# The Design and Implementation of a QBE System

Eva Liana Tandjung

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

for the degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

March 1983

## ABSTRACT

THE DESIGN AND IMPLEMENTATION OF A QBE SYSTEM

Eva Liana Tandjung

Query By Example (QBE) is one of the most popular query languages used for database applications. This interactive language requires users to operate on a two dimensional screen for specifying queries. In this major report a software system called QBE system is described which has two major components namely QBE Front End Process or QFP and QBE Host Process or QHP. The QFP is a software system written for the microprocessor MC6809 that will eventually become an integral part of the H19 CRT terminal used for user communication. The QHP software system is written to run on a mainframe like computer system as an interface between the QFP and the database management system. The QHP reported here is written in PASCAL for the CYBER 172 system and operates in the context of a relational database management system called RISS.

To My Dearest Parents

## Acknowledgements

I would like to take this opportunity to express my sincere gratitutes to my supervisor Dr. T. Radhakrishnan who outlined the philosophy and direction of this major report, read the numerous drafts of this report, and made suggestions for its improvement.

To Aciek, go my special thanks for his unfailing support during my study in Concordia. And to Rao, thanks for providing the QFP.

I would also like to thank professor Radhakrishnan and the Department of Computer Science for their financial assistance.

Contents

# LIST OF FIGURES

# C H A P T E R 1

## Introduction

As computer systems become more popular, there is an increasing need to have a good and cost effective communication between a database and its users. Consequently more and more attention is being focused on the prime element of the computer applications, namely the human element. Efficiency in the use of system resources can become ineffective if a system is not designed to match the needs and capabilities of its end users. This fact has led to the exploration of new research areas involving the human-oriented aspects of computer systems [Reisne79].

## 1.1 Objectives

The objective of this major report is twofold:

1. To develope an information retrieval system which interfaces through the use of Query By Example language (QBE).
2. To employ a front end microprocessor to control and carry out the necessary terminal oriented operations for QBE interface.

A software system called QBE system is developed for this purpose and it is based on H19 Heathkit terminal. This terminal supports limited graphics capabilities such as displaying horizontal and vertical lines on the screen.

## 1.2 Query By Example

QBE is an easy to use high level language with a convenient and unified interface to query, update, define, control and secure a database. In this language, when a user wants to perform an operation against the database, he fills in an example of that operation in skeleton tables displayed on the screen which are associated with some relations of the database. The software system presented in this report analyses the user supplied information through such graphical forms, reformulates the query in terms of relational operators operating on the specified relations. The system retrieves information using the RISS [Mcleod75] DBMS and displays it in the form of appropriate result relations. The philosophy behind QBE is to require users to have very little knowledge in order to get started and to minimize the number of objects and concepts that he has to learn to understand and use the language.

## 1.3. Design Phases

The design of a QBE system consists of three phases. Phase one is concerned with developing a front end process as the user interface named QBE Front End Process (QFP). It is developed on a MC6809 microprocessor and is stored in a ROM and kept as part of the H19 terminal. In phase two, an interface to handle the information retrieval is developed and named as QBE Host Process (QHP). Finally, in phase three the complete system is integrated and tested using an example data base on the RI8S [Mcleod75] data base management system.

## 1.4 Features of the QBE System

The following are the essential features of the QFP:

1. One to four relations can be displayed on the screen at the same time in the multi relation mode.
2. Users can view a selected relation in full on the entire screen and restore back to multi relation display by pressing a single key.
3. Users can readily refer to the status of the relation displayed on the 25th line in reverse video mode.
4. Users can select the mode to display one, two or four relations at a time on the screen using function keys.

5. The menu of the relations in the database can be used to formulate queries.

6. Scrolling can be done to display the next group or previous group of tuples or columns by means of the four cursor function keys. This can be done with any of the relations displayed on the screen.

The following are the features of the QHP:

1. Retrieval based on simple type.

2. Retrieval of qualified type.

3. Retrieval using pre-determined keywords.

4. Retrieval using common example elements (linking) within one relation.

5. Retrieval using common example elements between any two relations.

6. Imposing print operators before the first column to be considered as row-operators.

7. Editing facilities such as: insertion, deletion and update. This feature however, is not supported at this stage.

The QBE system involves two independent software systems: QFP and QHP. Figure 1 shows the complete QBE system.

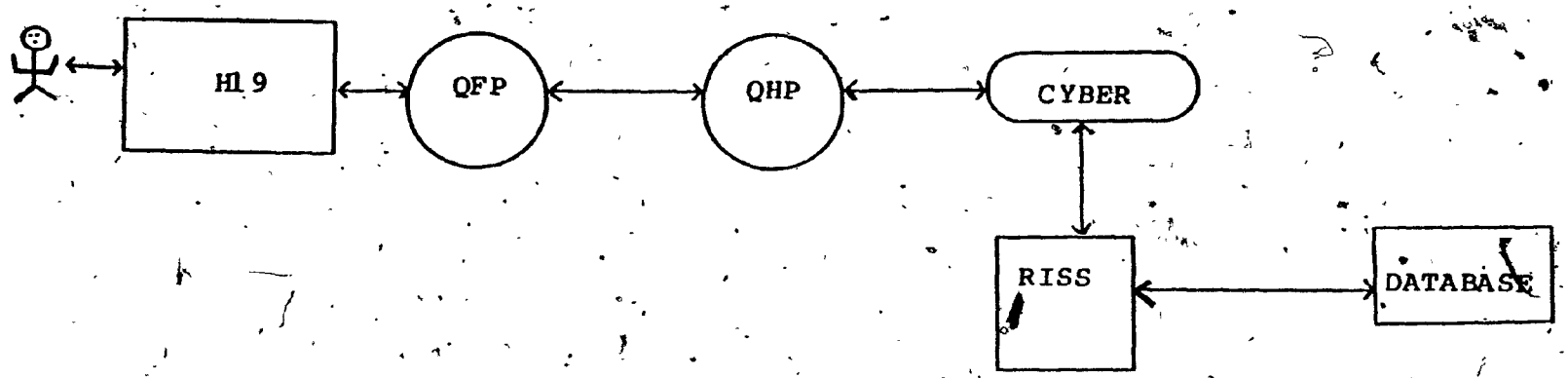


Figure 1. QBE SYSTEM

QFP acts as an end user interface which takes care of the transfer of information entered by a user from the H19 terminal ( micro computer based terminal ) to the host processor (QHP). QHP will then make an access to the database (RISS DBMS), extract the appropriate information and pass the answer back to QFP which will display the answer on the screen. The central computing facility used for this project is the Cyber 172 mainframe.

# CHAPTER 2

## Literature Review

Generally, data sublanguages, support a set of data base operators embedded in a host programming language. On the other hand query languages are stand alone languages through which an end user interacts directly with the database management system. In addition to their query capability, most query languages provide facilities to update, create and delete data. As compared to a typical data sublanguage, a query language is at a higher level, less procedural, and intended for a more casual user. Sometimes, however, the same basic set of operators can be found in both a data sublanguage and a query language. Below we present a brief review of the literature in the area of query languages.

### 2.1 Relational, Hierarchical and Network based Query Languages

Since the introduction of relational data model in 1970, various high level data base languages have been introduced to support this model. For almost all of them, relational calculus and relational algebra lay the ground

work. The relational calculus is further divided into two classes: a tuple relational calculus and a domain relational calculus. Alpha [Codd72] and Quel [Held75] are two typical examples of tuple calculus languages; FQL [Pirott77], ILL [Lacroix77a, Lacroix77b] and QBE [Zloof75] are typical examples of domain calculus languages. It is noted that not much research was conducted on query languages for network or hierarchical oriented databases. The principle reason for the interest in relational model of data is that it allows the user to express the desired results of a query in a high level non-procedural data language without specifying the access path to a stored data.

Either Relational calculus or relational algebra can be used to specify a complex query against the database. However, their underlying mathematics and the procedurality are not easy to understand for a casual user. As a result, much work has been done to develop languages which are as powerful as the relational calculus and the relational algebra but are easy for users of any background.

There are three query languages of this type worth mentioning: Query-By-Example [Zloof75], Square [Boyce75] and Sequel [Chambe74]. Query-By-Example (QBE) is an easy to use high level language with a convenient and unified interface to query, update, define, control and secure a database. In this language when a user performs an operation against the database( such as query or update), he fills in an example

of that operation in skeleton tables displayed on a screen by the system. The Philosophy behind QBE is to requireo a user to know very little to get started and to minimize the number of objects and concepts that he has to learn in order to understand and use the language.

The QBE language has been used for several database applications by non-programmers. A psychological test [Thomas75] shows that it requires less than three hours of instruction for casual users to be able to ask a fairly complex queries. [Dawei81] used the idea of QBE to construct another query language called FOBE ( Form Operation By Example ) to be applied to specify queries on forms. An algorithm for processing queries expressed in QBE is discussed in [Niebuh76].

Square [Boyce75] is a non-procedural language based on relational calculus intended to be used in ad hoc interactive problem solving by non-computer specialists. It also provides facilities for querying, insertion, deletion and update of tabular database. The query capabilities of Square were proven to be relationally complete. Users of Square can describe the data to be accessed using expressions based on "mappings". Queries using Square are claimed to be simpler and more concise than their equivalences in the relational calculus.

In 1974 the query language SEQUEL was introduced [Chambe74]. It is used as a data retrieval and as a data manipulation language. This language is based on the relational data model and intended to be used by non programmers. Sequel was later extended to Sequel2( SQL ) [Chambe76] to facilitate users to express complex queries. SQL uses a block-structured format and English keywords. Each SQL query block selects certain tuples from a relation by means of a AND/OR tree of selection predicates. These predicates may in turn contain nested query-blocks which select values from other relations. The notable feature of the language is the nesting of query blocks to an arbitrary depth. SQL may be thought of as a language consisting of several layers of increasing complexity. The casual user may learn only the simplest query features; more trained users may find use for more powerful features. SQL is the main external interface supported by system R as a data sublanguage embedded in PL/I. Sequel has been applied in a minicomputer environment. An example of such a system is discussed in [Ander78] as Minisequel for DBMS. Tablet discussed in [Charle81] is similar to SQL. SQL is based on Codd's relational calculus where as Tablet is based on Codd's relational algebra.

A study of human factors comparison of procedural and a non-procedural query language was conducted in [Charle81]. This study compared the use of Sequel and a Tablet based

method. It concluded that people more often write difficult queries correctly using a procedural query language like Sequel than they do by using a non-procedural method. [Reisne75] was conducting another study on human factors evaluation of Sequel and Square query languages. The goal of this study was threefold:

1. To determine whether the languages could be used by users without extensive training.
2. To determine whether there was a difference in usability of the two languages.
3. To discover the common errors frequently made by the users of the two languages.

This experiment was done on students of a university. The students were divided into two catagories: non-computer specialists and computer specialists. The results of the experiment showed that non-computer specialists were more comfortable with Sequel rather than Square. Both classes of users were able to manipulate the basic language features and combined them in several different ways. Errors frequently made were on syntactic and synonym type of errors.

Another psychological test was done on the the three popular query languages (QBE, SQUARE and SEQUEL) [Greenb78]. The emphasis of the test was to determine if any of the three languages has significantly better learning and application capabilities over the others. The result of the experiment showed that Zloof's QBE was superior.

[Mcleod76] discussed an approach to translate a QBE query to its Sequel equivalence. This paper suggested the usefulness of a translation algorithm that can facilitate the translation from one query language to another one, so it will provide a mechanism by which multiple query languages may be supported in a single data base system, allowing queries to be stated in a language most appropriate to the user and/or the query itself.

[Shipma81] discussed the functional data model and data language DAPLEX which has the following properties:

1. Formulation of data in terms of entities.
2. A functional representation for actual and virtual data relationships.
3. A rich collection of language constructs for expressing entity selection criteria.
4. A notation of subtype/ supertype relationships among entity types.

SLANG [Nicola81] is a statistical language designed to satisfy the need of UNIDO's division for industrial studies for an efficient, powerful and easy to use package for descriptive time series analysis.

There is a query language which provides the capabilities for querying and operating on databases [Manola82]. The language is named CQLF. CQLF is another

high level query language for accessing and manipulating data in databases described using the 1981 ANSI dpANS version of the CODASYL Data Description language.

APPLE [Robert76] is a language claimed to be the best query language. Users can formulate queries solely in terms of attribute names. The system software will determine the access path needed. This language is different from ALPHA in that it does not require users to have mathematical background and it is claimed to be better than SEQUEL or QBE. In SEQUEL or QBE users have to specify the relation name(s) and the attributes to be used to navigate across the relation boundaries, whereas in using APPLE users can only specify the attribute names. APPLE can handle single access path as well as multiple ones. This language is currently used as the Host language on a minicomputer based general purpose database [Patnai81].

QUEL(QUEry Language) is a language supported in INGRES which became first operational in March 1976 [Stoneb76]. It has points in common with Data Language ALPHA and SQUARE in that it is a complete language and frees users from concerning how the data structures are implemented and which algorithms are operating on stored data. So it facilitates a considerable degree of data independence.

PRTV is an interactive database system intended to be used either as a stand-alone system for simple data bases or

as a data subsystem for an application system [Todd76]. ISBL is used as PRTV's query language. PRTV is not a fullfledged database system, but rather an evolving prototype which is expected to aid in solving some of the problems that have been encountered in using databases.

[Antona78] talked about AQL which will provide an easy to use interface based on default options, synonyms, and definition of attributes, inferences, and the possibility of interactive completion of the query. Interactive method of filling the incomplete information in a query is further discussed in [Baxter78].

A query language that can interface either with a relational or a network databases was discussed in [Taimin80] called XQL( Extended Query Language ). XQL uses the relational calculus and a CODASYL-like data manipulation language as target languages to map the user's query into the data model. XQL has been implemented on a Honeywell 68/80 Multics system.

CASDAL [Su78] is a high level data language designed and implemented for the database machine CASSM. The language is used for the manipulation and maintenance of a database using an unnormalized relational data model. It also has facilities to define, modify and maintain the data model definition.

A query language for PDP-11 system was discussed in [Warner81] called FQUERY. It is a high level language that applies the concepts of the functional data model. The system uses RISS relational DBMS.

TAMALAN [Vandij77] was designed to be a casual user interface with no complexity of mathematical background. This language is also based on relational model. It was designed as a supplement to some existing query languages that were considered to have some difficult features such as:

1. The occurence of complicated sentences with nested operations to formulate a query.
2. The occurrence of nested quantifier.
3. The occurrence of mathematical variables.

In specifying a query, users of TAMALAN should follow some procedural aspects that use commands which are combined with some descriptive elements.

Query languages discussed so far were mostly based on the relational model of data. In the following we discuss network and hierarchical oriented query languages.

[Schlag82] disscussed a query language based on network model called NOAH. This language allows to formulate complex queries which among other things, include conditions for CODASYL-sets and conditions for n : m relationships.

NUL is another network oriented query language [Dehene76] which allows users to navigate from one record set to another one through the sets of the database. NUL allows an arbitrary search paths process and is very flexible in expressing selection conditions. The language looks very powerful compared to NOAH but the set of queries expressed in NUL is not a real superset of those expressible in NOAH. Looking at its structure, this language seems not too easy to be implemented.

QUEST [House179] can be applied to all three database models: relational, hierarchical and network. This query language, however, is meant to be a theoritical study only.

IDMS is a commercial database management system based on a network model and makes use of a CODASYL DBTG type of language. FORAL is the query language used by this system which is a non-procedural data specification and manipulation language based on binary associations[Goldsc78].

