CONF-9009176--2

CONF-9009176--2

DE90 012735

PROVIDING DYNAMIC SORTING AND SEARCHING CAPABILITIES USING SQL*FORMS

Jane P. Loftis

June 12, 1990

Prepared by the
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6285

Operated by

MARTIN MARIETTA ENERGY SYSTEMS, INC.
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-84OR21400

Title → Providing Dynamic Sorting and Searching Capabilities Using SQL*Forms

Jane P. Loftis
Oak Ridge National Laboratory

Head 1 → Abstract

One goal of developers of software systems is to build an interface that provides the capability of retrieving and displaying data in a variety of ways, depending on the current needs of the user. This paper describes two techniques being used for a prototype system under development at Oak Ridge National Laboratory (ORNL) that provide easy interactive capability between the user and the database using ORACLE's SQL*Forms.

The first technique demonstrates how to build an ORACLE application that allows users to decide whether they would like a subset of data retrieved to a multirecord display form and, if so, to define the boundaries for the subset. The second technique allows users to choose how these displayed data will be sorted. Neither of these techniques requires the user to understand SQL or the underlying database structure of table and column names.

Head 1 → Background

The Worldwide Household Goods Information System for Transportation Modernization (WHIST-MOD) system being designed and prototyped by ORNL is a decision support system for the various organizations of the Military Traffic Management Command (MTMC) that establish and implement the Personal Property Movement and Storage Program. The decision support system will benefit the staff of the Personal Property Program in their tasks of program evaluation and policy setting. This system is designed to access a centralized database through a powerful set of information management tools.

The prototype system offers users, even those with minimal computer experience, easy access to a large selection of shipment data and the ability to easily formulate complex queries. In addition, users may perform special studies and one-time-only ad hoc queries. WHIST-MOD will be a dynamic system that evolves to meet the changing needs of MTMC's analysts.

User requirements for the WHIST-MOD system were identified during the analysis phase of the project. During this phase ORNL identified three modules that needed to be prototyped. These modules include applications to provide a description of the database (system dictionary applications), applications to manage data (data acquisition and administration applications), and applications to retrieve and display data from the database (user applications).

There are two user interfaces in the user application module: a front-end interface and a postprocessing back-end interface. The front-end interface allows the user to choose, retrieve, and store a subset of the larger database, while the back-end interface allows the user to perform statistical analyses and generate graphical and tabular reports using the data subset. The front-end is being prototyped using the ORACLE Relational Database Management System (RDBMS) and SQL*Forms; the back-end is being prototyped using SAS, a statistical analysis package.

This paper discusses two methods ORNL used for the front-end interface that allow a user, untrained in SQL, to specify the objects of default WHERE and ORDER BY clauses in a SQL*Forms base table block. It also gives step-by-step instructions for implementing these capabilities.

Head 1 → General Requirements

Unlike many large database systems that are designed for data entry and update, WHIST-MOD is designed primarily to facilitate data retrieval to support decision-making. Users need to be able to retrieve subsets of shipment data for analysis of the various components of the Personal Property Program. Because users may not be familiar with computers or RDBMSs, the interface was designed to allow users to specify shipment data without understanding SQL or the underlying database structure of table and column names. The interface was also designed to be flexible and generic so that it could support the needs of all Personal Property Program staff, who need to produce different reports depending on their work division (i.e., Quality Assurance, Operations, Rates, or Management Support).

Head 1 → Basic Design of the Interface

The front-end user interface is designed to allow users to choose the subset of data they want to appear on a report. The users move through the interface screens selecting parameters for data they want to include on a report in the following order.

List →
1. They select an application category on the application category screen (e.g., Rates).
2. They select an application (report) on a user application screen (e.g., "Average Net Weight Shipped").
3. They select parameter categories (i.e., carriers, codes of service, geographic areas, and time periods) on the parameter selection menu screen.
4. They select specific parameters on any or all of the lower-level parameter selection screens (i.e., Code of Service Selection Screen, Time Interval Selection Screen, Carrier Selection Screen, and Area Selection Screen).

The lower-level parameter selection screens retrieve and display parameters according to the parameter category the user has chosen. For example, if the user has chosen "code of service" on the parameter category screen, the lower-level parameter screen performs a query that retrieves and displays all appropriate codes of service.

Users identify the subset of data they want to appear on a report by selecting any of the following:

List →
1. specific carriers (e.g., ABC Moving Company);
2. specific origins and destinations (e.g., Virginia to California);
3. specific codes of service (e.g., code 1A); and
4. specific time periods (e.g., May 1, 1990, to May 31, 1990).

As users make selections, their choices are stored in a database table called the Conditions Table. An operating system script that runs when the user asks to perform analysis builds a query that retrieves data from the shipment tables based on the user choices stored in the Conditions Table. For example, using the parameter choices given as illustrations in this section, the script would build a query that retrieves data for all shipments moved by ABC Moving Company from Virginia to California under code 1A during May 1990. The data retrieved from this query are then written to a dataset file that is used by SAS to generate reports and charts.

Head 1→ Enhancements to the Interface

The basic design of the interface worked well for users and provided the flexibility needed to generate a variety of reports. However, because of the large volume of reference data, it was sometimes difficult for users to locate specific parameters on the lower-level parameter screens. To make the interface easier to use, ORNL provided enhancements to address the following problems:

List → 1. the display of many parameter choices and
2. the ordering of the displayed parameters.

Because the basic design of the interface often retrieved and displayed more parameters than the users wanted, they needed the ability to have more control over which data were displayed. For example, up to 700 carrier names and codes could be displayed on the lower-level Carrier Selection Screen. Because only 10 records are displayed on one page, users had to scroll through many pages to locate a specific carrier. This scrolling proved to be a tedious process for users who needed a report for a few specific carriers.

In addition to limiting the number of parameters displayed, users needed to be able to control the ordering of the displayed records. In the basic interface design, the Carrier, Code of Service, and Area lower-level parameter screens displayed two columns: a code column and a description column (see Fig. 1).

→ INSERT FIGURE 1 HERE.

The records displayed on these lower-level screens were sorted on the code column. Because some users are more familiar with carrier names than with Standard Carrier Alpha Codes (SCACs), users wanted the option of sorting the displayed records on the description column so they could search for carrier by name.

