

A GRAPHICAL ALTERNATIVE TO DIRECT  
SQL BASED QUERYING

by

Johnita Beasley


Project/Report submitted to the Faculty of the  
Virginia Polytechnic Institute and State University  
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science and Applications

APPROVED:

  
\_\_\_\_\_  
C.J. Egyhazy, Chairman  
\_\_\_\_\_  
W.B. Frakes  
\_\_\_\_\_  
L.H. Crawford

May, 1993  
Blacksburg, Virginia

C.2

LD  
5655  
V851  
1993  
B437  
C.2

## 1.0 EXECUTIVE SUMMARY

SQL provides a fairly straightforward means of querying database data. However, as with all command languages, SQL can get very complicated, even for experienced programmers. This complexity can be intimidating to the novice or intermediate user who needs to access data from a database with complex SQL statements, especially when users don't want to know or even become familiar with a command oriented query language like SQL.

One application that has been developed to simplify database querying is Query-by-Example (Zloof, 1975). Query-by-Example is a database front-end that applies direct manipulation to database tables on the user display in a text-based format. The idea behind direct manipulation is to make objects and actions of interest visible and to replace complex command-language syntax with direct manipulation of the objects of interest (Shneiderman, 1992).

Graphical Query Language (Andyne, 1992) is another application that has been developed to simplify database querying. It applies direct manipulation for database querying in a graphical environment.

The goal of this project/report is to design a database front-end application based on techniques similar to those used in Query-by-Example and Graphical Query Language(GQL). This database front-end will be used for querying a SQL based database - An application that acts as an interface to SQL querying. This application is geared toward novice or intermediate users. Novice and intermediate users can be classified as non-programmer user types. Their interaction with a computer is in an operational capacity. The difference between the novice and intermediate user is the intermediate users familiarity with various application operations. Therefore, the application interface needs to be simple. Applying graphical direct manipulation techniques, like those used in GQL, is the key in creating a simple interface. This report also discusses GQL and how the proposed system addresses some of the inadequacies of this application.

The proposed database front-end application is called QUERI-EZ. Direct manipulation concepts are applied to the design of QUERI-EZ's interface to

provide a visually simple means of querying the database, while the behavior of the back-end of the application is expressed by applying structured design techniques/concepts. The proposed means of testing the QUERI-EZ application interface is also established.

## TABLE OF CONTENTS

1.0 EXECUTIVE SUMMARY .....	ii
2.0 INTRODUCTION .....	1
3.0 SQL BASED QUERYING .....	2
3.1 Brief History of SQL .....	2
3.2 Overview .....	3
3.3 Advantages and Disadvantages of Direct SQL Based Querying .....	4
4.0 HUMAN FACTORS .....	6
4.1 The User-Centered Design Process .....	6
4.2 Direct Manipulation .....	9
5.0 EXISTING SYSTEMS .....	11
5.1 Query-by-Example .....	11
5.1.1. Commentary .....	12
5.2 Visual Query Language .....	12
5.2.1 Commentary .....	13
5.3 Graphical Query Language .....	14
5.3.1 Commentary .....	17
6.0 PULLING IT ALL TOGETHER .....	19
7.0 A PROPOSED SYSTEM DESIGN .....	21
7.1 Introduction .....	21
7.2 System Specification .....	21
7.2.1 Scope .....	21
7.2.2 Requirements .....	21
7.2.2.1 Functional Description .....	21
7.2.2.2 Interface Definition .....	22
7.2.3 Characteristics .....	22
7.3 Software Requirements Specification .....	24
7.3.1 Scope .....	24
7.3.2 Functional Requirements .....	24
7.3.2.1 Description of Function .....	24
7.3.3 User Interface Requirements .....	25
7.3.3.1 Description of Interface .....	25
7.3.4 User Requirements .....	25
7.4 System Design .....	26
7.4.1 User Interface Design .....	26
7.4.1.1 Interface Presentation .....	26
7.4.1.2 Scenarios .....	30
7.4.1.2.1 The "Suppliers-Parts" Database .....	30
7.4.1.2.2 Scenario 1: Creating a Simple Query .....	31
7.4.1.2.3 Scenario 2: Retrieving an Existing Query .....	39
7.4.1.2.4 Scenario 3: Creating a Complex Query .....	42
7.4.1.3 State Transition Diagram .....	47
7.4.1.4 User Action Notations .....	49
7.4.1.5 User Interface Verification/Evaluation .....	56

7.4.2 Software Design .....	62
7.4.2.1 Context Diagram .....	62
7.4.2.2 Data Flow Diagram .....	62
7.4.2.3 Pseudocode Algorithms .....	64
7.4.2.3.1. General Pseudocode Algorithms .....	64
7.4.2.3.2 Pseudocode Algorithms for the Requirements .....	69
8.0 CONCLUSIONS .....	76
8.1 Summary .....	76
8.2 Evaluation of the Application Design .....	76
9.0 BIBLIOGRAPHY .....	81

## 2.0 INTRODUCTION

This project/report is a combination of three very significant areas in the computer science/information science industry. These areas include Database Management Systems (particularly, the means of accessing the data within them), Human-Computer Interaction (HCI), and Software Engineering.

The means by which database data is accessed is the targeted problem area. It is a very common and necessary operation performed by users at all levels. However, most current database access applications ignore the needs and abilities of a very important group of users - the novice/intermediate users. The concepts of HCI allow the system designer to remember the user when designing interactive applications. In particular, HCI concepts concentrate on the user interface and interactive features of an application.

This project/report discusses the advantages and disadvantages of accessing data through direct SQL based querying, and addresses the concepts of HCI as they apply to the problems surrounding direct SQL based querying. This project/report also presents some current systems that are attempting to address the problem area and comments on why it is felt that they do or do not resolve the areas of complexity. Given the concepts of HCI and the inadequacies of some current systems, interface engineering and software engineering techniques are applied to the direct SQL based querying to propose a database front-end application that is a graphical alternative to this querying approach.

## 3.0 SQL BASED QUERYING

Data is the basis of information systems and it is crucial that data is easily accessible. Many of today's information systems are developed for users whose knowledge and understanding of computers is basic. Therefore, the application with which they interact to retrieve data must be intuitive and easy to understand and operate.

Relational Database Management Systems (RDBMS's) are popular for storing data, and SQL is the means by which relational database data is created, accessed, and manipulated. SQL acts as an interface between the user and the data. Hence, the question arises: Does direct SQL based querying provide the novice/intermediate user with an easily understandable means of accessing database data? This section provides a brief history of SQL, a high level overview of the language, and a discussion of the advantages and disadvantages of direct SQL based querying, in order to address this question.

### 3.1 Brief History of SQL

SQL is the relational language on which most of today's Relational Database Management Systems (RDBMSs) are based. The language was first defined in San Jose, California by D.D. Chamberlain and others at the IBM Research Laboratory. The first prototype implementation was known under the name "System R", and it was built at the IBM San Jose Laboratory. Based on a stern set of usability and performance tests that were performed both inside and outside of IBM, the decision was made to develop a family of products based on the technology of System R. Today, the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) have both standardized a dialect of SQL. This dialect is the "official" interface to relational systems.

### 3.2 Overview

SQL is a data sub-language which consists of two distinct parts: a Data



Definition Language (DDL) and a Data Manipulation Language (DML).

The DDL is used to define the structure of a database, and the principle statements associated with it are as follows: CREATE TABLE, CREATE VIEW, CREATE INDEX, ALTER TABLE, DROP TABLE, DROP VIEW, DROP INDEX.

The DML is used to access and manipulate data within a database. There are four DML statements provided by SQL: SELECT, UPDATE, DELETE and INSERT.

There is also another aspect of SQL that doesn't fall under the DML or the DDL which is related to data security. There are two statements provided by SQL for this purpose: GRANT and REVOKE. These statements apply to user data access privileges.

In terms of data retrieval, the SELECT statement of the DML is its supporting construct. The general form of the SQL SELECT statement that this project/report will focus on is shown in Figure 3.2.1.

```
SELECT [DISTINCT] item(s)
FROM table(s)
[WHERE condition(s)]
[GROUP BY field(s)]
[HAVING condition]
[ORDER BY field(s)];
```

**Figure 3.2.1. General Form of SQL  
SELECT Statement**

The brackets indicate optional clauses of the statement, and the capitalized words are SQL keywords applicable to the SELECT statement (not exhaustive). As can be seen from Figure 3.2.1, the SELECT statement allows:

- 1) Simple retrievals,
- 2) Qualified retrievals (WHERE and HAVING clauses),

- 3) Retrieving with ordering (ORDER BY clause),
- 4) Retrieving with grouping (GROUP BY clause), and
- 5) Retrievals involving table joins (Two or more tables).

Less apparent from the general form of Figure 3.2.1, is the fact that sub-querying with multiple levels of nesting is also provided. This project/report refers to queries of this type as complex queries.

### 3.3 Advantages and Disadvantages of Direct SQL Based Querying

With all of the capabilities provided in the SELECT statement alone, the robustness of SQL in retrieving data is apparent. Many variations of the SELECT statement clauses can be combined to obtain desired results. Obviously, the fewer clauses utilized, the simpler the statement is to construct. Similarly, the more clauses applied to a SELECT statement, the more complex the SQL statement becomes. And, because many relational database structures can be fairly large and therefore complex, the SELECT statements required to retrieve data tend to be large and confusing. In effect, the more complex the database, the more complex the queries become.

With SQL acting as the interface for the user to the database, the user (experienced programmer or novice user) is faced with the challenge of constructing very complex SQL statements to access the data needed. For the experienced programmer, this challenge may be welcomed. However, for the novice or intermediate user who simply wants to see results and is not interested in what the SQL statement looks like, this challenge is by no means welcomed. This is the essence of the "good and bad" of direct SQL based querying. SQL based querying can be relatively simple however, it can also be extremely complex. Even in its simplest form, it is not intuitive because it requires the user to remember or have access to command names and command syntax. Therefore, direct SQL based querying does not always provide the novice/intermediate user with an intuitive and easily understandable means of accessing database data. Hence, the challenge to a systems designer is to provide a means by which the user can produce SQL data

retrieval statements (both simple and complex) without directly interfacing with SQL.

Table 3.3.1 summarizes the advantages and disadvantages of direct SQL based querying. It is important to note that the advantages and disadvantages of direct SQL based querying cited in this project/report are based on uncontrolled observations of a small group of novice/intermediate users and the opinions of the author.

**Table 3.1.1. Summary of SQL Advantages and Disadvantages**

Advantages	Disadvantages
Robust	not intuitive
Can be simple	command names and syntax to learn
	level of difficulty of command is based on the underlying database structure.

## 4.0 HUMAN FACTORS

As seen in the previous section, direct SQL based querying does not take into account the human factors related to interactive applications. These factors include an individual's ability to perceive concepts, solve problems, make decisions, recollect, and learn. "The remarkable diversity of human abilities, background, motivations, personalities, and work styles challenge the interactive system designers."<sup>10</sup> The interactive interface provides the means by which an end-user will interact with an application. Therefore, regardless of the abundance of functionality provided in an application, if the user cannot understand how to use the application, or is intimidated by the application, the application will not be successful. This brings about another point in interface and system design. Generally, the designer is thinking that the more functionality provided in a system, the easier it is to use. As a result, the interface becomes more complex. Further, personal capabilities vary and therefore, a system that is user-friendly to one person, may not be user-friendly to another; pleasing everyone is virtually impossible. A designer is, however, more likely to design a well received system if the design is centered around the user.

### 4.1 The User-Centered Design Process

User-Centered design (Norman, 1986) implies that the user is the focal point in the interactive system design. The user-centered design process is an iterative process as is the software development lifecycle, and the key to its success is the user.

The first step is to understand what the user requires from the system. Once an understanding has been obtained, an initial design of the system can be formulated, and a prototype can be produced. It is the prototype of the interactive interface which gives the user a clear understanding of the system and initiates the iterative behavior of the design process.

A common way for a prototype to be presented to the user is in the form of a usability lab. In a usability lab, the user interacts with the system and is studied

by an observer or a group of observers who analyze the user's reaction to the system. Quantitative areas such as time required to complete a task, the number of keystrokes that are required, the number of errors that are generated, the number of mouse clicks that are required, etc. can be used to evaluate the usability of a system when compared to expected values. This data can be used as feedback as changes are made to the interface to meet the user's needs.

The usability lab not only allows the designer to analyze the user's reaction to the system but, it gives the user the ability to visualize what was requested and how well the designer understood the request. This lends itself to the fact that the user is not always sure of what he/she wants. Therefore, as the user becomes more familiar with the system and what he/she wants, the user interface changes. And since the interface may change many times, it becomes apparent that there needs to be a means of separating (i.e. giving independence to) each major entity within the software system; in particular, separating the interface from the functional implementation. This would assist in a better design of the system interface because changes are not directly tied to the underlying software. The result is easier interface modification.

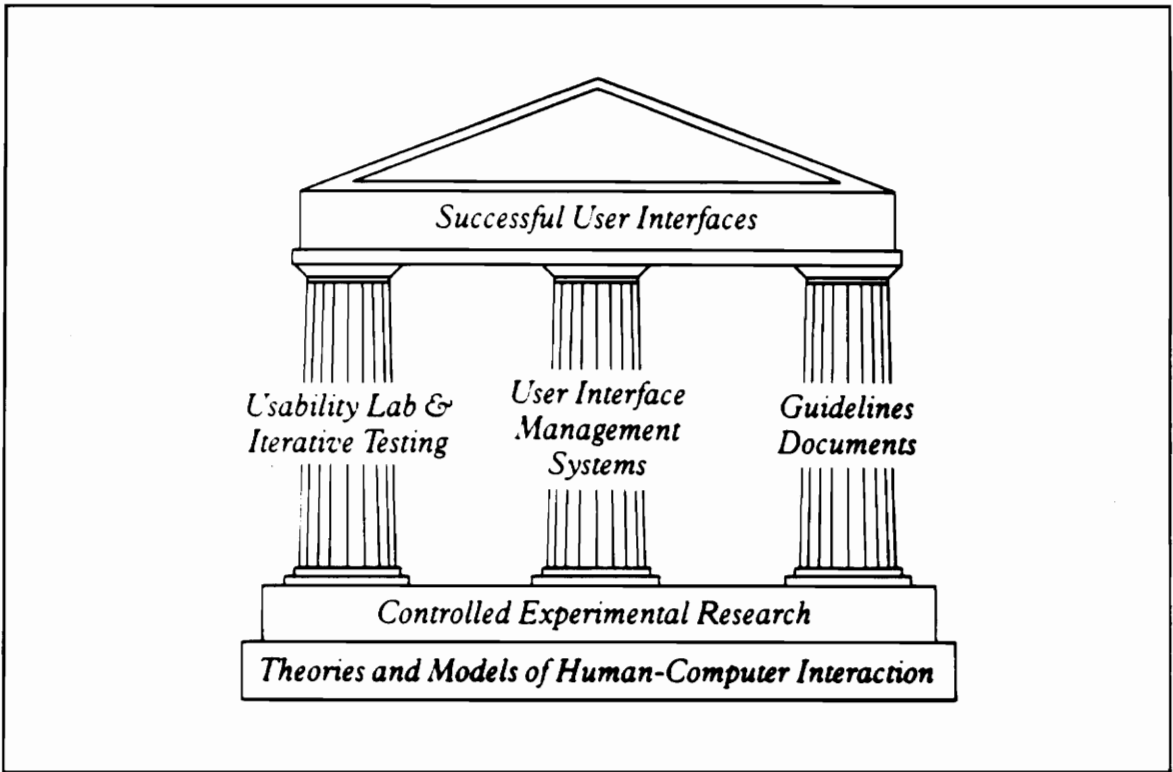
The idea of independence has been implemented with respect to DBMS's and the concept of **data independence**. Paralleling this concept is the User Interface Management System (UIMS) and the concept of **user interface independence** or **dialogue independence**. In DBMS's, data independence is a major objective and can be thought of as the ability of an application to be unconcerned with particular storage structures or access techniques. Similarly, in UIMSs, the goal is to separate the logical design of the user interface and the interface itself from the implementation of the application.

Another prevalent entity used in the user-centered design process is the Guidelines Document. This document is a means by which the user interface architect defines the standards/principles that application developers are to follow to ensure consistency in design across all products.

In all, a successful user interface design is based on a design process centered

around the system users. As discussed, this process is supported by usability labs, UIMS's, and Guidelines Documents. Ben Schneiderman refers to these areas of support as pillars that rest on the foundation of controlled experimental research and the theories and models of human-computer interaction. Figure 4.1.1 (from Schneiderman, 1992) graphically depicts this concept.

Even when a design is centered around the user, there still must exist concern for the method of presentation of the associated system interface. There are many types of well designed interactive systems whose interfaces can be classified as: forms based, menu driven, command driven, and direct manipulation oriented. The



**Figure 4.1.1. Three Pillars of a Successful Interface**

direct manipulation oriented systems are very popular across all user levels as discussed in the following section.

## 4.2 Direct Manipulation

The promise of direct manipulation as it relates to the user is that interactive interfaces graphically match the way the user thinks about a problem and/or the user's work style. The actions are performed by manipulating icons or pressing buttons. There are no hidden operations and no syntax or command names to learn.

Ben Shneiderman coined the term "Direct Manipulation" (see Shneiderman, 1992), and he focuses on three principles of direct manipulation:

- " 1) Continuous representation of the objects of interest
- 2) Physical actions or presses of labelled buttons instead of complex syntax
- 3) Rapid incremental reversible operations whose effect on the object of interest is immediately visible"<sup>10</sup>

Shneiderman goes on to say that if these three principles are applied, it is possible to design systems that have the following benefits:

- " 1) Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.
- 2) Experts can work rapidly to carry out a wide range of tasks, even defining new functions and features.
- 3) Knowledgeable intermittent users can retain operational concepts.
- 4) Error messages are rarely needed.
- 5) Users can immediately see if their actions are furthering their goals, and if the actions are counterproductive, they can simply change the direction of their activity.
- 6) Users experience less anxiety because the system is comprehensible, and because actions can be reversed so easily.
- 7) Users gain confidence and mastery because they are the initiators of action, they feel in control, and the system

responses are predictable."<sup>10</sup>

Clearly, these are reasons to apply direct manipulation concepts to interface design, particularly when the user group is of the novice/intermediate level.



## 5.0 EXISTING SYSTEMS

Today, there are many of applications that provide an alternative to direct SQL based querying and in some cases, applying direct manipulation to the interactive interface. This section describes three such systems and gives a subjective evaluation for each.

### 5.1 Query-by-Example

Query-by-Example (QBE) was proposed by Zloof (see Zloof, 1975) as a means of allowing the user to directly manipulate relations on the screen (Figure 5.1.1). The user is presented with table skeletons and is able to make

The screenshot shows a terminal window with two tables. The first table is a query skeleton for 'SKI-RESORTS' with columns NAME, CITY, STATE, LIFTS, and VERTICAL. The second table is the response, showing five rows of data for ski resorts.

```
Query:
SKI-RESORTS | NAME | CITY | STATE | LIFTS | VERTICAL
-----
| P. | P. | NY | | | P. >1200

Response:
SKI-RESORTS | NAME | CITY | VERTICAL
-----
| BELLEAYRE | HIGHMOUNT | 1340
| GORE | NORTH CREEK | 2100
| HUNTER | HUNTER | 1600
| SKI WINDHAM | WINDHAM | 1550
| WHITEFACE | WIMLINGTON | 3216
```

Figure 5.1.1. Zloof's Query-by-Example

literal entries in the tables as opposed to writing linear statements. The user of a QBE system can also specify fields to be printed, as well as specify variables to link between relations.