## 2.2 Natural Language Like Query Languages

A language for Relational Associative Processor (RAP) designed and implemented in 1976 is discussed in [Kersch76]. This language is called SYNGLISH (A Synthetic English Query Language). It is a very high

level query language based on natural English. Queries specified in Synglish, are allowed to be as complex as possible without any regard to the language structure. Queries written in Synthetic English are easily parsed, using semantic predication analysis and underlying graph, into primitive templates which are in one-to-one correspondence with the high-level machine language for the RAP.

[Codd78] discussed a problem concerning an experimental system intended for query formulation: Rendezvous version 1. It differs from other natural language query systems in the extent to which it talks back to the user about the entered query before any data base retrieval is executed. This paper discussed the types of dialog supported by this system and some other lessons learnt from the experimental system.

TORUS is a natural language understanding system intended for communication by casual users with a database management system. Users' request is found using a semantic network, knowledge about the database [Mylopo76]. Mylopoulus has researched on another language called TAXIS [Mylopu80] which was designed primarily for application systems that are highly interactive and make use of the database substantially. This language permits the specification of semantic integrity constraints, and handles exceptions arising out of it. While the notation retains

some programming constructs such as the concepts of classes and properties, its orientation is toward a natural language.

[Lacroix77a, Lacroix77b] discussed an English oriented retrieval language for relational data bases called ILL. The language was designed to investigate the correspondence between natural language queries and queries in domain-oriented predicate calculus languages. ILL was built on a structure of expressions nested one inside the other.

Another query language originally designed and implemented to provide a powerful and structured interface to a -CODASYL DBMS is called FQL [Bunema79]. FQL is an applicative language and has many of the ideas concerning functional programming systems. It regards a database as a collection of functions over various data types. Users have the freedom to combine the functions. A query in FQL is no more than another function over the database which can be combined with other queries. FQL can serve both as a tool to construct complex queries and as an intermediate language into which one's query language may be readily translated. As an example it can be used as a database interface for natural language systems.

## 2.3 Picture Query Languages

There is an increasing concern in the area of image processing about the query languages. Data bases of this kind are called Pictorial Database System( PDBS ). Non-alphanumeric information, such as digitized images, needs large amount of memory space. Efficient and economical storage, flexible retrieval, and the manipulation of a vast amount of pictorial information have become problems that need to be carefully considered. Digitized images include: geographical data processing, computer-aided design, remote sensing of earth resources, regional economic and health data processing, and cartographic and mapping applications. Conventional query languages for retrieving and manipulating alphanumeric data can still be used to find image locations through specified image registration. Query such as: " Find that portion occupied by Concordia University on a map", however, can not be expressed using any of the existing query languages. Therefore, Picture Query Languages were developed to express such a query. It can have additional capabilities such as: the retrieval of images through specified picture description and updating the picture descriptions. Data management concept such as: data independence, data integration, controlled redundancy, security and privacy are also imposed in a Picture Query Language.

IGL is an interactive end user language for interface with a geographic information system ATLAS [TSURUT81]. The database will contain geographic information concepts concerning semantic structure, topological structure and location structure. It can store various sorts of geographic and statistics data. IGL facilitates information retrieval, map production and map modification. It can express information retrieval conditions which use the relationships in geographic information structure concepts. The database is based on CODASYL type DBMS called ADBS.

[Chang81] talked about existing picture query languages to be used to retrieve information from non-alphanumeric database. Most picture query languages are based on the relational data model. A list of the existing picture query languages follows:

1. QPE, used in Pictorial Data-base system: IMAID.
2. GRAIN, used in Pictorial Data Base system: GRAIN.
3. IQ, used in Pictorial Data base system IMDS.
4. IDMS, used in Pictorial Data Base system: IDMS.
5. ARES, used in Pictorial Data Base system: ARES.
6. GADS, used in Pictorial Data Base system: GADS.
7. IDAMS, used in Pictorial Data Base system: GADS.
8. ADM, used in Pictorial Data Base system : ADM.
9. AQL, used in Pictorial Data Base system: AQL.

Details of these languages and references to the work concerned can be found in [Chang81].

# C H A P T E R 3 (*)

## Implementation Detail of QFP

QFP is the subsystem that handles the terminal control. This section will discuss the implementation detail of this subsystem.

### 3.1 QFP System Level Details

From the system point of view, QFP primarily performs the following functions:

1. Host-QFP Command Interface
2. Host-H19 Interface
3. H19 Terminal Control
4. H19 Command Handler

Figure 2 shows the pictorial representation of the interconnection of the above functions.

1. Host-QFP Command Interface:

It is an interface to transfer information in the form of frames between QHP and QFP.

(*) This subsystem was done by R. M. Kotamarti

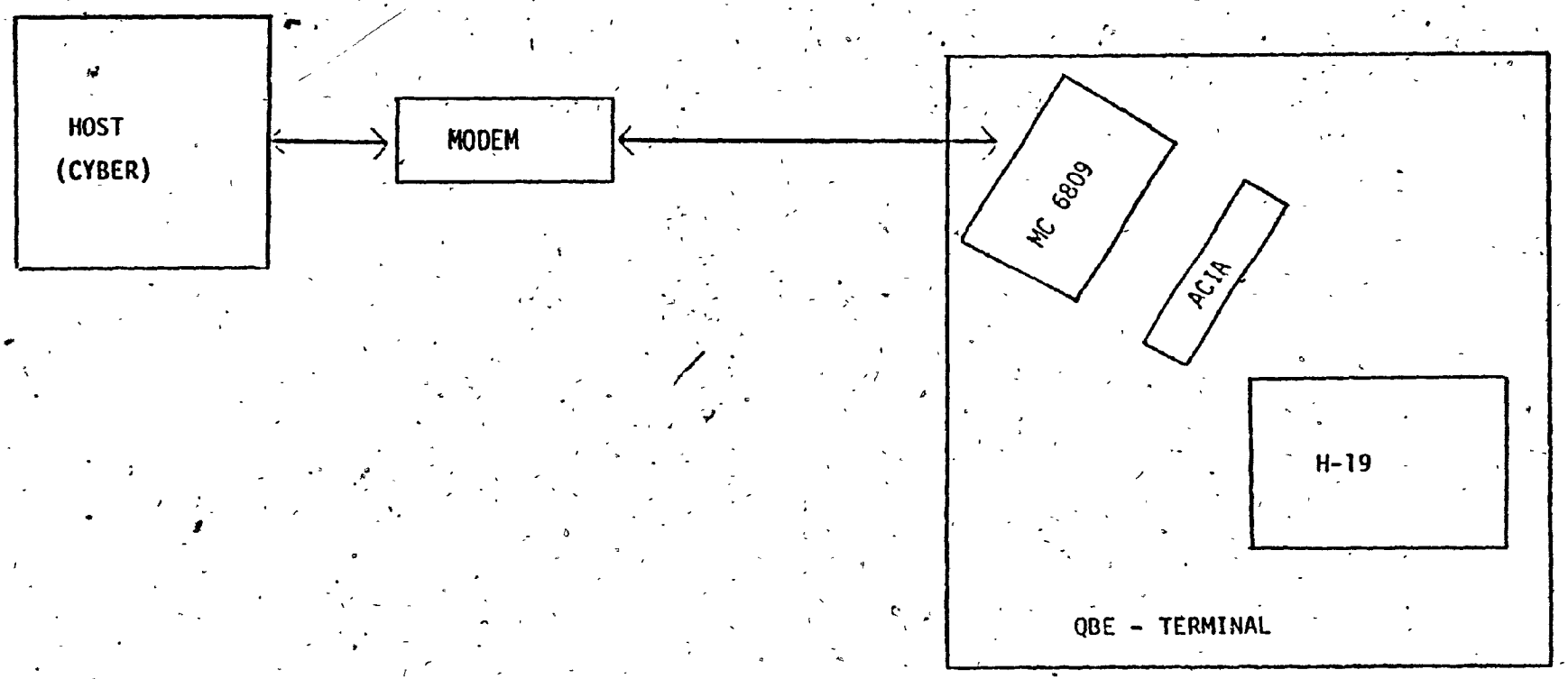


Figure 2. QBE Subsystems

2. Host-H19 Interface:

This is an interface to allow direct communication between Cyber and H19 terminal; thus H19 is used as an ordinary terminal to a mainframe.

3, H19 Terminal Control Interface:

This implements an interface to handle specific I/O with H19. This interface will also handle H19 mode changing capability which will be described in section 3.3.

4. H19 Command Handler Interface:

It is an interface to read and interpret user commands entered by pressing programmed keys as well as displaying information in various ways.

## 3.2 Data structures of QFP

The pictorial representation of QFP data structures is given in figure 3.

1. Main Information Block

As already stated, QFP can accomodate upto four relations at a time on the screen.

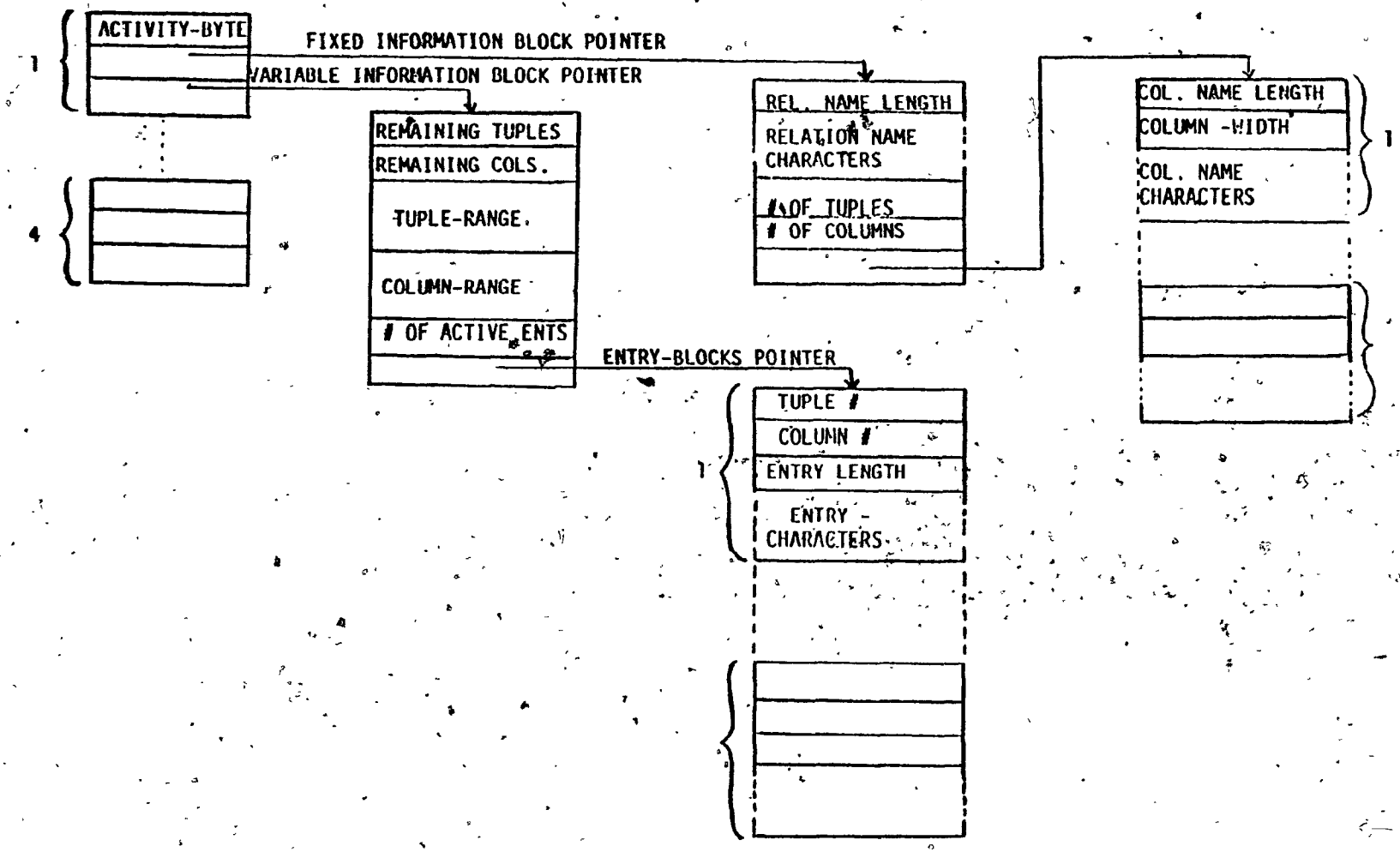


Figure 3. QFP Data Structures

Each of them is associated with a 5 byte block that contains information about the relation.

The following gives the description of each byte of the block.

Byte 1 : Activity Status Byte

Byte 2, 3 : Pointer to Fixed Information Block

Byte 4, 5 : Pointer to Variable Information Block

Activity Status Byte will indicate the presence of valid data in the data structures of a particular relation.

2. Fixed Information Block:

This block consists of information about a relation that does not change very frequently.

Description of the constants used:

RNMC = maximum number of characters in the relation name. ( = 40 currently )

Description of the block:

Byte 1 : length

Byte 2 to byte (RNMC) : relation name

Byte (RNMC + 1) : number of tuples in the relation

Byte (RNMC + 2) : number of columns in the relation

Byte (RNMC + 3),(RNMC+4) : Pointer to column-names-block

3. Column-names Block:

This block stores the column names and column widths.

Constants used :

CNMC = Maximum number of column names ( currently = 15 )

CMXM = Maximum number of characters in the column name

Each column is associated with a (2 + CNMC) bytes block.

Description of the block:

Byte 1 : length of column name in bytes

Byte 2 : column width

Byte 3 to (CNMC + 3) : Column name characters

4. Variable Information Block:

This block consists of information which is subjected to change whenever the corresponding relation requires to be manipulated.

Description of the constants:

EMXM= maximum number of entries in the relation (currently is 25)

ENMC= maximum number of characters in the entry (currently is 39)

Description of the block:

Byte 1 : Remaining tuples to be displayed

Byte 2 : Remaining columns to be displayed

Byte 3 - Byte 4 : Tuple range

Byte 5 - Byte 6 : Column range

Byte 7 : number of active entries in the relation

Byte 8 - Byte 9 : Pointer to entries block

5. Entries Block:

This block consists of maximum 25 entry blocks, each is (ENMC + 4) bytes in size.
Organization of the entries are in ascending order of tuple number as well as column number.

Description of the block:

Byte 1 : tuple number

Byte 2 : column number

Byte 3 : length of the entry

Byte 4 to (ENMC+3) : entry characters

6. Other Data Structures:

a. POINTR is a routine to access the various blocks in the QFP data structure. Detail description of this routine is in section 3.4.

b. BUFFER is a buffer of 75 bytes to be used to keep the framed information to be sent or to be received from HOST. BUFPTR points to the next character available. BUFSIZ contains the number of characters that are still left to be read from the buffer.

c. Main information blocks for all the four relations start at label 'QBERLS' one after another in ascending order.

Fixed information blocks have a common 4 character prefix 'RELF', and variable information blocks have a prefix 'RELV'. Column-name blocks have a prefix 'RELC' and entries block have a prefix 'RELEN'. At the specified labels the corresponding information begins.

d. Each of the relations is associated with a 7 byte buffer intended for keeping information about tuple operators. These buffers have a 5 character prefix 'ROWOP'. I-th byte in the 'ROWOP' buffer corresponds to i-th tuple where 1<= i <=7. Each byte is used as a flag with a value 0 or 1, on which '1' indicates that a print operator 'p' has been initiated.