Head 1 → Discussion of Enhancements

Initially an option was added to the lower-level parameter screens that allows users to enter specific codes or to retrieve a complete list of them. This enter option, however, solves the problem of quickly specifying parameters only if users know the specific codes they need for a report. Often users know only the first letters of a code or description. Therefore, a substring search enhancement was added that allows users to search the database for records that begin with certain letters.

A further enhancement was added to the lower-level parameter screens that allows users to choose whether they want displayed parameters to be sorted by the code field or by the description field. This sort enhancement allows users to see the retrieved data ordered in a familiar way.

Figure 2 shows the new design of the Carrier Selection Screen. This design incorporates the sorting and substring search enhancements.

→ INSERT FIGURE 2 HERE.

Head 1 → Technical Description of Enhancements

This section describes the technical aspects of the basic design and gives step-by-step instructions for implementing the sort and substring search enhancements. The following discussion begins with a technical description of the basic design, describing each enhancement one-by-one until the complete design with enhancements is presented. Throughout this section the Carrier Selection Screen is used to provide examples. To facilitate the discussions of the enhancements, only the components of the basic design that are relative to the enhancements are discussed. Fields and user-defined triggers are referred to by the actual names they have in the code for the Carrier Selection Screen.

Head 2 → Background

This section describes the technical design of the Carrier Selection Screen. Initially there were three components in the design of this screen:

List → a header control block,
a second control block,
a displayed base table/view block, and
a hidden base table block.

The header block is a display-only control block that is used primarily to orient the user. It consists of three fields:

List → the user name,
the name of the report, and
the service category.

The "user_name" field contains the ORACLE user name, and the "app_name" field contains the name of the report selected by the user in a menu screen. The "service_category" field contains control information that is not relevant to this discussion.

Block zero is a control block that is used to fire different triggers, depending on whether the user wants to display or enter carrier parameters. The block contains only one field, "list_or_enter", which is an enterable, non-base-table field. Users may enter "L" to execute a query on the Carrier Approval View, block one, and to display all appropriate carrier codes and names. They may enter "E" to move directly to block one and enter desired carrier codes.

The displayed base table/view block, block one, has the Carrier Approval View as its base table. This block serves two purposes. It displays multiple (up to ten) carrier names and SCACs, and it allows users to select particular carriers for inclusion on a report. Figure 3 illustrates the process flow of the basic design of this form.

→ INSERT FIGURE 3 HERE.

Block one contains three fields:

List → "star",
"carrier_code", and
"carrier_name".

The "star" field is an enterable, non-base-table field that indicates user selections. The form executes either an "lentqry" or an "eentqry" trigger, depending on whether the user is displaying or entering carriers. The appropriate "entqry" trigger inserts an asterisk in the "star" field to indicate that the user has selected a particular carrier or deletes the asterisk to indicate that the user has canceled a previous selection. The "carrier_code" and "carrier_name" fields are base table fields that display the carrier data retrieved from the query of Carrier Approval View.

The hidden base table block, block two, has the Conditions Table as its base table. Block two is used to insert carrier parameters selected by the user into this table. The details about the functionality of block two are not relevant to this discussion of enhancements to the basic design.

Head 2 → The Sort Enhancement

The sort enhancement was added to the basic design to allow users to choose the column for sorting displayed data parameters. This section provides the steps needed to add this enhancement. The sort enhancement uses the control block that contains the list or enter option, but its functionality is used only if the user chooses the list option in the "list_or_enter" field.

Steps → Step One

Create a one-character field, "sort", in block zero that is positioned below the "list_or_enter" field.

Step Two

Add control to the "list_or_enter" field so that if the user chose "L" in the "list_or_enter" field, the cursor moves to the "sort" field. The user enters "C" to sort by code or "D" to sort by description.

Step Three

Create a nondisplayed field, "sortby", that will contain the string "CARRIER_CODE" if the user has entered "C" or "CARRIER_NAME" if the user has entered "D".

Step Four

Create a "check_sort" trigger that branches to call a "put_scac" trigger if the user has entered "C" or a "put_carrier_name" trigger if the user has entered "D".

Step Five

Create a "put_scac" trigger that puts the string "CARRIER_CODE" into the "sortby" field.

Step Six

Create a "put_carrier_name" trigger that puts the string "CARRIER_NAME" into the "sortby" field.

The remaining steps use an undocumented feature in ORACLE that allows users to choose the column on which the displayed parameters are sorted. This functionality is easy to implement in SQL*Plus using an ampersand, as the following example illustrates.

```
SELECT columns FROM table ORDER BY &sortby
```

However, SQL*Forms does not support the use of an ampersand in an ORDER BY clause. To provide this functionality in SQL*Forms, the programmer must create a SELECT/ORDER BY statement in a PRE-QUERY trigger. The syntax for this statement follows.

PRE-QUERY:

Code →
```
SELECT '#LIKE  ''' ||:base_block.last_field ||'%''
ORDER BY  ' ||:control_block.order_by_field
INTO :base_block.last_field FROM dual
```

When this PRE-QUERY trigger is fired, it places the LIKE/ORDER BY string into the last field of the base table block and appends the LIKE/ORDER BY string to the base table query. This field in the base table block must be long enough to contain the string. Depending on the names of the blocks and fields used, the required length of the last field will vary. Steps Seven and Eight implement this feature using the "sortby" field created in the previous steps. In the example, "carrier_name" is the last field in the base table block on the Carrier Selection Screen.

Step Seven
Create a PRE-QUERY trigger in block one.

Code →
```
PRE-QUERY:
SELECT '#LIKE  '" ||:one.carrier_name ||'%" ORDER BY  ' ||:zero.sortby
INTO one.carrier_name FROM dual
```

Step Eight
Increase the size of the "carrier_name" field, the last field in block one, to a length of 48 characters.

These new fields and triggers enhance the design for users who want to query the database for a list of appropriate carrier codes and names. They may now choose how the displayed data will be ordered by entering a "C" or a "D" in the "sort" field.

Head 2 → The Substring Search Enhancement

The substring search enhancement was added to the basic design to allow users to enter up to two characters to search the database for records that begin with certain letters. (This feature could allow users to search for any number of beginning, ending, or middle characters; however, WHIST-MOD users needed only the initial two-character search capability.) This section provides the steps needed to add this enhancement. The substring search enhancement uses the same control block as the sort enhancement. Its functionality is used only if the user chooses the list option on the "list_or_enter" field.

Steps → Step One
Create a two-character enterable field, "string", in block zero positioned below the "sort" field.

