CUPID: A GRAPHICS ORIENTED FACILITY FOR SUPPORT OF NON-PROGRAMMER INTERACTIONS WITH A DATA BASE

by

Nancy Harriet McDonald

Memorandum No. ERL-M563

12 November 1975

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CUPID: A Graphics Oriented Facility for Support of Non-Programmer Interactions with a Data Base

Nancy Harriet McDonald

Abstract

CUPID (Casual User Pictorial Interface Design) is a facility designed to support non-programmer interactions with a data base system. It is a front-end user interface for the relational data base system, INGRES, and compiles "pictures" into the query language, QUEL supported by INGRES.

The thesis describes CUPID's data sublanguage and its definition capability. The data sublanguage is complete, high level and picture oriented, depending almost entirely on "menu-move" operations on a CRT terminal. Hence, users have no need to type extensive English text and difficult natural language processing can be completely avoided. The definition feature serves two purposes:

1. to resolve ambiguous interactions, and
2. to allow the user to define individual terms for local and global purposes.

Since a major goal of this work is to analyze the feasibility of a picture query language, this thesis details the working implementation and describes design for extension of the system. In conjunction with the prototype implementation, a human factors experiment was designed and conducted to compare the CUPID system to the more formal language, QUEL. This experiment is described and the results discussed. Subjects learned both systems. Statistics were tabulated on their performances and preferences. The results, presented here, indicate that CUPID is a viable casual user system.

Signature____________________________
Chairman of Committee

ACKNOWLEDGMENTS

I am pleased to express my deepest appreciation to Professor M. Stonebraker for his efforts in guiding my research and in editing this thesis. I am also indebted to Professor E. Wong for his helpful and supportive comments and to Professor D. Foley for his time spent and interest in this project. I'd like to give special thanks to Professor P. Varaiya for introducing me to the data base project and for the confidence he expressed in me.

During the implementation of this work, Dr. H. Holmes and other members of the Lawrence Berkeley Laboratory were very patient and helpful consultants. And special gratitude is due my fellow students and staff members on the INGRES project. Their comments, assistance and camaraderie were most helpful and encouraging.

I am, as always, deeply grateful to my husband, Gene and our daughter, Pamela, for their loving support and understanding.

I would also like to thank IBM and the Department of EECS at the University of California for the financial support given this effort. Research was also supported by the National Science Foundation Grant DCR75-03839.

TABLE OF CONTENTS

CHAPTER 1

Introduction

The main goal of this research is to explore the feasibility of the picture query language system, CUPID (Casual User Pictorial Interface Design). CUPID is a facility designed to support non-programmer interactions with a data base management system. CUPID contains a picture oriented data sublanguage and a defintion capability. The language is complete [CODD72], high level and depends almost entirely on "menu-move" operations on a CRT terminal. Very little English text is typed by the user, thus avoiding difficult natural language processing by the system. The definition facility helps to resolve ambiguous operations and to allow the user to define individual terms.

A prime concern is for the response of a casual user to such an interface. To this end, CUPID was

1) designed to have a simplistic, yet flexible, syntax for ease of use

2) designed to contain a user definition capability for ease of expression

3) implemented in part to present some of the major features for actual performance and preference evaluation

4) compared to another query language system in a human

factors test

This chapter provides some background and motivation for the CUPID project. Since the data base system, INGRES, and the data sublanguage, QUEL, are important components of and incentives for CUPID, a brief description of INGRES and QUEL is presented in section 1.1 to help familiarize the reader with some terminology and the underlying systems. Sections 1.2 and 1.3 present the motivation for CUPID. In section 1.2, the value of the human-machine interface is discussed along with an alternative approach (natural language). Section 1.3 lists the benefits expected from a pictorial approach to the human-machine interface problem. Finally, section 1.4 provides an overview of the remainder of this dissertation.

1.1 Background- INGRES and QUEL

INGRES (Interactive Graphics and REtrieval System)[MCDO74,HELD75a,HELD75,STON75a,WONG75,ZOOK75] is a relational data base system which is implemented on a PDP-11/40 based hardware configuration at Berkeley. INGRES runs as a normal user job on top of the UNIX operating system developed at Bell Telephone Laboratories [RITC74a]. The implementation of INGRES is primarily programmed in "C" [RITC74], a high level language in which UNIX itself is written. Parsing is done with the assistance of YACC

[JOHN74]}, a compiler-compiler available on UNIX.

The advantages of a relational model for data base management systems have been eloquently detailed in the literature, [CODD70,CODD74,DATE74,DATE75] and hardly require further elaboration. The choice of the relational model was particularly influenced by (a) the high degree of data independence that such a model affords, and (b) the possibility of providing a high level and entirely procedure free facility for data definition, retrieval, update, access control, support of views, and integrity verification.

INGRES runs as three processes which communicate via the UNIX interprocess communication facility, together with a fourth "front end" process. One of these front ends is an interactive monitor which allows the user to formulate, edit, print and execute interactions in the data sublanguage, QUEL.

QUEL (QUEry Language) has points in common with Data Language/ALPHA [CODD71], SQUARE [BOYC73] and SEQUEL [CHAM74] in that it is a complete [CODD72] query language which frees the programmer from concern for how data structures are implemented and what algorithms are operating on stored data. As such it facilitates a considerable degree of data independence [STON74].

The QUEL examples in this section all concern the following relation.

	NAME	DEPT	SALARY	MANAGER
	Smith	toy	10000	Jones
EMPL	Jones	toy	15000	Johnson
	Adams	candy	12000	Baker
	Johnson	toy	14000	Harding
	Baker	admin	20000	Harding
	Harding	admin	40000	none

Indicated here is an EMPL relation with domains NAME, DEPT, SALARY, and MANAGER. Each employee has a manager (except for Harding who is presumably the company president), a salary, and is in a department.

A QUEL interaction includes at least one RANGE statement of the form:

```
RANGE OF variable-list IS relation-name
```

The symbols declared in the range statement are variables which will be used as arguments for tuples. These are called TUPLE VARIABLES. The purpose of this statement is to specify the relation over which each variable ranges.

Moreover, an interaction includes one or more statements of the form:

```
Command Result-name ( Target-list )
          WHERE Qualification
```

Here, Commmand is either RETRIEVE, APPEND, REPLACE, or

DELETE. For RETRIEVE and APPEND, Result-name is the name of the relation which qualifying tuples will be retrieved into or appended to. For REPLACE and DELETE, Result-name is the name of a tuple variable which, through the qualification, identifies tuples to be modified or deleted. The Target-list is a list of the form

```
Result-domain = Function ...
```

Here, the Result-domain's are domain names in the result relation which are to be assigned the value of the corresponding function.

The following suggest valid QUEL interactions. A complete description of the language is presented in [HELD75a].

Example A   Find the manager of employee Jones.

```
RANGE OF E IS EMPL
RETRIEVE INTO W (MGR = E.MANAGER)
WHERE E.NAME = 'Jones'
```

Here, E is a tuple variable which ranges over the EMPL relation and all tuples in that relation are found which satisfy the qualification E.NAME = 'Jones'. The result of the query is a new relation, W, which has a single domain, MANAGER, that has an entry for each qualifying tuple. If the result relation is omitted, qualifying tuples are printed on the user's terminal or returned to the calling program. Also, in the Target-list, the 'Result-domain =' (i.e MGR =) may be omitted if Function is of the form; Variable.Attribute (i.e. NAME = E.NAME may be written as E.NAME).

Example B   Insert the tuple (Jackson,candy,13000,Baker) into EMPL.

```
        APPEND TO EMPL(NAME = 'Jackson', DEPT = 'candy',
                SALARY = 13000, MGR = 'Baker')
```

Here, the result relation EMPL is modified by adding the indicated tuple to the relation. If less than all domains are specified, the remainder default to zero for numeric fields and null for character strings.

Example C   Delete the information about employee Jackson.

```
        RANGE OF E IS EMPL
        DELETE E WHERE E.NAME = 'Jackson'
```

Here, the tuples corresponding to all employees named Jackson are deleted from EMPL.

Also, QUEL contains aggregation operators including COUNT, SUM, MAX, MIN, and AVG. Two examples of the use of aggregation follow.

Example D   Replace the salary of all toy department employees by the average toy department salary.

```
        RANGE OF E IS EMPL
        REPLACE  E(SALARY  BY  AVG(E.SALARY  WHERE  E.DEPT =
'toy') )
                WHERE E.DEPT = 'toy'
```

Here, the AVG is to be taken of the salary attribute for those tuples satisfying the qualification E.DEPT = 'toy'. Note that AVG(E.SALARY WHERE E.DEPT= 'toy') is scalar valued and consequently will be called an AGGREGATE. More general aggregations are possible as suggested by the following example.