### 5.1.1. Commentary

Query-by-Example is a good first step towards the application of direct manipulation. However, it overlooks many of the human factors that should be considered when designing the interface.

First, the user's ability to memorize the keywords used to supplement the direct manipulation style is a major factor (i.e. remembering that the code 'P.' is required to print the attribute or field). This should not be a factor in the user's ability to efficiently use an application; however with QBE, it is necessary.

Second, because the display is text-based as opposed to graphical, the goal of being immediately intuitive is never reached. When a user sits down in front of a relational table skeleton, it is not intuitive that entering information into that table is equivalent to defining retrieval commands.

Finally, as the user's need to perform more complex functions increases, so do the associated operations. The goal of hiding complex functionality is not achieved because the user is required to have knowledge of the associated syntactic structures.

## 5.2 Visual Query Language

Most commercial RDBMS's provide a 4GL database access application bundled with their toolsets. Such RDBMS's include, but are not limited to, Oracle, Ingres, DB2, Rdb, and Sybase.

Visual Query Language (VQL) is a database front-end bundled with Sybase's SQL Toolset. It provides a menu-based method for generating and executing SQL queries. The VQL window is shown in Figure 5.2.1.

Just about every operation in VQL requires a menu. The VQL menu bar contains the following actions:

- Use lets the user choose tables and construct expressions and functions.

- Add lets the user add columns and expressions to queries.

- Open lets the user display the information in a window so that

a query can be reviewed and/or edited.

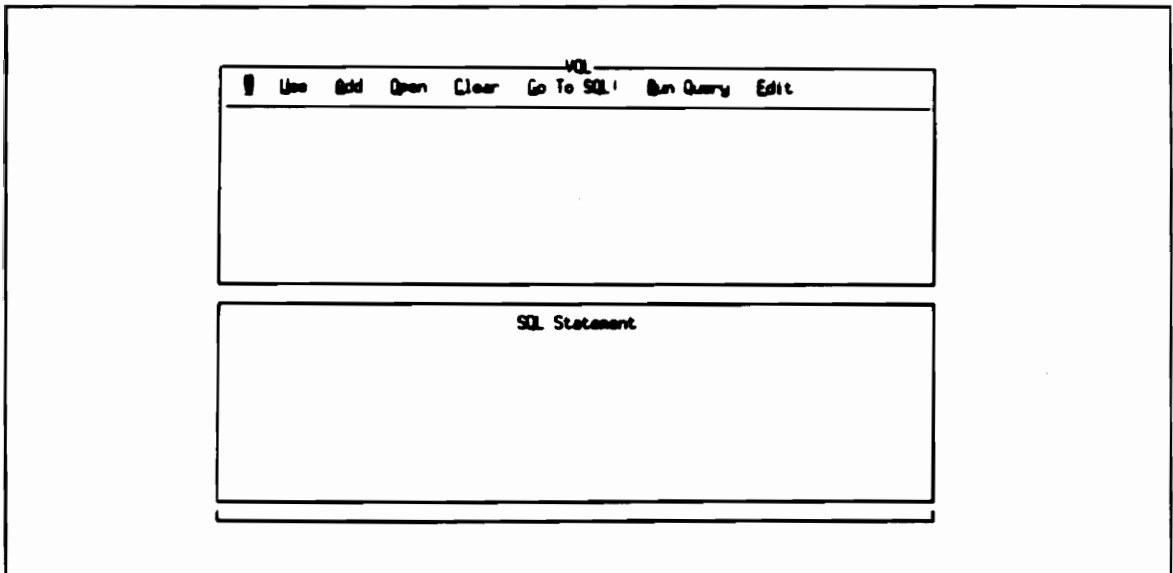
**Clear** lets the user erase one or more query parts, the query, or the query and all of the tables displayed in the VQL window.

**Go To SQL** lets the user add to or complete a query in SQL.

**Run Query** lets the user run the query and display, print, or save the results.

**Edit** lets the user modify the tables and expression lists in the VQL window.

As apparent from the menu bar description, there are several options



**Figure 5.2.1. Visual Query Language Window**

available under each menu selection, some with several levels of sub-menus.

As queries are generated, they are displayed in the bottom area of the screen labelled SQL statement.

### 5.2.1 Commentary

VQL is an improvement over QBE in that VQL saves the user the effort of having to memorize any syntax or keywords, and it eliminates the need to type the

same words over and over again. However, what VQL makes up for in terms of user memory requirements, it gives up in deep levels of nesting of menu options. Such levels of nesting can cause the user to get lost in menus. Further, it challenges the user to remember where a particular menu option lies if the user doesn't want to have to traverse the menu hierarchy to find it.

Another problem with VQL is that it prevents the user from utilizing the mouse to the fullest. Every single operation requires a mouse click. It does not suffice to slide the mouse over the option the user wants and release the mouse button, the user must click the mouse. This is extremely cumbersome since there are so many levels in the menu hierarchy, and the user must click through every step. Choosing not to use the mouse at all may be faster. There are also times when the user must use the keyboard because the mouse does not provide an acceptable response. This violates the rule of consistency within the interface. It should be the user's choice to switch between the keyboard and the mouse when entering information, not something the user is forced to do by the system.

Another limitation of VQL is that the user can't create sub-queries (nested queries) with it; the user is forced to use joins.

And finally, as new windows are opened in VQL, the underlying windows are visually lost. This, in conjunction with the deep menu hierarchy, can get confusing, causing anxiety in the user. This is something direct manipulation can reduce, if not alleviate.

### 5.3 Graphical Query Language

Graphical Query Language (GQL) is a custom application and uses the concept of a data model to represent the target database. The querying environment is actually presented to the user when the desired data model has been indicated. Figure 5.3.1 shows GQL's data model window. The icons represent database tables, and the diamonds represent the relationships between those tables. The "Executive Buttons" operate on pre-defined queries. By selecting the report icon, the user can indicate to GQL that they want to generate a report in a pre-

defined format. The executive buttons can be defined by the user so that common operations can be performed by simply pressing the appropriate button.

When generating Ad Hoc queries, table objects can be selected by double-clicking on the table icons. GQL then displays another window to the user which shows all of the attributes in the active database table. The user now has the ability to specify the attributes he/she wants to appear in the Ad Hoc query. When an attribute is selected, it changes to boldface and a bullet appears in front of it. Once the attribute is selected, the user can specify the aggregate function to be applied to it, the conditions to be applied to it, whether or not the attribute will be used in grouping the output, and what role the attribute will play, if any, in sorting the output. Figure 5.3.2 shows the GQL attribute window.

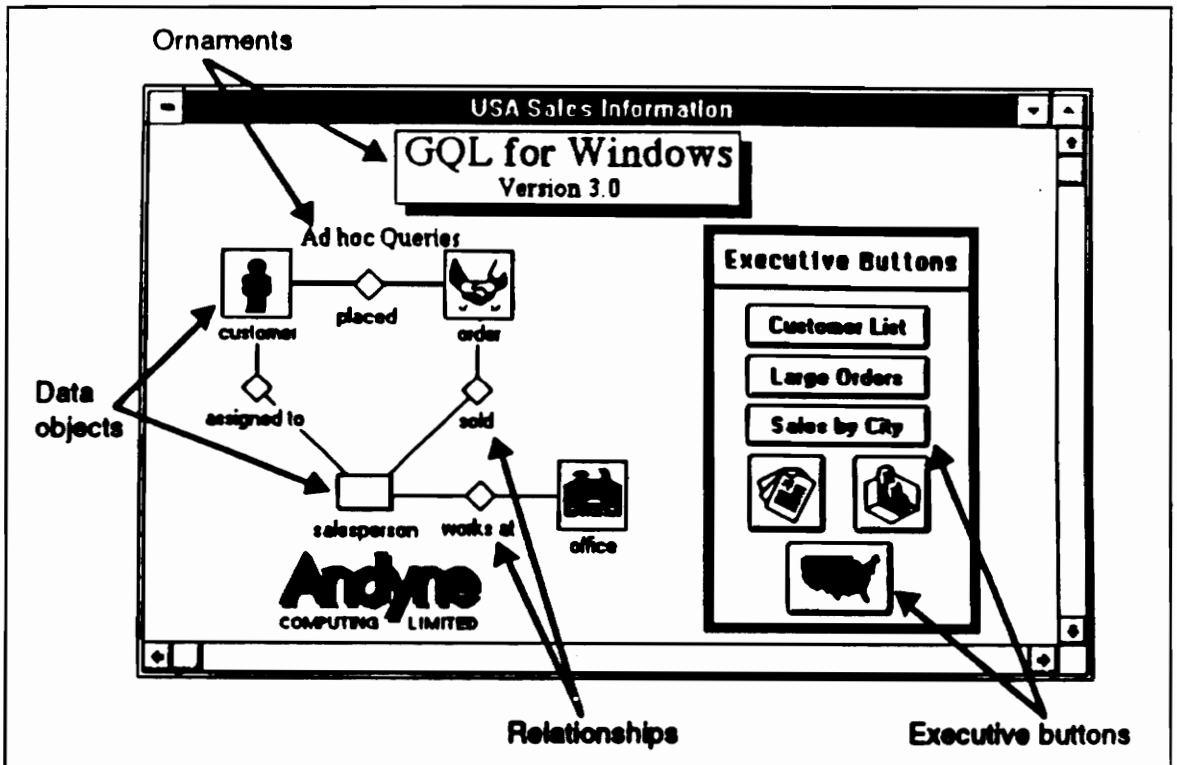


Figure 5.3.1. GQL Data Model Window.

Attribute	Function	Quality	Group	Sort
(?) Ord #				
(?) Name				
(?) Address 1				
(?) Address 2				
(?) City				
(?) State				
(?) Zipcode				

Figure 5.3.2. GQL Attribute Window.

The parent window of GQL is a window with several menu options across the menu bar at the top of the window. The menu options provide the user with the ability to manage such things as the system windows, generated queries, and report specifications.

### 5.3.1 Commentary

Graphical Query Language (GQL) is an excellent example of applying direct manipulation to relational database access operations. It far exceeds the efforts of QBE and VQL; however, windows are used to allow the user to specify and/or view every operation. This results in numerous windows all over the screen. This can get confusing when the user needs to see several things simultaneously.

Another area of concern with GQL is the amount of effort required to customize the application interface to several different database systems. The data model representing the target database is not only a graphical view of the database structure, but it is the means by which the user indicates what tables he/she wants to use when generating Ad Hoc queries. It follows that the underlying association between query generation and the visual data model is very tightly coupled. At first glance this seems to be appropriate; however, think about the possible need of a DBA to add a table to a database. If this new table is not reflected in the data model, the user doesn't have access to it. This then limits the user's ability to access database data. And, considering the effort required in designing this graphical representation, the time required to bring the application up to date is fairly significant.

The ability to perform join operations on database tables can get complex because of the number of windows involved in creating queries. If the Data Model window is visible there isn't much of a problem; however, if the Data Model window manages to get nested under many layers of windows and the user needs to specify a table join, the process can become exhaustive.

Lastly, GQL makes it very inconvenient to transfer between databases. This is going to be a major factor since many if not all of today's RDBMS's allow the user

the ability to create several different databases within the DBMS. Hence the desire to utilize different databases may arise, and the application should allow for this. With GQL, the user must exit the system and upon re-entry specify the new database he/she wants to be the target database.



## 6.0 PULLING IT ALL TOGETHER

Data access is a crucial operation. It is the means by which users are able to retrieve, view and manipulate system data. And, since software systems are data based, and many of the users interacting with these systems are not experienced programmers, the need also arises for a simplistic and intuitive means of accessing the data. More specifically, in many systems that are based on data, the data is stored in and managed by a DBMS; the most popular being the RDBMS. And the standard that is used to provide access to RDBMS data is SQL. Therefore, the question arises as to whether direct SQL based querying provides the novice/intermediate user with an easily understandable means of accessing RDBMS data. The answer is no because although SQL can be simplistic it is not intuitive and it's complexity is tightly tied to the underlying structure of the database. Since many of today's systems are large and complex, the SQL retrieval statements are also complex. It therefore follows that there needs to be an alternative to direct SQL based querying. But, what techniques needs to be applied to ensure that a proposed alternative addresses the targeted problem area? The answer is the concepts of HCI. That is, understanding that the means by which an individual communicates with a system is provided by some type of interface, and that the human factors that are tied to interactive systems are directly related to the capabilities of an individual. Everyone's abilities are different, so how does a designer determine the best way to have such diverse individuals interact with a particular system? The answer is using a user- centered design process. If the design of the system is such that the actual user or users are the center of focus, the designer doesn't get caught up in attempting to please everyone, but he/she does focus on pleasing the user who is the most important aspect of the system and the system design.

The next question that arises is what techniques should be applied in the formulation of the interface? Direct Manipulation has become the overwhelming choice of designers because of it's ability to make the user feel as though they are in control of the system, as opposed to the system being in control of them.

That said, are there any systems in existence today that attempt to address the targeted problem area? The answer is yes, and the goal in designing an alternative to direct SQL based querying, as it relates to these systems, is to improve on the inadequacies of them.

## **7.0 A PROPOSED SYSTEM DESIGN**

### **7.1 Introduction**

This section will walk through the design of a system which is a proposed alternative to direct SQL based querying. The system is called QUERI-EZ. The interface is presented, as well as the underlying software organization. In order to logically represent the steps taken in the software design, structured design techniques have been applied. These techniques are by no means meant to be exhaustive; however, they do serve as a clear means of describing the evolution of QUERI-EZ.

Short format stubs have been taken from suggested document formats (General Electric Company, 1986) to produce a System Specification and a Software Requirements Specification, each of which has been modified to highlight the features of QUERI-EZ, in terms of system operation.

### **7.2 System Specification**

#### **7.2.1 Scope**

This System Specification (SS) indicates the overall functional and operational characteristics of the QUERI-EZ system.

#### **7.2.2 Requirements**

This section contains a descriptive definition of QUERI-EZ's functional and interface requirements, as well as the characteristics of the system.

##### **7.2.2.1 Functional Description**

QUERI-EZ is a Graphical User Interface (GUI) based application which shall be used to provide an intuitive means of querying a database. It is not intended to be a data manipulation application and shall, therefore, be designed to act as a supplement to a larger database system.

QUERI-EZ shall allow the user to input data by means of a mouse and a keyboard. Output shall be available to the user on the display screen or an on-line

printer. Figure 7.2.2.1.1 provides a conceptual view of QUERI-EZ.

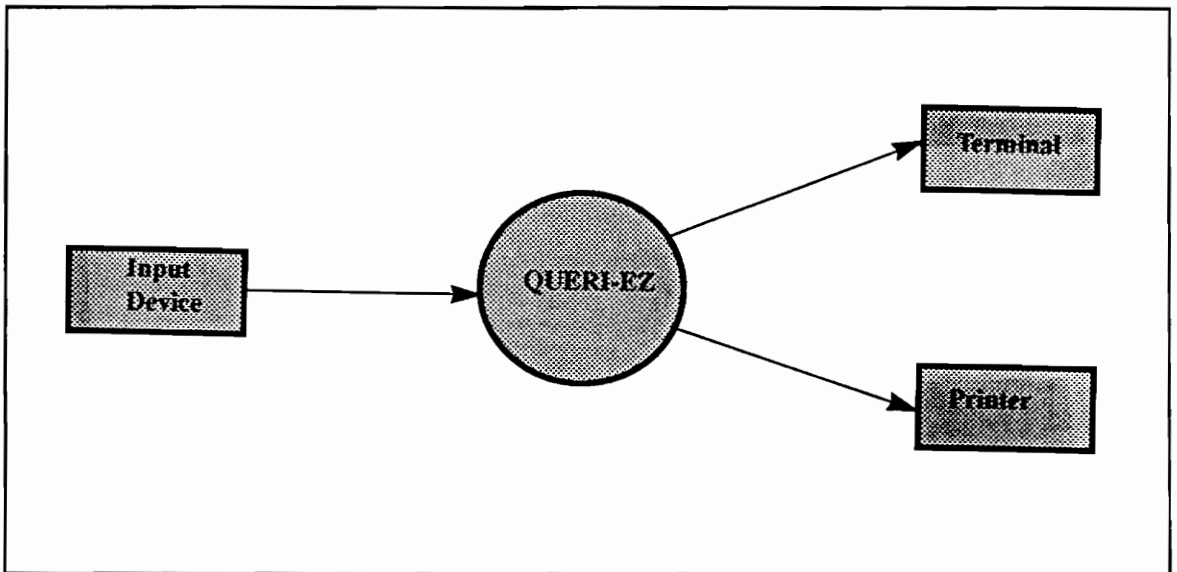


Figure 7.2.2.1.1. Conceptual View of QUERI-EZ

### 7.2.2.2 Interface Definition

QUERI-EZ is not intended to be a replacement for SQL. However, it shall provide an alternative for interfacing with SQL directly when querying a database. It shall run on top of an Interactive SQL (ISQL) application. Figure 7.2.2.2.1 depicts the way QUERI-EZ shall interface with other system elements.

### 7.2.3 Characteristics

QUERI-EZ shall be run in a windowing environment and must interface with a SQL based relational database. The user group to which QUERI-EZ will be focused is the novice/intermediate user group.

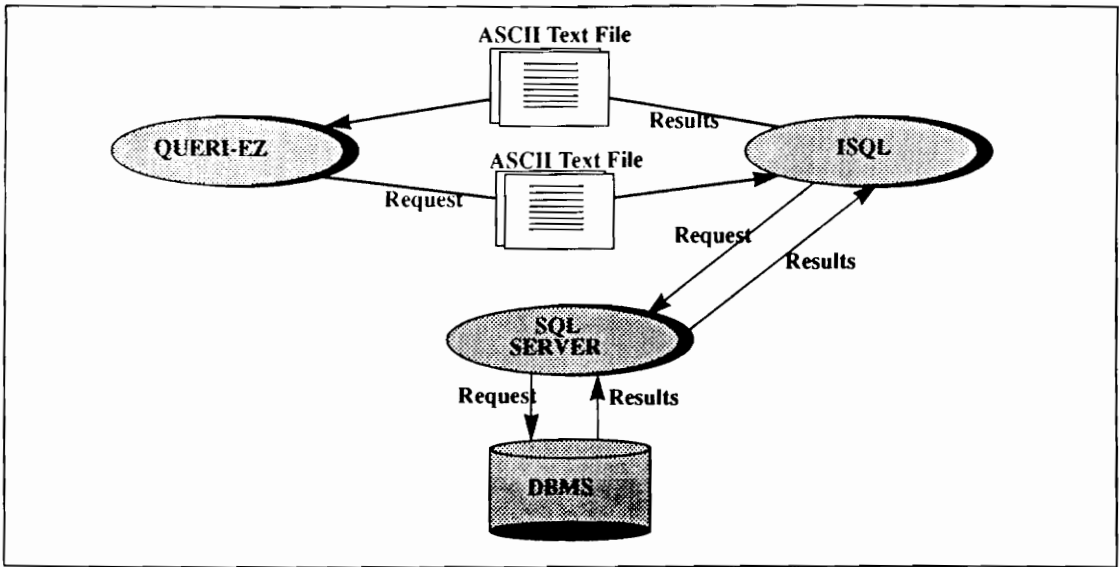


Figure 7.2.2.1. QUERI-EZ interface/operation.

## **7.3 Software Requirements Specification**

### **7.3.1 Scope**

This Software Requirements Specification (SRS) establishes detailed requirements for the design of QUERI-EZ.

### **7.3.2 Functional Requirements**

This section describes the functional requirements of QUERI-EZ.