Step Two
Create a field-level trigger on the NEXT FIELD key that branches based on the value in the string field. (This branching is optional, depending on the purpose of the form.) If the value is null, execute a "find_others" trigger. If the value is not null, execute a "from_list" trigger.

Step Three
Create a "find_others" trigger that checks the Conditions Table to see if the user has made previous selections. The details of checking for previous selections are not relevant to this discussion. This trigger also executes a query on block one and positions the cursor in the "star" field.

Step Four
Create a "from_list" trigger that deletes previous selections from the Conditions Table that do not match the string entered by the user. The details of this deletion process are not relevant to this discussion. This trigger also executes a query on block one and positions the cursor in the "star" field.

Step Five
Modify the default WHERE/ORDER BY clause in block one, the base table block, to include the following code.

Code →

```
WHERE
(:zero.sortby = 'CARRIER_CODE' and
carrier_code like :zero.string ||'%')
or
(:zero.sortby = 'CARRIER_NAME' and
carrier_name like :zero.string ||'%')
or
(:zero.string is null and
zero.sortby like '%')
```

The fields and triggers added for the substring search enhancement are used in the default WHERE clause to retrieve data from the Carrier Approval View when a query is executed on this block. The substring search enhancement works in conjunction with the sort enhancement. If the user enters "C" in the "sort" field, the WHERE clause restricts the query to those carrier CODES that begin with the value in the "string" field. If the user enters "D" in the sort field, the WHERE clause restricts the query to those carrier NAMES that begin with the value in the "string" field. If the user chooses not to enter a substring for searching, the query retrieves all carrier names and codes.

Head 1 →

Summary

This paper describes techniques for adding two enhancements to the front-end interface already prototyped at ORNL. These enhancements greatly improved the ease of use and the flexibility of the WHIST-MOD interface. They could readily be adapted, singly or in combination, to other interfaces on projects using SQL-based RDBMSs.

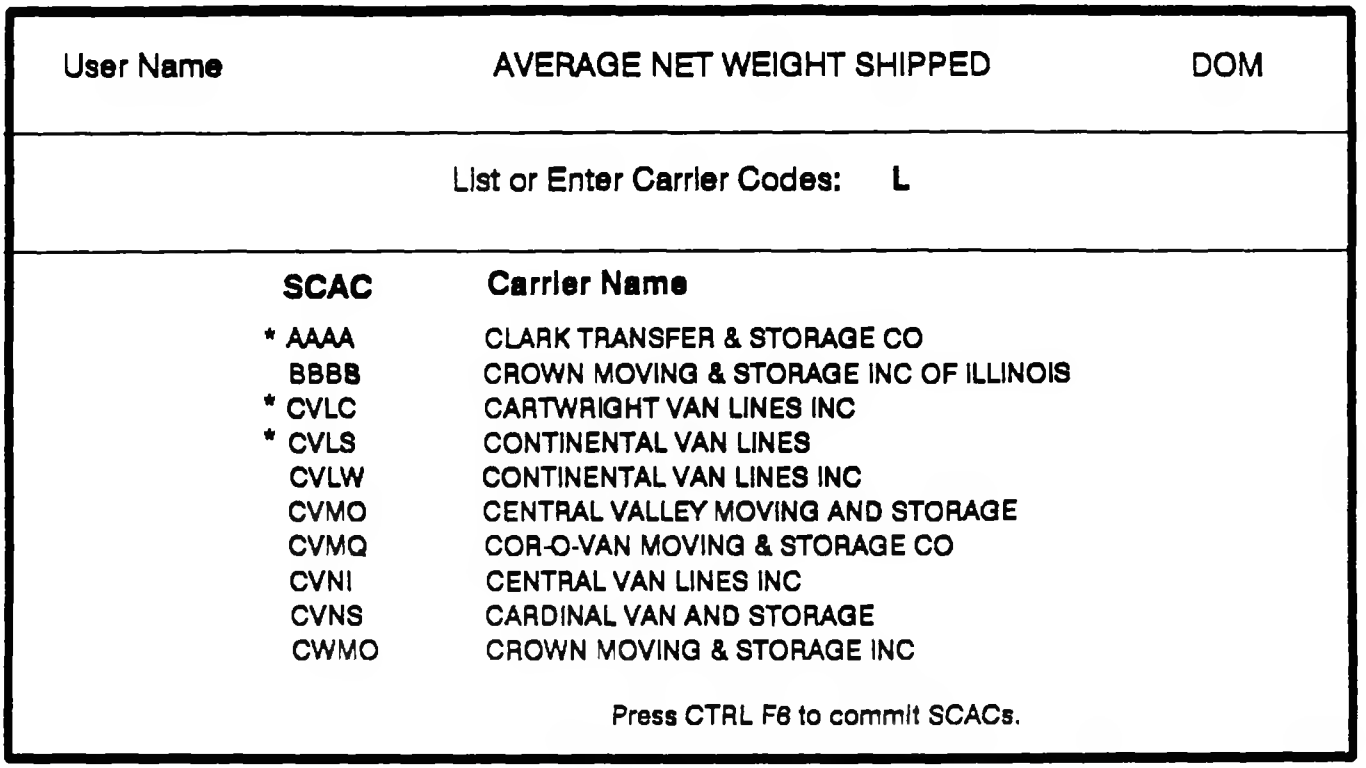


Fig. 1. The basic design of the Carrier Selection Screen. The code column contains the Standard Alpha Carrier Code (SCAC), and the description column contains the name of the carrier.