## 3.3 Design of the Display Format

QFP uses lines 1 to 23 on the screen to display the relations and lines 24 to 25 to display the status of the relations.

The status of a relation is displayed in reverse video mode and contains information such as the number of tuples and columns to be displayed, tuple range and column range of the part of the relation that is currently displayed and size of the relation in terms of tuples and columns.

A status line is formatted as below:

R(XX,YY) T(XX,YY) C(XX,YY) S(XX,YY)

R(XX,YY) : there are XX tuples and YY columns for the corresponding relation which are not displayed currently.

T(XX,YY) : tuples XX to YY are shown

C(XX,YY) : Columns XX to YY are shown .

S(XX,YY) : XX and YY are the actual number of tuples and columns respectively.

Due to space limitation, it is not always possible to display the complete status details of the relations that are maintained internally. Therefore, most of the time only partial status is displayed for each relation. If the complete status is required, function key RED (FRED) can be pressed followed by the logical relation number of the relation (LRNB) for which full details are required.

In QBE system there are two main modes and two special modes of display. These special modes are called SUBMODEs. The two main modes are SPLIT2 mode and SPLIT4 mode.

SPLIT2 main mode can display only two relations, one is on the upper half and the other one is on the lower half of the screen with a seperation line in the middle of the screen. Consequently, lines 24-25 (status area) are divided

into two halfs so that each relation is associated with a status area. The values for LRNB in thare mode is '1' or '2'.

SPLIT4 mode is the same as SPLIT2 mode except that SPLIT4 mode allows display of four relations, each of which occupies one of the four corners of the screen. Seperation lines are also displayed and these lines form a '+' shape. Lines 24-25 are now split into four parts and each part serves as the status area for a relation. The LRNB values for this mode range from 1 to 4.

Not all columns in lines 24-25 are used to display status information. Columns 5 to 80 are allocated for status and the remaining is used for user communication area. This area is further divided into 'command window' and 'value window'. The former occupies columns 1 to 3 of line 24 while the latter occupies columns 1 to 3 of line 25. Column number 4 is always blank and is used to seperate user communication area from status area.

In QBE system there is a facility to view the tuples of a displayed relation page by page through the so called 'window' operation. This operation allows a user to move a window across the relation in all four directions. Sometimes however, it is desirable to see all the tuples and columns of a relation. For this purpose, user can utilize special feature provided (submode) to display the relation

and its entire status on the whole screen, so more tuples and more columns can be shown. This mode is known as SPLITØ mode. Whenever only the entire status of a displayed relation is required, users can press function key red followed by the LRNB of the relation. This is also considered as a special feature, and it can be done irrespective of the current operating main mode. Since a submode changes the screen display format, QFP provides another command that allows suspension of a special feature and restore screen status. The selection of a special feature is not allowed while one is in effect. Figures 8 to 1Ø describe the display format and LRNBs for the relations displayed for all the display modes.

## 3.4 Description of QFP Modules

This section will give detail descriptions of certain major modules of QFP system. The pictorial representation of the interconnections of these modules is given in figure 4 to 7.

COLDS: This module performs QFP initialization as the followings:

- Initialize stack pointer (U register set up)
- Initialize MAIN MODE to SPLIT2 and set up screen display format

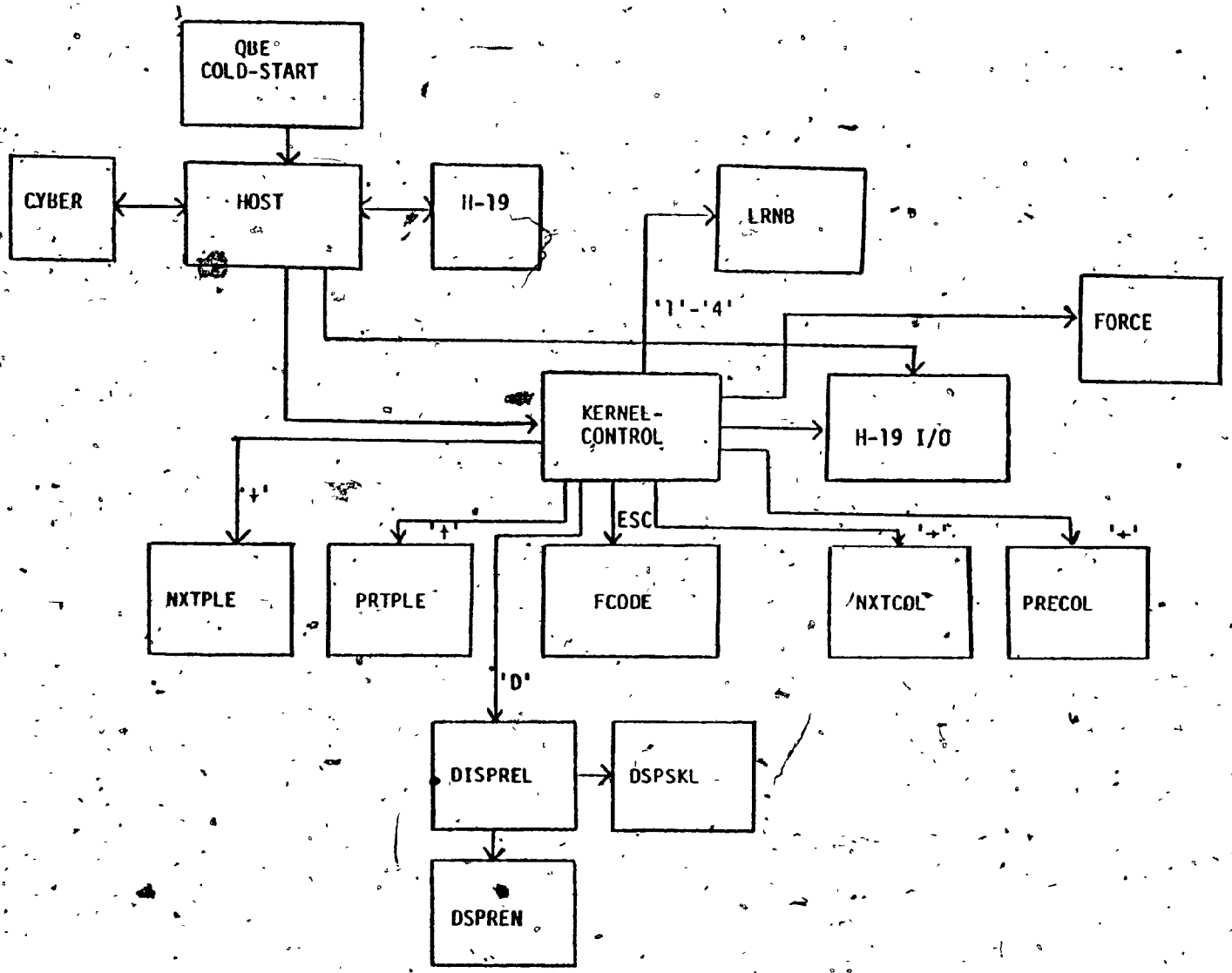
QBE
COLD-START
CYBER
HOST
H-19
LRNB
'1'-'4'
FORCE
KERNEL-
CONTROL
H-19 I/O
'+'
ESC
NXTPLE
PRTPLE
FCODE
/NXTCOL
PRECOL
'D'
DISPREL
DSPSKL
DSPREN

Figure 4.

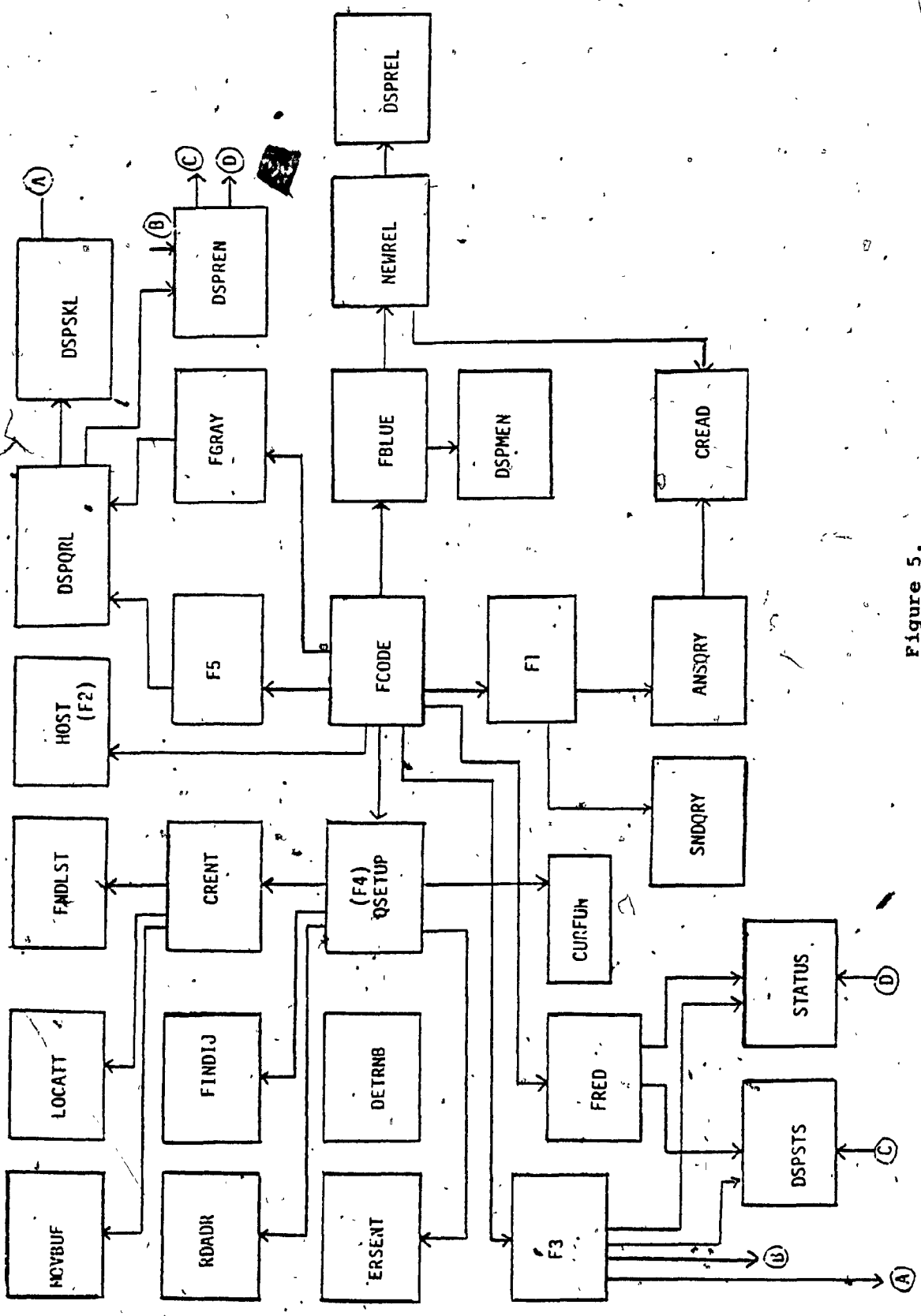
DSPREL
NEWREL
DSPSKL
DSPREN
FGRAY
FBLUE
DSPMEN
CREAD
DSPQRL
F5
FCODE
F1
ANSQRY
HOST (F2)
SNDQRY
FNDLST
CRENT
(F4) QSETUP
CURFUN
STATUS
LOCATT
FINDIJ
DETRNB
FRED
DSPSTS
MOVBUF
RDADR
ERSENT
F3
A
B
C
D

Figure 5.

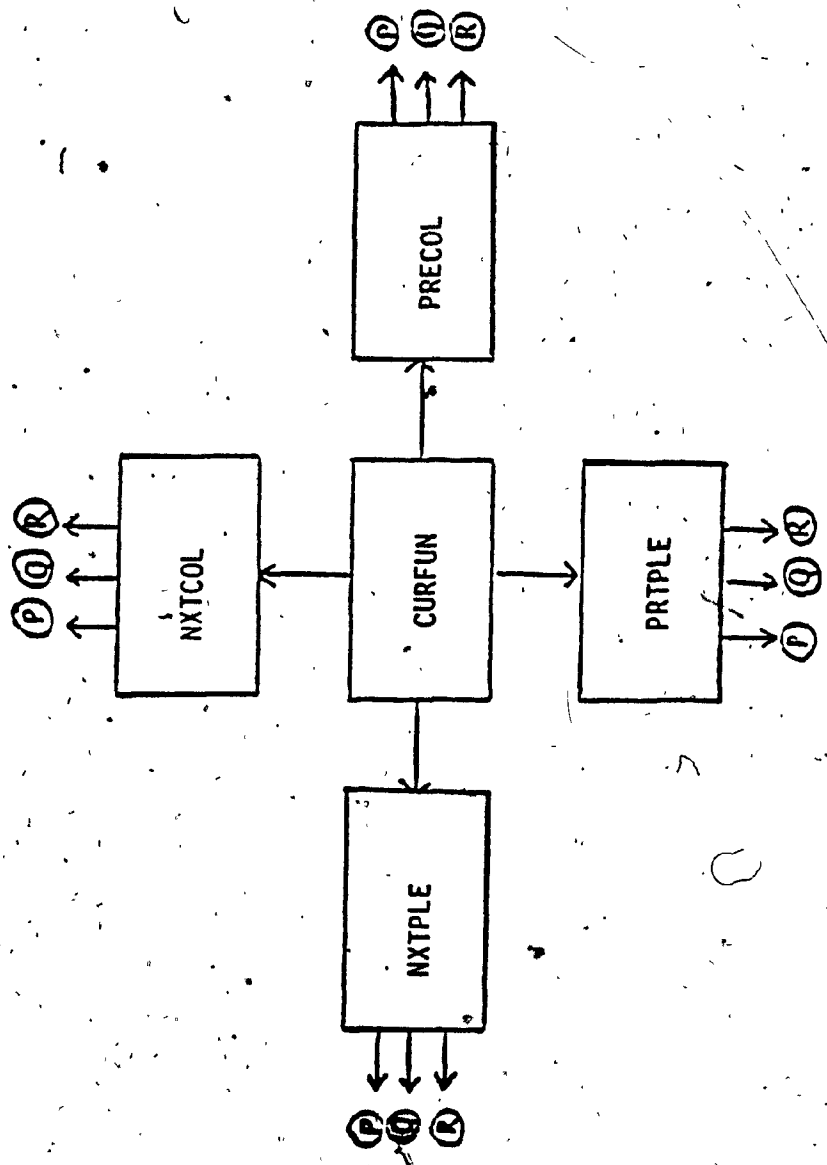


Figure 6.

