1988

An alternative language interface for the mistress relational database patterned after IBM's query-by-example

Susan C. Vogel

Rochester Institute of Technology
School of Computer Science and Technology

An Alternative Language Interface For the Mistress Relational Database Patterned After IBM's Query-by-Example

by
Susan C. Vogel

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by: Jeffrey A. Lasky
Professor Jeffrey A. Lasky (chair)

Henry A. Etlinger
Professor Henry A. Elinger

Chris Comte
Professor Chris Comte

April 26, 1988

Title of Thesis: An Alternative Language Interface
for the Mistress Relational Database
Patterned after IBM's Query-by-Example

I, Susan C. Vogel hereby grant permission to the Wallace Memorial Library, of RIT, to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: 5/3/88

Abstract

This thesis effort developed a user-oriented query language interface, patterned after IBM's Query-by-Example, for the Mistress relational database. The interface, Mistress/QBE, is written entirely in C and uses the UNIX *curses* library of subroutines to allow full screen input and output. Mistress/QBE allows the user to issue commands to draw pictorial representations of tables which exist in the database. The user then enters values and operators into the tables to specify a query by indicating attributes to be used in conditional selections, sort and grouping orders, and output formats. Mistress/QBE decodes the information entered on the screen and formulates a Mistress Query Language command which is passed to the Mistress standard C language interface for execution. With a few minor exceptions, any query which can be written in the Mistress Query language can also be written in Mistress/QBE. The interface also includes a high-level operator, called grouping, which is supported by IBM's QBE but not by native Mistress.

Key Words and Phrases

QBE, Query-by-Example, Mistress

Computing Review Subject Codes

H.2.3. - Information Systems, Database Management, Languages

Table of Contents

1. Introduction and Background

The objective of this thesis effort was to design and implement a QBE-like interface to the Mistress relational database system.

1.1. Previous Work

1.1.1. Initial Work Done At IBM

For many years there has been a trend to develop database query languages which can be easily learned and used by non-professionals with little or no computer background. Query-by-Example (QBE), a non-procedural relational query language, is an example of that trend. Query-by-Example was first proposed by Zloof at IBM in 1975 [23] and expanded in 1977 [24]. In 1978, IBM released a commercial version of QBE which is now marketed as part of its Query Management Facility (which supports the DB2 relational database system) [17]. QBE is also part of the System for Business Automation (SBA) research project [5], an executable design language for office automation applications.

1.1.2. Zloof's Proposal

Query-By-Example as proposed by Zloof [24] is a high-level data base management language that provides a convenient and unified style to define, query, and update a relational data base. The language syntax is simple, yet it allows complex queries to be specified through the use

of the same operations for definition, retrieval, and manipulation. These language operations mimic, as much as possible, manual table manipulation and the formulation of a query is designed to capture the user's thought process. The user needs to know little to formulate simple queries and, by design, the number of concepts needed to learn and understand the whole language is minimized.

Programming in QBE is done by filling in two-dimensional skeleton tables with an example of a solution using constant elements, example elements, and operators. Example elements are variables which represent an example of a possible answer. They are also used to link tables together and to link attributes within the same table. Constant elements represent a required condition. Operators specify actions to be taken such as printing, sorting, counting, etc. A condition box can be used to express one or more conditions which are too difficult to express in the tables.

For example, the following simple query will print all the information in a table called *loans* for the employee named Mosca. p. is the print operator and 'Mosca' is a constant element.

loans	name	date	amount
p.	'Mosca'		

Insertions, deletions, and updates are done in the

same style as query operations but the print operator is replaced by the appropriate insert, delete, or update operator.

Tables can also be created by drawing a blank skeleton table, filling in the table and column heading names, and specifying the data types, sizes, etc. under the column headings. After a table has been created it can be expanded, reduced, or deleted from the database. Snapshots and views can also be created. A snapshot is a new table into which data from another table, or tables, is copied. A view is a new table which contains pointers to data in other tables but the data is not physically copied.

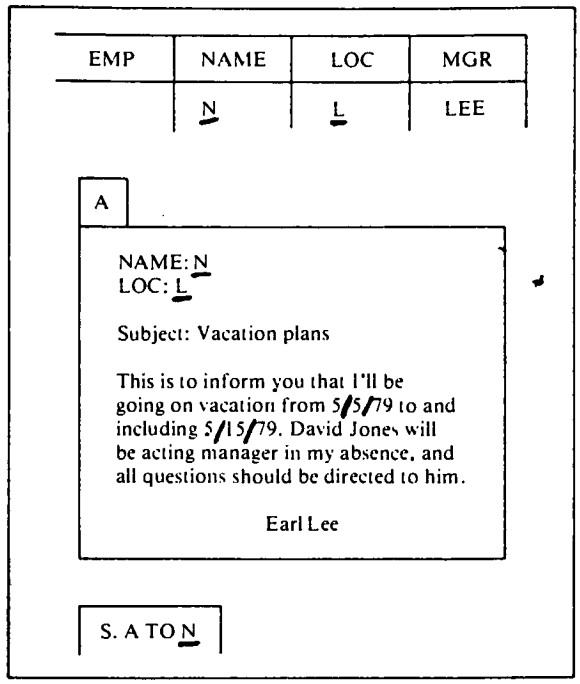
1.1.3. Recent Developments

In addition to the original work done at IBM, general theoretical work has also been undertaken and several specific applications using QBE-like languages have been proposed or implemented. The following material is presented for the purpose of illustrating the broad appeal of the QBE structure.

1.1.3.1. Development Proposals

SBA contains a unified high level nonprocedural language called OBE (for Office-Procedures-By-Example) [22], which is a superset and natural extension of QBE. It represents an attempt to combine subsets of such computer domains as word processing, data processing, electronic

mail, report writing, graphics, security features, and application developments within a single interface. The approach of OBE is the same as that of QBE: direct programming within two-dimensional pictures of business objects. The objects in OBE include letters, forms, reports, charts, graphs, and audio documents. The users create the objects on the screen in much the same way that they create the objects manually on paper. Also, users can extract data from the QBE database management system and map it into the body of the objects. The objects can be edited and sent through a communication subsystem to other nodes by specifying the receiver's ID. The users can also express various triggers which, when activated, result in an action or several actions. An example, of a trigger is a message that would be sent to managers when they exceed their budgets. See figure 1.1.3.1a for an example of an OBE program.



Object distribution program. The example element N is used as a destination address. This causes each of Lee's employees to receive a letter personally addressed with his name and location.

figure 1.1.3.1a - sample OBE program [22, p.17]

An extension to QBE was proposed in 1980 to automatically enforce semantic integrity constraints [19] The system allows the user to state integrity constraint declarations which then become a system enforcement responsibility. Integrity constraints can be one of six types:

1. perpetual check (PC)
2. pre-operative check (BC)
3. pre-operative action (BA)
4. post-operative check (AC)
5. post-operative action (AA)
6. post-operative request (AR)

A perpetual check checks a control field either before a

user starts a task or after the user terminates the task. A pre-operative check checks for a condition before the requested operation is executed. A pre-operative action requires that an action be automatically performed by the system before the requested operation is executed. A post-operative check checks a value after the operation has been executed. A post-operative action requires the system to perform the action after the operation has executed and a post-operative request requires the system to display a message to the user after executing the requested operation.

For example, given a relation called DEPARTMENT containing the dept. number, name, budget, and number of employees, the following QBE expression could be used to insure that no department containing employees is deleted:

DEPARTMENT	DEPT#	DNAME	DBUDGET	#EMP
BC(D)				=0

The BC(D) indicates a pre-operative check before a delete operation. The =0 indicates that the number of employees must be equal to zero. If a user tries to delete a department with the number of employees greater than zero, the system will not allow it.

A Generalized Query-By-Example (GQBE) data manipulation language [8] was proposed in 1983 to solve the following problem: given a collection of distributed

heterogenous databases (ie. relational, network, hierarchical) how can a user uniformly access each database without having to learn all the data manipulation languages? GQBE can be layered onto most existing databases and it supports retrieval, insertion, deletion, and update operations. GQBE extends Zloof's QBE from the relational to the heterogenous case to create a user-friendly common interface by which one can issue data manipulation commands. This is achieved by using the generalized CALCULUS data manipulation language. Each database has a defined conceptual view and a relational external view layered on top of the conceptual view. Each GQBE query is translated into a generalized CALCULUS query for the external view which is then translated into the generalized CALCULUS query for the conceptual view and subsequently executed. The user sees only the relational view of each database.

A Unified-Query-By-Example (UQBE) information manipulation language [7] has also been proposed, as an alternative to GQBE, that can be used as a stand-alone interactive language. The user can use UQBE to access a collection of distributed heterogeneous databases by communicating with the network data manager. Like GQBE, UQBE extends Zloof's QBE from the relational to the heterogeneous case.

An architecture for a Universal Office System [10] has

been proposed that is capable of supporting any office language that is causal and unambiguous. The user universe is captured in a conceptual schema and each individual user interface is mapped to the conceptual schema, using a modified QBE language. The advantage of this system is that many user interfaces can be supported while guaranteeing the compatibility of the data and other shared resources among all the users. This enables the user to migrate from one interface to another. This system is a multi-interface one which is capable of interacting in more than one type of language (menu-based, command-based, etc.). A QBE-like language on a one table database is all that is required for expressing the changes needed for a new user interface. A trichotomous organization is used which contains a user level (with several user views), a unified conceptual level, and the internal representation. The mapping from conceptual to internal level is done by the designers of the Universal Office System and the mapping from user to conceptual level is done by non-designers. Therefore, the mapping from user to conceptual level must be able to be done easily. The system will support office languages that use a CRT for input, are causal (each action must have a cause), unambiguous, deterministic, and two-dimensional.

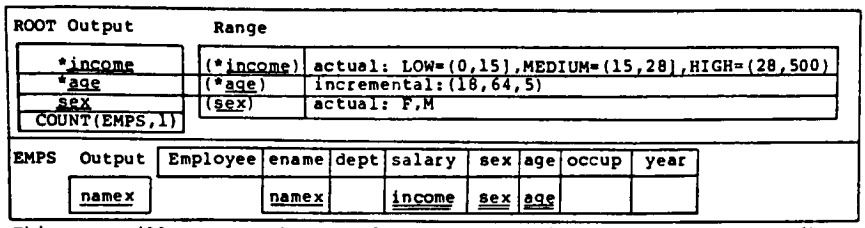
IMAID [4] is an integrated relational database system

interfaced with an image analysis system. By using pattern recognition and image processing manipulation functions, pictorial descriptions can be extracted from images and then integrated into a relational database. QPE, Query-By-Pictorial-Example, is a QBE-like language proposed for IMAID. The manipulation capabilities of QPE fall into six types: conventional manipulation, pictorial entity construction, pictorial attribute manipulation, image-sketch-relation conversion, pictorial example, and similarity manipulation. Conventional manipulation is tabular queries as defined by QBE. Pictorial entity construction allows an entity to be constructed as a point, line segment, or region of a previously stored entity. Pictorial attribute manipulation involves the retrieval of information about an entity such as area, length, or perimeter. Image-sketch-relation conversion converts a picture sketch into the pictorial attributes in a relation. Pictorial example allows selected portions of a displayed picture to be used to specify further queries. Similarity manipulation allows entities similar to a given entity to be retrieved.

1.1.3.2. Implementations

Summary-Table-By-Example (STBE) [16] is a high level nonprocedural language used to manipulate summary data in databases. Summary tables are tabular representations of summary data (ie. data that is aggregated by the application of a statistical function). In STBE the relational model is extended by introducing a new object, called the summary table. The language is used to directly manipulate summary tables and relations. It is similar to QBE in that it uses the example query concept. A summary table is represented in the extended relational model as a ternary tuple containing a list of cell attributes, a row attribute forest, and a column attribute forest. Each forest consists of zero or more attribute trees. An STBE window (or subquery) is a box and consists of a name, an output skeleton, zero or more relation skeletons, an optional condition box, and an optional range box. A range box is used to dynamically define the relations over which variables range. The window at the top level is called the ROOT. Any window may call other windows by referencing their names. An STBE query consists of one ROOT query and its subqueries. STBE is believed to be user-friendly and powerful enough to be used in medical research, health planning, scientific experiments, political planning, and office automation. See figure 1.1.3.2a for an example of an STBE query.

Example 4.1: Print a summary table of employees by age, income and sex with cells containing count of employees. Group income into three integer sets as LOW=(0,15], MEDIUM=(15,28], HIGH=(28,500) where each integer x denotes x*1000 dollars. Group age incrementally starting with 18 and incrementing by 5 up to and including 64. Group sex as male and female.



This query illustrates the use of set-valued variables as category attribute values. Both *income and *age range over the set-values specified in the range box.

figure 1.1.3.2a - sample STBE query [16, p. 200]

The ILEX research project [12] implemented a QBE-like language in Prolog. ILEX is a unified approach for all relational query languages and includes a universal interface supporting both SQL and QBE-like languages. Each query language is supported in a separate component which can be modified independently from the rest of the system, and the underlying database software can be changed independently of the user interface.

A system using a specialized QBE-like language was implemented to handle the information processing needs of the newborn intensive care unit of the University of New Mexico Hospital [11]. The computer system was designed to readily and rapidly access patient records, extract specific information, make comparisons, and compile data. It is a menu-driven, user-friendly software package which

runs on a WICAT 150 (M68000 based) multi-user, multi-tasking microcomputer. The system needed to be easily available to medical staff not familiar with computer technology, flexible, able to be used as a data management and research tool, and report generation and data investigation had to be fast and readily available. The system includes QBE routines to allow searches of all the information in the top nine menus and 28 fields of diagnoses and procedures.

The TITAN database management system [6] was designed to be a portable, interactive, and simple to use database management system for a microcomputer. It allows a user to manage the database using a relational model. The main menu allows access to three kinds of commands: database processing (to create, open, and close a database); relation processing (to create, delete, and list relations); and data management (to insert, delete, and search). The data management commands use a non-procedural, semi-graphical language based on Query-by-Example. The language allows joins and the results of a query can be used to create a new relation. Condition boxes are not supported. Each attribute can have at most one condition associated with it. The DBMS is written in UCSD PASCAL and has been implemented on an APPLE II plus microcomputer with 54k bytes of main storage and three mini diskette drives.

Geobase [2] is a geographical database system being designed (circa 1981) which allows the recursive description of images through abstract data types and mechanisms for transferring attributes through the various layers of images defined by the recursion. In this system, the QBE-like query language will not allow updates (which must be done by the database administrator) and the output is in the form of maps or tables. Special operators intended for the manipulation of pictorial data will be included as well as an operation box in which geometric operations can be specified which will act on the results of previous geometric operations.

1.1.4. Human Factors Research

There have also been several human factors studies on the ease of learning and use of QBE which have had contradicting and inconsistent results. One study [20] using college and high-school students explored the ease of learning of QBE. The subjects received about one hour and 45 minutes of training and then wrote translations of twenty test questions. The subjects then received another seventy minutes of instruction, followed by another twenty question test. Two weeks later, some of the subjects were given another twenty question test, followed by a one hour refresher course and another twenty question test. The results indicated that Query-By-Example is easy to learn

and use and that it would be hard to imagine a powerful formal language system which could be learned much more rapidly than QBE. Also, the subjects who returned after two weeks wrote queries nearly as correctly as they did when they initially learned the language, suggesting that people can easily retain the language.

In a second study [18], undergraduate students were taught both SQL and QBE. The researchers concluded the following:

1. QBE involved less training time.
2. QBE required shorter exam times.
3. The subjects had more correct QBE queries.
4. QBE required less time per query.
5. The subjects were more confident of their QBE answers.

However, only the differences in query writing time and subjects' confidence were statistically significant, indicating that QBE may not be easier to learn than SQL.

In a third study [3], SQL and QBE were taught to a group of Business Administration students and a group of secretaries, none of whom had any previous data processing experience. The students learned SQL in about two-thirds the time it took them to learn QBE. They had difficulty learning the QBE concepts of implicit AND/OR, the ALL. operator, the condition box, and the use of output tables; and the SQL concept of linking tables. In writing queries,

there was little difference in query formulation time for simple queries and queries involving logic and comparisons. However, SQL was faster for queries involving arithmetic, built-in functions, and data modification. QBE was faster for queries involving more than one table and links within a single table. When asked which of the two languages they preferred, two-thirds of the students chose SQL. The secretaries also required about two-thirds the time to learn SQL as opposed to QBE but there was no significant differences for the time to produce correct queries. However, there were twice as many unsolved queries for QBE as there were for SQL. Seven of the eight secretaries said they preferred SQL over QBE.

1.1.5. Summary

The broad appeal of QBE stems from the following:

1. It is consistent with the ease of understanding associated with the relational model.
2. In general, the use of QBE is relatively simple, as supported by the following points:
    a. The user only needs to know table names - not attribute names
    b. QBE is non-structured - the user can design a query in any order which makes sense to him/her
    c. It is easy to correct mistakes
    d. It is easy to expand a simple query to make it more complex

1.2. IBM's Query-by-Example

A commercial version of QBE has been implemented by IBM and is marketed as part of its Query Management Facility, which supports the DB2 relational database system [17]. In this version of QBE, the user forms a query by filling in the rows of a graphical representation of a table (or tables) with an example of a possible answer. This example answer is used as a pattern to select all matching records from the database.

QBE divides the terminal screen into two parts as shown in figure 1.2a. The upper (query) part of the screen is where most of the work is done. It is where the tables are drawn and where the parameters are entered which define the query. Logically, this part of the screen can consist of several pages (vertically and horizontally) of physical screens. The lower part of the screen consists of two lines defining the PF key functions, a line for messages, and a command line. The command line is used mostly for entering commands which alter the upper part of the screen in some way, such as drawing tables and scrolling.