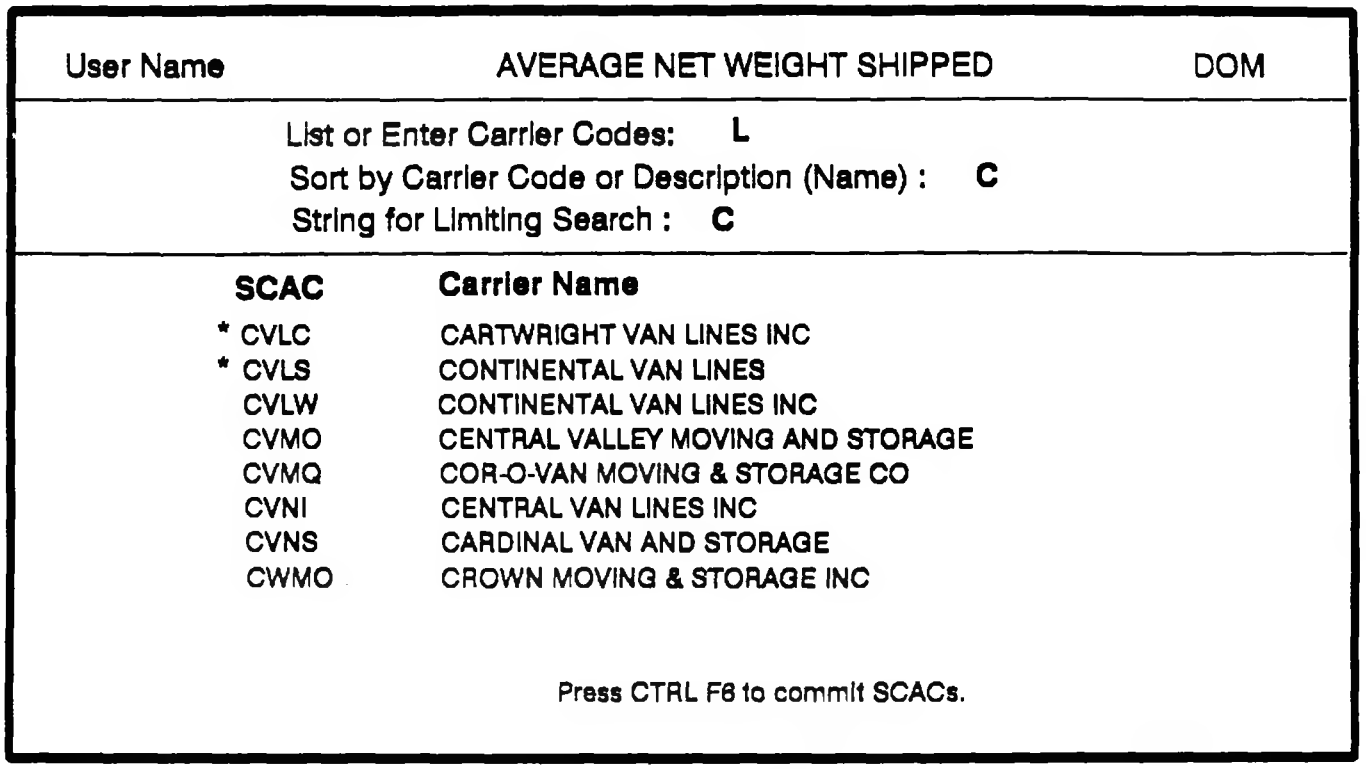


Fig. 2. The design of the Carrier Selection Screen with the substring search and the sort enhancements. This screen displays carrier parameters, ordered by SCACs, for SCACs that begin with "C."

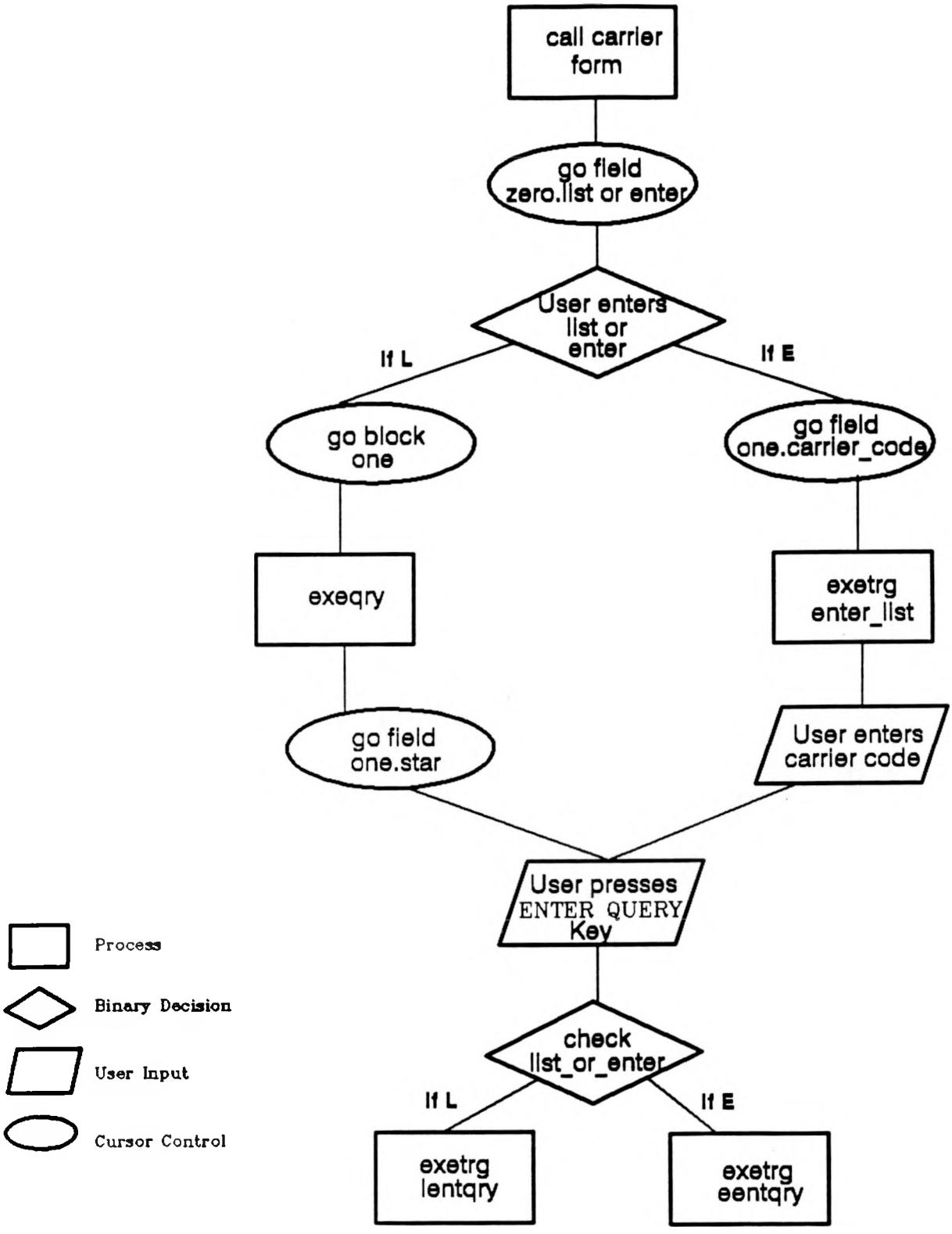


Fig. 3. Process flow diagram of the basic design.