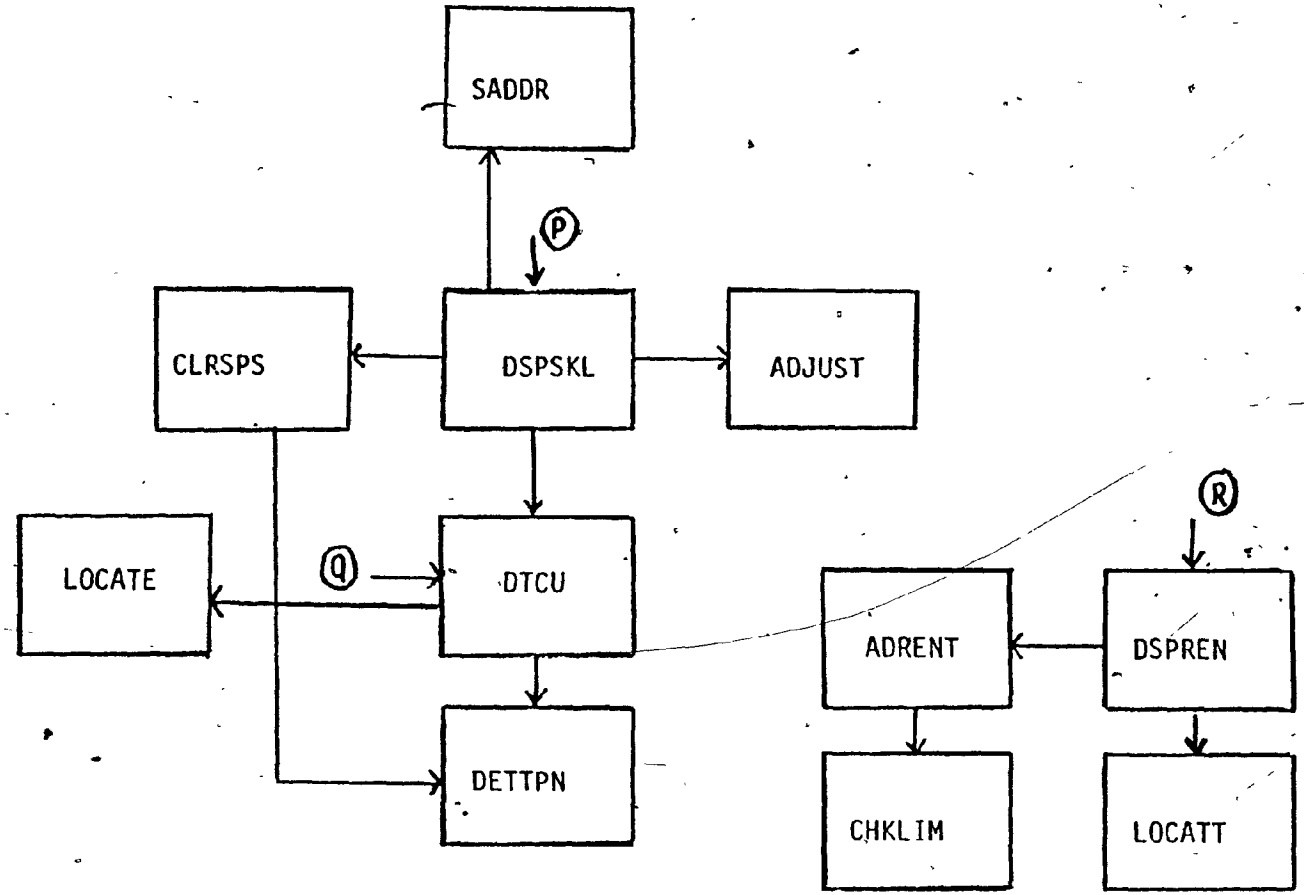
SADDR
P
CLRSPS
DSPSKL
ADJUST
LOCATE
Q
DTCU
R
ADRENT
DSPREN
DETTPN
CHKLIM
LOCATT

Figure 7.

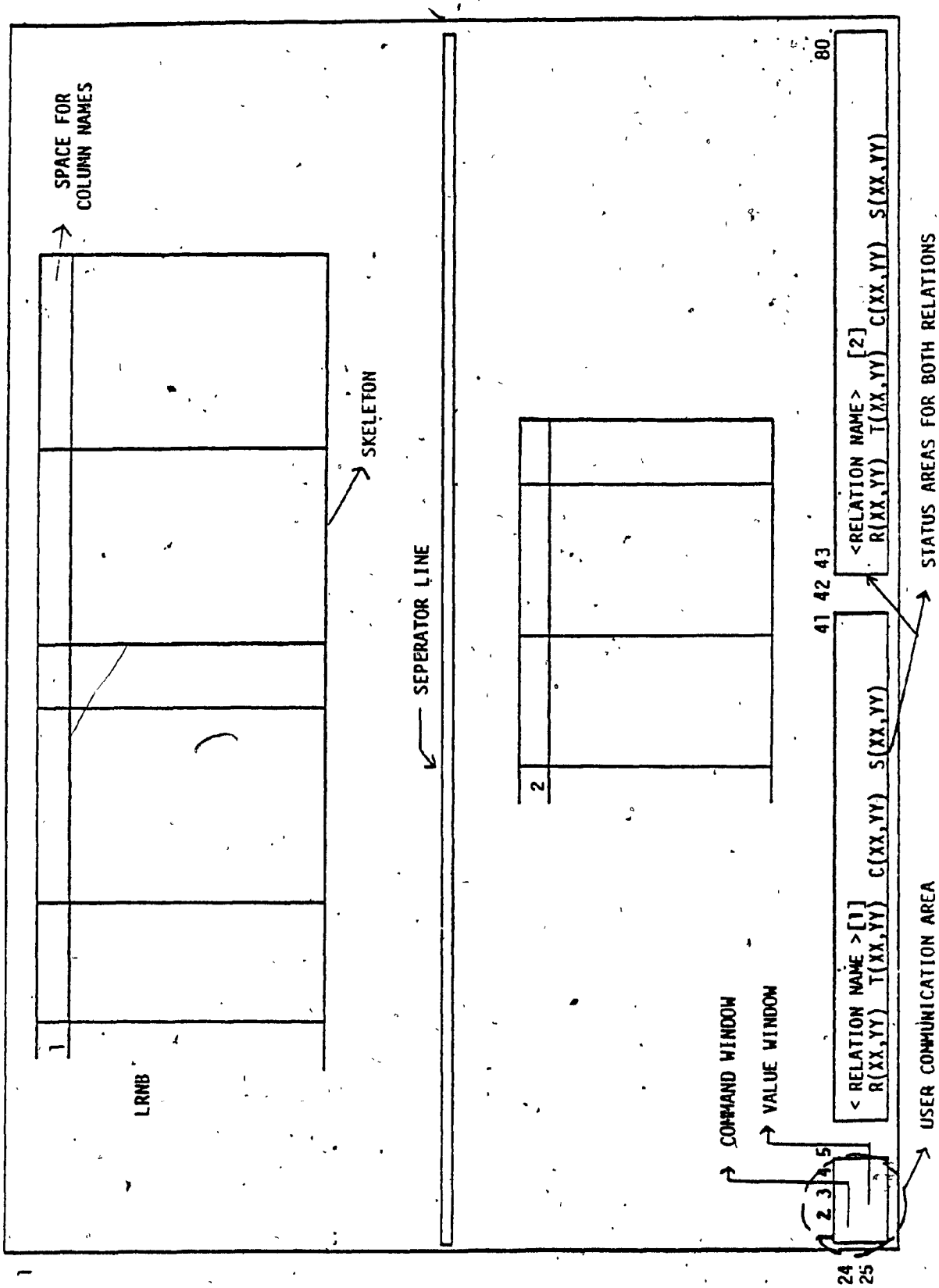


Figure 8. SPLIT2 Mode

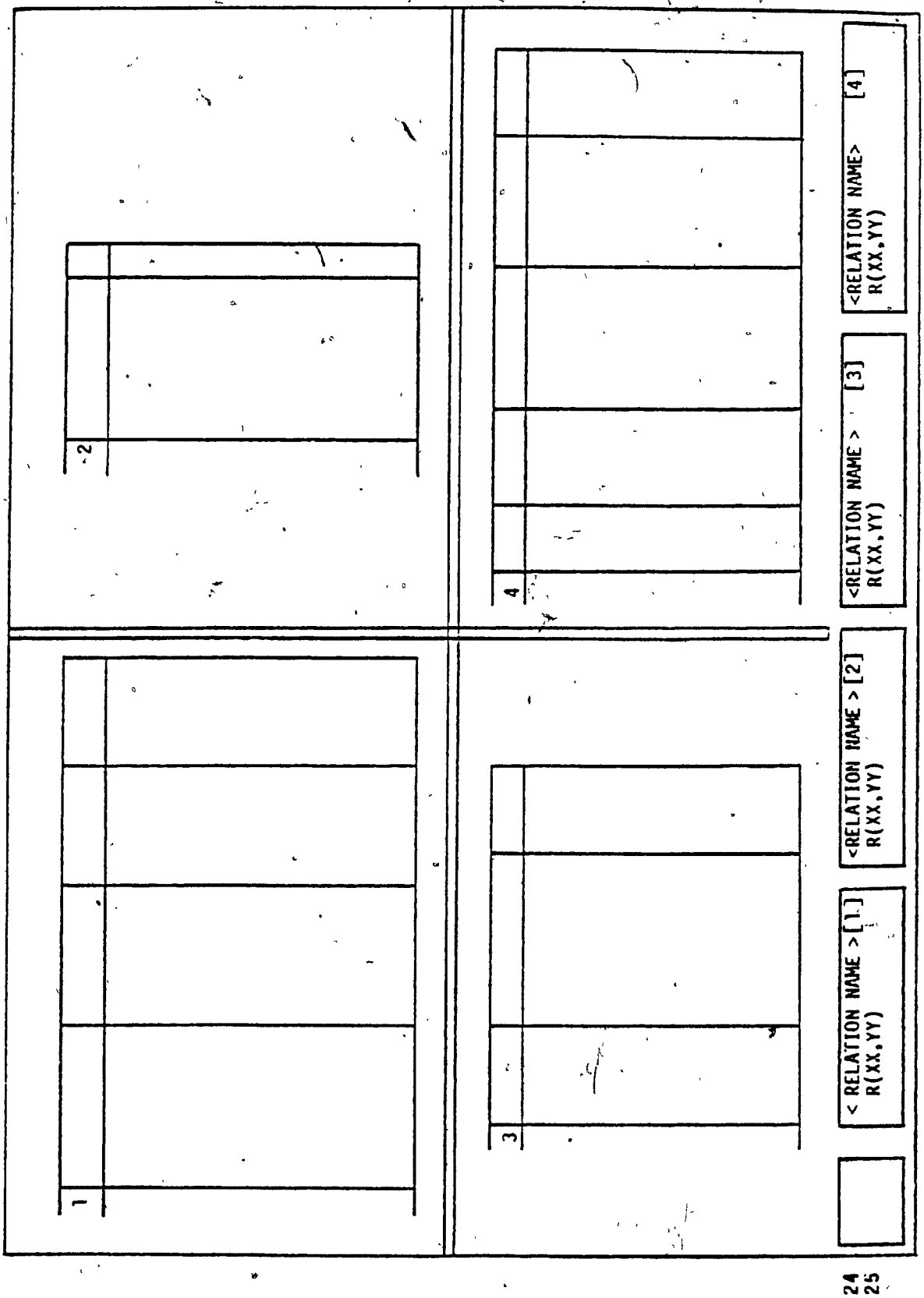


Figure 9. SPLIT4 Mode

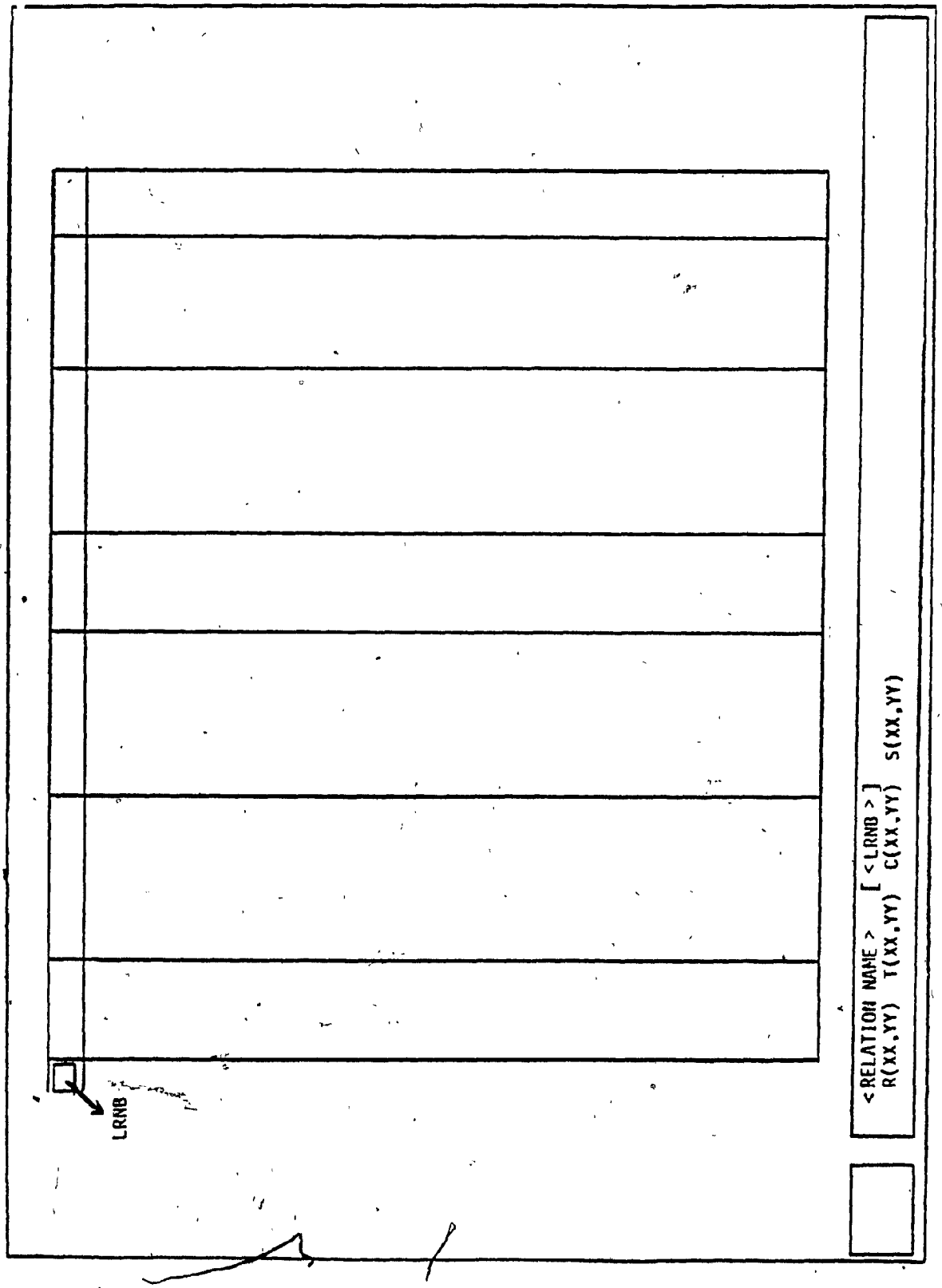


Figure 10. SPLIT0 Mode

- Save the data of all RELF's and RELV's in 'TSTBUF'
- Clear all RELFS and RELVS
- Branch to WARM START

WARMS: This module reads a command and checks for its validity.

CTBLKP: This module performs table look up for command character in the A register. It will set VLDCOM flag if match found and reset it otherwise.

CMDLUP: This module serves as the QFP command loop and functions performed are reading a command, checking for its validity, performing a table look up and branching to an appropriate routine if command is valid, or returning to the beginning of the loop otherwise.

Modules called:

WARMS, to read a valid command character

CTBLKP, to perform a table look up

FCODE: This module services all function key commands. It will determine where the program control should be transferred.