#### **7.3.2.1 Description of Function**

The following list specifies the functionality QUERI-EZ shall provide. The system shall :

- 1) Provide the user with the ability to generate simple queries.
- 2) Provide the user with the ability to apply mathematical expressions to output columns to retrieve computed values.
- 3) Provide the user with the ability to place conditions on queries.
- 4) Provide the user with the ability to order rows of output by column(s).
- 5) Provide the user with the ability to group rows of output by column(s).
- 6) Provide the user with the ability to join two or more tables together.
- 7) Provide the user with the ability to specify the following aggregate functions: COUNT, MAX, MIN, SUM, and AVG.
- 8) Provide the user with the ability to construct conditions using wild card characters for string matching.
- 9) Provide the user with the ability to generate subqueries with multiple levels of nesting (complex queries).
- 10) Provide the user with a two mode results window which shall allow the user to tell the system whether the output fields will be formatted or tabular.
- 11) Provide the user with the ability to rename output fields.
- 12) Provide the user with the ability to retrieve existing queries.

- 13) Provide the user with the ability to save queries.
- 14) Provide the user with the ability to view generated SQL.
- 15) Provide the user with the ability to clear results from the results screen.
- 16) Provide status messages to the user.
- 17) Provide the user with the ability to quit the creation of a query and start over without having to restart the system.
- 18) Provide the use with the ability to switch between databases without having to exit the system.

### **7.3.3 User Interface Requirements**

The following section describes the user interface.

#### **7.3.3.1 Description of Interface**

The QUERI-EZ user interface shall be a graphical user interface. It shall be prototyped using a User Interface Management System. The interface shall be simplistic and intuitive, and it shall allow the user to perform all of the functions listed in section 7.3.2.1.

#### **7.3.4 User Requirements**

The user shall be able to provide input to QUERI-EZ using the keyboard and/or the mouse. Further, the user will not be required to memorize syntax or command names.

## 7.4 System Design

The design of QUERI-EZ is presented in two phases: (1) the User Interface Design, and (2) the Software Design. As required, the proposed design adds two additional layers on top of ISQL. These layers provide the user interface and the processing of the data received from that interface to be sent to the ISQL application. Figure 7.4.1 shows the current concept versus the proposed concept.

### 7.4.1 User Interface Design

The user interface design includes a presentation of the interface based on the system and software requirements specifications, as well as scenarios, state transition diagrams, and the corresponding user action notations.

#### 7.4.1.1 Interface Presentation

Figure 7.4.1.1.1 shows the interactive interface of QUERI-EZ. There are four major areas to the interface: (1) the Main Menu, (2) the Query Generator Window, (3) the Results Window, and (4) the Data Structure Window.

Upon first entering the QUERI-EZ application, the only thing available to the user is the main menu. The "HELP" button provides a scrollable document which explains how to use the system. The "EXIT" button removes the user from the QUERI-EZ application. The "QUERY" button actually starts providing the user with functionality. When pressed, it initiates the display of the "QUERY GENERATOR" and "REPORTS" windows.

The "QUERY GENERATOR" window is where the user selects the appropriate database, database tables, and table column names, as well as constructs expressions and functions. It is based on the hierarchical structure of database entities. The "QUIT" button near the bottom of the window allows the user to stop the creation of a query, and the "RETRIEVE" button is used to retrieve existing queries. The status bar at the bottom of the window is the application's way of keeping the user informed about the queries being



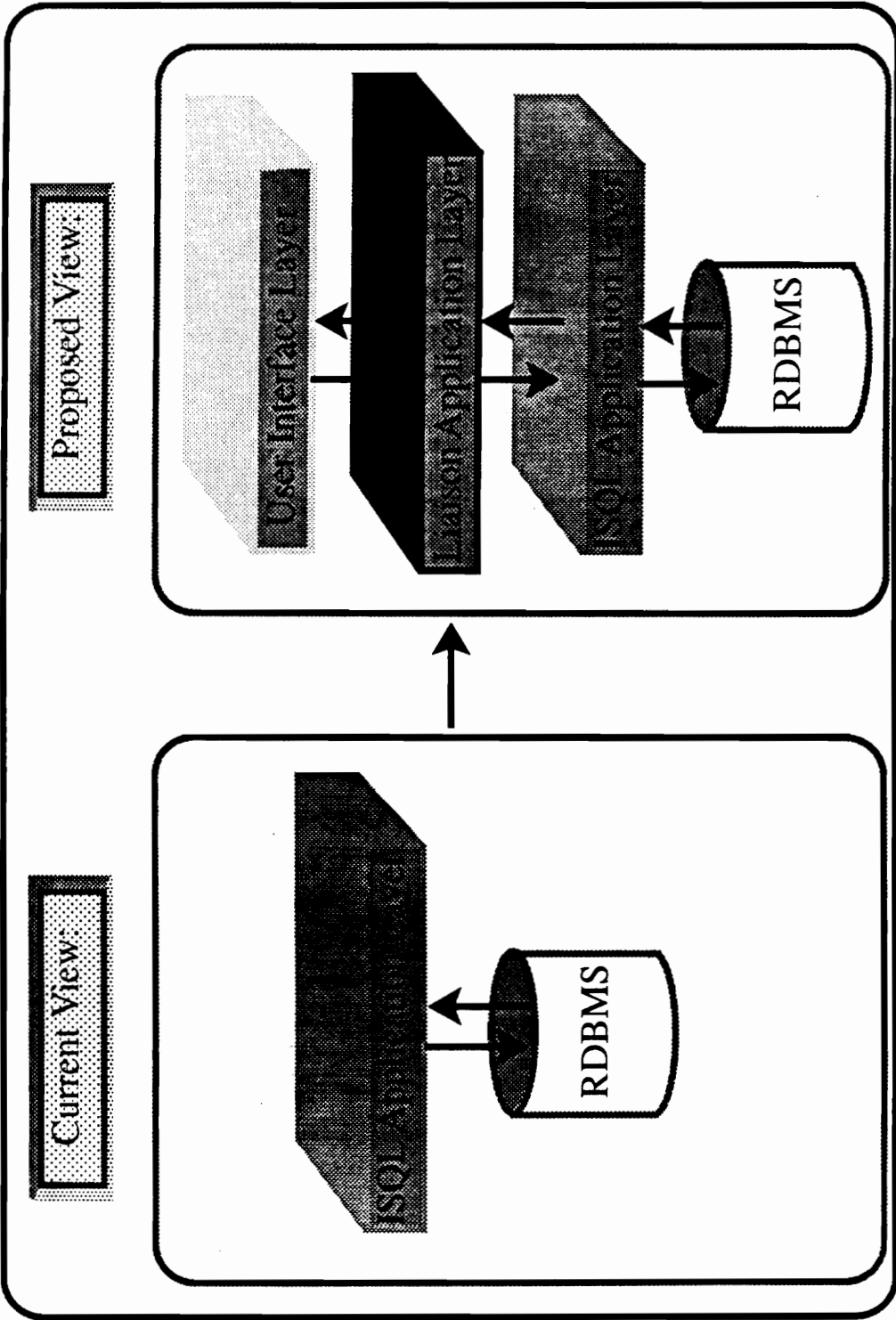


Figure 7.4.1. Conceptual View of Proposed Layer Structure

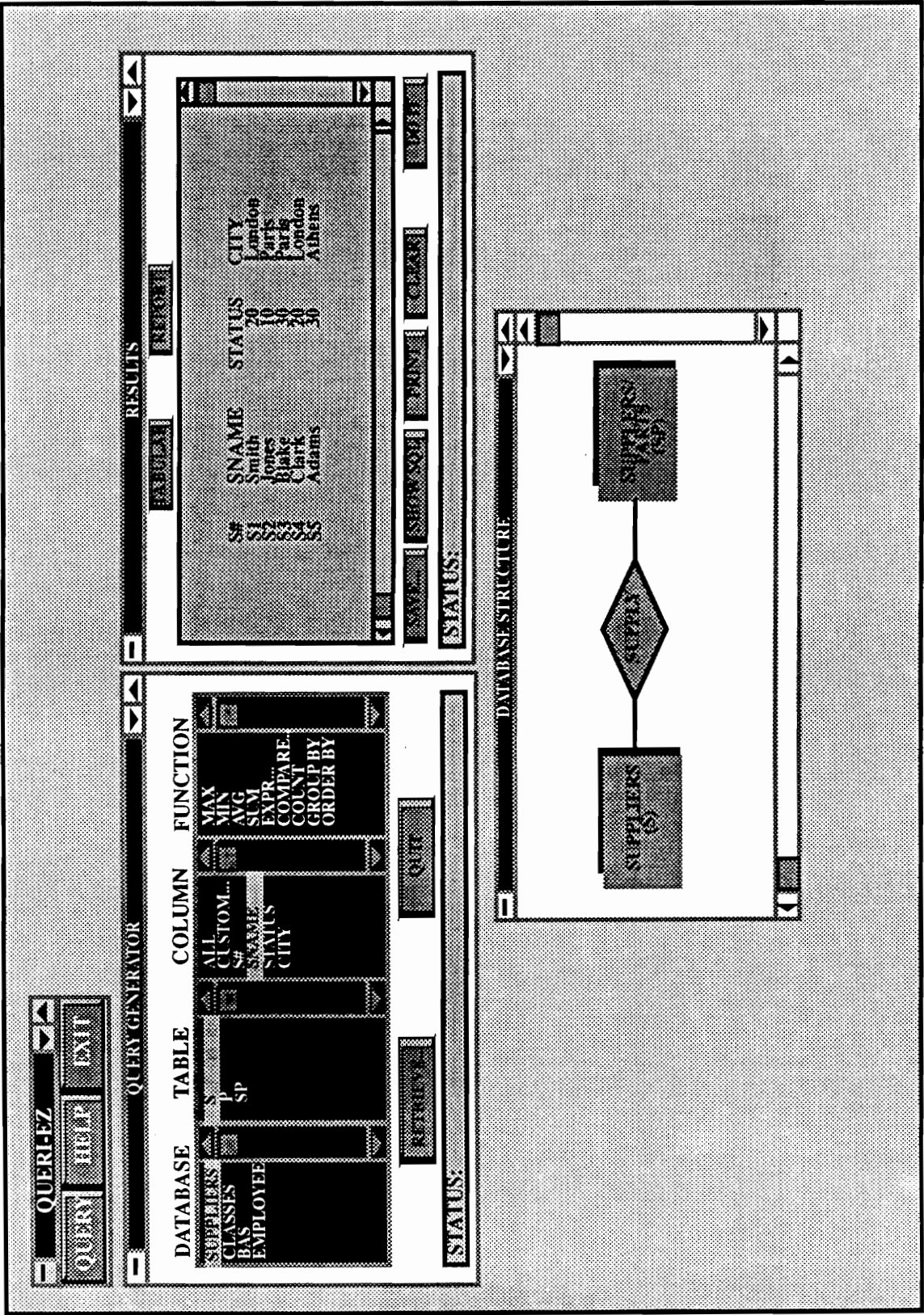


Figure 7.4.1.1. QUERI-EZ User Interface

generated.

The "RESULTS" window is where the results of the user's queries are displayed. The user can change the mode of this window using the "TABULAR" and "REPORT" buttons near the top of the window. "TABULAR" is the default mode, and in this mode the user is able to change the name of fields for output purposes. The layout of the results will be tabular and will not be modifiable. However, by placing the window in "REPORT" mode, in addition to changing the field names, the user can format the results as desired. Simply by using the mouse to select a field, and dragging it to the appropriate area on a screen, the user is able to dictate the layout of the results.

The buttons near the bottom of the results window apply to the query results and/or the SQL which created the results. The "SAVE" button allows the user to save a SQL query that produced the results displayed in the results window. The "SHOW SQL" button allows the user to view the generated SQL. The "CLEAR" button allows the user to clear the results window, and the "PRINT" button allows the user to print the results in the "RESULTS" window to a printer. The "DO IT" button initiates execution of a SQL statement. Finally, the status bar at the bottom of the window gives the user status on the results of the current query.

The "DATABASE STRUCTURE" window is a scrollable window that displays the current structure of the active database. When the user enters QUERI-EZ and selects the "QUERY" button, this window is not yet displayed. It is first displayed when a database is selected in the "QUERY GENERATOR" window. Its contents change as table names are selected to show the relationships a particular table has to other tables in the database.

Any three of the windows can be closed at any time. However, closing the "QUERY GENERATOR" window is equivalent to exiting out of the system because nothing can be done without it. In addition, each window has maximize and minimize features so that they may be closed to icons or enlarged to engulf the entire work space.

### 7.4.1.2 Scenarios

A scenario is a series of interactive screen states that show the way the display looks as the user performs various functions. In this section, three scenarios will be shown, and a sample database is defined to make the scenarios more realistic.

#### 7.4.1.2.1 The "Suppliers-Parts" Database

Figure 7.4.1.2.1.1 shows the E-R diagram for the "Suppliers-Parts" database (Date, 1991), and Figure 7.4.1.2.1.2 shows the structure of the tables of the database.

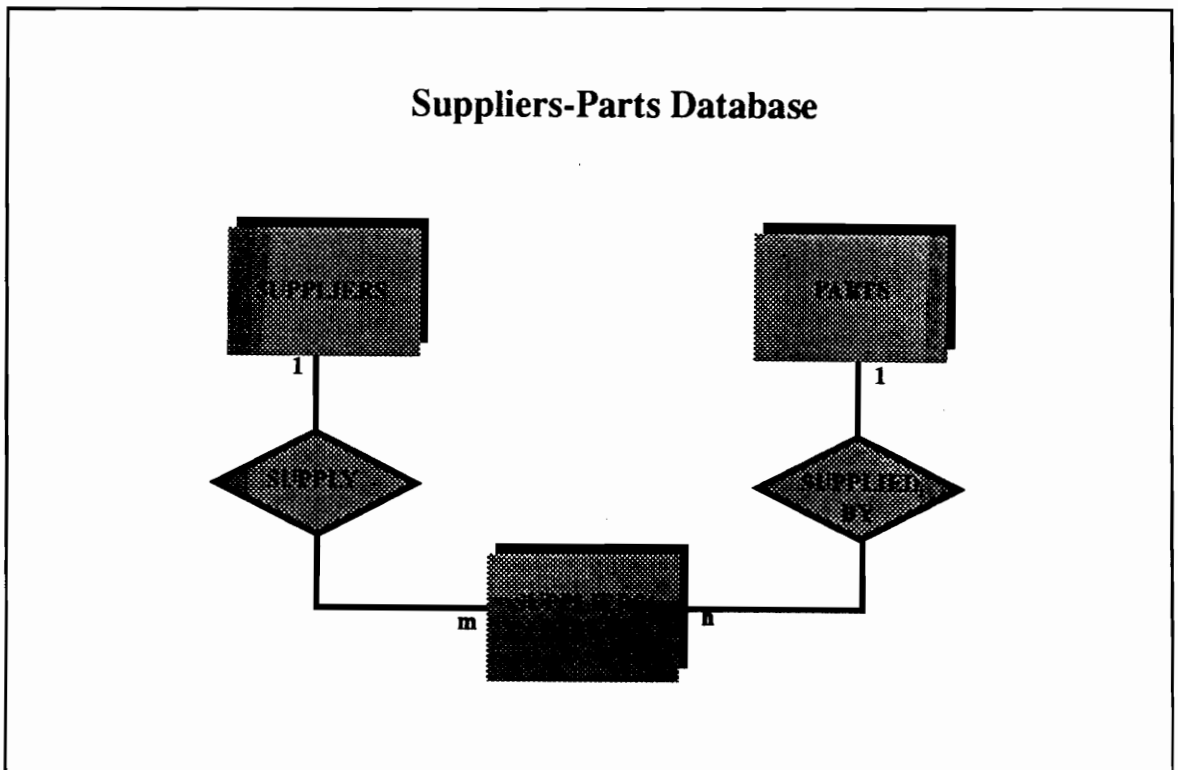


Figure 7.4.1.2.1.1. E-R Diagram for Suppliers-Parts Database

S	S#	SNAME	STATUS	CITY	SP	S#	P#	QTY
	S1	Smith	20	London		S1	P1	300
	S2	Jones	10	Paris		S1	P2	200
	S3	Blake	30	Paris		S1	P3	400
	S4	Clark	20	London		S1	P4	200
	S5	Adams	30	Athens		S1	P5	100
						S1	P6	100
						S2	P1	300
						S2	P2	400
						S3	P2	200
						S4	P2	200
						S4	P4	300
						S4	P5	400

Figure 7.4.1.2.1.2. Suppliers-Parts Database Structure.

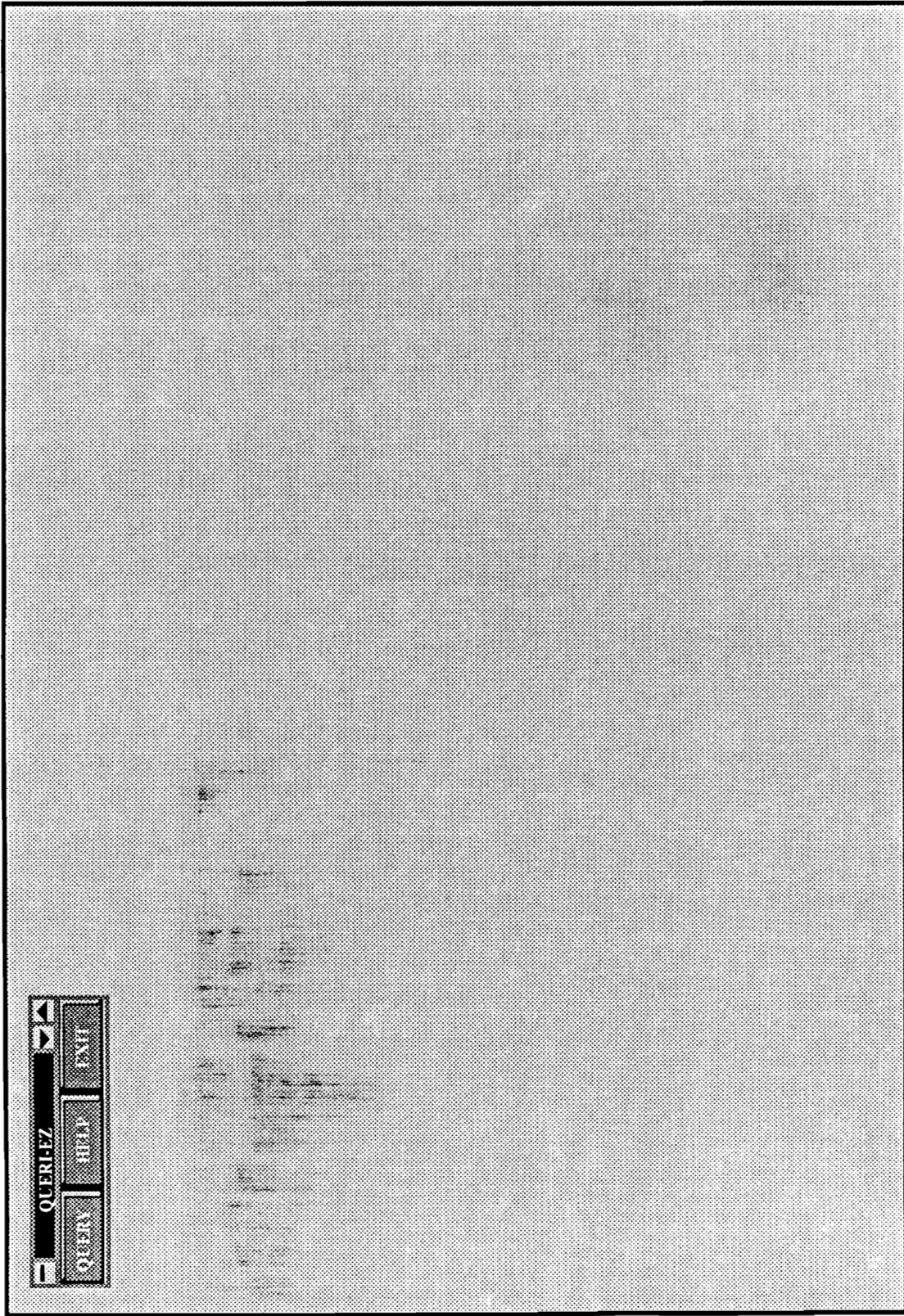
### 7.4.1.2.2 Scenario 1: Creating a Simple Query

This query involves creating a single level query and formatting the results.

