SOFL ITEM LINKED BY ITEM = ITMN
WHERE ANY ITEM = 111 BY KEY BY LIST
```

Query Output:

```
*************************************************
SALES ORDER NUMBER:  75
ITEM           QUANTITY       DESCRIPTION
----           --------       -----------
333               1           GOLDEN RING
555               2           GREEN JEANS
111               3           PURPLE WIDGET
777               4           RED HERRING
*************************************************
```

Figure 4-35. LINKED BY Clause

4.12.2 LINKED BY File Hierarchy

Multifile relationships have the following characteristics:

- A single top-level file is required. This is the first file specified in the FROM clause.
- The LINKED BY clause must define an access path from the top-level file to all other files. However, the access path need not follow directly from the top-level file to the lower-level files; the path can be indirect (through another lower-level file).
- Access paths must link a higher-level file to a lower-level file. This is achieved by linking a field or key in the higher-level file to a primary or secondary key in the lower-level file. An example of a Query statement using a secondary key (ITEM) in a lower-level file is as follows:

```
LIST SONM ITEM DESC
FROM ITEM SOFL LINKED BY ITMN = ITEM
```

Access through the primary key of the lower-level file associates a lower-level file record with each occurrence of the top-level file. An example of a Query statement using the primary key (ITMN) of the lower-level file is as follows:

```
LIST SONM ITEM DESC
FROM SOFL ITEM LINKED BY ITEM = ITMN
```

Access through a secondary key in the lower-level file associates all lines having that secondary key value with each occurrence of the top-level file.

When more than one access path has been defined for a lower-level file, a THRU clause must be used in each report line that uses fields from the lower-level file to indicate which access path to use. The syntax for the THRU clause is as follows:

THRU-clause ::= THRU file-type

Figure 4-36 shows a Query using the LINKED BY with three fields and the THRU clause.

Query Statement:

```
LIST 'SALES ORDER NUMBER: ' SONM
     FOOTING ' ';
     'SHIP TO: ' NAME THRU SHIP;
     TAB 10 STRT THRU SHIP;
     TAB 10 CITY:8 ',' STAT ZIPC THRU SHIP;
     TAB 4 ITEM TAB 13 QUAN TAB 21 DESC:12 UPRC
     HEADER SKIP
     '-----------------------------------------'
     ' ITEM NO.   QUAN.  DESCRIPTION   UNIT COST'
     FOOTING
     '-----------------------------------------' SKIP 5;
FROM SOFL CUST ITEM LINKED BY BILL = CUSN SHIP = CUSN ITEM = ITMN
NO HEADER
```

Query Output:

```
SALES ORDER NUMBER: 300

SHIP TO: HARRY ABLE
         123 MAIN
         AUSTIN  ,TX  78701

-----------------------------------------
 ITEM NO.   QUAN.  DESCRIPTION   UNIT COST
   777         5    RED HERRING      .500
-----------------------------------------
```

Figure 4-36. LINKED BY Example Using the THRU Clause (Sheet 1 of 2)

```
SALES ORDER NUMBER: 100

SHIP TO: ED JONES
         4242 12TH
         CHICAGO ,IL  33333

-------------------------------------------
 ITEM NO.    QUAN.  DESCRIPTION   UNIT COST
   333          1    GOLDEN RING   100.000
   555          2    GREEN JEANS    15.000
-------------------------------------------



SALES ORDER NUMBER:  75

SHIP TO: PAT SMITH
         999 WEST
         MIAMI   ,FL  12345

-------------------------------------------
 ITEM NO.    QUAN.  DESCRIPTION   UNIT COST
   333          1    GOLDEN RING   100.000
   555          2    GREEN JEANS    15.000
   111          3    PURPLE WIDGE    1.500
   777          4    RED HERRING      .500
-------------------------------------------



SALES ORDER NUMBER:  50

SHIP TO: ED JONES
         4242 12TH
         CHICAGO ,IL  33333

-------------------------------------------
 ITEM NO.    QUAN.  DESCRIPTION   UNIT COST
   555        101    GREEN JEANS    15.000
   777          5    RED HERRING      .500
-------------------------------------------
```

Figure 4-36. LINKED BY Example Using the THRU Clause (Sheet 2 of 2)

4.12.3 THRU Clause

The THRU clause is used in a report line to specify an access path between two different link fields to one file. The THRU clause indicates which link field accesses the key. Only one THRU clause should be used per report line. If no THRU clause is used, the access path defaults to the first linkage defined.

4.12.4 IN Clause

When more than one file specified in the LINKED BY clause has fields with a common field name, the IN clause designates which field name is used with the file ID.

4.13 CHANGE DATA CONSTANTS

Change data constants allow you to supply constant values while the Query statement is executing for use in CONTENTS clauses, WHERE conditions, DEFINE clauses, and report-line literals. You can use change data constants in both stand-alone Query and in the application program interface. You can also use change data constants to chain Query executions together by using the output of one Query statement to drive another Query statement as change data input.

4.13.1 Change Data Constants and Stand-Alone Query

When used in stand-alone Query, the constant values must be stored in a file. Answer NO to the DEFAULT REPORT PARAMETERS prompt in the Query command; then, enter the change data file pathname in response to the CHANGE DATA PATHNAME prompt of the REPORT PARAMETERS screen.

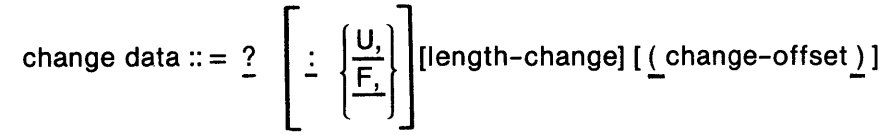
The Query statement reads a single record from the change data file and picks up all change data constant values from that record. The constant values are incorporated into the executing Query statement. Then, the statement completes execution, writing all report lines or performing all modifications indicated. The processor is then reinitialized except for totals, counts, and the main heading. Another record is then read from the change data file, the new constants are incorporated into the statement, and the execution is repeated. The process continues until the entire change data file has been read. Then, the totals, counts, and main footings are written.

4.13.2 Change Data Constants and Application Programs

When using the application program interface, initialize the Query statement by calling either the QINIT or QCOMP subroutine. Use the QSEND subroutine to pass a buffer containing the change data constants for the entire Query statement. You can then execute the Query by using either subroutine QEXEC or QRECV. When you use the application program interface, totals, counts, averages, main headings, and footings are returned or listed every time you call QSEND.

4.13.3 Change Data Constant Format

The format for a change data constant is as follows:



The U and F specify whether the value is unformatted or formatted. If you specify U (unformatted), no conversion on the change data value occurs before it is incorporated into the Query statement. If you specify F (formatted), numeric literals are converted to the proper format for the context in which they are used. If you specify neither U nor F, the parameter defaults to F.

Length is an integer specifying the field length of the change data constant in the records of either the change data file or the QSEND buffer. Offset is an integer specifying the character position in the record where the change data constant field begins. You must specify length for constants used as report line literals. In a CONTENTS clause, length defaults to the length of the field to which the change data is being assigned. In a WHERE clause, length defaults to the length of the field to which the change data is being compared. In a DEFINE clause, length defaults to the length of the DEFINE variable. If the constant is unformatted, the length specified in either the DDL declaration for the file or the DEFINE variable definition is used. If no offset is specified, it defaults to the next character position after the end of the previous change data field.

The following condition reads 15 characters, starting in the 30th character position in the change data file record, and converts them to the proper format for the field MNAM:

"WHERE MNAM = ?:F,15(30)"

The following examples illustrate how change data constants make it easier to write Query statements when large numbers of constant values are needed.

Figure 4-37 shows how the test file CUST is loaded with data using the INSERT function without using change data constants:

Query Statement:

```
 DXQUERY       1.3.0  81.275    QUERY-990     02/16/82  08:19:01 PAGE     1
INSERT 01 CONTENTS CUSN= '3', NAME='PAT SMITH', STRT='999 WEST',
          CITY='MIAMI', STAT='FL', ZIPC='12345', CRED='EXCELLENT';
       01 CONTENTS CUSN= '5', NAME='HARRY ABLE', STRT='123 MAIN',
          CITY='AUSTIN', STAT='TX', ZIPC='78701', CRED='GOOD';
       01 CONTENTS CUSN= '10', NAME='ED JONES', STRT='4242 12TH',
          CITY='CHICAGO', STAT='IL', ZIPC='33333', CRED='POOR';
       01 CONTENTS CUSN= '15', NAME='MARY BROWN', STRT='23 PECAN',
          CITY='OAKLAND', STAT='CA', ZIPC='29157', CRED='GOOD';
       01 CONTENTS CUSN= '22', NAME='BOB CARTER', STRT='187 MONEY',
          CITY='LAS VEGAS', STAT='NE', ZIPC='93487', CRED='EXCELLENT';
FROM CUST
```

Figure 4-37. Query with INSERT Without Change Data Constants (Sheet 1 of 2)

Query Output:

```
 DXQUERY       1.3.0  81.275   QUERY-990     02/16/82  08:19:01 PAGE      2
LINE  LOC1       LOC2       CUSN   NAME                    STRT
CITY                    STAT  ZIPC   CRED
 01  00000000    ****       3      PAT SMITH               999 WEST
MIAMI                   FL    12345  EXCELLENT

LINE  LOC1       LOC2       CUSN   NAME                    STRT
CITY                    STAT  ZIPC   CRED
 01  00000066    ****       5      HARRY ABLE              123 MAIN
AUSTIN                  TX    78701  GOOD

LINE  LOC1       LOC2       CUSN   NAME                    STRT
CITY                    STAT  ZIPC   CRED
 01  000000CC    ****       10     ED JONES                4242 12TH
CHICAGO                 IL    33333  POOR

LINE  LOC1       LOC2       CUSN   NAME                    STRT
CITY                    STAT  ZIPC   CRED
 01  00000132    ****       15     MARY BROWN              23 PECAN
OAKLAND                 CA    29157  GOOD

LINE  LOC1       LOC2       CUSN   NAME                    STRT
CITY                    STAT  ZIPC   CRED
 01  00000198    ****       22     BOB CARTER              187 MONEY
LAS VEGAS               NE    93487  EXCELLENT
```

Figure 4-37. Query with INSERT Without Change Data Constants (Sheet 2 of 2)

Figure 4-38 shows the change data constants loading file for the test file CUST.

```
3   PAT SMITH    999 WEST    MAIMI       FL  12345  EXCELLENT
5   HARRY ABLE   123 MAIN    AUSTIN      TX  78701  GOOD
10  ED JONES     4242 12TH   CHICAGO     IL  33333  POOR
15  MARY BROWN   23 PECAN    OAKLAND     CA  29157  GOOD
22  BOB CARTER   187 MONEY   LAS VEGAS   NE  93487  EXCELLENT
```

Figure 4-38. Change Data File Contents for INSERT Example

Figure 4-39 shows a Query using both the INSERT function and the change data constants. This Query has the same results as the Query in Figure 4-37 and could be used to insert more than 5 lines with a single execution.

Query Statement:

```
 DXQUERY       1.3.0  81.275   QUERY-990     02/16/82  08:33:57 PAGE     1
INSERT 01 CONTENTS CUSN = ?:F,2(1), NAME = ?:F,12(5), STRT = ?:F,9(17),
                   CITY = ?:F,9(28), STAT = ?:F,2(39), ZIPC = ?:F,5(43),
                   CRED = ?:F,9(50);
FROM CUST
```

Query Output:

```
 DXQUERY       1.3.0  81.275   QUERY-990     02/16/82  08:33:57 PAGE      2
LINE  LOC1       LOC2       CUSN    NAME                    STRT
CITY                    STAT  ZIPC    CRED
 01  00000000     ****       3       PAT SMITH              999 WEST
MAIMI                   FL     12345  EXCELLENT

LINE  LOC1       LOC2       CUSN    NAME                    STRT
CITY                    STAT  ZIPC    CRED
 01  00000066     ****       5       HARRY ABLE             123 MAIN
AUSTIN                  TX     78701  GOOD

LINE  LOC1       LOC2       CUSN    NAME                    STRT
CITY                    STAT  ZIPC    CRED
 01  000000CC     ****       10      ED JONES               4242 12TH
CHICAGO                 IL     33333  POOR

LINE  LOC1       LOC2       CUSN    NAME                    STRT
CITY                    STAT  ZIPC    CRED
 01  00000132     ****       15      MARY BROWN             23 PECAN
OAKLAND                 CA     29157  GOOD

LINE  LOC1       LOC2       CUSN    NAME                    STRT
CITY                    STAT  ZIPC    CRED
 01  00000198     ****       22      BOB CARTER             187 MONEY
LAS VEGAS               NE     93487  EXCELLENT
```

Figure 4-39. Change Data Constants and INSERT Function

5

Optimization

5.1 INTRODUCTION

This section discusses optimization of Query-990 using key and clause combinations.

5.2 OPTIMIZATION

Optimization of Query-990 consists of using conditions with key combinations in such a way that the Query processor need not read all the keys in the file. The Query processor reads only the keys that meet the condition specified in the WHERE clause, creating a condition of optimum data access.

5.2.1 Record-Level Conditions

Record-level conditions have the following combinations:

RANDOM PRIMARY KEY IN DATA BASE FILES

Use BY KEY BY LIST or BY KEY for optimization. The record-level condition must test only the primary key for equality to a specific value. Therefore, the only relational operator allowed is EQ or =. You can use an OR connector as long as the remaining record conditions test only the primary key for equality to a specific value.

```
LIST MNAM MRAT FROM PAY1 BY KEY
WHERE MNUM = 55555
```

SEQUENTIAL PRIMARY KEY IN DATA BASE OR KIFs

Use BY KEY BY LIST or BY KEY for optimization. The record-level condition can test the primary key using all relational operators except NE or <>. You can use an OR or an AND connection as long as the remaining record conditions test only the primary key against a specific value.

```
LIST MNAM MRAT FROM PAY1 BY KEY BY LIST
WHERE MNUM GT 40000 AND MNUM LT 60000
```

RANDOM SECONDARY KEY IN DATA BASE FILES

Use BY KEY BY LIST or BY KEY with UNIQUE for optimization. The record-level condition can test for equality to a specific value when the line type that contains the secondary key has zero or one occurrence per record. The line type must be declared UNIQUE. The only relational operator allowed is EQ or =. You can use an OR connector if the remaining record conditions test only the secondary key for equality to a specific value.

```
LIST MJOB DEGR FROM PAY1
UNIQUE CU BY KEY
WHERE ANY MSSN = 487265478 OR ANY MSSN = 852417931
```

SEQUENTIAL SECONDARY KEY IN DATA BASE OR KIFs

Use BY KEY BY LIST or BY KEY with UNIQUE for optimization. The record-level condition can test for equality to a specific value when the line type that contains the secondary key has zero or one occurrence per record. The line type must be declared UNIQUE. All relational operators are allowed except NE or < >. You can use the connectors AND and OR if the remaining record conditions test the secondary key against a specific value.

```
LIST MJOB DEGR FROM PAY1
UNIQUE CU BY KEY BY LIST
WHERE ANY MSSN NE 487265478 AND ANY MSSN NE 852417931
```

5.2.2 Line-Level Conditions

Line-level conditions have the following combinations:

RANDOM PRIMARY KEY IN DATA BASE FILES

Use BY KEY BY LIST, BY KEY, or BY LIST for optimization. The line-level condition can test the primary key for a specific value as long as there is only one report line. The relational operator EQ or = is allowed. You can use the connectors AND and OR if the remaining line-level conditions test only the primary key for equality to a specific value.

```
LIST MNAM MJOB WHERE MJOB = 'PROGRAMMER'
FROM PAY1 BY KEY BY LIST
```

SEQUENTIAL PRIMARY KEY IN DATA BASE FILES

Use BY KEY BY LIST, BY KEY, or BY LIST for optimization. The line-level condition can test the primary key for a specific value as long as there is only one report line. All relational operators except NE or < > are allowed. You can use the connectors AND and OR if the remaining line-level conditions test only the primary key against a specific value.

```
LIST MNAM MRAT WHERE MRAT GT 1000 AND MRAT LT 2000
FROM PAY1 BY KEY
```

RANDOM SECONDARY KEY IN DATA BASE FILES

Use BY LIST for optimization. The line-level condition can test the secondary key for equality to a specific value. The relational operator EQ or = is allowed. You can use the OR connector if the remaining line-level conditions test only the secondary key for equality to a specific value.

```
LIST MNAM ADDR WHERE MSSN = 487265478
FROM PAY1 BY LIST
```

SEQUENTIAL SECONDARY KEY IN DATA BASE KIFs

Use BY LIST for optimization. The line-level condition can test the secondary key against a specific value. All relational operators are allowed except NE or < >. You can use the AND and OR connectors if the remaining line-level conditions test only the secondary key against a specific value.

```
LIST MNAM ADDR WHERE MSSN GT 400000000 AND MSSN NE 487265478
FROM PAY1 BY LIST
```

6

Program Language Interface Subroutines

6.1 INTRODUCTION

You can access Query from Pascal, COBOL, and FORTRAN programs through a set of assembly language subroutines. These subroutines interface between the Query processor and the application task. Data is transferred between the calling program and Query via interprocess communication. You can link the following subroutines to the calling task:

- QCOMP — Compiles, loads, and prepares a Query statement for execution. The Query statement is passed from the application task as an array of characters.
- QINIT — Loads and prepares for execution a Query statement that has already been compiled (using QCOMP) and stored as an object file.
- QEXEC — Executes a Query statement started by QCOMP or QINIT and lists the results to an output file.
- QRECV — Processes one cycle of a Query statement. For example, if the Query is a LIST function, QRECV returns one logical report line.
- QSEND — Resets and sends change data values, using the contents of the data buffer.
- QCLR — Reinitializes the Query processor for a particular Query statement (a clearing function).
- QEND — Terminates the Query processor for a particular Query statement.

6.2 CALLING FORMATS

The calling formats for COBOL, Pascal, and FORTRAN are similar. Example calls to the QCOMP subroutine from each language are as follows:

COBOL

```
CALL "QCOMP" USING QUERY-NUMBER, RETURN-STATUS, RETURN-CODE,
                   QUERY-STATEMENT, STATEMENT-LENGTH,
                   PASSWORD, FORMAT, LIST-TEXT,
                   LISTING-PATHNAME, PAGELENGTH, PAGEWIDTH,
                   ALT-FILE.
```

Pascal

```
QCOMP(QUERY_NUMBER, RETURN_STATUS, RETURN_CODE, EXECUTE_FLAG,
      QUERY_STATEMENT, STATEMENT_LENGTH, PASSWORD, FORMAT,
      LIST_TEXT, LISTING_PATHNAME, PAGE_LENGTH, PAGE_WIDTH,
      ALT_FILE);
```

FORTRAN

```
CALL QCOMP(QRYNUM,STATUS,CODE,QSTATE,STLEN,PASSW,
           FORMAT,LIST,LPATH,PGLEN,PGWDTH,ALTFIL)
```

6.3 QCOMP — COMPILE AND INITIALIZE

The format of QCOMP syntax is as follows:

```
CALL QCOMP(QUERY-NUMBER, RETURN-STATUS, RETURN-CODE,
           QUERY-STATEMENT,STATEMENT-LENGTH,PASSWORD,
           FORMAT,LIST-TEXT,LIST-PATHNAME,PAGELENGTH-0,
           PAGEWIDTH-0,ALT-FILE)
```

QCOMP compiles a Query statement that you supply in the application program as an array of characters. The compiler builds a Query object file, bids a Query executor, and loads the Query object. The parameters are as follows:

QUERY-NUMBER
: The data type is integer or COMP-1. Assign an integer to be associated with the Query processor that this call bids. Subsequent Query subroutine calls will use this number to identify the Query processor that is to receive the command. The main purpose of this number is to allow multiple Query executions (up to five) to operate simultaneously. If the Query statement contains change data constants, the Query processor waits for a QSEND call with a buffer containing these constants. After QSEND has been called or if there are no change data constants, call QEXEC or QRECV to execute the statement.

RETURN-STATUS
: The data type is integer or COMP-1. The status is returned to the application program upon completion of the call. The status is zero if no error occurs; if an error does occur, the status is a nonzero integer indicating the particular error condition. (See Table 6-1 for status codes.)

RETURN-CODE
: The data type is a two-character string. If an I/O, DBMS, or DD error terminates execution, RETURN_CODE is the two-character system I/O, DBMS, or DD status code.

QUERY-STATEMENT
: The data type is a string of no more than 480 characters. This is the Query text to be compiled and executed.

STATEMENT-LENGTH
: The data type which is integer or COMP-1; specifies the length in bytes of the Query text.

PASSWORD
The data type is a four-character string. If security is used, this must be a valid password.

FORMAT
The data type is Boolean, integer, or COMP-1. FORMAT is set to 1 to indicate Query output in report format and set to 0 to specify unformatted binary.

LIST-TEXT
The data type is BOOLEAN, integer, or COMP-1. LIST-TEXT is set to 1 to include the Query statement listing in the listing file (specified by LIST-PATHNAME) and set to 0 for no listing.

LIST-PATHNAME
The pathname contains the Query statement and any errors detected by the compiler.

PAGELENGTH
The page length for formatted output. If you enter 0, the default (60) is used.

PAGEWIDTH
The page width for formatted output. If you enter 0, the default (80) is used.

ALT-FILE
The pathname contains the alternate collating sequence file. Set to blanks if none is desired.

6.4 QINIT — INITIALIZE QUERY INTERPRETER

The format of QINIT syntax is as follows:

```
CALL QINIT(QUERY-NUMBER, RETURN-STATUS, RETURN-CODE,
           OBJECT-PATHNAME, PASSWORD)
```

Use this procedure to bid a Query executor using Query object that has already been compiled by the stand-alone compiler before execution of the application program. The parameters are as follows:

QUERY-NUMBER
The data type is integer or COMP-1. Assign an integer to be associated with the Query processor that this call bids. Subsequent Query subroutine calls use this number to identify the Query processor that is to receive the command. The main purpose of this number is to allow multiple Query executions (up to five) to operate simultaneously. If the Query statement contains change data constants, the Query processor waits for a QSEND call with a buffer containing these constants. After the QSEND or if there are no change data constants, call the QEXEC or QRECV subroutine to execute the statement.

RETURN-STATUS
The data type is integer or COMP-1. The status is returned to the application program upon completion of the call. If no error occurs, the status is zero; if an error does occur, the status is a nonzero integer indicating the particular error condition. (See Table 6-1 for status codes.)

RETURN-CODE
The data type is a two-character string. If an I/O, DBMS, or DD error that terminates execution occurs, RETURN-CODE is the two-character system I/O, DBMS, or DD status code.

OBJECT-PATHNAME
The data type is a 48-character string that indicates the file pathname where the Query object can be found. Synonym substitution is performed. The pathname must end with a blank if it is less than 48 characters long.

PASSWORD
The data type is a four-character string. If security is used, this must be a valid password.

6.5 QEXEC — EXECUTE AND LIST QUERY RESULTS

The format of QEXEC syntax is as follows:

```
CALL QEXEC(QUERY-NUMBER, RETURN-STATUS, RETURN-CODE,
           OUTPUT-PATHNAME, EXTEND)
```

QEXEC completes execution and lists the results of a Query statement started by QINIT or QCOMP. The listing is sent to the file or device indicated by the output pathname parameter. If the last Query executed was a LIST function, the output is a report or data. If the last Query was a modification function, the output is the trace. The parameters are as follows:

QUERY-NUMBER
The data type is integer or COMP-1. This is the number associated with the desired Query processor, assigned by QCOMP or QINIT.

RETURN-STATUS
The data type is integer or COMP-1. The status is returned to the application program upon completion of the call. If no error occurs, the status is zero; if an error does occur, the status is a nonzero integer indicating the particular error condition. (See Table 6-1 for status codes.)

RETURN-CODE
The data type is a two-character string. If an I/O, DBMS, or DD error that terminates execution occurs, RETURN-CODE is the two-character system I/O, DBMS, or DD status code.

OUTPUT-PATHNAME
The data type is a 48-character string that indicates the file pathname or device name that is to receive the Query output. Synonym substitution is performed. The pathname must end with a blank if it is less than 48 characters long.

EXTEND
The data type is integer, Boolean, or COMP-1. To initialize the output file, set the extend parameter to 0. To open the file extended so that the results of the Query will be added to the end of the file, set the extend parameter to 1.

6.6 QRECV — RECEIVE QUERY DATA

The format of QRECV syntax is as follows:

```
CALL QRECV(QUERY-NUMBER, RETURN-STATUS, RETURN-CODE,
           DATA-BUFFER, BUFFER-LENGTH)
```

Use this procedure to retrieve the results of one execution cycle of the Query processor associated with the specified Query number. The results are returned in the defined data buffer. If the Query performs a LIST function, the data buffer receives one logical report line. If the Query is a modification function, the data buffer receives a trace line. The parameters are as follows:

QUERY-NUMBER
: The data type is integer or COMP-1 and indicates the number associated with the desired Query processor, assigned by QCOMP or QINIT.

RETURN-STATUS
: The data type is integer or COMP-1. This is returned to the application program upon completion of the call. The status is set to indicate normal operation, end of processing, or an error status. (See Table 6-1 for status codes.) If the end of processing is signaled, the Query processor does not terminate. Use QEND to terminate it.

RETURN-CODE
: The data type is a two-character string. If an I/O, DBMS, or DD error that terminates execution occurs, RETURN-CODE is the two-character system I/O, DBMS, or DD status code.

DATA-BUFFER
: The data type is a string with a maximum length of 480 characters. The results of the execution are returned in this buffer.

LENGTH
: The data type is integer or COMP-1, defining the length of the data buffer.

6.7 QSEND — SEND CHANGE DATA CONSTANTS

The format of QSEND syntax is as follows:

```
CALL QSEND(QUERY-NUMBER, RETURN-STATUS, RETURN-CODE,
           DATA-BUFFER, BUFFER-LENGTH)
```

This procedure resets change data constant values in the Query statement. These values are indicated in the text by a question mark (?). The contents of DATA-BUFFER are the change data values (Section 4). The parameters are as follows:

QUERY-NUMBER
: The data type is integer or COMP-1 and specifies the number associated with the desired Query processor, assigned by QCOMP or QINIT.

RETURN-STATUS
The data type is integer or COMP-1. The status is returned to the application program upon completion of the call. If no error occurs, the status is zero; if an error does occur, the status is a nonzero integer indicating the particular error condition. (See Table 6-1 for status codes.)

RETURN-CODE
The data type is a two-character string. If an I/O, DBMS, or DD error that terminates execution occurs, RETURN-CODE is the two-character system I/O, DBMS, or DD status code.

DATA-BUFFER
The data type is a string with a maximum length of 480 characters. This buffer contains the change data values.

LENGTH
The data type is integer or COMP-1. This defines the length of the data buffer.

6.8 QCLR — REINITIALIZE QUERY PROCESSOR

The format of QCLR syntax is as follows:

```
CALL QCLR(QUERY-NUMBER, RETURN-STATUS)
```

QCLR reinitializes the Query processor associated with the Query number. Specifically, QCLR resets location pointers to asterisks and clears conditions. The parameters are as follows:

QUERY-NUMBER
The data type is integer or COMP-1 and specifies the number associated with the desired Query processor, assigned by QCOMP or QINIT.

RETURN-STATUS
The data type is integer or COMP-1. The status is returned to the application program upon completion of the call. If no error occurs, the status is zero; if an error does occur, the status is a nonzero integer indicating the particular error condition. (See Table 6-1 for status codes.)

6.9 QEND — END QUERY PROCESSOR

The format of QEND syntax is as follows:

```
CALL QEND(QUERY-NUMBER, RETURN-STATUS)
```

QEND terminates the Query processor associated with the designated Query number. Files built by that Query processor in its last execution are lost if QEXEC or QRECV are not executed. The parameters are as follows:

QUERY-NUMBER
The data type is integer or COMP-1 specifying the number associated with the desired Query processor, assigned by QCOMP or QINIT.

RETURN-STATUS

The data type is integer or COMP-1. The status is returned to the application program upon completion of the call. If no error occurs, the status is zero; if an error does occur, the status is a nonzero integer indicating the particular error condition. (See Table 6-1 for status codes.)

Table 6-1. Interface Subroutine Status Codes

Code	Meaning
1	Number table is full. A maximum of five different queries can be bid at one time.
2	Query number does not exist. You have attempted to access a Query task that was not initialized (by a QCOMP or QINIT call), or the previous call returned a fatal error.
3	Data Base or Data Dictionary error (Query interpreter). Check the return code field for the error returned by the Query processor.
4	Unable to bid interpreter task. The Query interpreter task is not installed on the program file S$QUERY.PROG.
5	Unable to bid compiler task. The Query compiler task is not installed on the program file S$QUERY.PROG.
6	Query task no longer active. The Query task terminated either because it could not get an interprocess communication buffer or because of an abnormal task termination. Check the system log for an abnormal termination message for installed task ID <C3 or <81 through <85.
7	Cannot allocate buffer for message. The language interface was unable to get an interprocess communication buffer.
8	Bad parameter list. The wrong number of parameters was passed to the language interface routine.
9	Invalid command. The command sequence was incorrect; e.g., QRECV or QEXEC must follow a QSEND call.
10	Duplicate Query number. The same Query number was used for more than one QCOMP or QINIT call.
11	Invalid Query number. Zero is an invalid Query number.
12	Syntax errors (Query compiler). Check the QCOMP listing pathname for a description of the syntax errors.
13	Unable to access object pathname. Check the return code for the operating system error code.

Table 6-1. Interface Subroutine Status Codes (Continued)

Code	Meaning
14	Unable to access listing pathname. Check the return code for the operating system error code.
15	Unable to access alternate collating pathname. Check the return code for the operating system error code.
16	Bad object file (Query interpreter). The object file does not contain valid Query object. Check that the correct object pathname was used.
17	High-order truncation of numeric constant. A number larger than the field size was sent in the QSEND buffer.
18	Negative sign used in unsigned number. Invalid data was entered for an unsigned field.
19	Number conversion error. Invalid data was sent for a numeric field.

6.10 USING THE INTERFACE SUBROUTINES

The interface subroutines provide a flexible interface and can accommodate many possible applications by using different sequences and combinations of calls. Descriptions of three common designs and implementations are as follows.

Design:

The application generates a Query statement by prompting the user for the necessary information. The results are listed to a file or displayed on the screen.

Implementation:

1. Prompt the user for the required information (such as field names, file name, and so on).
2. Use this information to build a Query statement in a buffer in memory.
3. Call QCOMP and pass the statement buffer.
4. Prompt the user for the output destination. If the output goes to a file, call QEXEC with the file pathname. If it goes to the user's screen, call QRECV repeatedly and display each report line as it is processed.
5. Call QEND to terminate the Query processor.

Design:

The application displays descriptions of several reports and prompts the user to choose one. The user also specifies the records to be used in the report. The application writes the resulting report to a file. A single execution of the application program can generate more than one report.

Implementation:

1. Write the Query statement for each standard report, using change data constants for selection criteria values that will be obtained from the user. Compile the Query statements by using the QCOMP command, and store the object output in a file.

2. Prompt the user to choose a report, and call QINIT with the proper Query object file pathname.

3. Prompt the user for selection criteria (such as primary key values) and position the values in a buffer so that they match the corresponding change data constant positions in the precompiled Query statements.

4. Call QSEND with this buffer of change data values.

5. Call QEXEC to complete execution and write the output to the file.

6. If the user wants another report using different selection criteria, call QCLR and return to step 3; otherwise, call QEND.

Design:

The user has two (or more) interdependent files. One file contains new data used to update another file. The objective is to read the file of new data and use that information to change the other file.

Implementation:

1. Write two Query statements. One retrieves data from the new data file with a LIST statement. The second applies the changes by using change data constants in the CONTENTS clause of an UPDATE function. Compile the statements by using the QCOMP command, and store the object output in a file.

2. In the application program, call QINIT for both Query statements.

3. Call QRECV for the LIST Query. If the data in the buffer is not in the proper locations for the change data constants in the UPDATE Query statement, move the data to the proper positions.

4. Call QSEND for the UPDATE Query with the change data constant values just obtained from the LIST Query.

5. Call QEXEC for the UPDATE Query and execute the updates. Call QCLR for the UPDATE Query.

6. Loop to step 3 until no more data is received.

7. Call QEND for both Queries.

6.11 EXAMPLE PROGRAMS

The following examples demonstrate calls to the Query subroutines from Pascal, FORTRAN, and COBOL language programs. The Pascal, and FORTRAN programs are equivalent, using four of the seven Query subroutines. The COBOL program demonstrates all seven subroutines.

6.11.1 Example Pascal Program

```
PROGRAM PEXPL;

(* This program provides examples of the external definitions needed
   in a TI Pascal  program to access any of the Query subroutines.
   In addition, PEXPL illustrates calls to the
   following subroutines:

    QINIT (initializes a Query processor)
    QEXEC (executes the Query object and lists the results to a file)
    QRECV (performs the function of QEXEC, but one line at a time)
    QEND  (deactivates a Query processor)

*)
TYPE

 C80 = PACKED ARRAY[1..80] OF CHAR;
 C40 = PACKED ARRAY[1..40] OF CHAR;
 C16 = PACKED ARRAY[1..16] OF CHAR;
 C4  = PACKED ARRAY[1..4]  OF CHAR;
 C2  = PACKED ARRAY[1..2]  OF CHAR;

VAR

 QUERY_NUMBER,R_STATUS,EXTEND,LNG : INTEGER;
 PATHNM : C40;
 R_CODE : C2;
 PASSWORD : C4;
 DBUFF : C80;

(* The external definitions needed to call all of the interface
   routines are as follows:
*)

PROCEDURE QCOMP(VAR QUERY_NUMBER : INTEGER;
                VAR RETURN_STATUS : INTEGER;
                VAR RETURN_CODE : C2;
                VAR QUERY_STATEMENT : C2;
                VAR STATEMENT_LENGTH : INTEGER;
                VAR PASSWORD : C4;
                VAR FORMAT   : INTEGER;
                VAR LIST_TEXT : INTEGER;
                VAR LIST_PATHNAME C2;
                VAR PAGE_LENGTH; PAGE_WIDTH : INTEGER;
                VAR ALT_FILE : C2); EXTERNAL;
```

```
(* NOTE--QCOMP is not used in this example.*)

PROCEDURE QINIT(VAR QUERY_NUMBER : INTEGER;
                VAR RETURN_NUMBER : INTEGER;
                VAR RETURN_CODE : C2;
                VAR PATHNM : C2;
                VAR PASSWORD : C4); EXTERNAL;

PROCEDURE QEXEC(VAR QUERY_NUMBER : INTEGER;
                VAR RETURN_STATUS : INTEGER;
                VAR RETURN_CODE : C2;
                VAR OUTPUT_PATHNAME : C2;
                VAR EXTEND : INTEGER); EXTERNAL;

PROCEDURE QCLR(VAR QUERY_NUMBER : INTEGER;
               VAR RETURN_STATUS : INTEGER); EXTERNAL;

(* NOTE--QCLR is not used in this example.*)

PROCEDURE QEND(VAR QUERY_NUMBER : INTEGER;
               VAR RETURN_STATUS : INTEGER); EXTERNAL;

PROCEDURE QRECV(VAR QUERY_NUMBER : INTEGER;
                VAR RETURN_STATUS : INTEGER;
                VAR RETURN_CODE : C2;
                VAR DATA_BUFFER : C2;
                VAR BUFFER_LENGTH : INTEGER); EXTERNAL;

PROCEDURE QSEND(VAR QUERY_NUMBER : INTEGER;
                VAR RETURN_STATUS : INTEGER;
                VAR RETURN_CODE : C2;
                VAR DATA_BUFFER : C2;
                VAR BUFFER_LENGTH : INTEGER); EXTERNAL;

(*NOTE--QSEND is not used in this example.*)


BEGIN(*PEXPL*)

 REWRITE(OUTPUT);
 PASSWORD := 'DBMS';

(*
  Assume that prior to the execution of this program the desired Query
  has been compiled using QCOMP, and further assume that the resulting
  object resides on 'X.TEST.QUERYOBJ'.

  Assign the object pathname, choose a number to associate with a Query
  processor, and initialize that processor.
*)

 PATHNM::C16 := 'X.TEST.QUERYOBJ ';(* NOTE-Trailing blank is mandatory*)
 QUERY_NUMBER := 2;
 QINIT(QUERY_NUMBER,R_STATUS,R_CODE,PATHNM::C2,PASSWORD);
 WRITELN('RETURN STATUS FROM QINIT = ',R_STATUS);

(* Check for normal completion of the initialization, specify a
  pathname for the result of the Query execution, execute the Query,
  and end the Query processor.
*)
```

```
 IF R_STATUS = 0 THEN
  BEGIN
    PATHNM::C16 := 'X.TEST.QUERYLST ';  (*NOTE-Trailing blank *)
    EXTEND := 0;
    QEXEC(QUERY_NUMBER,R_STATUS,R_CODE,PATHNM::C2,EXTEND);
    WRITELN('RETURN STATUS FROM QEXEC = ',R_STATUS);
    QEND(QUERY_NUMBER,R_STATUS);
    WRITELN('RETURN STATUS FROM QEND = ',R_STATUS);
  END;

(*This section demonstrates a call to QRECV. Note that QEND has
  disassociated the Query processor and the assigned Query number;
  consequently, this association must be reset. The pathname
  must be reset to the object file.

*)

  QUERY_NUMBER := 2;
  PATHNM::C16 := 'X.TEST.QUERYOBJ ';   (*NOTE-Trailing blank*)
  QINIT(QUERY_NUMBER,R_STATUS,R_CODE,PATHNM::C2,PASSWORD);
  WRITELN('RETURN STATUS FROM QINIT = ',R_STATUS);

 (* Make repeated calls to QRECV until there are no more output lines *)

  IF R_STATUS = 0 THEN
   BEGIN
     REPEAT
      LNG := 80;
      QRECV(QUERY_NUMBER,R_STATUS,R_CODE,DBUFF::C2,LNG);
      IF LNG <> 0 AND R_CODE = '**' THEN
       WRITELN(DBUFF);
     UNTIL LNG = 0 OR R_CODE <> '**';

     WRITELN('RETURN STATUS FROM QRECV = ',R_STATUS);
     WRITELN('RETURN CODE FROM QRECV = ',R_CODE);

     QEND(QUERY_NUMBER,R_STATUS);
     WRITELN('RETURN STATUS FROM QEND = ',R_STATUS);
   END;
 END.
```

6.11.2 Example FORTRAN Program

```
C This program is equivalent to the Pascal example program, PEXPL,
C and demonstrates calls to the following Query subroutines:
C (for FORTRAN-78)
C
C   QINIT (initializes a Query processor)
C   QEXEC (executes the Query object and lists the results to a file)
C   QRECV (performs the functions of QRECV, one line at a time)
C   QEND  (deactivates a Query processor)
C
C DECLARATIONS:
C
      INTEGER QNUMBR,KNT,RSTAT,RCODE,EXTEND,LNG
      INTEGER OBPATH(8),LSPATH(8),PSWORD(2),DBUFF(40)
C
C INITIALIZATIONS
C
      DATA PSWORD /'DBMS'/
      DATA OBPATH /'X.TEST.QUERYOBJ '/
      DATA LSPATH /'X.TEST.QUERYLST '/
C
C Assume that prior to the execution of this program, the desired
C Query has been compiled using QCOMP, and further assume that the
C resulting object resides on 'X.TEST.QUERYOBJ'.
C
C The pathname has been assigned in the initializations section.
C Choose an integer to associate with a Query processor and
C initialize that processor.
C
      QNUMBR=2
      CALL QINIT(QNUMBR,RSTAT,RCODE,OBPATH,PSWORD)
      WRITE(6,1)RSTAT
1     FORMAT('RETURN STATUS FROM QINIT= 'I2)
C
C Check for normal completion of the initialization, execute the
C Query, and end the Query processor.  Note that the pathname in the
C call to QEXEC is different from the pathname in the call to QINIT.
C
      IF(RSTAT.EQ.0)GO TO 2
      GO TO 3
2     EXTEND=0
      CALL QEXEC(QNUMBR,RSTAT,RCODE,LSPATH,EXTEND)
      WRITE(6,5)RSTAT
5     FORMAT('RETURN STATUS FROM QEXEC= 'I2)
      CALL QEND(QNUMBR,RSTAT)
      WRITE(6,6)RSTAT
6     FORMAT('RETURN STATUS FROM QEND= 'I2)
3     CONTINUE
C
C END SCOPE OF LAST IF
C
C This section demonstrates a call of QRECV.  Note that QEND has
C disassociated the Query processor and the assigned Query number;
C consequently, this association must be reset.  The pathname
C must be reset to the object file.
C
      QNUMBR=2
      CALL QINIT(QNUMBR,RSTAT,RCODE,OBPATH,PSWORD)
      WRITE(6,1)RSTAT
C
C
      IF(RSTAT.EQ.0)GO TO 9
      GO TO 10
```

```
C
C Make repeated calls to QRECV until no more lines exist.
C
9       LNG=80
        CALL QRECV(QNUMBR,RSTAT,RCODE,DBUFF,LNG)
        IF((LNG.EQ.0).OR.(RCODE.NE.>2A2A))GO TO 11
        WRITE(6,12)DBUFF
12      FORMAT(40A2)
11      CONTINUE
C
C THE "UNTIL" TEST  NOTE-- >2A2A IS HEX FOR '**'
C
        IF((LNG.NE.0).AND.(RCODE.EQ.>2A2A))GO TO 9
C
C END OF THE REPEAT LOOP
C
        WRITE(6,13)RSTAT
13      FORMAT('RETURN STATUS FROM QRECV= 'I2)
        WRITE(6,14)RCODE
14      FORMAT('RETURN CODE FROM QRECV= 'A2)
        CALL QEND(QNUMBR,RSTAT)
        WRITE(6,6)RSTAT
10      CONTINUE
C  END THE SCOPE OF LAST IF
        END
```

6.11.3 Example COBOL Program

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  QUERY-COBOL-INTERFACE-TEST.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.   TI-990.
OBJECT-COMPUTER.   TI-990.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT OUT-FILE
     ASSIGN TO OUTPUT, "UNIT6"
     FILE STATUS IS ERROR-CODE.

     SELECT IN-FILE
     ASSIGN TO INPUT, "UNIT5"
     FILE STATUS IS ERROR1-CODE.


DATA DIVISION.

FILE SECTION.
FD   OUT-FILE
     RECORD CONTAINS 80 CHARACTERS.
01   OUT-REC              PIC X(80).

FD   IN-FILE
     RECORD CONTAINS 80 CHARACTERS.
01   IN-REC               PIC X(80).
```

```
 WORKING-STORAGE SECTION.
 01    ERROR-CODE          PIC XX.
 01    ERROR1-CODE         PIC XX.
 01    ALT-FILE            PIC XX.
 01    QUERY-NUMBER        PIC 9(5) COMP-1.
 01    RESULT-STATUS       PIC 9(5) COMP-1.
 01    EXT                 PIC 9(5) COMP-1.
 01    LNG                 PIC 9(5) COMP-1.
 01    RESULT-CODE         PIC XX.
 01    BUFFER              PIC X(80).
 01    PN                  PIC X(48).
 01    PASSWORD            PIC X(4).
 01    PAGELEN             PIC 9(5) COMP-1.
 01    PAGEWID             PIC 9(5) COMP-1.
 01    R-CNT               PIC 9(5) COMP-1.
 01    P-LINE.
       05    W-LITERAL     PIC X(30).
       05    W-STAT        PIC X(10).
 01    QUERY-STATEMENT.
       05    Q-STMT-LINE OCCURS 4 TIMES INDEXED BY Q-KNT
                           PIC X(80).
 01    Q-STMT-LENGTH       PIC 9(5) COMP-1.
 01    Q-FORMAT            PIC 9(5) COMP-1.
 01    LIST-TEXT           PIC 9(5) COMP-1.

 PROCEDURE DIVISION.
 MAIN.
       OPEN OUTPUT OUT-FILE
            INPUT  IN-FILE.
*
* Assume that a Query source statement already exists on
* the file designated  "UNIT5" and is known to be four lines
* long.  Read the Query statement into the variable of the
* same name.
*
       PERFORM READ-LINE VARYING Q-KNT FROM 1 BY 1
         UNTIL
           ( Q-KNT IS > 5 ) OR ( Q-KNT EQUAL 5 ).
*
* Compile the Query source statement by calling QCOMP.
* First, set up the arguments for the call.
*
       PERFORM SET-ARGS-COMP.
       CALL "QCOMP" USING QUERY-NUMBER, RESULT-STATUS,
                          RESULT-CODE, QUERY-STATEMENT,
                          Q-STMT-LENGTH, PASSWORD, Q-FORMAT,
                          LIST-TEXT, PN, PAGELEN,
                          PAGEWID, ALT-FILE.
       MOVE "RETURN STATUS FROM QCOMP = " TO W-LITERAL.
       PERFORM WRITE-STAT.

*
* The next step is to execute the Query object using QEXEC.
* Specify that the Query listing be sent to a file whose
* pathname is X.TEST.QUERYCM.
*

       PERFORM CALL-EXEC THRU CALL-END.
*
* This section of the program demonstrates the use of
* QSEND,  which allows you to replace a question mark (?)
* in the Query statement with actual text (assume that the
* Query has already been compiled using QCOMP.)
*
```

```
       MOVE "X.TEST.QUERYOB3" TO PN.
       MOVE 2 TO QUERY-NUMBER.
       CALL "QINIT" USING QUERY-NUMBER, RESULT-STATUS,
                          RESULT-CODE, PN, PASSWORD.
       MOVE "RETURN STATUS FROM QINIT = " TO W-LITERAL.
       PERFORM WRITE-STAT.

*
* First send a 4 to replace the question mark and specify
* that the listing be sent to X.TEST.QUERYLST.   Then,
* send a 3 to replace the question mark and end the
* processor.
*
       IF RESULT-STATUS EQUAL ZERO
         PERFORM SEND-4
         PERFORM SEND-3.
       MOVE 2 TO QUERY-NUMBER.
       PERFORM CALL-END.

*
* This section demonstrates the use of QRECV and QCLR.
* QRECV performs essentially the same function as QEXEC
* except that QRECV returns output lines one at a time.
* QCLR allows the application to abort a QRECV without
* having to accept every line of output.
*

       MOVE 2 TO QUERY-NUMBER.
       MOVE "X.TEST.QUERYOBJ" TO PN.
       CALL "QINIT" USING QUERY-NUMBER, RESULT-STATUS,
                          RESULT-CODE, PN, PASSWORD.
       MOVE "RESULT STATUS FROM QINIT = " TO W-LITERAL.
       PERFORM WRITE-STAT.
       IF RESULT-STATUS EQUAL ZERO
         PERFORM Q-RECV VARYING R-CNT FROM 1
                 BY 1 UNTIL R-CNT EQUAL 4.
       PERFORM Q-CLR.
       MOVE 2 TO QUERY-NUMBER.
       PERFORM CALL-END.
       CLOSE OUT-FILE
             IN-FILE.
       STOP RUN.

 READ-LINE.
       READ IN-FILE.
       MOVE IN-REC TO Q-STMT-LINE(Q-KNT).
       MOVE SPACES TO IN-REC.


*
* Set up the arguments for a call to QCOMP and specify
* the following options:
*   assign the integer 2 to the Query processor to be bid
*   set the Query statement length to 320 characters (4 lines)
*   request a report format for the output
*   request that the Query statement listing from the
*    compiler be included in the output
*   assign the file that resides at X.TEST.QCOMLST as the
*    listing file
*   request default page length and width for the listing
```

```
*
 SET-ARGS-COMP.
      MOVE 2 TO QUERY-NUMBER.
      MOVE SPACES TO ALT-FILE.
      MOVE 320 TO Q-STMT-LENGTH.
      MOVE 1 TO Q-FORMAT.
      MOVE 1 TO LIST-TEXT.
      MOVE "X.TEST.QCOMLST" TO PN.
      MOVE ZERO TO PAGELEN.
      MOVE ZERO TO PAGEWID.
      MOVE "DBMS" TO PASSWORD.

 CALL-EXEC.
      MOVE "X.TEST.QUERYCM" TO PN.
      MOVE ZERO TO EXT.
      CALL "QEXEC" USING QUERY-NUMBER, RESULT-STATUS,
                         RESULT-CODE, PN, EXT.
      MOVE "RETURN STATUS FROM QEXEC=" TO W-LITERAL.
      MOVE RESULT-STATUS TO W-STAT.
      WRITE OUT-REC FROM P-LINE.
      MOVE SPACES TO OUT-REC.

 CALL-END.
      CALL "QEND" USING QUERY-NUMBER, RESULT-STATUS.
      MOVE "RETURN STATUS FROM QEND=" TO W-LITERAL.
      MOVE RESULT-STATUS TO W-STAT.
      WRITE OUT-REC FROM P-LINE.
      MOVE SPACES TO OUT-REC.

*
* Replace the "?" with "4".
*
 SEND-4.
      MOVE "4    " TO BUFFER.
      MOVE 4 TO LNG.
      CALL "QSEND" USING QUERY-NUMBER, RESULT-STATUS,
                         RESULT-CODE, BUFFER, LNG.
      MOVE "RESULT STATUS FROM QSEND = " TO W-LITERAL.
      PERFORM WRITE-STAT.
      IF RESULT-STATUS EQUAL ZERO
       MOVE "X.TEST.QUERYLST" TO PN
       MOVE ZERO TO EXT
       CALL "QEXEC" USING QUERY-NUMBER, RESULT-STATUS,
                          RESULT-CODE, PN, EXT
       MOVE "RETURN STATUS FROM QEXEC = " TO W-LITERAL
       PERFORM WRITE-STAT.
```

```
*
* Replace the "?" with a "3". Note that the extension
* argument in QEXEC is set to 1 in this call, indicating
* a request to extend (not rewrite) the listing file.
*
 SEND-3.
      MOVE "3    " TO BUFFER.
      MOVE 4 TO LNG.
      CALL "QSEND" USING QUERY-NUMBER, RESULT-STATUS,
                         RESULT-CODE, BUFFER, LNG
      MOVE "RESULT STATUS FROM QSEND = " TO W-LITERAL.
      PERFORM WRITE-STAT.
      IF RESULT-STATUS EQUAL ZERO
       MOVE "X.TEST.QUERYLST" TO PN
       MOVE 1 TO EXT
       CALL "QEXEC" USING QUERY-NUMBER, RESULT-STATUS,
                          RESULT-CODE, PN, EXT
       MOVE "RETURN STATUS FROM QEXEC = " TO W-LITERAL
       PERFORM WRITE-STAT.

 WRITE-STAT.
      MOVE RESULT-STATUS TO W-STAT.
      WRITE OUT-REC FROM P-LINE.
      MOVE SPACES TO OUT-REC.

 Q-RECV.
      MOVE 80 TO LNG.
      CALL "QRECV" USING QUERY-NUMBER, RESULT-STATUS,
                         RESULT-CODE, BUFFER, LNG.
      MOVE "RETURN STATUS FROM QRECV = " TO W-LITERAL.
      PERFORM WRITE-STAT.
      IF RESULT-STATUS EQUAL ZERO
       WRITE OUT-REC FROM BUFFER
       MOVE SPACES TO OUT-REC.
*
* At this point, the application can manipulate the data
* returned in buffer and determine whether this particular
* Query is returning the required data.  Using QCLR, the
* program can abort the QRECV without making repeated calls
* to return all the Query output.
*

 Q-CLR.
      CALL "QCLR" USING QUERY-NUMBER, RESULT-STATUS.
      MOVE "RETURN STATUS FROM QCLR = " TO W-LITERAL.
      PERFORM WRITE-STAT.
```

6.12 LINKING THE INTERFACE SUBROUTINES

The program language interface subroutines are located in the directory S$QUERY.PLIOBJ. The link control file for a program includes this directory and an interface module specific to the program language, along with the run-time modules and your task. The following paragraphs contain example link control files for Pascal, FORTRAN, and COBOL programs.