Example E Find those departments whose average salary exceeds the company wide average salary, both averages to be taken only for those employees whose salary exceeds $10000.

```
        RANGE OF E IS EMPL
        RETRIEVE INTO HIGHPAY(E.DEPT)
        WHERE   AVG(E.SALARY  BY  E.DEPT  WHERE  E.SALARY >
10000)
                >
                AVG(E.SALARY WHERE E.SALARY > 10000)
```

Here, AVG(E.SALARY BY E.DEPT WHERE E.SALARY>10000) is an AGGREGATE FUNCTION and takes a value for each value of E.DEPT. This value is the aggregate AVG(E.SALARY WHERE E.SALARY>10000 AND E.DEPT = value). The qualification expression for the statement is then true for departments for which this aggregate function exceeds the aggregate AVG(E.SALARY WHERE E.SALARY>10000).

For a complete description of the currently operational INGRES commands, the reader is referred to [ZOOK75].

Initial user reaction to the early INGRES system indicated that QUEL was not particularly user-friendly. Users who were not familiar with computer programming had difficulty adjusting to QUEL´s formalism. The next section provides some thoughts on this problem.

1.2 The Importance of the Human Interface

If the computer is to become an everyday tool of the nonprofessional, the needs and desires of the casual user must be considered. In the past "...narrow technical considerations and immediate cost constraints dominated computer technology..., in large part at the expense of human ease, convenience and social effectiveness" [MART73]. It is generally agreed that the cost of instruction-execution (hardware) is decreasing while the cost of programming (software labor) is increasing. This strongly suggests more effort should be expended in reducing the human labor needed to interact with a computer.

Information retrieval and data base management are areas which might benefit from easy human-machine interactions. This author feels there are two approaches to facilitate such interactions.

1 providing the casual user with an English-like

dialog capability that uses artificial intelligence methods for natural language processing

2. providing the user with a picture oriented graphics language

The difficulties of the first approach are well summarized by one of its proponents, E. F. Codd: "It is very unlikely that any two English-speaking persons understand precisely the same English" [CODD74a]. Since English is well known to be a non-finite state grammar, users have the capacity for generating infinite sets of well and ill-formed utterances. This makes the processing of arbitrary English textual input extremely difficult. Some specific problems are:

a. typographic and spelling mistakes may be present
b. English is a language full of ambiguity
c. a large vocabulary is involved
d. syntactic analysis is difficult
e. semantic analysis is very difficult

The next section provides the motivation for a graphic language.

1.3 Why a Picture Language?

Some of the reasons a pictorial representation of a query is more desirable than a linear representation such as English include:

1 It is speculated that users will be more successful at phrasing interactions in a picture language. A great deal of information can often be more precisely, accurately and clearly stated in two dimensions than in one.

2 Restricting the representation to a specific diagram instead of allowing any number of words and phrases will aid in making the picture language a finite state grammar. This will eliminate many of the anomalies and complexities of the non-finite state grammar, English.

3 Punctuation errors and their resultant ambiguities will be of minimal concern. CUPID´s syntax is simple and precise enough to need no punctuation.

4 Fewer typographical errors will occur because only a very small portion of any query expressed by pictures need be entered at the keyboard (all other entries can be made via lightpen on a display device in ´menu-move´ mode). This should be contrasted to English in which virtually everything is typed on a keyboard.

5 The non-procedural and unordered manner of phrasing should appeal to users who have no previous computer experience. Generally, natural English requires a procedural and ordered expression due to its linear format.

6 Due to the addition of the preceeding five points and the omission of points a - e of section 1.2, this author

expects a significant reduction of the implementation effort over the effort necessary to implement a natural language.

There has been very little work in the area of graphic languages. The graphics version of CSMP (Continuous System Modelling Program) [WALS71] is an example of graphic modelling applied to simulation problems. The only effort, to this author's knowledge, in this direction in a data base context is the work of Zloof [ZLOO75]. However, as with natural language processing, such difficulties as:

1. complex syntax allowing arbitrary amounts of information within a 'box'
2. unlimited text strings
3. typographic and spelling error possibilities

present problems for the implementation of his proposal. The system described here has points in common with that in [ZLOO75] without the above drawbacks.

1.4 Qverview

The rest of this thesis is organized as follows. Chapters 2 and 3 describe the language, CUPID. Chapter 2 presents the syntax through annotated examples of retrieval and update operations. The third chapter presents the design of a definition capability. This aspect of CUPID was devised to provide user freedom in expressing and system learning in understanding user defined terms. Algorithms for

(1)resolution of ambiguities and (2) defining new terms are detailed. Chapter 4 discusses various implementation considerations and human factors design. The fifth chapter describes the design and results of an experiment performed to compare CUPID and QUEL from the user's point of view. Finally, the sixth chapter provides a summary and conclusion along with some suggestions for further work in this area.

CHAPTER 2

Syntax

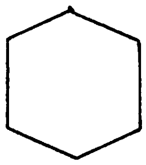
The goals of the CUPID syntax are:

1 simplicity
2 naturalness
3 ease of use
4 equivalency to QUEL

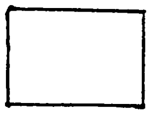
In order to meet these goals, various picture representations were considered. The visual interpretation illustrated in this chapter is but one possibility. Due to the powerful graphics modelling system, PICASSO [HOLM75], on which the picture processing portion of CUPID resides, one can experiment easily with other possible picture representations.

2.1 Symbols

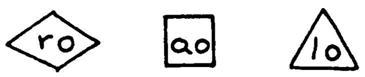
The major components of a picture query in the representation chosen for the CUPID prototype are:



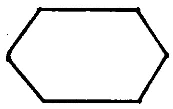
hexagon containing relation name



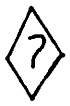
rectangle containing domain name



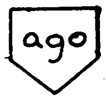
various shaped forms containing relational(ro), arithmetic(ao), and logical(lo) operators



horizontal hexagon within which constants are entered



upright diamond containing "?"



pentagon containing aggregate(ago) operator

connecting line

A formal syntax of CUPID is provided in Appendix A. A brief outline of the basic format follows.

2.2 General Format

Because CUPID is designed to be equivalent to QUEL, the notions of "target list" and "qualification" are maintained. The general format of a CUPID query is a diagram.

I Relations and domains are represented by vertical hexagons (HEX) and rectangular boxes (BOX) respectively. Each contains the desired name within it. A HEX is abutted to the left of one or more connected BOXes. (all other symbols are attached via lines). The juxtaposition of the HEX and the BOXes forms the basic CUPID unit that defines which relations and domains are involved in a query.

II Any item or expression with a vertical diamond containing a "?" (QBOX) attached is taken to be part of a target list (i.e. those item(s) being targetted by the query).

III Portions of the diagram unattached to QBOXes are qualifications of the target list.

IV Horizontal hexagons (CONS) are reserved for typing in constant values.

V Operators are connected to their respective operands (see Appendix D for ordering constraints). Arithmetic and aggregate operators may be linked to other operators. This indicates that the result of the arithmetic or aggregate operation is one operand for the next operator. Aggregate operators have their own targets which may or may not be qualified.

Examples of this format are provided in the following sections. The pictures are photographs of actual CUPID drawings as they appear on the GT42 screen. The eight pointed star-like image in each is known as a "tracking cross". This cross is used in conjunction with the lightpen to select and place various elements of the picture (see Appendix E for further details).

2.3 Retrieval Examples

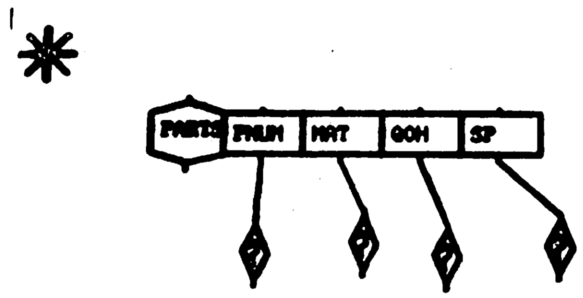
The following examples illustrate the retrieval mode of the language in a sequence of increasingly complex queries. Updates will be described in the next section. Please note the format of the examples emphasizes the difference between English language queries and their CUPID equivalents as they appear on the screen.

The data base referenced in all examples is a sample warehouse system. PARTS(PARTS) are incoming from SUPPliers(SUPP) and go out to customers via ORDERs. The relations involved are:

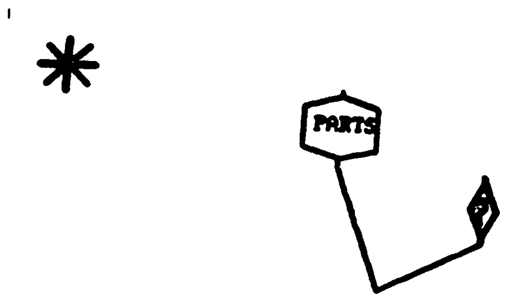
```
PARTS(PNUM, MAT, QOH, SP)
SUPP(SNUM, SLOC)
ORDER(CNUM, PNUM, CLOC, QUAN)
PRICE(SNUM, PNUM, BP)
```

The PARTS relation contains part numbers (PNUM), material (MAT), quantity on hand (QOH) and the selling price (SP) of each part. The SUPP relation lists the supplier number (SNUM) and the supplier location (SLOC). The domains of the ORDER relation include: customer number (CNUM), part number (PNUM), customer location (CLOC), and the quantity (QUAN). Finally, the PRICE relation records the supplier number (SNUM), part number (PNUM) and buying price (BP).