Setup: Database data needs to be accessed, and the user knows that the query to get the results has already been generated.

Figures 7.4.1.2.2.1a through 7.4.1.2.2.1g show the stages the interface goes through as the user implements this task.





**Figure 7.4.1.2.2.1a. Scenario 1: QUERI-EZ main menu.** When the user enters QUERI-EZ, the main menu is the only object displayed.

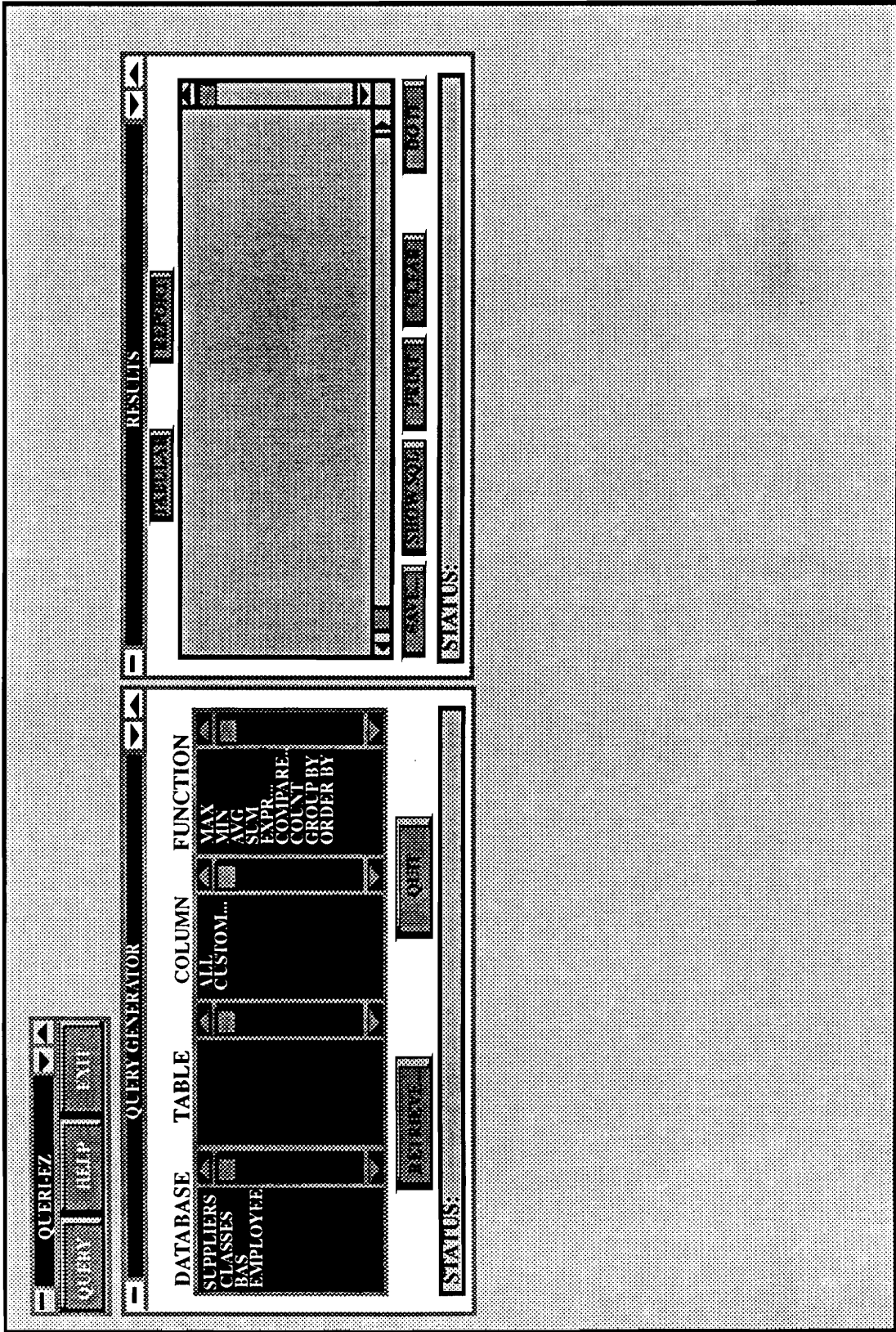
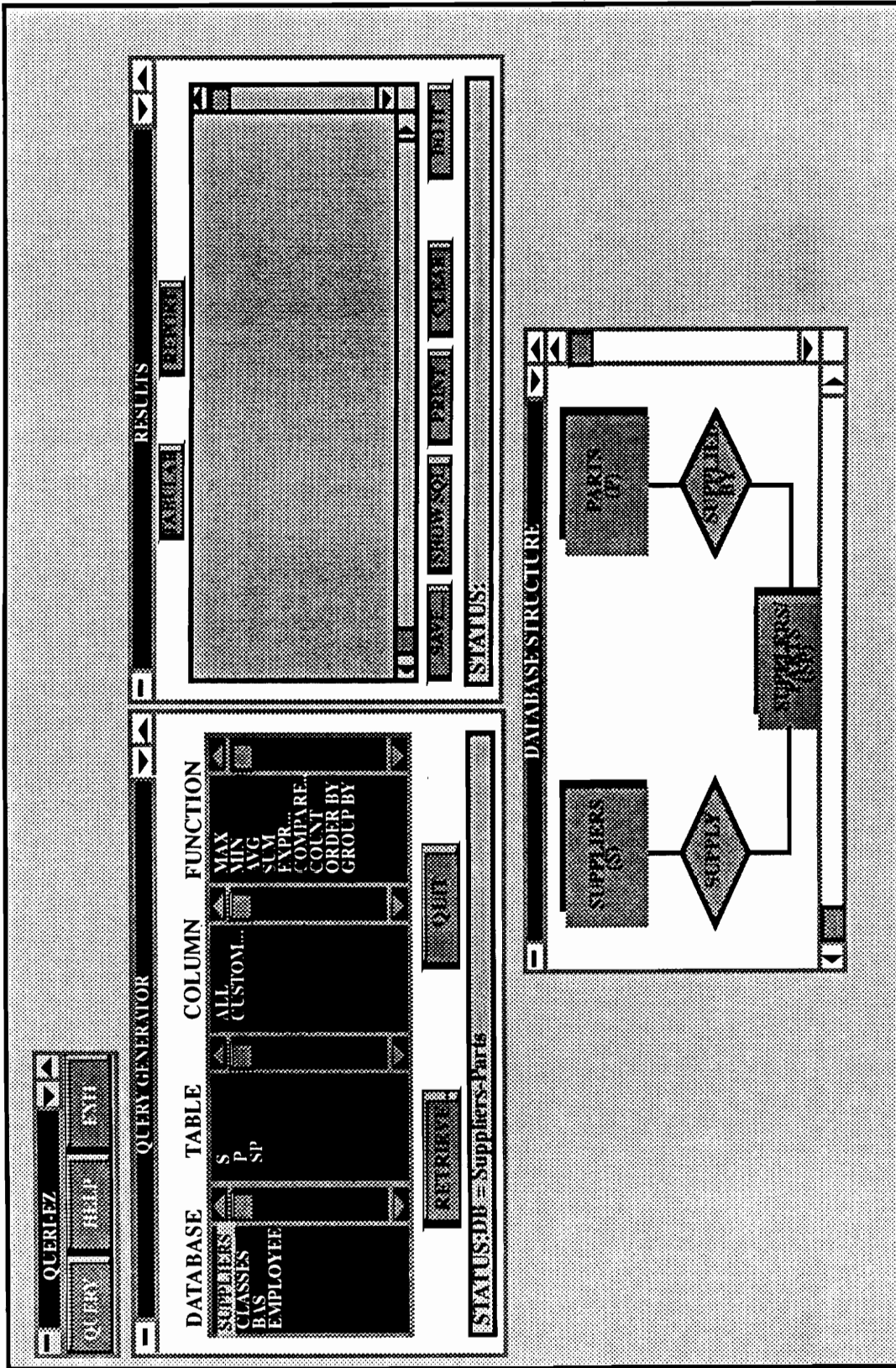
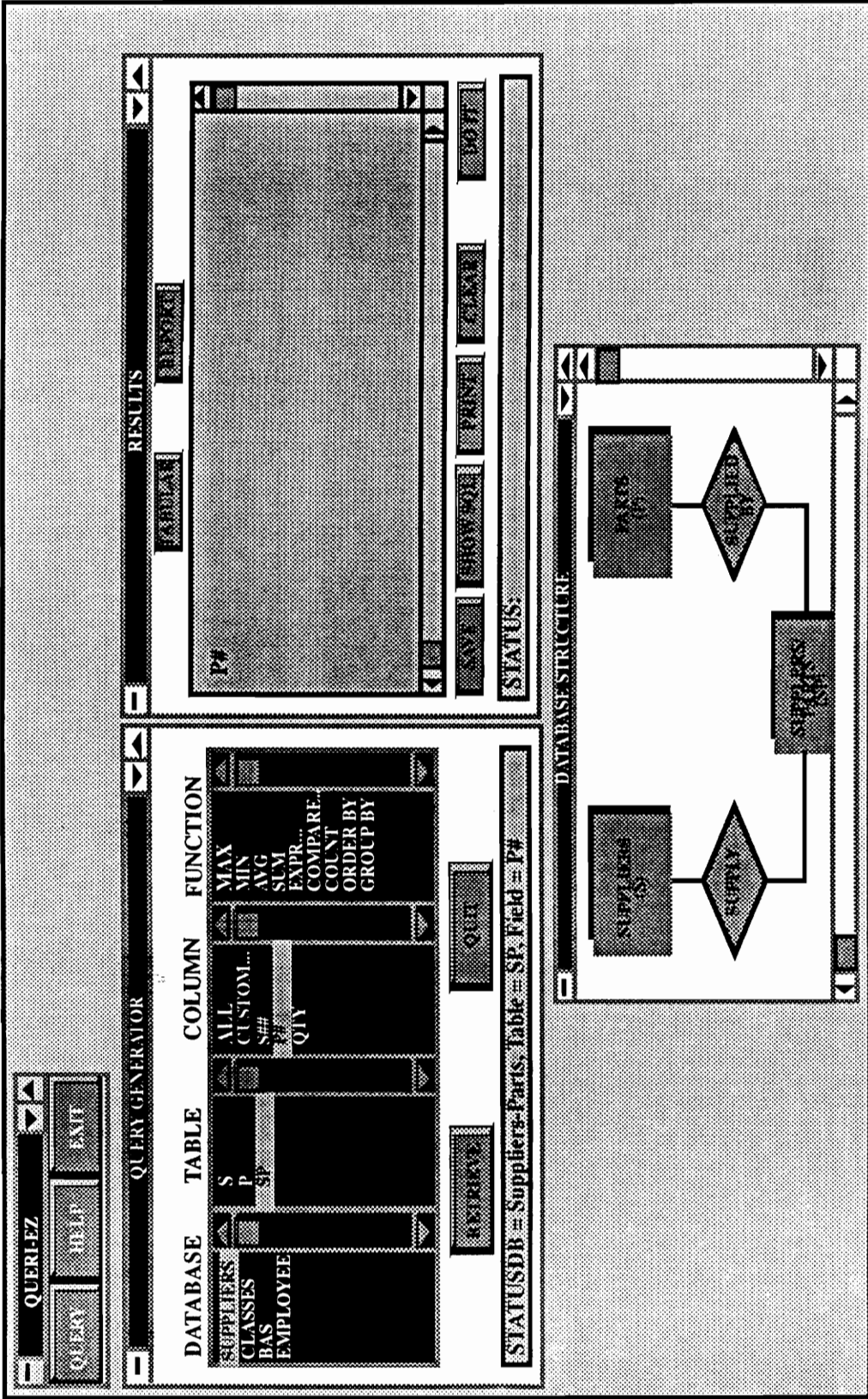


Figure 7.4.1.2.2.1b. **QUERY GENERATOR and REPORT windows.** When the user selects the “QUERY” option from the main menu, these windows are displayed, and the user can begin to generate queries.



**Figure 7.4.1.2.1c. Database selection.** The first step in generating a query is to select the database holding the target data. Notice that when a database name is selected, the data structure window is displayed with an E-R diagram showing the relationships among all of the tables in the database.





**Figure 7.4.1.2.2.1d. Table and column name selection.** When the user selects the desired table, the data structure window is updated to show the relationships to only the selected table. When the column name is selected, it is displayed in the results window for output purposes. The user now has generated the SELECT and FROM clauses. The same step is taken to select the QTY column from the SP table.

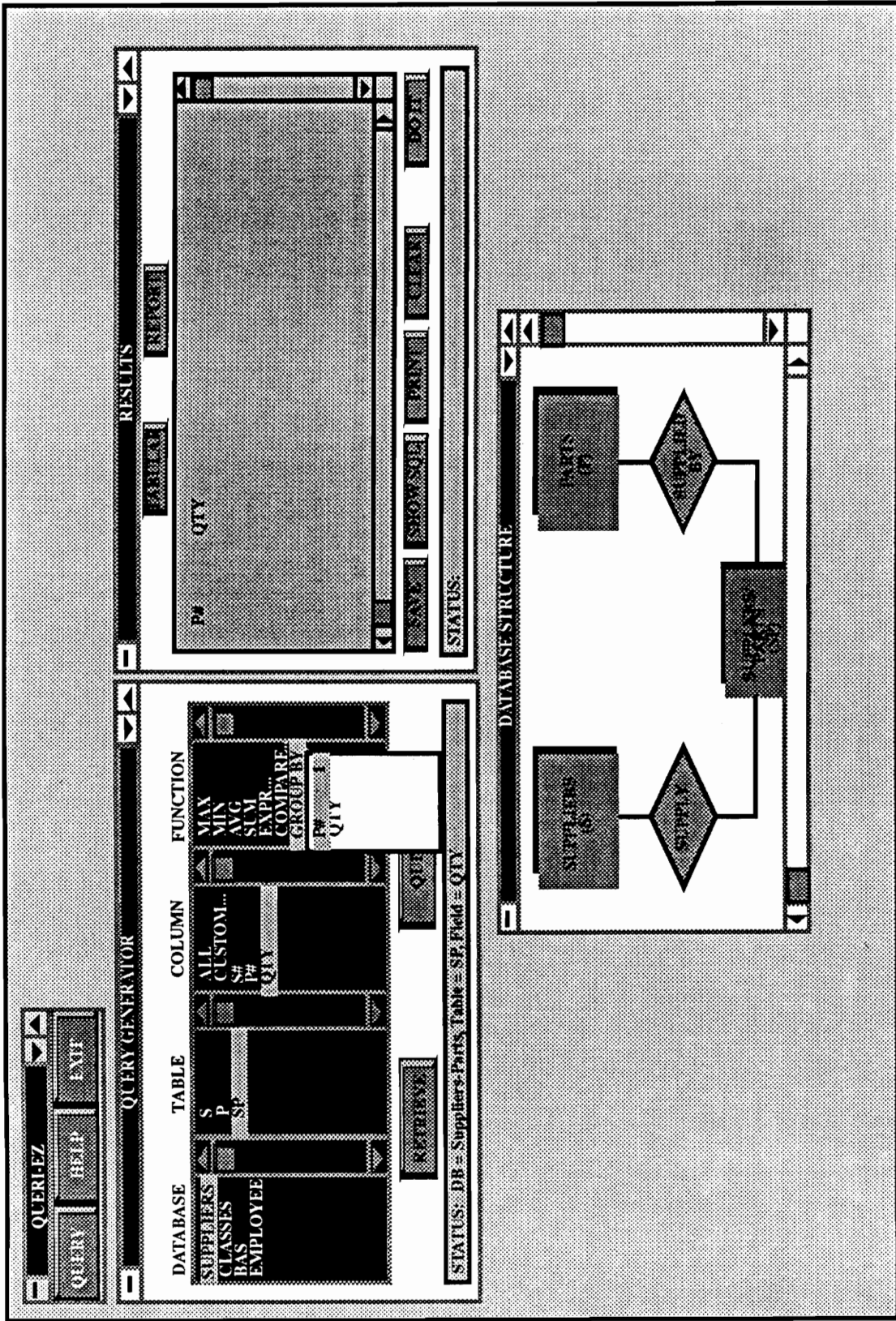
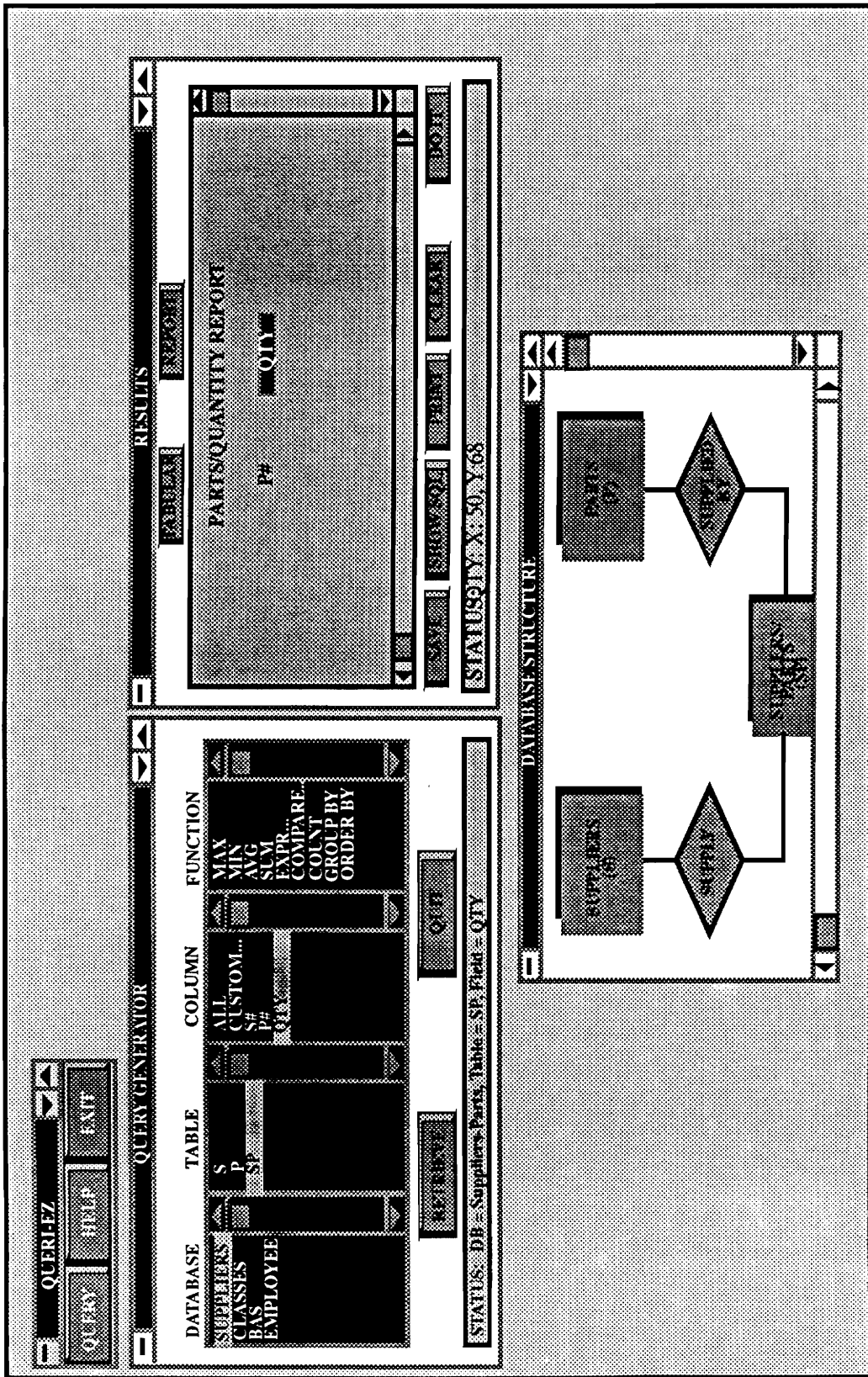


Figure 7.4.1.2.2.3e. Utilizing the "GROUP BY" clause. When selecting the columns to group by, the user has the ability to select from columns that are currently in the SQL SELECT clause.