6.12.1 Linking Pascal Programs

An example link control file for a Pascal program is as follows:

```
FORMAT IMAGE,REPLACE
LIBRARY .SCI990.S$OBJECT
LIBRARY .S$TIP.OBJ
TASK <task name>
INCLUDE (MAIN)
INCLUDE <user task pathname>
INCLUDE S$QUERY.PSCINT
INCLUDE S$QUERY.PLIOBJ
END
```

6.12.2 Linking FORTRAN Programs

An example link control file for a FORTRAN program is as follows:

```
FORMAT IMAGE,REPLACE
NOSYMT
LIBRARY .SCI990.S$OBJECT
LIBRARY .FORT78.OSLOBJ
LIBRARY .FORT78.STLOBJ
TASK <task name>
INCLUDE <user task pathname>
INCLUDE S$QUERY.FTNINT
INCLUDE S$QUERY.PLIOBJ
END
```

6.12.3 Linking COBOL Programs

An example link control file for a COBOL program is as follows:

```
FORMAT IMAGE,REPLACE
LIBRARY .SCI990.S$OBJECT
PROC RCOBOL
INCLUDE .S$SYSLIB.RCBPRC
DUMMY
TASK <task name>
INCLUDE .S$SYSLIB.RCBTSK
INCLUDE .S$SYSLIB.RCBMPD
INCLUDE <user task pathname>
INCLUDE S$QUERY.COBINT
INCLUDE S$QUERY.PLIOBJ
END
```

7

Guided Query Utility

7.1 INTRODUCTION

The Guided Query utility allows you to gather data from a file without extensive knowledge of Query-990. You can also use this utility as a training tool for learning the syntax of the Query statement. The Guided Query utility provides easy-to-read displays of the structure for the file being accessed. Guided Query builds a Query statement containing all of the necessary elements. At the end of each Guided Query session, you can display and then save the resulting Query statement. The Query processor can then edit or execute the saved statement.

NOTE

Although the Guided Query utility is an aid to the beginner, it is not intended to be the primary mode of operation for Query sessions.

Refer to the *Model 990 Computer Data Base Administrator User's Guide* for instructions on starting the data base.

7.2 GQUERY COMMAND

To initiate the Guided Query utility, enter the SCI command GQUERY, as follows:

```
[ ] GQUERY
```

If security is installed, the following prompt appears:

```
GUIDED QUERY <VERSION L.V.R YY.DDD>
                    PASSWORD:
```

Respond to this prompt by entering a password that allows read access to the file name you intend to query. Get the password and the file ID from your data base administrator (DBA) or from the person handling such duties. The password entered is saved for later verification.

7.3 CONTROL KEYS

The control keys used in the Guided Query utility are the NEW LINE/RETURN key and the function keys. Pressing the NEW LINE/RETURN key processes the current entry. It also passes control to the next prompt or to the next screen. The NEW LINE/RETURN key is the orange key on the right side of the main (alphabetic) keyboard.

The function keys used throughout the Guided Query session are the F1 through F5 keys located above the main keyboard. Function key one is F1, function key two is F2, and so on. The F1 and F2 keys are the scroll keys. The F1 key causes the display on the screen to move up (toward the end of the file), while the F2 key causes the screen display to move down.

Pressing the F3 key transfers control to the next step if the information displayed is sufficient to continue. Pressing the F4 key transfers control to the previous step. During the Guided Query session, you can press the F4 key at any time to display the previous step. The only step that will not be redisplayed is Step 1.

The F5 function key displays help information when it is available. The help information assists you in completing specific screens.

7.4 GUIDED QUERY SCREENS

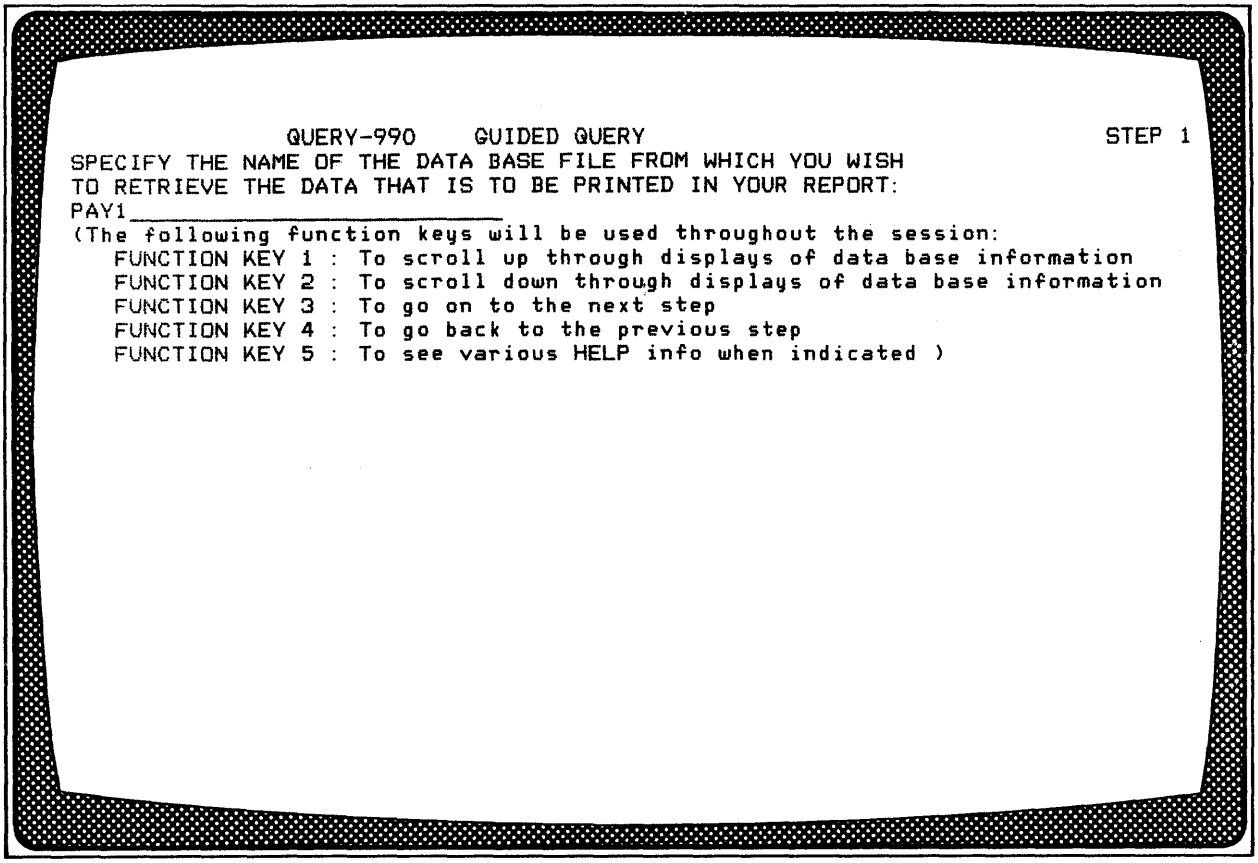
Guided Query screens are divided into five main categories:

- File identification
- Report specifications
- Record-level conditions
- Sort specifications
- Final screens

Except for the final screens, each screen is identified with a step number (for example, Step 1). The screens appear in consecutive order unless you make a request to repeat a group of screens. For example, you can repeat both the report specification and condition screens. However, you cannot repeat the overall sequence. To obtain the help screens, (associated with the more complex steps), press the F5 key. The following paragraphs describe the screens in detail.

7.4.1 File Identification (Step 1)

The following screen appears after the password (if needed) is entered. Note the use of keys F1 through F5.



This screen requests the file name. The file name specified can be the four-character ID for the file's DDL, such as PAY1 in the example, or an alias name from 1 to 20 characters long. When accessing a key indexed, relative record, or sequential file, alternate names can be up to 30 characters long. For information on alias names, refer to the model *Model 990 Computer Data Base Administrator User's Guide.*

Get the alias or standard DDL ID for the file from the DBA or from the person handling the DBA duties. Confirm the availability of the file with the DBA.

7.4.2 Report Specifications

You specify the report lines with nine screens and must complete at least one full cycle of the screens. The nine screens are as follows:

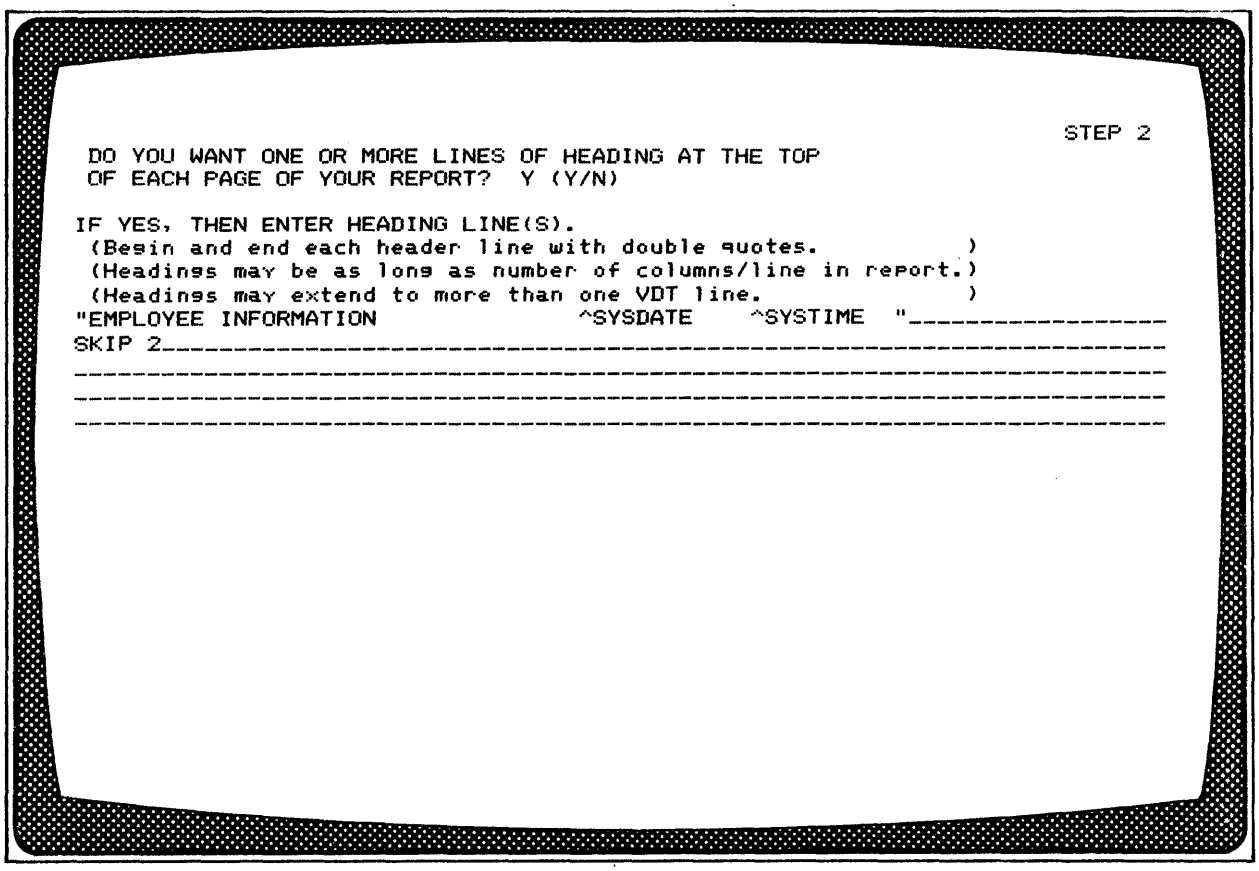
- Main heading (Step 2)
- Line specification (Step 3)
- Field specification (Step 4)
- Listing, counting, totaling, and averaging (Step 5)
- Line heading (Step 6)
- Line-level conditions (Steps 7 through 9)
- Output continuation (Step 10)

7.4.2.1 Main Heading (Step 2). The main heading screen requests the heading that is to appear on your report immediately after the system heading. The system heading identifies the system, date, and time. The main heading should contain general information pertaining to the entire Query. The main heading appears on each page of the output.

For the Guided Query utility, the standard length of the report line is 80 columns. Although the length of the main heading is not limited to any specific number, the heading is truncated if it is longer than 80 characters for a Guided Query execution.

If you specify multiple lines for the heading, the report left justifies all of them. Each line of the heading must begin and end with double quotation marks.

The following example shows the Step 2 screen with a main heading entry:



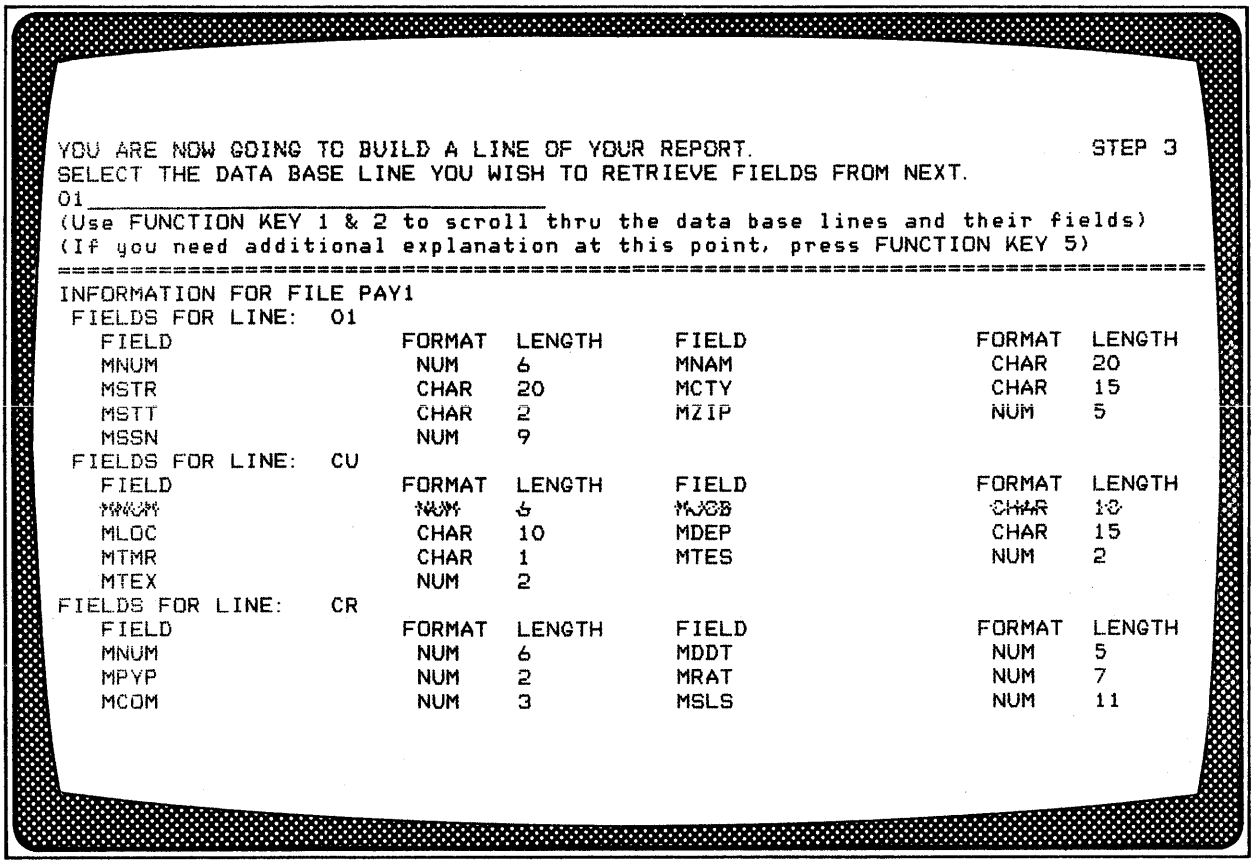
You must respond to the first prompt. If you want to include a main heading, answer Y (yes) to this prompt; if not, answer N (no). If your response is Y, use the available lines to specify the heading you want. Then, press the F3 or NEW LINE/RETURN key to proceed to the next prompt. If your response is N, proceed to the next prompt by pressing the F3 or NEW LINE/RETURN key.

7.4.2.2 Line and Field Specifications (Steps 3 and 4). The line specification screen requests the name of the data base line to be used in building the next report line. This name is chosen from the display on the lower portion of the screen. Enter the name of the desired line in the upper portion of the screen where the prompt characters appear. Use the scroll up (F1) and scroll down (F2) keys to view the display.

Alternate names up to 30 characters long or alias names up to 20 characters long appear on the screen if they have been assigned. Otherwise, the standard DDL ID appears. The format and length of each field is also displayed. Group key fields have type GRP.

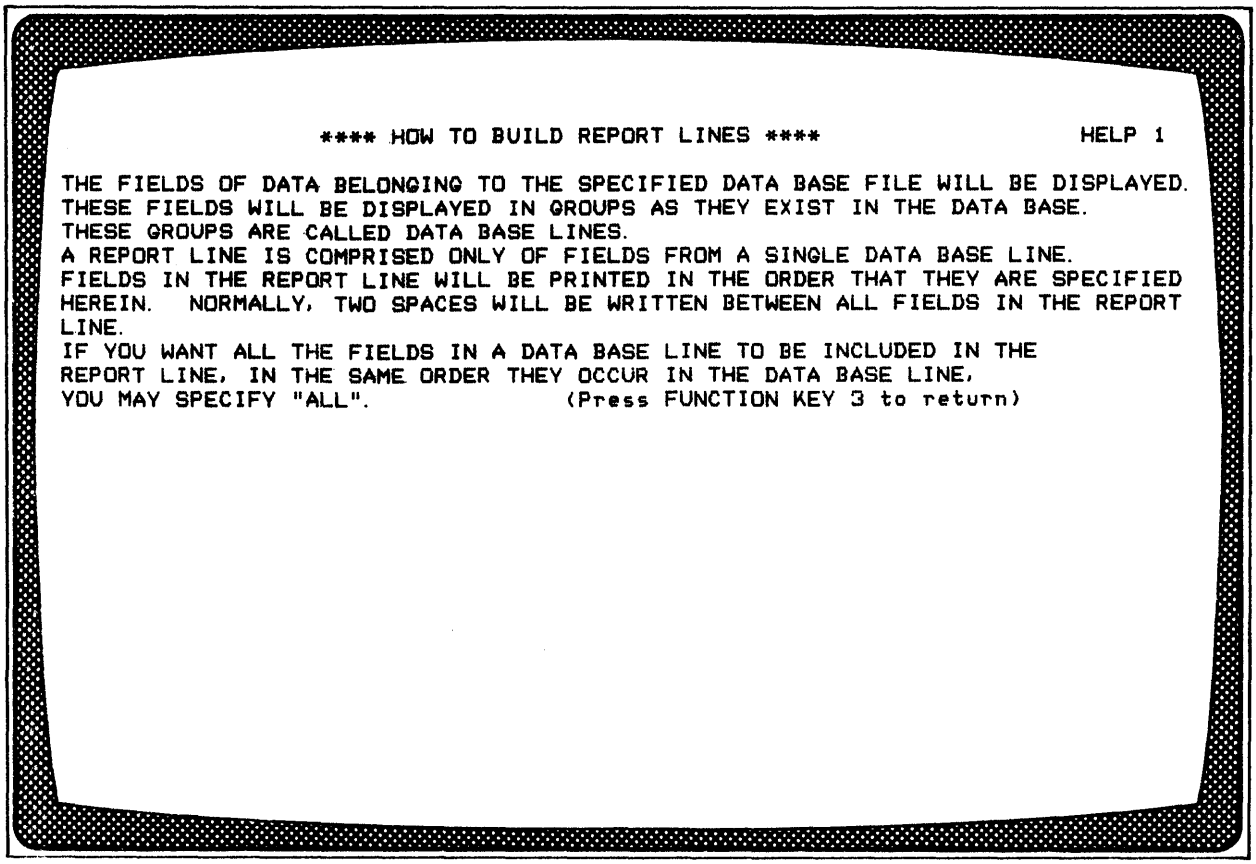
The screen displays the lines, the respective fields, and the corresponding format and length of the fields from the file specified in Step 1. The format code for character fields is CHAR, and the code for numeric fields is NUM. Each field displayed in the lower half of the screen contains a length specification to aid in defining column headings. Step 6 specifies headings. To build effective column headings, consider the length of each field. For each output line, two spaces are automatically placed between each output field.

The following example shows the Step 3 screen with a line specification:



```
YOU ARE NOW GOING TO BUILD A LINE OF YOUR REPORT.                    STEP 3
SELECT THE DATA BASE LINE YOU WISH TO RETRIEVE FIELDS FROM NEXT.
01_____________________________
(Use FUNCTION KEY 1 & 2 to scroll thru the data base lines and their fields)
(If you need additional explanation at this point, press FUNCTION KEY 5)
================================================================================
INFORMATION FOR FILE PAY1
 FIELDS FOR LINE:  01
   FIELD              FORMAT  LENGTH    FIELD              FORMAT  LENGTH
   MNUM                NUM      6      MNAM                CHAR    20
   MSTR                CHAR    20      MCTY                CHAR    15
   MSTT                CHAR     2      MZIP                NUM     5
   MSSN                NUM      9
 FIELDS FOR LINE:  CU
   FIELD              FORMAT  LENGTH    FIELD              FORMAT  LENGTH
   MNUM                NUM      6      MJOB                CHAR    10
   MLOC                CHAR    10      MDEP                CHAR    15
   MTMR                CHAR     1      MTES                NUM     2
   MTEX                NUM      2
FIELDS FOR LINE:  CR
   FIELD              FORMAT  LENGTH    FIELD              FORMAT  LENGTH
   MNUM                NUM      6      MDDT                NUM     5
   MPYP                NUM      2      MRAT                NUM     7
   MCOM                NUM      3      MSLS                NUM     11
```

The HELP 1 screen associated with Step 3 is as follows:



Press the F3 key to proceed to the next prompt.

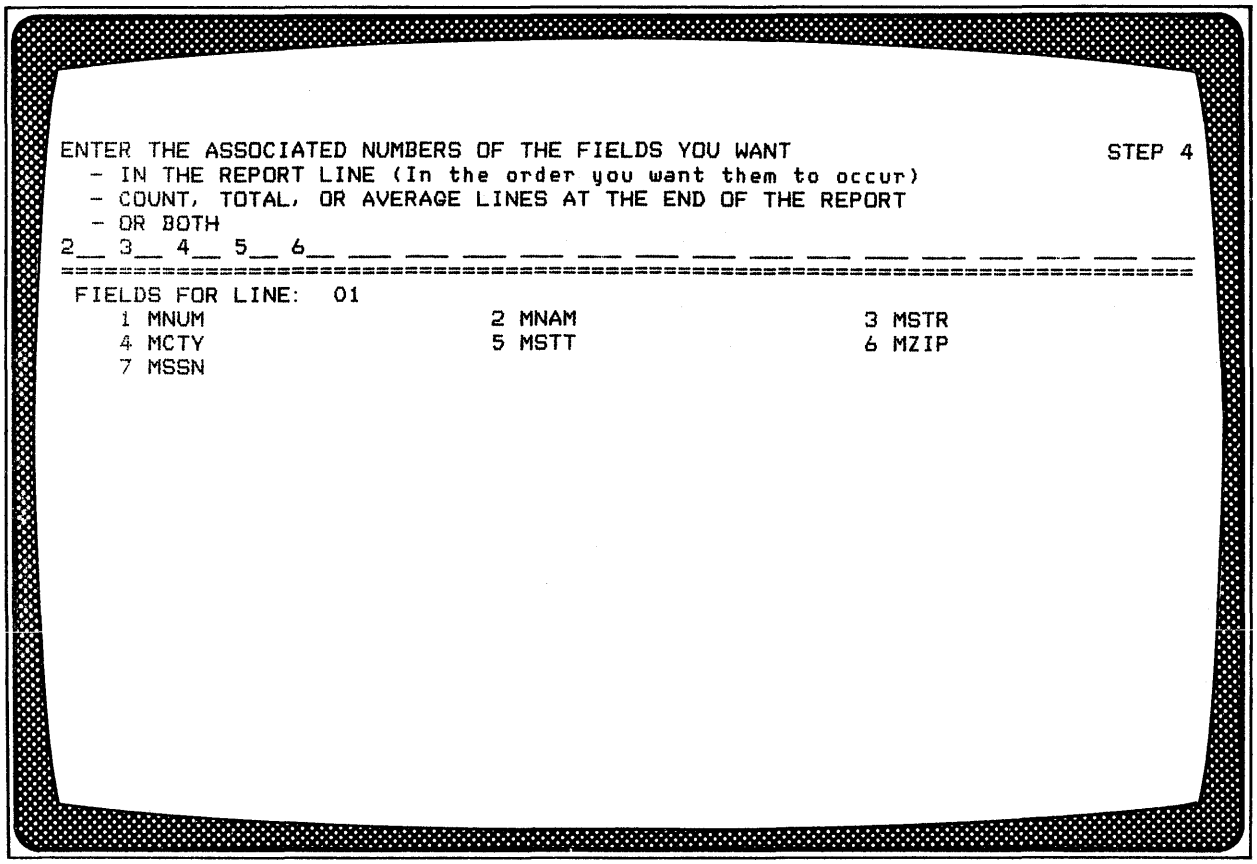
The field specification screen requests the fields that are to be listed, counted, averaged, or totaled in the generated report line. All of the fields for the line specified in Step 3 are displayed with numbers to the left of each field name.

Enter the number associated with each field in the space provided near the middle of the screen. Leading zeros are not required. You can enter a maximum of 20 numbers. Press the NEW LINE/RETURN key to enter each field number and to pass control to the next entry. Press F3 when you complete the number entries for this report line.

Enter ALL to specify all fields of a line. Then, press F3 to proceed to the next screen.

The fields are listed on the report in the order specified in this step. If you enter ALL, the order of the output is the order of the fields in the line.