1. List the entire PARTS relation.



-or-



Note:

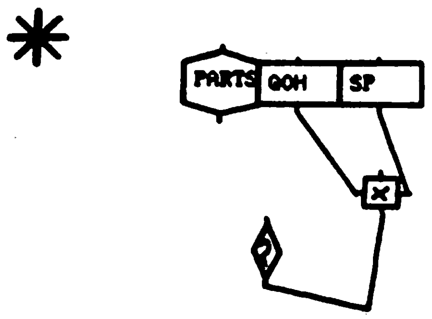
1 Words like "entire" can present problems in natural language processing.

2 The special character "?" indicates what the user wishes to see (i.e. the target list).

3 The ordering of the domains is immaterial.

4 The sequence of steps the user takes in formulating the query is immaterial.

2. Find the total value (SELLING PRICE times QUANTITY-ON-HAND) for each part.

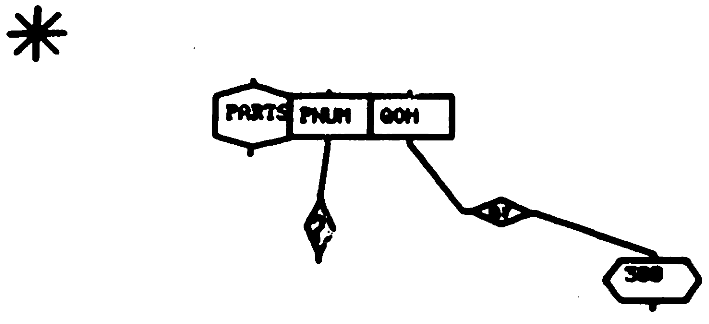


Note:

1 The English version is ambiguous without the parenthetic remark.

2 Only those domains referenced need be in the picture.

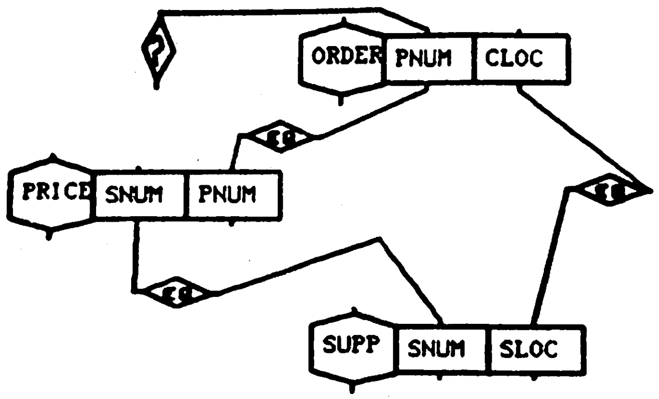
3. Get all PART#s whose QUANTITY-ON-HAND is greater than 300.



Note:

The addition of a qualification to the query allows the only keyboard entry -- the constant (in the horizontal hexagon).

4. Retrieve the PART#s whose destination and origin are the same city.

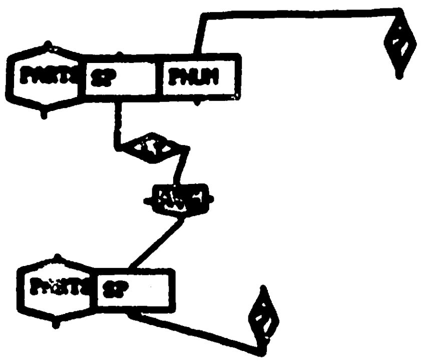


*

Note:

The unmentioned intermediate relation may make processing the English language version difficult. On the other hand, should the CUPID picture be improperly drawn due to such an omission, straightforward algorithms (presented in Chapter 3) can correct the problem.

5. List PART#s whose SELLING PRICE is less than the average SELLING PRICE.



*

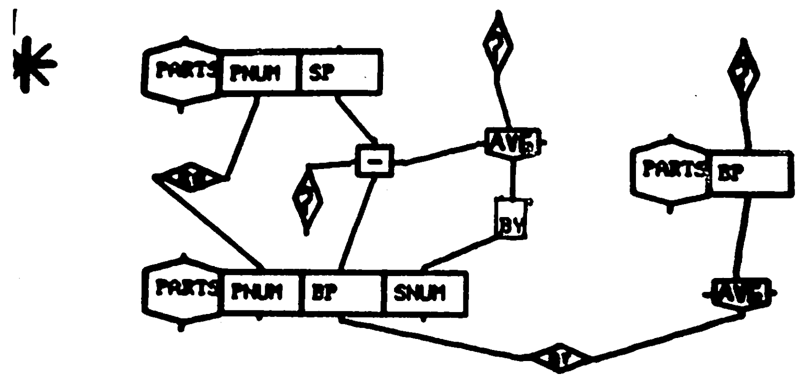
Note:

1 Implied here is the cross-product of the PARTS relation with itself.

2 The SP domain with the "?" attached is the target of

the aggregate while the PNUM domain with the attached "?" is the target of the entire query. This aggregate operator (AVG) has no qualification; but the target-of-the-query's qualification is "SELLING PRICE less than the average SELLING PRICE".

6. Find the average markup for each SUPPlier for those PART#s whose BUYING PRICE exceeds the average BUYING PRICE.



Note:

1 This last query involves an aggregate function.

2 Query 6 demonstrates the nesting of aggregation. The "average BUYING PRICE" is within the qualification of the AVG denoting the "average markup".

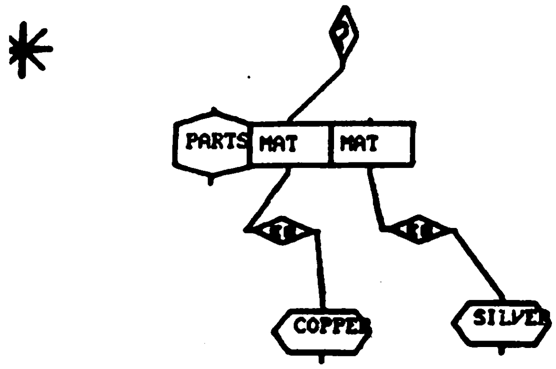
3 The aggregate (AVG) with the attached "?" is the target of the entire query. The "?" on the minus operator indicates that the subtraction operation is this AVG's target. The "by" linked to the SNUM denotes the "by clause" (as in QUEL). Everything else is considered qualification.

2.4 Update Examples

Updates involve only a simple modification of the retrieval operations and will be described in this section. The design for implementing update commands is detailed in Chapter 4. Initially, the Query Form phase assumes a retrieval mode. The user must select a separate command, UPDATE, to enter the update mode. The update mode contains three additional commands, REPLACE, APPEND and DELETE. Here we assume that the appropriate commands, REPLACE, APPEND and DELETE are available at the right of the screen.

The following three examples show the picture representation of a REPLACE, APPEND, and DELETE update.

1 Replace silver by copper wherever is occurs in the MATERIAL column of the PARTS relation.



Note:

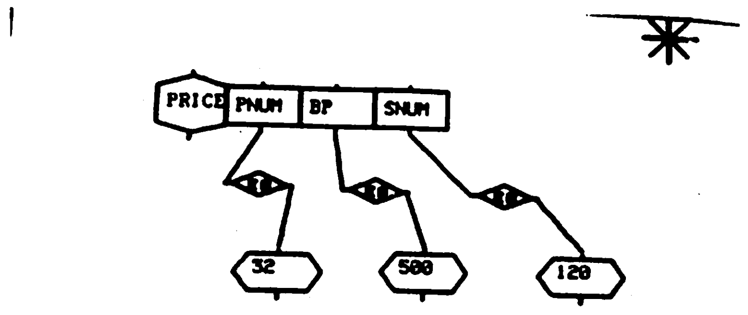
1 Targetted items are replaced by the attached constant.

2 Column boxes and names must be repeated when the qualification involves the targeted domain.

3 The replacing-value can be any expression.

4 This is a legal retrieve statement. The command - REPLACE in the update mode makes it a replace statement.

2 Append a new supplier to the PRICE table with the following information: SUPPLIER#=120, PART#= 32, BUYING PRICE= 500.

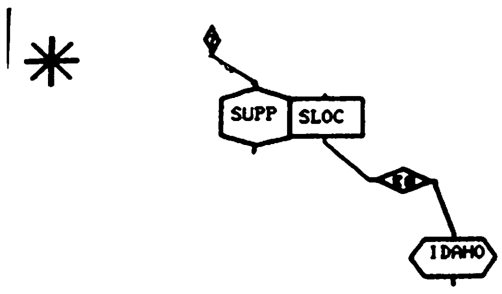


Note:

1 Appended values may be constants or expressions.

2 More than one tuple may be appended in a single query by attaching constant boxes directly to the constant boxes now present.