**Figure 7.4.1.2.2.3f. Formatting output in the results window.** The user has the ability to move the output fields in order to format the results. QUERY-EZ allows the user to set the mode of the window before they initiate the generation of a query. In this query the window was put in REPORT mode.



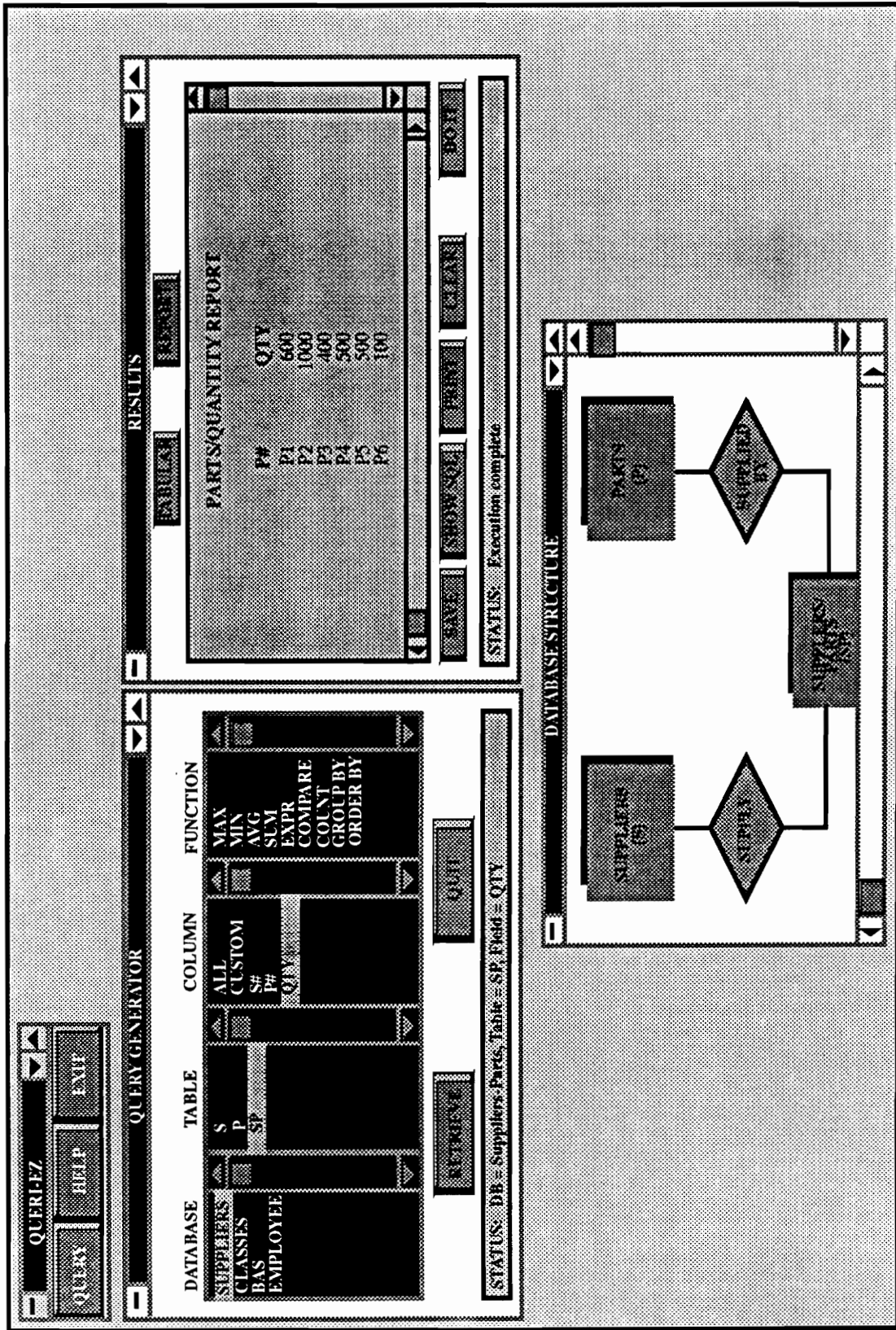


Figure 7.4.1.2.2.3g. Formatted window with results. The user selects "DO IT" when they are ready to execute the generated SQL statement. The results of the simple query are shown.

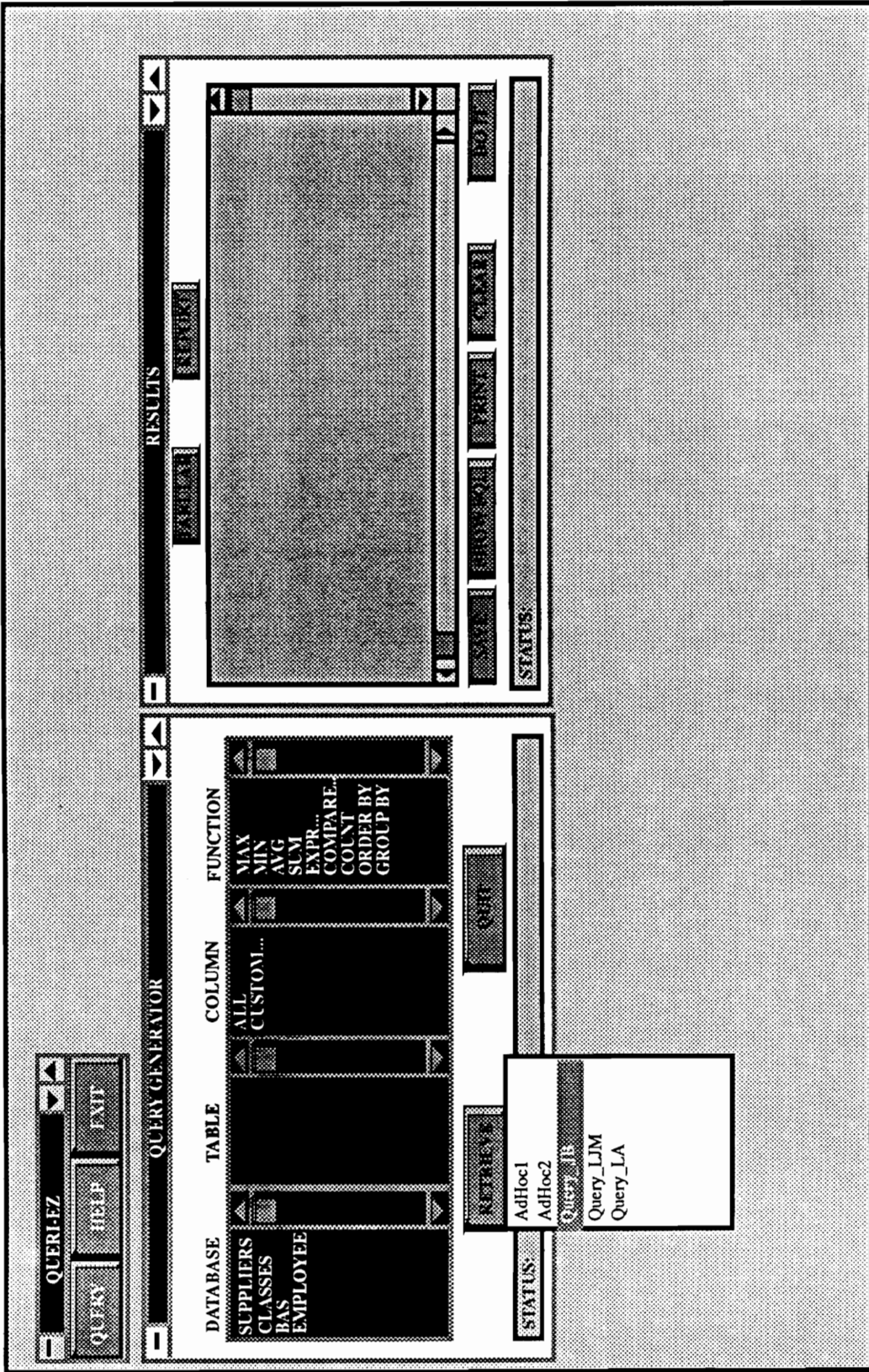
### 7.4.1.2.3 Scenario 2: Retrieving an Existing Query

This scenario will involve retrieving and executing the query generated in the scenario of section 7.4.1.2.2.

Setup: The user needs to determine, for each part supplied, the part number and total shipment quantity. The SQL that needs to be generated is shown below:

```
SELECT P#, SUM (QTY)
FROM SP
GROUP BY P#;
```

Figures 7.4.1.2.3.1a through 7.4.1.2.3.1b show the stages the interface goes through as the user implements this task.



**Figure 7.4.1.2.2.a. Retrieving an existing query.** The user is able to retrieve existing queries by pressing the retrieving button, and highlighting the query they want to load. This view of the interface assumes that the user just entered the system.



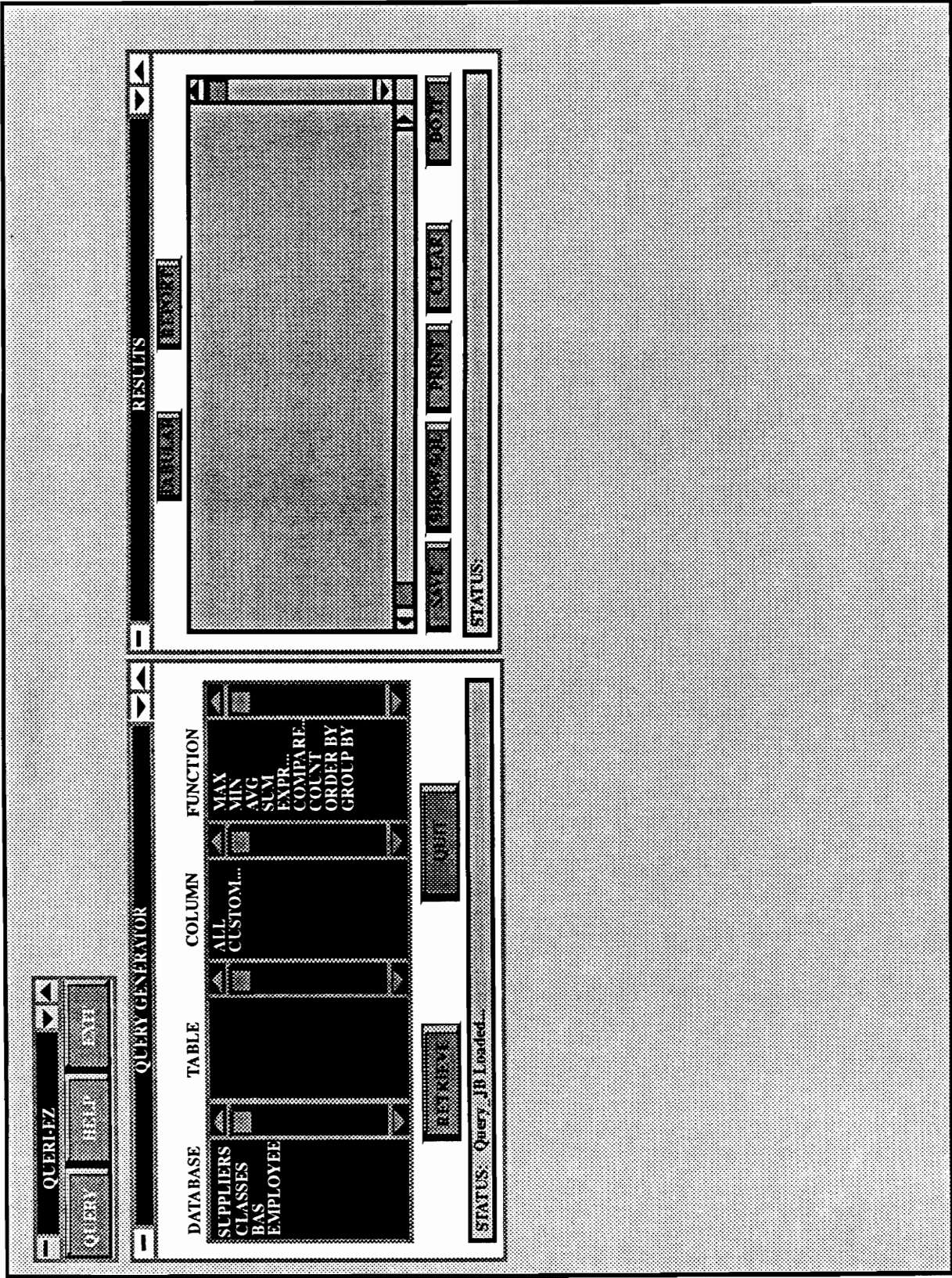


Figure 7.4.1.2.2.b. QUERY-EZ response to query retrieval. The user can now select the "DOIT" button in the Results window to execute the loaded query.

#### 7.4.1.2.4 Scenario 3: Creating a Complex Query

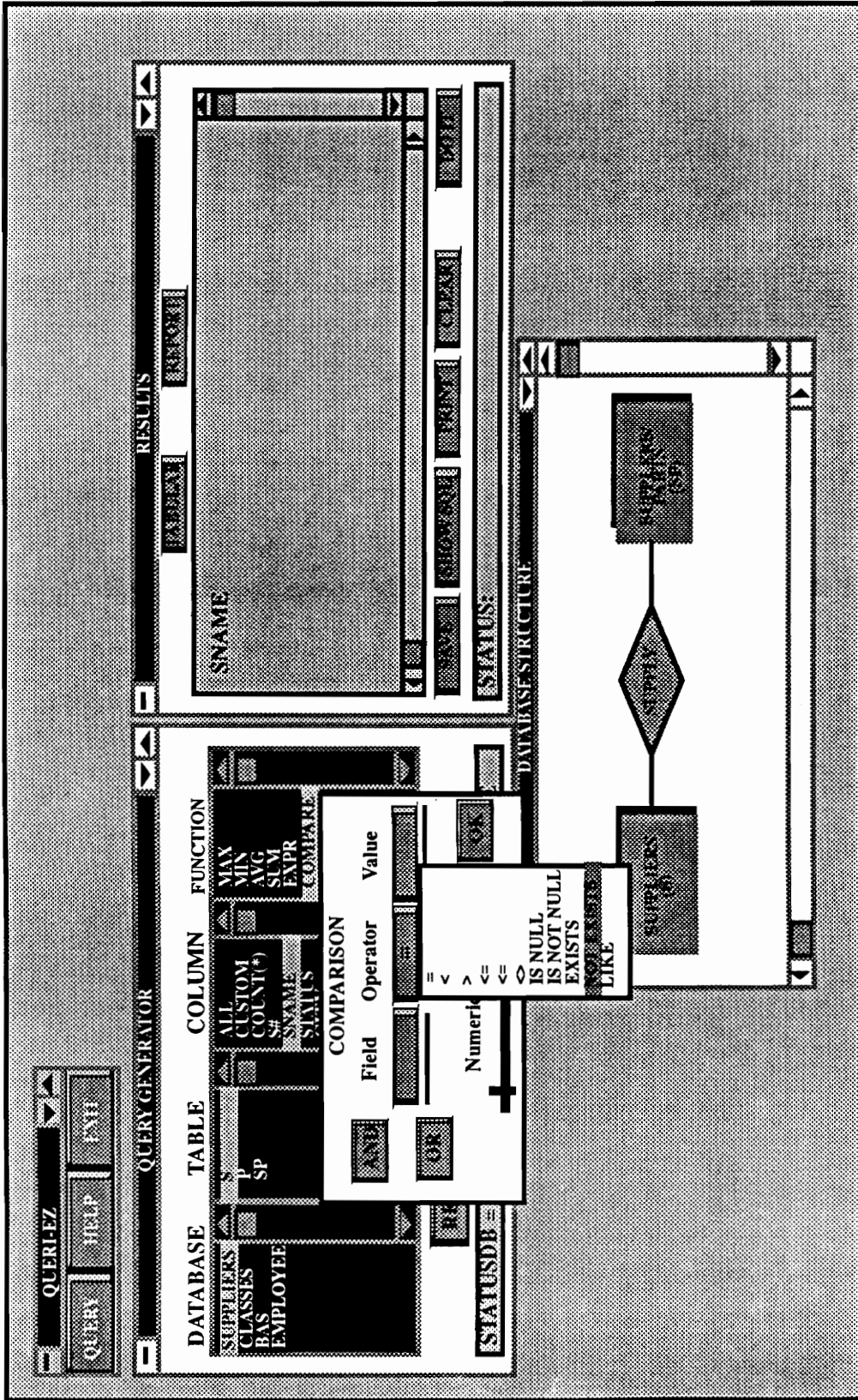
This scenario will involve creating a query using multiple levels of nesting, viewing the generated SQL, and saving the SQL statement for future use.

Setup: The user needs to determine the names of all suppliers who supply all parts. The SQL statement that needs to be generated is shown below:

```
SELECT SNAME
FROM S
WHERE NOT EXISTS
  (SELECT *
   FROM P
   WHERE NOT EXISTS
     (SELECT *
      FROM SP
      WHERE S# = S.S#
            AND P# = P.P# ) ) ;
```

Figures 7.4.1.2.4.1a through 7.4.1.2.3.1d show the steps the user would take in generating this query.





**Figure 7.4.1.2.2.3a. Specifying “WHERE” clause.** In creating this complex query, the specification of the database, table, and column names are the same as in the previous scenarios. Since this is a nested query, the operands for the “NOT EXISTS” operator are the SELECT statements on either side of it. Once the “NOT EXISTS” operator is selected, the comparison window can be closed by selecting the “OK” button. To generate the remainder of the SQL statement, the previously described steps are repeated.