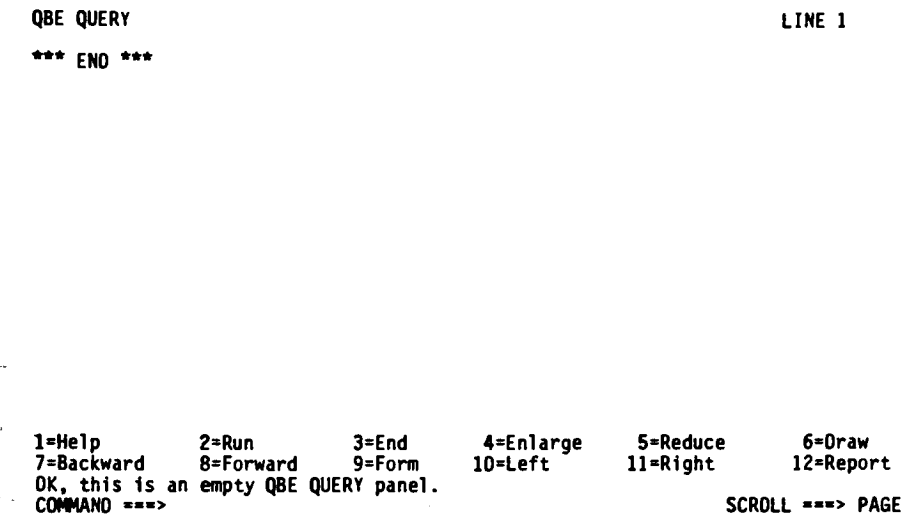


figure 1.2a - An Empty QBE Query Screen [17, p. 26]

The basic entities which define a QBE query are table representations, condition boxes, and comment boxes. Table representations can be filled in by the user with example elements, constants, expressions, and operators. Example elements are used like variables to link tables together, form conditions, and to define expressions. Each example element is tied to an attribute or attributes. Constants and conditional expressions are used to determine which records are to be selected from the tables. Operators are used to specify the type of query (select, update, insert, or delete) and to modify the results of a query by sorting, summing, counting, etc. and by restricting which attributes are selected from the table. Condition boxes are used to

define complex record selection criteria. They contain conditional expressions which are made up of example elements, conditional operators and constants. Comment boxes are used to document the query and do not effect its execution. Comment boxes are useful if a query is to be saved for later execution. See figure 1.2b for an example of a simple QBE query.

Additional details about IBM'S Query-by-Example can be found in the Query Management Facility User's Guide and Reference [17].

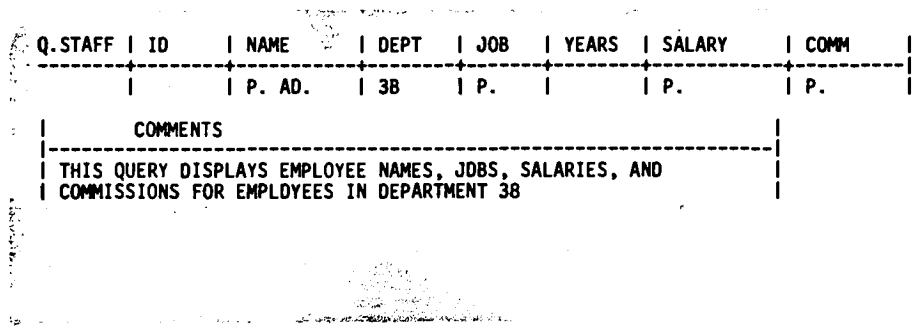


figure 1.2b - A Simple QBE query [17, p. 27]

1.3. IBM's QBE Versus Zloof's Proposal

There are several capabilities contained in Zloof's proposal that have not been implemented by IBM. The original proposal allowed new tables and views to be created and existing tables to be expanded (by adding new attribute columns), or changed (by changing the attribute names or data types). It also allowed the user to list the database directory to obtain a report of all the available table names in the database with their attributes. IBM's version of QBE does not allow any of the above operations. They must be executed using SQL. The original proposal allowed the user to fill in the headings for temporary tables; the IBM version uses arbitrary names (like 'EXPRESSION 1') for the headings. Also, in the proposal example elements can be used in conditions within the table, but in the IBM version they can only be used within condition boxes.

The IBM version also looks somewhat different than the original proposal, probably because of terminal restrictions. For example, the original proposal underlines example elements. It uses partial underlines to indicate pattern matching and double underlines to indicate grouping. Because many terminals do not allow underlining, IBM defines an example element by starting it with an underscore, and uses separate operators for pattern matching ("LIKE"), and grouping (G.).

1.4. IBM's QBE Versus Mistress/QBE

Although Mistress/QBE looks very much like IBM's QBE, there are several differences, most of which are related to differences between the IBM Structured Query language (SQL) and Mistress's Query language (which will also be called SQL). However, some of the differences make the Mistress version simpler and easier to use than the IBM version. One difference between the structured query languages is that the IBM version allows sub-selects and mathematical expressions and the Mistress version does not. (A sub-select is where the result of a query is used as part of the condition of a second query.) The same difference carries over to the two QBE versions. Because expressions are not allowed in Mistress/QBE, it eliminates one of the reasons for allowing the user to add new columns to tables or draw blank tables. Also, the IBM version allows inserts of records from one table into another table as part of the insert operation and the Mistress version allows it as part of the select operation. This difference also carries over to the two QBE versions. The IBM and Mistress versions of SQL use different complex conditional operators: IBM uses "between" and Mistress uses "range", IBM uses "like" and Mistress uses "match", and IBM has an "in" operator and Mistress does not. Finally, Mistress allows the output of a query to be in either "list" or "dump" format and IBM

does not. To handle this, the l. and du. operators have been added to Mistress/QBE.

The most important difference in the Mistress/QBE interface, which was designed to simplify use, is that the column names in the tables are protected to prevent the user from typing over them. This decreased the chance of user errors but required that other changes be made to allow the user to change headings for reports and change the order in which attributes are printed. To allow headings to be changed, a new operator (ch.) was added to the interface. To allow the order in which attributes are printed to be changed, an optional priority number was added to the print (p.) operator. These two changes also had the effect of eliminating the need for allowing the user to add blank columns or tables to the screen. Another change which was made to simplify the interface, is that the all. operator has been eliminated. In IBM's QBE, the default can be to select all records or only unique records depending on how many rows are in the example table. In Mistress/QBE the default is always to select all records. To select only unique records, the unq. operator must be specified.

There are several minor changes between the terms used in IBM QBE and those used in Mistress/QBE and also some spelling changes in some of the operators. These changes were made to either make the interface more

compatible with Mistress and UNIX, to simplify the interface program, or simply because the new words make more sense. These changes are as follows:

IBM	Mistress
end or exit	stop
export	save
import	retrieve
reset	clear
run	exec
backward	up
forward	down
top	up (as many times as needed)
bottom	down (as many times as needed)
count.	cnt.
g.	gr., gr().

Also, all scrolling in Mistress/QBE is done at the page level.

The final difference is that the Mistress/QBE screen looks slightly different in that the command portion of the screen has three message lines and does not show any PF key functions.

1.5. Mistress/QBE Versus the Mistress Query Language

Any command which can be expressed in Mistress SQL can also be expressed in Mistress/QBE, except that complex

table names containing tab characters cannot be used with Mistress/QBE and the current implementation of Mistress/QBE can only be run from tvi925, gigi, pc7300, and vt52 type terminals. Also, Mistress/QBE allows grouping operations with functions and SQL does not. In general, it can be said that SQL is better for simple queries and Mistress/QBE for more complex queries. The same query in SQL and QBE will take roughly equal amounts of time to execute, but a simple query can be typed in SQL faster than in QBE. However, a complex query can be more easily typed into QBE because the query can be decomposed into several small steps and it is easier to change a query in QBE by typing over the incorrect information than it is to change a query in SQL using the Mistress line editor. Also, multiple inserts, updates, and deletes are much easier in QBE.

1.6. A Sample Mistress/QBE Query

To illustrate how a Mistress/QBE query is constructed, a sample database containing two tables has been created. The tables are called *personnel* and *loans* and they contain the records shown in tables 1.6a and 1.6b. These tables are based on those used in the Mistress manual [15].

number	name	credit_limit	total_loans
10	Kilroy	$500.00	$250.00
5	Mosca	$750.00	$350.00
17	Wladislaw	$50.00	$50.00
3	Jones	$500.00	$358.95
8	Peterson	$250.00	$50.00
4	Scarlatti	$100.00	$0.00
9	Jordan	$250.00	$0.00

table 1.6a - contents of personnel table

name	date	amount
Mosca	02/02/87	$150.00
Jones	02/07/87	$33.95
Kilroy	02/16/87	$250.00
Wladislaw	02/27/87	$55.00
Jones	04/03/87	$25.00
Mosca	05/04/87	$200.00
Wladislaw	05/12/87	$25.00
Peterson	06/06/87	$50.00
Wladislaw	06/25/87	$75.00
Jones	08/12/87	$300.00

table 1.6b - contents of loans table

To print the number, name, credit limit, date, and amount for all loans taken out in 1987, the following steps could be taken:

step 1: draw the *personnel* table

step 2: draw the *loans* table

step 3: draw a condition box

step 4: move the cursor to the *personnel* table and type the print operator (p.) under the *number*, *name*, and *credit_limit* columns - type the example element *_nam* under the *name* column

step 5: move the cursor to the *loans* table and type the print operator under the *date* and *amount* columns - type the example element *_nam* under the *name* column and *_dat* under the *date* column

step 6: move the cursor to the condition box and type the following:

_dat >= '01-01-87' and _dat <= '12-31-87'

The screen will now look like figure 1.6c.

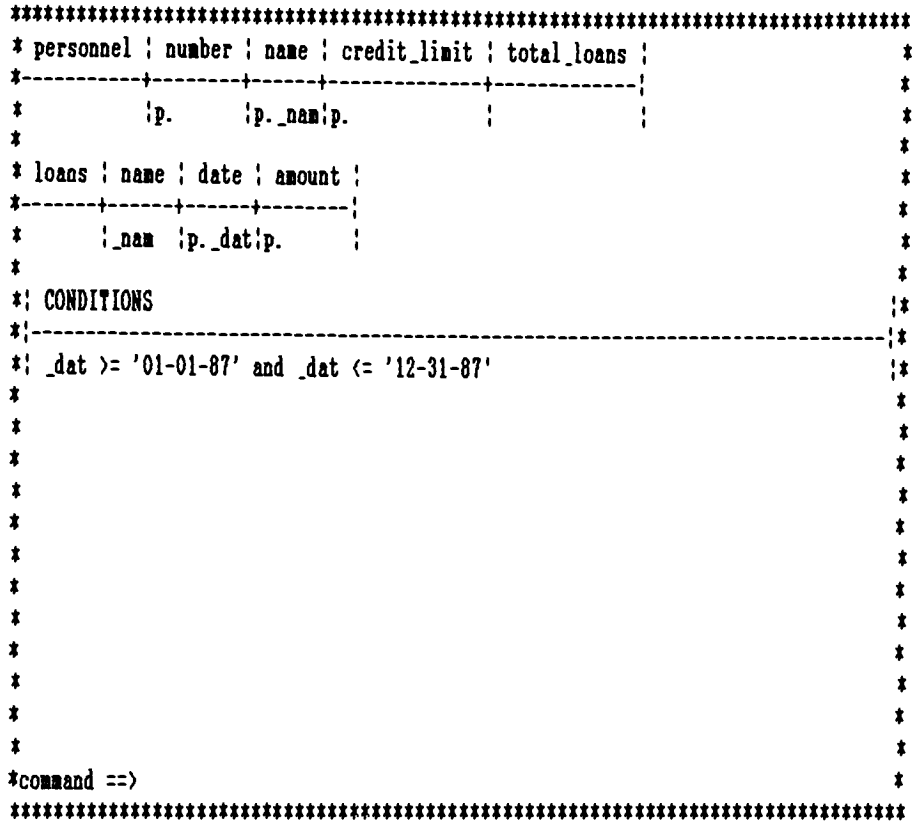


figure 1.6c - sample query

The print operator (p.) indicates which attributes will be printed on the report. The example element *_nam* is used to link the two tables together and the example element *_dat* is used to specify which attribute is to be used in the condition.

step 7: move the cursor to the command line and type *exec* to execute the query

The resulting report is shown in figure 1.6d.

number	name	credit_limit	date	amount
10	Kilroy	$500.00	02/16/87	$250.00
5	Mosca	$750.00	02/02/87	$150.00
5	Mosca	$750.00	05/04/87	$200.00
17	Wladislaw	$50.00	02/27/87	$55.00
17	Wladislaw	$50.00	05/12/87	$25.00
17	Wladislaw	$50.00	06/25/87	$75.00
3	Jones	$500.00	02/07/87	$33.95
3	Jones	$500.00	04/03/87	$25.00
3	Jones	$500.00	08/12/87	$300.00
8	Peterson	$250.00	06/06/87	$50.00

figure 1.6d - result of sample query

1.7. Performance

In general, the performance of Mistress/QBE is directly related to the number of changes being made to the screen or the number of equivalent Mistress SQL commands being executed. For example, moving the cursor is done instantaneously and redrawing the screen takes about five seconds. Each non-grouping report query requires one Mistress SQL command, and inserts, updates, and deletes require one SQL command per row in the example table. Grouping queries require six SQL commands (to select the data needed for the query, to create temporary tables, and display the final result) plus three SQL commands per each line in the output report (to select the data for each group and move data between the temporary tables). Therefore, grouping and multiple inserts, deletes, and updates can be quite slow.

2. Architectural Design of Mistress/QBE

2.1. Programming Language, Interfaces, and Subroutines

Mistress/QBE is written entirely in the C language and uses the Mistress standard C interface [13], the Mistress MR routines [14], and the UNIX *curses* library of subroutines [1]. The Mistress standard C interface is invoked from a C program by calls to Mistress which specify the database directory and a Query Language command. Mistress then executes the command. The Mistress MR routines are very similar to a procedural programming language and allow the programmer to sequentially read and update tables in the Mistress database. The UNIX *curses* routines are used to move the cursor around the screen and allow full screen input and output.

Mistress/QBE uses the *curses* routines to accept commands from the user and then to carry out the commands by drawing tables and boxes on the screen. The *curses* routines also allow the user to move around the entire screen to fill in the parameters of the query. After the query has been defined, Mistress/QBE processes the data on the screen and builds the equivalent Query Language command which is executed by the Mistress standard C interface. The Mistress MR routines allow the interface to implement grouping operations which are not directly supported by the Mistress Query Language.

2.2. Top Level Design

The Mistress/QBE interface program is divided into three sections. The first section is the main program. This section accepts input from the user and, based on what the user has entered on the command line, determines the action to be taken. This section prevents the user from typing into a protected area of the screen and handles the differences in cursor keys between the different terminal types. It executes Mistress commands not directly supported by QBE. It also includes the global utility functions which can be called from any other function in the program. The second section of the program executes the screen altering commands. This includes functions to draw tables and boxes on the screen, delete tables and boxes from the screen, enlarge or reduce tables and boxes, clear the screen, save and retrieve queries, scroll the current page up, down, left, or right, display the help screen, refresh the current page, and move the cursor. The third section contains the functions necessary to execute the current query. This includes functions to insert records into a table, update existing records in a table, delete records from a table, and print reports. See figures 2.2a and 2.2b for a structure diagram of the higher level functions of the program.

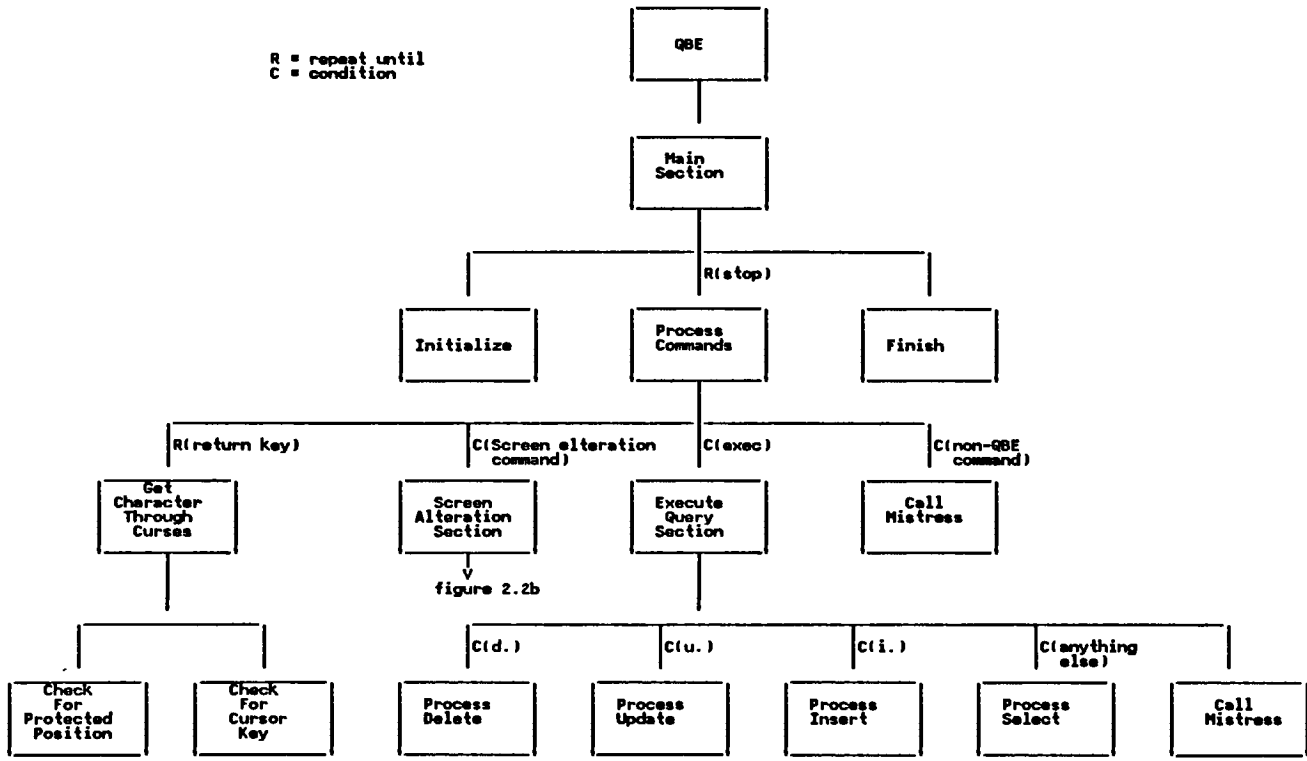


figure 2.2a - page 1 of structure diagram

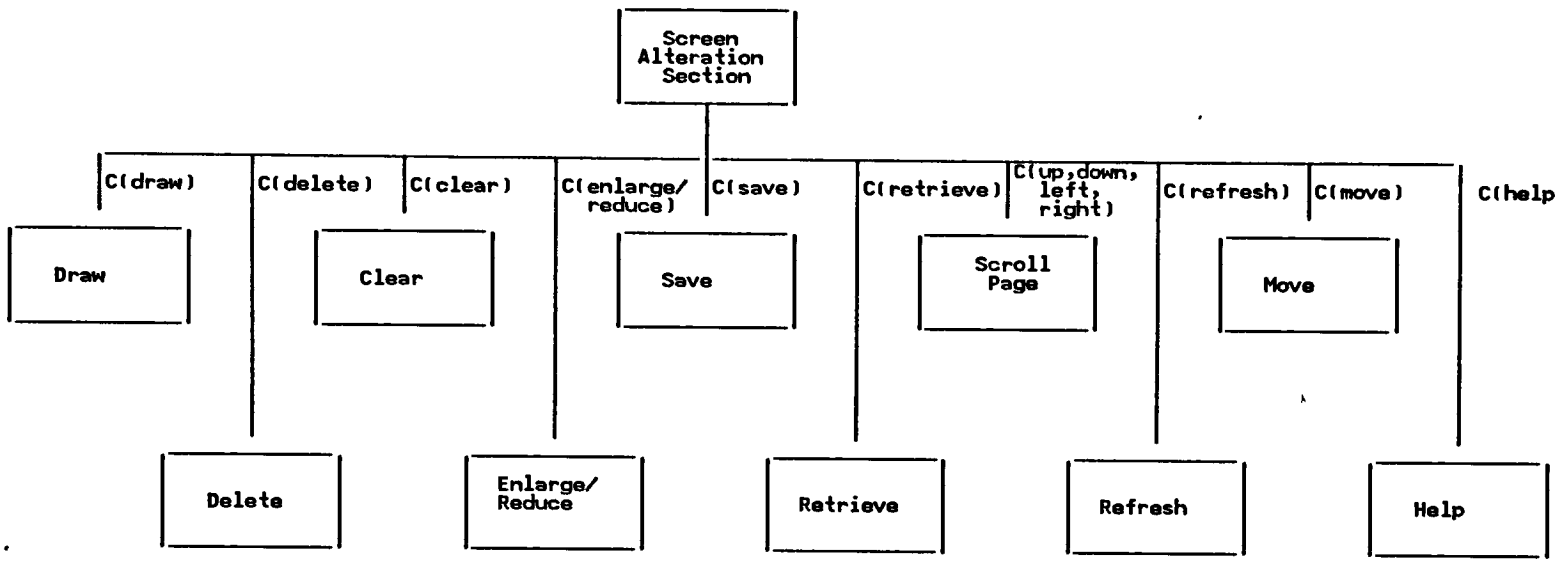


figure 2.2b. - page 2 of structure diagram

2.3. Data Structures

There are seven principle data structures used by the program to pass data between the three sections of the program. The first two are linked lists used to hold information about the tables, boxes, and attributes. The third is a table which is used to determine which positions on the screen are protected and which can be typed into. The fourth is another table which is used as a cross-reference between each line on the screen and its associated table or box. The fifth structure is the window structure used by *curses* to hold the information typed on the screen. There is one window for each physical page of the query. The sixth structure is a linked list used to hold information about each page. The last structure is another linked list which holds information about example elements. See figure 2.3a for a diagram showing the relationship between the data structures, the Mistress database, and the three sections of the program.