Modules called:

F1, to service function key number 1

F2, to service function key number 2

F3, to service function key number 3

F4, to service function key number 4

F5, to service function key number 5

FBLUE, to service function key BLUE

FRED, to service function key RED

FGREY, to service function key GREY

CURFUN, to service cursor key commands

F1: Transmit the query to HOST and display information retrieved in the appropriate relation. The sequence of functions performed is as the following:

1. Determine the number of relations used for setting up a query.

2. If the number = 0, no query is set up, output error message.

3. Send '+Q+' command to HOST.

4. Wait for input prompt from HOST.

5. Send query in the following format (shows the BNF):

QUERY::=!< of relations>!<relation block> {,<relation block}

<relation block>::= <LRNB>!<relation name>!
<# of active entries>!
<entry block>!
<row operators block>!

<entries block>::=<entry block>{, <entry block>}

<entry block>::=<tuple #>!<col #>!<entry>!

<row operator block>::=<# of row operators>!
<row #>{,<row #>}

6. Send a <cr> to indicate end of query transmission.

7. Read HOST response.

8. If a positive acknowledgement ('@+@') is not received, there is an error in the query. Output error message.

9. Receive the query answer in the following format:

Answer::=!<# of relations>!<relation block>!

{<relation block>!}

<relation block>::=<LRNB>!<entries block>

Refer to BNF shown in step number 5 for <entries block> format.

10. As the information is received, modify the corresponding relations.

11. Display the information received from HOST in terms of new entries in the appropriate relations.

OUTINT: This module converts the value in A register into characters and transmits them to HOST with '!' at the end of the frame. This module is called from 'F1'.

Modules called:

CONCHR, to convert integer to characters

INPINT: This module reads digits until a '!' is received from HOST, converts the digit into an integer value and puts it in A register. This module is called extensively from module 'F1'.

Modules called:

CONINT, to convert a string of characters to an integer.

F2: This module services function key number 2. The function performed is branching to a loop that allows full duplex communication between H19 and CYBER. QHP on HOST is terminated first by sending a '+T+' command to it.

QBEMES: This module displays a nice QBE start up message. It is called from module 'COLDS'.

LRNB: This module handles commands '1', '2', '3' and '4' as the followings:

1. If SPLIT2 main mode is initiated and value in A register (command) is > '2', display error message.
2. If value in A register is < Ø display error message.
3. If value in A register is > 4 display error message.
4. Converts the command into its integer equivalent.
5. Save it in 'LRNBV' indicating that the corresponding relation is active.

TEST: This module is used for testing and also demonstrating some of the features of QFP. This module handles 'T' command which can be issued only at the very beginning right after start up message.

Function performed is:

Restoring data in RELF's and RELV's which is saved in the COLDSTART. Once this command is issued, QFP must not be

used for any other commands except 'F', '1'-'4', and cursor function key commands.

FORCE: This module is used to set the activity status byte of a relation even though the relation is not explicitly brought in by an 'FBLUE' command. This module services 'F' command and requires LRNB input from user. In using this command, LRNB value can not be other than '1'.

INIQBE: This module initializes the ACIA of H19 and the modem and establishes communication channel with CYBER.

F3: This module services function key number 3. Functions performed are as the followings:

1. Read LRNB.
2. If valid LRNB, save the current status of the selected relation.
3. Set up the screen for SPLITØ mode.
4. Display the relation on the whole screen after computing available number of rows and columns on the screen.
5. Display status.

Modules called:

LRNB, to set LRNBV

DSPSKL, to display skeleton

DSPREN, to display entries

F4: This module services function key number 4 which is used when query set up is required.

Functions performed are reading LRNB, clearing the entries in the corresponding relation and allowing user to fill in the entries.

F5: This module services function key number 5 which is used to switch between main modes. Function performed is displaying as many relations as possible in the current main mode.

FBLUE: This module services function key BLUE. Functions performed are as the followings:

1. Ask user to enter relation name.
2. If user enters function keys BLUE again, display a menu of all the relations in the data base and repeat the question.
3. If user enters cursor function key, displays the corresponding set of column names or relation names if possible and repeat the question.
4. Once the user enters the relation name, restore the screen status if necessary and signal HOST that information about a relation is required.
5. If negative acknowledgement is received from HOST, ring a bell and repeat the question.
6. Start reading information sent by HOST and insert in the corresponding relation's data structure. Information from

HOST is received in the following format:

!<relation name>!<# of columns>!<column name block>
{,<column name block>}

<column name block>::=<column name>!<column width>!

7. Display skeleton of the relation.

Menu will be sent by HOST.

FRED: This module services the function key RED. Functions performed include reading the LRNB and displaying its entire status in rows 24 and 25.

FGREY: This module services the function key GREY. Function performed is restoring the screen status that was altered by a submode.

CRENT: This module creates an entry block in the ENTRIES block of the active relation depending on the input parameters( tuple # and column # ). This module employs insertion sort method.

FNDLST: This module traverses entry list in the ENTRIES block of the active relation and sets X register to point to the first byte after the last entry block.

FNDNBS: This module determines the number of bytes between the first byte of the current entry block of the active relation (X register has this value) and the last byte of

the last entry block and returns this value in B register.

RDADR: This module reads the address of CURSOR and makes it available in A and B registers.

DETRNB: This module checks if the current address of the cursor is on one of skeleton lines or outside the relation space and sets a flag accordingly.

ERSENT: This module erases the entries displayed in the active relation, clears tuple range and number of active bytes in the corresponding RELF and RELV.

FINDIJ: This module determines the tuple number and column number of the entry that is being set up.

DSPQRL: This module displays as many relations as possible for the current main mode. It also performs initialization of the screen if necessary.

OUTCOM: This module sends the command pointed to by X register to the HOST.

RDRESP: This module reads HOST respond which is three bytes long.

CONINT: This module converts a string of characters whose

address and length are specified in X register and B register into an integer.

CREAD: This module handles CYBER communication protocol for receiving frames of information.
If buffer is empty, it sends a command to HOST and reads maximum 65 characters until 'cr' is reached. This module supplies next character from the information sent by HOST to the calling routine.

CRDELY: This module creates a delay approximately 10 millisecond by executing a memory reference instruction several times.

DSPREL: This module reads LRNB from user and displays the skeleton, entries and status of the corresponding relation if possible.

WRBLN: This module is used to fill in status area with blanks.

CURFUN: This module handles the cursor function keys' command by transferring control to appropriate routine.

READY: This module reads a value entered in the value window.

NXTUPLE: This module displays the next set of tuples if possible and updates RELV parameters accordingly.

PRTPLE: This module displays previous set of tuples if possible and updates RELV parameters accordingly.

NXTCOL: This module displays next set of columns if possible and updates RELV parameters accordingly.

PRECOL: This module displays previous set of columns if possible and updates RELV parameters accordingly.

CLRSPS: This module blanks the screen space needed by the active relation to be displayed.

CHKABS: This module checks the absence of submodes such that no two submodes exists at the same time.

STATUS: This module sets up the status area in the internal data structure(RELS).

MOVBUF: This module handles movement of data bytes in both directions.

BLNKBF: This module is used to fill in a buffer with blanks.

CONCHIR: This module converts an integer to string of

characters.

DIVIDE: This module divides value in A register and B register, leaving the value in A register and the remainder in B register.

DSPSKL: This module displays the skeleton for a relation and column names. Other modules are called to compute tuple range, column range etc.

SADDR: This module computes the starting position of upper left corner of the relation which is active.

DTCUL: This module computes the tuple range and column range of the active relation which is to be displayed.

DETTPN: This module determines amount of free space in terms of rows and columns for any main mode.

LOCATE: This module traverses COLUMN-NAME blocks until the block with column number same as the one supplied in A register is reached. A register contains the output parameter.

LOCATT: This module traverses the entry block of the currently active relation until the entry block with tuple number same as the one in A register is reached. X register

contains the output parameter.

DSPREN: This module displays entries in ( clears entries of ) the active relation depending on 'DSPFLG' value.

ADRENT: This module computes screen address for an entry whose tuple and column numbers are supplied in A and B registers.

CHKLIM: This module checks if the value in A register is within the specified limits.

POINTR: This module returns the pointer value depending on the value in 'LRNB' and the A register.

The table below explains the input/ output parameters.

| A-reg. | : POINTER VALUE |
|---|---|
| 0 | :Pointer to the corresponding activity status byte |
| 1 | :Pointer to the corresponding RELF |
| 2 | :Pointer to the corresponding RELV |
| 3 | :Pointer to the number of tuples in relations (an address in RELF) |
| 4 | :Pointer to the column names buffer |
| 5 | :Pointer to the tuple range in RELV |
| 6 | :Pointer to the column range in RELV |
| 7 | :Pointer to the number of active entries |

8 :Pointer to the status area

-1 :Pointer to the row operator block

READC: Reads a command character until a valid character is read.
Valid characters are 'A'..'Z', 'Ø'..'9'.

OUTMS1: Outputs an error message and branches to command loop.

OUTBUF: Outputs the contents of a buffer to the terminal. X register has the address of the buffer while B register has the length.

ADRCUR: Sets the cursor to the address specified by A and B registers.

ENBCAD: This module enables cursor addressing.

OUTESC: Writes an 'ESC' to the H19.

ENBRVD: This module enables reverse video mode.

CLEARS: This module clears the screen

ENBGRP: This module enables graph mode.

EXTGRP: This module exits the H19 from graph mode.

ENB25L: This module enables line number 25.

DRWHLN: Draws a horizontal line whose length is specified in B register.

DRWVLN: Writes the character in A register n times in graph mode where n is the number of times specified in B register.

LFR: Writes a line feed.

BSR: Writes a backspace.

INSPØ: Initializes the screen for SPLITØ mode.

INSP2: Initializes the screen for SPLIT2 mode.

INSP4: Initializes the screen for SPLIT4 mode.

DRMDLN: Draws a horizontal line in the middle of screen.

# C H A P T E R 4

## Implementation Details of QHP

QHP is the software system that performs the information retrieval from a data base and passes the retrieved information to the end user via QFP. This interface is written in PASCAL and resides in CDC-172 mainframe. To test the system, RISS DBMS is used to generate a Host Database. This chapter will discuss the implementation of QHP in detail.

### 4.1 QHP System Level Details

As stated above, QHP is an interface between QFP and the database. There are two prime functions that QHP has to perform:

1. QFP-QHP Command Interface
2. QHP-DBMS Interface

1. QFP-QHP Command Interface:

This interface will initiate QFP commands by sending a code ('@m@') to QFP. QFP will then transmit the query

tuples in the form of frames. (The BNF definition of the framed information was given in section 3.4). Upon receiving the framed information from QFP, it is seperated into several tuples' information. Each tuple is named a 'query tuple' and it will be translated into its internal data structures. Further retrieval operations will base on these query tuples. Syntax checking is also done at this stage of analysis.

Information from QFP is classified into 4 different types of entry:

1. Constant entry
2. Example entry
3. Print entry
4. Keyword entry

Each entry has its own internal buffer to keep the associated parameters. The description of these buffers can be found in section 4.3. The answer to the query if any, will be 'framed' by this interface and sent back to QFP.

## 2. QHP-DBMS Interface

This interface directly communicates with the data base. Since in this major report RISS DBMS is used as the database, this interface makes use of some modules of the RISS Application Level Interface by means of external calls. Detailed description of these modules and how they can be

used is given in [Risshdr80].

There are four different types of queries on which the data base can be searched, each one is briefly described below:

1. Simple Type of Query: This results in a retrieval process where there is no constant entry in the query. This type of query is answered by means of the projection operation.

2. Qualified type of query: This is a retrieval process in which there is at least one constant entry in the query. Query of this type needs select and project operations.

3. Query Using Keywords: This is a retrieval process in which there is at least one keyword in the query.

4. Query Using Common Example Elements: This is a retrieval process in which there are one or more common example elements between any two tuples within one relation or between two different relations.

## 4.2 Major Groupings of QHP Software System

QHP software system is divided into eight major groups based on the functions performed. Each group may work independently or depend on other groups. The description of each group and the modules involved will be described below, the pictorial representation of the modules contol flow is given in figure 11.

1. Initialization Interface:

Performs the initialization process of the internal buffers. It also assigns the initial values of some fields of these buffers.
Module involved: Init.

2. QFP-QHP Receive Command Interface:

This interface serves as QHP command loop interface. It receives command sent by QFP, checks for a valid command and transfers control to an appropriate interface routine.
If command is 'M', 'A', 'B', 'C' or 'D', control will be transfered to group 3.
If command is 'R', control will be transfered to group 4.
If command is 'Q', control will be transfered to group 5.

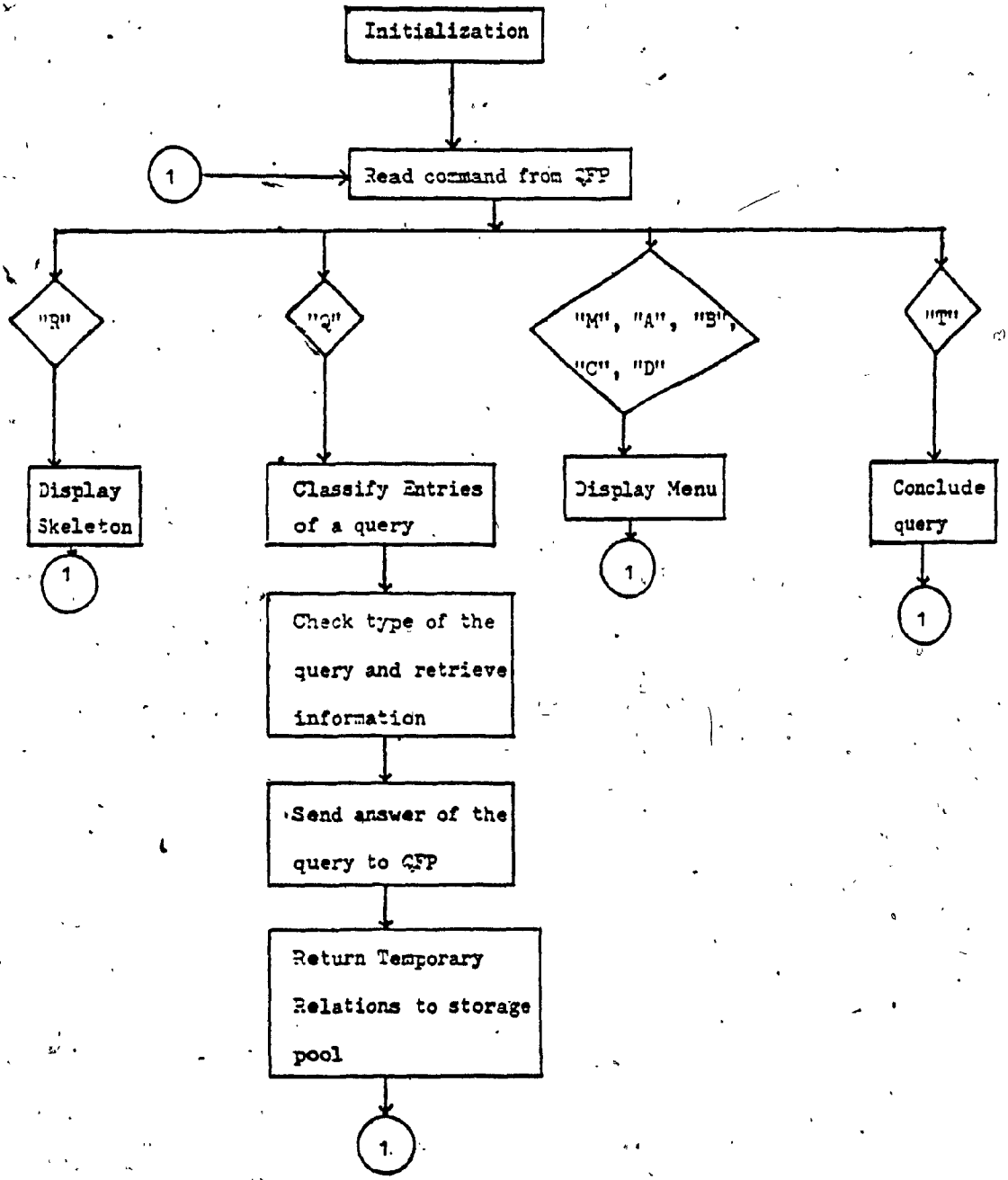


Figure 11. QHP System Control Flow

If command is 'T', the query session is concluded.