3 Remove suppliers located in Idaho from the SUPPlier table.



Note:

1 Since DELETE removes entire tuples, only a qualification expression in needed.

2 This simple delete picture resembles an append update in format. The command selected will differentiate. In general, however, the delete picture may involve other operators and more complex expressions.

The advantages of this syntax over either a natural language system or a more formal language system are now summarized.

1 no unbounded text strings (as in an English statement)

2 only constants are allowed to be entered from the keyboard thus eximinating most

a. typographic errors

b. spelling errors

3 easy parsing due to simple structure

4 powerful (designed to be equivalent to QUEL)

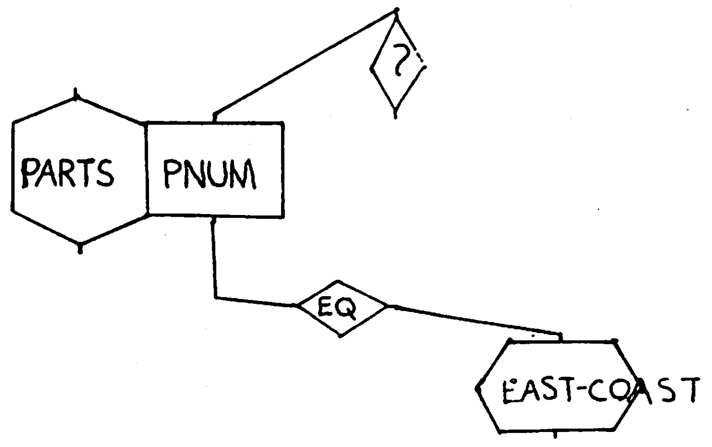
5 user-friendly with menu-move operations

CHAPTER 3

Definition Capability

A definition capability is a mechanism which allows the system to "learn " new concepts, i.e. demonstrate semantic learning. Previous attempts to devise a general semantic learner involved very large programs to recognize a small subset of English within a small universe of discourse [COLE69,QUIL66,WINO71,WOOD66]. In CUPID, the area of user definition (and, thus the amount of semantic learning necessary) is very restricted. Therefore, this system is not only pragmatically viable, but also moderate in size.

To illustrate this facility of "learning" we utilize an example. Suppose the user draws the following picture



asking for a listing of "east-coast parts". This query presents two problems:

1 Defining the unknown term "east-coast" .

2 Resolving the misplacement of the constant "east-coast" since it is not a known value for PNUM.

The following algorithms are applied to all pictures to resolve both problems.

3.1 Defining algorithm

Algorithm I. For each constant do:

1) If constant is numeric, go to 3a.

2) If constant is non-numeric, then check User Definition Table (see 3.3 Definition Representation section) for previously defined constants and present user with all established definitions of the term. If one is acceptable to user, use it and exit; otherwise:

3) Ask user to select a "type" (see Comment A) for the constant from all known domain types and compare selected type with type of the domain to which the new term is connected (i.e. by a network composed only of connectors and operators).

Comment A: The set of all domains is categorized at the time the relation is created into comparable subsets by the classification referred to here as "type". A comparability matrix of all domains in a given data base is generated for easy domain comparisons.

East-coast Example: Assuming the use of the data base described in section 2.3, the following provides type information:

Relation	Domain	Type
PARTS	PNUM	alphanum(P#)
	MAT	alphabetic
	QOH	numeric
	SP	money
SUPP	SNUM	alphanum(S#)
	SLOC	geographic
ORDER	CNUM	alphanum(C#)
	PNUM	alphanum(P#)
	CLOC	geographic
	QUAN	numeric
PRICE	SNUM	alphanum(S#)
	PNUM	alphanum(P#)
	BP	money

The comparability matrix, in which ones indicate domains of comparable type, would be:

	PNUM	MAT	QOH	SP	SNUM	SLOC	CNUM	PNUM	CLOC	QUAN	SNUM	PNUM	BP
PNUM	1							1				1	
MAT		1											
QOH			1							1			
SP				1									1
SNUM					1						1		
SLOC						1			1				
CNUM							1						
PNUM	1							1				1	
CLOC						1			1				
QUAN			1							1			
SNUM					1						1		
PNUM	1							1				1	
BP				1									1

When the system asks for a type in the case of "east-coast", assume the user chooses - geographic.

3a) If the types are comparable, perform a RETRIEVE (or UPDATE) operation to see if the value exists for the given domain.

1. Value exists, exit.

2. Value is not found -- tell user and offer a list of comparable domains and ask if constant is a value for some domain in this list or allow the user to exit from the Define phase (indicating the null result was accepted as the answer).

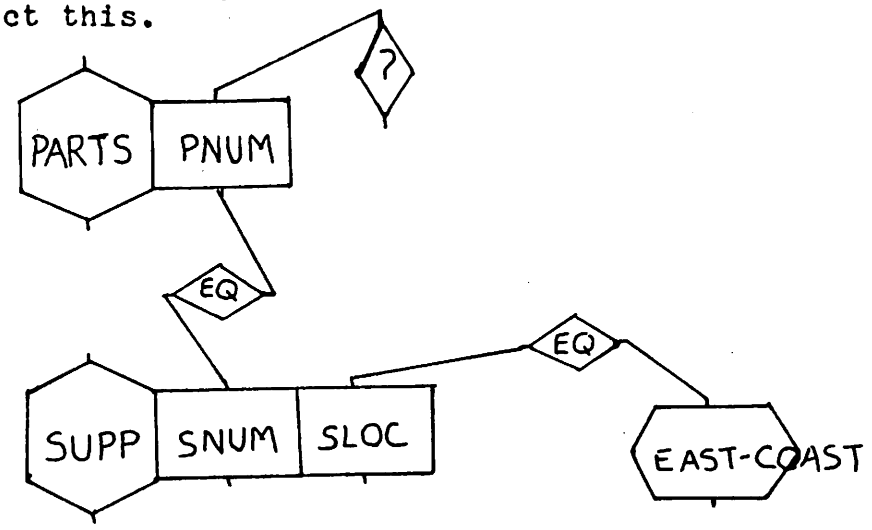
    1) If the user selects another domain, redraw the figure with new domain and repeat the algorithm.

    2) If user refuses a new domain, utilize a domain type defining procedure (see Comment B) to define the new term, then exit.

Comment B: A defining routine exists for each unique domain type (this routine must be provided whenever a new type is declared). Since "east-coast" is declared to be of type "geographic", a map of the U. S. might be displayed and the user asked to circumscribe the area meaning "east-coast". Some numeric typed domains might require the user to specify an interval out of the domain range. The last resort is enumeration, or tuple selection. The enumeration routine would present the user with the values, tuple by tuple, out of the attached domain. The user would indicate which values fit the unique definition.

3b) If types are not comparable, list all comparable domains and ask user to select the correct one. Draw a new picture and reexecute algorithm.

East-coast Example: In this case only the domains SLOC from the SUPP relation and CLOC from the ORDER relation are of type "geographic". Suppose the user really wanted to see the PNUM of parts from "east-coast" suppliers. In this case, he might respond to step 3b) by indicating SLOC. If so, the system will redraw the picture to reflect this.



This concludes the Definition Algorithm. At this point all constants are correctly defined. The Comparability Algorithm is then invoked to check that connecting (i.e. connected by a network composed of connectors and operators only) domains are of the same type.

3.2 Comparability Algorithm

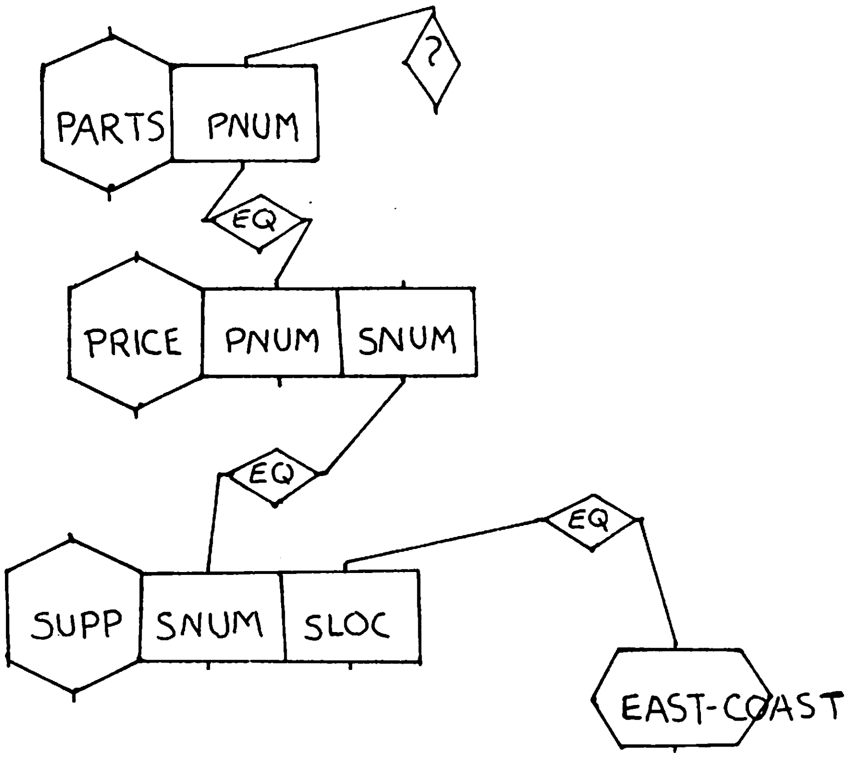
Algorithm II: For any two domains connected by a relational or arithmetic operator, check if they are of the same type. If so, exit; otherwise:

Beginning at either end of the connector:

1) List the domain type of the domain and all comparable domains. Ask user to select the appropriate domain from the list.