PROVIDING DYNAMIC SORTING AND SEARCHING CAPABILITIES USING SQL*FORMS

Jane P. Loftis

Center for Transportation Analysis
Energy Division

September 23 - 28, 1990

to be presented at the
ORACLE Users Group 1990 International
User Week, Anaheim, California

Prepared by the
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831
operated by
MARTIN MARIETTA ENERGY SYSTEMS, INC.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

PROVIDING DYNAMIC SORTING AND SEARCHING CAPABILITIES USING SQL*FORMS

J. P. Loftis
Energy Division
Oak Ridge National Laboratory*

*Oak Ridge National Laboratory is operated by Martin Marietta Energy Systems, Inc., under Contract No. DE-AC05-84OR21400 with the U.S. Department of Energy

WHIST-MOD SYSTEM

- DoD project
- Decision Support System

WHIST-MOD MISSION

design a decision support system given the following definition:

"... a decision support system should provide the staff at MTPP with access to the information required to help them make intelligent, informed decisions regarding operation and management of their organization. More specifically, a decision support system should ensure that the data provided is relevent and of the highest level of quality, be readily accessible to users of various skill levels, and have the flexibility to adapt to changing demands."
- Final Draft, WHIST-MOD Concept Paper, April 12, 1989, pp. 35.

COMPONENTS OF WHIST-MOD

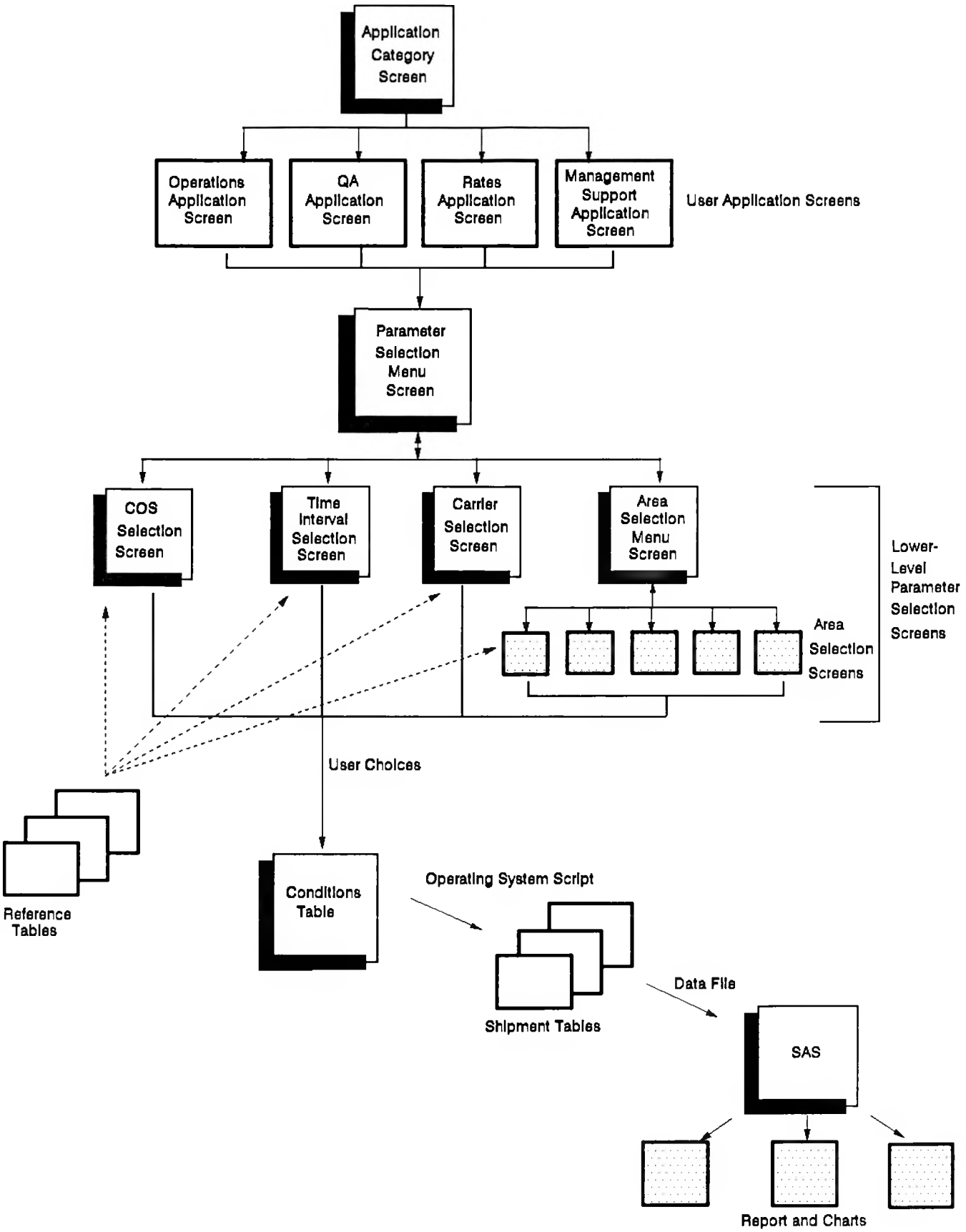
- System dictionary
- Data acquisition and administration
- User interface

USER INTERFACE REQUIREMENTS

- Flexible and generic
- Modular
- Easy to use

EASE OF USE OF THE INTERFACE

- Single keystrokes
- Extensive help messages
- Knowledge of database structure not required
- Knowledge of standard query language (SQL) not required


Application Category Screen
Operations Application Screen
QA Application Screen
Rates Application Screen
Management Support Application Screen
User Application Screens
Parameter Selection Menu Screen
COS Selection Screen
Time Interval Selection Screen
Carrier Selection Screen
Area Selection Menu Screen
Lower-Level Parameter Selection Screens
Area Selection Screens
User Choices
Reference Tables
Conditions Table
Operating System Script
Shipment Tables
Data File
SAS
Report and Charts

USER INTERFACE SCREEN FUNCTIONS

- Select an application category
  (e.g., Rates)
- Select a report type
  (e.g., Net Weight Shipped)
- Select parameter categories
- Select specific parameters