Modules involved: Main and Readcommand.

3. Menu Display Interface:

This interface sends a set of relations in the database dictionary to QFP. It provides the facilities to display the next set or the previous set of relations in the dictionary as well as the next columns or previous columns of the relations.

Module involved: Menu.

4. Skeleton Display Interface:

This interface sends the skeleton of a particular relation in the dictionary to QFP.

Module involved: Skeleton.

5. Entries Classifier:

This interface will classify the query entries sent by QFP into the appropriate types and store them into the respective internal buffers.

Primary Module: Checksyntax.

Supporting modules: Qualifier, Keyword, Example, Storeconstant, Storexample, Storeprint, Storekeyword,

Relopprocess, Keyprocess, Exampleprocess, Initvalue, Check.

6. Database Retrieval Interface:

This interface will classify the query into the appropriate type and make retrieval based on the query type.
Primary Module: Checktype.
Supporting modules: Processkey, Report, Project, Select, Processlink, Checkunion.

7. QHP-QFP Send Answer Interface:

This interface will send the answer of a query to QFP in the form of frames.
Module involved: Transmit.

8. Garbage Collection Interface:

After each query session, all temporary relations used are returned to the storage pool by this interface.
Module involved: Purgetemp.

## 4.3 Data Structures on QHP

This section will discuss about the structure of the internal buffers used and the description of each field on the buffers. The pictorial representation of the data structure is shown in figure 12. There are currently five internal buffers used, namely: TUPLEBUF, CONSTANTBUF, EXAMPLEBUF, KEYBUF and PRINTBUF. The complete description of these buffers and some other supporting buffers is presented below:

1. TUPLEBUF

Framed information received from QFP is kept in this main internal buffer tuple by tuple. Currently twenty such buffers are being used to accomodate information of the query tuples.

The following will give the description of each field of this buffer.

| | |
|---|---|
| LRNB | : integer |
| RELNAME | : alfa |
| TEMPNAME | : alfa |
| STARTCONST, MAXCONST | : integer |
| STARTPRINT, MAXPRINT | : integer |

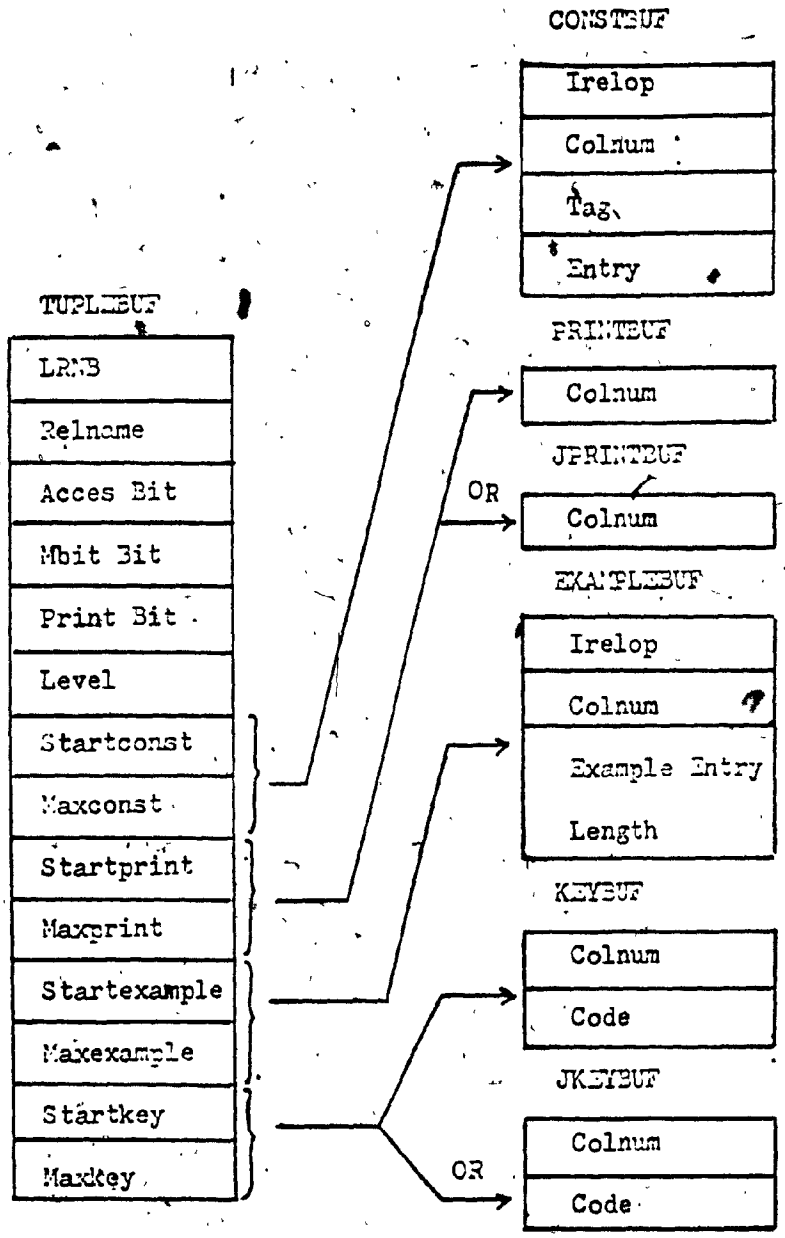


Figure 12. Data Structures of QHP

STARTEXAMPLE, MAXEXAMPLE : integer

STARTKEY, MAXKEY : integer

ACCESS, MBIT, PRINT : boolean

LEVEL : integer

LRNB: Is the logical record number of the relation.

RELNAME: Is the relation's name imposed by a user.

TEMPNAME: The result of a process like Select, Project or Join will be put in a temporary relation. The name of the temporary relation is shown by this field. It is initialized to 'zzzzzzzzzz' and upon completion of one process, a function called NEWTEMP will be called to supply the new name for that relation. This temporary relation will always be referred whenever the original relation is referenced. After a join process, the temporary name will start with a 'j' to show that the print elements and key elements of the relation if any, are stored in JPRINTBUF and JKEYBUF respectively.

STARTCONST: Is a pointer to buffer CONSTANTBUF to show the first position of constant(s) a particular query tuple has.

MAXCONST: Is the total number of constant elements a query tuple has in CONSTANTBUF.

STARTPRINT: Is a pointer to buffer PRINTBUF or JPRINTBUF to show the first position of print element(s) a particular query tuple has.

MAXPRINT: Is the total number of print elements a query tuple has in PRINTBUF.

STARTEXAMPLE: Is a pointer to buffer EXAMPLEBUF to show the first position of example element(s) a particular query tuple has.

MAXEXAMPLE: Is the total number of example elements a query tuple has in EXAMPLEBUF.

STARTKEY: Is a pointer to buffer KEYBUF to show the first position of key element(s) a particular query tuple has.

MAXKEY: Is the maximum number of key elements a query tuple has in KEYBUF.

ACCESS: Is a flag indicating the presence of valid data in the particular position in TUPLEBUF.

MBIT: Is a flag to indicate whether the particular tuple's information is still valid. It will be set in either process JOIN, PROJECT, SELECT or CHECKUNION if the corresponding process failed or the particular relation will

not be needed any longer.

PRINT: Is a flag to indicate whether a row operator is initiated in the particular query tuple.

LEVEL: This field is used to determine the level of a query tuple which is important in processing the common elements. A query tuple with the lowest level will be processed first.

2. CONSTANTBUF

This buffer is used to store information about a constant. There are currently twenty such buffers. The following will give the description of each field of the buffer.

IRELOP : integer

COLNO : integer

TAG : char

ENTRY : depends on the tag, the value can be integer, real, character or string

IRELOP: Denotes a relational operator used in a tuple encoded as follows:

'<' = 1

'>' = 2

'<=' = 3

'>=' = 4

'<>' = 5

'=' = 6

Initially the IRELOP field of all query tuples will be assigned to 6.

COLNO: Is the column number in the relation on which the constant entry occurred.

TAG: Is a flag to indicate whether the constant entry is of type integer, real, character or string .

ENTRY: Can be an integer number, a real number, a character or a string depending on TAG field.

3. PRINTBUF: Is a buffer used to store the column number of a print element that occured in a query tuple. There are currently 20 such buffers. After the JOIN operation between two particular relations, information on this buffer which belongs to those relations is transferred to JPRINTBUF.

4. EXAMPLEBUF: This buffer is used to store information about an example element. There are currently 20 such buffers.
The following will give the description of each field in this buffer.

COLNO :integer

SENTRY :string

LEN :integer

IRELOP :integer

SENTRY: This is the example element without '^' prefix.

LEN: Is the length in characters of the example element.

The definition of other fields is the same as the ones in CONSTBUF.

5. KEYBUF: Is used to store information about a key element. There are currently 20 such buffers.

The following will give the description of each field on the buffer.

CODE : integer

COLNO : integer

CODE: Is the keyword's code. The pre-determined keywords and their codes will be shown below.

| Keywords | Codes |
|---|---|
| AVG | 1 |
| CNT | 2 |
| MAX | 3 |
| MIN | 4 |
| SUM | 5 |

Other Data Structures:

1. JPRINTBUF: Is used to store the column numbers of print elements on query tuples after a Join process. This buffer is referred by module PROJECT and REPORT.

2. JKEYBUF: Is used to store information about the key elements of query tuples after a Join process. It is referred by module PROCESSKEY, PROJECT and REPORT.

3. GCOLNUMBERS: Is used to store the column numbers of print elements of all query tuples currently imposed. GTOTALCOLS will show the total number of PRINT elements in each query tuple. This buffer is referred by module PROCESSLINK and PROJECT.

4. TPOOL: Is a buffer to supply the temporary name of a relation. It is used by module NEWTEMP.

5. DBUF: Is a buffer of 100 characters to load the framed information from QFP that will be verified later. It is Used by module CHECKSYNTAX.

6. WORD: Is a buffer for the five pre-determined keywords and is used by module KEYWORD.

## 4.4 Description of QHP Modules

This section gives the detail description of the major modules of QHP system.

QUERY

This module is called if command is 'Q'.
It performs the following functions:

1. Initialize all parameters of the internal buffers.
2. Read the number of relations to be queried.
3. Loop and call module CHECKSYNTAX as many as the number of relations.
4. Read and insert row-operator into the appropriate tuple of buffer TUPLEBUF.

CHECKSYNTAX

This module will perform the following functions:

1. Read the various entries of the query.
2. Check the type of the entry.
3. Check the syntax of the query entries.
4. Insert the entry in the appropriate buffer ( CONSTBUF, PRINTBUF, EXAMPLEBUF or KEYBUF )
5. Insert the query tuple's information in the appropriate position in TUPLEBUF as shown below:

- TCOUNT = the first available position in TUPLEBUF. Initially is 1.

- TPLNB = The position of a query tuple within one relation.

- TOTALTPLS = position in TUPLEBUF on which information about a tuple's entries will be placed. It is equal to (TCOUNT + TPLNB - 1).

CHECKTYPE

This module will determine the type of a query. Retrieval can then be made based on the query type. This module has several supporting modules to service the four different query types. The pictorial representation of the modules flow of control is exhibited in figure 13 which in turn is discussed below:

1. Service the constant elements in the query if any, tuple by tuple by calling module SELECT.
2. When there is more than one query tuple in the query, call module PROCESSLINK to process the common example elements in the query if any.
3. Process the key elements in the query if any, tuple by tuple by calling module PROCESSKEY.
4. Union all query tuples associated with same original query relation name.

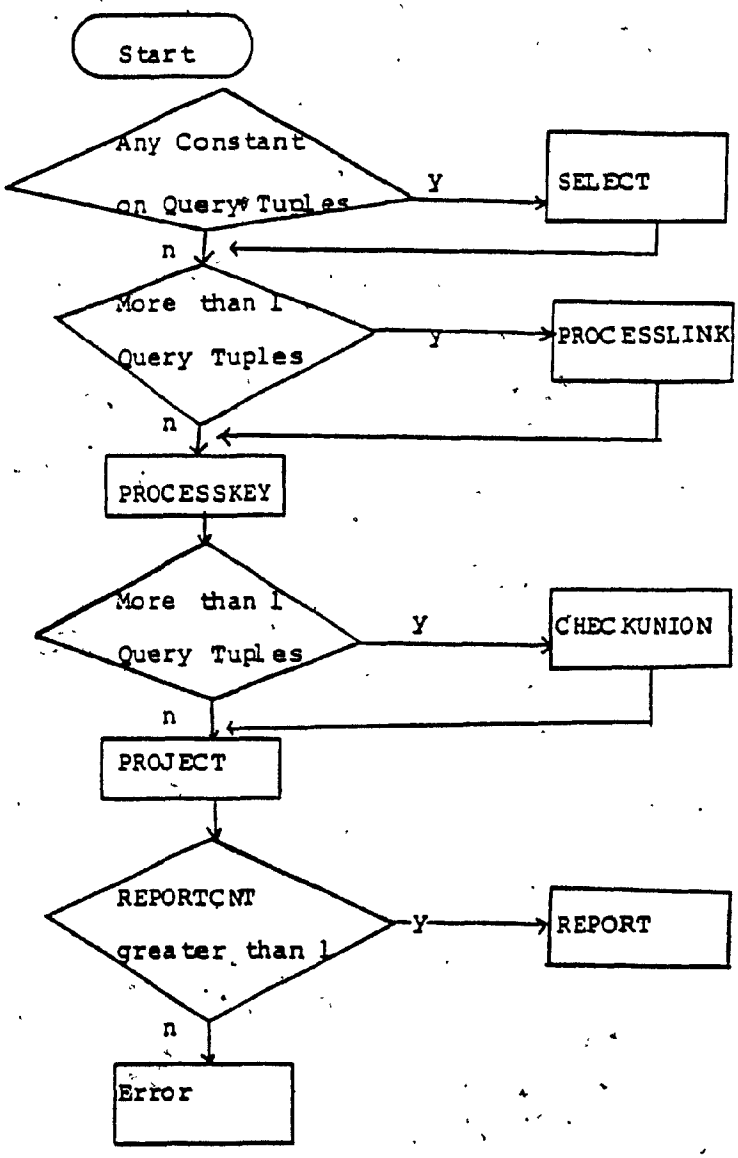


Figure 13. Flowchart of CHECKTYPE Module

5. At this stage, there exists a set of query tuples which have print elements. Module PROJECT is called to collect the entries of some relations under which print elements were initiated in certain columns.

6. If REPORTCNT > Ø, call module REPORT to frame the output of the query; send negative acknowledgement to QFP otherwise.

SELECT

This module will do the selection process on the relation as appeared on the RELNAME field of TUPLEBUF buffer. This process will use the constant entries in CONSTANTBUF as pointed by STARTCONSTANT and as many as MAXCONSTANT. If MAXCONSTANT is greater than one, an 'AND' predicate is used. The output of the process if any will be kept in a temporary relation. The name of the relation is supplied by function NEWTEMP.