2a) If the user selects a comparable domain, redraw the connection and reexecute the algorithm.

East-coast Example: After two iterations of 1) and 2a) (through the PRICE relation) the user should arrive at the following picture:



This expresses the query - to find the part numbers of parts supplied by east-coast suppliers.

2b) If the user insists on a non-comparable domain, process the query allowing the normal software conversion to take place where possible. A system error may occur.

After Algorithms I and II, the query is processed. Expressions with illegal syntax are caught by QUEL parsing and error messages are returned to the user.

3.3 Definition Representation

While many definitions are peculiar to their respective definer (user), some global learning can take place in an attempt to provide efficiency for commonly used terms. For this reason, a definition table exists containing the new term, the user's identification, the domain and relation with which the new term is initially associated, the type, and a pointer to where the definition is stored (Fig. 1).

In Fig. 1 the Type of Definition domain contains the following information. The "0" indicates that a numeric interval defining routine was used for the term "big". The "1" is keyed to the enumeration routine used to define "peculiar". The four tuples from the EMP relation with 1's in the PECULIAR domain are the four NAMES deemed peculiar by user 27. The "2" in the Type of Def. domain indicates a geographic city selection routine was used for the term "east-coast"; while the other "east-coast" term was defined by an area circumscription routine as indicated by "3".

If enumeration was the method of definition, the definition may be stored as a relation with two domains:

1 a duplicate of the associated domain from the original relation

2 a binary valued domain to indicate which tuples of 1 fit the definition

For those definitions not accomplished through enumeration, the meaning is stored as one or more program routines - in a manner not unlike Winograd's semantic learner [WINO71].

Certainly, this is a form of learning from the past. When a new user asks for "east-coast", the system can flash the picture so defined and ask if that is what is meant. If it is, the established definition is used -- if not, a new definition of "east-coast" will be developed and added to the table.

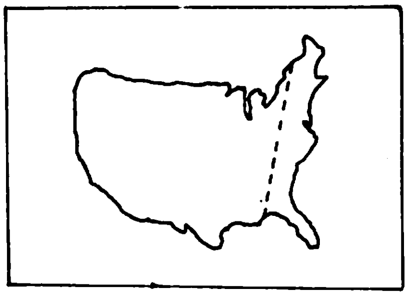
Fig. 1 Definition Representation

User Definition Table

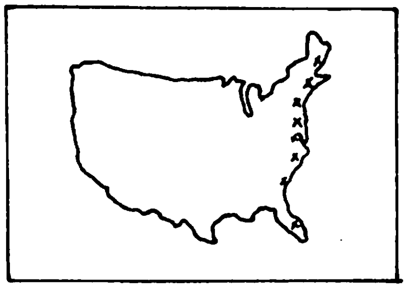
new term	id	type	connected domain/rel.	def. type	stored
eastcoast	1	geo	SLOC/SUPP	2	
big	43	num	POP/CEN	0	
eastcoast	3	geo	CLOC/ORDER	3	
peculiar	27	alph	NAME/EMPL	1	

Peculiar27 Relation

Name	Peculiar
Jones	0
Smith	1
Brown	0
Black	1
George	1
Farquar	1



1000000 ; 9000000



CHAPTER 4

Implementation

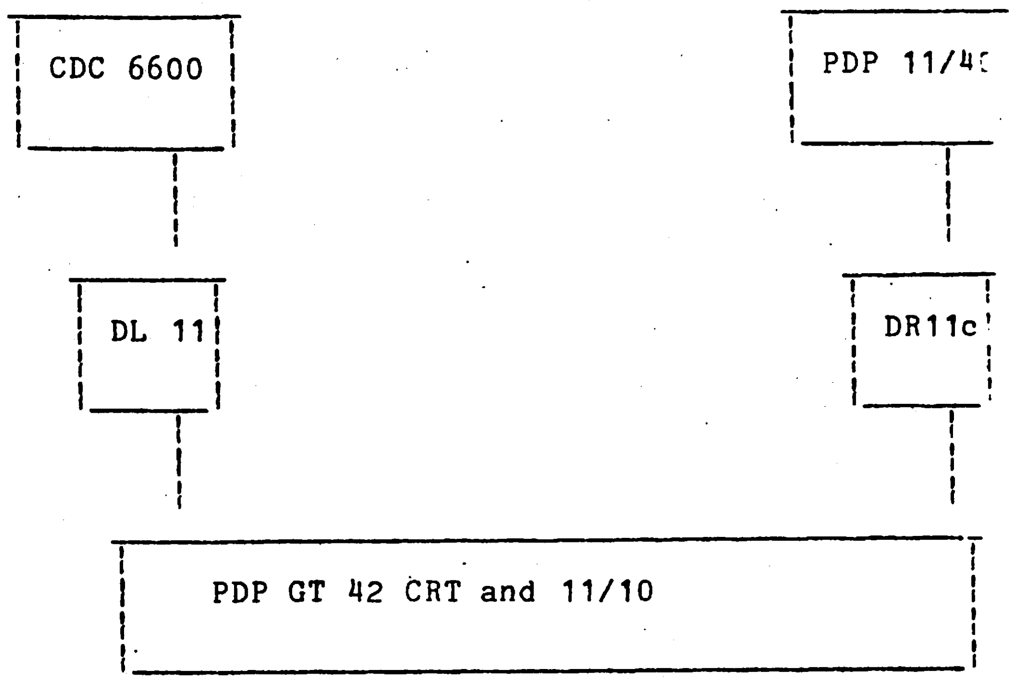
Despite hardware complications, great care was taken tc implement CUPID as designed.

A major problem arose out of the decisions to use PICASSO on the CDC 6000 computer for picture processing and use INGRES and UNIX provided facilities on the PDP 11/40 machine for data base and language processing. In order to maintain the apparent simplicity necessary to a casual user interface, the complex communications between the two machines had to be made automatic and totally transparent to the user. Once logged onto both machines (see Appendix H), it is desirable that the user merely diagram the query, observe the answer, and decide whether to issue another query. The user should not be bothered by the intricasies of the machine transfers. To this end, a unique hardware configuration was assembled together with a somewhat specialized software package.

4.1 Hardware

The computer configuration for CUPID (shown in Fig. 2) consists of a CDC 6000 series computer linked to a PDP-11/40 computer through a DEC GT-42 graphics device and its dedicated PDP-11/10 computer. Interactions (query drawin and editing) between the GT42 (PDP 11/10) and the CD machine take place at 9600 baud over a DEC DL11 interface. Information (the output of the CUPID/PICASSO system) from the CDC machine to the PDP 11/40 first passes over the DL1 interface into the PDP 11/10 where a resident monito directs the appropriate information to the PDP 11/40 over a DEC DR11-c interface. The data base response returns to the screen via the DR11-c.