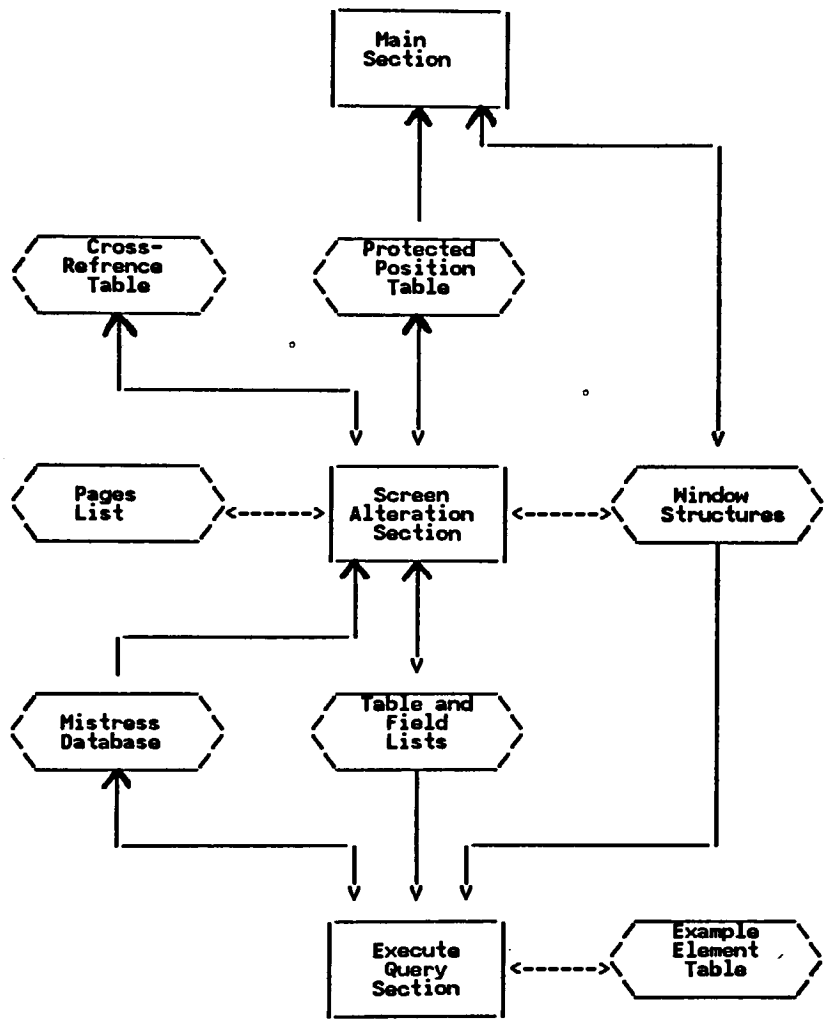


figure 2.3a - data flow diagram

2.4. Functional Specifications

The following is a brief description of the higher level functions of the program:

1. Get data from screen - This function uses the *curses* routines to get input from the user from anyplace on the terminal screen. Characters are read until the return key or escape key is typed. Cursor control keys (which can send a one to three character code depending on the terminal) are checked for. Input is whatever is typed at the terminal (a command on the command line and/or information in the query portion of the screen.) Output is an echo of what is typed.
2. Execute non-QBE Mistress commands - This function calls Mistress using whatever was typed on the command line (*sort, display, create, rename, drop, empty, database, run from, insert from, update from, do, !,* or *stop*). The *stop* command causes the QBE cleanup routines to be performed before exiting the program. Input is the Mistress command. Output is the same as if the command was executed in SQL.
3. Execute screen alteration commands - This function executes the commands which alter the current screen. These commands are: *clear, draw, delete, up, down, right, left, save, retrieve, help, refresh, move, enlarge,* and *reduce*. The *clear* command deletes all tables and boxes. The *draw* command causes a table,

condition box, or comment box to be drawn. *Delete* removes tables or boxes from the screen. *Up, down, right,* and *left* scroll the screen one page in the desired direction. *Save* writes the current query to a file. *Retrieve* reads in a previously *saved* query. *Help* displays a help screen. *Refresh* redraws the screen when it is in error. *Move* moves the cursor. *Enlarge* adds lines to a table, widens columns, adds lines to a box, or widens a box depending on where the cursor is located. *Reduce* is the opposite of *enlarge.* Input is the appropriate command typed on the command line. The output is the altered screen.

4. Test mode - This function executes commands used in the testing mode. These commands are: *trace, octal, run, dump,* and *dumpall. Trace* sets tracing on or off. *Octal* displays the previous command in octal. *Run* takes commands from a file instead of the command line. *Dump* writes the contents of the current page to a file. *Dumpall* writes the contents of several variables and tables to a file.

5. Execute query - This function determines what operation is to be performed by looking at the query portion of the screen and calls the appropriate function (select, insert, update, delete). Input is the *exec* command. Output is the result of the called sub-function.

6. Selects - This function has to be able to execute any select which can be executed by the Mistress SQL, including joins and alternate formats. In addition, it also executes grouping operations. The user instructs the program to draw pictorial representations of the tables involved, condition boxes, and comment boxes and then fills in the tables and boxes to specify the parameters of the query. The output is the same as for the corresponding SQL query.
7. Insertions - This function inserts records into a table. The user instructs the program to draw a pictorial representation of the table and then fills in the information to be added to the table. This can be done one record at a time or several at a time. Output is an acknowledgement that the insertions have been completed.
8. Updates - This function updates existing records in a table. The user instructs the program to draw the table and then fills in the updated information. This can be done one row at at a time or several at a time. Output is an acknowledgement.
9. Deletions - This function deletes records from a table. The user instructs the program to draw a table and then fills in the conditions which determine which records will be deleted. This can be done one row at a time or several at a time. Output is an acknowledgement

2.5. Algorithm

The following is a pseudo coded algorithm for the higher level functions of the program:

```
main function:
    initialize
    repeat until command = stop
        process the command
    cleanup routine
process command function:
    get a screen of data through curses
    check command line
    if screen alteration command
        process the command
    else if test mode command and test mode is on
        process the command
    else if non-QBE Mistress command
        call Mistress
    else if command = exec
        execute query
    else if invalid command
        display error message
get screen of data function:
    repeat until return key or escape key is hit
        get a character through curses
        check for cursor key - if found, move cursor
        check for escape key - if found, build command
```

```
        check for protected position - if found, tab to
            next unprotected position
         if character is printable
             echo character
screen alteration command function:
    if command = draw tablename
        get table column names from Mistress
        display the table
        update data structures
    else if command = draw cond
        draw blank condition box
        update data structures
    else if command = draw comm
        draw a blank comment box
        update data structures
    else if command = up, down, right, or left
        scroll 1 page in indicated direction by moving
            the desired window to the current curses
            screen
    else if command = enlarge or reduce
        determine position of cursor
        determine what table or box the cursor is in
        enlarge or reduce the indicated table or box
        update data structures
    else if command = delete
```

```
        determine position of cursor
        delete indicated table or box
        update data structures
    else if command = clear
        delete all tables and boxes
        update data structures
    else if command = save
        save current query to a file
    else if command = retrieve
        delete all tables and boxes
        read a previously saved query from a file
        update data structures
    else if command = help
        display help screen
        re-display current screen
    else if command = refresh
        redraw current screen
    else if command = move
        move the cursor to the indicated position
test mode function:
    if command = trace
        turn tracing mode on or off
    else if command = octal
        print previous command in octal
    else if command = run
```

```
        read commands from a file instead of the
            command line
    else if command = dump
        write the current screen to a file
    else if command = dumpall
        write variables and data structures to a file
execute query function:
    look for i., d., or u.;
    if i.
        process insertion
    else if d.
        process deletion
    else if u.
        process update;
   else if anything else
        process select
select function:
    search tables and boxes
    determine what columns are to be printed
    determine "from" clause table names
    process example elements, constants, and conditions
    formulate Mistress select command
    call Mistress
insertion function:
    for each row of table
        formulate Mistress insert command
```

```
        call Mistress;
deletion function:
    for each row of table
        process example elements, constants, and
            conditions
        formulate Mistress delete command
        call Mistress;
update function:
    for each row of table
        look for updated columns
        process example elements, constants, and
            conditions
        formulate Mistress update command
        call Mistress;
```

3. Systems Documentation

3.1. Source Files

The following UNIX files contain the source code for the Mistress/QBE interface program:

qbe.def - defined constants

qbe.h - external definitions of qlobal variables

qbem.c - main program

qbeg.c - global utility functions

qbet.c - testing mode functions

qbesa.c - 1st file of screen alteration functions

qbesb.c - 2nd file of screen alteration functions

qbesg.c - common functions called from qbesa.c and qbesb.c

qbeqm.c - main query execution functions

qbeqs.c - "select" type query execution functions

qbeqg.c - common functions called from qbeqm.c and qbeqs.c

The commands to compile the above source code (except the qbet.c file) are located in the *qbecc* file. The commands to run the interface are located in the *qbe* file.

3.2. Test Mode

The Mistress/QBE program has a built in testing mode which is useful when debugging changes to the program. This mode is turned on by defining the constant TEST_RUN in

the *qbe.def* file and compiling the *qbet.c* file using the following command:

```
cc -c qbet.c -lcurses -ltermlib
```

The qbet.o file must be added to the last line of the *qbecc* file and the *qbecc* file must be executed to recompile the rest of the program.

When executing in testing mode a file called *qbe.lst* will be created which will contain the SQL version of every query executed and any other Mistress commands which are executed. The following additional commands will also be available:

trace [on, off] - this will turn tracing on and off - when tracing is on, the name of most functions will be written to the *qbe.lst* file when the function is entered and the word *exit* will be written when the function is exited. (This can be abbreviated *tr*).

octal - this will display the previous command in octal on the error message lines. (abbreviation is *oct*)

run *filenm* - this will cause commands to be read from the indicated UNIX file instead of from the command line.

dump - this will dump the contents of the current page to *qbe.lst*. (abbreviation is *du*)

dumpall - this will dump the contents of several global variables and the contents of the protected

position, cross-reference, and page tables to *qbe.lst*. (abbreviation is *duall*).

3.3. Data Structure Details

The first of the principle data structures mentioned in section 2.3. is a linked list of structures used to hold information about tables and boxes. There are actually three separate lists; one for tables, one for condition boxes, and one for comment boxes. The structure is called TABLE_INFO and contains the following information about each table or box: the type of entity (table or box); the name; whether the name is simple or complex; the database if different from the default; the length of the column on the screen; the first line used on the screen; the last line used on the screen; the last position used on the screen; and pointers to the attribute list for the table and the next and prior structures in the list. It also contains several variables which are useful only during the actual execution of the query and which are reset for each new execution.

The second principle data structure is a linked list used to hold information about each attribute of a table. There is one list for each TABLE_INFO structure. This structure is called FIELD_INFO and contains the following information: the attribute name; whether the name is simple or complex; the length of the column on the screen;

the first position on the screen; and pointers to the next and prior structures in the list. It also contains several variables which are useful only during execution of the query and which are reset for each new execution.

The third data structure is a table which determines which positions on the screen can be typed into and which are protected. There are actually two tables. The first is called BSKIP and controls the command portion of the screen. It is a fixed length 4 X 80 array. The second table is called SKIP and controls the query portion of the screen. It is implemented using pointers. There are 20 pointers for each vertical page of the query and each pointer points to an area with a length of 80 times the number of horizontal pages on the screen plus one position to hold the null character so that each area can be handled as a string. Each position in these tables is set to '1' for protected or '2' for unprotected.

The fourth data structure is the cross-reference table. This table is called Y_TAB and consists of 20 pointers for each vertical page of the screen. Each pointer points to the TABLE_INFO structure for the table or box located on that line. If the line is blank, the pointer will be set to null.

The fifth structure is the *curses* window structure. This structure contains information on the size of the

window, the current position within the window, and the information on the window. For more information, see the section on *curses* in the UNIX Programmer's Manual [1]. The interface program uses several windows. There is a window to hold the information on the current terminal screen. This is where all the information from the user is typed and where the command portion of the screen is located. There is a window to hold the help screen and two work windows. There is also a window which holds all the information for all the pages in the query. This window contains 20 lines for each vertical page and 80 positions for each horizontal page. Each page is defined as a 20 X 80 sub-window of the larger window.

The sixth structure is called PAGE_INFO and is used to hold information about each page in the query. There is one PAGE_INFO structure for each physical page and each structure contains the following information: a pointer to the associated sub-window; pointers to the next page up, down, left, and right; the next available line on the page; and the page number.

The seventh structure is a linked list of structures called VAR_INFO which contain information about example elements. There is one VAR_INFO structure for each example element defined in the query. Each structure contains the following information: the name of the element; its line number on the screen; pointers to the associated table and

attribute; and pointers to the next structure in the list and the next structure with the same name. This structure is only used during execution of the query and is reset at the beginning of each new execution.

In addition to the above seven data structures which are used to hold information about the query, there is also a structure which is used when building the strings needed for the SQL command. This structure is called STR_INFO and is used to hold information about dynamically allocated variable length strings. Each structure contains a pointer to the string and the current size of the string. If a fixed length string needs to be used in a function requiring a variable length string, a STR_INFO structure must be set up which points to the fixed length string.

3.4. Function Details

Each function in the Mistress/QBE program has a short name and a long name. The short name identifies the source file, the relative position of the function within the file, and (usually) the calling function. The long name is descriptive of the purpose of the function. For example, in the function name "sa320_build_condition_box", "sa320" is the short name and "_build_condition_box" is the long name. The "sa" means that the function is located in the *qbesa.c* source file. The "320" gives the location in the file (the functions are in numerical order) and indicates

that it is called from function "sa300". The long name indicates that the purpose of this function is to build a condition box.

Most functions (except those that are called too often) have the *TRACE* macro as the first statement. This will cause the function name to be printed if tracing is on in testing mode. Any function which contains the *TRACE* macro should use the *RETURN* macro instead of the *return* command. This will cause the word "exit" to be printed.

Figures 3.4a - 3.4j contain a detailed hierarchy chart of all the functions in the program. If there are any questions about the purpose of a function, refer to the program listing.