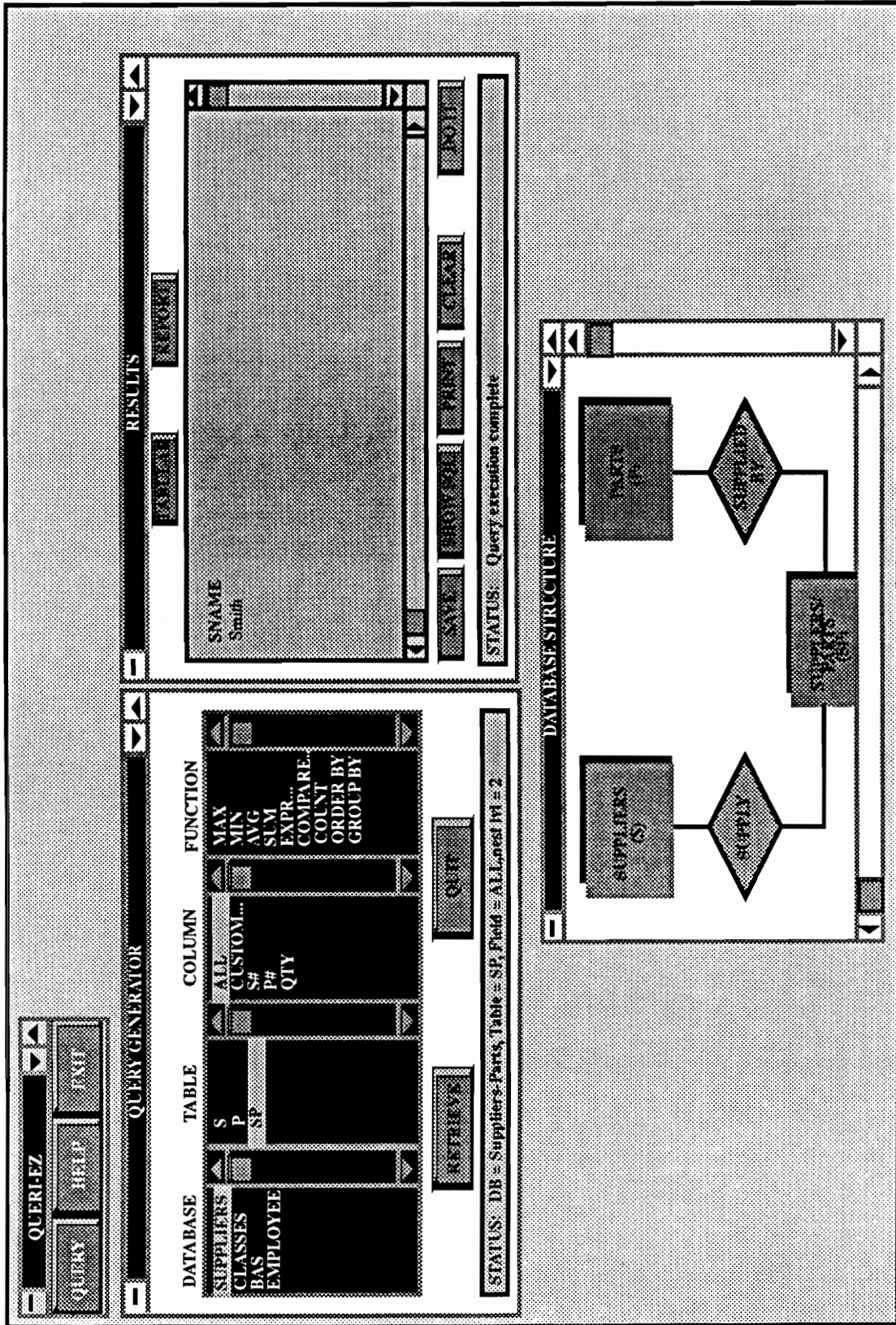


Figure 7.4.1.2.2.3b. Executing a query. Once the user completes the generation of the SQL statement, it is executed by pressing the "DO IT" button in the results window, and the results are displayed.

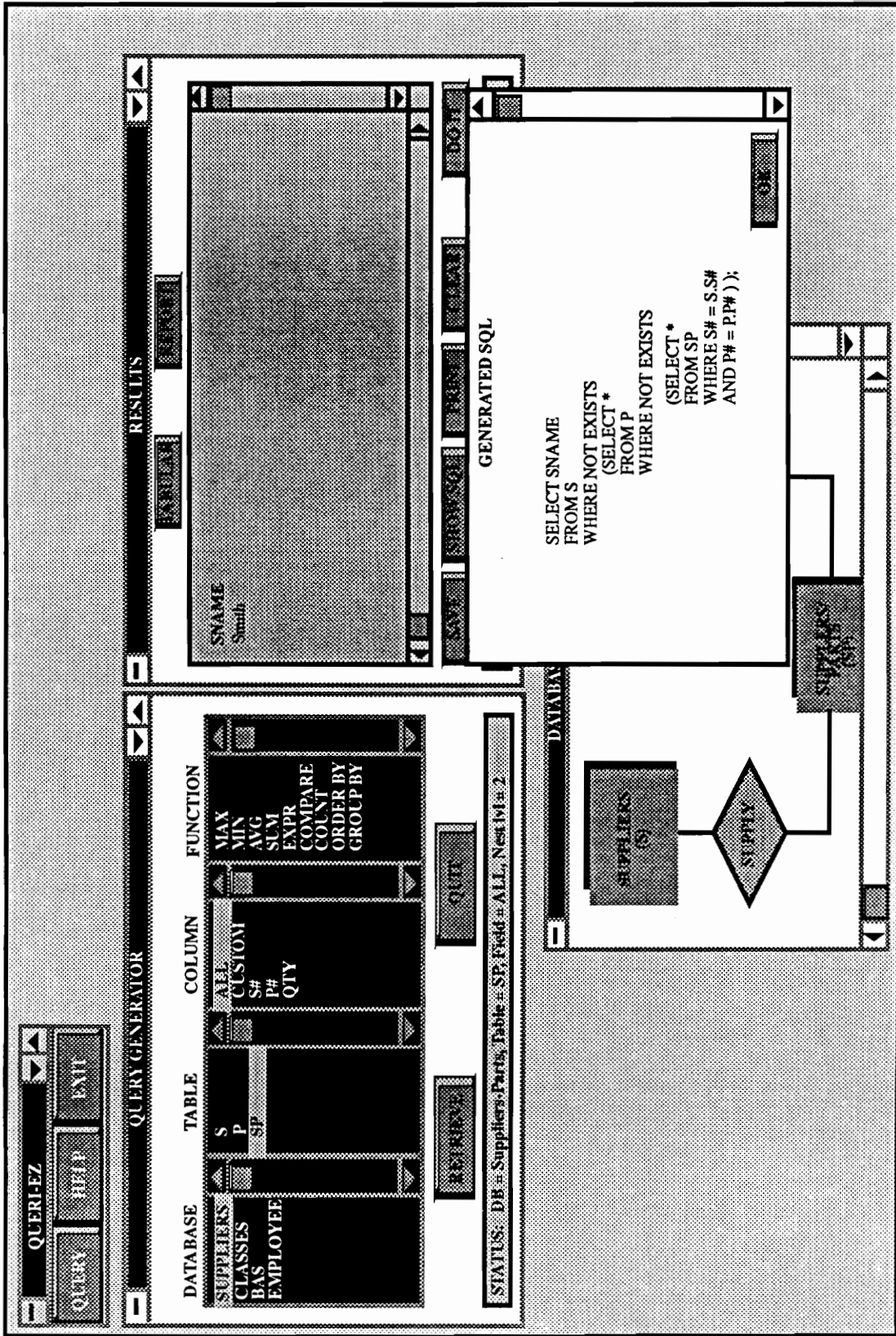


Figure 7.4.1.2.2.3c. Viewing generated SQL. Once the query has been generated, the user can view the SQL by selecting the "SHOW SQL" button.



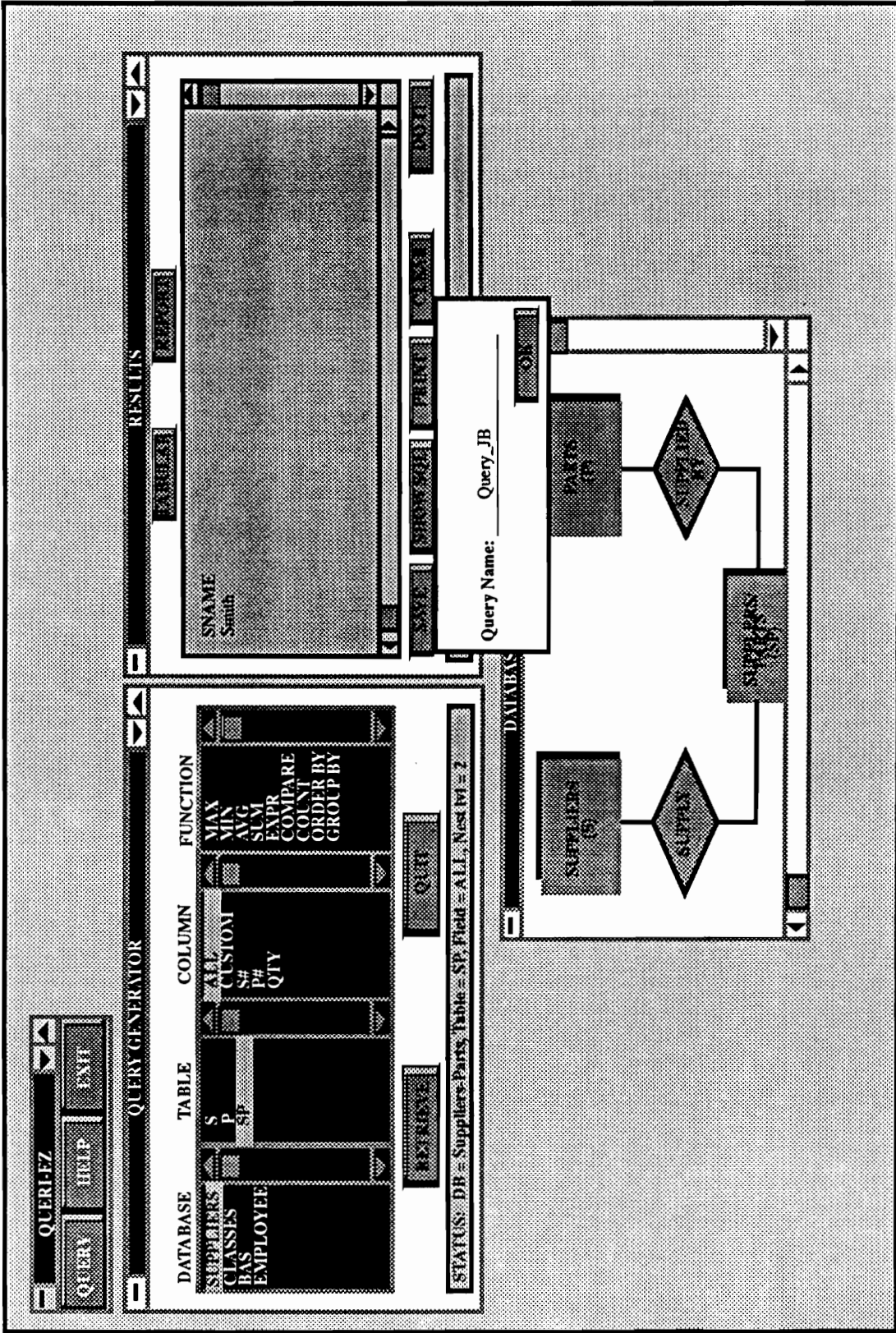


Figure 7.4.1.2.2.3d. Saving generated SQL. Once the query has been generated, the user can save the SQL by selecting the "SAVE" button.

### 7.4.1.3 State Transition Diagram

Normally, there would be a state transition diagram for each scenario. However, in the case of QUERI-EZ, the interface is very simple, and most actions are fairly repetitive. Therefore, the various states for the application, at any one instant in time, can be depicted in one diagram as shown in Figure 7.4.1.3.1. Each "bubble" represents a state of the interface, and the arrows represent the user action/selection which caused the state change. A state change is defined as any major alteration in the appearance of the QUERI-EZ interface.

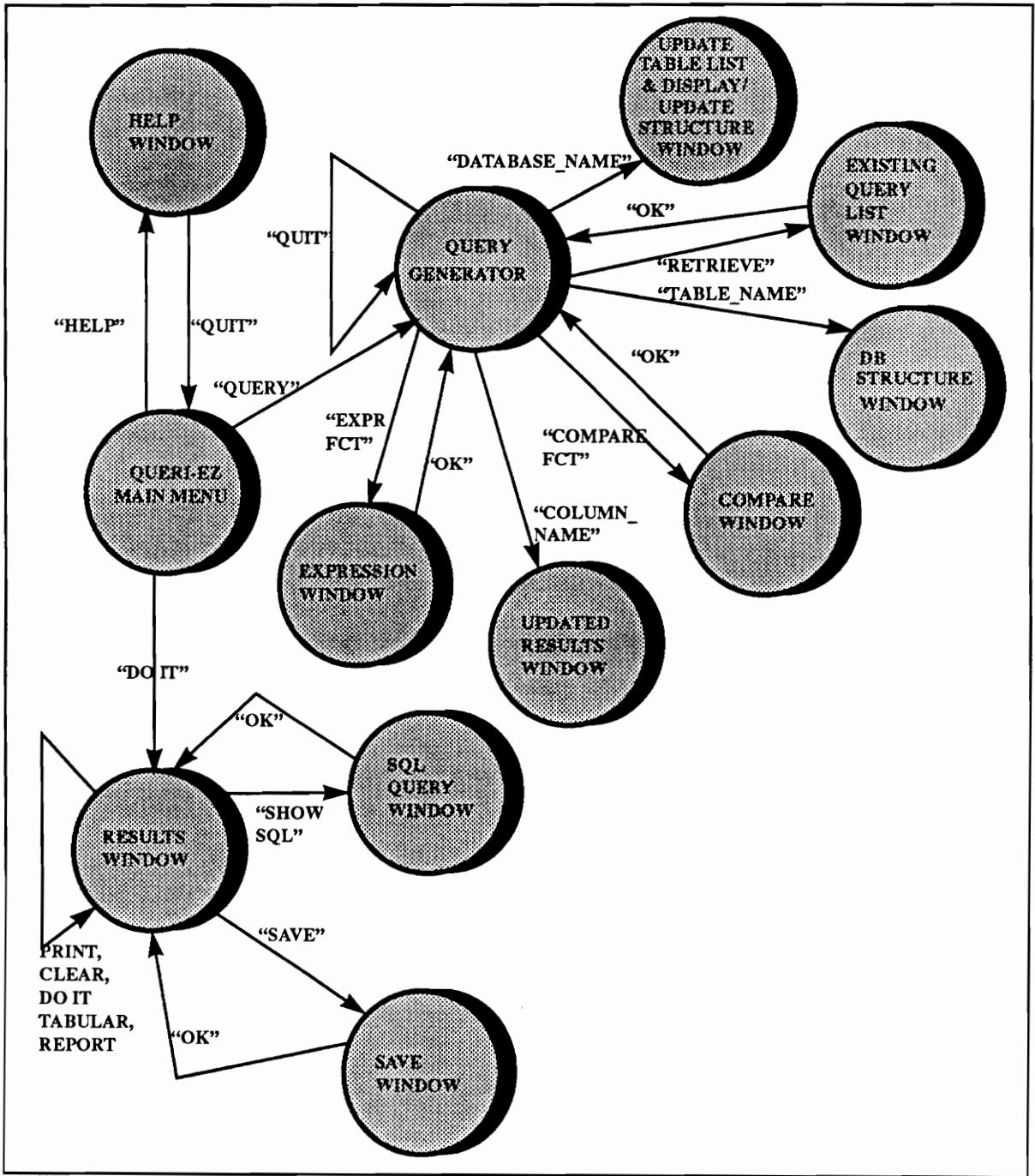


Figure 7.4.1.3.1. QUERI-EZ State Transition Diagram.

#### 7.4.1.4 User Action Notations

The user action notation is a "notation for behavioral representation of asynchronous, direct manipulation interface designs."<sup>7</sup> It combines notation for representing user actions, interface states, interface feedback, and computation connections to describe the user's interaction with the interface and the interface responses to those actions.

Table 7.4.1.4.1 (Hartson, 1990) provides a summary of the UAN symbols. Only a select group of the provided notations are used in the specifications for the QUERI-EZ interface. A notation is provided for each of the three scenarios presented in Sections 7.4.1.2.2 through 7.4.1.2.4.

**Table 7.4.1.4.1 Summary of UAN Symbols**

Action	Meaning
-	move the cursor
X	the context of object X, the "handle" by which X is manipulated
- X	move cursor into context of object X
-(x, y)	move the cursor to (arbitrary) point x, y outside any object
-(x, y in A)	move the cursor to (arbitrary) point within object A
-(X in Y)	move to object X within object Y (e.g., {OK_icon in dialogue_box})
[X]-	move cursor out of context of object X
v	depress
^	release
Xv	depress button, key, or switch called X
X^	release button, key, or switch X
Xv^	idiom for clicking button, key, or switch X
X"abc"	enter literal string, abc, via device X
X(xyz)	enter value for variable xyz via device X
{ }	grouping mechanism
*	iterative closure, task is performed zero or more times
+	task is performed one or more times
[]	enclosed task is optional (performed zero or one time)
A B	sequence: perform A, then B (same if A and B are on separate, but adjacent, lines)
OR	disjunction, choice of tasks (used to show alternative ways to perform a task)
&	order independence: connected tasks must all be performed, but relative order is immaterial
⇄	interleavability: performance of connected tasks can be interleaved in time
	concurrency: connected tasks can be performed simultaneously
⊥	task interrupt symbol: used to indicate that user may interrupt the current task at this point (the effect of this interrupt is specified as well, otherwise it is undefined, i.e., as though the user never performed the previous actions)
v	for all
•	separator between condition and action or feedback
Feedback	Meaning
h	highlight object
dh	dehighlight object
h'	same as h', but use an alternative highlight
h <sup>o</sup>	blink highlight
h <sup>o</sup> n	blink highlight n times
g(x, y)	at point x, y
g X	at object X
display(X)	display object X
erase(X)	erase object X
X > -	object X follows (is dragged by) cursor
X >> -	object X is rubber-banded as it follows cursor
outline(X)	outline of object X

Table 7.4.1.4.1 shows that the way to represent clicking a button, key, or switch of an object X is  $XV^{\wedge}$ . In the following UANs, the object is the mouse and is represented by the symbol M. Therefore clicking the mouse is represented as  $MV^{\wedge}$ .

#### 7.4.1.4.2. UAN Describing Scenario: Creating a Simple Query

In this scenario, the user is going to create a query that gets the part number and the total shipment quantity for that part, for each part supplied.

When first entering QUERI-EZ, the main menu is displayed, and in order to generate queries, the user must pull up the QUERY GENERATOR and RESULTS windows by selecting the QUERY button.

Task: Select "QUERY" main menu option			
User Action	Interface Feedback	Interface State	Connection to Computation
$\sim$ [QUERY_menu_item] $MV^{\wedge}$	QUERY_menu_item-!: QUERY_menu_item!		
	display(QUERY_GENERATOR_window) display(RESULTS_window)		

Task: Select mode of Results window			
User Action	Interface Feedback	Interface State	Connection to Computation
$\sim$ [REPORT_button in Results window] $MV^{\wedge}$	display (report mode in status area)		create text file for format

Now the user needs to select the database to use in generating the query.

Task: Select the "Suppliers-Parts" database			
User Action	Interface Feedback	Interface State	Connection to Computation
$\sim$ [Suppliers-Parts_database_item in Query Generator Window] $MV^{\wedge}$	Supplier-Parts_database_item-!: Supplier-Parts_database_item! Vdatabase_item':! Vdatabase_item'-! display(Database_Structure_Window) display[Supplier-Parts_tables]	selected = Supplier-Parts_database_item	ASCII text file updated with database name

Task: Select table "SP"			
User Action	Interface Feedback	Interface State	Connection to Computation
$\sim$ [SP_table_item in Query Generator window] $MV^{\wedge}$	SP_table_item-!: SP_table_item! Vtable_item':! Vtable_item'-! display (SP_table_relationship in Database Structure window) display (SP_table_col_items)	selected - SP_table_item	



Now the column names are selected.