The following example shows the Step 4 screen with a field specification:

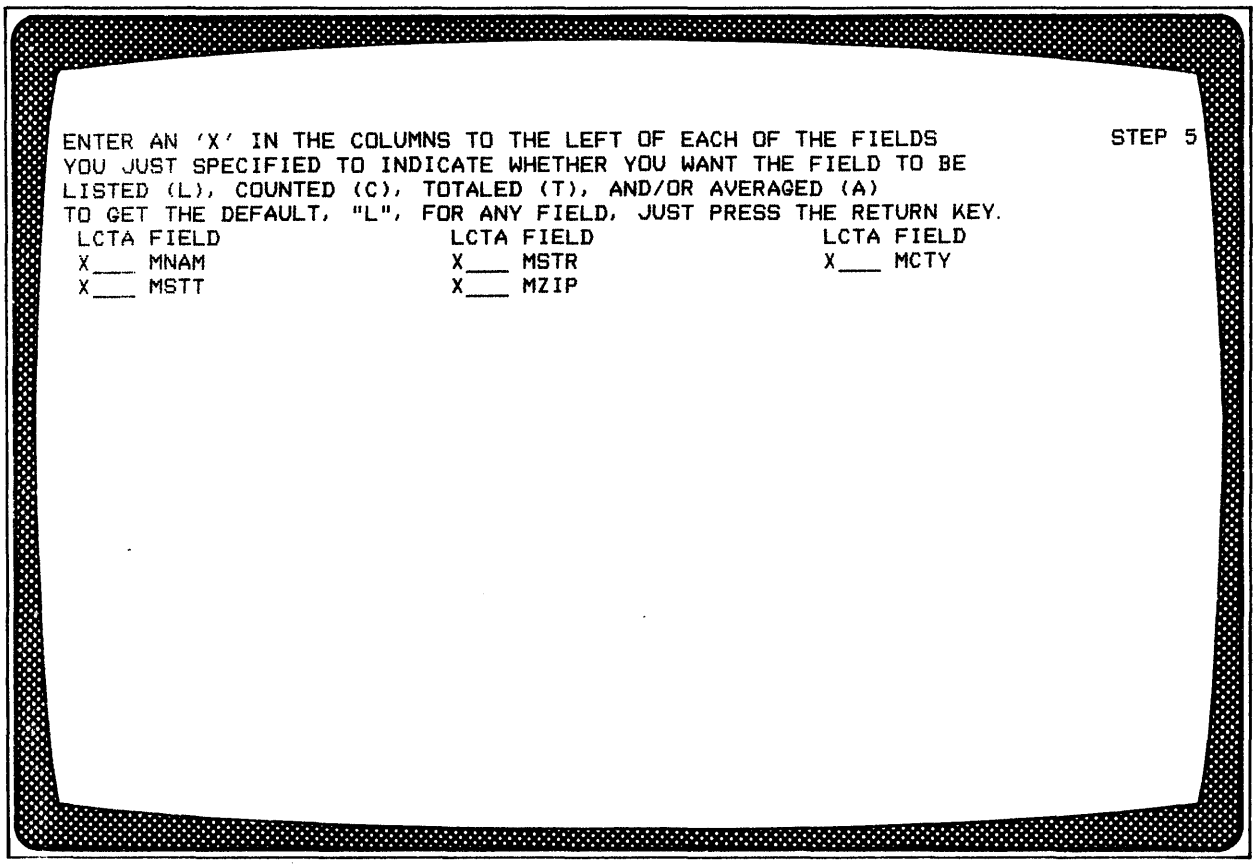


7.4.2.3 List, Count, Total, and Average (Step 5). This screen allows you to select various field options: listing for output, counting occurrences, averaging values of all occurrences, and totaling the values of all occurrences.

You can select one or all of the options by entering an X below the appropriate letter, to the left of the field name. L represents list output, C represents counting, T represents totaling, and A represents averaging. To select the default option (L), press the NEW LINE/RETURN key. After you finish selecting the options, display the next screen by pressing either the F3 or the NEW LINE/RETURN key.

Fields that are listed for output are positioned two spaces apart in the report.

The following example shows the Step 5 screen with entries for listing data from five fields:

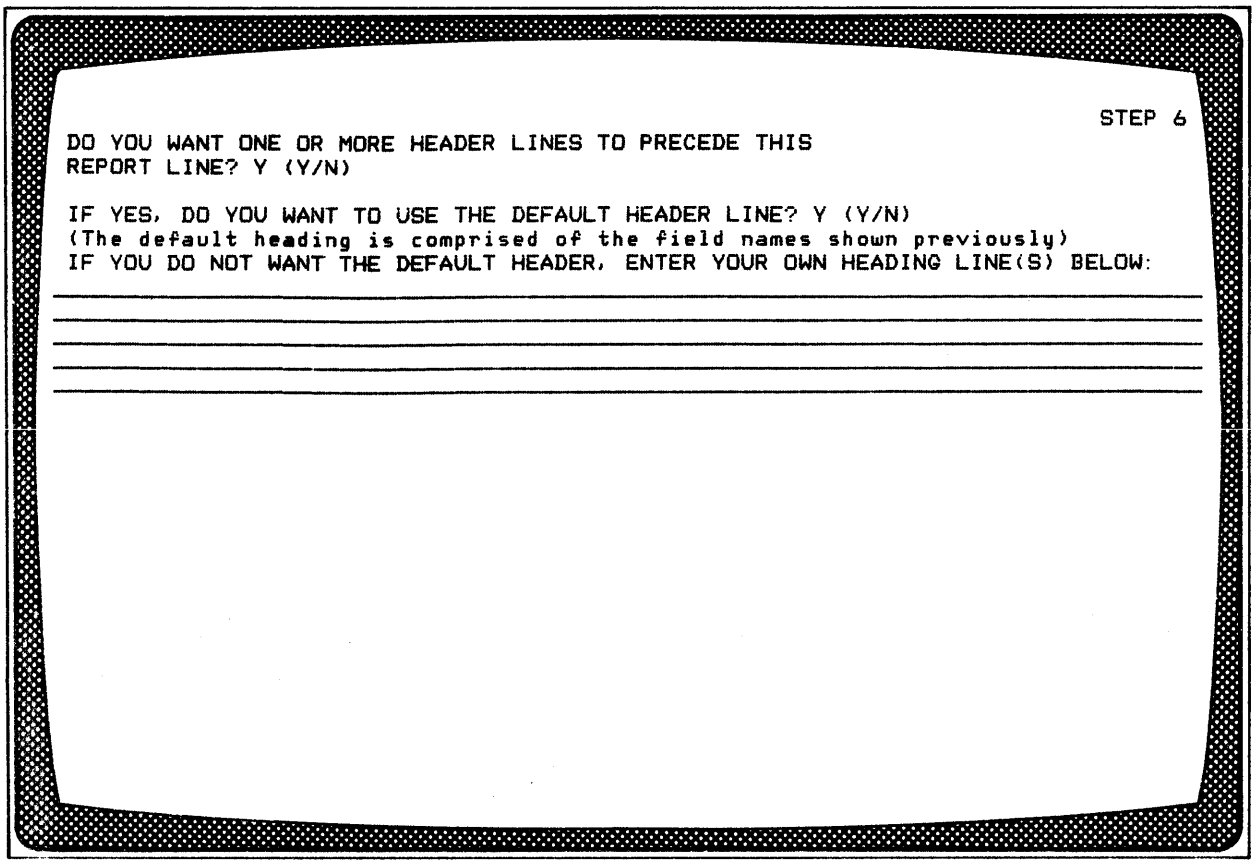


7.4.2.4 Line Heading (Step 6). The line heading screen requests line headings that are to appear above the report line. Any information can be used for a heading. Normally, a column heading is built with field descriptions above each output field. Use the field lengths specified in Step 4 to aid in the calculation of the heading sizes, centering, and so on.

You must respond to the first prompt. A response of N (no) indicates that no headings are desired and passes control to the next screen. If you respond Y (yes), you can choose either the default line headings or supply your own headings. The default heading contains the names of the fields specified in Step 4, positioned above the data for that field. Answer yes to this question to accept the default heading. Control immediately passes to the next screen. To enter headings other than the default heading, answer no to this question and enter the headings as you want them to appear on the report.

If you request line headings, enter each heading within double quotation marks in the space provided at the bottom of the screen. Specify a blank line heading by entering two consecutive quotation marks or by entering SKIP 1. Headings can be a maximum of 132 characters long, but those over 80 characters will be truncated when displayed on a VDT. Once you have entered your headings, press the F3 key to display the next screen.

The following example shows the Step 6 screen specifying the default line header:

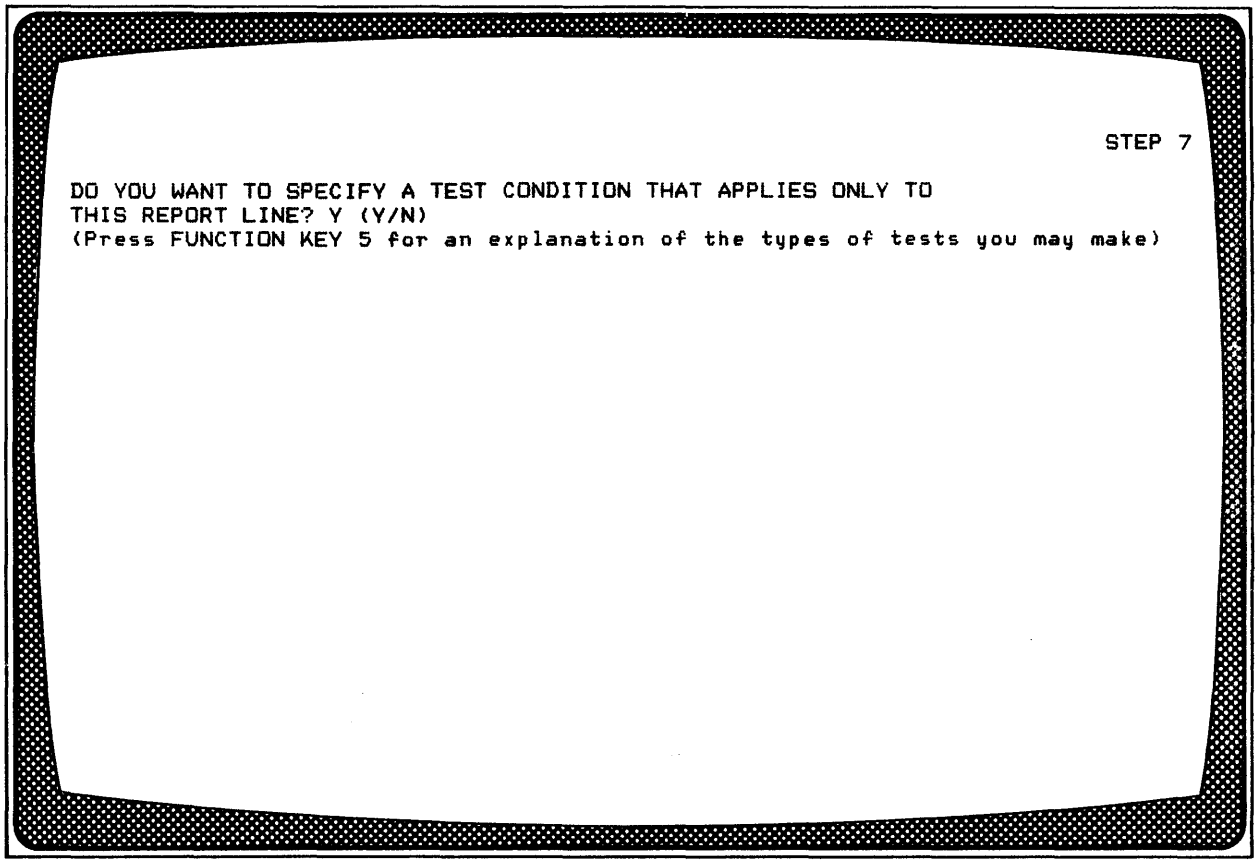


7.4.2.5 Line-Level Conditions (Steps 7, 8, and 9). You can select lines for output based on the value of a specific field within a line. This screen allows you to select those lines.

To build a test condition, you need to know the type of condition to be built, the fields involved, and the desired relationships of the fields. A condition consists of two operands, or fields, and a relational operator. The second operand must be from the same line as the first operand or it must be a constant. Relational operators form the comparison between the two operands. (See Table 4-1 for a list of the operators and the meaning of each.) Group key identifiers (type GRP) cannot be specified as operands.

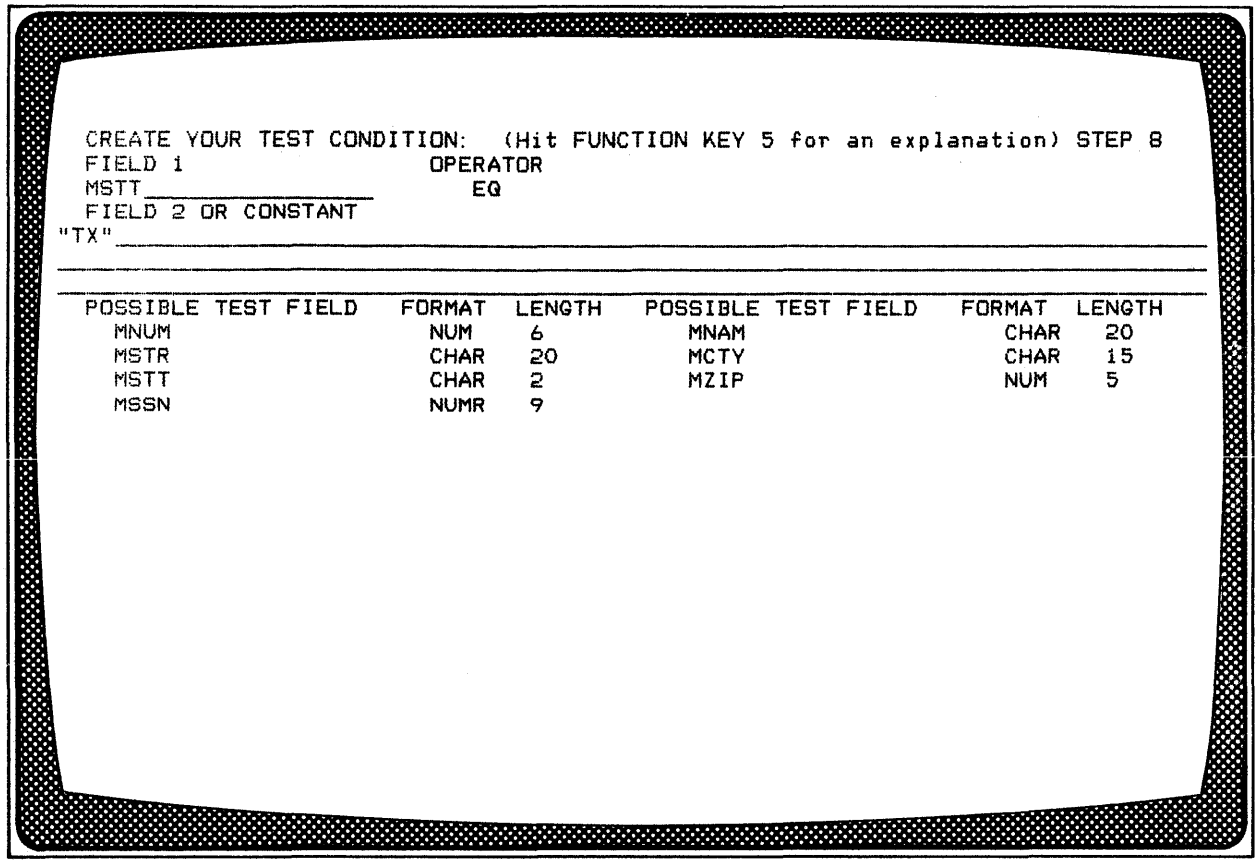
A condition can have multiple statements of relationships or conditions. The Boolean operators AND and OR combine relationships to form complex conditions. If two conditions are connected by AND, both conditions must be true in order for the entire test condition to be true. If two conditions are connected by OR, only one condition need be true for the entire complex condition to be true. If AND and OR operators are mixed in the same complex condition, AND conditions are evaluated first in a left-to-right sequence. Any complex condition is evaluated in a left-to-right sequence.

The first screen (Step 7) for entering the conditions asks if you want to specify a line-level condition. The screen for Step 7 is as follows:



If you want the report to contain all of the lines in the file, answer N (no) to this prompt. As a result, control transfers to Step 10. If you want to establish selection conditions, answer Y (yes). The next screen displayed is Step 8, which allows you to specify the test condition.

The following example screen shows test condition MSTT EQ "TX". Within every eligible line, the field MSTT must equal TX for that line to be included in the report.

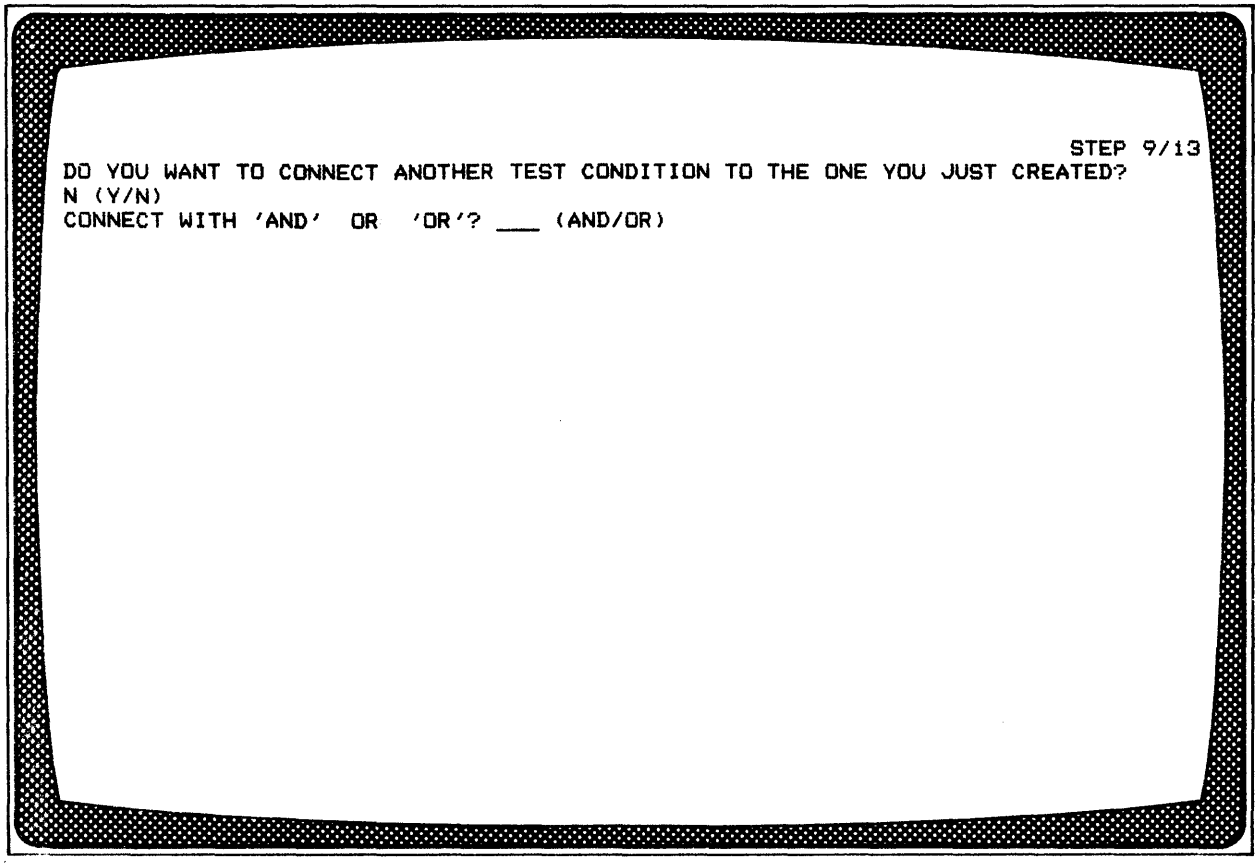


```
CREATE YOUR TEST CONDITION:  (Hit FUNCTION KEY 5 for an explanation) STEP 8
FIELD 1                  OPERATOR
MSTT_____________           EQ
FIELD 2 OR CONSTANT
"TX"_____________________________________________________________________
_____________________________________________________________________________

POSSIBLE TEST FIELD   FORMAT  LENGTH   POSSIBLE TEST FIELD   FORMAT  LENGTH
  MNUM                  NUM     6        MNAM                  CHAR    20
  MSTR                  CHAR    20       MCTY                  CHAR    15
  MSTT                  CHAR    2        MZIP                  NUM     5
  MSSN                  NUMR    9
```

The first prompt requests the field ID or alias to be used as the first operand. The second prompt requests the relational operator for the condition: EQ for equal, LT for less than, and so on. The HELP screen for this step explains the relational operators.

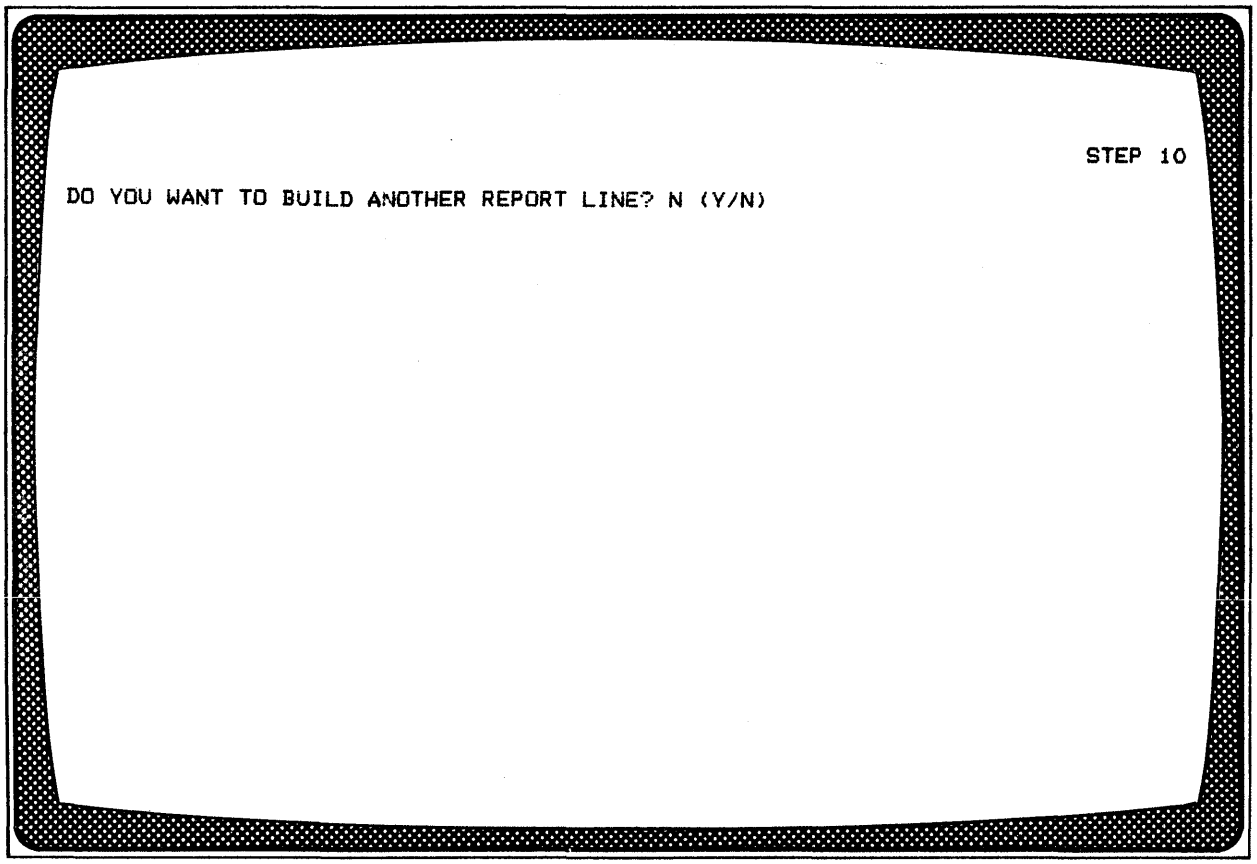
The third prompt requests the second operand. This can be either a field ID alias or alternate name selected from the names listed on the screen, or a numeric constant or literal string. If the second operand is a field ID or alias, the format and length must match the format and length of the FIELD 1 entry. Otherwise, if the format of the first operand is NUM, the second operand must be a numeric constant; if the format of the first operand is CHAR, the second operand must be a character string. You must enclose a literal string in double quotation marks. After you have entered the second operand, press the F3 key to pass control to the next step.

The third screen (Step 9) allows you to attach another condition to the one just created. An example of the screen for Step 9 is as follows:



To create a compound condition, answer Y (yes) to the first prompt. If no other conditions are required for this line, answer N (no) to this prompt. If your response is no, control passes to Step 10. If your response is yes, answer the second prompt with AND or OR. After you respond to this prompt, control returns to Step 8.

7.4.2.6 Output Continuation (Step 10). The output continuation screen asks if more report lines are to be built. The screen for Step 10 is as follows:



A Y (yes) response passes control back to Step 3 to build another report line. An N (no) response passes control to Step 11.

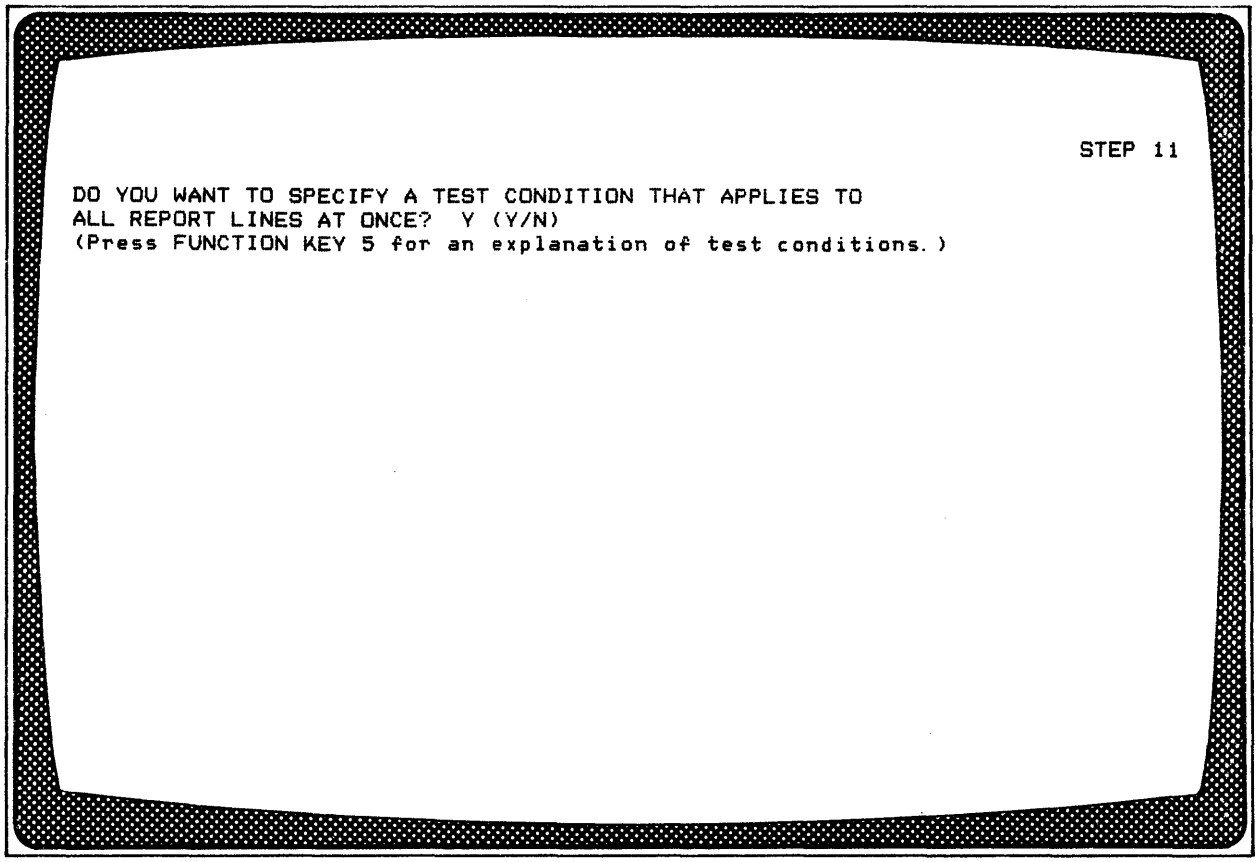
7.4.3. Record-Level Conditions (Steps 11 Through 13)

A record-level condition applies to an entire record. You can apply a record-level test to fields in different lines of the file. Although you can specify multiple line-level conditions, you can specify only one record-level condition.