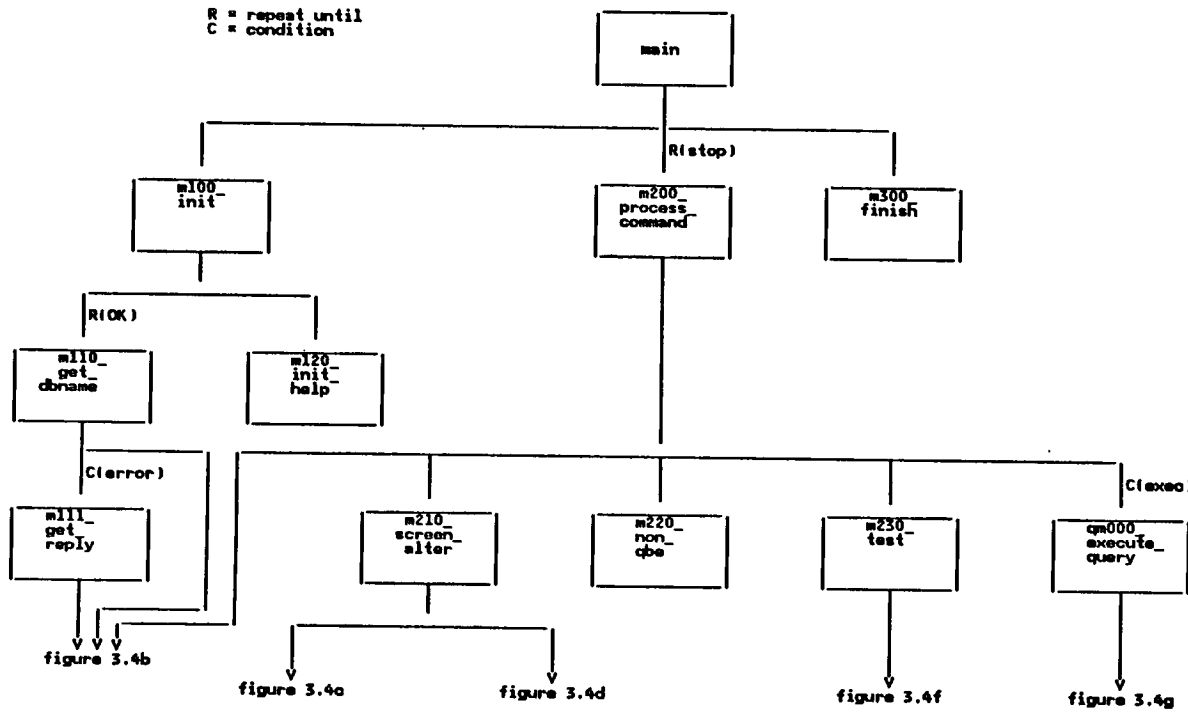


figure 3.4a - main program (qbem.c)

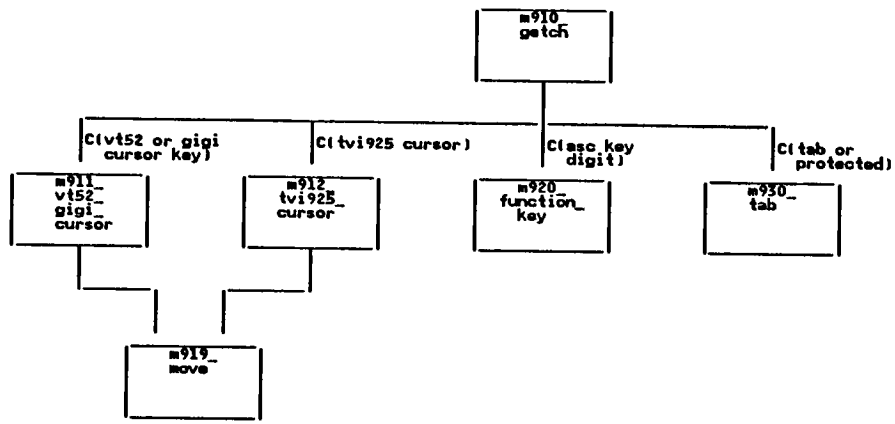


figure 3.4b - main program - this is a function which can be called from several functions in qbem.c

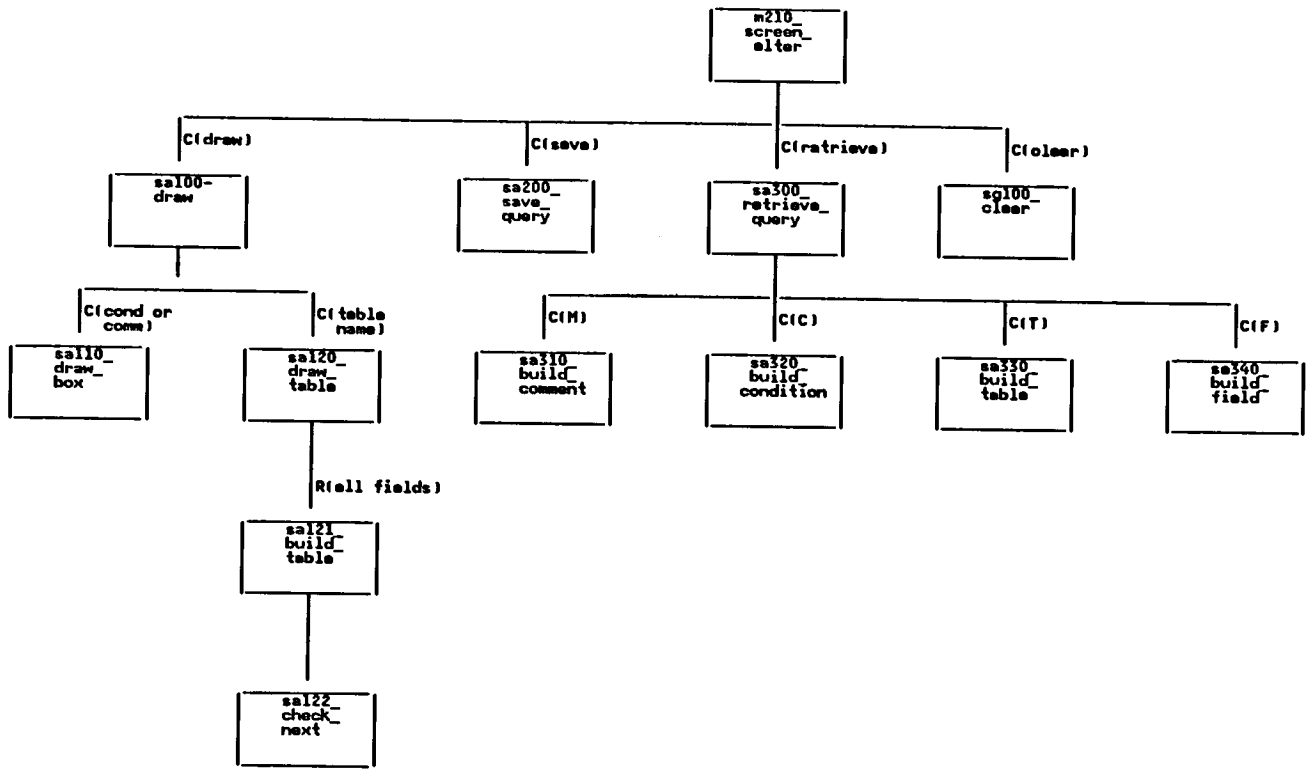


figure 3.4c - screen alteration functions (qbase.c)

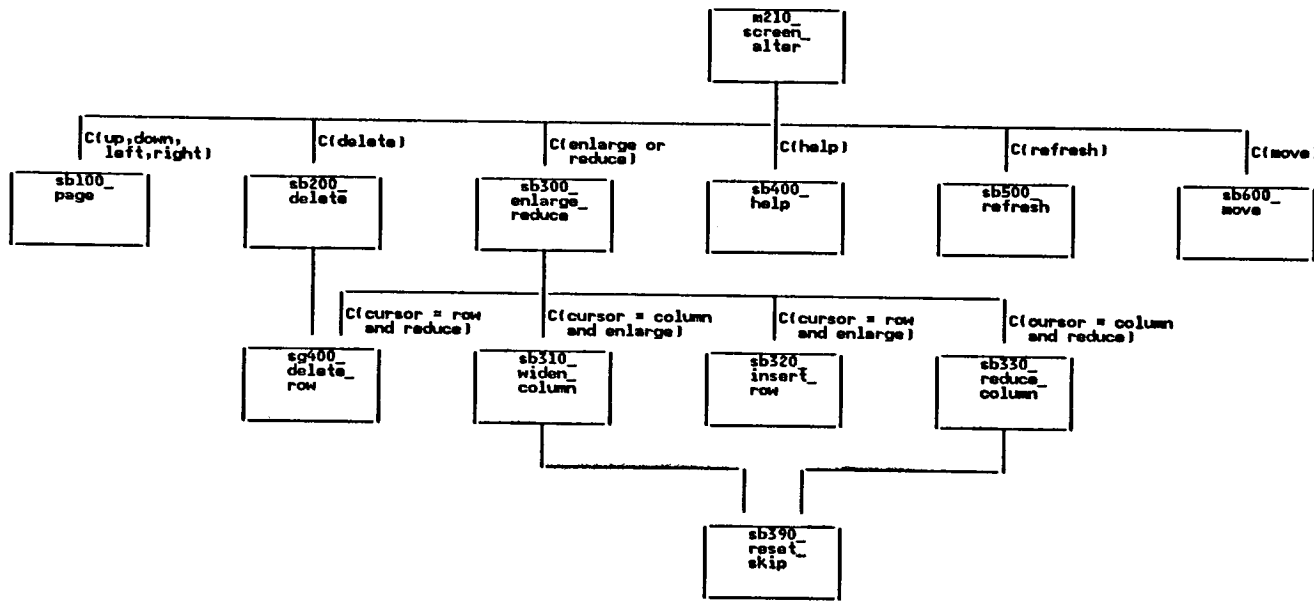


figure 3.4d - screen alteration functions (qbesb.c)

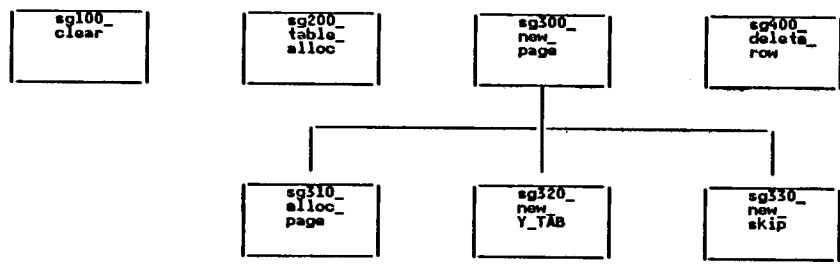


figure 3.4e - screen alteration functions - functions common to qbese.c and qbesb.c (qbesg.c)

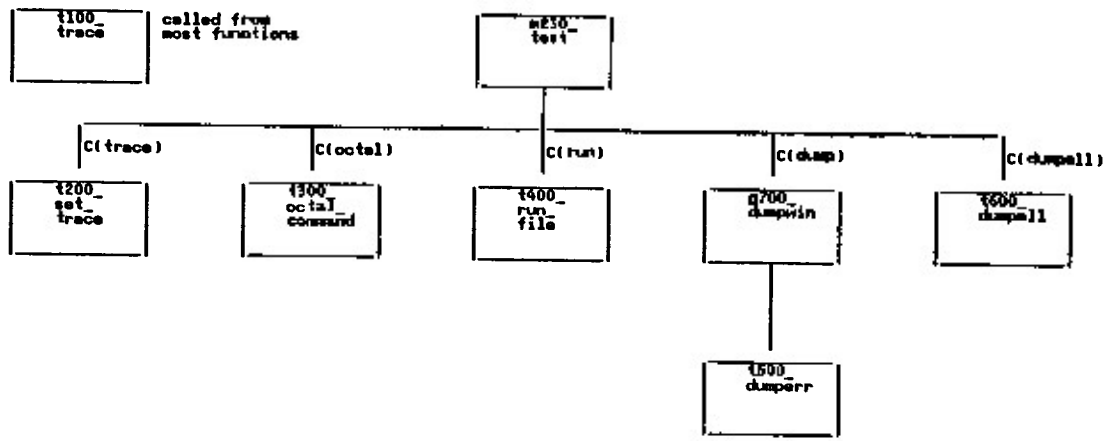


figure 3.4f - testing mode functions (qbet.c)

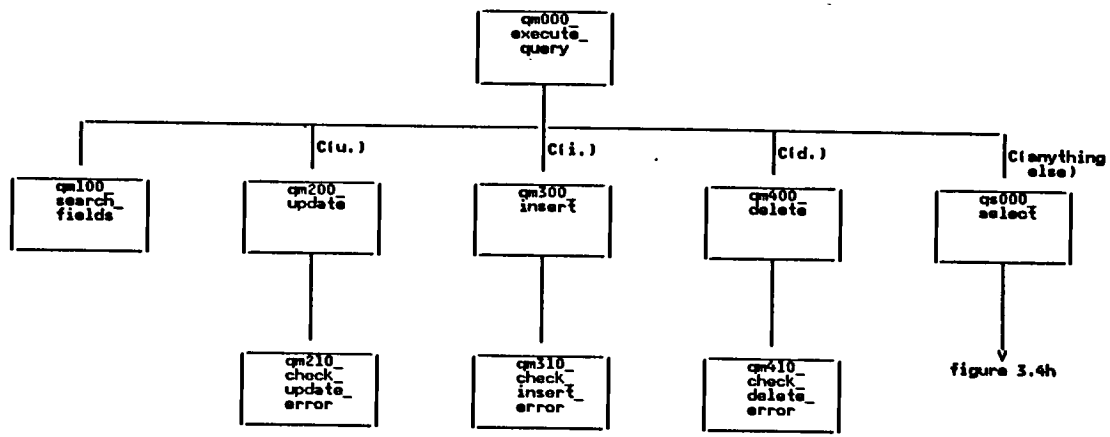


figure 3.4g - query execution - main functions (qbeqm.c)

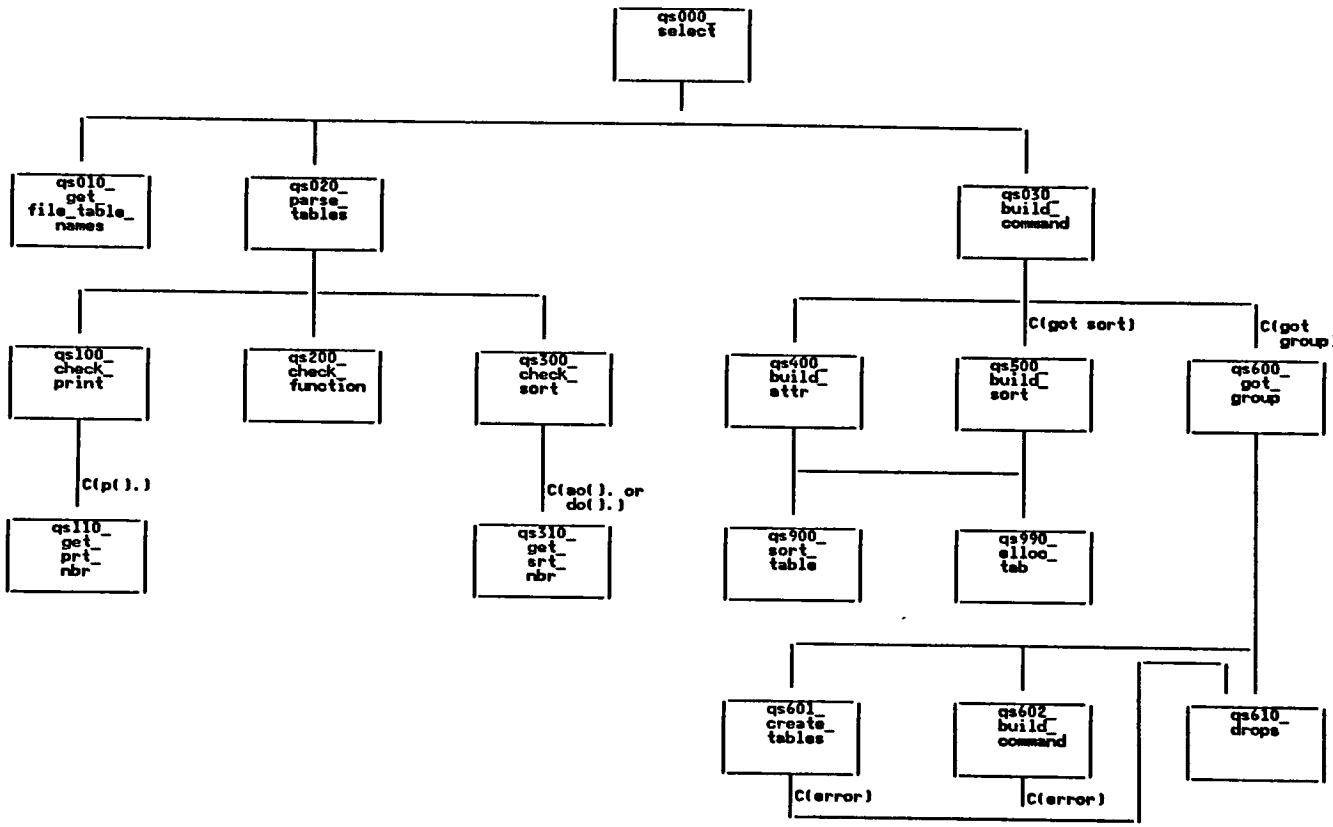


figure 3.4h - query execution - select query functions (qbeqs.c)

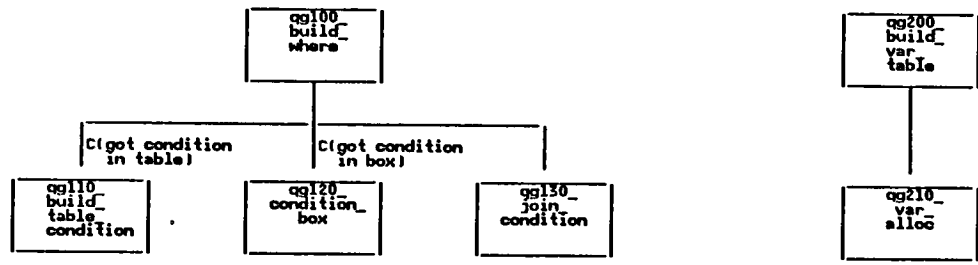


figure 3.4i - query execution - functions common to qbeqm.o and qbeqs.o (qbeqg.o)

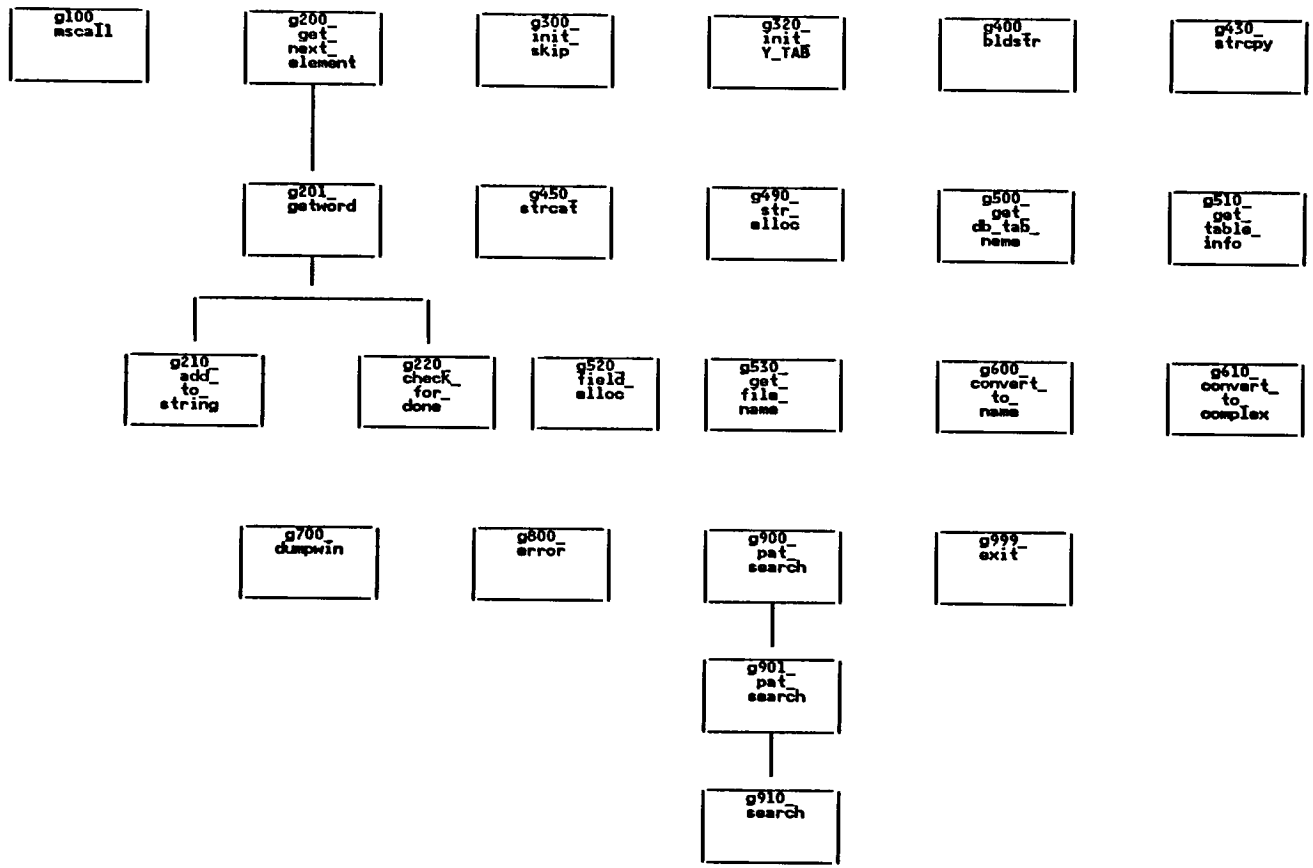


figure 3.4j - global utility functions (qbeg.c)

3.5. How Grouping Is Implemented

Since grouping is not directly supported by Mistress, it must be simulated using several steps and several Mistress commands. The first step is to execute the QBE query as though all the attributes containing the grouping (gr.) and function operators contained the print (p.) operator instead and ignoring the unique (unq.), list (l.), and dump (du.) operators. The result of this query is inserted into a table called *qbetab2*. This results in a table containing all the attributes needed for the grouping operation from the records which satisfied the conditions of the query. The unique grouping attributes (those which were selected by the gr. operators) are then selected from *qbetab2* and inserted into a table called *qbetab3*. There will be one line on the report for each record in *qbetab3*. Using the MR routines, the records from *qbetab3* are inserted one at a time into a table called *qbetab4*. This table will never contain more than more one record. Each record is also inserted into a table called *qbetab5*. This table will contain the records used to print the final report. A Mistress SQL command will then be built to apply the function to the records in *qbetab2* which match the record in *qbetab4*. If the unq. operator was specified in the original query, it will be applied at this time. The result of the function will be dumped into a file called *qbetab*. This value will then be inserted into *qbetab5*.