Fig. 2 Hardware



4.2 Software

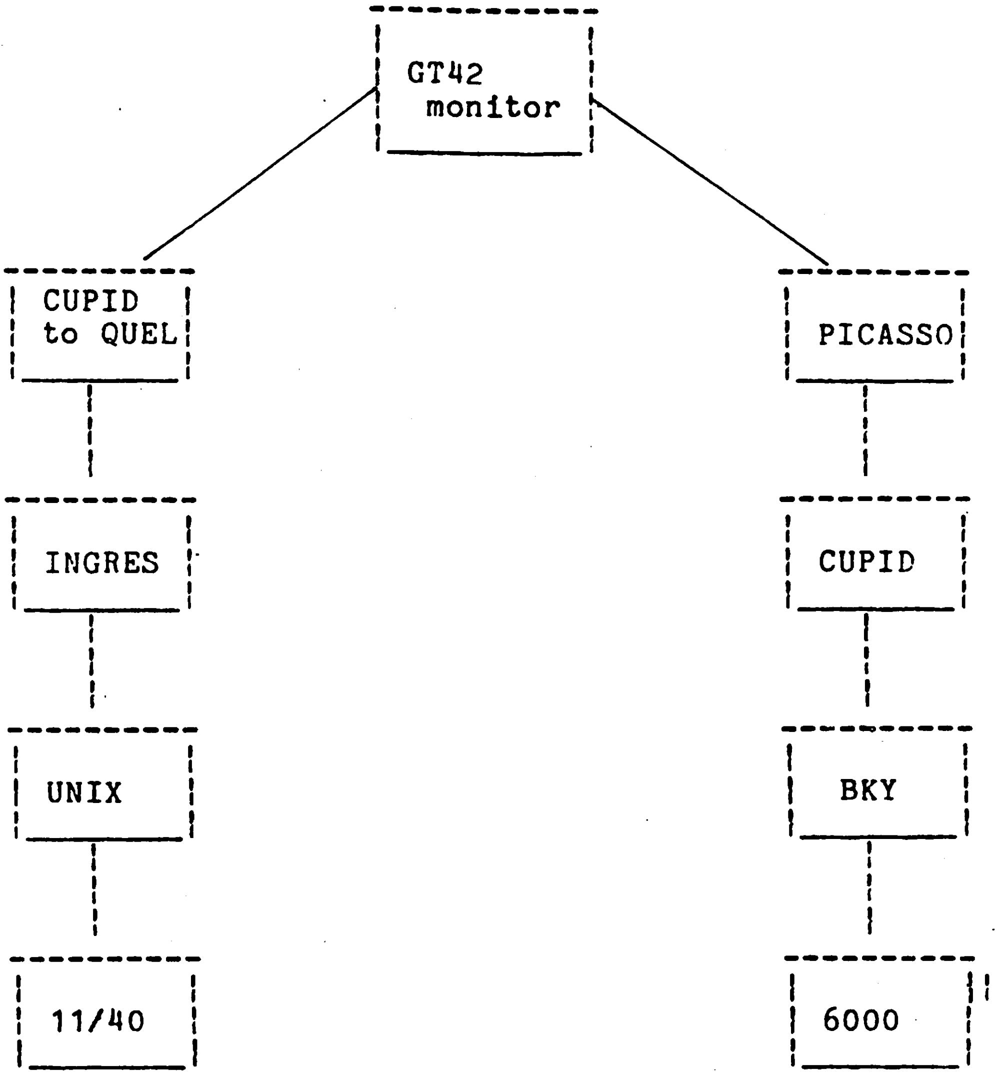
A great deal of thought was given to translating PICASSO into the language "C" to run on the PDP-11. Due to PICASSO's elaborate data structures and heavy machine dependencies, this task would have been too time consuming and tedious for this study. Thus, the decision to use PICASSO on the CDC machine.

The picture drawing and processing routines of CUPID are written in Fortran for the BKY[LAWR74] operating system of the CDC 6000. These Fortran routines are initiated from PICASSO. In order to make use of PICASSO's data structures, file maintenance routines and library facilities, CUPID was implemented as part of the PICASSO system. CUPID's initial routine, which directs the flow of control, is called as a subroutine by PICASSO. Once invoked, CUPID directs the operation, making calls to appropriate PICASSO subroutines for assistance. PICASSO routines provide CUPID with an interactive line drawing capability and analysis facility which outputs a text string corresponding to the picture. CUPID routines manipulate this text string to perform some syntax checking and partial parsing before passing the string on to the PDP 11.

The GT-42 monitor, a PDP-11 assembly language program, displays the pictures and text necessary to formulate and execute a CUPID query. Once this is completed, the text output of PICASSO/CUPID is detected by the GT-42 monitor and directed to the PDP-11/40 for final language processing and eventual query response. The response is similarly detected by the monitor and displayed on the screen for user viewing.

The final language processing (see Appendix B) is written in the language "C" and operate on top of the UNIX time sharing system. The PICASSO/CUPID text string is compiled into QUEL using the compiler-compiler, YACC. The QUEL statement is immediately passed into an already invoked INGRES data base system. See Fig. 3.

Fig. 3   Software



4.3 Human Factors

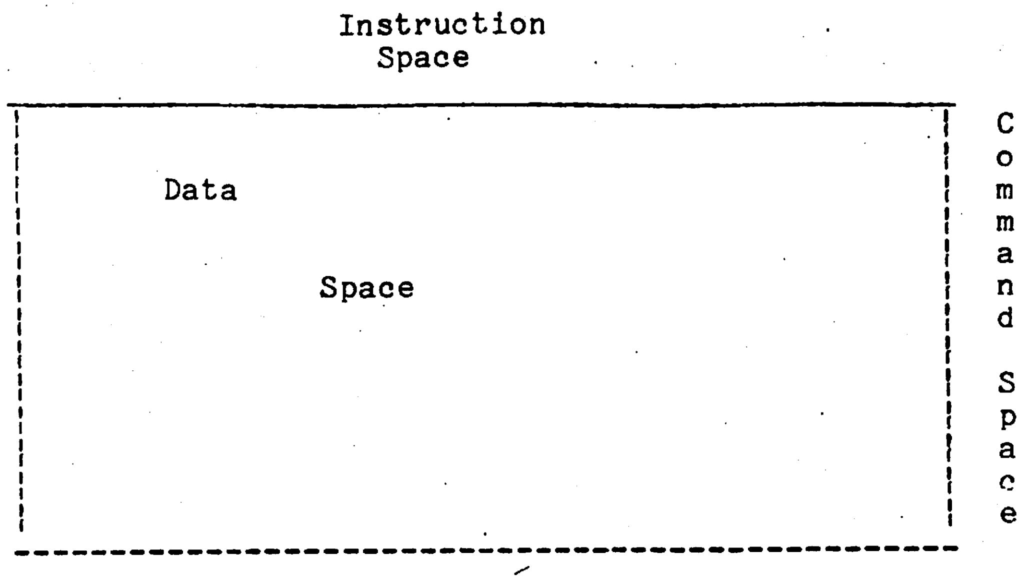
At all times during implementation, human factors considerations were paramount. The primary objectives were:

1 to make CUPID easy to use

2 to make CUPID easy to learn

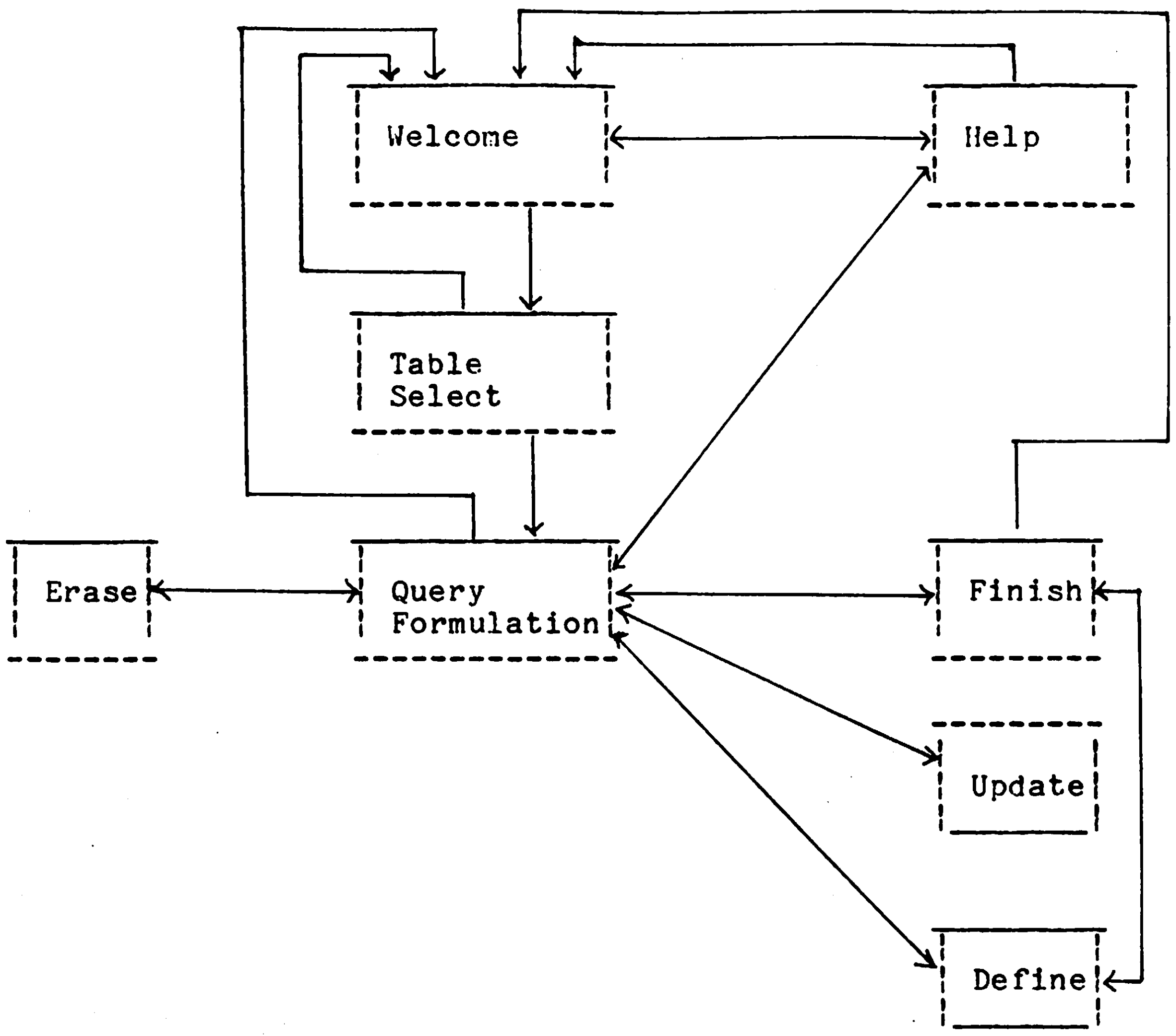
To these ends the general screen configuration, flow of control, editing facility and "help" facility were designed to be simple and forgiving. The general screen configuration is shown in Fig. 4. There is a specially delineated portion of the screen in which queries are drawn or information provided which is called the data space. Directly above the data space is the instruction space where the name of the phase is displayed along with prompting instructions to the user. To the right of the data space is the command space which provides the user with commands to direct the flow of control.