PARAMETER CATEGORIES

- Carriers
- Origins and destinations
- Codes of service
- Time periods

User Name	AVERAGE WEIGHT SHIPPED	26-FEB-90

Parameter Selection Menu

Service Category:

Code of Service

_ ALL SELECTED

Pickup Date

_ ALL SELECTED

Carrier

_ ALL SELECTED

Origin & Destination

_ ALL SELECTED

COMPONENTS OF THE INITIAL SCREEN DESIGN

- Header block
- Control Block
- Base table/view block
- Hidden block

User Name | AVERAGE NET WEIGHT SHIPPED | DOM

List or Enter Carrier Codes: L

	SCAC	Carrier Name
*	AAAA	CLARK TRANSFER & STORAGE CO
	BBBB	CROWN MOVING & STORAGE INC OF ILLINOIS
*	CVLC	CARTWRIGHT VAN LINES INC
*	CVLS	CONTINENTAL VAN LINES
	CVLW	CONTINENTAL VAN LINES INC
	CVMO	CENTRAL VALLEY MOVING AND STORAGE
	CVMQ	COR-O-VAN MOVING & STORAGE CO
	CVNI	CENTRAL VAN LINES INC
	CVNS	CARDINAL VAN AND STORAGE
	CWMO	CROWN MOVING & STORAGE INC

Press CTRL F6 to commit SCACs.

COMPONENTS OF THE HEADER BLOCK

- User name
- Report name
- Service category

COMPONENTS OF THE CONTROL BLOCK

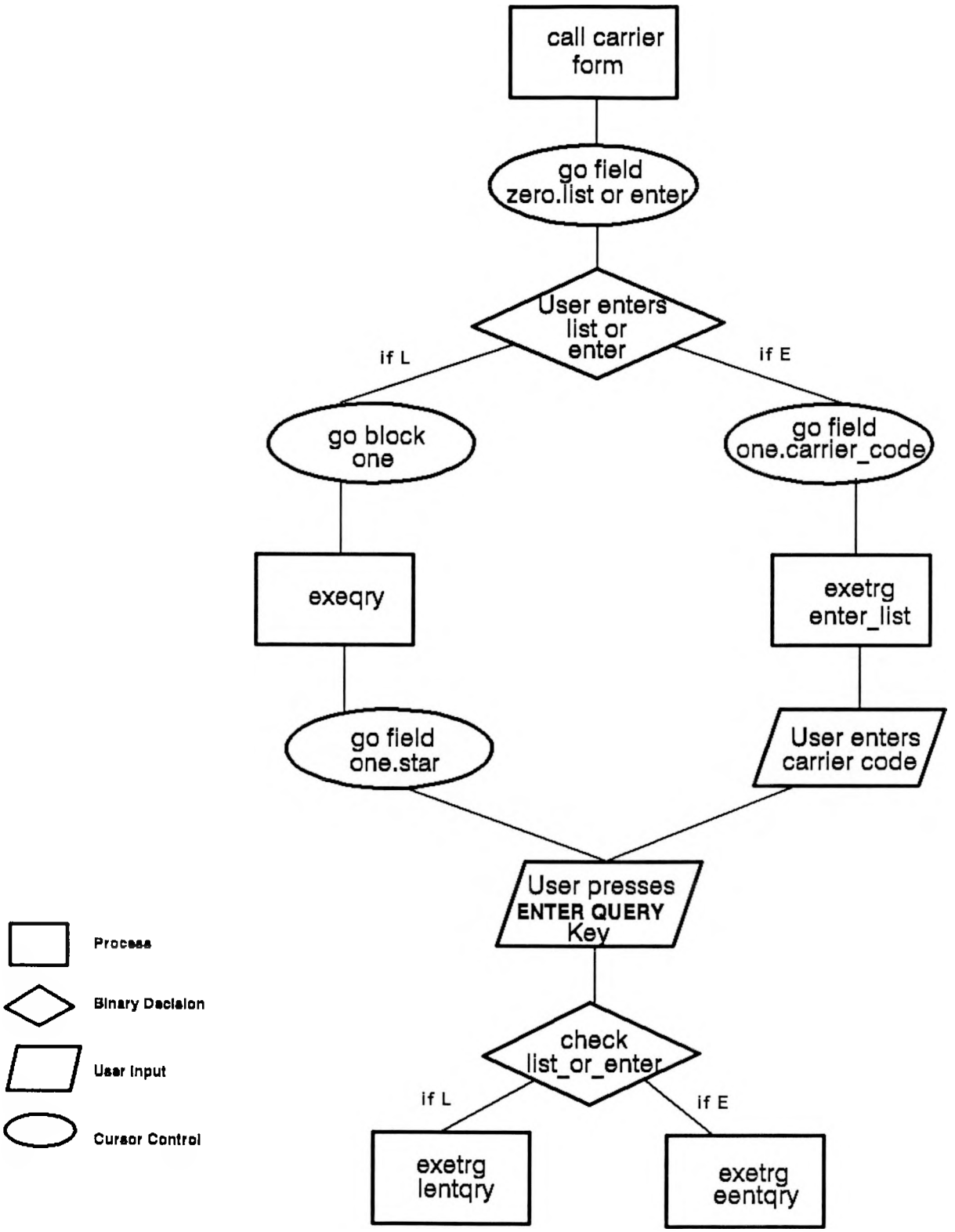
- List or enter

COMPONENTS OF THE BASE TABLE/VIEW BLOCK

- Star
- Code
- Description

COMPONENTS OF THE HIDDEN BLOCK

- Conditions table fields


call carrier form
go field zero.list or enter
User enters list or enter
if L
if E
go block one
go field one.carrier_code
exeqry
exetrg enter_list
go field one.star
User enters carrier code
User presses ENTER QUERY Key
check list_or_enter
if L
if E
exetrg lentqry
exetrg eentqry
Process
Binary Decision
User Input
Cursor Control

NEED FOR DESIGN ENHANCEMENTS

- Difficulty of locating records
- Need for interactive sorting

ENHANCEMENTS TO THE BASIC DESIGN

- Substring search capability
- Interactive "Order By" capability

User Name AVERAGE NET WEIGHT SHIPPED DOM

List or Enter Carrier Codes: L
Sort by Carrier Code or Description (Name) : C
String for Limiting Search : C

	SCAC	Carrier Name
*	CVLC	CARTWRIGHT VAN LINES INC
*	CVLS	CONTINENTAL VAN LINES
	CVLW	CONTINENTAL VAN LINES INC
	CVMO	CENTRAL VALLEY MOVING AND STORAGE
	CVMQ	COR-O-VAN MOVING & STORAGE CO
	CVNI	CENTRAL VAN LINES INC
	CVNS	CARDINAL VAN AND STORAGE
	CWMO	CROWN MOVING & STORAGE INC

Press CTRL F6 to commit SCACs.