After all the records in *qbetab3* have been processed, a Mistress SQL command will be built to select all the records from *qbetab5* and either display them on the terminal or list or dump them into a file if it was specified in the original QBE query. All the temporary tables used are then dropped from the database.

3.6. How to Add a New Terminal Type

To add a new terminal type, the entry for the terminal must be found in the */etc/termcap* database. This entry will give the number of lines and columns for the terminal and also the character(s) sent by the cursor keys. The number of columns are given after the "co#" entry, the number of lines after "li#", the up cursor key after "ku=", the down cursor key after "kd=", the right cursor key after "kr=", and the left cursor key after "kl=". For example, the gigi graphics terminal has the following entries:

```
    co#84       (84 columns)
    li#24       (24 lines)
    ku=\EOA     (esc, O, A)
    kd=\EOB     (esc, O, B)
    kr=\EOC     (esc, O, C)
    kl=\EOD     (esc, O, D)
```

and the tvi925 terminal has the following entries:

```
    co#80       (80 columns)
    li#24       (24 lines)
```

```
ku=^K       (cntl K - sends octal 13)
kd=^V       (cntl V - sends octal 26)
kr=^L       (cntl L - sends octal 14)
kl=^H       (cntl H - sends octal 10)
```

(The octal values of the control characters were found by writing a short test program.) If the cursor keys are the same as a terminal type which has already been defined, only function *m100_init* will need to be changed to assign the variable *Ttynum* the same value for the new terminal type as for the existing type. (The default is tvi925, so if the new terminal has the same keys as a tvi925 then no changes need to be made.) If the cursor keys are different, then the *qbe.def* file must be changed to add a defined constant for the new terminal type. The constants currently defined are:

```
TVI925      1
GIGI        2     (also used for PC7300 and unixpc)
VT52        3
```

The *m100_init* function will have to be changed to assign the new defined constant to the variable *Ttynum*. The *m910_getch* function will have to be changed to check for the new terminal type and its associated cursor keys. If the left cursor key has an octal value of 10, then it is the same as the backspace key. If the down cursor key has an octal value of 12, then it is the same as the new line

key (and, for the purposes of this program, the return key).

If the new terminal type has less than 80 columns or less than 24 lines, then the *COMLINE*, *ERRY*, *NUMLINES*, *LASTY*, *LINESIZE*, *LASTX*, *COMSIZE*, and/or *COMMAX* defined constants must be changed and all files must be recompiled. Also, some queries which were saved using the larger terminal size may no longer work.

4. Conclusions

4.1. Problems Encountered and Solved

The only technical problems encountered involved learning to use the *curses* routines. This was especially difficult because I could not find anyone else who had ever used the routines, the article [1] is not very clear in some places, and contains some errors. For example, it took me several weeks to find the best way of accepting input from the user. At first I allowed all the defaults to be in effect, but this resulted in characters being echoed on the screen but not picked up by the program and some characters which were picked up were invalid. I discovered that although the article stated that the default was *cbreak* mode, it was actually *raw* mode and this was causing most of the problems. I also discovered that I could not allow *curses* to automatically echo characters on the screen because it would echo non-printable characters (like the cursor keys) as well as printable characters. Also, although pressing the cursor keys would cause the cursor to move, the *curses* routines would not realize that the cursor had moved. I had to check for the cursor keys and specifically tell *curses* that the cursor had moved. Towards the end of the project, I discovered that two of the subroutines (winsertln and wdeleteln) do not always work correctly and *curses* appears to allocate about four times as much storage for the windows than is necessary.

4.2. Suggestions For Future Extensions

There are several enhancements which could be made to the program to either make it more user friendly or more compatible with the IBM version. The first would be to set up a way to erase to the end of a field and to erase all the unprotected areas of the screen. This could be done using escape sequences in a manner similar to the way program function keys are simulated using an escape key/digit combination. However, care must be taken not to use the same combination sent by the cursor keys.

The help screen could be expanded to include several screens, possibly one for each command and operator. This could function as an on-line user manual. To save on storage this could be implemented by storing the information for each screen in a file or files to be read only when needed.

Currently, the Mistress/QBE program only allows scrolling at the page level and uses a fixed amount of eight when enlarging or reducing column sizes. This could be changed to allow the user to type an amount on the command line which would be used to either scroll the screen the indicated number of lines or columns in any direction or enlarge or reduce a column by the indicated amount. Also, *top* and *bottom* commands could be added to allow the user to scroll to either the first or last page.

New tables and boxes are always drawn after the last

table or box already on the screen. The program could be changed to allow the user to indicate by cursor position where the next table or box should be drawn.

Another, more difficult, enhancement would be to make the Mistress/QBE program optionally menu driven. This would make it easier for an inexperienced user to use the system until becoming familiar with the commands. The user could choose the desired command from a menu instead of typing it on the command line. In the case of the *draw* command, the user could then choose the desired table from another menu.

To make Mistress/QBE more like the IBM version, the ability to evaluate mathematical expressions and sub-selects would have to be added. To use expressions, the interface would have to do several things. First, it would have to allow blank columns to be added to tables to hold expressions to be printed. Second, it would have to parse the expressions, both those in conditions and those to be printed, and translate them into the appropriate C language statements. Third, and last, the interface would have to process the records in the table(s) sequentially using the MR routines in order to substitute the values of the appropriate attributes into the expressions.

To implement sub-selects, the interface would have to first execute the sub-select and put the resulting value or

values into a file or table. The main select would then have to be executed using the values in the created file or table to select the records. This would require processing the table(s) of the main select sequentially. The interface would also have to implement the *any*, *all*, and *exists* operators of the IBM Structured Query Language to insure that the value(s) found by the sub-select are processed correctly by the main select. The interface program would have to be flexible enough to handle several levels of sub-selects.

5. Bibliography

1. Arnold, Kenneth, "Screen Updating and Cursor Movement Optimization: A Library Package", *UNIX Programmer's Manual*, section 3. This article describes the *curses* library of subroutines.

2. Barrera, R. and Buchmann, A., "Schema Definition and Query Language for a Geographical Database System", *1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, 1981, pp. 250-256. GEOBASE, a geographical database system using a QBE-like language, is described.

3. Boyle, J.M., Bury, K.F., and Evey, R.J., "Two Studies Evaluating Learning and Use of QBE and SQL", *Proceedings of the Human Factors Society 27th Annual Meeting*, vol.2, 1983, pp. 663-667. Two studies were performed using college business students and secretaries to evaluate and compare the learning and use of QBE and SQl.

4. Chang, N.S. and Fu, K.S., "Picture Query Languages for Pictorial Data Base Systems", *Computer*, vol. 14, no. 11, Nov. 1981, pp. 23-33. The authors discuss pictorial query languages for manipulating images in image databases.

5. De Jong, S.P., "The System for Business Automation (SBA): A Unified Application Development System", *Information Processing 80. Proceedings of the IFIP Congress 80*, 1980, pp. 469-474. SBA is a system which allows the end-users to directly describe their applications on the computer. The approach is for the user to deal with the two-dimensional objects he is familiar with when he performs the application manually.

6. Falquet, G., Petitpierre, D., Magnenat-Thalmann, N., and Thalmann D., "A Portable Relational Data Base Management System for Microcomputers", *Microprocessors and Microprogramming*, vol. 9, No. 1, Jan. 1982, pp. 17-25. The authors describe the design and implementation of a microcomputer-based DBMS intended for the non-specialist user which includes a relational database model and a QBE-like query language.

7. Hsieh, Y.I., "A Unified-Query-By-Example Information Manipulation Language Based on Functional Data Model", *Proceedings COMPSAC83: The IEEE Computer Society's Seventh International Computer Software and Applications Conference*, Nov. 1983, pp. 571-579. A UQBE language allows a user to access a collection of distributed heterogeneous databases from an interactive terminal of a network.

8. Jacobs, B.E. and Walczak, C.A., "A Generalized Query-by-Example Data Manipulation Language Based on Database Logic", *IEEE Transactions on Software Engineering*, vol. 9, no. 1, Jan. 1983, pp. 40-57. The authors discuss the GQBE data manipulation lanuage which is similar to QBE but which can be built on top of most existing databases including network and hierarchical databases.

9. Kernighan, Brian and Ritchie, Dennis, *The C Programming Language*, Prentice-Hall, Inc., 1978.

10. Krishnamurthy, R. and Hochgesang, G.T., "Architecture for a Universal Office System", *4th Jerusalem Conference on Information Technology (JCIT). Next Decade in Information Technology*, 1984, pp. 554-564. The authors propose an architecture using a modified QBE language for a universal office system that is capable of supporting any office language that is casual and unambiguous.

11. Lander, R., Luger, G., and Stibbard, P., "Automated Information Handling in the Newborn Intensive Care Unit of the University of New Mexico Hospital", *Proceedings of the Seventh Annual Symposium on Computer Applications in Medical Care*, 1983, pp. 159-162. The new computer system at the hospital is presented including examples of the menu system, the QBE routines, and the reports the software can generate.

12. Li, D.Y. and Heath, F.G., "ILEX: An Intelligent Relational Database System", *1983 ACM Conference on Personal and Small Computers*, Dec. 1983, pp. 7-9. The authors describe the architectural over-view of a research project called ILEX which includes an interface supporting three typical query languages based on ISBL, SOL, and QBE.

13. *Mistress: The Host Language Interface*, Rhodnius Inc. This manual describes the host language interfaces for Mistress; the shell interface, the standard C interface, and the MX database manipulation procedures.

14. *Mistress: The MR Routines*, Rhodnius Inc. This manual describes the Mistress MR database manipulation procedures

15. *Mistress: The Query Language*, Rhodnius Inc. This manual describes the Mistress structured query language, the executive interface, and the interactive interface.

16. Ozsoyoglu, Z.M. and Ozsoyoglu, G., "Summary-Table-By-Example: A Database Query Language for Manipulating Summary Data", *International Conference on Data Engineering*, 1984, pp. 193-202. STBE is a high-level nonprocedural language similar to QBE which may be used in general purpose and statistical databases to extract and format summary data in a tabular form.

17. *Query Management Facility: Users' Guide and Reference*, IBM Corp. form SC26-4096-2. This manual describes how to use the Query Management Facility's structured query langage (SQL) and query-by-example language (QBE).

18. Reisner, Phyllis, "Human Factors Studies of Database Query Languages: A Survey and Assessment", *Computing Surveys*, vol. 13, no. 1, March 1981, pp. 13-33. The author discusses the different studies which have been conducted in the field of database query languages.

19. Theerachetmongkol, A. and Montgomery, A.Y., "Semantic Integrity Constraints in the Query by Example Data Base Management Language", *Australian Computing Journal*, vol. 12, no. 1, Feb. 1980, pp. 28-42. The authors describe a method of extending QBE to automatically force semantic integrity.

20. Thomas, John C. and Gould, John D., "A Psycological Study of Query by Example", *Proceedings, National Computer Conference*, May 1975, pp. 439-445. The authors discuss a study of QBE designed to determine the time to learn the language and the time, accuracy, and subjects' confidence in translating test questions stated in English into QBE.

21. Weber Systems, Inc. Staff, *C Language User's Handbook*, Ballantine Books, 1984.

22. Zloof, M., "QBE/OBE: A Language for Office and Business Automation", *Computer*, May 1981, pp. 13-22. A unified high-level nonprocedural language is described for the automation of office and business applications. QBE is a subset of OBE.

23. Zloof, M.M., "Query by Example", *Proceedings, National Computer Conference*, May 1975, pp. 431-437. The author introduces the QBE query language for relational databases

24. Zloof, M.M., "Query-by-Example: A Data Base Language", *IBM Systems Journal*, vol. 16, no. 4, 1977, pp. 324-343. The author discusses a high level database management language that provides the user with a convenient and unified interface to query, update, define, and control a database.

6. Appendix 1 - User Manual

Table of Contents

1. Introduction

1.1. General Information

Mistress/QBE (or just QBE) is an alternative method of formulating queries against an existing Mistress database. It can be used to insert records into a table, delete records from a table, change existing records in a table, or produce reports from one or more tables. Most Mistress Query Language commands not directly supported by QBE (such as *create*) can still be executed from QBE to avoid having to leave QBE to execute them. For example, although tables cannot be created directly in QBE, the user can enter the *create* command into QBE. Control will be transferred to the Mistress Interactive Interface where the table will be created and then control will be transferred back to the QBE program.

To use QBE, the user instructs the interface to draw pictorial representations of the tables needed by a query and then fills in the tables to formulate the query. To specify complex record selection criteria, the user can instruct the interface to draw a condition box where the condition is specified in a manner very similar to a Mistress Query Language "where" clause. If the QBE query is to be saved for later execution, the user can also instruct the interface to draw a comment box. Information entered into this type of box does not affect the execution of the query but simply documents what the query does and

any other information the user wishes to save with the query.

1.2. Setting up a Database Directory

A database directory is used to store the tables of a database. The command to set up a database directory is:

msmkdb *dirname*

This will create a UNIX directory called *dirname* and then initialize it to be used with Mistress [1]. *dirname* must be a valid UNIX filename. The directory can then be used with either the Mistress Query Language or Mistress/QBE. However, the user should not execute either form of Mistress from within a database directory to avoid corrupting the database.

1.3. Terminals Supported by QBE

Because QBE needs to check for cursor control characters which vary from terminal to terminal, only a small number of terminals can currently be used to run QBE. These terminals have the following names in the */etc/termcap* database: tvi925; gigi; PC7300; 7300; unixpc; and vt52. If a user tries to run QBE on a terminal with a different name, it will default to tvi925.

1.4. What a QBE Terminal Screen Looks Like

A QBE terminal screen is composed of two portions: a query portion and a command portion. The query portion is

the top part of the screen and is where the tables and boxes are drawn and where the user defines the query. This portion of the screen can consist of several physical pages both vertically and horizontally. Each page is 20 lines long and 80 characters wide. The command portion uses the last four lines of the screen. The first three lines are used for error and other messages from QBE to the user. The last line is where the user types commands to QBE. See figure 1.4a. for an example of a filled in QBE screen after execution of a query which resulted in an error message. In the query portion of the screen, the table and box headings and the bars (¦) are protected. Also, the blank lines between the tables and boxes and below the last box are protected. In the command portion of the screen, only the last line after the *command ==>* is unprotected. However, the last position on the last line is protected to avoid causing the screen to scroll illegally.

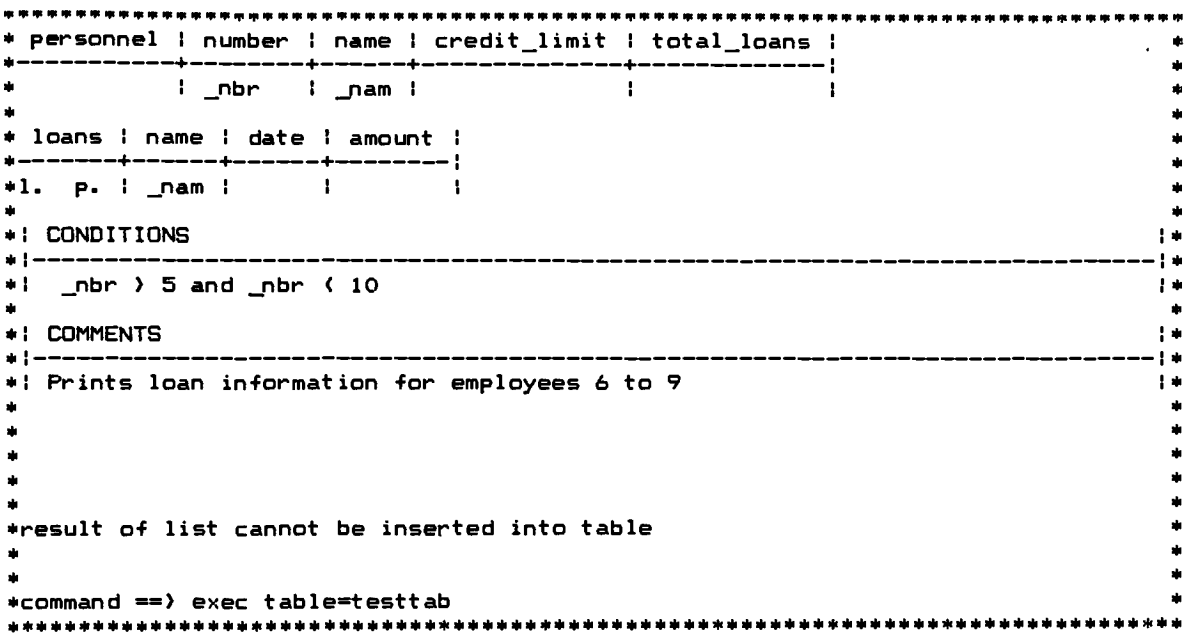


figure 1.4a - sample QBE screen with error message

1.5. Summary of Commands and Operators

The following is a summary of the commands (typed on the command line) and operators (typed within tables) which are valid in QBE. Many of the commands can also be executed by using an abbreviation or an escape sequence (see section 4.)