Task: Select table "Part Number (P#)"			
User Action	Interface Feedback	Interface State	Connection to Computation
~[SP_table_item in Query Generator window]MV^	P#_col_item-!: P#_col_item! col_item'!: col_item'-! display(P#_col_item in results window);	selected = P#_col_item	SELECT and FROM clauses updated in ASCII text file

Task: Select the "QTY" column and use the SUM function on it			
User Action	Interface Feedback	Interface State	Connection to Computation
~[QTY_col_item in Query Generator Window]MV^	QTY_col_item-!: QTY_col_item! Vcol_item'!: Vcol_item'-! display (QTY_col_item in Results window)	selected = QTY_col_item	SELECT Clause written to text file
~[SUM_fct in Query Generator window]MV^	SUM_fct-!: SUM_fct! Vfunction'!: Vfunction'-! display (Sum applied in status area)	selected = SUM_fct	Update SELECT Clause

Now the user has to indicate that the results are to be grouped by part number.

Task: Select Group from the function list			
User Action	Interface Feedback	Interface State	Connection to Computation
~[GROUP_BY_fct in Query Generator window]MV^	display (pulldown_menu_list)		
~[P#_col_item in pulldown menu_list]MV^	display ("Group by Status" in Status area)	selected = P#	
~[GROUP_BY_fct in Query Generator window]MV^	erase (pulldown_menu_item)		ASCII file updated

Task: Format Results			
User Action	Interface Feedback	Interface State	Connection to Computation
~[P#_col_item in Results window]MV	P#_col_item-!: P#_col_item! col_item'!: col_item'-!	selected = P#_col_item	
~[x,y in Results window]*	outline-!		
M^	P#_col_item-!		x,y coordinates of P#_col_item updated to format file.

This task is also completed for the QTY column.

#### 7.4.1.4.2 UAN Describing Scenario: Retrieving an Existing Query

In order to retrieve an existing query, the user must select the "RETRIEVE" button in the Query Generator window.

Task: Retrieve "Query_JB"			
User Action	Interface Feedback	Interface State	Connection to Computation
~[RETRIEVE_button in Query Generator window]MV^	display(Query list)		
~[QUERY_JB_item in Query list]MV^	Query_JB_item-!: Query_JB_item! VQuery_item'!: VQuery_item'-!	selected = Query_JB_item	
~[OK_button in Query list]MV^	erase (Query List)		
			retrieve Query_JB_file
	display ("query loaded" in status area		

Task: Execute "Query_JB"			
User Action	Interface Feedback	Interface State	Connection to Computation
~[DO_IT_button in Results_window]MV^			ASCII file sent to ISQL
	display (Query Results)		

### 7.4.1.4.3. UAN Describing Scenario: Creating a Complex Query

When first entering QUERI-EZ, the main menu is displayed, and in order to generate queries, the user must pull up the QUERY GENERATOR and RESULTS windows by selecting the QUERY button. As with scenario 2, it is assumed that the user as already entered the application.

The user needs to select the database to use in generating the query.

Task: Select the "Suppliers-Parts" database			
User Action	Interface Feedback	Interface State	Connection to Computation
~[Suppliers-Parts_database_item in Query Generator Window]MV^	Supplier-Parts_database_item-!: Supplier-Parts_database_item! Vdatabase_item'!: Vdatabase_item'-! display(Database_Structure_Window) display[Supplier-Parts_tables]	selected = Supplier-Parts_database_item	ASCII text file updated with database name

Task: Select table "S"			
User Action	Interface Feedback	Interface State	Connection to Computation
~[S_table_item in Query Generator Window]MV^	S_table_item-!: S_table_item! Vtable_item'!: Vtable_item'-! display(S-table_col_items) display(S_table_relationships in Database Structure Window)	selected = S_table_item	

Now the user selects the name of the table that they want to select columns from.

Task: Select the "SNAME" column			
User Action	Interface Feedback	Interface State	Connection to Computation
~[SNAME_col_item in Query Generator Window]MV^	SNAME_col_item-!: SNAME_col_item! Vcol_item'!: Vcol_item'-! display(SNAME_col_item in Results window)	selected = SNAME_col_item	FROM and SELECT clauses added to the ASCII text file

Since the user needs to know the names of suppliers, the SNAME column needs to be selected to indicate that it is part of the output.

Task: Select the "SNAME" column			
User Action	Interface Feedback	Interface State	Connection to Computation
~[SNAME_col_item in Query Generator Window]MV^	SNAME_col_item-!: SNAME_col_item! Vcol_item'!: Vcol_item'-! display(SNAME_col_item in Results window)	selected = SNAME_col_item	FROM and SELECT clauses added to the ASCII text file

Since the user is asking for suppliers, such that there doesn't exist a part that they do not supply, a nested query must be formed with multiple levels. The user can indicate the conditions by selecting the comparisons from the function list, and then selecting the "NOT EXISTS" operator from the comparison window.

Task: Select the "NOT EXISTS" operator			
User Action	Interface Feedback	Interface State	Connection to Computation
~[compare_fct in Query Generator window]MV^	compare_fct-!: compare_fct! Vfunction'!: Vfunction'-! display(compare_Window)	selected = compare_fct	
~[op_pulldown_menu in compare window]MV^	display(pulldown_menu_list)		
~[NOT_EXISTS_op in pulldown_menu_list]MV^	NOT_EXISTS_op-!: NOT_EXISTS_op! Voperator'!: Voperator'-!	selected = NOT_EXISTS_op	
~[OK_button in compare window]MV^	erase(compare window)		WHERE clause added to ASCII text file Nest level incremented

Now the selection from the parts table needs to be done. This can be done by repeating the tasks performed above for selecting the table and column names. The same is true for the last portion of the SQL statement. The only thing left to do is specify the WHERE clause.

Task: Generate the condition where the supplier and part numbers in the "SP" table are equal to the corresponding supplier and part numbers in the suppliers and parts tables, respectively.			
User Action	Interface Feedback	Interface State	Connection to Computation
~[compare_fct in Query Generator window]MV^	compare_fct-!: compare_fct! Vfunction'!: Vfunction'-! display(compare window)	selected = compare fct	
Compound Stmt Clause: ~[AND_button in compare window]MV^			
~[left_operand_field in compare window]MV^	Cursor!!		
left_operand_field "S#"			

Task: Generate the condition where the supplier and part numbers in the "SP" table are equal to the corresponding supplier and part numbers in the suppliers and parts tables, respectively. (Con't)			
User Action	Interface Feedback	Interface State	Connection to Computation
~[op_pulldown_list in compare window]MV^	display(pulldown_menu_list)		
~[equal_op in pulldown_menu_list]MV^	equal_op-! equal_op! Voperator'! Voperator'-!	selected = equal_op	
~[right_operand_field in compare window]MV^	Cursor!!		
right_operand_field "S.S#"			
~[OK_button in compare window]MV^	erase(compare window)		WHERE clause added to ASCII text file

This task is repeated for the compound WHERE clause.

Task: Execute SQL statement			
User Action	Interface Feedback	Interface State	Connection to Computation
~[DO_IT_button in Results Window]MV^			ASCII text file updated, closed and sent top ISQL
	display (Query Results)		

Task: View generated SQL			
User Action	Interface Feedback	Interface State	Connection to Computation
~[ShowSQL_button in Results window]MV^	display (SQL window)		
~[OK_button in SQL window]MV^	Cursor!!		

Task: Save generated SQL			
User Action	Interface Feedback	Interface State	Connection to Computation
~[Save_button in Results window]MV^	display(save window)		
~[filename field in Save window]MV^	cursor!!		
filename field "Query_JB"			
~[OK_button in Save window]MV^	erase(save window)		query saved to file Query_JB

#### **7.4.1.5 User Interface Verification/Evaluation**

As discussed in Section 4, in order to determine if the user is comfortable with the system and in particular, the system interface, a prototype of the user interface is presented in the form of a usability lab. This section describes the way in which the usability lab will be organized to evaluate QUERI-EZ.

Evaluating the user interface requires the generation of Benchmark Tests, a Usability Specification, and a Subjective Evaluation, each of which is utilized in the Usability Lab. The Benchmark Tests are specified system tasks that the user is required to perform during system use. For each task, there is a corresponding usability specification which documents objective quantities like the expected number of steps to perform a specific task. When the tasks are complete, the designer can compare the specified results with the actual results to determine if the tasks were performed as expected. Once the user has completed the benchmark tests, the subjective evaluation is administered to the user to give him/her an opportunity to rate the system in the areas of: (1) general feeling, (2) the screen, (3) terminology and system information, (4) learning, and (5) system capabilities. After all of these phases are complete, the designer can pull all of the information together to perform a statistical analysis and formative evaluation. This evaluation results in recommended changes to the interface which would increase the user's acceptance of the system. Once these changes are implemented, the usability labs are re-opened, and the cycle is repeated. The entire process is repeated until the optimal interface is produced, yielding optimal system usability.

The QUERI-EZ usability lab will be administered to a group of two hundred users with abilities that range from those of a novice user to those of an intermediate user. There will be 100 novice and 100 intermediate users tested., In order to properly categorize the users, interviews will be conducted to determine the level of understanding of a graphical user interface as well as the mouse usability. From each user, during benchmark testing, the following objective measures will be taken:

- 1) The number of steps to complete the specified tasks. A step will be considered a mouse click (except when specifying operations in the Results window that require a double-click). Due to the fact that every users typing ability may vary, the number of keystrokes required when typing field names is not taken into account.
- 2) The amount of time it takes to complete a task.
- 3) The number of mistakes that were made in carrying out the task.

From each of these measures, for each user, the mean and standard deviation from the mean will be computed. In addition to the standard deviation from the mean, the standard deviation from the expected number of steps is also computed. These objective measures will be analyzed over time (i.e. multiple usability lab sessions). The goal is for the difference between the statistical measure for each user and the expected values to continually decrease until the difference is as small as possible. Tables 7.4.1.5.2.1 and 7.4.1.5.2.2 show the forms that will be used to capture the objective statistics. Figure 7.4.1.5.3.1 shows the form that will be used to capture the subjective responses of each user. It is expected that there will be a distinct correlation between the subjective evaluations and the objective statistics. As the user becomes more familiar with the system, the simpler system operation should become. Over time, this theory should be supported by the objective and subjective measures taken from the user. Future enhancements of the QUERI-EZ application will be evaluated based on these results. Table 7.4.1.5.1.1 shows a sample Benchmark Test form.

### 7.4.1.5.1 Benchmark Tests

During the iterative testing phase, the user is presented with the form provided below (Table 7.4.1.5.1.1) which walks the user through the steps to be performed.

Table 7.4.1.5.1.1. QUERI-EZ Benchmark Test

<p style="text-align: center;"><b>QUERI-EZ</b> <i>Benchmark Tests</i></p> <p><b><u>Instructions:</u></b> <i>Perform each of the tasks listed below. You are not under any time constraints therefore, do not rush through execution of the tasks. If helpful, talk yourself through the steps you are taking. Verbally express areas of confusion.</i></p> <p><b><u>TASKS: All tasks are based on the Suppliers-Parts Database.</u></b></p> <ol style="list-style-type: none"><li>1) Get the supplier numbers for suppliers in Paris with a status less than 20.</li><li>2) Retrieve all combinations of suppliers and part information such that, the supplier and part in question are located in the same city.</li><li>3) Determine the names of the suppliers who supply at least one read part.</li><li>4) View the SQL generated in task 3.</li><li>5) Save the SQL generated in task 3.</li><li>6) Repeat task 1, renaming the S# field to the supplier number.</li><li>7) Repeat task 2, grouping the output by the supplier number.</li><li>8) Determine the total quantity of part P2 supplied.</li><li>9) Repeat task 8, doubling the total quantity output. <b>Hint:</b> Use the EXPR function against the quantity column. When the Expression window appears, enter an expression that would double the quantity value.</li><li>10) Get all parts whose name begins with the letter C.</li></ol>
---



### 7.4.1.5.2 Usability Specifications

The Usability Specification for QUERI-EZ is comprised of the expected measure values for user performance and a form which is used to document actual user performance for each of the specified tasks in the Benchmark Testing. Tables 7.4.1.5.2.1 and 7.4.1.5.2.2 show the forms used to document this data.

**Table 7.4.1.5.2.1. QUERI-EZ Usability Specification - Performance Expectation**

<div style="border: 1px solid black; border-radius: 15px; padding: 5px; background-color: #e0e0e0;"> <b>QUERI-EZ</b>  <i>Usability Specifications</i> </div>		
<b><u>TASKS: All tasks are based on the Suppliers-Parts Database.</u></b>		
1) Get the supplier numbers for suppliers in Paris with a status less than 20.		
2) Retrieve all combinations of suppliers and part information such that, the supplier and part in question are located in the same city.		
3) Determine the names of the suppliers who supply at least one read part.		
4) View the SQL generated in task 3.		
5) Save the SQL generated in task 3.		
6) Repeat task 1, renaming the S# field to the supplier number.		
7) Repeat task 2, grouping the output by the supplier number.		
8) Determine the total quantity of part P2 supplied.		
9) Repeat task 8, doubling the total quantity output.		
10) Get all parts whose name begins with the letter C.		
<b><u>Specification of Quantitative Measures:</u></b>		
<u>Task Number</u>	<u>Expected Number of Steps</u>	<u>Expected Time to Complete</u>
1	.	.
2	.	.
3	.	.
4	TBD	TBD
5		
6	by designer using	by designer using
7	actual prototype	actual prototype
8	.	.
9	.	.
10	.	.

**Table 7.4.1.5.2.2. QUERI-EZ Usability Specification - Actual Performance Measures.** There is a one-to-one relationship between this form and an individual task in the Benchmark Test.

<b>QUERI-EZ</b> <i>Usability Specifications</i>			
Date of Test: _____ QUERI-EZ Version: _____			
<u>Task # 1</u>			
<u>Subject</u>	<u>Number of Steps</u>	<u>Time</u>	<u>Number of Mistakes</u>
A-N	_____	_____	_____
B-N	_____	_____	_____
C-N	_____	_____	_____
D-N	_____	_____	_____
E-N	_____	_____	_____
Average	_____	_____	_____
Standard Dev. from Mean	_____	_____	_____
Standard Dev. from Expected	_____	_____	_____
A-I	_____	_____	_____
B-I	_____	_____	_____
C-I	_____	_____	_____
D-I	_____	_____	_____
E-I	_____	_____	_____
Average	_____	_____	_____
Standard Dev. from Mean	_____	_____	_____
Standard Dev. from Expected	_____	_____	_____
Key: 1) N = Novice User/Non-Windows Users 2) I = Intermediate User/Windows Knowlegable			

### 7.4.1.5.3 Subjective Evaluation

After completing the Benchmark Test, the user is required to fill out the evaluation in Table 7.4.1.5.3.1. This will give the designer a feel for the user's appreciation of the system. The questions that comprise the questionnaire is a subset of those found in the QUIS questionnaire (Shneiderman, 1992).

**Table 7.4.1.5.3.1. QUERI-EZ Subjective Evaluation Form - User Questionnaire**

**QUERI-EZ**  
*Subjective Evaluation*

Date of Test: \_\_\_\_\_  
 QUERI-EZ Version: \_\_\_\_\_

Please answer the following questions about QUERI-EZ, in order for us to evaluate your satisfaction with the system. In addition, please provide any additional comments you might have so that QUERI-EZ can be tailored to best suit your needs.

<p><b>(1) General Feeling</b></p> <p>1. Overall reactions to the system              terrible                      wonderful              1 2 3 4 5 6 7 8 9</p> <p>2. frustrating                      satisfying              1 2 3 4 5 6 7 8 9</p> <p>3. dull                      stimulating              1 2 3 4 5 6 7 8 9</p> <p>4. difficult                      easy              1 2 3 4 5 6 7 8 9</p> <p>5. rigid                      flexible              1 2 3 4 5 6 7 8 9</p> <p>6. Exploration of features by trial and error?              discouraging                      encouraging              1 2 3 4 5 6 7 8 9</p>	<p><b>(3) Terminology and Information</b></p> <p>9. Use of terminology?              inconsistent                      consistent              1 2 3 4 5 6 7 8 9</p>
<p><b>(2) The Screen</b></p> <p>7. Were screen layouts helpful?              not at all                      very              1 2 3 4 5 6 7 8 9</p> <p>8. The sequence of screens?              confusing                      clear              1 2 3 4 5 6 7 8 9</p>	<p><b>(4) Learning</b></p> <p>10. Learning to operate the system?              difficult                      easy              1 2 3 4 5 6 7 8 9</p>
<p><b>(5) System Capabilities</b></p> <p>11. inadequate power                      adequate power              1 2 3 4 5 6 7 8 9</p>	

**Additional Comments:**

## 7.4.2 Software Design

This section includes a context diagram and a data flow diagram which are intended to depict the organization of the software. Also included is pseudocode for each of the processes in the application. The pseudocode is provided to express the purpose each process is intended to serve in the QUERI-EZ application and not specific coding details.

Here is a good point to reiterate that this is by no means intended to be a complete attempt at software design. It is merely a compact means of expressing the operation of the software at a high level.

The GUI environment forces an event driven application. This means that particular areas of the software are executed because of some action the user has taken at the interface level. In other words, there is a direct connection between the object of interest on the interface and the action(s) performed by the software. The event driven nature of the GUI basically forces a more object-oriented type of behavior such that the structured concepts apply to the way the software behaves in relation to a user action.

### 7.4.2.1 Context Diagram

The context diagram shown earlier, in Figure 7.2.2.1.1, depicts the QUERI-EZ application from a conceptual point of view.

### 7.4.2.2 Data Flow Diagram

The data flow diagram of Figure 7.4.2.2.1 shows the processes of QUERI-EZ and the data that flows between them. The dashed lines represent control flows. These flows indicate the order in which processing between two processes must occur. The lines going into and out of the processes are labelled with the type of data they receive and/or produce.

Each of the processes exists at the same level of the application. That is, none of the processes are comprised of subprocesses. They are leaf processes with no children processes, and they may communicate with each other.

The pseudocode algorithms presented in section 7.4.2.3 describes the behavior of these processes.