Before you can select a record for output, the record-level conditions must be met. The variable aspect of record-level conditions is in the use of quantifiers ANY and EVERY. ANY means that only one occurrence need be true for the condition to be true, and EVERY means that every occurrence must be true for the condition to be true. For a complete explanation of conditions and quantifiers, refer to Section 3.

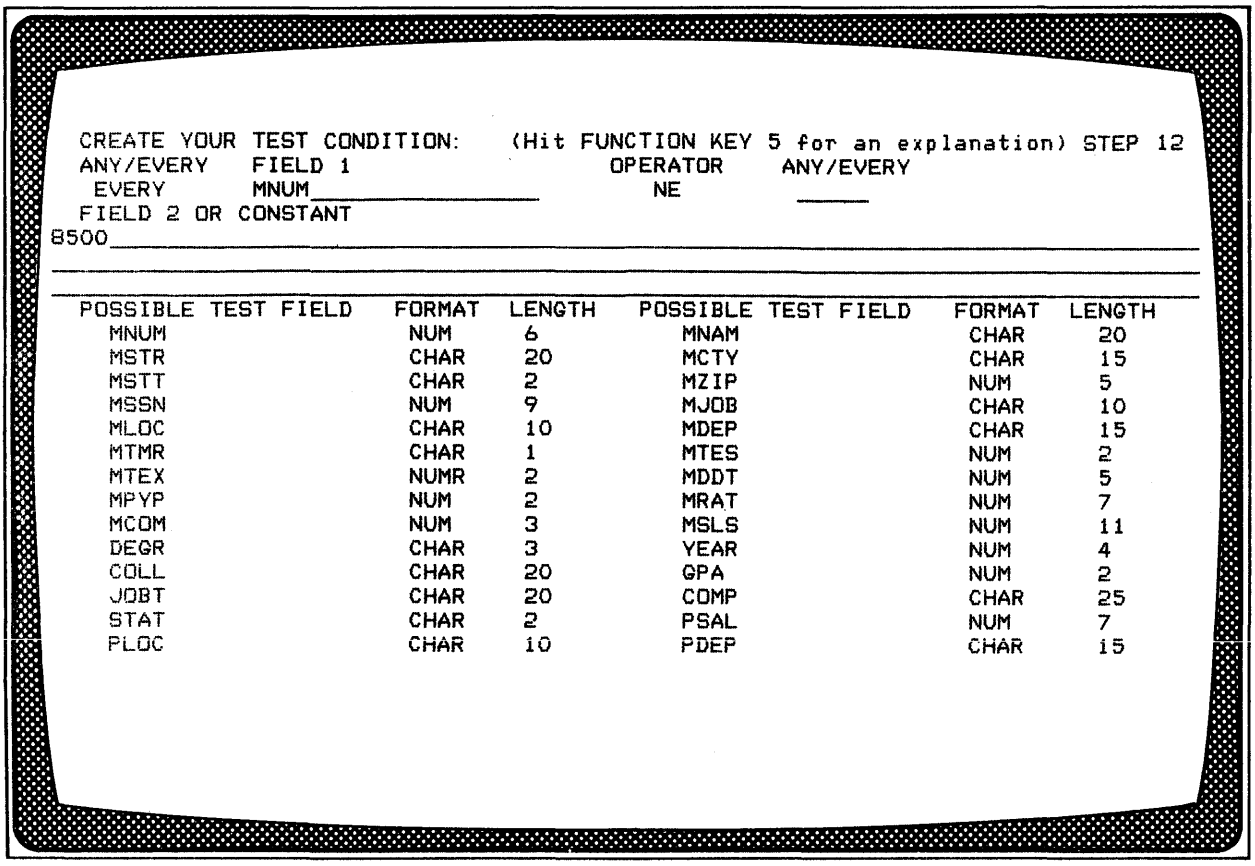
The screens for record-level conditions are similar to the screens for line-level conditions; the primary difference is in the use of ANY and EVERY for record-level tests.

The first screen (Step 11) asks if you want to specify a record-level condition. The following is an example of the screen for Step 11:



If you respond Y (yes) to this question, control transfers to the next screen, Step 12; otherwise, control transfers to Step 14.

The following example shows the Step 12 screen specifying a record-level test condition:



For the first prompt, enter ANY or EVERY. ANY means that only one occurrence in a record of FIELD 1 must meet the test condition for the condition to be true. EVERY means that every occurrence in a record of FIELD 1 must meet the condition for the condition to be true.

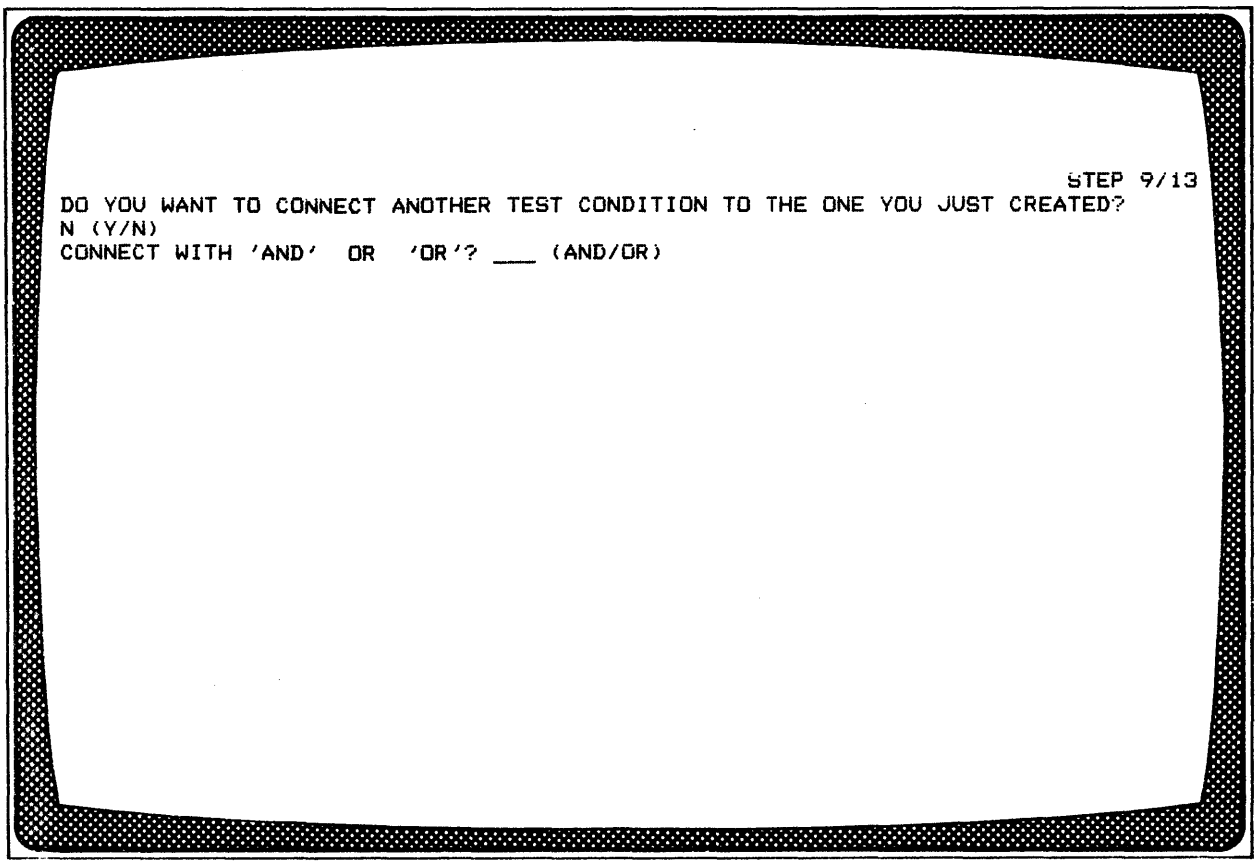
The second prompt requests the first operand (FIELD 1). Enter the desired field ID or alias to be used as the first operand in the test condition. Make a selection from the fields displayed in the lower half of the screen. Use the F1 and F2 function keys to scroll up and down through the various fields.

For the third prompt, enter the desired relational operator that defines the relationship of the operands (such as EQ for equal or LT for less than). The HELP screen describes the relational operators. To display the HELP screen, press the F5 function key. To return from the HELP screen, press the F3 function key.

If FIELD 2 is to be a field ID alias or alternate name, enter either ANY or EVERY for the fourth prompt. If FIELD 2 is to be a numeric constant or a literal string, skip the fourth prompt by pressing the NEW LINE/RETURN key.

The fifth prompt requests the field name, alias, alternate name or constant for the second operand (FIELD 2) of the condition. A nonconstant operand should be a field ID, alias, or alternate name from one of the possible test fields displayed in the lower half of the screen. The format and length of the field ID, alias, or alternate name must match the format and length of the field ID, alias, or alternate name specified for FIELD 1. You must enclose a literal string in double quotation marks, but a numeric constant does not require them. Use the format of the field specified in FIELD 1 to determine the type of constant to enter. If the format of FIELD 1 is CHAR, specify a literal string. If the format of FIELD 1 is NUM, specify a numeric constant.

The third screen allows you to build compound statements for the record-level condition. An example of the screen for Step 13 is as follows:

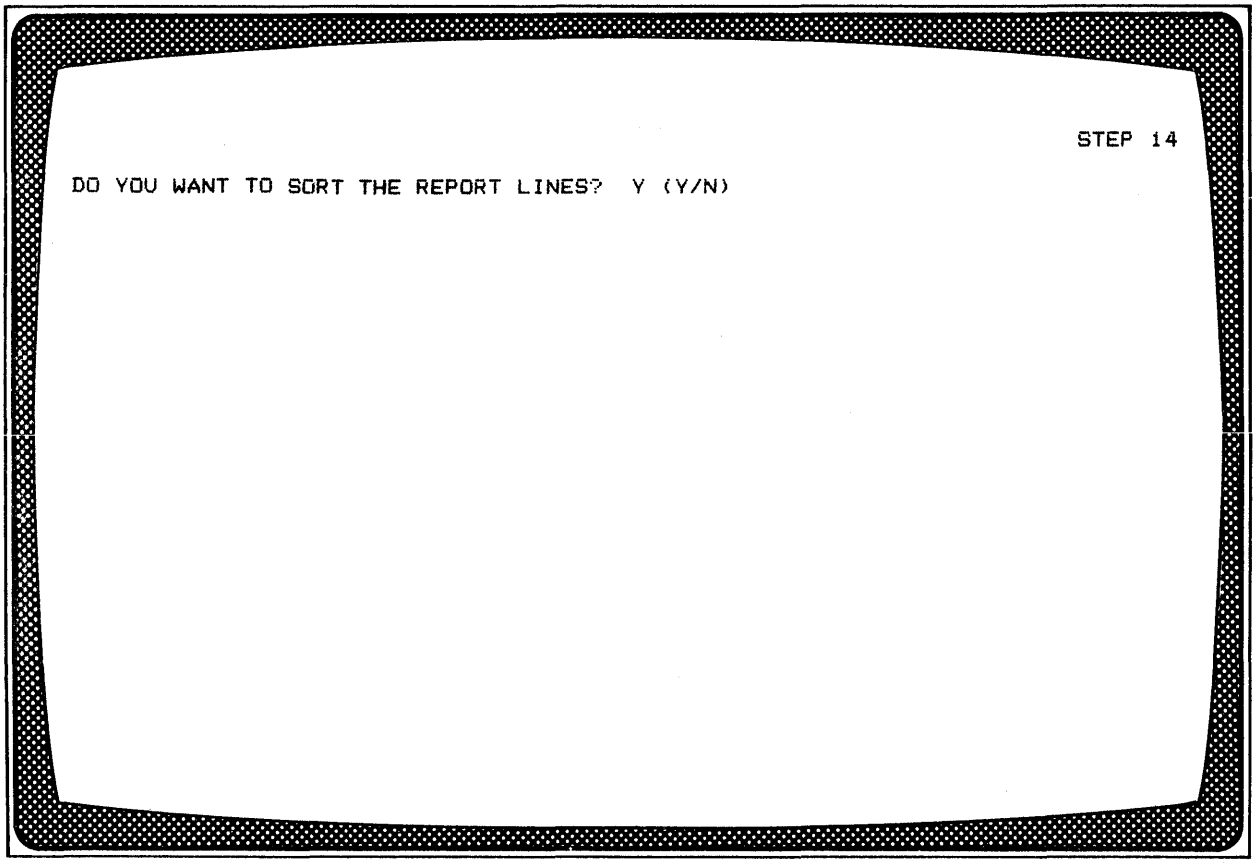


If you respond N (no) to the first prompt, control immediately passes to Step 14; otherwise, you must answer the second prompt. Enter either AND or OR in this field. When you press either the NEW LINE/RETURN key or the F3 key, control returns to Step 12.

7.4.4 Sort Specifications (Steps 14 Through 17)

You can specify the order in which the report lines are printed by completing the sort specification screens. However, you cannot select the sort option unless the Sort/Merge utility has been installed on your system program file.

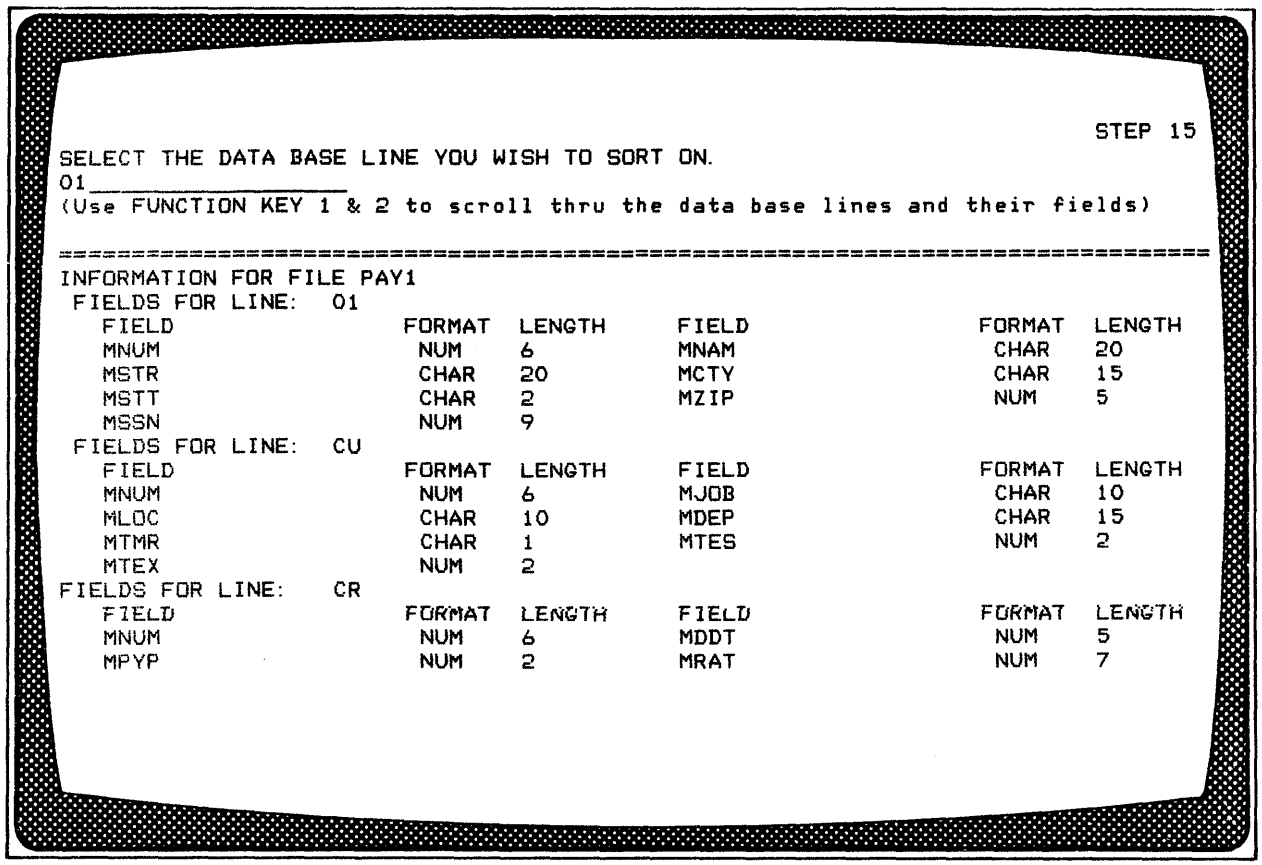
The first screen of the sort specifications (Step 14) asks if the report lines are to be sorted. An example of this screen is as follows:



If you respond N (no) to this prompt, control passes to Step 18. If you respond Y (yes), the next screen (Step 15) appears.

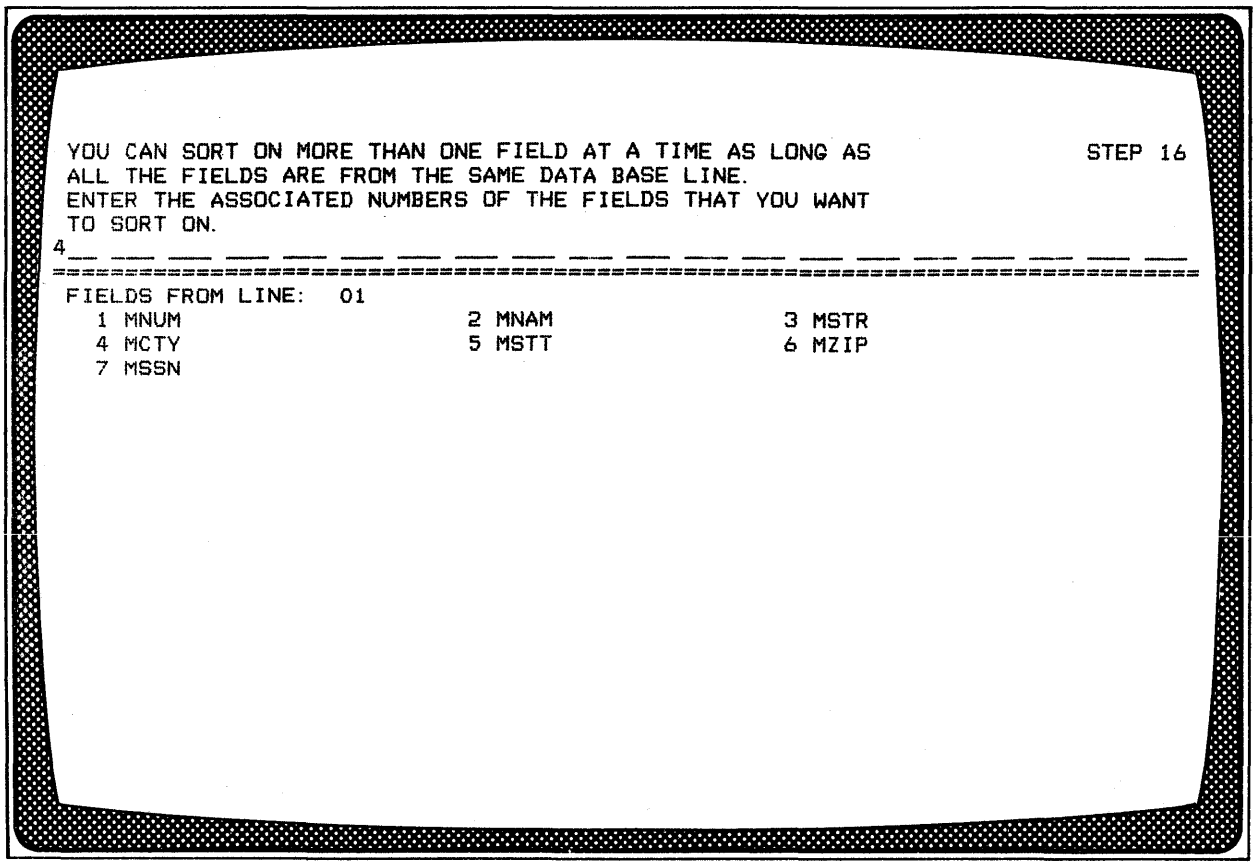
The next screen allows you to specify the line on which to sort. You can specify sorting on more than one data base field, but all sort fields must be from the same data base line.

Enter the number of the line to be sorted. Then, press the F3 key or the NEW LINE/RETURN key to proceed to the next screen. The following is an example of the screen for Step 15 when sorting is specified on the line EMPLOYEE-INFO:



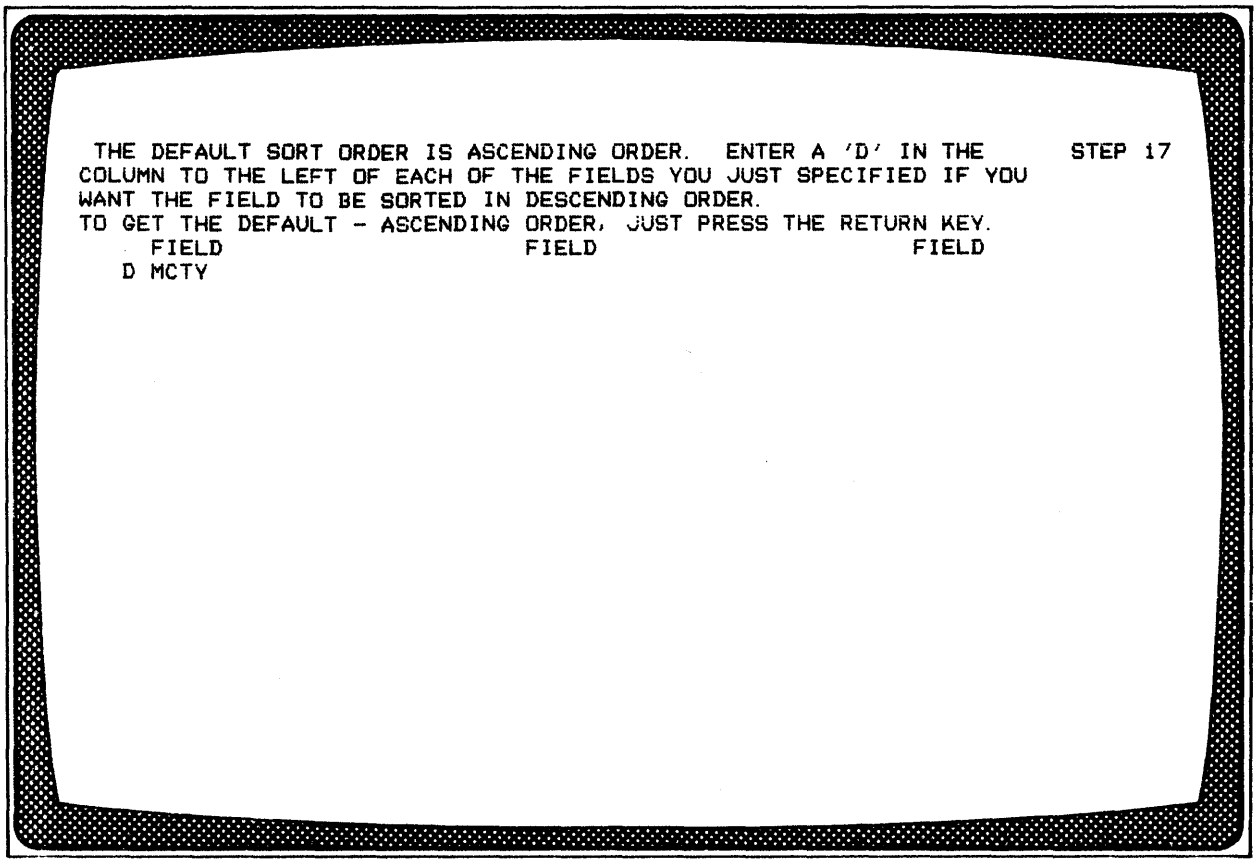
```
                                                                    STEP 15
SELECT THE DATA BASE LINE YOU WISH TO SORT ON.
01_____________________
(Use FUNCTION KEY 1 & 2 to scroll thru the data base lines and their fields)

================================================================================
INFORMATION FOR FILE PAY1
 FIELDS FOR LINE:   01
   FIELD                FORMAT  LENGTH     FIELD                FORMAT  LENGTH
   MNUM                  NUM    6          MNAM                  CHAR   20
   MSTR                  CHAR   20         MCTY                  CHAR   15
   MSTT                  CHAR   2          MZIP                  NUM    5
   MSSN                  NUM    9
 FIELDS FOR LINE:   CU
   FIELD                FORMAT  LENGTH     FIELD                FORMAT  LENGTH
   MNUM                  NUM    6          MJOB                  CHAR   10
   MLOC                  CHAR   10         MDEP                  CHAR   15
   MTMR                  CHAR   1          MTES                  NUM    2
   MTEX                  NUM    2
FIELDS FOR LINE:   CR
   FIELD                FORMAT  LENGTH     FIELD                FORMAT  LENGTH
   MNUM                  NUM    6          MDDT                  NUM    5
   MPYP                  NUM    2          MRAT                  NUM    7
```

The next screen (Step 16) requests the fields that are to be used as the sort keys. This screen is similar to the screen used in Step 4. You can enter a maximum of 20 numbers. After you have entered all of the numbers, press the F3 or NEW LINE/RETURN key to proceed to the next screen. The following is the screen for Step 16 when the field to be sorted is the fourth field, MCTY:



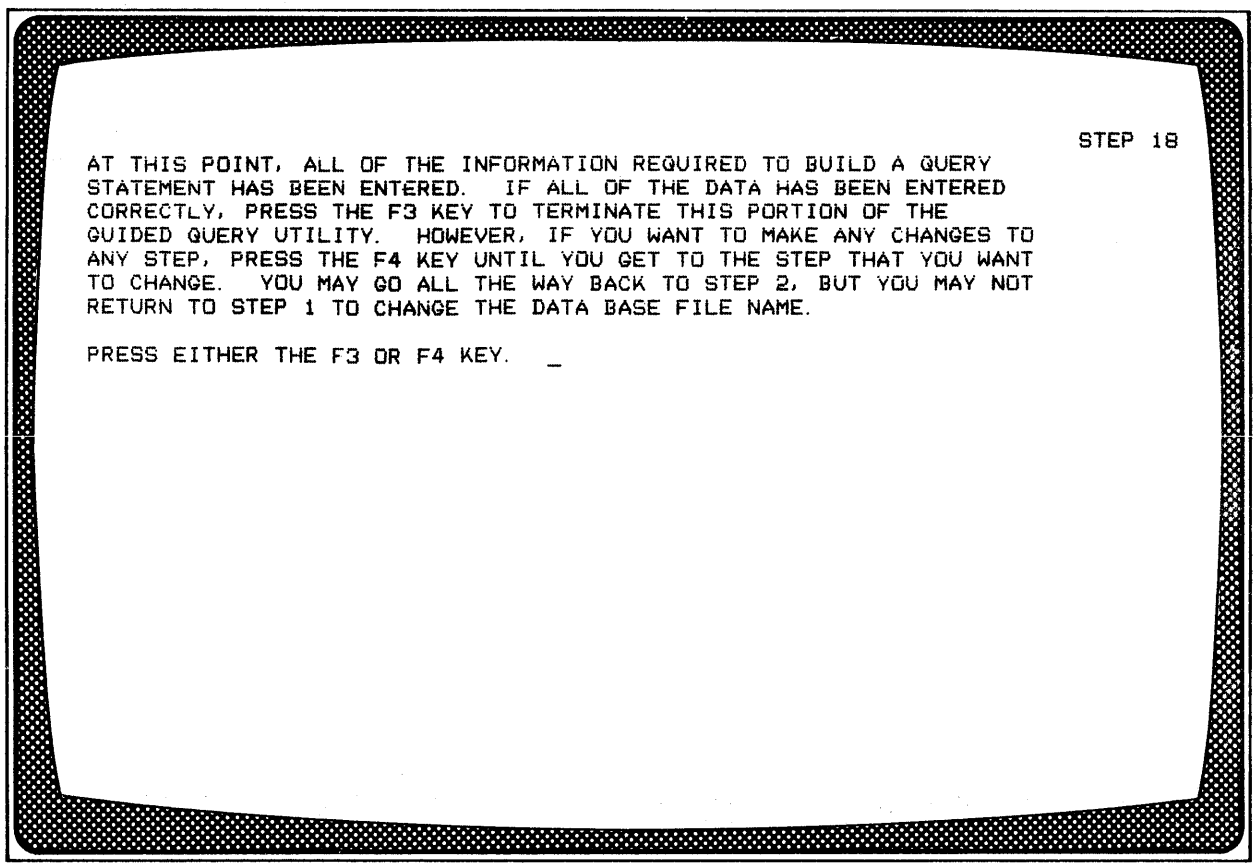
You can sort the field in either ascending or descending order. Since the default sort order is ascending, you need not enter anything to sort in ascending order. Enter a D next to each field to be sorted in descending order. When you have completed all entries, press the F3 or NEW LINE/RETURN key to proceed to the next screen.