QBE commands:

draw	- draw a table or box on the screen
delete	- delete a table or box from the screen
clear	- delete all tables and boxes
enlarge	- enlarge a table

reduce - reduce a table

save - save the current query to a file

retrieve - retrieve a query from a file

down - scroll the screen down one page

up - scroll the screen up one page

right - scroll the screen to the right one page

left - scroll the screen to the left one page

help - display the help screen

refresh - redraw the current page correctly

move - move the cursor to a specified position on the current page

exec - execute the current query

stop - exit QBE

Mistress Query Language commands which can be executed from QBE:

display - display information about a database or table

create - create a new table or index

rename - rename a table or attribute

drop - drop a table or index

sort - sort a table

empty - empty a table

insert - insert records into a table from a file

update - update records in a table from a file

database - display current database name or change to a new database

do - execute a UNIX command

! - execute a UNIX command

QBE operators:

i. - insert a record

d. - delete a record or records

u. - update a record or records

p. - print an attribute or record

p(). - print an attribute with priority

ao. - sort in ascending order

ao(). - sort in ascending order with priority

do. - sort in descending order

do(). - sort in descending order with priority

l. - output will be in "list" format

du. - output will be in "dump" format

unq. - only select unique records

gr. - grouping field

gr(). - grouping field with priority

cnt. - count number of selected records

sum. - find the sum of an attribute

avg. - find the average of an attribute

min. - find the minimum of an attribute

max. - find the maximum of an attribute

1.6. Sample Database

The examples used in this manual will use a sample database called *Empdata* which consists of three tables called *personnel*, *loans*, and *name and address*. See tables 1.6a, b, and c for the contents of each of the tables. The tables in this database were chosen to illustrate the concepts of QBE. They are not normalized and contain more redundancy than is necessary. The *personnel* and *loans* tables are based on those used in the Mistress manual [1].

number	name	credit_limit	total_loans
10	Kilroy	$500.00	$250.00
5	Mosca	$750.00	$350.00
17	Wladislaw	$50.00	$50.00
3	Jones	$500.00	$358.95
8	Peterson	$250.00	$50.00
4	Scarlatti	$100.00	$0.00
9	Jordan	$250.00	$0.00

table 1.6a - contents of personnel table

name	date	amount
Mosca	02/02/87	$150.00
Jones	02/07/87	$33.95
Kilroy	02/16/87	$250.00
Wladislaw	02/27/87	$55.00
Jones	04/03/87	$25.00
Mosca	05/04/87	$200.00
Wladislaw	05/12/87	$25.00
Peterson	06/06/87	$50.00
Wladislaw	06/25/87	$75.00
Jones	08/12/87	$300.00

table 1.6b - contents of loans table

number	last_name	first_name	address line 1	address line 2	phone
10	Kilroy	James	1 Jefferson Rd.	Henrietta, NY	426-9681
5	Mosca	Thomas	2 Bailey Rd.	Henrietta, NY	544-2243
17	Wladislav	David	3 Pond Rd.	Holcomb, NY	723-6073
3	Jones	Bradley	4 Rice Rd.	Holcomb, NY	667-2951
8	Peterson	Steven	5 Henrietta Rd.	Henrietta, NY	978-6060
4	Scarlatti	Jeffrey	6 Caulkins Rd.	Henrietta, NY	961-7363
9	Jordan	Michael	7 Clover St.	Henrietta, NY	964-3335

table 1.6c - contents of name and address table

2. How to Invoke and End Mistess/QBE

To invoke Mistress/QBE, enter *qbe* from any UNIX directory which is not also a database directory. The user will then be asked to enter a database name:

Enter database name -

The database name will be checked for validity. If no name is entered the program will end. If the name is invalid, an error message will be displayed and the user will be asked if he wants to quit:

do you want to quit?

If the reply is 'y', the program will end; otherwise the user will be asked to enter the database name again. When a valid database name has been entered, the initial QBE screen will be displayed. This is a blank screen, except for the last line (the command line) which contains:

command ==>

The cursor will be positioned on the command line in the first unprotected position. At this point only the blank part of the command line will be unprotected.

To end QBE, type *stop* on the command line.

3. Naming Conventions

3.1. Table names

Table names in QBE must follow the same conventions as in the Mistress Query Language. They can be either simple or complex. A simple table name can be up to 31 characters in length, must begin with a letter, and consist of only letters, digits, and underscores. A complex name can also be up to 31 characters in length but may contain any printable ASCII characters (except tabs) and must not contain any leading blanks. Complex names must be enclosed in quotes (either single or double) and any quotes within the name must be typed twice. For example, a table name of *shop's data* could be entered as *"shop's data"* or as *'shop''s data'*.

Tables from databases other than the default database must include the database name as part of the table name in the format database:table. For example, if the *shop's data* table were located in a database called *shop.2*, it could be specified as
"shop.2":"shop's data".

3.2. Database and File Names

Database and file names can also be either simple or complex. A simple name can be up to 14 characters long and must not contain any special characters, blanks, or periods. A complex name must be enclosed by quotes and may

contain any characters allowed by UNIX. A complex name may also be a full pathname. In QBE, the maximum length of a file or database name is constrained by the length of the command line, which is 68 characters. Also, no file names beginning with 'qbetab' should be used as these are reserved to QBE.

3.3. **Example Element (Variable) Names**

Example elements are used within QBE tables and boxes to specify implicit and complex conditions. An example element can be up to 18 characters in length and consists of an underscore followed by a string of letters and digits.

4. Special Keys

Some terminal keys have special meaning in QBE. The Return key will cause the cursor to return to the command line and execute whatever command has been typed there.

The arrow keys are used to move the cursor in the indicated direction. If the cursor reaches the edge of the screen, it will wrap around to the opposite side of the screen.

The backspace key will put a space in the current position and then move the cursor back one position. If the current position is protected, QBE will search backwards for the next unprotected position in the current portion of the screen. On tvi925 terminals the backspace key acts exactly like the left arrow key because both keys generate an identical octal code.

The forward tab key will cause the cursor to move forward to the next field in the current portion of the screen. A field is considered to be an unprotected area of the screen separated from other fields by at least one protected position.

The escape key is used to simulate program function keys. To use this feature, press the escape key and then one of the digit keys. (If an invalid key is pressed after an escape key, it will be ignored.) This will have the same effect as typing one of the following commands on the command line:

digit	command
1	exec
2	refresh
3	right
4	left
5	up
6	down
7	delete
8	clear
9	enlarge
0	reduce

If any other unprintable key is pressed, it will be ignored.

5. Executing non-QBE Mistress Commands From QBE

Some Mistress commands which cannot be executed directly by QBE can still be executed from QBE by typing them on the command line. These commands must be typed in exactly as they would be typed in the Mistress Query Language. See the Mistress [1] manual for details. These commands and their effect are as follows:

! - UNIX escape - any UNIX command is typed after the '!' - control will be passed to UNIX, the command will be executed, and control will be passed back to QBE

do - UNIX escape - this works the same as '!' but the UNIX command must be in quotes

create *table* - create a table - control will be passed to the Mistress Interactive Interface where the table will be created and then control will be passed back to QBE

create *table* from *file* - create a table from a file

create index on *table.attr* - create an index

rename *table* to *table* - rename a table or attribute

sort *table* on *attr* - sort a table by an attribute(s)

empty *table* - delete all records from a table

drop *table* - drop a table from the database

drop index on *table.attr* - drop an index

database (or db) - display the name of the current database

database (or db) *dbname* - change to the indicated database

display database - display the tables in the current database

display *table* - display the attributes in a table

insert into *table* from *file* - insert records into a table from a file

update *table* from *file* - update the records in a table from a file

6. Executing Screen Altering Commands

6.1. draw (or dr)

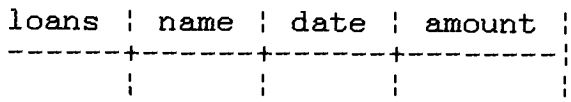
The *draw* command is used to add a pictorial representation of a table or condition or comment box to the screen. There are three forms of this command:

draw *tablename*

draw cond

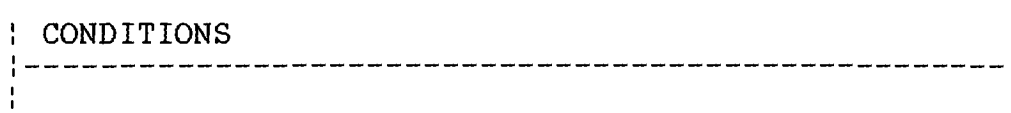
draw comm

The *draw tablename* command is used to add a table to the screen. There will be a column for the table name and a column for each attribute name. For example, the command *draw loans* will produce this:

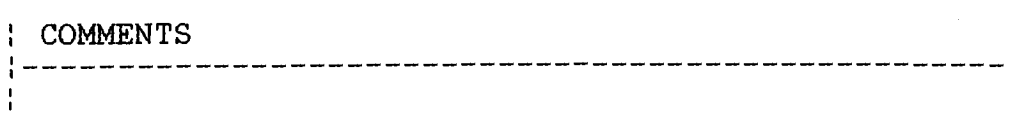


If the table name is complex, it must be enclosed in quotes and follow the rules for typing a complex name (see section 3.1.). Up to 26 tables can be drawn in a single query.

The *draw cond* command adds an empty condition box to the screen:



The *draw comm* command adds an empty comment box to the screen:



Each new table or box is added at the end of the

query. If there is not enough room on the page, the table or box may be split across two pages.

6.2. delete (or de or esc/7)

The *delete* command is used to remove a table or box from the screen. The command is typed on the command line, the cursor is moved to a position within the item to be deleted, and then the return key is pressed. (Or, the cursor can be moved to the item to be deleted and then the esc/7 keys can be pressed.) Any tables or boxes below the deleted item will be moved up to fill in the empty space.

6.3. clear (or cl or esc/8)

The *clear* command is used to delete all tables and boxes from the screen. Type *clear* on the command line or press the esc/8 keys.

6.4. enlarge (or enl or esc/9)

The *enlarge* command is used to insert lines into a table or box and to widen columns within a table or box. To use this command, type *enlarge* on the command line, move the cursor to the desired position, and then press the return key. (Or, move the cursor and then press the esc/9 keys.)

To insert a line into a table, position the cursor within the table name column and on or below the horizontal line. The new line will be added below the line with the

cursor. For example, executing an *enlarge* command when the query portion of the screen looks like this:

```
loans ¦ name ¦ date ¦ amount ¦
------+------+------+--------¦
 p. _ ¦      ¦      ¦        ¦
```

will result in this:

```
loans ¦ name ¦ date ¦ amount ¦
------+------+------+--------¦
 p.   ¦      ¦      ¦        ¦
   _  ¦      ¦      ¦        ¦
```

To insert a line into a condition or comment box, position the cursor on the left vertical bar and on or below the horizontal line. The new line will be added below the line with the cursor.

To widen the table name column, position the cursor within the table name column and above the horizontal line.

To widen an attribute column, position the cursor within the column and on or below the horizontal line.

To widen a condition or comment box, position the cursor within the box and on or below the horizontal line.

In general, a column will be widened eight positions. If this would cause the table or box to overflow onto the next page, the column will only be widened enough to extend the table to the end of the page. If the table or box is already at the end of the page, the column will be widened eight positions and the table or box will be allowed to extend onto the next page.

6.5. reduce (or red or esc/0)

The *reduce* command is used to delete a line or column from a table or box and to reduce the width of a column or box. To use this command, type *reduce* on the command line, move the cursor to the desired position, and press the return key. (Or, move the cursor and then press the esc/0 keys.)

To delete an attribute column from a table, position the cursor within the column and above the horizontal line. For example, executing a *reduce* command when the screen looks like this:

```
loans ¦ name ¦_date ¦ amount ¦
------+------+------+--------¦
      ¦      ¦      ¦        ¦
```

will result in this:

```
loans ¦ name ¦ amount ¦
------¦------¦--------¦
      ¦      ¦        ¦
```

To delete a line from a table, position the cursor under the table name column on the line to be deleted.

To delete a line from a box, position the cursor on the left bar on the line to be deleted.

To reduce the width of the table name column, position the cursor within the table name column and above the horizontal line.

To reduce the width of an attribute column, position the cursor within the column and on or below the horizontal line.

To reduce the width of a box, position the cursor within the box and on or below the horizontal line.

In general, the size of a column will be reduced by eight positions. However, it will not be reduced to a size less than the length of the column name.

6.6. move (or mo)

This command is used to move the cursor to any position on the physical screen. The format of the command is:

move *line#* *col#*

where *line#* is the vertical line on the screen (relative to 0) and *col#* is the horizontal column number (also relative to 0). *line#* cannot be greater than 23 and *col#* cannot be greater than 79. This command has the same effect as using the arrow keys to move the cursor to the desired position but is faster if the cursor is being moved several positions.

6.7. refresh (or ref or esc/2)

This command is used to redraw the screen if it is incorrect. The screen can become incorrect if Mistress displays a message which uses more than the three error lines. The best way to use this command is to press the esc/2 keys because you may not be able to tell where the beginning of the command line is.

6.8. right (or ri or esc/3)
left (or l or esc/4)
down (or dn or esc/6)
up (or esc/5)

The *right* command is used to scroll the physical screen one full page to the right.

The *left* command is used to scroll the physical screen one full page to the left.

The *up* command is used to scroll the physical screen one full page towards the top.

The *down* command is used to scroll the physical screen one full page towards the bottom.

6.9. help (or he)

This command is used to display a screen which summarizes the available commands which can be typed on the command line (both non-QBE Mistress and screen altering) and also shows the commands associated with each *esc/digit* combination. See figure 6.9a.

```
*********************************************************************************
*                          Non-QBE Mistress                                     *
*COMMAND(abbr.)               commands          Esc Sequences                  *
*--------------               ---------------   -------------                  *
*help(he)                     !                 1 = exec                       *
*draw(dr)                     do                2 = refresh                    *
*delete(de)                   sort              3 = right                      *
*save(sa)                     rename            4 = left                       *
*retrieve(ret)                database          5 = up                         *
*down(dn)                     db                6 = down                       *
*up(up)                       drop              7 = delete                     *
*right(ri)                    empty             8 = clear                      *
*left(l)                      insert            9 = enlarge                    *
*enlarge(enl)                 update            0 = reduce                     *
*reduce(red)                  display                                          *
*clear(cl)                    create                                           *
*refresh(ref)                                                                  *
*move(mo)                                                                      *
*exec                                                                          *
*stop                                                                          *
*                                                                              *
*                                                                              *
*                                                                              *
*                                                                              *
*Hit space bar to continue                                                     *
*********************************************************************************
```

figure 6.9a - help screen

6.10. save (or sa)

This command is used to save the current query to a file so that it can be retrieved and executed at a later time. The format is:

save *filename*

If the *filename* is complex, it should be enclosed in quotes. When the save is finished, the message 'query saved' will be displayed. The file will be compressed using the UNIX *compress* command before it is saved and will be stored with a .Z appended to the end of the name.

6.11. retrieve (or ret)

This command is used to retrieve a query which was previously saved to a file. The format is:

retrieve *filename*

If the *filename* is complex, it should be enclosed in quotes. When the retrieve is finished, the message 'query retrieved' will be displayed.

7. Creating and Executing Queries

To create and execute any query, all the tables and boxes must first be drawn and filled in and then the *exec* command must be executed. This can be done by typing *exec* on the command line or by pressing the esc/1 keys.

7.1. Simple Queries

7.1.1. Selecting All Attributes of All Records

To have a report displayed on the terminal which shows all attributes of all the records in a table, simply draw the table and put the p. operator under the table name. For example, to display all the records in the *personnel* table, the query would look like this:

personnel	number	name	credit_limit	total_loans
p.				

and would produce the output shown in table 1.6a from the sample database.

7.1.2. Selecting Some Attributes of All Records

There are two ways to eliminate unwanted attributes from a report. The first way is to delete the unwanted columns from the table after it is drawn and then put the p. operator under the table name. The second way is to leave all the columns in the table and put the p. operator under the columns which are wanted in the report. For example, if a report was desired which showed only the *number* and *name* attributes from the *personnel* table, the query could look like this:

personnel	number	name
p.		

or this:

personnel	number	name	credit_limit	total_loans
	p.	p.		

Both of the above queries would produce the output shown in figure 7.1.2a.

number	name
10	Kilroy
5	Mosca
17	Wladislaw
3	Jones
8	Peterson
4	Scarlatti
9	Jordan

figure 7.1.2a - result of selecting some attributes

7.1.3. Using "list" format

To produce a report in "list" format put the 1. operator under the table name. For example, this query:

```
personnel ¦ number ¦ name ¦
----------+--------+------¦
 p. 1.    ¦        ¦      ¦
```

would produce the output shown in figure 7.1.3a. (For more information on "list" format, see the Mistress [1] manual.)

```
number: 10
name: Kilroy

number: 5
name: Mosca

number: 17
name: Wladislaw

number: 3
name: Jones

number: 8
name: Peterson

number: 4
name: Scarlatti

number: 9
name: Jordan
```

figure 7.1.3a - result of "list" format

7.1.4. Using "dump" format

To produce a report in "dump" format, put the du. operator under the table name. For example, this query:

personnel	number	name	credit_limit	total_loans
du.	p.	p.		

would produce the output shown in figure 7.1.4a. (For more information on "dump" format, see the Mistress [1] manual.)

```
10Kilroy
5Mosca
17Wladislaw
3Jones
8Peterson
4Scarlatti
9Jordan
```

figure 7.1.4a - result of "dump" format

7.1.5. Printing Attributes in Different Orders

The attributes in a report are usually displayed in the same order in which they occur in the table. To change this order, a priority number can be added to the p. operator. The number can be up to three digits long and must be enclosed in parentheses. The numbers used do not have to be consecutive. For example, to print the *personnel* table with *name* first, then *credit_limit*, then *total_loans*, and then *number*, the following query could be used:

```
personnel ¦ number ¦ name ¦ credit_limit ¦ total_loans ¦
----------+--------+------+--------------+-------------¦
          ¦ p(10). ¦ p(1).¦ p(3).         ¦ p(5).       ¦
```

This would produce the output shown in figure 7.1.5a.

name	credit_limit	total_loans	number
Kilroy	$500.00	$250.00	10
Mosca	$750.00	$350.00	5
Wladislaw	$50.00	$50.00	17
Jones	$500.00	$358.95	3
Peterson	$250.00	$50.00	8
Scarlatti	$100.00	$0.00	4
Jordan	$250.00	$0.00	9

figure 7.1.5a - result of selecting in different order

7.1.6. Printing With Different Headings

To change the heading printed on the report for an attribute, add a line to the table with the ch. operator under the table name and put the desired heading under the column(s) to be changed. If the new heading contains any blanks or special characters, it must be enclosed in quotes. For example, to print the *personnel* report using 'emp number' instead of 'number' and 'employee' instead of 'name', the following query could be used:

personnel	number	name	credit_limit	total_loans
p.				
ch.	'emp number'	employee		

This would produce the report shown in figure 7.1.6a.

emp number	employee	credit_limit	total_loans
10	Kilroy	$500.00	$250.00
5	Mosca	$750.00	$350.00
17	Wladislaw	$50.00	$50.00
3	Jones	$500.00	$358.95
8	Peterson	$250.00	$50.00
4	Scarlatti	$100.00	$0.00
9	Jordan	$250.00	$0.00

figure 7.1.6a - result of selecting with different headings

7.2. Sorted Output

Records are usually printed on a report in the order in which they are stored in the table. To change this order, the ao. and do. operators can be used. These operators are put in the attribute column(s) which are to be used to sort the report. To sort in ascending order, use ao.. To sort in descending order, use do.. Optionally, a priority number can be added to each of these operators to specify the high and lower order sort attributes. The priority number can be up to three digits long and must be enclosed in parentheses. The numbers do not need to be consecutive. For example, to sort the *personnel* table by highest to lowest *credit_limit* and then by *number*, the following query could be used:

personnel	number	name	credit_limit	total_loans
p.	ao(2).		do(1).	