THE SORT ENHANCEMENT

STEP ONE

Create a one-character field, "sort", in block zero that is positioned below the "list_or_enter" field.

STEP TWO

Add control to the "list_or_enter" field so that if the user chose "L" in the "list_or_enter" field, the cursor moves to the "sort" field. The user enters "C" to sort by code or "D" to sort by description.

THE SORT ENHANCEMENT (Cont'd.)

STEP THREE

Create a nondisplayed field, "sortby", that will contain the string "CARRIER_CODE" if the user has entered "C" or "CARRIER_NAME" if the user has entered "D".

THE SORT ENHANCEMENT (Cont'd.)

STEP FOUR

Create a "check_sort" trigger that branches to call a "put_scac" trigger if the user has entered "C" or a "put_carrier_name" trigger if the user has entered "D".

THE SORT ENHANCEMENT (Cont'd.)

STEP FIVE

Create a "put_scac" trigger that puts the string "CARRIER_CODE" into the "sortby" field.

THE SORT ENHANCEMENT (Cont'd.)

STEP SIX

Create a "put_carrier_name" trigger that puts the string "CARRIER_NAME" into the "sortby" field.

SQL*PLUS ORDER BY FUNCTIONALITY

SELECT columns FROM table ORDER BY &sortby

SQL*FORMS SOLUTION

```
PRE-QUERY:
SELECT '#LIKE  ''' ||:base_block.last_field ||'%''
ORDER BY  ' ||:control_block.order_by_field
INTO :base_block.last_field FROM dual
```

THE SORT ENHANCEMENT (Cont'd.)

STEP SEVEN

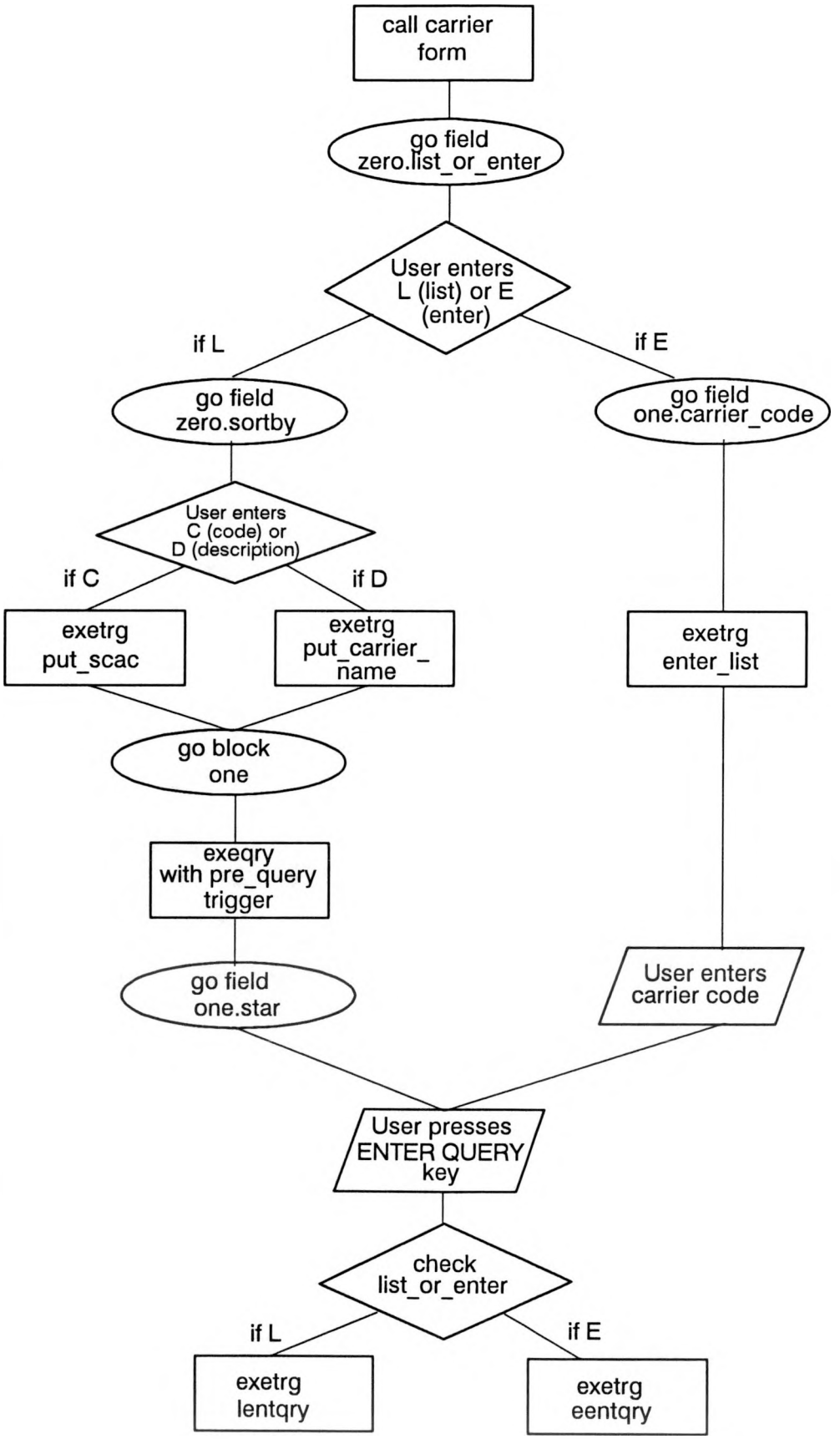
Create a PRE-QUERY trigger in block one.

```
PRE-QUERY:
SELECT '#LIKE ''' ||:one.carrier_name ||'%'' ORDER
BY  ' ||:zero.sortby
INTO one.carrier_name FROM dual
```

THE SORT ENHANCEMENT (Cont'd.)