If there is no answer to this process, the MBIT field of TUPLEBUF will be set.

Modules called:

SELECTTPL which returns true if a tuple in the relation satisfies the constant constraint, false otherwise.

PROJECT

This module will perform the projection process of all columns of a query tuple which have PRINT entries or a query tuple which has a row-operator. If a row-operator is initiated, the entire relation will be displayed. The relation used can be the original relation or the temporary relation depending on whether the name of the temporary relation is 'zzzzzzzzzz' or something else. If the temporary relation is used and the name starts with a 'j', JPRINTBUF will supply the columns to be printed out. For any other cases, PRINTBUF will do it. If there is no answer to this process, the MBIT field of TUPLEBUF will be set.

REPORT

This module will send the answer of a query to QFP in the form of frames. Since the answer has been prepared by module PROJECT and stored in a temporary relation, this module will always refer to the temporary relation of the query tuple.

NEWTEMP

This is a function to supply the name of a temporary

READNB

This module will extract the characters in DBUF until a '!' is hit. The characters will then be converted into the corresponding integer number.

READREAL

This module will extract the characters in DBUF until a '!' is hit. The characters will then be converted into the corresponding real number.

READSTRING

This module will extract characters in DBUF until a '!' is hit. It will then return the string and the length of the string.

QUALIFIER

This is a function to return true if the first one or two characters available in DBUF is a qualifier or false otherwise.

The codes assigned to the qualifiers are as the following:

| Qualifiers | codes |
|---|---|
| < | 1 |
| > | 2 |
| <= | 3 |
| >= | 4 |

<>  5

=  6

KEYWORD

This is a function which will be using a sequential search technique to check whether the first 3 characters available in DBUF is one of the pre-determined keywords in buffer WORD.

EXAMPLE

This is a function to decide whether the first character available in DBUF is the prefix for an example entry (' ^ ').

STORECONSTANT

This module stores the constant values supplied by a user into buffer CONSTANTBUF and update the corresponding buffer's parameters (CONSTCOUNT and MAXCONST).

Modules called:

READNB to read the constant integer number

READSTRING to read the constant string

READREAL to read the constant real number

STOREXAMPLE

This module extracts an EXAMPLE entry from DBUF and insert it in buffer EXAMPLEBUF with the length of the EXAMPLE entry and the column number where it occured. The corresponding buffer's parameters (EXAMPLECOUNT and MAXEXAMPLE) will then be updated.

STOREPRINT

This module will insert the column number of PRINT entries in buffer PRINTBUF and update the corresponding buffer's parameters (PRINTCOUNT and MAXPRINT).

STOREKEYWORD

This module will store the code and the column number of a keyword in buffer KEYBUF and update the buffer's parameters (KEYCOUNT and MAXKEY).

RELOPPROCESS

This module is called if the current token is a qualifier ('<', '>', '<=', '>=', '<>', '=' ). It will check whether the next token following the qualifier is a CONSTANT or an EXAMPLE entry and store the token in the corresponding buffer by calling STORECONSTANT or STOREXAMPLE respectively.

KEYPROCESS

This routine is called if the current token is a keyword. The next token can be an EXAMPLE entry or nothing at all.

Modules called:

STOREKEYWORD to store the current keyword.

STOREXAMPLE to store the next token if it is an EXAMPLE entry.

EXAMPLEPROCESS

This routine is called if the current token is an EXAMPLE entry. The expected next token is a qualifier followed by an EXAMPLE entry or a qualifier followed by a CONSTANT entry or nothing at all.

Modules called:

STOREXAMPLE to store the current example element.

STORECONSTANT or STOREXAMPLE again depending on the type of the next entry.

INITVALUE

This module will initialize the value of each field on buffer TUPLEBUF.

CHECK

This module will check the type of current token and go to the appropriate module.

Modules called:

RELOPPROCESS if the current token is a qualifier.
KEYPROCESS if the current token is a keyword.
EXAMPLEPROCESS if the current token is an EXAMPLE entry.
STORECONSTANT if the current token is none of the above.

GETCOLUMN

This module will return a column number which entry is the same as the supplied entry. This module is called because there is a possibility of changes in the position of the column after either PROJECT process or JOIN process.

PURGETEMP

After each query session, a garbage collection action is taken by this module. All temporary relations used are returned to the storage pool. This action guarantees that a temporary relation is readily available.

PROCESSLINK

This module will process the linkage of two query tuples within 1 relation or between 2 different relations. Initially all query tuples are assigned to the highest possible level (HPL). They will later be demoted or promoted to other levels on the basis of dependencies (linkages) with other tuples. The steps in the sequencing scheme are shown in figure 14. All actions are taken on these normal query tuples which exist at the level which is currently specified as the lowest level. Each time the lowest level is processed the following will take place:

1. A test is made to determine if there is any direct linkage (common example elements) between any of the query tuples at this level. If no direct linkage is found and the lowest level is the HPL then the task is completed. If there is no direct linkage and the tuples are at a level below the HPL, the tuples from this level are reassigned to the next higher level where they will be processed with tuples which were previously at the next higher level. This effectively changes what is designated as the lowest level for the next pass. Alternately, if tuple direct linkage is found, then go to substep 2.

2. A prime tuple is selected from the set of tuples existing at this level by calling module SELECTPRIME.

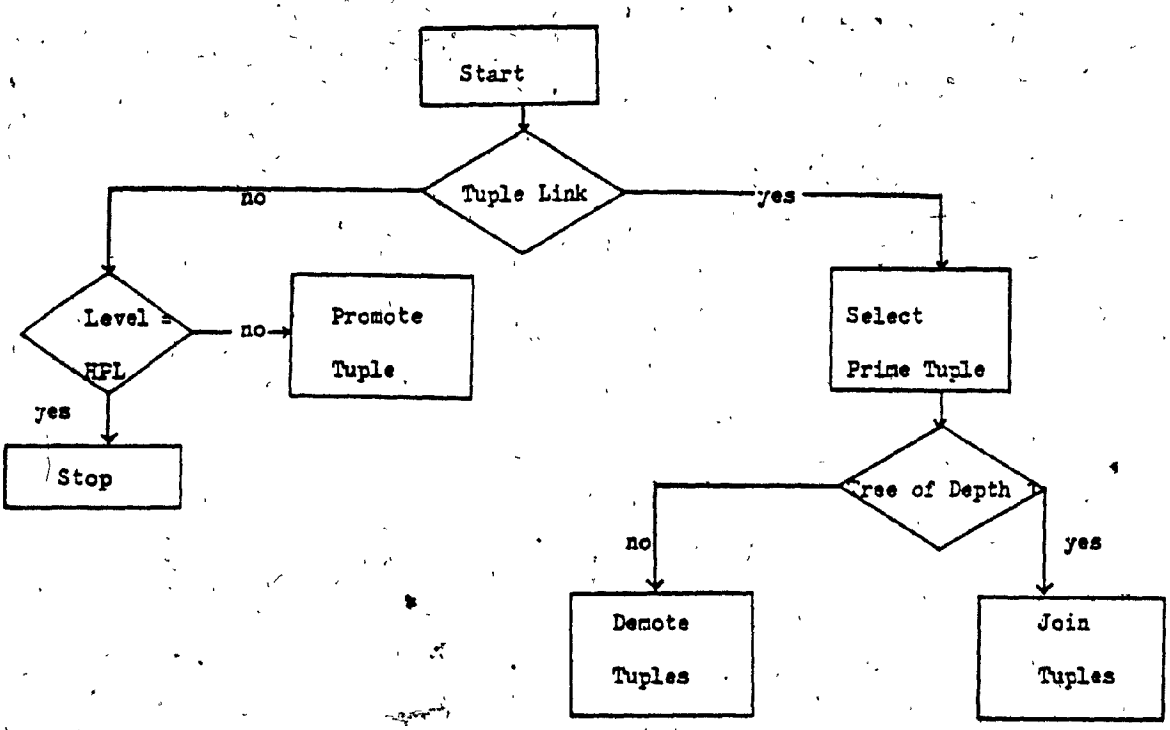


Figure 14. Process of Linked Tuples

3. A query tuple dependency graph is a graph that consists of a set of nodes which corresponds to query tuples, and a set of edges which indicates at least one direct linkage (dependency) between the connected query tuples (nodes). Direct links exist between query tuples having the same example elements. It is desired to join tuples to the prime tuple only if a query tuple dependency graph can be constructed for the tuples at this level which has a tree of depth one with the prime tuple as the root. A test is made to determine if any tuples at this level which do have direct linkage with the prime tuple also have direct linkage to other tuples also at this level. If such direct linkage is found, go to substep 4, otherwise go to substep 5.

4. An attempt is made to reassign all tuples from this level except the prime tuple to newly created lower levels. All tuples with direct linkages to the prime are assigned to the next lower level. All remaining tuples with direct linkages to this new lower level are assigned to the next lower level, etc. Those tuples which cannot be demoted in this manner are left at the level holding the prime tuple.

5. The prime tuple ( to be defined in module SELECTPRIME ) may now be joined with tuples at this level which directly link to the prime level. All leaf tuples can be joined with the root tuple in a series of binary joins between each leaf and the current root (the root tuple characteristics change as a result of any prior joins).

The sequencing of joins resulting from substep 1-5 has the characteristic of assuring that joins involving the query tuple with the most output (p) elements are dealt with at the very last moment. This delays the need to carry columns involving output elements of at least one query tuple in temporary relations, until late in the sequences of join operations.

SELECTPRIME

This module will return a prime tuple to module PROCESSLINK. Candidates for prime tuple must have direct linkage with some other tuples at the level of interest. If the lowest level under examination is the HPL, then the selection of the prime tuple is normally based on the query tuple which has the most elements destined for user output(e.g., 'p' elements or elements which will eventually mapped to explicit join relations). If the level of interest is lower than the HPL, the selection is based on the query tuple having the most direct linkages with the set of query tuples which are currently assigned to the next higher level.

JOIN

This module will join the active columns of two query tuples having one or more common example elements. It is

called by module PROCESSLINK. The result of the Join process is kept in a temporary relation which name is supplied by the pool of names and will start with a 'j'. The column numbers of PRINT elements of the two tuples are moved to JPRINTBUF based on the position of the PRINT elements in the temporary relation. As for the PRINT elements, the column numbers of the KEY elements of the two tuples are also moved to JKEYBUF based on the position of the KEY elements in the temporary relation.

# C H A P T E R 5

## Concluding Remarks and Further Work

### 5.1 On Intelligent Front End Terminals:

Microcomputers are increasingly become popular and cost effective in many applications. Almost all of the modern day intelligent CRT terminals contain at least one microprocessor resident in each of them. Such microprocessor based terminals, invariably, have the following features:

1. Cursor addressability and cursor control keys.
2. Programmable function keys.
3. Limited graphics or business graphics support ( ability to draw vertical lines, horizontal lines, arbitrary piece-wise linear curves, histograms etc.)
4. Scrolling.

The Heathkit H19 terminal is one such terminal. An intelligent CRT terminal of this kind can be used to support user communication through QBE. The graphics feature is useful to draw the skeleton of a relation in QBE; the programmable function keys are useful to invoke a pre-programmed operation; the cursor facility is useful to

enter a query on the displayed skeleton; and finally the scrolling facilities are helpful for the user to scan the tuples of a retrieved relation when all its tuples are too large to fit into the screen.

The resident microprocessor of an intelligent terminal that is used for user communication through QBE can be programmed to take into account of several front end functions.

Following are some examples of such functions:

1. Partition the screen into disjoint parts for displaying the multiple relations involved in query formulation.
2. Check and prevent the user typing across the partition when he types a constant or an example variable in a selected relation.
3. Display and control status details of the relations displayed on the screen.
4. Perform scrolling (horizontally and vertically) with the help of the locally stored information and exchange information in 'large chunks' with main computer.

The use of front end processing for the functions cited above has two major advantages:

a. It relieves the main computer free of some processing.
b. It will provide quicker response than otherwise is possible.

The resident microprocessor of an intelligent terminal can be programmed to perform such front end processing. Optionally, an additional microprocessor with required amount of memory (RAM and ROM) can be added to the terminal to take care of the front end processing.

## 5.2 Contributions of This Project

An information retrieval system based on QBE that involves two processes which communicate to each other is proposed. The first process takes care of the end user communication and the second process performs the data base retrieval operations.

The process that interacts with end users is named QBE Front end Process (QFP) and the one that communicates with the database is named QBE Host Process (QHP). The functions of the Front end Process are:

1. Extract user commands from the terminal and transfer the information to QHP in the form of frames.
2. Allow direct communication between Cyber and H19 terminal.
3. Handle specific I/O with H19 terminal. It also handles the changing of the display modes; it can display one, two or four relations at a time.
4. Read and interpret user commands entered by pressing

programmed keys as well as displaying status information in various ways.

The Host Process, on the other hand, has the following features:

1. Receive QFP commands and translate them into QHP internal representation.
2. Check the type of user's queries and make retrieval based on the query type.
3. Send a set of relations in the database dictionary if requested. It also allows the display of the next set or previous set of the details of relations stored in the data dictionary.
4. Send the answer of the query to QFP, if any, or a negative acknowledgement.

The data structures involved in these processes are distinct. Both processes however, have to be aware of the communication links that can occur at any time between them. The communication data structure is the frame.

QFP was written for the MC6809 Microprocessor by Mr. R. M. Kotamarti and QHP was developed in PASCAL for Cyber 172 and tested with an example database supported by the RISS database management system.

## 5.3 Further Work

One further work could be aimed to integrate the QFP software with a terminal by incorporating an additional microcomputer and to study and evaluate the usefulness of such a front end processing.

Modifications to QFP system can be made to free the system from the following bugs:

1. Sometimes some relations could not be displayed perhaps due to an inherent 'bug'. However, after the relations were renamed or some column names of the relations were renamed, they could be displayed and query could be formulated accordingly.
2. In SPLIT2 mode, a query involving two relations were formulated. Only relation that was in LRNB 2, however, displayable.

The ability to handle multiple links with more than two relations in query formulation would greatly increases the applicability of the QBE system. Further work could be done to provide QFP with the facility to allow the display of a new relation that results from the retrieval process.

## References

1.[Ander78] Anderson, N.D., Burkhard, W.A.
"Minisequel Relational Data Management System"
Databases: Improving Usability and Responsiveness
Schneiderman, B.. Academic Press, New York, 1978, pp.57-76.

2. [Anton78] Antonacci, F., Dell'Orco,P., Spadavecchia, V.N.
"AQL: A Problem Solving Query Language for Relational Data Bases".
IBM Journal Res. Develop., Vol 22, no.5, 1978, pp. 554-559.

3.[Baxter78] Baxter, A. Q., Johnson, R. R.
"A Block Structured Query Language for Accessing a Relational Data Base".
Proc. ACM-SIGIR, Int. Conf. On Inf. Stor. And Retrieval, May 1978, pp.109-130.

4. [Boyce75] Boyce, R. F., Chamberlin, W. F., Hammer, M. M.
"Specifying Queries as Relational Expression: The SQUARE Data Sublanguage".
Comm. ACM, vol. 18, no. 11( nov 1975 ), pp.621-28.

5. [Buneman79] P. Buneman, R.E. Frankel.
"FQL-- A Functional Query Language".
ACM-SIGMOD 1979, International Conf. On Management of Data

6. [Chaiho80] Chaiho C. Wang
"A Probabilistic Approach to Storage Compression of Large Natural Language Data Bases".
Proceedings 4 COMPSAC 80, October 27-31, 1980 pp. 552-558.