This would produce the report shown in figure 7.2a.

number	name	credit_limit	total_loans
5	Mosca	$750.00	$350.00
3	Jones	$500.00	$358.95
10	Kilroy	$500.00	$250.00
8	Peterson	$250.00	$50.00
9	Jordan	$250.00	$0.00
4	Scarlatti	$100.00	$0.00
17	Wladislaw	$50.00	$50.00

figure 7.2a - result of sorting output

7.3. Selecting Unique Output

Usually, a query will print all records which match the specified conditions, even if this would result in duplicate lines. For example, if a report were printed showing only the *name* attribute from the *loans* table, each name would be printed once for each loan. If a person had more than one loan, the name would appear multiple times on the report. To avoid this, the unq. operator can be placed under the table name. This will cause only unique lines to be printed on the report and there will be no duplicates. It will also cause the report to be sorted. For example, this query could be used to print a report of all employees with outstanding loans:

```
loans ¦ name ¦ date ¦ amount ¦
------+------+------+--------¦
 unq. ¦ p.   ¦      ¦        ¦
```

The resulting output is shown in figure 7.3a.

```
name

Jones
Kilroy
Mosca
Peterson
Wladislaw
```

figure 7.3a - result of selecting unique output

7.4. Using Functions

There are five function operators which can be used to produce statistics about a query: cnt., max., min., sum., and avg.. If a function is specified, the report cannot be in "list" format. Also, the p. operator cannot be used in the same query with a function and (usually) only one function can be specified per query.

The max. and min. functions are used to determine the maximum or minimum value of the specified attribute. For example, to find the highest outstanding loan amount, the following query could be used:

```
loans ¦ name ¦ date ¦ amount ¦
------+------+------+--------¦
      ¦      ¦      ¦ max.    ¦
```

This would produce the following output:

$300.00

The sum. and avg. functions are used to find the sum or average of the specified numeric attribute. The unq. operator can also be used with these two functions. If unq. is used, only the non-duplicate attributes will be summed or averaged. For example, the following query would find the total of all outstanding loans:

```
loans ¦ name ¦ date ¦ amount ¦
------+------+------+--------¦
      ¦      ¦      ¦ sum.    ¦
```

The resulting output would be:

$1,163.95

The cnt. function is used to count the number of

selected records. It is usually typed under the table name and can optionally be used with unq. For example, the following query will determine the number of non-duplicate records in the *loans* table:

```
loans          ¦ name ¦ date ¦ amount ¦
---------------+------+------+--------¦
cnt.unq.       ¦      ¦      ¦        ¦
```

The output will be:

```
Number of Records = 10
```

If unq. is used, the cnt. operator can be put in one or more attribute columns instead of in the table name column. This will cause only those records with non-duplicates for the specified attributes to be counted. For example, to find out how many different employees have loans, the following query could be used:

```
loans ¦ name ¦ date ¦ amount ¦
------+------+------+--------¦
 unq. ¦ cnt. ¦      ¦        ¦
```

This would produce the following output:

```
Number of Records = 5
```

7.5. Selecting Into Files and Tables

The results of all queries can be put into a UNIX file instead of being displayed on the terminal and the result of most queries can be inserted into another table. To specify a file name, the *exec* command must be typed on the command line in the following format:

```
exec file=filename
```

If *filename* is complex, it must be enclosed in quotes. If the file already exists, it will be overwritten.

To specify a table name, the *exec* command must be typed on the command line in the following format:

exec table=*tablename*

If *tablename* is complex, it must be enclosed in quotes. If the table does not exist it will be created. If it does exist, the attributes must be compatible with the results of the query. The result of a function cannot be inserted into a table. Also, if the output is in "list" or "dump" format, it cannot be inserted into a table. After the query has finished, the message 'query completed' will be displayed.

7.6. Specifying Conditions

Conditions are used in a query to limit the amount of output produced based on the values of the attributes. Simple conditions can be specified within a table. More complex conditions must be specified in a condition box.

7.6.1. Conditions in the Table

The simplest condition is to specify that an attribute must have a specific value in order for the record to be selected. This is done by typing the value in the column for the attribute. Non-numeric values must be enclosed in quotes. For example, to produce a report showing all the

loans for the employee named Mosca, the following query could be used:

loans	name	date	amount
p.	'Mosca'		

This would produce the report in figure 7.6.1a.

name	date	amount
Mosca	02/02/87	$150.00
Mosca	05/04/87	$200.00

figure 7.6.1a - result of selecting only 'Mosca'

A value can also be preceeded by one of the following operators:

operator	meaning
=	equals (default)
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

To produce a report showing all Mosca's loans which are greater than $150, the following query could be used:

loans	name	date	amount
p.	'Mosca'		> 150

This would produce the report shown in figure 7.6.1b.

name	date	amount
Mosca	05/04/87	$200.00

figure 7.6.1b - result of selecting 'Mosca' and) 150

A value can also be considered to be a pattern and can be preceeded by the following operators:

operator	meaning
match	matches the pattern - ignores differences in case
smatch	matches the pattern - case is significant
!match	does not match the pattern - ignores case differences
!smatch	does not match the pattern - case is significant

(For details on how to construct a pattern, see the Mistress [1] manual.) For example, to find all employees whose names begin with 'J', the following query could be used:

personnel	number	name
p.		smatch 'J*'

This would produce the report shown in figure 7.6.1c.

number	name
3	Jones
9	Jordan

figure 7.6.1c - result of using 'smatch'

Another type of condition which can be specified within a table has the format:

range *constant* [exclusive] to *constant* [exclusive]

This will select all records with an attribute value within the specified range. If 'exclusive' is used after the constant, that constant will not be included in the range. For example, to produce a report of all loans between $100 and $199, the following query could be used:

loans	name	date	amount
p.			range 100 to 200 exclusive

This would produce the report shown in figure 7.6.1d.

name	date	amount
Mosca	02/02/87	$150.00

figure 7.6.1d - result of using 'range'

When two or more conditions are on the same line of a table, then they must all be true for the record to be selected. However, when two conditions are on different lines, then the record will be selected if either condition

is true. For example, to select all loans which are less than $100 or greater than $200, the following query could be used:

loans	name	date	amount
p.			< 100
			> 200

The resulting report is shown in figure 7.6.1e.

name	date	amount
Jones	02/07/87	$33.95
Kilroy	02/16/87	$250.00
Wladislaw	02/27/87	$55.00
Jones	04/03/87	$25.00
Wladislaw	05/12/87	$25.00
Peterson	06/06/87	$50.00
Wladislaw	06/25/87	$75.00
Jones	08/12/87	$300.00

figure 7.6.1e - result of implicit 'or'

The last type of condition which can be used within a table is an implicit equal condition. This type of condition requires the use of example elements. When the same example element is used under two attributes on the same line of the same table, it implies that the two attributes must be equal. For example, to produce a list of all employees whose total loans are equal to their credit limit, the following query could be used:

personnel	number	name	credit_limit	total_loans
	p.	p.	_amt p.	_amt

This would produce the report in figure 7.6.1f.

number	name	credit_limit
17	Wladislaw	$50.00

figure 7.6.1f - result of implicit '='

7.6.2. Conditions in Condition Boxes

A condition box should be used to avoid widening a column to hold a long condition or to refer to the same attribute more than once or to use parentheses in a complex condition to change the order of precedence. Each condition must be contained on a single line in a box. Usually, all the conditions on all the lines in all the condition boxes must be true for a record to be selected. However, if two rows of a condition box or boxes refer to different rows in the same table, then if either condition is true, the record will be selected. A condition in a condition box is written in the same way as a Mistress Query Language "where" clause, except that example elements are used in place of attribute names. For more details on "where" clauses, see the Mistress [1] manual. Each example element used in a condition box must be defined in a table. For example, to print a report of all Mosca's and Jones' loans which are over $100, any of the queries in figures 7.6.2a, b, or c could be used. They would all produce the report shown in figure 7.6.2d.

```
* loans | name          | date | amount |                                                *
*-------+---------------+------+--------|                                                *
*p.     | 'Mosca'       |      | ) 100  |                                                *
*       | 'Jones'       |      | ) 100  |                                                *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*command ==)                                                                            *
****************************************************************************************
```

figure 7.6.2a - query using conditions in table

```
****************************************************************************************
* loans | name          | date | amount |                                                *
*-------+---------------+------+--------|                                                *
*p.     | _nam1         |      | _amt1  |                                                *
*       | _nam2         |      | _amt2  |                                                *
*                                                                                       *
*| CONDITIONS                                                                          |*
*|-------------------------------------------------------------------------------------|*
*| _nam1 = 'Mosca'                                                                     |*
*| _amt1 ) 100                                                                         |*
*                                                                                       *
*| CONDITIONS                                                                          |*
*|-------------------------------------------------------------------------------------|*
*| _nam2 = 'Jones'                                                                     |*
*| _amt2 ) 100                                                                         |*
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*                                                                                       *
*command ==)                                                                            *
****************************************************************************************
```

figure 7.6.2b - query using conditions in table and box

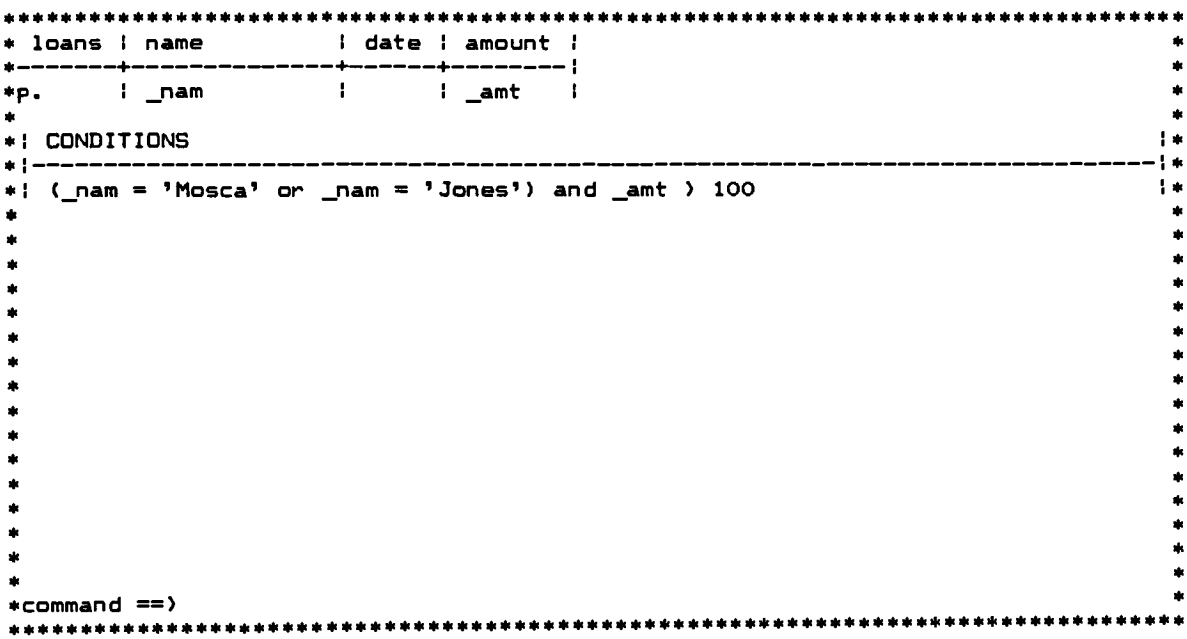


figure 7.6.2c - query using only condition box

name	date	amount
Mosca	02/02/87	$150.00
Mosca	05/04/87	$200.00
Jones	08/12/87	$300.00

figure 7.6.2d - result of query using condition box

7.6.3. Order of Evaluation

Conditions in a query are evaluated in the following order:

1: all implicit conditions on the first line of the first table

2: all other conditions on the first line of the first table

3: all conditions in condition boxes which refer to the first line of the first table

4: all the conditions for the other lines of the first table in the same manner as the first line

5: all the conditions in the other tables in the same manner as the first table

This results in an equivalent Query language "where" clause with the format shown in figure 7.6.3a.

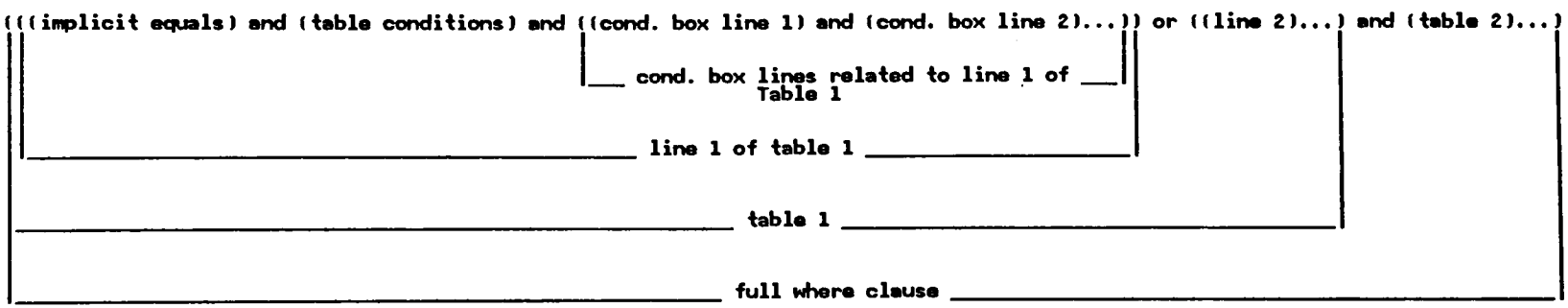


figure 7.6.3a - structure of where clause

7.7. Joining Tables

Two or more tables can be joined together in a query to produce a single report. This is done by using the same example element under compatible attributes of the tables to be joined. For example, to print the name, phone, and credit limit of all employees with a loan taken out after April, 1987, either of the queries in figures 7.7a, or 7.7b could be used. The resulting report is shown in figure 7.7c. If no join condition is specified to link the tables together, then the resulting report will be a merge of the two tables with all possible combinations of the records being printed. This is usually not very useful.

```
*********************************************************************************
* personnel | number | name | credit_limit | total_loans |                     *
*-----------+--------+------+--------------+-------------|                     *
*unq.       |        |_nam  |  p.          |             |                     *
*                                                                               *
* name and address | number | last_name | first_name | phone |                   *
*------------------+--------+-----------+------------+-------|                 *
*                  |        |_nam   p.  |p.          |p.     |                 *
*                                                                               *
* loans | name | date          | amount |                                       *
*-------+------+---------------+--------|                                       *
*       |_nam  | >= 04/01/87   |        |                                       *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*command ==>                                                                    *
*********************************************************************************
```

figure 7.7a - query using a join

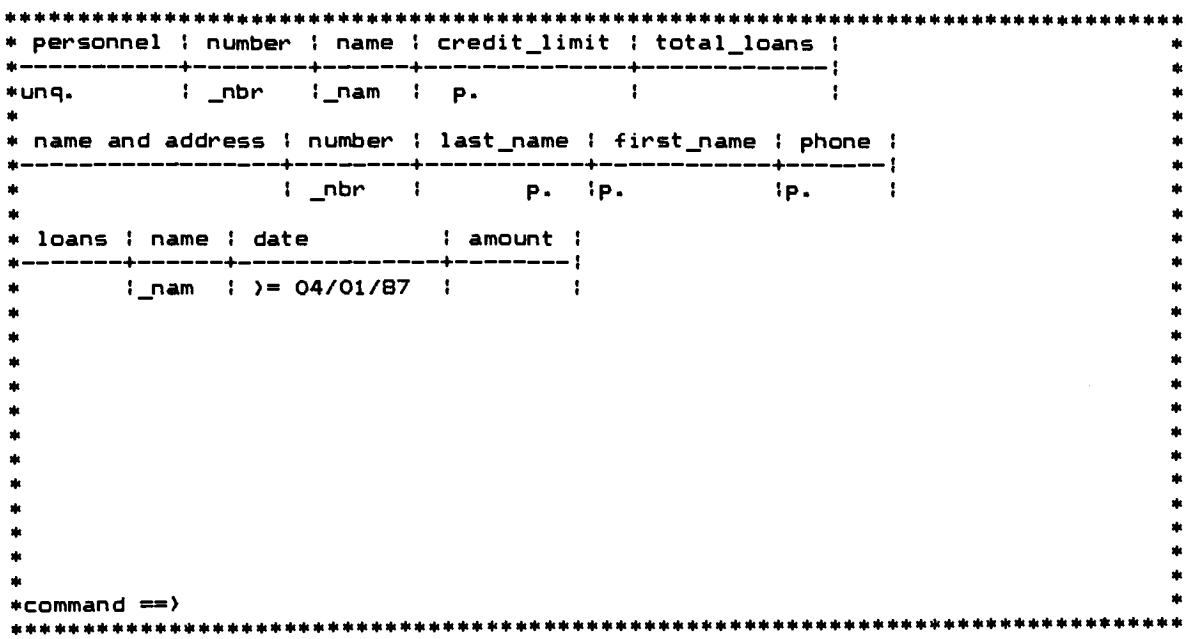


figure 7.7b - query using a join

credit_limit	last_name	first_name	phone
$50.00	Wladislaw	David	723-6073
$250.00	Peterson	Steven	978-6060
$500.00	Jones	Bradley	667-2951
$750.00	Mosca	Thomas	544-2243

figure 7.7c - result of join query

7.8. Grouping

Grouping is an operation which is not supported by the Mistress Query language but is supported by Mistress/QBE. Grouping will print a report showing the result of a function as applied to each value of an attribute or attributes instead of as applied to the table as a whole. For example, grouping could be used to find the maximum loan for each employee instead of for all the employees. To use grouping, type the gr. operator under each attribute by which the report is to be grouped. A priority number can optionally be used with gr. to specify the high and low order attributes. This number can be up to three digits long and must be enclosed in parentheses. The numbers do not have to be consecutive. Type the desired function operator under the appropriate attribute. For example, to produce a report showing the number of loans for each employee, the query in figure 7.8a could be used. The resulting report is shown in figure 7.8b. The sort operators (ao. and do.) are not valid with grouping. Because each grouping operation requires several Mistress commands to be executed, it can be quite slow. A count of the number of lines on the report is displayed on the first error message line as the query is being executed.