STEP EIGHT

Increase the size of the "carrier_name" field, the last field in block one, to a length of 48 characters.


call carrier form
go field zero.list_or_enter
User enters L (list) or E (enter)
if L
if E
go field zero.sortby
go field one.carrier_code
User enters C (code) or D (description)
if C
if D
exetrg put_scac
exetrg put_carrier_ name
exetrg enter_list
go block one
exeqry with pre_query trigger
go field one.star
User enters carrier code
User presses ENTER QUERY key
check list_or_enter
if L
if E
exetrg lentqry
exetrg eentqry

THE SUBSTRING SEARCH ENHANCEMENT

STEP ONE

Create a two-character enterable field, "string", in block zero positioned below the "sort" field.

THE SUBSTRING SEARCH ENHANCEMENT (Cont'd.)

STEP TWO

Create a field-level trigger on the NEXT FIELD key that branches based on the value in the string field. (This branching is optional, depending on the purpose of the form.) If the value is null, execute a "find_others" trigger. If the value is not null, execute a "from_list" trigger.

THE SUBSTRING SEARCH ENHANCEMENT (Cont'd.)

STEP THREE

Create a "find_others" trigger that checks the Conditions Table to see if the user has made previous selections. The details of checking for previous selections are not relevant to this discussion. This trigger also executes a query on block one and positions the cursor in the "star" field.

THE SUBSTRING SEARCH ENHANCEMENT (Cont'd.)

STEP FOUR

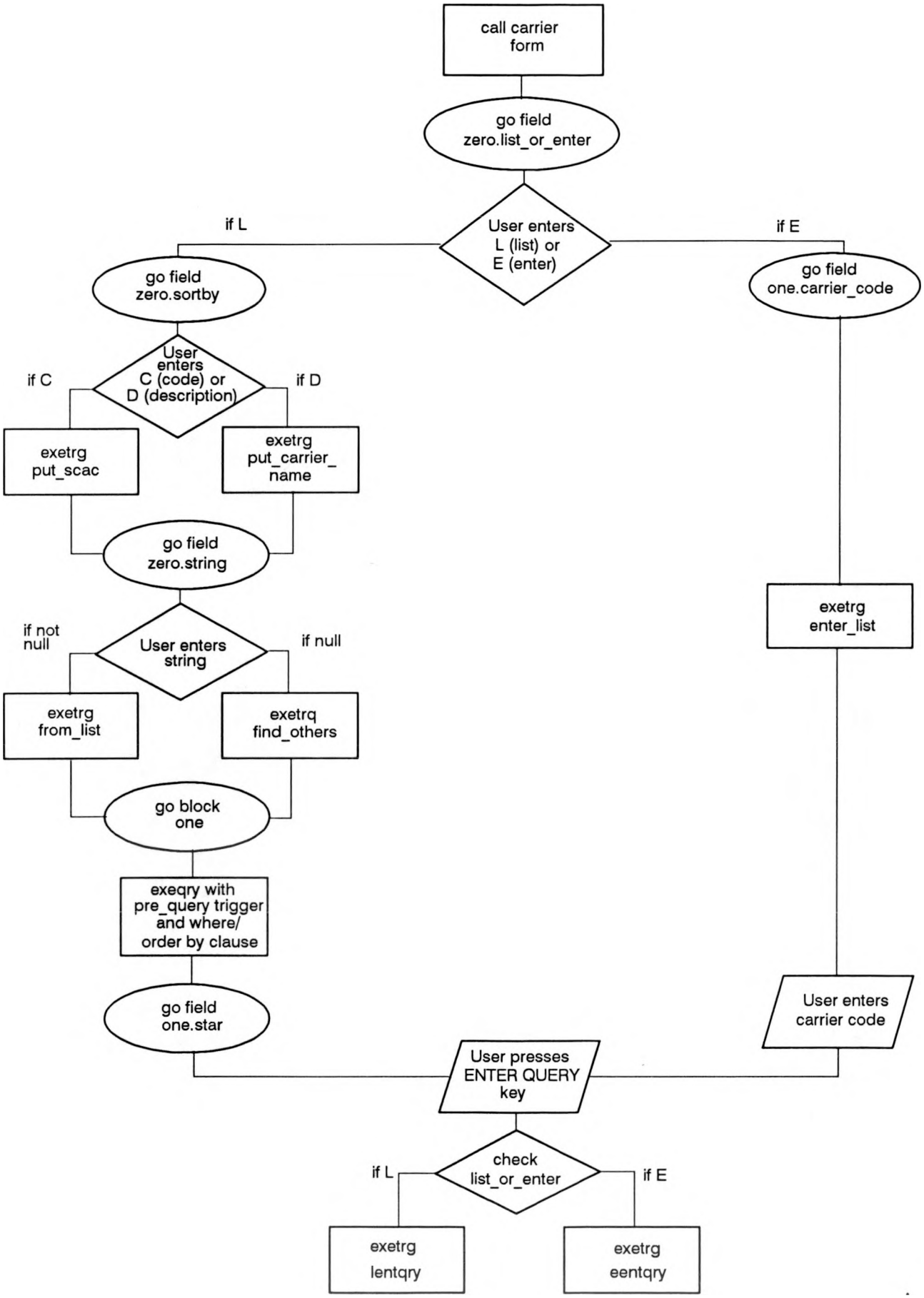
Create a "from_list" trigger that deletes previous selections from the Conditions Table that do not match the string entered by the user. The details of this deletion process are not relevant to this discussion. This trigger also executes a query on block one and positions the cursor in the "star" field.

THE SUBSTRING SEARCH ENHANCEMENT (Cont'd.)

STEP FIVE

Modify the default WHERE/ORDER BY clause in block one, the base table block, to include the following code.

```
WHERE
(:zero.sortby = 'CARRIER_CODE' and
carrier_code like :zero.string ||'%')
or
(:zero.sortby = 'CARRIER_NAME' and
carrier_name like :zero.string ||'%')
or
(:zero.string is null and
zero.sortby like '%')
```


call carrier form
go field zero.list_or_enter
User enters L (list) or E (enter)
if L
if E
go field zero.sortby
go field one.carrier_code
User enters C (code) or D (description)
if C
if D
exetrg put_scac
exetrg put_carrier_ name
go field zero.string
exetrg enter_list
User enters string
if not null
if null
exetrg from_list
exetrq find_others
go block one
exeqry with pre_query trigger and where/ order by clause
go field one.star
User enters carrier code
User presses ENTER QUERY key
check list_or_enter
if L
if E
exetrg lentqry
exetrg eentqry

CONCLUSIONS

- Improved flexibility
- Improved ease of use