7. [Chambe74] Chamberlin, D. D., Boyce, R. F.
"SEQUEL: A Structured English Query Language".
Proc. 1974 ACM Sigfidet Workshop, Ann Arbor, Michigan, April 1974, pp.249-264.

8. [Chambe76] Chamberlin, D. D., et al.
"SEQUEL2: A Unified approach to data definition, manipulation and control".
IBM journal of research and development, 20( 6 ), Nov. 1976.pp. 560-575.

9. [Chang76] Chang, C.L.
"DEDUCE: a deductive query language for relational data bases".
Pattern Recognition and Artificial Intelligence. Academic Press (1976) pp.108-134.

10. [Chang78] Chang, C.L.
"DEDUCE2: further investigation of deduction in relational

databases". Logic and Data bases, Gallaire and Minker editors, Plenum Press (1978), pp.201-236.

11. [Chang81] Ning-San Chang and King-Sun Fu
"Picture Query Languages"
Computer, Nov. 1981, pp. 23-33

12. [Charle81] Charles Welty.
"Human Factors comparison of a procedural and a non-procedural query language".
ACM Trans. On Data Base System, vol.6 no. 4, Dec. 1981.
Pp. 626-649.

13. [Codd72] Codd, E. F.
"A Data Base Sublanguage founded on the relational calculus".
Proc. Of 1972 ACM SIGFIDET WORKSHOP on Data Description, Access and Control, pp.35-68.

14. [Codd78] E. F. Codd.
"How about recently? "
(English dialog with relational data bases using RENDEZVOUS Version 1)
Databases: Improving Usability and Responsiveness.

15. [Dawei81] Dawei luo and S. Bing Yao.
"Form Operation By Example--a language for office information processing.
ACM-SIGMOD 1981, Nat. Conf. On Management of data. Ed.Y.

Edmund Lien. Pp.213-223.

16. [Dehene76] C. Deheneffe and H. Hennebert.
"NUL: A Navigational User's Language for network structured data bases".
Proc. ACM-SIGMOD International Conf. On Management of Data, June 1976. Pp.135-142.

17. [Greenb78] Greenblatt D., Waxman J.
"A Study of Three Database Query Languages".
Databases: Improving Usability and Responsiveness. B. Shneiderman, Ed., Academic Press, New york 1978.

18. [Held75] Held, G. H. Et al.
"INGRES--A Relational data base system".
Proc. AFIPS NCC, vol. 44, May 1975, pp.409-416.

19. [Housel79] B. C. Housel
"QUEST: A high level data manipulation language for network, hierarchical, and relational database".
IBM Research Report RJ2588, July 1979.

20. [Kambay77] Kambayashi, Y. Et al.
"A Relational Data Language with Simplified binary relational handling capability".
Proc. Of 3rd Int. Conf. On VLDB, Tokyo Japan, Oct. 1977, pp.338-350.

21. [Kameny78] I. Kameny, J. Weiner, M. Crilley, J. Burger, R. Gates and D. Brill

"EUFID: The End User Friendly Interface to Data Base Management Systems".
Fourth Int. Conf. On VLDB. 1978. Pp.380-391.

22. [Kersch76] L. Kerschberg, E. A. Ozkarahan.
"A Synthetic English Query Language for a Relational Associative Processor".
2nd Int. Conf. On Software Engineering 1976 pp. 505-519.

23. [Lacroix77a] Lacroix, M., Pirotte, A.
"Domain-oriented Relational Languages".
Proc. Of 3rd Int. Conf. On VLDB, Tokyo Japan, Oct. 1977, pp.370-378.

24. [Lacroix77b] Lacroix M., Pirotte A.
"ILL: an English Structured Query Language for Relational data bases".
Proc. IFIP TC-2 working conf. On modelling in data base management systems, Nice( january 1977 ), Nijssen editor, North-Holland ( 1977 ), pp.237-260.

25. [Mcleod75] Mcleod, D. J. And M.J. Meldman
"RISS: A Generalized Minicomputer Relational Data Base Management System".
Proceedings of National Computer Conference. May 1975.

26. [Mcleod76] Dennis Mcleod
"The Translation and Compatibility of Sequel and Query By Example".

2nd Int. Conf. On Software Engineering. Oct 1976 pp. 520-526.

27. [Mylopo76] Mylopoulis, j., Borgida, A., Cohen, P., Roussopoulis, N., Tsotsos, J., Wong, H.
"TORUS: a Step Towards Bridging the Gap Between Databases abd the Casual User".
Information System (GB), vol.2, no.2, (1976), pp.49-64

28. [Mylopo80] Mylopoulos, J. Bernstein, P. A., Wong, H. K. T.
"A Language Facility for Designing Data Base Intensive Applications".
ACM Trans. Database Systems vol. 5, no. 2, pp. 185-207. June 1980.

29. [Nicola81] M. Nicolai
"SLANG: a statistical language".
Computer Bull.(GB), Ser. 2, no 27, pp.4-6 ( March 1981 ).

30. [Pirott77] Pirotte A., Wodon P.
"A Comprehensive Formal Query Language for a Relational Database : FQL"
R.A.I.R.O. Informatique/ Computer Science, vol. 11-2, pp.165-183, 1977.

31. [Pirott79] A. Pirotte
"Fundamental and Secondary Issues in the Design of non-procedural Relational language".

Proc. Fifth Int. Conf. On VLDB, ACM 1979 pp.239-250.

32. [Plath76] Plath, W. J.
"REQUEST: A Natural Language Question-Answering System".
IBM j. Res. Develp. 20 ( July 1976 ), pp.326-335.

33. [Reisne75] Reisner, P. Boyce, R.F. And Chamberlin, D.D.
"Human Factor Evaluation of two Data Base Query Languages--SQUARE and SEQUEL". AFIPS Proc. Vol. 44, 1975, pp. 447-452.

34. [Reisne79] Reisner, P.
"Use of Psychological Experimentation as an aid to development of a query language".
IEEE Trans. On Software Engin. SE-3,3 (1977) pp. 218-229.

35. [Risshdr] Riss Application Level Interface Modules
Get, Risshdr/un=kesfell

36. [Robert76] C. Robert Carlson, Robert S. Kaplan
"A Generalized Access Path Model and its Application to a Relational Database System".
1976 Int. Conf. On Management of Data, pp.143-154.

37. [Schlag82] Schlageter, G., Rieskamp, M., Pradel, U., Unland, R.
"The Network Query Language"
ACM-SIGMOD 1982, Int. Conf. On Management of Data.

38. [Shipma81] David W. Shipman
"The Functional Data Model and the Data language Daplex".
ACM Trans. On Data base System, vol. 6(1) march 1981, pp.140-173.

39. [Stoneb76] M. Stonebraker, et al.
"The Design and Implementation of Ingres".
ACM Trans. On Database Systems, vol. 1, no. 3, Sept 1976.

40. [Sue82] Sue M. Dintelman, A. Timothy Maness.
"An Implementation of a Query Language Supporting Path Expression".
Int. Conf. On management of Data , ACM Sigmod 1982. June 2-4, pp. 87-93.

41. [Su78] Su S. Y. W., Emam, A.
"CASDAL: CASSM's data language".
ACM Trans. On database Systems. Vol. 3, no. 1, 1978, pp.57-91.

42. [TAIMING80] Tai-Ming Parng and Baw-Jhiune Liu
"Extended Relational Calculus--A Formalism for High Level Relational Query Languages".
Proceed. 4 COMPSAC 80 Oct 27-31, 1980. Pp. 801-807.

43. [Theera80] Theerachetmongkol, A., Montgomery, A. Y.
"Semantic Integrity Constraints in the Query By Example Data Base Management Language".
Australia Computer Journal, vol.12, no.1, 1980 pp.28-42.

44. [Thomas75] Thomas, J. C. And Gould, J. D.
"A Psychological Study of Query By Example".
AFIPS Proceed., vol.44, 1975, pp.439-445.

45. [Todd76] S. J. Todd
"The Peterlee Relation Test Vehicle--A System overview".(PRTV)
IBM System J. No. 4( 1976 )

46. [Tsurut81] T. Tsurutani, Y. Kasaharu, M.Naniwada.
"Data Structure and Language of the Goegraphic Information System ATLAS".
NEC Res, and Develop., Japan, no. 62, pp.57-64 ( July 1981 ).

47. [Vandij77] Vandijk, E.
"Toward a more familiar relational language".
Information System, vol. 2, no.4, 1977, 159-169.

48. [Warner81] H. D. Warner, D. Odle
"A High-level Functional Query Language for a Small Relational System".( FQUERY)
SIGSMALL Newsl.(USA, vol.7, no.2, pp.90-5 Oct 1981)

49. [Wonkim82] Won kim
"On optimizing an SQL-like Nested Query"
ACM Trans. On Data base systems. Sept 1982, vol.7, no.3. Pp.443-469.

50. [Zloof75] M. M. Zloof

"Query By Example"
AFIPS PROCEEDINGS, vol. 44, 1975, pp.431-438

51. [Zloof77a] Zloof M. M.
"Query By Example: A data base language".
IBM System j., vol.16, no.14, 1977. Pp.324-343

52. [Zloof77b] Zloof M. M., De Jong S. P.
"The System for Business Automation ( SBA )".
Comm. ACM, vol.20, no.6. June 1977 pp. 385-396.

# User Manual

## I. How to initiate QBE.

1. Power up QBE system, turn on modem and load FLEX operating system.
2. Enter QBE.BIN.1 to initiate the system.
3. System will ask you to set up communication channel with CYBER.
4. Go to CYBER and execute procedure QHP on cyber.
5. A 'start up' message will be displayed and you are to enter <cr> to start QFP command interface.

Note: the display format is SPLIT2 mode with no special feature selected.

6. Cursor can now be seen in the first column of row #24 on the screen. You can enter any programmed keys which will be defined later.

## II. Query Set Up:

1. Initiate QBE as described in previous section.
2. Select proper main mode (SPLIT2 or SPLIT4) by pressing function key #5.
3. Press function key BLUE to bring in a skeleton of a relation.
4. The system will ask you to enter a relation name. Enter

the relation name if known (backspacing is allowed; maximum number of character you can input is 15). You can enter ESC if you like to terminate the command entirely.

If the relation name is not known, press function key BLUE again to display a menu of relations in the database dictionary. Cursor function keys are permitted to display the next set or previous set of the relations. The system will ask you to enter the relation name again after servicing the command.

5. Now the system will ask to enter the LRNB for that relation (Refer to QFP display format section for exact information). If the relation requested is in the database, its skeleton will be displayed very shortly, otherwise a bell will be heard and you will have to start all over, i.e. press function key BLUE again. Once you have the skeletons of the relations on the screen, you can set up a query as follows.

6. Press function key #4 followed by the LRNB of the relation in which you would like to fill in entries. Now you can see the cursor displayed in the first entry of the relation; and you are to set up the entries. To move cursor inside the relation, use space bar, backspace, line feed and up arraow keys. To access the columns that are not displayed because lack of room, use function keys '<-' and '->'.

You can set up the entries in any order you like. Terminate

entry input by a <cr>. QFP will warn you if you try to enter a character on any of the skeleton lines or outside the relation.

Print operator is small 'p' and should be used only as the first character of any entry. You can specify an example element as p^xxx where 'p' is the print operator, '^' is not echoed but inserted in the corresponding entry buffer. It affects the display such that the rest of the entry characters which you enter will be seen in reverse video mode. You can apply print operator to the entire tuple by entering a small 'p' in the column just before the first column, i.e. beneath LRNB.

7. After finish setting up the entries for a relation, enter CONTROL G. Set up all the relations you need to form a query. Note: You are not allowed to edit a relation to modify entries which you have set up.

8. Press function key #1 to transmit query and retrieve information. The answer to the query if any, will be displayed shortly or a bell is rung and command will be aborted.

Note: Before the retrieved information is displayed in a relation, entries that are currently in that relation are cleared.

If you would like to see one of the relations shown partially on the whole screen, press function key #3

followed by the corresponding LRNB.

III. Description of Programmed Keys:

Function Key #1:

Transmit the query that has been set up and retrieve information from HOST then display the answer of the query in the appropriate relation.

Function Key #2:

Terminate QHP on CYBER, and allow the H19 terminal to be used as an ordinary terminal to a mainframe.

Function Key #3:

Allow a special feature to display a relation on the whole screen such that at most possible tuples and columns can be seen.

Function Key #4:

Initiate query set up for a selected relation.

Function Key #5;

Allow switch between the two main modes (SPLIT2 and SPLIT4).

Function Key BLUE:

Display the skeleton of a selected relation or display a menu if pressed twice.

Function Key Red:

Display complete status of a selected relation. This is considered as a special feature.

Function Key GREY:

Suspend special features like SPLITØ mode or complete status mode and restore screen status.

IV., How to Perform A Quick Test:

1. Initiate QBE as described before.

2. Enter 'T'.

(To store data in the internal data structures of the relations)

3. Enter 'F' followed by '1'.

(To set the activity status byte of the specific relation).

4. Enter 'D' followed by '1'.

(To display the specific relation)

Relations will be displayed with some entries. Now you can test function key F5, F4, FRED, FGREY, F3 and all cursor function keys.

Note: You CAN NOT test FBLUE and F1.

Example Query Sessions

1. Main Mode = SPLIT2, LRNB used = 1.

Print Subassembly such as 'car' if assembly = train and quantity > 3.

( 1 ) Materil

| Assembly | Subassemb | Quantity |
|---|---|---|
| train | p^car | > 3 |
| | | |

2. Main Mode = SPLIT2, LRNB used = 2.

Print Labcost if Assembly = car or caboose.

( 2 ) Labcost

| Assembly | Labcost |
|---|---|
| car | p^xxx |
| caboose | p^yyy |
| | |

3. Main Mode = SPLIT2, LRNB used = 1.

Print Subassembly if its associated assembly is the same as assembly of 'truck'.

( 1 ) 'Materil

| Assembly | Subassemb | Quantity |
|---|---|---|
| p^xxx | truck | |
| p^xxx | p^car | |
| | | |

4. Main Mode = SPLIT4. LRNB used are 1 and 3.

What is the price of an assembly which subassembly is a truck.

( The common example elements will not be printed out in the answer )

( 1 ) Materil

| Assembly | Subassemb | Quantity |
|---|---|---|
| p^xxx | truck | |
| | | |

( 3 ) Price

| Assembly | Price |
|---|---|
| p^xxx | p^zz |
| | |

5. Main Mode = SPLIT2, LRNB = 2.

Print the sum of quantities, the average quantity, the maximum quantity and the minimum quantity of materil.

( 1 ) Materil

| Assembly | Subassemb | Quantity |
|---|---|---|
| | | psum |
| | | pavg |
| | | pmax |
| | | pmin |
| | | |

Sample Database

Materil

| Assembly | Subassemb | Quantity |
|---|---|---|
| train | truck | 1 |
| train | car | 3 |
| train | caboose | 1 |
| truck | body | 1 |
| truck | big | 2 |
| body | cab | 1 |
| plane | body | 1 |
| car | big | 2 |
| car | truck | 2 |
| body | big | 4 |
| truck | caboose | 3 |
| cab | caboose | 5 |
| car | body | 4 |
| car | cab | 34 |

Price

| Assembly | Fgprice |
|---|---|
| truck | 11.99 |
| car | 6.45 |
| caboose | 6.75 |
| train | 46.95 |

Labcost

| Assembly | Labcost |
|---|---|
| car | 92.00 |
| caboose | 64.00 |
| base | 99.00 |
| body | 98.00 |
| truck | 98.00 |
| train | 89.00 |