Fig. 4 General Screen Configuration

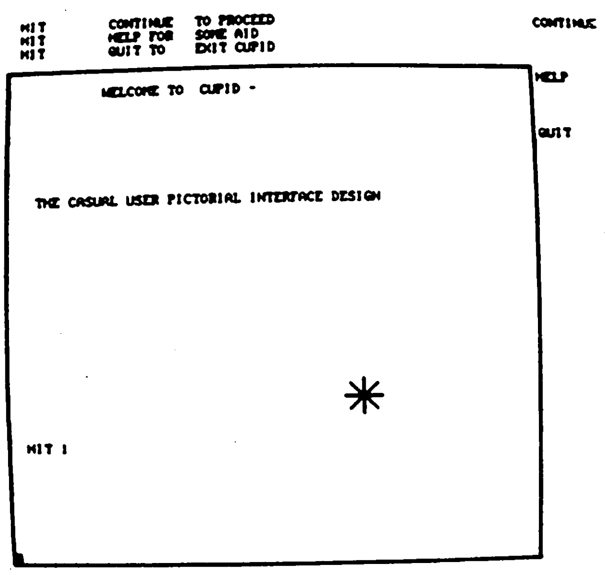


The flow of control is diagrammed in Fig. 5. Each block in Fig. 5 represents a phase. Arrows indicate the entry and exit points of each phase. An arrow emanating from the bottom of a phase-block indicates the natural progression from phase to phase in CUPID and is accomplished by selecting the CONTINUE command of the phase being exited. An arrow originating from the side of a phase-block indicates a specific command has been issued to direct the flow. In each case, the command corresponds directly to the name of the phase to be entered. All arrows emanating from the top of a phase-block return the user to the beginning of CUPID, the Welcome Phase.

Fig. 5   Flow of Control



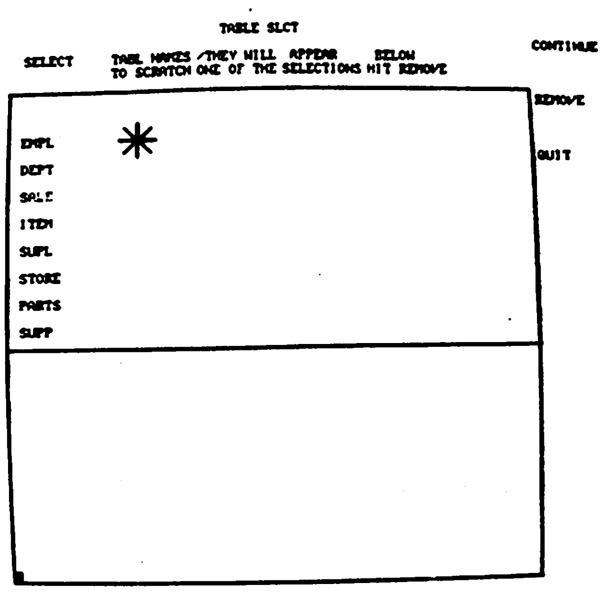
The following photographs of an actual CUPID session depict this flow and display the screen appearance at the beginning of each phase.



```
HIT     CONTINUE   TO PROCEED
HIT     HELP FOR   SOME AID
HIT     QUIT TO    EXIT CUPID

      WELCOME TO  CUPID -

   THE CASUAL USER PICTORIAL INTERFACE DESIGN

   HIT !
```

Pic. 1 Welcome Phase

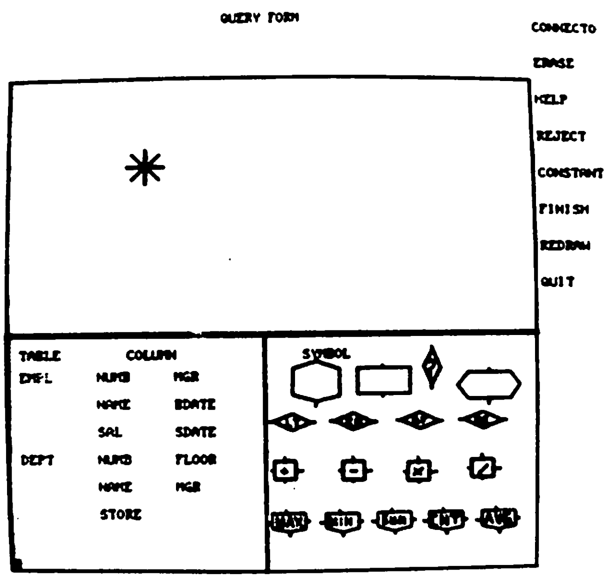
Entered at the start of CUPID and whenever the QUIT command is issued.



TABL NAMES /THEY WILL APPEAR BELOW
TO SCRATCH ONE OF THE SELECTIONS HIT REMOVE

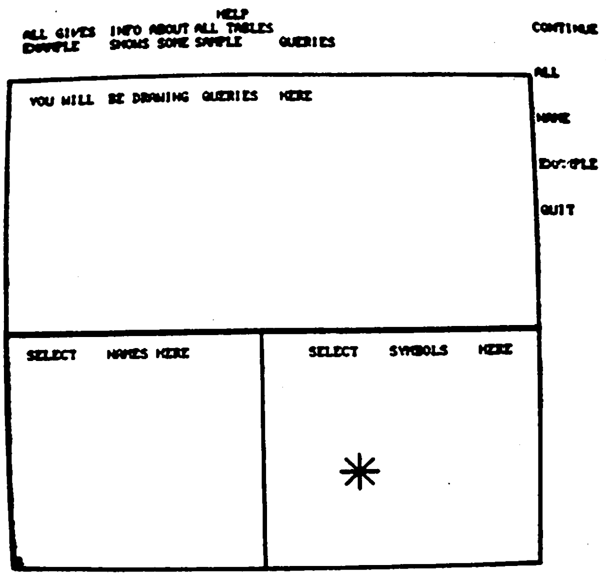
Pic. 2 Table Select Phase

Entered after issuing the CONTINUE command of the Welcome Phase.



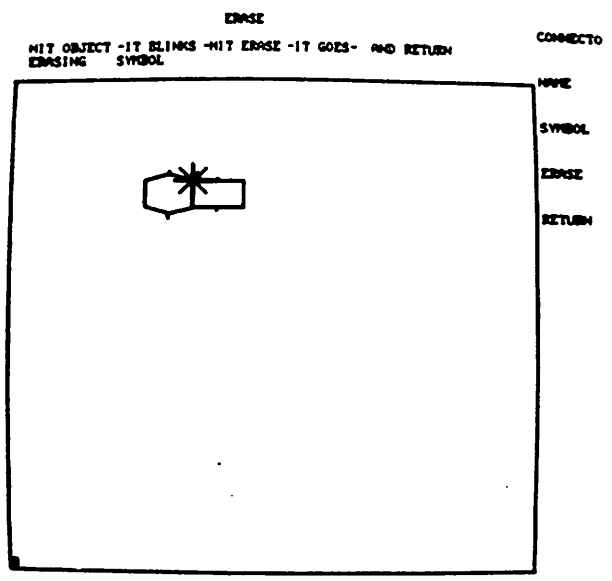
Pic. 3 Query Formulation Phase

Entered after invoking the CONTINUE command of the Table Select Phase.