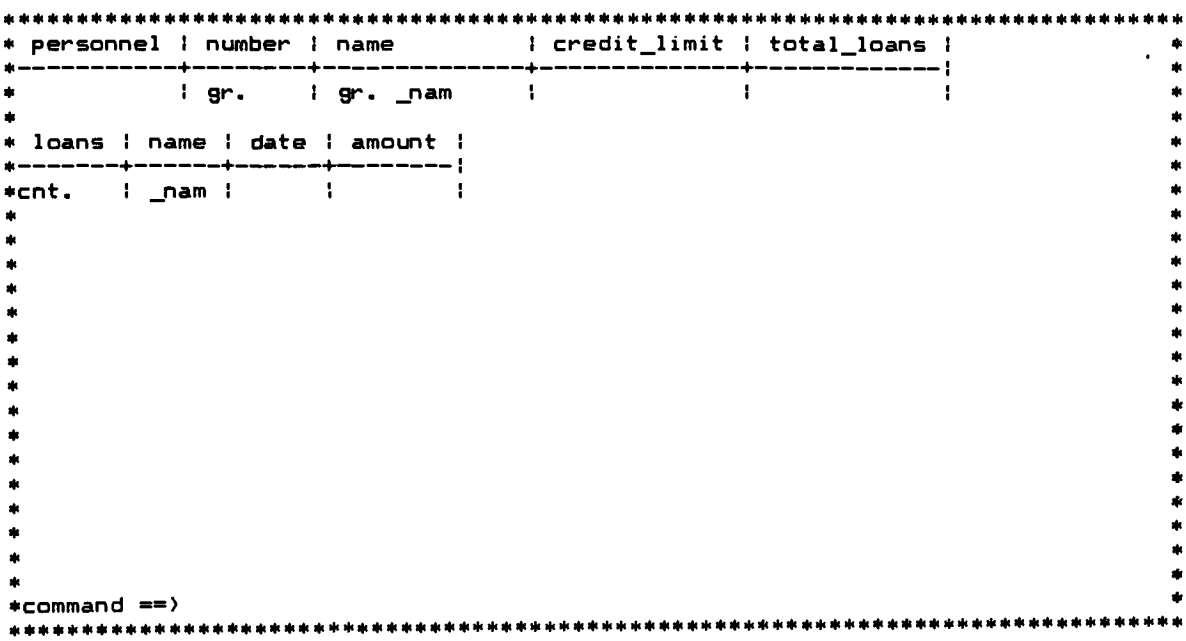


figure 7.8a - query using grouping

number	name	record count
3	Jones	3
5	Mosca	2
8	Peterson	1
10	Kilroy	1
17	Wladislaw	3

figure 7.8b - result of grouping query

7.9. Inserting Records Into a Table

To insert a record into a table, put the i. operator under the table name and the information to be inserted in the appropriate attribute columns. Non-numeric data must be enclosed in quotes if it contains any blanks or special characters. Multiple records can be inserted into multiple tables using one query. A count of the number of lines processed will appear on the first error message line as the query is being executed. The queries in figures 7.9a, b, and c could have been used to load the data into the sample database. When all lines have been processed, the message 'insert completed' will be displayed.

```
*********************************************************************************
* personnel | number | name        | credit_limit | total_loans |                *
*-----------+--------+-------------+--------------+-------------+                *
* i.        | 10     | Kilroy      | 500          | 250         |                *
* i.        | 5      | Mosca       | 750          | 350         |                *
* i.        | 17     | Wladislaw   | 50           | 50          |                *
* i.        | 3      | Jones       | 500          | 358.95      |                *
* i.        | 8      | Peterson    | 250          | 50          |                *
* i.        | 4      | Scarlatti   | 100          | 0           |                *
* i.        | 9      | Jordan      | 250          | 0           |                *
*                                                                               *
* name and address | number | last_name | first_name | address line 1             *
*------------------+--------+-----------+------------+--------------------------*
* i.               | 10     | Kilroy    | James      | '1 Jefferson Rd.'        *
* i.               | 5      | Mosca     | Thomas     | '2 Bailey Rd.'           *
* i.               | 17     | Wladislaw | David      | '3 Pond Rd.'             *
* i.               | 3      | Jones     | Bradley    | '4 Rice Rd.'             *
* i.               | 8      | Peterson  | Steven     | '5 Henrietta Rd.'        *
* i.               | 4      | Scarlatti | Jeffrey    | '6 Caulkins Rd.'         *
* i.               | 9      | Jordan    | Michael    | '7 Clover St.'           *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*command ==>                                                                    *
*********************************************************************************
```

figure 7.9a - left hand page of first insert screen

```
*******************************************************************************
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*      ¦ address line 2          ¦ phone          ¦                          *
*------+-------------------------+----------------¦                          *
*      ¦'Henrietta, NY'          ¦ 426-9681       ¦                          *
*      ¦'Henrietta, NY'          ¦ 544-2243       ¦                          *
*      ¦'Holcomb, NY'            ¦ 723-6073       ¦                          *
*      ¦'Holcomb, NY'            ¦ 667-2951       ¦                          *
*      ¦'Henrietta, NY'          ¦ 978-6060       ¦                          *
*      ¦'Henrietta, NY'          ¦ 961-7363       ¦                          *
*      ¦'Henrietta, NY'          ¦ 964-3335       ¦                          *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*command ==>                                                                  *
*******************************************************************************
```

figure 7.9b - right hand page of first insert screen

```
*******************************************************************************
* loans ¦ name          ¦ date          ¦ amount ¦                            *
*-------+---------------+---------------+--------¦                            *
* i.    ¦ Mosca         ¦ 02-02-87      ¦ 150    ¦                            *
* i.    ¦ Jones         ¦ 02-07-87      ¦ 33.95  ¦                            *
* i.    ¦ Kilroy        ¦ 02-16-87      ¦ 250    ¦                            *
* i.    ¦ Wladislaw     ¦ 02-27-87      ¦ 55     ¦                            *
* i.    ¦ Jones         ¦ 04-03-87      ¦ 25     ¦                            *
* i.    ¦ Mosca         ¦ 05-04-87      ¦ 200    ¦                            *
* i.    ¦ Wladislaw     ¦ 05-12-87      ¦ 25     ¦                            *
* i.    ¦ Peterson      ¦ 06-06-87      ¦ 50     ¦                            *
* i.    ¦ Wladislaw     ¦ 06-25-87      ¦ 75     ¦                            *
* i.    ¦ Jones         ¦ 08-12-87      ¦ 300    ¦                            *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*                                                                             *
*command ==>                                                                  *
*******************************************************************************
```

figure 7.9c - second insert screen

7.10. Deleting Records From a Table

Records can be deleted from a table by putting the d. operator under the table name and specifying a condition which determines which records are to be deleted. Multiple delete lines for multiple tables can be specified using one query. A count of the number of lines processed will appear on the first error message line as the query is being executed. The query in figure 7.10a could be used to delete all references to Mosca and Jones from the sample database. When all lines have been processed, the message 'delete completed' will be displayed.

```
* personnel | number | name        | credit_limit | total_loans |
*-----------+--------+-------------+--------------+-------------|
*d.         |        | 'Mosca'     |              |             |
*d.         |        | 'Jones'     |              |             |
*
* loans | name         | date | amount |
*-------+--------------+------+--------|
*d.     | 'Mosca'      |      |        |
*d.     | 'Jones'      |      |        |
*
* name and address | number | last_name | first_name |
*------------------+--------+-----------+------------|
*d.                |        | 'Mosca'   |            |
*d.                |        | 'Jones'   |            |
*
*command ==> dump
```

figure 7.10a - delete query

7.11. Updating Records In a Table

Records can be updated in a table by putting the u. operator with the new data under each attribute to be changed and by specifying a condition to determine which records are to be updated. If a condition refers to an attribute which is being updated, an example element and condition box must be used. Multiple update lines for multiple tables can by specified using one query. A count of the number of lines processed will appear on the first error message line as the query is being executed. The query in figure 7.11a could be used to change all occurrences of the name Kilroy to Smith and all occurrences of the name Jordan to James in the sample database. When all lines have been processed, the message 'update completed' will be displayed.

```
* personnel | number | name          | credit_limit | total_loans |
*-----------+--------+---------------+--------------+-------------|
*           |        |u. Smith _nam1|              |             |
*           |        |u. James _nam2|              |             |
*
* loans | name          | date | amount |
*-------+---------------+------+--------|
*       |u. Smith _nam1|      |        |
*       |u. James _nam2|      |        |
*
* name and address | number | last_name          | first_name |
*------------------+--------+--------------------+------------|
*                  |        |u. Smith _nam1      |            |
*                  |        |u. James _nam2      |            |
*
*| CONDITIONS                                                                    |
*|-------------------------------------------------------------------------------|
*| _nam1 = 'Kilroy'                                                              |
*| _nam2 = 'Jordan'                                                              |
*
*
*
*
*command ==> dump
```

figure 7.11a - update query

8. A Sample Complex Query Using Multiple Pages

This section will show step by step the commands necessary to construct a query which will print a report showing employee and loan information for the employees named Peterson and Wladislaw. It will only print loans taken out between February and July 1987. The query will require all three tables from the sample database and a condition box. It will also include a comment box at the top to document the query.

step 1: draw comm (draw the comment box)

step 2: move 2 0 (move the cursor to the comment box)

step 3: esc/9 (add a line to the comment box)

step 4: fill in the comment box and press the return key

step 5: draw personnel (draw the *personnel* table)

step 6: move the cursor to the personnel table and position it under the table name column and press the esc/9 keys (add a line to the table)

step 7: draw 'name and address' (draw the *name and address* table)

step 8: draw loans (draw the *loans* table)

step 9: draw cond (draw the condition box - the first 2 lines will be on page 1 and the last line will be on page 2 - you will be looking at page 2)

step 10: move 0 0 (move the cursor to the condition box)

step 11: esc/9 (add a line to the condition box)

At this point the query will consist of four pages - two across and two down. The upper left page will look like figure 8a. The upper right page will look like figure 8b and the lower left page will look like figure 8c. The lower right page will be blank.

step 12: esc/5 (scroll the page up to the upper left page)

step 13: put a p. under the table name of the *personnel* table

step 14: move the cursor to the name column and press esc/9 to widen the column

step 15: fill in the first line of the name column with _nam'Peterson'

step 16: fill in the second line of the name column with 'Wladislaw'

step 17: move to the last name column of the *name and address* table and fill it in with _nam

step 18: put p. under the address and phone columns

step 19: move to the name column of the *loans* table and fill it in with _nam

step 20: put p. under the date and amount columns

step 21: put _dat under the date column

step 22: esc/6 (scroll down to the second page)

step 23: move to the first line of the conditon box and fill it in with _dat >= '01/01/87'

step 24: move to the second line of the condition box and fill it in with _dat <= '07/31/87'

At this point, the three non-blank pages of the query will look like those in figures 8d, e, and f.

step 25: esc/1 (execute the query)

The resulting report is shown in figure 8g.

```
*****************************************************************************
*| COMMENTS                                                                 |*
*|--------------------------------------------------------------------------|*
*| This query will print a report showing employee and loan information for |*
*| Peterson and Wladislaw for Feb. to July 1987                             |*
*                                                                           *
* personnel | number | name | credit_limit | total_loans |                  *
*-----------+--------+------+--------------+-------------|                  *
*           |        |      |              |             |                  *
*           |        |      |              |             |                  *
*                                                                           *
* name and address | number | last_name | first_name | address line 1 | address 1*
*------------------+--------+-----------+------------+----------------+----------*
*                  |        |           |            |                |          *
*                                                                           *
* loans | name | date | amount |                                            *
*-------+------+------+--------|                                            *
*       |      |      |        |                                            *
*                                                                           *
*| CONDITIONS                                                              |*
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*command ==>                                                                *
*****************************************************************************
```

figure 8a - upper left page of query before filling in tables

```
*****************************************************************************
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*ine 2 | phone |                                                           *
*------+-------|                                                           *
*      |       |                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*                                                                           *
*command ==>                                                                *
*****************************************************************************
```

figure 8b - upper right page of query before filling in tables

```
*command ==>
```

figure 8c - lower left page of query before filling in tables

```
*| COMMENTS
*|-------------------------------------------------------------------------|*
*| This query will print a report showing employee and loan information for |*
*| Peterson and Wladislaw for Feb. to July 1987                             |*
*
* personnel | number | name          | credit_limit | total_loans |
*-----------+--------+---------------+--------------+-------------|
*p.         |        |_nam'Peterson'|              |             |
*           |        | 'Wladislaw'  |              |             |
*
* name and address | number | last_name | first_name | address line 1 | address l*
*------------------+--------+-----------+------------+----------------+----------*
*                  |        | _nam      |            | p.             | p.       *
*
* loans | name | date | amount |
*-------+------+------+--------|
*       | _nam |p._dat| p.     |
*
*| CONDITIONS                                                              |*
*|-------------------------------------------------------------------------|*
*
*command ==>
```

figure 8d - upper left page of query after filling in tables

figure 8e - upper right page of query after filling in tables

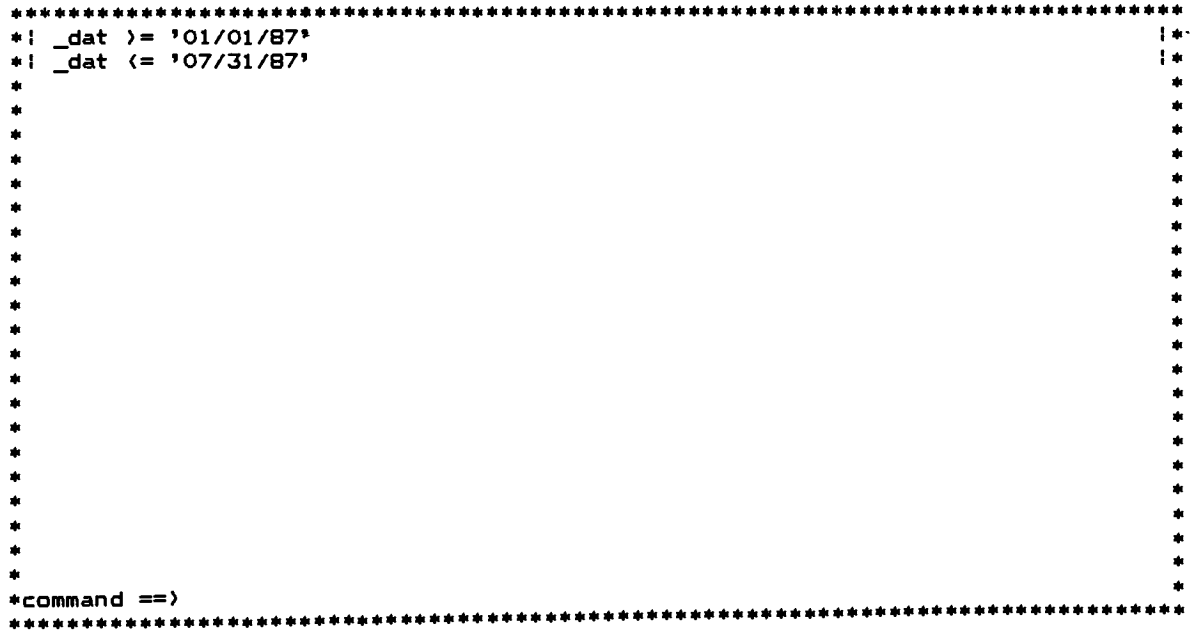


figure 8f - lower right page after filling in tables

number	name	credit_limit	total_loans	address line 1	address line 2	phone	date	amount
17	Wladislav	$50.00	$50.00	3 Pond Rd.	Holcomb, NY	723-6073	02/27/87	$55.00
17	Wladislav	$50.00	$50.00	3 Pond Rd.	Holcomb, NY	723-6073	05/12/87	$25.00
17	Wladislav	$50.00	$50.00	3 Pond Rd.	Holcomb, NY	723-6073	06/25/87	$75.00
8	Peterson	$250.00	$50.00	5 Henrietta Rd.	Henrietta, NY	978-6060	06/08/87	$50.00

figure 4g - result of executing query

9. Error Messages

There are two kinds of error messages produced by this program: those displayed by Mistress and those displayed by QBE. The Mistress messages have the format:

*** user error *** explanation

Mistress errors are generated when an error is detected that could not be detected by QBE. An example of this type of error would be if the user specified an implicit equal condition between incompatible attributes. The QBE messages have no set format but are quite explicit so there should be no problem determining what the problem is. The cursor will be positioned as close to the error as possible. The following is a list of all the QBE messages: (comments are in parentheses)

invalid command (on command line)
no table name specified
no file name specified
missing database name before :
table not found
file not found
cannot open file
invalid name - unmatched quote (see section 3)
name too long (see section 3)
table name too long (see section 3)
no more pages (tried to scroll too far)
cursor not in table or box
cursor not on valid line
cursor not on or below line
cursor not in valid column
no room to reduce column
cannot delete this line
cannot delete last line in table
invalid row number (on *move* command)
invalid column number (on *move* command)
too many tables (maximum is 26)
no insert, delete, update, or select (specified in query)

field is blank (on update operation)
no characters allowed under table name for update
extra characters under table name
extra characters under *fieldname*
all lines must begin with i. (for insert)
all fields are blank (for insert)
all lines must begin with d. (for delete)
no condition specified (for delete)
function not allowed with p.
function required with grouping
grouping not allowed with p.
sorting not allowed with grouping
function not allowed with 1.
result of function cannot be inserted into table
result of 1. or du. cannot be inserted into table
ch. cannot be only line in table
only one ch. line allowed per table
conflicting styles - 1. and du.
p. under both table and field names
with unq., sort field must be selected for print
illegal print number (must be (0) - (999))
cnt. under both table and field names
cnt. under field name requires unq.
unq. not allowed with max. or min.
function not allowed in same field with gr.
illegal sort number (must be (0) - (999))
invalid operation under *fieldname*
variable not defined
variable name too long

10. References

1. *Mistress: The Query Language*, Rhodnius, Inc