In the following example of Step 17, descending order is selected:



The next step, Step 18, allows you to return to any of the Guided Query screens and make changes or corrections. If all of the data has been entered correctly, press F3 key to proceed to the termination screens. If corrections or changes are necessary, press the F4 key until the appropriate step appears. You can return all the way to Step 2; however, you cannot return to Step 1 to change the file ID.

An example of the screen for Step 18 is as follows:



7.4.5 Termination Screens

At this point, you have entered all of the data required to build a Query statement. The information collected has been formatted into an input Query statement. The termination screens save the Query statement and execute it.

The Guided Query utility has two termination screens. The first has two prompts, as follows:

```
END OF GUIDED QUERY ENTRY
        SAVE FILE PATHNAME:
DO YOU WANT TO SEE IT?:  YES
```

For the first prompt, enter the pathname of the file in which you want to save the Query statement that the Guided Query utility just built. The Query processor can execute the saved Query information or statement again by using the QUERY command. Press the NEW LINE/RETURN key, leaving a blank as your response, if you do not want to save the statement.

The next prompt asks if the Query statement just built should be displayed on the screen. Enter either Y (yes) or N (no). You can accept the default entry (Y) by pressing the NEW LINE/RETURN key; as a result, the statement appears. Enter N (no) when you do not want to display the statement.

By displaying the Query statement, you become familiar with Query statement syntax (Sections 2 through 5). To proceed to the next screen after observing the Query statement, press the HELP/CMD key.

The final termination screen allows you to execute the Query that the Guided Query utility built:

```
END OF GUIDED QUERY ENTRY
 DO YOU WANT TO EXECUTE IT?:  YES
   REPORT OUTPUT PATHNAME:
```

An N (no) response causes immediate termination of the session. To select the Y (yes) response, press the NEW LINE/RETURN key. Control then passes to the next prompt. When you enter the pathname for the output of the execution and execute the Query statement, the session ends.

Figure 7-1 shows the statement and output produced using the responses provided in the preceding example screens.

Query Statement:

```
LIST
MNAM MSTR
MCTY MSTT MZIP
HEADER
WHERE
MSTT EQ "TX"
;
BY KEY BY LIST
FROM PAY1
SORTED BY MCTY :D

HEADER
"EMPLOYEE ONFORMATION                ^SYSDATE     ^SYSTIME   "
SKIP 2
WHERE
EVERY MNUM NE 8500
```

Query Output:

```
EMPLOYEE ONFORMATION                 06/07/82     13:20:37

MNAM                 MSTR                 MCTY             MSTT  MZIP
HOWELL, JOHN         555 RIO GRANDE       GRANGER          TX    78787
MNAM                 MSTR                 MCTY             MSTT  MZIP
MEREDITH, JOHN       98 N. LAMAR          GOLIAD           TX    89898
MNAM                 MSTR                 MCTY             MSTT  MZIP
STEPHENS, JANET      56 PURNAM DR         ECHO             TX    87989
MNAM                 MSTR                 MCTY             MSTT  MZIP
HAYNES, BILL         500 LAIRD            DEL CURTO        TX    85269
```

Figure 7-1. Guided Query Example

8

Error Messages

8.1 INTRODUCTION

This section describes Query-990 error messages and defines the action to be taken in response to each message. A table of corresponding internal message codes, Table 8-1, is included at the end of this section.

The two main categories of errors are Query processor errors and Guided Query errors. Error messages in this section are listed in numerical order within the appropriate error category.

The prescribed actions are guidelines only. The action might be inappropriate when a mistake other than the one noted is causing the problem. For example, the beginning quotation mark for a literal could be missing and Query-990 would most likely examine it as an alias. If the literal were more than 20 characters, it would be flagged because an alias cannot be over 20 characters in length. Although the corresponding error and action would seem correct to Query-990, they would be inappropriate. Therefore, use judgement in determining the problem, and the action to take following an error.

8.2 QUERY PROCESSOR ERRORS

Query processor errors occur during the execution of the Query processor. The two categories of Query processor errors are run-time errors and Query statement errors.

8.2.1 Run-Time Errors

Run-time errors have similar meanings but involve different files; consequently, you should read carefully all information associated with the error message. Run-time errors appear on the terminal after execution of the statement. When you are running a batch stream Query, run-time errors appear in the batch stream listing.

8.2.2 Query Statement Errors

Invalid syntax or semantic errors used in the Query statement cause Query statement errors. These are errors in the way the Query statement is worded or errors in the meaning of the Query statement as it relates to the current DBMS-990 or DD-990 environment. When Query statement errors occur, the message FATAL ERRORS IN QUERY STATEMENT appears on the screen. (The message is written in the batch stream listing instead of the screen if the Query is executed in batch mode.) The individual errors that caused this message are output to the listing file. When you request a listing of the Query statement, Query statement errors appear on the first page of the listing file along with the Query statement. The two types of Query statement errors are miscellaneous Query statement errors and syntax errors.

8.2.2.1 Miscellaneous Query Statement Errors. Miscellaneous Query statement errors relate to the overall conformity of the Query statement. Miscellaneous Query statements appear at the end of the Query listing and involve the construction of the entire statement. These errors are often general in nature and do not relate to any specific line in the Query statement.

8.2.2.2 Syntax Errors. Syntax errors deal with the formulation of the Query statement. Each error message appears on a single line below the Query line that contains the error. A corresponding exclamation point directs you to the error in the statement. When you request a listing of the Query statement, the syntax errors appear on the first page of the listing file along with the Query statement.

Figure 8-1 is an example of an invalid Query statement that contains two errors following the BY clause. The first error resulted from a missing second operand in the third condition of the WHERE clause. The exclamation point for the first error is positioned after the error, which is as close as Query-990 can determine. The exclamation point for the second error points to the second BY clause because only one BY clause is needed. Query-990 assumes that the first one is correct and flags any subsequent BY clauses.

```
WHERE EVERY LTR1 EQ "A" AND
      EVERY CLR1 EQ "BLUE"
      ANY   CLR2 EQ
BY KEY  FROM EXMP BY LIST
!                 !
** ERROR ** MISSING OPERAND
** ERROR ** LISTING ORDER ALREADY SPECIFIED
LIST LTR1,CLR1;
     LTR2,CLR2 HEADER "LINE 2 INFO";
     LTR3;
```

Figure 8-1. Example Syntax Errors

8.2.2.3 Query-990 Error Message Format. Query-990 error messages are in the following form:

aaa QUERY-nnnn <message>

The aaa is the error source, which can be one, two, or three characters as follows:

I — informative
W — warning
U — user fatal error
S — system fatal error
H — hardware fatal error
US — user or system fatal error
UH — user or hardware fatal error
SH — system or hardware fatal error
USH — user, system, or hardware fatal error

The nnnn is the QUERY message number.

For example, if the message number is 0001, you can use the Show Expanded Message (SEM) command to display the message explanation, as follows:

```
SHOW EXPANDED MESSAGE
        MESSAGE CATEGORY:  QUERY
                MESSAGE ID:  0001
```

The following appears:

Explanation:
The Query aborts because the user has executed a Kill Task (KT). This message can also indicate an internal Query error.

User Action:
Resubmit the Query. If an internal Query error is suspected, you might need to restart DBMS-990. Contact customer support if necessary.

The Query processor error messages are as follows:

US QUERY-0001 INTERNAL QUERY ERROR — CODE:?1 WP:?2 PC:?3 ST:?4

Explanation:
The Query aborts because the user has executed a Kill Task (KT). This message can also indicate an internal Query error.

User Action:
Resubmit the Query. If an internal Query error is suspected, you might need to restart DBMS-990. Contact customer support if necessary.

US QUERY-0002 FATAL ERRORS IN QUERY STATEMENT

Explanation:
Errors were encountered in compilation of the Query statement that prevented it from being executed.

User Action:
Check the Query listing to see what the errors were and correct them.

US QUERY-0003 ERROR STATUS RETURNED FROM DBMS/DM—?1

Explanation:
DBMS-990 or DD-990 returned an abnormal status code during the execution phase of the Query. ?1 is the status code.

User Action:
Find the exact meaning of this status code in Appendix A of the *Model 990 Computer Data Base Management System Programmer's Guide* or Section 8.5 of the *Model 990 Computer Data Dictionary User's Guide*. Take the necessary and appropriate action based on the description of the error.

US QUERY-0004 UNABLE TO OPEN CHANGE DATA FILE PATHNAME

Explanation:
Change data constants were included in the Query statement, but no change data file pathname was specified in the QUERY procedure.

User Action:
Respond with NO to the prompt DEFAULT REPORT PARAMETERS?, and give the pathname of the data file as the response to CHANGE DATA PATHNAME.

US QUERY-0005 UNABLE TO OPEN INPUT QUERY STATEMENT PATHNAME

Explanation:
This file is the input file for using a saved Query statement. The cause of the error could be an incorrect pathname, a full directory, or a full disk space.

User Action:
Locate the cause of the problem by executing a List Directory (LD) command to get the number of entries or a Show Volume Status (SVS) command to determine the amount of available disk space. Check other possible causes for the error, and then respond accordingly.

US QUERY-0006 UNABLE TO RETRIEVE PARAMETERS ON BID OF QUERY

Explanation:
Query was unable to retrieve one of the required parameters on the bid of the Query task.

User Action:
Check that the bid parameters are in their proper order and contain valid values. Contact customer support if necessary.

US QUERY-0007 UNABLE TO OPEN OUTPUT QUERY STATEMENT PATHNAME

Explanation:
This file is the output file for saving the Query statement text. The cause of the error could be an incorrect pathname, a full directory, or a full disk space.

User Action:
Locate the cause of the problem by executing a List Directory (LD) command to get the number of entires or a Show Volume Status (SVS) to determine the amount of available disk space. Check other possible causes of the error, and then respond accordingly.

US QUERY-0008 UNABLE TO OPEN REPORT/TRACE ACCESS NAME

Explanation:
The output file contains the retrieved data from the files. The cause of the error could be an incorrect pathname, a full directory, or a full disk space.

User Action:
Locate the cause of the problem by executing a List Directory (LD) command to get the number of entries or a Show Volume Status (SVS) to determine the amount of available disk space. Check other possible causes for the error, and then respond accordingly.

W QUERY-0009 NO DATA SELECTED FOR OUTPUT

Explanation:
No data was found that met all of the conditions specified.

User Action:
No action is needed (informative message).

US QUERY-0010 DATA TYPE NOT SUPPORTED BY QUERY

Explanation:
One of the data types not supported was encountered.

User Action:
Verify that all fields used in the Query statement have supported data types. Use the DDL listing of the file for verification.

US QUERY-0011 QUERY ABORTED

Explanation:
The user entered "A" (for abort) after pressing the CMD or ENTER key during the edit of a Query statement.

User Action:
None required.

US QUERY-0012 UNFORMATTED CAN'T BE OUTPUT TO DEVICE

Explanation:
Unformatted output was requested in the report parameters, and the REPORT/TRACE ACCESS NAME was a device name rather than a file name.

User Action:
Send unformatted output, which contains binary data, only to a sequential or relative record file.

US QUERY-0013 EXPRESSION OPERAND MISSING

Explanation:
An operand is missing in the expression portion of a DEFINE clause.

User Action:
Check to see that the expression syntax is valid.

US QUERY-0014 UNMATCHED RIGHT PARENTHESES

Explanation:
A DEFINE expression has more right parentheses than left parentheses.

User Action:
Count the parentheses and correct the expression.

US QUERY-0015 READ PAST EOF — QUERY ENDS UNEXPECTEDLY

Explanation:
The Query parser has reached an end-of-file before the logical end of the Query statement.

User Action:
Check the listing and verify that the Query statement is valid.

US QUERY-0016 SYNTAX ERROR IN DEFINE EXPRESSION

Explanation:
A syntax error of an indeterminate nature was detected in a DEFINE expression.

User Action:
Check the syntax and spelling in the DEFINE clause.

US QUERY-0017 DEFINE VARIABLE EXPECTED

Explanation:
A DEFINE variable is expected after the keyword DEFINE or after a semicolon within the DEFINE clause.

User Action:
Check to see if the DEFINE clause syntax is valid.

US QUERY-0018 EXPRESSION OPERATOR MISSING

Explanation:
An operator is missing in the expression portion of a DEFINE clause.

User Action:
Check to see if the expression syntax is valid.

W QUERY-0019 NO DATA SELECTED FOR MODIFICATION

Explanation:
No data was found that met all conditions specified while performing an INSERT, DELETE, or UPDATE.

User Action:
No action is needed (informative message).

US QUERY-0020 UNABLE TO ACCESS ALTERNATE COLLATING SEQUENCE FILE

Explanation:
The format for the alternate collating sequence file is incorrect.

User Action:
Verify that the file is in the exact format as specified in Appendix C.

US QUERY-0021 QUERY BEGINS WITH INVALID FUNCTION

Explanation:
Query has detected an invalid clause as the first element of the statement.

User Action:
Check the statement and move the offending clause to its proper position.

US QUERY-0022 FIELD NAME OR LINE IDENTIFIER EXPECTED

Explanation:
An invalid element, character, etc., in the field list was found. Query-990 looks for a field list element such as a semicolon, field name, alias, line type, literal, or space character.

User Action:
Examine the field list at the indicated position.

US QUERY-0023 INVALID SYNTAX — UNABLE TO PROCESS

Explanation:
A meaningless character or word was encountered in the Query statement.

User Action:
Examine the statement where the exclamation point occurs for a possible typing error.

US QUERY-0024 DIGIT EXPECTED

Explanation:
A colon following a field name was not followed by a number specifying the output length.

User Action:
Follow the colon with a number specifying the output length or remove the colon.

US QUERY-0025 INVALID REPORT LINE ELEMENT IN UNFORMATTED OUTPUT

Explanation:
Requests were made for headers, spacing, output lengths, or embedded literals in unformatted output.

User Action:
Remove the syntax causing the error, as indicated by the exclamation point.

US QUERY-0026 NO REPORT LINES DEFINED

Explanation:
No report line was specified with the LIST function or the keyword LIST was omitted.

User Action:
Insert the necessary report lines after the keyword LIST.

US QUERY-0027 MULTIPLE REPORT LINES WITH NO ORDER GIVEN

Explanation:
Multiple report lines were specified and no BY clause was entered.

User Action:
Insert the required BY clause.

US QUERY-0028 NO FILE DEFINED WITH "FROM" CLAUSE

Explanation:
The FROM clause was not specified anywhere in the Query statement.

User Action:
Insert the required FROM clause.

US QUERY-0029 FILE IDENTIFIER MUST FOLLOW "FROM"

Explanation:
The file name was not specified after the keyword FROM.

User Action:
Specify the required file name.

US QUERY-0030 INCOMPLETE QUERY STATEMENT

Explanation:
A Query statement was processed but not all of the indicated clauses and/or elements were included.

User Action:
Complete the statement where the exclamation point is shown.

US QUERY-0031 "KEY" OR "LIST" MUST FOLLOW "BY"

Explanation:
Something other than BY KEY, BY LIST, or BY KEY BY LIST was specified.

User Action:
Correct the BY clause where indicated by the exclamation point.

US QUERY-0032 LISTING ORDER ALREADY SPECIFIED

Explanation:
More than one BY clause was specified in a Query statement.

User Action:
Remove the extra BY clause(s).

US QUERY-0033 UNEXPECTED CHARACTER IN CONDITION

Explanation:
Within a condition, an invalid character was specified.

User Action:
Correct the character where the exclamation point occurs.

US QUERY-0034 MULTIPLE FILES DEFINED IN MODIFICATION FUNCTION

Explanation:
Multiple FROM clauses were specified.

User Action:
Choose one FROM clause and eliminate the others.

US QUERY-0035 MISSING OPERAND

Explanation:
An operand was omitted in a simple or complex condition.

User Action:
Check each relational operator for two operands and each Boolean expression for two simple conditions.

US QUERY-0036 UNBALANCED PARENTHESES

Explanation:
A parenthesis is missing somewhere in the WHERE clause.

User Action:
Match all left parentheses with right parentheses.

US QUERY-0037 MISSING OPERATOR

Explanation:
A relational operator is missing in a condition.

User Action:
Insert the required relational operator.

US QUERY-0038 ALIAS LONGER THAN 30 CHARACTERS

Explanation:
A DBMS-990 or DD-990 element assumed to be an alias contained more than 30 characters.

User Action:
Specify the correct identifier.

US QUERY-0039 END OF REPORT LINE INDICATED WITH NO FIELDS SPECIFIED

Explanation:
A semicolon was encountered before any DBMS-990 or DD-990 elements were specified.

User Action:
A report line must specify at least one DBMS-990 or DD-990 element. Either specify one or more elements or make this report line into a report line heading or main heading.

W QUERY-0040 HEADING LITERAL PAST PAGE WIDTH — TRUNCATED HERE

Explanation:
This is a warning and does not prevent the Query statement from executing. A heading or footing literal cannot be longer than the page width specified.

User Action:
Break the literal into two or more literals, if necessary.

US QUERY-0041 ALL FIELDS IN REPORT LINE MUST COME FROM SAME LINE TYPE — CHECK ?1

Explanation:
A report line contains at least one field ?1 that is not part of the same line type.

User Action:
Check the report line containing the field or alias mentioned in the message, and verify that it is part of the same line type as the rest of the fields.

PAGE 8-10

US QUERY-0042 DBMS/DD ERROR IN COMPILE PHASE — STATUS ?1

Explanation:
Query-990 is unable to retrieve information about a field because of a data base or data manager error.

User Action:
Refer to Appendix A of the *Model 990 Computer Data Base Management System Programmer's Guide* or Section 8.5 of the *Model 990 Computer Data Dictionary User's Guide* for the exact meaning of this error code.

US QUERY-0043 UNABLE TO PROCESS FIELD — ?1

Explanation:
In this case, Query-990 was unable to retrieve information about the field.

User Action:
Check the spelling of the field name; check to see that the field name corresponds to the DDL name.

W QUERY-0044 CONDITIONS WITH NO QUANTIFIERS DEFAULT TO "ANY"

Explanation:
If a quantifier is specified in a simple condition, both operands should contain a quantifier. ANY is the default for the operand that does not contain the quantifier. Since this message is only a warning, the statement will be executed.

User Action:
Check the meaning of the statement that has ANY as the default. Add the appropriate quantifier where applicable unless ANY is desired as the default in all cases.

US QUERY-0045 CONDITION FIELDS MUST BE IN SAME LINE TYPE IF NO QUANTIFIERS USED

Explanation:
Operands must be of the same line type if you are comparing fields in a simple condition when not using quantifiers.

User Action:
Change the appropriate operands to correspond to the same line type.

US QUERY-0046 FIELDS COMPARED WITH DIFFERENT FORMATS, SEE FIELD ?1

Explanation:
In a simple condition, both operands were specified as fields but are of different lengths or formats.

User Action:
Ensure that both fields in a condition have the same format.

US QUERY-0047 QUANTIFIERS NOT ALLOWED WHEN ORDERED "BY LIST"

Explanation:
Record-level conditions are allowed only when output is ordered BY KEY or BY KEY BY LIST.

User Action:
Remove the quantifiers or change the order.

US QUERY-0048 LINE-LEVEL CONDITION NOT SAME LINE TYPE AS REPORT LINE

Explanation:
When the sequence is BY LIST or modification is being performed, line-level conditions must come from the same line type as the report or modification line.

User Action:
Check the DDL to see if the fields are in the same line.

US QUERY-0049 MAIN HEADINGS AND FOOTINGS LONGER THAN PAGE LENGTH

Explanation:
The total number of main headings and footings specified is longer than the specified page length.

User Action:
Increase the number of lines per page.

US QUERY-0050 TOTALING A NON-NUMERIC FIELD IS NOT ALLOWED — SEE FIELD ?1

Explanation:
Totaling is permitted only on numeric fields.

User Action:
Remove the totaling option from the field specified in the message.

US QUERY-0051 DATA TYPE NOT SUPPORTED BY QUERY — SEE FIELD ?1

Explanation:
One of the data types not supported was encountered.

User Action:
Verify that all fields used in the Query statement have supported data types. Use the DDL listings for verification.

US QUERY-0052 MORE THAN ONE RECORD LEVEL CONDITION IS SPECIFIED

Explanation:
Multiple record-level conditions were specified in the WHERE clause. You can specify only one record-level condition containing quantifiers.

User Action:
Retain the one record-level condition desired. Remove the other conditions with ANY or EVERY contained in them.

W QUERY-0053 REPORT LINE LONGER THAN MAXIMUM — TRUNCATED

Explanation:
After formatting, the total length of all elements of a report line exceeds 480 characters.

User Action:
Split the report line into two lines or include length parameters with fields that are longer than necessary.

US QUERY-0054 HIGH ORDER TRUNCATION OF NUMERIC CONSTANT ?1

Explanation:
A constant was specified that contained too many significant digits.

User Action:
Check all constants to see if they conform with the type of the field with which they are associated (as defined in the DDL).

US QUERY-0055 MORE THAN ONE LINE-LEVEL CONDITION SPECIFIED FOR LINE ?1

Explanation:
When all line-level conditions are specified in the main WHERE clause, only one line-level condition per line type is allowed.

User Action:
Respecify the Query statement with line-level conditions specified with the report line to which they apply, using separate individual WHERE clauses.

US QUERY-0056 CONDITION WITH NO ASSOCIATED REPORT LINE — LINE ?1

Explanation:
When line-level conditions were specified in the main WHERE clause, one of them used a line type that had no corresponding report line.

User Action:
Check all conditions for a corresponding report line.

US QUERY-0057 LINE LEVEL SORT NOT ALLOWED WITH "BY KEY"

Explanation:
Individual report lines come out in file order when the sequence is BY KEY and cannot be sorted.

User Action:
Change the sequence to BY KEY BY LIST.

US QUERY-0058 RECORD LEVEL SORT NOT ALLOWED WITH "BY LIST"

Explanation:
The BY LIST sequence does not use records in outputs, so output cannot be sorted on a record basis.

User Action:
Change the sequence to BY KEY BY LIST.

US QUERY-0059 SORT AND OUTPUT FIELDS TOO LARGE — MUST TOTAL <350

Explanation:
The sort key length plus the total output line length must be less than 350 characters.

User Action:
Reduce the sort keys or the output line size.

US QUERY-0060 INVALID SORT CLAUSE ELEMENT

Explanation:
An invalid word or symbol was included in a SORT clause. For instance, a line identifier was specified.

User Action:
Check the SORT clause syntax.