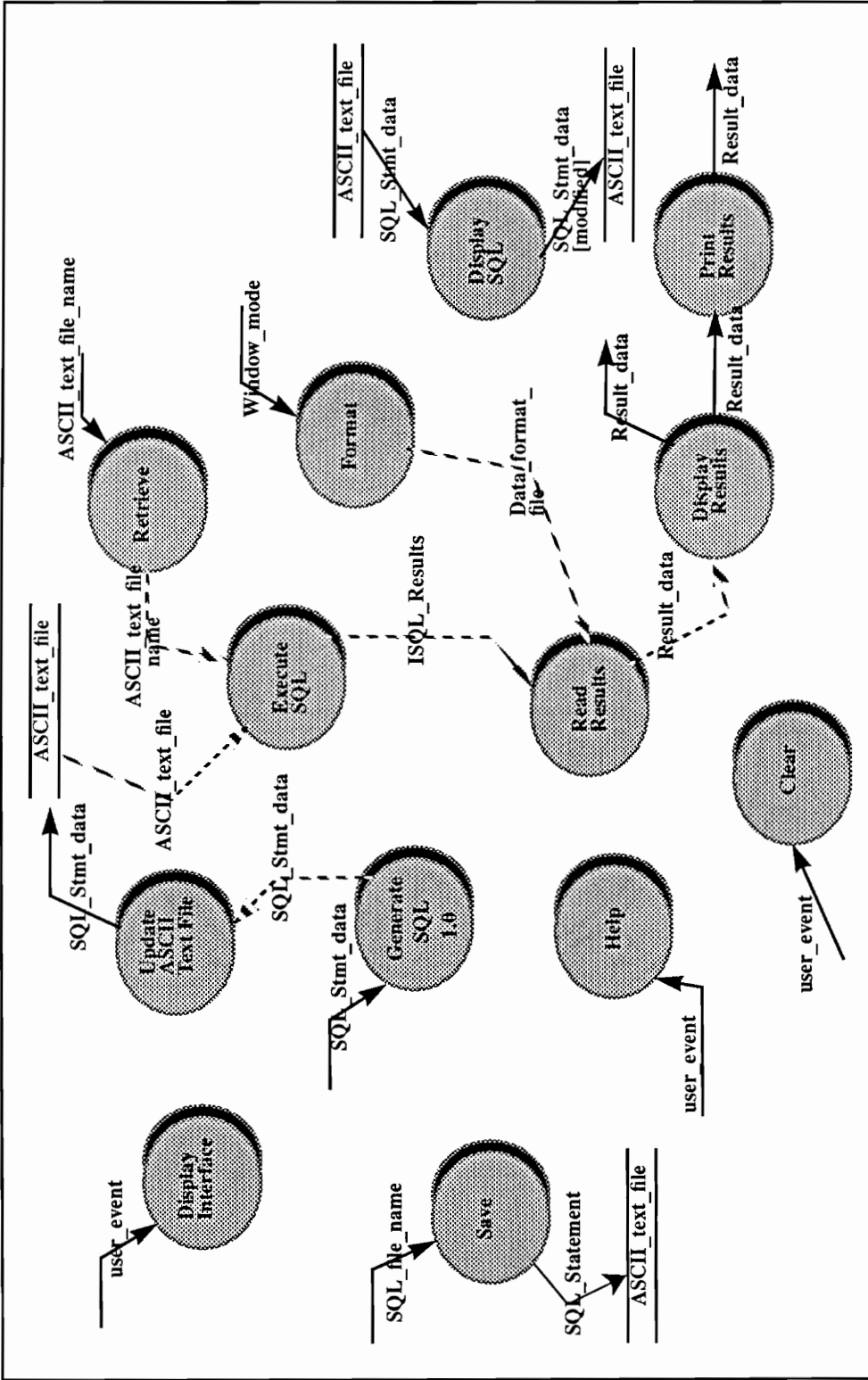


Figure 7.4.2.2.1. QUERI-EZ Data Flow Diagram

### 7.4.2.3 Pseudocode Algorithms

This section contains two levels of pseudocode algorithms that are provided to give a better understanding of how QUERI-EZ operates behind the interactive interface. The first level is a general view of QUERI-EZ and contains a high level algorithm for each of the processes identified in the Data Flow Diagram of Figure 7.4.2.2.1. The second level provides algorithms for each of the requirements listed in the Software Requirements Specification (Section 7.3.2.1).

#### 7.4.2.3.1. General Pseudocode Algorithms

The pseudocode algorithms provided in Figure 7.4.2.3.1 through 7.4.2.3.12 describe the processing that needs to occur in each of the processes shown in Figure 7.4.2.2.1. The lowest level of the figure numbers correspond to the process numbers in the Data Flow Diagram.

```

OPEN ASCII text file
SET default mode of results
INITIALIZE nest level
REPEAT
    RETRIEVE data from screen
    CALL UPDATE to update SQL ASCII text file
    DISPLAY status
    IF column selected THEN
        IF "CUSTOM" column selected THEN
            DISPLAY "CUSTOM" window
            RETRIEVE user input
            DISPLAY status
        END
        CALL FORMAT to set field attributes
            appropriately
        DISPLAY status
    END
    IF function selected THEN
        IF function = "EXPR" THEN
            DISPLAY Expression Dialogue
            RETRIEVE input data
            CALL UPDATE to update the SQL
                ASCII text file
            DISPLAY status
        END
        ELSE
            IF function selected = "COMPARE"
                THEN DISPLAY Comparison Dialogue
            RETRIEVE INPUT
            CALL UPDATE to update the SQL
                ASCII text file
            IF Comparison operator is a nest
                operator THEN
                INCREMENT nest level by 1
                DISPLAY status
            END
        END
    END
END
UNTIL "Execution" requested

```

Figure 7.4.2.3.1.1. Generate (SQL)

```
IF "RETRIEVE" selected THEN
  DISPLAY the Retrieve list box
  RETRIEVE format file
  RETRIEVE query file
  DISPLAY status
END
```

**Figure 7.4.2.3.1.2. Retrieve (Existing Query)**

```
IF "DO IT" button pressed THEN
  CLOSE SQL statement (i.e. closing parenthesis, semicolon,
  etc.)
  CLOSE ASCII Text files (Format and QUERY files)
  INVOKE ISQL with the SQL text file text file as data
  TRAP RESULTS returned from the ISQL execution in an
  ASCII text file
END
```

**Figure 7.4.2.3.1.3. Execute (SQL)**

```
READ the ASCII text file, discarding TDS informational data
READ format file to determine the format of the results
READ actual query results
DISPLAY results in user format in the results window
```

**Figure 7.4.2.3.1.4. Read Results**



```
RETRIEVE mode from user entry
ASSIGN attributes to field names
RETRIEVE field coordinates from results window and
WRITE to format text file
```

**Figure 7.4.2.3.1.5. Format (Results)**

```
IF "Clear" button pressed THEN
  REFRESH results screen
  ERASE contents of the format file
  ERASE contents of the SQL file
  CLOSE all files
END
```

**Figure 7.4.2.3.1.6. Clear (Results Window)**

```
IF "Save" button pressed THEN
  DISPLAY SAVE Dialogue
  RETRIEVE users input
  SAVE query to named file
  IF specified file exists and user wants to delete it THEN
    OVERWRITE existing file
  ELSE
    CLEAR filename from dialogue window and
    wait for new input
  END
END
```

**Figure 7.4.2.3.1.7. Save (Generated Query)**

```
IF "Print" button pressed THEN
  PRINT contents of the Results window
END
```

**Figure 7.4.2.3.1.8. Print (Results Window)**

```
WRITE query results to the results window in the format
specified by the user
```

**Figure 7.4.2.3.1.9. Display Results**

```
IF "Show SQL" button selected THEN
  OPEN new window
  DISPLAY SQL ASCII Text in the window
END
```

**Figure 7.4.2.3.1.10. Display SQL**

```
IF "Query" menu button selected THEN
  DISPLAY the "Query Generator" window
  DISPLAY the RESULTS window
END
```

**Figure 7.4.2.3.1.11. Display Interface**

```
IF "Help" main menu button selected THEN
  DISPLAY scrollable window containing a file
  explaining the application
END
```

**Figure 7.4.2.3.1.12. Help**

#### **7.4.2.3.2 Pseudocode Algorithms for the Requirements**

The pseudocode algorithm provided in Figures 7.4.2.3.2.1 through 7.4.2.3.2.12 describe the way in which the software will satisfy the Functional Requirements of Section 7.3.2.1. For areas where various requirements overlap, the algorithm is provided only once.

```

IF function selected THEN
  IF (function = GROUP BY) or (function = ORDER BY)
    THEN UPDATE GROUP BY or ORDER
      BY clause in the SQL text file
    DISPLAY status
  END
  IF (function <> Expr) and (function <> Compare) THEN
    UPDATE SELECT clause field
      name with the appropriate
      aggregate function
    DISPLAY status
  ELSE
    IF function = "Expr" THEN
      DISPLAY Expression window
      GET user input
      UPDATE the SELECT clause in the SQL text file
      DISPLAY status
    ELSE
      DISPLAY Compare window
      GET user input
      UPDATE WHERE clause in SQL text file
      DISPLAY status
    END
  END
END
UNTIL "Results/DO IT" button selected

```

Note: The queries that pull the data in the Query Generator scroll windows are the result of Embedded SQL queries. They are executed each time the user selects fields in these windows. As a result, if any changes occur to the database, the application does not need to be changed. It always supplies the user with information as correct as the database to which it is connected.

**Figure 7.4.2.3.2.1. Generating Simple Queries (con't)**

```

IF function selected THEN
  REPEAT
    IF function = "Expr" THEN
      DISPLAY Expression window
      (All user columns currently appearing in SELECT
      clause are available in the pull down menu)
      GET user data
      UPDATE SELECT clause in the SQL text file to reflect
      the expression
      DISPLAY status
    END
  UNTIL function <> "Expr"
END

```

**Figure 7.4.2.3.2.2. Retrieving Computed Values**

```

IF function selected THEN
  REPEAT
    IF function = "COMPARE" THEN
      READ SELECT statement in text file looking for
      WHERE clause
      IF found THEN
        DISPLAY Compare window with boolean operators
        (AND and OR) grayed out
      ELSE
        DISPLAY Compare window with boolean values
        active
      END
      RETRIEVE user supplied data
      UPDATE WHERE clause in ASCII text file
      DISPLAY status
    END
  UNTIL function <> "COMPARE"
END

```

**Figure 7.4.2.3.2.3. Placing Condition on queries**

```

INITIALIZE counter to current order by count + 1
IF function selected THEN
  IF function = ORDER BY THEN
    DISPLAY Pull down list
    REPEAT
      GET user column selection
      ASSIGN selection a position (based on the counter)
      INCREMENT the counter
    UNTIL pull down list closed
  END
END
END

```

**Figure 7.4.2.3.2.4. Ordering rows of Output by Column**

```

INITIALIZE counter to current group by count + 1
IF function selected THEN
  IF function = ORDER BY THEN
    DISPLAY Pull down list
    REPEAT
      GET user column selection
      ASSIGN selection a position (based on the counter)
      INCREMENT the counter
    UNTIL pull down list closed
  END
END
END

```

**Figure 7.4.2.3.2.5. Grouping rows of Output by Column**



```
IF table selected THEN
  READ SQL statement
  IF FROM clause exists THEN
    READ to end of clause
    WRITE a comma to separate table names
    UPDATE the FROM clause in the SQL text file with the
      table name
  END
END
```

**Figure 7.4.2.3.2.6. Joining tables together**

```
IF function selected THEN
  IF function = aggregate function THEN
    READ SQL statement looking for currently selected file
      name
    UPDATE SELECT clause field name to reflect applied
      function
    DISPLAY status
  END
END
```

**Figure 7.4.2.3.2.7. Utilizing aggregate functions**

```
IF function selected THEN
  IF function = Compare THEN
    IF operator in ("IN", "NOT IN", "EXISTS", "NOT EXISTS")
      THEN UPDATE WHERE clause in SQL statement
      WRITE and open parenthesis "(" to SQL text file
      INCREMENT nesting level by one
    END
  END
END
```

**Figure 7.4.2.3.2.8. Generating Subqueries with multiple levels of nesting**

```
SET mode = Tabular
IF Report button selected THEN
  mode = Report
END
```

**Figure 7.4.2.3.2.9. Results window mode specification**

```
IF user "double-clicks" on field in Results window THEN
  HIGHLIGHT field for modifying
  GET user supplied field name
  UPDATE format text file
  DISPLAY status
END
```

**Figure 7.4.2.3.2.10. Renaming Output Fields**



```
IF any user action THEN
  system responds
  standard message retrieved
  PRINT message to status area of the appropriate window
END
```

**Figure 7.4.2.3.2.11. Providing the user with status**

```
IF "Quit" button selected THEN
  ERASE SQL and format ASCII files
  CLOSE SQL and format ASCII files
  CLEAR Results window
  DE-SELECT all items selected
END
```

**Figure 7.4.2.3.2.12. Re-Starting a Query**

## 8.0 CONCLUSIONS

### 8.1 Summary

In review, this project/report has combined several major areas in the computer science/information science industry to offer an improved alternative to data access in relational database management systems.

It has been shown that direct SQL based querying can be a very intimidating process for a user who does not possess strong technical abilities. Therefore, to alleviate the anxiety produced by such an intimidating process, the designer is presented with the challenge of applying the appropriate concepts to reduce, if not eliminate this anxiety. It has also been shown that the concept of direct manipulation is appealing to most users, particularly, the non-programmer type of user, because it speeds up the learning process. The user can see the direct effect of his/her actions, and as a result, the user feels more in control of the system. And finally, it has been shown that there are some systems in existence today that attempt to meet the challenge of providing simple and intuitive data access systems that can be useful, to some degree, to all levels of users. And although these systems have made good attempts, there is definite room for improvement. The way to improve is to place the user at the center of the design. If the designer starts with this user-centered approach, he/she ends up with a user that feels that the world of the application revolves around him/her; thus, implying that he/she is important and in control. The result is a successful user driven application.

### 8.2 Evaluation of the Application Design

Based on an initial review by the designer, QUERI-EZ meets the goals of a successful application design because it is simple and intuitive. It gives the user ample functionality with reduced anxiety - the user is in control. There are no commands or language syntax to remember, and there is no deep menu hierarchy to become familiar with. Everything the user wants and/or needs to do is encompassed within the three windows which comprise the interactive interface, and the exceptions are minimal. And although QUERI-EZ is designed for the novice/intermediate user, it also lends itself to the experienced programmer.

QUERI-EZ is also designed to be an easily portable application because the

contents of the interactive interface is always done at run-time. In other words, a dynamic database structure does not limit QUERI-EZ's ability to provide the user with access to the current database data or prohibit the user from accessing data that doesn't exist. This reduces the need for the obvious types of error checking.

One good way to show the assets of QUERI-EZ, is to compare it to a commercial system attempting to meet the same goals. Table 8.2.1 shows a comparison of QUERI-EZ and GQL. Of the current systems evaluated, GQL is most like QUERI-EZ.

**Table 8.2.1. Comparison of QUERI-EZ and GQL**

	<b>Graphical Query Language (GQL)</b>	<b>QUERI-EZ</b>
Generating Queries	Simple	Simple
Performing Joins	Can get cumbersome if data model window becomes nested	Simple; simply click on the table names
Window Layers Possible	Many	None; only the three windows on the interface
Use of graphics to represent database tables	Good; This is a way to avoid non-descriptive table names	Not used; Does not allow for unambiguous table names
Promise of accurate results	Only if there is a guarantee that the underlying database structure does not change	Yes; database entity are dynamically made available to the user
Ease of database switching	Cumbersome; must exit the system and re-enter	Easy; simply click on the new database name

The ability to easily add new functionality to QUERI-EZ, without changing the appeal of the simple interface is also a promising feature of the application. The following is a list of possible enhancements to the QUERI-EZ application:

1) The COMPARE and EXPR function windows could be modified to allow the user to enter as many constraints and/or expressions as possible in one dialogue session by adding another button labelled "APPLY" and changing the current button label to "CLOSE". This way after each condition or expression has been entered, the associated SQL text file would be updated when the "APPLY" button is pressed, and the user would not be required to repeatedly bring up the dialogues.

2) The tables in the Data Structure window could be graphical icon representations of the table content.

3) An "APPLY" and "CLOSE" button could be added to the SQL View window to allow the user to update SQL statement data. This would definitely appeal to programmer types.

4) The SAVE and RETRIEVE dialogues could include a list of the files and directory structures to further reduce user memory requirements.

5) The REPORT mode of the RESULTS window could be enhanced to allow the user to specify the FONT, SIZE, ALIGNMENT, and STYLE of report titles, as well as specify the date and time the report was produced.

As do most systems, QUERI-EZ does have drawbacks. First, because QUERI-EZ is actually run on top of an interface (an ISQL application), the user will definitely notice a performance hit. QUERI-EZ has to formulate the user's request, send it to the ISQL application to be processed, trap the returned results, and format the results as specified by the user. The question becomes: Does the time QUERI-EZ saves the user in formulating the request for data cover the additional time required to allow QUERI-EZ to act as an indirect interface to the ISQL

application?

Another drawback of QUERI-EZ is that the user must understand the organization of the data within the relational database. He/she must understand what it is they want and where it lives in the database. In essence, QUERI-EZ is only removing the ability for the user to have to know SQL. The user's understanding of what SQL operates on must exist.

Also, QUERI-EZ does not perform semantic validation on a user's query. If it does not make logical sense to the RDBMS, although it may be correct in terms of the syntax, QUERI-EZ simply acts as the ISQL application acts. Semantically incorrect queries do not return results in an ISQL application therefore, QUERI-EZ does not return any results.

An area of discussion that arises with QUERI-EZ is its true ability to shield the user from complex queries. In other words, does the user really have the ability to construct a complex SQL query without having to understand the structure of SQL? The answer is probably no. However, if the user is truly a non-programmer type, would he/she attempt to formulate a query the caliber of the one presented in section 7.4.1.2.2? Again the answer is probably no. Basically, what QUERI-EZ provides the novice/intermediate user with is the ability to formulate simple queries, and it provides the experienced user with the ability to formulate both simple and complex queries. Regardless of the level of complexity of the query, the experienced user is saved from a lot of typing mistakes and has direct visual reference to the database structure. This is also the case for the novice user, but the later may or may not be appreciated as much. So, should the user group focus be altered because of the fact QUERI-EZ happens to provide positive assistance to the experienced user group? The answer is no. The initial intent of QUERI-EZ was to focus on the novice/intermediate user group and it does. It hides the extended capabilities from the intended user, and as a result the intended user is not overwhelmed with functionality that he/she does not need. Perhaps, the better solution is to restate the problem statement to say that:

***Novice/Intermediate users need the ability to perform Ad-Hoc queries on database data which typically amount to the equivalent of simple SQL queries. However, a user at this level***

*simply wants to access the data, not learn a query language. Therefore, there needs to be an database front-end application that can shield the user from the low level command-language.*

In essence, the simple SQL query is complex to the novice/intermediate user.

## 9.0 BIBLIOGRAPHY

- 1) Andyne Computing Limited, 1992. GQL/User V3.0 for Windows Demo Guide: Canada.
- 2) Bolt, Richard A., 1984. The Human Interface - When People and Computers Meet: Lifetime Learning Publications, California.
- 3) Booth, Paul A., 1989. An Introduction to Human-Computer Interaction: Lawrence Erlbaum associates Ltd.
- 4) Date, C.J., 1991. An Introduction to Database Systems, Volume 1, Fifth Edition: Addison-Wessley Publishing Company, Inc.
- 5) Date, C.J., 1985. An Introduction to Database Systems, Volume 2: Addison-Wessley Publishing Company, Inc.
- 6) General Electric Company, 1986. Software Engineering Handbook, McGraw-Hill Book Company.
- 7) Hartson, Rex H., Siochi, Antonio C., and Hix, Deborah, 1990. The UAN: A User-Oriented Representation for Direct Manipulation Interface Designs. ACM Transactions on Information Systems, Volume 8, No.3.
- 8) Monk, Andrew, 1984. Fundamentals of Human-Computer Interaction: Academic Press, Inc. (London) Ltd.
- 9) Norman, Donald A., and Draper, Stephen W., 1986. User Centered System Design - New Perspectives on Human-Computer Interaction: Lawrence Erlbaum Associates Ltd, New Jersey.
- 10) Shneiderman, Ben. 1992. Designing the User Interface: Strategies for Effective Human-Computer Interaction, Second Edition: Addison-Wessley Publishing Company, Inc.
- 11) Sybase, Inc., 1992. SYBASE SQL Toolset Release 5.0: Data Workbench User's Guide: Sybase Technical Publications Department.
- 12) SQL Standard ... (TBD)...
- 13) Understanding SQL/2 ....(TBD)...