US QUERY-0061 NUMBER OR LITERAL EXPECTED IN CONTENTS CLAUSE

Explanation:
A number or literal should follow the equal sign in a CONTENTS clause.

User Action:
Check the CONTENTS clause syntax.

US QUERY-0062 " = " EXPECTED IN CONTENTS CLAUSE

Explanation:
A field name was specified in a CONTENTS clause but was not followed by an equal sign (=).

User Action:
Check the CONTENTS clause syntax.

US QUERY-0063 FIELD NAME EXPECTED IN CONTENTS CLAUSE

Explanation:
A field name should appear at this point in a CONTENTS clause.

User Action:
Check the CONTENTS clause syntax.

US QUERY-0064 SEMICOLON EXPECTED

Explanation:
A semicolon is expected at this point in the Query statement.

User Action:
Insert a semicolon.

US QUERY-0065 "CONTENTS" EXPECTED

Explanation:
You must specify a CONTENTS clause for each modification line when the function is INSERT or UPDATE.

User Action:
Specify the CONTENTS clause.

US QUERY-0066 ASCENDING OR DESCENDING INDICATOR EXPECTED

Explanation:
A colon was specified after a field name in a SORT clause but was not followed by A, D, ASCENDING, or DESCENDING.

User Action:
Include the appropriate keyword or do not specify the colon.

W QUERY-0067 PAGE COMMAND NOT ALLOWED IN MAIN HEADING OR FOOTING

Explanation:
A PAGE command in a main heading or footing would cause the processor to loop indefinitely, printing the main heading or footing. This message is a warning only; the Query executes but the PAGE is not performed.

User Action:
Move the page to the first report line heading or the last report line footing expected to be performed.

US QUERY-0068 INVALID TAB SETTING — MUST BE GREATER THAN PREVIOUS TAB

Explanation:
Tab settings in a report line must be specified in ascending order to avoid overwriting fields.

User Action:
Rearrange the report line elements so that tab settings are in ascending order.

US QUERY-0069 " = " EXPECTED WITH . . . "XXX" . . . FUNCTION

Explanation:
The condition contains a string surrounded by ellipses. You can perform this string operation only when the relational operator is EQ or =.

User Action:
Change the relational operator to EQ or =.

US QUERY-0070 INVALID FIELD FORMAT FOR TYPE OF COMPARE — SEE FIELD ?1

Explanation:
You attempted to perform string operations on noncharacter data.

User Action:
Do not use string operations on numeric data.

US QUERY-0071 INVALID TYPE SPECIFICATION ?1

Explanation:
You specified an invalid type in defining a format for a DEFINE variable.

User Action:
Check the DDL types for all DEFINE variables. CH is not allowed.

US QUERY-0072 ILLEGAL LENGTH ?1

Explanation:
You used an illegal length in defining a format for a DEFINE variable.

User Action:
Check the length to see if it matches the DDL type.

US QUERY-0073 CANNOT MIX COBOL WITH FORTRAN/PASCAL TYPES ?1

Explanation:
The result type of the expression does not match the type catagory of the fields used in the expression, or fields within the expression are in different type categories.

User Action:
Check all of the components of the expression to see if they fall in the same category.

US QUERY-0074 ILLEGAL TYPE FOR FIELD IN EXPRESSION ?1

Explanation:
A field was used in an expression that has a DDL format definition of CH.

User Action:
Do not use the field, or redefine the file to use numeric types.

US QUERY-0075 ERROR IN CONVERSION OF REAL OR INTEGER CONSTANT ?1

Explanation:
An error was produced on conversion of a constant associated with a field of type integer or real.

User Action:
Check all constants for too many digits or decimal points in integer constants.

US QUERY-0076 NEGATIVE SIGN USED IN UNSIGNED CONSTANT ?1

Explanation:
You cannot specify a negative constant when the DDL field type is AN or CN.

User Action:
Specify only unsigned constants.

US QUERY-0077 UNABLE TO INITIALIZE SORT/MERGE DUE TO INTERNAL QUERY ERRORS

Explanation:
When Query-990 attempted to initialize Sort/Merge, an abnormal termination occurred.

User Action:
Check to see that a current version of Sort/Merge is installed properly. If a new version has been installed since the last QGEN (Query Generation) has been performed, redo the QGEN and reinstall Query-990.

US QUERY-0078 UNABLE TO BID SORT/MERGE

Explanation:
Sorting was specified in Query but Query-990 could not bid Sort/Merge.

User Action:
Check the state of the system to see why the task cannot be bid. Check the system log to see if the Sort/Merge aborted.

US QUERY-0079 UNABLE TO INITIALIZE SORT/MERGE — NOT INSTALLED ON THE SYSTEM

Explanation:
Query-990 must have a Sort/Merge package installed in the system to perform sorting.

User Action:
Install Sort/Merge, then reinstall Query-990.

US QUERY-0080 UNABLE TO INITIALIZE SORT/MERGE — SORT/MERGE ABNORMAL TERMINATION

Explanation:
When Query-990 attempted to initialize Sort/Merge, an abnormal termination occurred.

User Action:
Check to see that a current version of Sort/Merge is installed properly. If a new version has been installed since the last QGEN (Query Generation) has been performed, redo the QGEN and reinstall Query-990.

US QUERY-0081 ERROR IN SENDING RECORD TO SORT/MERGE

Explanation:
Should not occur during normal operation of Query-990. An internal error condition has occurred.

User Action:
Report the problem to the customer support line.

US QUERY-0082 INTERTASK ERROR IN SENDING RECORD TO SORT/MERGE

Explanation:
Query-990 was unable to use intertask communication to send a record to Sort/Merge.

User Action:
Check that intertask was generated as greater than 500 bytes long. Reboot the system. Make sure all Sort/Merge patches have been applied.

US QUERY-0083 ERROR IN RECEIVING RECORD FROM SORT/MERGE

Explanation:
Should not occur during normal operation of Query-990. An internal error condition has occurred.

User Action:
Report the problem to the customer support line.

US QUERY-0084 INTERTASK ERROR IN RECEIVING RECORD FROM SORT/MERGE

Explanation:
Query-990 was unable to use intertask communication to receive a record from Sort/Merge.

User Action:
Check that intertask was generated as greater than 500 bytes long. Reboot the system. Make sure all Sort/Merge patches have been applied.

US QUERY-0085 FIELD IN CONTENTS CLAUSE MUST HAVE SAME FORMAT ?1

Explanation:
A field in the CONTENTS clause has been set equal to another field or define variable that has different format or length.

User Action:
Check to see that the formats and lengths of the fields are identical.

US QUERY-0086 RECORD TOTAL OR COUNT NOT ALLOWED WITH "BY LIST"

Explanation:
The BY LIST sequence does not use records in outputs, so fields cannot be counted or totaled on a record basis.

User Action:
Change the sequence to BY KEY BY LIST.

US QUERY-0087 CANNOT SORT GROUP OR LINE ?1

Explanation:
You cannot specify a group or line as a sort key.

User Action:
Specify all of the field components of the group or line.

US QUERY-0088 LINE-LEVEL CONDITION NOT ALLOWED IN MAIN "WHERE" CLAUSE

Explanation:
In a modification function, all line-level conditions should be specified with their corresponding modification lines.

User Action:
Check the syntax for modification lines. Move the condition to its correct location.

US QUERY-0089 COUNT ALLOWED ONLY ON DBMS-990 FIELDS ?1

Explanation:
The count operator must have a DBMS-990 field as its only operand.

User Action:
Change the expression to count a DBMS-990 field.

US QUERY-0090 DEFINE FIELD IN CONDITION MUST COME FROM SAME LINE TYPE — ?1

Explanation:
The define fields in a condition must come from the same line type.

User Action:
Verify that all the define fields in the condition come from the same line type.

US QUERY-0091 "BY" EXPECTED AFTER "LINKED"

Explanation:
The keyword LINKED in the FROM clause was followed by something other than the word BY.

User Action:
Correct the syntax of the LINKED BY clause.

US QUERY-0092 FILE NAME EXPECTED AFTER 'IN'

Explanation:
The IN clause was used after a field name to identify a file in which the field is located. The IN was not followed by a field name identifier.

User Action:
Check the spelling and syntax.

US QUERY-0093 " = " EXPECTED IN "LINKED BY" CLAUSE

Explanation:
An equal sign is missing between two linkage fields in the LINKED BY clause.

User Action:
Check the syntax; there should be no commas or semicolons.

US QUERY-0094 CANNOT USE GROUP NAME IN CONTENTS CLAUSE ?1

Explanation:
In a modification function, the user is attempting to assign a value to a group. This is not allowed since the members of a group may have different formats and types.

User Action:
Break the group into its component fields and assign a value to each separately.

US QUERY-0095 REPORT LINES MUST ACCESS SINGLE PATH IN MULTI-FILE

Explanation:
In a multiple-file Query, a single report line cannot be composed from fields that follow more than one access path between the files.

User Action:
Break the report line into several report lines that follow a single access path.

US QUERY-0096 CHARACTER FIELDS MUST BE USED WITH STRING OPERATORS

Explanation:
When a condition contains a string with ellipses (. . .'XXX'. . .), only CH fields may be compared to the string.

User Action:
Modify the condition to reflect a numeric comparison or change the DDL of the file to make the field a CH type.

US QUERY-0097 LINKED TO FIELD MUST BE PRIMARY OR SECONDARY KEY ?1

Explanation:
In the FROM clause of a multifile Query, the link field in the lower-level file must be a primary or secondary key and should be on the right side of the equal sign (=).

User Action:
Make sure the link fields are on the proper sides of the equal sign. Execute a CPYFIL on the file, change the DDL so that the link field is a secondary key, and execute a RLDFIL. (See the *Model 990 Computer DNOS Data Base Management System Programmer's Guide*.)

US QUERY-0098 TOP-LEVEL FILE MAY NOT BE LINKED TO

Explanation:
The first file following the keyword FROM is the top-level file. No lower-level file can link to a field in this file.

User Action:
Redefine the order of the files so that no lower-level files link to the top-level file.

US QUERY-0099 LOWER-LEVEL FILE DEFINED WITHOUT PATH TO TOP-LEVEL ?1

Explanation:
All files must have some access route defined to them from the top-level file, directly or indirectly. This means that some field in the top-level file must be linked to a key in the lower-level file or to a key of another lower-level file that links to this lower-level file.

User Action:
Define an access path to all lower-level files or do not include that file in this Query statement.

US QUERY-0100 CYCLE DETECTED IN LINKAGE ?1

Explanation:
In the LINKED BY clause, you have defined a sequence of linkages that loops back to a higher-level file.

User Action:
Determine which linkage makes a loop and omit it.

US QUERY-0101 LINK FROM FIELDS MUST COME FROM THE SAME FILE — ?1

Explanation:
When using a concatenated key in a LINKED BY, all fields must come from the same file.

User Action:
Alter the concatenated key to use only fields from the same file.

US QUERY-0102 LINKED FIELDS DO NOT HAVE SAME LENGTH — SEE ?1

Explanation:
The fields must have the same format, length, and number of decimal places to be linked in the LINKED BY clause.

User Action:
Check the DDL to see if the formats differ. Change the DDL to match the formats.

US QUERY-0103 FILE NAME EXPECTED AFTER "IN"

Explanation:
The IN clause was used after a field name to identify the file in which the field is located. The IN clause was not followed by a file name identifier.

User Action:
Check spelling and syntax.

US QUERY-0104 NO LINK FIELD DEFINED ?1

Explanation:
THRU was used to indicate a report line's access path. The field used in the THRU clause was not found among any of the link fields in the LINKED BY clause.

User Action:
Check spelling. Make sure all linkages have been specified.

US QUERY-0105 MULTIPLE FILES NOT ALLOWED WHEN SEQUENCE IS "BY LIST"

Explanation:
Multifile Queries require record association and cannot be sequenced BY LIST.

User Action:
Change the sequence to BY KEY or BY KEY BY LIST.

US QUERY-0106 NO LINK FILE DEFINED ?1

Explanation:
A THRU clause was used to indicate a report line's access path. The file name used after IN to indicate the origin of the link field was not found in the FROM clause.

User Action:
Check spelling. Make sure all file names have been defined.

US QUERY-0107 LINE IDENTIFIER EXPECTED IN UNIQUE CLAUSE

Explanation:
A valid line type indentifier must follow UNIQUE in the UNIQUE clause.

User Action:
Check the syntax and spelling.

US QUERY-0108 DEFINE FIELD IS NOT A LEGAL EXPRESSION

Explanation:
The define field flagged is not a legal expression.

User Action:
Correct the define field syntax.

US QUERY-0109 QUANTIFIERS NOT ALLOWED IN LINE-LEVEL CONDITION

Explanation:
Quantifiers are allowed only in record-level conditions and should not be used in conditions that are specified with a report line.

User Action:
If the condition is meant to be record-level, move it to the main WHERE clause; otherwise, remove the quantifiers.

US QUERY-0110 LINE ID OR "KEY" EXPECTED IN POSITIONING CLAUSE

Explanation:
The word AFTER or BEFORE was used in a modification function but was not followed by the word KEY or a line identifier.

User Action:
Check syntax and spelling.

US QUERY-0111 ATTEMPT TO SPECIFY MORE THAN ONE "FROM" CLAUSE

Explanation:
The user specified FROM more than once.

User Action:
Combine all files into one FROM clause. Eliminate extra FROM clauses.

US QUERY-0112 ATTEMPT TO SPECIFY MORE THAN ONE FUNCTION CLAUSE

Explanation:
Only one function keyword (LIST, INSERT, UPDATE, or DELETE) is allowed in a single Query statement.

User Action:
Decide which function you wish the Query statement to perform and eliminate all others.

US QUERY-0113 UNABLE TO ACCESS FILE ?1

Explanation:
Query was unable to access the file indicated.

User Action:
This message is followed by DBMS/DM ERROR IN COMPILE PHASE and the bad status received. Check the *Model 990 Computer DNOS Data Base Management System Programmer's Guide* or the *Model 990 Computer Data Dictionary User's Guide* for the exact meaning of the error code.

US QUERY-0114 ATTEMPT TO INSERT OR UPDATE PRIMARY KEY ONLY

Explanation:
Insert or update was specified without any fields named other than the primary key.

User Action:
Specify at least one other field name in the CONTENTS clause.

US QUERY-0120 DELETE NOT ALLOWED ON SEQUENTIAL FILES

Explanation:
It is illegal to delete from a sequential file.

User Action:
Ensure that no delete is attempted on any sequential files.

US QUERY-0121 POSITION CLAUSE NOT ALLOWED ON NON-DBMS FILES

Explanation:
It is illegal to indicate a position when inserting to a non-DBMS file. No BEFORE or AFTER position clauses are allowed.

User Action:
Remove the position clause from the Query statement.

US QUERY-0122 MUST NOT SPECIFY KEY VALUE FOR SEQUENTIAL FILE INSERT

Explanation:
Since all inserts to a sequential file occur at the end-of-file, it is meaningless to specify a key value.

User Action:
Remove the key value from the Query statement.

US QUERY-0123 LINE LEVEL CONDITION SPECIFIED IN RECORD LEVEL WHERE CLAUSE

Explanation:
Query has taken the WHERE clause that occurred after the FROM clause and made it a line-level condition.

User Action:
To make a true record-level condition, include the keyword ANY or EVERY before the conditional field or remove any nonkey fields from the condition.

US QUERY-0124 PARTIAL GROUP KEY SPECIFIED IN CONTENTS CLAUSE

Explanation:
Each field in a group key must be included when the key is used in a CONTENTS clause.

User Action:
Add the missing fields to the Query statement.

US QUERY-0125 'ON' OR 'BEFORE' SHOULD FOLLOW 'BREAK'

Explanation:
The keyword BREAK in a report line must be followed by the sequencing word ON or BEFORE.

User Action:
If the report line is a header line, use ON. If it is a total line, use BEFORE.

8.3 GUIDED QUERY ERRORS

Guided Query errors pertain to error conditions that occur while using the Guided Query utility. Guided Query errors appear at the bottom of the screen. In response, press either the NEW LINE/RETURN key or the HELP/CMD key. The NEW LINE/RETURN key positions the cursor at the origin of the error. The HELP/CMD key aborts the Guided Query. The Guided Query error messages are as follows:

US QUERY-0151 DOUBLE QUOTE MARKS MISSING OR MISMATCHED

Explanation:
A double quotation mark is missing for an alphabetic constant. This message applies to Steps 2 and 6.

User Action:
Ensure that all beginning quotation marks have a corresponding ending quotation mark.

US QUERY-0152 NOT A VALID LINE NAME ?1

Explanation:
The line name entered in Step 3 or 10 does not match any of the displayed line names.

User Action:
Locate the intended line name and enter it.

US QUERY-0153 INVALID FIELD # — ?1

Explanation:
The number displayed to the right of this message is not a valid number.

User Action:
Reenter the correct field number from the selections at the bottom of the screen.

US QUERY-0154 MARK YOUR OPTIONS WITH AN "X"

Explanation:
To choose LIST, COUNT, TOTAL, or AVERAGE options, you must mark the appropriate column with an X.

User Action:
Change other characters to X.

US QUERY-0155 AT LEAST ONE OPTION MUST BE MARKED

Explanation:
When a field is specified for the report, you must choose at least one option (such as, LIST, COUNT, TOTAL, or AVERAGE).

User Action:
Mark one option.

US QUERY-0156 CANNOT SPECIFY TOTAL OR AVERAGE ON CHARACTER FIELD

Explanation:
You can specify TOTAL or AVERAGE only on numeric fields.

User Action:
Delete the X under TOTAL or AVERAGE.

US QUERY-0157 NOT A VALID LINE TYPE ?1

Explanation:
The user has entered a line type identifier that is not legal. If an alias or long name is available for a line type, the alias or long name must be used rather than the DDL name.

User Action:
Check the displayed line identifiers and make sure your entry matches.

US QUERY-0158 INVALID FIELD NAME FOR OPERAND 1 ?1

Explanation:
The specified field name does not match any field name listed.

User Action:
Correct the operand field name to match a listed field name.

US QUERY-0159 NO QUANTIFIERS FOR OPERAND 2 CONSTANT

Explanation:
When entering record-level conditions, quantifiers are not meaningful for constants.

User Action:
Leave the quantifier field for operand 2 blank.

US QUERY-0160 OPERAND 1 AND OPERAND 2 MUST HAVE MATCHING FORMATS

Explanation:
Operands 1 and 2 must have the same format and length.

User Action:
Compare only matching operands.

US QUERY-0161 INVALID FIELD NAME FOR OPERAND 2 ?1

Explanation:
The specified field name does not match any field name listed.

User Action:
Correct the operand field name to match a listed field name.

US QUERY-0162 NOT A VALID FILE NAME ?1

Explanation:
The file name specified in Step 1 does not correspond to a valid file. The cause could be an incorrect file name, DBMS-990 or DD-990 down, or access privileges not granted.

User Action:
Locate the cause and take the appropriate action.

US QUERY-0163 FIRST OPERAND MUST BE PRECEDED BY "ANY" OR "EVERY"

Explanation:
Record test must have ANY or EVERY specified before the first operand.

User Action:
Specify ANY or EVERY or return to Step 9.

US QUERY-0164 ILLEGAL OPERATOR, PRESS FUNCTION KEY 5 FOR LIST OF LEGAL OPERATORS

Explanation:
Operator must be specified and be one of the six legal relational operators.

User Action:
Correct the operator.

US QUERY-0165 DBMS/DM DOWN

Explanation:
Data base or data manager is not running.

User Action:
Start data base or data manager.

US QUERY-0166 INVALID PASSWORD ?1

Explanation:
You do not have proper security access privileges for the file specified.

User Action:
Obtain a valid password from the DBA.

US QUERY-0167 ENTER "A" OR "D" ONLY

Explanation:
You have entered a bad response to the sort specification prompt.

User Action:
Enter the correct response.

US QUERY-0168 OPERAND MUST NOT BE A GROUP — SEE FIELD ?1

Explanation:
You may not use a group name in a conditional.

User Action:
Specify a complex condition which contains all the necessary group conditions.

US QUERY-0169 GROUPS MAY NOT BE USED IN CONDITIONS

Explanation:
Groups may not be used in conditions.

User Action:
Specify a complex condition which contains all the necessary group conditions.

8.4 INTERNAL MESSAGE CODES

If the error message file is not installed, the user receives internal message codes. Table 8-1 allows the user to look up the message ID and obtain the expanded text for the error message.

Table 8-1. Internal Message Codes

Internal Code	Message ID	Internal Code	Message ID	Internal Code	Message ID
>0001	0001	>002D	0045	>0056	0086
>0002	0002	>002E	0046	>0057	0087
>0003	0003	>002F	0047	>0058	0088
>0004	0004	>0030	0048	>0059	0089
>0005	0005	>0031	0049	>005B	0091
>0007	0007	>0032	0050	>005D	0093
>0008	0008	>0033	0051	>005E	0094
>0009	0009	>0034	0052	>005F	0095
>000A	0010	>0035	0053	>0061	0097
>000B	0011	>0036	0054	>0062	0098
>000C	0012	>0037	0055	>0063	0099
>000D	0013	>0038	0056	>0064	0100
>000E	0014	>0039	0057	>0066	0102
>0010	0016	>003A	0058	>0067	0103
>0011	0017	>003B	0059	>0068	0104
>0012	0018	>003C	0060	>0069	0105
>0013	0019	>003D	0061	>006A	0106
>0014	0020	>003E	0062	>006D	0109
>0016	0022	>003F	0063	>006E	0110
>0017	0023	>0040	0064	>006F	0111
>0018	0024	>0041	0065	>0070	0112
>0019	0025	>0042	0066	>0071	0113
>001A	0026	>0043	0067	>0072	0114
>001B	0027	>0044	0068	>0097	0151
>001C	0028	>0045	0069	>0098	0152
>001D	0029	>0046	0070	>0099	0153
>001E	0030	>0047	0071	>009A	0154
>001F	0031	>0048	0072	>009B	0155
>0020	0032	>0049	0073	>009C	0156
>0021	0033	>004A	0074	>009D	0157
>0022	0034	>004B	0075	>009E	0158
>0023	0035	>004C	0076	>009F	0159
>0024	0036	>004D	0077	>00A0	0160
>0025	0037	>004E	0078	>00A1	0161
>0026	0038	>004F	0079	>00A2	0162
>0027	0039	>0050	0080	>00A3	0163
>0028	0040	>0051	0081	>00A4	0164
>0029	0041	>0052	0082	>00A5	0165
>002A	0042	>0053	0083	>00A6	0166
>002B	0043	>0054	0084	>00A7	0167
>002C	0044	>0055	0085	>00A8	0168
				>00A9	0169

Appendix A

Query-990 Syntax

A.1 GENERAL

The Query-990 syntax consists of definitions using words, brackets, braces, and ellipses.

Each element of the language is defined by an equation-like rule. The entity being defined is written to the left of the symbol ::= and the definition is written to the right. The definition can be expressed in terms of the language elements that are defined separately. The following symbols are used in writing definitions:

Symbol	Description
::=	Used in writing definitions; means "is defined to be"
[ ]	Encloses entities that are optional
|	Indicates alternatives (e.g., A | B | C means A or B or C)
{ }	Encloses one or more entities from which you must select at least one
. . .	Indicates the position at which a previous item may be repeated as required
_	Underlined keywords and symbols must appear exactly as shown

For both brackets and braces, a choice is indicated by vertically stacking the possibilities. When brackets or braces enclose a portion of a definition but only one possibility is shown, the brackets or braces delimit the portion of the format to which a following ellipsis applies.

Underlined uppercase words are keywords. Lowercase words are symbols representing a language element defined elsewhere in the syntax definition.

A.2 SYNTAX DEFINITION

The syntax definition for Query-990 is as follows:

alias ::= longname

alpha ::= A | B | C | . . . | Z

alphanum ::= { alpha | digit }

ascii-character ::= { alpha | digit | <space> | ! | " | # | $ | % | & | , | (|) | * | + | ' | - | . | / | : | ; | < | = | > | ? | @ | [|] | ^ | _ }

BREAK-clause ::= BREAK { ON | BEFORE } field-type [field-type]...

BY-clause ::= { BY KEY BY LIST | BY KEY | BY LIST }

change data ::= ? [: { U, | F, }] [length-change] [(change-offset)]

change-offset ::= unsigned-integer

complex-condition ::= [( ] simple-condition [log-op complex-condition] [ )]

concat-field ::= field-type [^ concat-field]

condition ::= { simple-condition | complex-condition }

conditional-literal ::= [...] string [...]

constant ::= { string | number }

CONTENTS-clause ::= CONTENTS { field-type / key-type } = { constant / variable / change-data }
[[,] { field-type / key-type } = { constant / variable / change-data }] ...

DEFINE-clause ::= DEFINE [variable : type = define-expression [;]] ...

define-expression ::= [(] subexpression [operator define-expression] [)]

DELETE-clause ::= DELETE [trace-indicator] { RECORD / delete-line [; delete-line]... } [;]

delete-line ::= line type [WHERE-clause]

digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 8 | 9

field-name ::= { id / alias } [IN file-name]

file-name ::= { id / alias }

fractional-part ::= . digit [digit]...

FROM-clause ::= FROM file-name [EXCLUSIVE] [file-name [EXCLUSIVE]] ...
[LINKED-BY-clause] [SORT-clause] [HEADER-clause]
[BY-clause] [UNIQUE-clause]

function-clause ::= { LIST-clause | INSERT-clause | UPDATE-clause | DELETE-clause }

HEADER-clause ::= [NO HEADER] [HEADER [header-element]] [FOOTING [header-element]]

HEADER-element ::= { PAGE [digit] [digit] | SKIP [digit] [digit] | DEFAULT | header-literal }

header-literal ::= " [literal-element]. . . " { " [literal-element]. . . " | ' [literal-element]. . . ' }

id ::= { alpha | alpha alphanum | alpha alphanum alphanum | alpha alphanum alphanum alphanum }

INSERT-clause ::= INSERT [trace-indicator] insert-line [; insert-line]. . .[;]

insert-line ::= line-type [position-clause] CONTENTS-clause [WHERE-clause]

integer ::= { + | - } digit [digit]. . .

key-type ::= { id | alias } [IN file-type]

length ::= 1 | 2 | 3 | 4 |. . .| 256

length-change ::= unsigned-integer

line-type :: = { digit digit | alpha alpha | alias } [IN file-type]

LINKED-BY-clause :: = LINKED BY { field-type | concat-field } = key-type
[, { field-type | concat-field } = key-type] ...

LIST-clause :: = LIST report-line [; report-line]. . .[;]

literal-element :: = { ascii-character | ΛSYSTIME | ΛSYSDATE | ΛPAGENUM }

log-op :: = { OR | AND }

longname :: = alpha [{ alphanum | $ | - | _ }] ...

modification-line :: = line-type [position-clause] CONTENTS-clause [WHERE-clause]

number :: = { integer | integer fractional-part | + fractional-part }

op1 :: = { field-type | key-type | variable }

op2 :: = { op1 | conditional-literal | number | change-data }

operator ::= + | - | * | /

option ::= [: length] [TOTAL [ONLY]] [AVERAGE [ONLY]] [COUNT [ONLY]]

order-indicator ::= { : A | : ASCENDING | : D | : DESCENDING }

position-clause ::= { BEFORE | AFTER } { FIRST line-type [WHERE-clause] | KEY }

quantifier ::= { ANY | EVERY }

query ::= [DEFINE-clause] function-clause FROM-clause [WHERE-clause]

rel-op ::= EQ | GT | LT | NE | LE | GE | = | < | > | <> | <= | >=

report-line ::= report-line-element [[,] report-line-element ...]
[HEADER-clause] [WHERE-clause] [SORT-clause]

report-line-element ::= { field-type [option] [THRU-clause] | line-type [option] | key-type [option] | variable [option] | BREAK-clause | X length | SPACE length | TAB digit [digit] | string | change-data }

simple-condition ::= [(] [quantifier] op1 rel-op [quantifier] op2 [)]

SORT-clause ::= SORTED BY { field-type / key-type / variable } [order-indicator]
[{ field-type / key-type / variable } [order-indicator]] ...

string ::= " [ascii-character]... "

subexpression ::= [( ] { [RECORD] TOTAL field-type / [RECORD] COUNT field-type / field-type / variable / change-data } [ )]

THRU-clause ::= THRU file-type

trace-indicator ::= TRACE { ONLY / OFF }

type ::= (see Appendix B)

UNIQUE-clause ::= UNIQUE line-type [line-type]...

unsigned-integer ::= digit [digit]...

UPDATE-clause ::= UPDATE [trace-indicator] modification-line
[; modification-line]... [;]

variable ::= longname

WHERE-clause ::= WHERE condition

Appendix B

Calculation Data Types

Code	Description	Example	Formats
AN	Arithmetic without sign. Decimal places are allowed. User zero for no decimal places.	AN/8.2 COBOL: FORTRAN: Pascal:	 PIC 9(6)V9(2) COMP. <none> <none>
AS	Arithmetic signed. Length (n) must include sign, and decimal places are allowed. Use zero for no decimal places.	AS/8.2 COBOL: FORTRAN: Pascal:	 PIC S9(5)V9(2) COMP. <none> <none>
CH	Character string. Length includes total characters. Decimal places not allowed.	CH/20 COBOL: FORTRAN: Pascal:	 PIC X(20). <A format> PACKED ARRAY [1. . .20] OF CHAR;
CN	Character numeric. Decimal places are allowed. Use zero for no decimal places.	CN/6.2 COBOL: FORTRAN: Pascal:	 PIC X(20). <none> <none>
CS	Character numeric signed. Length (n) must include the sign. Decimal places are allowed. Use zero for no decimal places.	CS/8.5 COBOL: FORTRAN: Pascal:	 PIC S9(2)V9(5) <none> <none>
ID	Double precision integer. Contained in two 16-bit words and may be signed. Length (n) default is 4; if specified, it must be 4.	ID/4 COBOL: FORTRAN: Pascal:	 <none> INTEGER*4 LOGINT;

Code	Description	Example	Formats
IS	Single-precision integer. Contained in one 16-bit word. Length (n) default is 2; if specified, it must be 2. Field may contain a sign.	IS/2 COBOL: FORTRAN: Pascal:	 PIC 9(5) COMP-1. INTEGER*2 INTEGER;
LG	Logical variable. Length (n) default is 2; if specified, it must be 2.	LG/2 COBOL: FORTRAN: Pascal:	 <none> LOGICAL BOOLEAN
PK	Packed decimal. Digit length (n) must be even and includes the sign. Decimal places are allowed, and zero indicates no decimal places. Contained in n/2 bytes.	PK/6.2 COBOL: FORTRAN: Pascal:	 PIC S9(3)V9(2) COMP-3. <none> <none>
RD	Double-precision real. Contained in four 16-bit words and may be signed. Length (n) default is 8; if specified, it must be 8.	RD/8 COBOL: FORTRAN: Pascal:	 <none> REAL*8 REAL(16)
RS	Single-precision real. Contained in two 16-bit words and may be signed. Length (n) default is 4; if specified, it must be 4.	RS/4 COBOL: FORTRAN: Pascal:	 <none> REAL*4 REAL

Appendix C

Alternate Collating Sequences

Query-990 supports collating sequences other than the standard USASCII collating sequence. The current release of Query-990 includes two alternate collating sequence files. These are the collating sequences for the Germany/Austria and Sweden/Finland character sets. These collating sequences are compatible with the operating systems' international support for key indexed files (KIFs).

The format for the collating sequence file has been strictly defined. It contains eight records, and each record contains eight pairs of characters, left justified. Alternate character pairs must be defined for all of the characters in the set (hexadecimal 40 through 7F).

The file S$QUERY.ALTSEQGA contains the collating sequence for the Germany/Austria character set; the file S$QUERY.ALTSEQSF contains the collating sequence for the Sweden/Finland character set. The following is an example of the collating sequence for the Germany/Austria character set:

```
40404141 5B424243 43444445 45464647
47484849 484A4A4B 4B4C4C4D 4D4E4E4F
4F505C51 50525153 52545355 54565557
5D585659 575A585B 595C5A5D 5E5E5F5F
606061617B626263636464656566666 7
67686869696A6A6B6B6C6C6D6D6E6E6F
6F707C71707271737274737 57E767477
75787D79767A777B7B7C797D7A7E7F7F
```

GERMAN/AUSTRIAN
TO U.S. ASCII
ALTERNATE COLLATING
SEQUENCE TABLE

You can use the right half of each record for comments.

You can select the internationalization option at Query-990 generation. If you do select this option, you must specify the file name containing the desired collating sequence. The QGEN processor then builds the correct SCI QUERY command procedure.

To specify alternate collating sequencing to the Query processor, use the bid parameters in the QUERY command procedure. The seventeenth argument indicates the file name containing the desired collating sequence. If you do not specify a parameter for this argument, the USASCII collating sequence is used.

The following is an example of the bid parameters for the SCI procedure QUERY. These parameters specify the collating sequence contained in the file S$QUERY.ALTSEQGA:

```
 BID TASK = OCO, LUNO = @S$QUERY.PROG

PARMS = (2000, 3000, @$MR$, &PASSWORD, @$Q$LPP,
@$Q$CPL, @$Q$OUT, @$Q$FMT,
@$Q$NEW, @$Q$EDT, @$Q$LST, NO, @$Q$SAVE, @$Q$VCF,
"@&INPUT STATEMENT PATHNAME",
@LISTING, S$QUERY.ALTSEQGA,
S$QUERY.ERRMSG,
"@&OUTPUT STATEMENT PATHNAME", @$MT$)
```

To specify a different collating sequence, change the seventeenth argument in the bid parameters for the SCI Query command procedure.

To use an alternate collating sequence for the Query output from the Guided Query utility, modify the SCI procedure GQUERY2 to contain the file name for the desired collating sequence.

Appendix D

DDL Listings for Example Files

The following DDL listings are for data base example files.

Item File

```
FILE=ITEM,LINES=64
ID=ITMN=CH/4,VOL=50,ACCESS=RANDOM/1
*
LINE=01
  FIELD=DESC=CH/20
  FIELD=UPRC=CN/6.3
  FIELD=QTYO=CN/4.0
  FIELD=QTYH=CN/4.0
  ENDL
END.
```

Customer File

```
FILE=CUST,LINES=57
ID=CUSN=CH/5,VOL=50,ACCESS=RANDOM/1
*
LINE=01
  FIELD=NAME=CH/20
  GROUP=ADDR
    FIELD=STRT=CH/20
    FIELD=CITY=CH/20
    FIELD=STAT=CH/2
    FIELD=ZIPC=CH/5
    ENDG
  FIELD=CRED=CH/20
  ENDL
END.
```

PAY1 File

```
FILE=PAY1,LINES=300
ID=MNUM=CN/6.0,VOL=30,ACCESS=RANDOM/1
*
LINE=01
  GROUP=ADDR
    FIELD=MNAM=CH/20
    FIELD=MSTR=CH/20
    FIELD=MCTY=CH/15
    FIELD=MSTT=CH/2
    FIELD=MZIP=CN/5.0
    ENDG
  FIELD=MSSN=CN/9.0
  ENDL
*
LINE=CU
  FIELD=MJOB=CH/10
  FIELD=MLOC=CH/10
  FIELD=MDEP=CH/15
  FIELD=MTMR=CH/1
  FIELD=MTES=CN/2.0
  FIELD=MTEX=CN/2.0
  ENDL
*
LINE=CR
  FIELD=MDDT=CN/5.2
  FIELD=MPYP=CN/2.0
  FIELD=MRAT=CN/7.2
  FIELD=MCOM=CN/3.3
  FIELD=MSLS=CS/11.3
  ENDL
*
LINE=ED
  FIELD=DEGR=CH/3
  FIELD=YEAR=CN/4.0
  FIELD=COLL=CH/20
  FIELD=GPA =CN/2.1
  ENDL
*
LINE=PE
  FIELD=JOBT=CH/20
  FIELD=COMP=CH/25
  FIELD=STAT=CH/2
  FIELD=PSAL=CN/7.2
  ENDL
*
LINE=PP
  FIELD=PLOC=CH/10
  FIELD=PDEP=CH/15
  FIELD=PJOB=CH/10
  ENDL
*
 SECONDARY-REFERENCES
 MSSN=VOL=30,ACCESS=RANDOM/1
 END.
```

Sales Order File

```
FILE=SOFL,LINES=344
ID=SONM=CH/6,VOL=50,ACCESS=RANDOM/1
*
LINE=BL
  FIELD=BILL=CH/5
  FIELD=LOCK=CH/2
  ENDL
*
LINE=02
  FIELD=SHIP=CH/5
  ENDL
*
LINE=03
  FIELD=ITEM=CH/4
  FIELD=QUAN=CN/4.0
  ENDL
*
SECONDARY-REFERENCES
BILL=VOL=50,ACCESS=RANDOM/1
SHIP=VOL=50,ACCESS=RANDOM/1
ITEM=VOL=200,ACCESS=RANDOM/1
END.
```

The following DDL listings are for KIF example files.

Item File

```
FILE=ITEM,TYPE=KIF
*
ID=ITMN=CH/4,DUP=Y,MOD=N
*
LINE=01
FIELD=ITMN=CH/4
FIELD=DESC=CH/20
FIELD=UPRC=CN/6.3
FIELD=QTYO=CN/4.0
FIELD=QTYH=CN/4.0
ENDL
*
END.
```

Customer File

```
FILE=CUST,TYPE=KIF
*
ID=CUSN=CH/5,DUP=Y,MOD=N
*
LINE=01
FIELD=CUSN=CH/5
FIELD=NAME=CH/20
GROUP=ADRS
FIELD=STRT=CH/20
FIELD=CITY=CH/20
FIELD=STAT=CH/2
FIELD=ZIPC=CH/5
ENDG
FIELD=CRED=CH/20
ENDL
*
END.
```

PAY1 File

```
FILE=PAY1,TYPE=KIF,TAG
*
ID=MNUM=CN/6.0,DUP=Y,MOD=N
*
LINE=01
FIELD=TAG1=IS/2,VALUE=1
FIELD=MNUM=CN/6.0
GROUP=ADDR
FIELD=MNAM=CH/20
FIELD=MSTR=CH/20
FIELD=MCTY=CH/15
FIELD=MSTT=CH/2
FIELD=MZIP=CN/5.0
ENDG
FIELD=MSSN=CN/9.0
ENDL
*
LINE=CU
FIELD=TAG2=IS/2,VALUE=2
FIELD=MNUM=CN/6.0
FIELD=MJOB=CH/10
FIELD=MLOC=CH/10
FIELD=MDEP=CH/15
FIELD=MTMR=CH/1
FIELD=MTES=CN/2.0
FIELD=MTEX=CN/2.0
ENDL
```

```
*
LINE=CR
FIELD=TAG3=IS/2,VALUE=3
FIELD=MNUM=CN/6.0
FIELD=MDDT=CN/5.2
FIELD=MPYP=CN/2.0
FIELD=MRAT=CN/7.2
FIELD=MCOM=CN/3.3
FIELD=MSLS=CS/11.3
ENDL
*
LINE=ED
FIELD=TAG4=IS/2,VALUE=4
FIELD=MNUM=CN/6.0
FIELD=DEGR=CH/3
FIELD=YEAR=CN/4.0
FIELD=COLL=CH/20
FIELD=GPA=CN/2.1
ENDL
*
LINE=PE
FIELD=TAG5=IS/2,VALUE=5
FIELD=MNUM=CN/6.0
FIELD=JOBT=CH/20
FIELD=COMP=CH/25
FIELD=STAT=CH/2
FIELD=PSAL=CN/7.2
ENDL
*
LINE=PP
FIELD=TAG6=IS/2,VALUE=6
FIELD=MNUM=CN/6.0
FIELD=PLOC=CH/10
FIELD=PDEP=CH/15
FIELD=PJOB=CH/10
ENDL
*
SECONDARY-REFERENCES
MSSN,DUP=Y,MOD=Y
END.
```

Sales Order File

```
FILE=SOFL,TYPE=KIF,TAG
*
ID=SONM=CH/6,DUP=Y,MOD=N
*
LINE=BL
FIELD=TAG1=IS/2,VALUE=1
FIELD=SONM=CH/6
FIELD=BILL=CH/5
FIELD=LOCK=CH/2
ENDL
*
LINE=02
FIELD=TAG2=IS/2,VALUE=2
FIELD=SONM=CH/6
FIELD=SHIP=CH/5
ENDL
*
LINE=03
FIELD=TAG3=IS/2,VALUE=3
FIELD=SONM=CH/6
FIELD=SITM=CH/4
FIELD=QUAN=CN/4.0
ENDL
SECONDARY-REFERENCES
BILL,DUP=Y,MOD=Y
SHIP,DUP=Y,MOD=Y
SITM,DUP=Y,MOD=Y
END.
```

Appendix E

Example Query Application

E.1 GENERAL

This appendix describes a simple way to produce interactive data-retrieval applications for DBMS-990 and Query-990 users. The application described displays a predefined report based on variable selection criteria. Some examples are as follows:

- A personnel report lists an employee's name, department, job title, mailing address, social security number, and previous work experience after a user enters the employee number.
- A sales organization report displays current credit information for a specified customer.
- A report for an airline ticket agent shows the remaining seats on a specified flight, or whether a certain passenger's reservation is still active and the corresponding flight number.

The example in this appendix uses the Query and SCI languages. For information on the SCI language, consult the *Model 990 Computer DNOS System Command Interpreter (SCI) Reference Manual.*

E.2 CREATING THE PROCEDURE

To generate the appropriate report, first develop an SCI procedure to prompt the operator for the selection criteria (for example, employee number, customer number, or flight number). Then, within the same procedure, use the SCI primitive .DATA to build the Query statement. The following Query displays information for the employee whose number is entered by the operator.

```
LIST "MASTER EMPLOYEE NUMBER =" MNUM  "     SSN# = " MSSN;
 MNAM MSTR MCTY MSTT MZIP HEADER SKIP
 "NAME                    ADDRESS";
 MJOB MLOC MDEP  HEADER SKIP "CURRENT JOB"
 "TITLE      LOCATION    DEPARTMENT";
 PE HEADER SKIP "PREVIOUS JOBS"
 "TITLE                   COMPANY                    STATE SALARY";
NO HEADER HEADER "**** MASTER EMPLOYEE INFORMATION ****" SKIP
FROM PAY1

WHERE MNUM EQ '@$EMPNO'
BY KEY BY LIST
```

The employee number entered is substituted for '@$EMPNO' by using the SUBSTITUTION option of the SCI primitive .DATA. Because the SUBSTITUTION option removes double quotes, thereby invalidating the report heading definitions, divide the Query at the point shown by the line of asterisks. This line separates the fixed part, which contains all of the double quotes, from the variable part, which contains the selection criteria (employee number).

Next, execute the Query using the .BID primitive; finally, display the results using the .SHOW primitive.

The following steps build an example employee information report using the data base file PAY1. You can follow these steps, with appropriate modifications, to create similar reports.

1. Choose a name for your SCI command. This one is called EMPINFO and displays the following prompts:

```
QUERY ON EMPLOYEE INFO
                    PASSWORD:
ENTER EMPLOYEE NUMBER:
  VIEW GENERATED QUERY:
```

   The password is required only if security is installed on your system. The operator enters the employee number, which is then inserted into the text of a Query statement. The operator can choose whether or not to view the generated report.

2. Use the Text Editor to create an SCI language procedure to display the desired prompts. The following lines are needed for EMPINFO:

```
EMPINFO (QUERY ON EMPLOYEE INFO),
PASSWORD = STRING,
ENTER EMPLOYEE NUMBER = INT,
VIEW GENERATED QUERY = YESNO
.SYN $EMPNO = "&ENTER EMPLOYEE NUMBER"
```

   The .SYN primitive assigns the employee number entered to the synonym $EMPNO.

3. Enter the first part of the Query in a .DATA primitive without any options. Send the output of .DATA (the Query statement) to a file, such as .QSRC@$$ST in the example. It is good practice to use a file name of six or fewer characters with the synonym @$$ST concatenated to it. $$ST is the operator's station number. Using a file name with @$$ST appended to it allows all stations to use the command simultaneously. The first .DATA primitive for EMPINFO is as follows:

```
*
* Build the Query, beginning with the fixed portion
*
.DATA .QSRC@$$ST
LIST "MASTER EMPLOYEE NUMBER =" MNUM  "     SSN# = " MSSN;
 MNAM MSTR MCTY MSTT MZIP HEADER SKIP
 "NAME                    ADDRESS";
 MJOB MLOC MDEP  HEADER SKIP "CURRENT JOB"
 "TITLE      LOCATION    DEPARTMENT";
 PE HEADER SKIP "PREVIOUS JOBS"
 "TITLE                   COMPANY                    STATE SALARY";
NO HEADER HEADER " **** MASTER EMPLOYEE INFORMATION ****" SKIP
FROM PAY1
.EOD
```

4. Next, define a second .DATA primitive with the EXTEND and SUBSTITUTION options set to YES, as follows:

```
*
* Append the variable portion, inserting the
* employee number into the condition
*
.DATA .QSRC@$$ST, EXTEND = YES, SUBSTITUTION = YES
WHERE MNUM EQ '@$EMPNO'
BY KEY BY LIST
.EOD
```

The EXTEND option appends this second .DATA primitive to the previous one, completing the Query statement.

5. Add the following to execute the Query and display the listing:

```
*
* Bid Query
*
.BID TASK=OCO, LUNO=S$QUERY.PROG,
*
PARMS=(2000, 3000, @$MR$, &PASSWORD, 60, 80,
.LISTIN@$$ST, R, NO, NO, &VIEW GENERATED QUERY,
NO, NO, ,.QSRC@$$ST, .LISTIN@$$ST, , , ,@$MT$)
*
* Display the listing file
*
.SHOW .LISTIN@$$ST
*
* Delete the employee number synonym and the listing file
*
.SYN $EMPNO=""
DF P=.LISTIN@$$ST
```

Depending on the application, you might want to make the following changes to the .BID parameters:

a. Change the file name .QSRC@$$ST; be sure to match the file name used in the .DATA sections.

b. Replace the string &VIEW GENERATED QUERY with NO if you choose not to use this prompt. (Also, remove VIEW GENERATED QUERY from the list of prompts.)

c. To direct the output to a printer or file, replace the pathname .LISTIN@$$ST with the printer or file name and delete the .SHOW command.

6. Store the file containing your command procedure under the directory .S$CMDS. The EMPINFO command procedure file would have the pathname .S$CMDS.EMPINFO. The complete command procedure is as follows:

```
EMPINFO (QUERY ON EMPLOYEE INFO),
PASSWORD = STRING,
ENTER EMPLOYEE NUMBER = INIT,
VIEW GENERATED QUERY = YESNO
.SYN $EMPNO = "&ENTER EMPLOYEE NUMBER"
*
* Build the query, beginning with the fixed portion
*
.DATA .QSRC@$$ST
LIST "MASTER EMPLOYEE NUMBER =" MNUM  "   SSN# = " MSSN;
MNAM MSTR MCTY MSTT MZIP HEADER SKIP
"NAME                    ADDRESS";
MJOB MLOC MDEP  HEADER SKIP "CURRENT JOB"
"TITLE    .LOCATION     DEPARTMENT";
PE HEADER SKIP "PREVIOUS JOBS"
"TITLE                   COMPANY                  STATE SALARY";
NO HEADER HEADER " **** MASTER EMPLOYEE INFORMATION ****" SKIP
FROM PAY1
.EOD
*****************************************************************
*
* Append the variable portion, inserting the
* employee number into the condition
*
.DATA .QSRC@$$ST, EXTEND = YES, SUBSTITUTION = YES
WHERE MNUM EQ '@$EMPNO'
BY KEY BY LIST
.EOD
*
* Bid Query
*
.BID TASK=OCO, LUNO=S$QUERY.PROG,
*
PARMS=(2000, 3000, @$MR$, &PASSWORD, 60, 80,
.LISTIN@$$ST, R, NO, NO, &VIEW GENERATED QUERY,
NO, NO, ,.QSRC@$$ST, .LISTIN@$$ST, , , ,@$MT$)
*
* Display the listing file
*
.SHOW .LISTIN@$$ST
*
* Delete the employee number synonym and the listing file
*
.SYN $EMPNO=""
DF P=.LISTIN@$$ST
```

7. Enter the name of the command (EMPINFO) to execute the procedure. The following prompts appear on the screen:

```
QUERY ON EMPLOYEE INFO
                    PASSWORD:
      ENTER EMPLOYEE NUMBER:
        VIEW GENERATED QUERY:
```

The operator responds as follows:

```
QUERY ON EMPLOYEE INFO
                    PASSWORD:  ZZZZ
      ENTER EMPLOYEE NUMBER:   55555
        VIEW GENERATED QUERY:  YES
```

As a result, the following appears on the screen:

```
 DNQUERY      1.3.0  82.167   QUERY-990    06/07/82  13:32:11 PAGE     1
LIST "MASTER EMPLOYEE NUMBER =" MNUM  "    SSN# = " MSSN;
MNAM MSTR MCTY MSTT MZIP HEADER SKIP
"NAME                    ADDRESS";
MJOB MLOC MDEP  HEADER SKIP "CURRENT JOB"
"TITLE      LOCATION    DEPARTMENT";
PE HEADER SKIP "PREVIOUS JOBS"
"TITLE                   COMPANY                  STATE  SALARY";
NO HEADER HEADER " **** MASTER EMPLOYEE INFORMATION ****" SKIP
FROM PAY1
WHERE MNUM EQ '55555'
BY KEY BY LIST


 **** MASTER EMPLOYEE INFORMATION ****

MASTER EMPLOYEE NUMBER = 55555    SSN# = 875247964

NAME                    ADDRESS
PASCHAL, JIMMY          1000 ACORN OAKS       LIBERTY HILL     MO  79666

CURRENT JOB
TITLE       LOCATION    DEPARTMENT
VICE PRES   MT. VIEW    SALES

PREVIOUS JOBS
TITLE                   COMPANY                  STATE  SALARY
SALESMAN                EQUIPMENT MFG.             CA   1500.00
```

Alphabetical Index

Introduction

HOW TO USE INDEX

The index, table of contents, list of illustrations, and list of tables are used in conjunction to obtain the location of the desired subject. Once the subject or topic has been located in the index, use the appropriate paragraph number, figure number, or table number to obtain the corresponding page number from the table of contents, list of illustrations, or list of tables.

INDEX ENTRIES

The following index lists key words and concepts from the subject material of the manual together with the area(s) in the manual that supply major coverage of the listed concept. The numbers along the right side of the listing reference the following manual areas:

- Sections — Reference to Sections of the manual appear as "Sections x" with the symbol x representing any numeric quantity.

- Appendixes — Reference to Appendixes of the manual appear as "Appendix y" with the symbol y representing any capital letter.

- Paragraphs — Reference to paragraphs of the manual appear as a series of alphanumeric or numeric characters punctuated with decimal points. Only the first character of the string may be a letter; all subsequent characters are numbers. The first character refers to the section or appendix of the manual in which the paragraph may be found.

- Tables — References to tables in the manual are represented by the capital letter T followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the table). The second character is followed by a dash (-) and a number.

  Tx-yy

- Figures — References to figures in the manual are represented by the capital letter F followed immediately by another alphanumeric character (representing the section or appendix of the manual containing the figure). The second character is followed by a dash (-) and a number.

  Fx-yy

- Other entries in the Index — References to other entries in the index preceded by the word "See" followed by the referenced entry.

CUT ALONG LINE

USER'S RESPONSE SHEET

Manual Title: Query-990 User's Guide (2276554-9701)

Manual Date: 15 July 1982 **Date of This Letter:** _____

User's Name: _____ **Telephone:** _____

Company: _____ **Office/Department:** _____

Street Address: _____

City/State/Zip Code: _____

Please list any discrepancy found in this manual by page, paragraph, figure, or table number in the following space. If there are any other suggestions that you wish to make, feel free to include them. Thank you.

Location in Manual	Comment/Suggestion
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____
_____	_____
	_____
	_____
	_____

NO POSTAGE NECESSARY IF MAILED IN U.S.A.
FOLD ON TWO LINES (LOCATED ON REVERSE SIDE), TAPE AND MAIL

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7284 DALLAS, TX

POSTAGE WILL BE PAID BY ADDRESSEE

TEXAS INSTRUMENTS INCORPORATED
DIGITAL SYSTEMS GROUP

ATTN: TECHNICAL PUBLICATIONS
P.O. Box 2909 M/S 2146
Austin, Texas 78769

FOLD

TEXAS INSTRUMENTS
INCORPORATED

DIGITAL SYSTEMS GROUP
P.O. BOX 2909 • AUSTIN, TEXAS 78769