HELP
ALL GIVES INFO ABOUT ALL TABLES
EXAMPLE SHOWS SOME SAMPLE QUERIES
CONTINUE
ALL
NAME
EXAMPLE
QUIT
YOU WILL BE DRAWING QUERIES HERE
SELECT NAMES HERE
SELECT SYMBOLS HERE

Pic. 4 Help Phase

This is the initial screen configuration. Commands invoked within this phase may utilize other screen configurations. One can enter this phase by issuing the HELP command in the Welcome or Query Form phase.

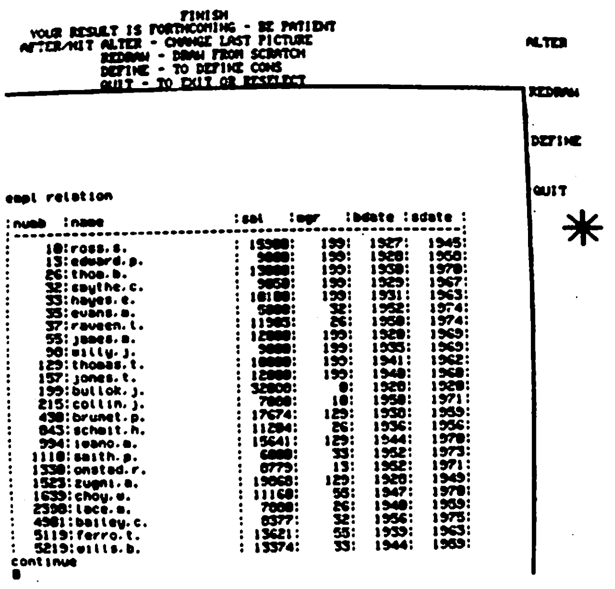

ERASE

HIT OBJECT -IT BLINKS -HIT ERASE -IT GOES- AND RETURN
ERASING SYMBOL

Pic. 5 Erase Phase

This is the basic editinq facility of CUPID.

It is entered from the Query Form phase via ERASE.



```
                    FINISH
   YOUR RESULT IS FORTHCOMING - BE PATIENT
 AFTER/HIT ALTER - CHANGE LAST PICTURE                    ALTER
           REDRAW - DRAW FROM SCRATCH
           DEFINE - TO DEFINE CONS
           QUIT - TO EXIT OR RESELECT
                                                          REDRAW

                                                          DEFINE

empl relation                                             QUIT

:numb  :name                 :sal    :mgr   :bdate :sdate :
:----------------------------------------------------------
:    10:ross.s.              : 15908:   199:  1927:  1945:
:    13:edward.p.            :  9000:   199:  1928:  1958:
:    26:thom.b.              : 13000:   199:  1938:  1970:
:    32:smythe.c.            :  9050:   199:  1929:  1967:
:    33:hayes.e.             : 10100:   199:  1931:  1963:
:    35:evans.m.             :  5000:    32:  1952:  1974:
:    37:raveen.l.            : 11985:    26:  1950:  1974:
:    55:james.m.             : 12000:   199:  1920:  1969:
:    98:willy.j.             :  9000:   199:  1935:  1969:
:   129:thomas.t.            : 10000:   199:  1941:  1962:
:   157:jones.t.             : 12000:   199:  1940:  1960:
:   199:bullok.j.            : 32000:     0:  1920:  1920:
:   215:collin.j.            :  7000:    10:  1950:  1971:
:   430:brunet.p.            : 17674:   129:  1938:  1959:
:   843:schmit.h.            : 11204:    26:  1936:  1956:
:   994:iwano.m.             : 15641:   129:  1944:  1970:
:  1110:smith.p.             :  6000:    33:  1952:  1973:
:  1330:onsted.r.            :  8779:    13:  1952:  1971:
:  1523:zugni.a.             : 19868:   129:  1928:  1949:
:  1639:choy.w.              : 11160:    55:  1947:  1970:
:  2398:lace.m.              :  7880:    26:  1940:  1959:
:  4901:bailey.c.            :  8377:    32:  1956:  1975:
:  5119:ferro.l.             : 13621:    55:  1939:  1963:
:  5219:wills.b.             : 13374:    33:  1944:  1959:
continue
```

Pic. 6 Finish Phase

Enter this phase from the Query Form phase and the future Define Phase via the command FINISH.

Points of human factors interest include:

1 Each phase (Welcome, Help, Table Select, Query Formulation, Erase, Finish) has a unique screen configuration which includes the phase name. The user always knows where he is within CUPID.

2 Brief, but helpful instructions at the top of the screen prompt the user.

3 All phases except Erase and Define, cycle back to the beginning (Welcome) through a QUIT command. This provides protection from premature, unwanted exiting.

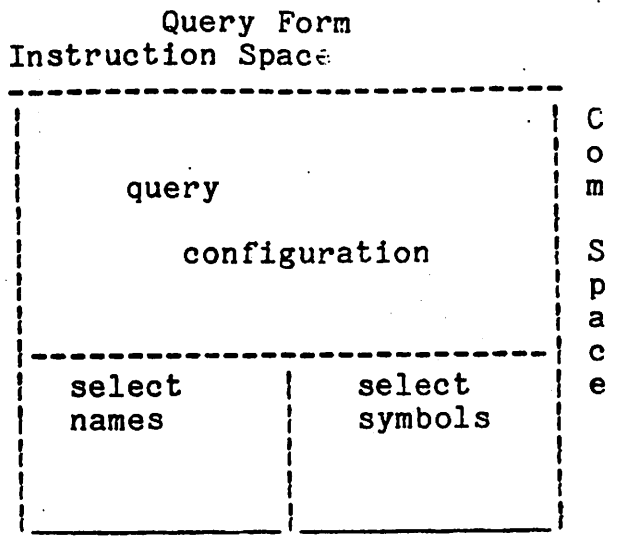
4 The user can get aid from Help while formulating his query without losing what he has already drawn.

5 Syntactic error checking in the Finish phase prevent the query from executing and erroneous data base accessing from occurring. If error diagnostics occur the user has the option to alter the present query or draw a new one.

6 The screen configuration (Fig. 6) of the Query Form phase is a combination of the general screen configuration and an implicit "menu-move" area as in [EVAN69] for selecting and placing names and symbols. The top of the screen is the modelling space (labelled "query configuration") into which CUPID symbols (from the bottom right) are moved. The bottom left portion of the screen

contains preselected relation and domain names. The only keyboard interactions occur when constant values are used.

Fig. 6 Query Form Screen



4.4 Working Implementation

Due to the lack of any graphics routines in UNIX, the decision was made to utilize those available in PICASSO. Since this project was primarily a feasibility study, the modeling capacity of PICASSO was extremely attractive. However, it was still necessary to spend one person-month altering PICASSO to conform to CUPID's design criteria. The following list describes some of the major modifications.

1 A routine (CONNECT) was added to provide a "near-hit" facility. Whenever a particular point of the screen is indicated by the user, this routine searches the data structure for the closest point within a user specified quantum. If such a point exists, then it is assumed to be what the user meant; otherwise, a new point is added to the data structure. This allows the user to be slightly inaccurate without great frustration.

2 A routine (MENU) was added to provide uniform system reaction to a user's selection of a particular command in the command space for PICASSO and CUPID.

3 The line drawing routine (PABLO) was modified to assume the "connector" drawing facility necessary in CUPID. The routine was more general than necessary, so appropriate entry and exit points (and necessary related code) were established to utilized only the required code.

4 Modifications to the editing facility (ERASER) provides

the specialized editing facility (the Erase Phase) for CUPID.

The CUPID routines are invoked from PICASSO by selecting the PICASSO command, USER COM. These routines direct the flow of control, provide a "help" facility and perform syntax checking and further specialized analysis. These Fortran routines constitute four person-months of effort. The major routines include:

1 USERCMD - called from the PICASSO program, directs the control of CUPID

2 BEGIN - provides the initial "welcoming" phase

3 HELP - initiates the "help" phase

4 EXAMPL and TABVIEW - present examples of queries and information about all relations of the demonstration data base in the "help" phase

5 TABSEL - provides the "table selection" phase

6 SCREEN - sets up the basic screen configuration for the "query formulation" phase

7 NAME, NAME1, SYMB and SYMB1 - display relation, domain names and picture symbols and acknowledge the user's selection of same

8 PLACEQS - places names and/or symbols in the query space

9 ANALY - provides a text analysis of the picture

10 THREAD - threads the relations and their domains together via tuple variables

11 SYNCHK - does a simple syntax check for such things as missing names, unconnected symbols and no target items

12 ORDER - orders the text corresponding to "range" statements and target list items to appear first in the output string

13 QUAL - orders the qualifications

14 AGG - handles aggregate operations

15 SEND - sends the appropriate information to the PDP 11/40

After the query has been formed it is passed on to the PDP-11/40 for language processing. In this phase, the GT-42 assembly language monitor had to be expanded to handle a second host computer over the DR11-c interface. Issues of timing and buffering were dealt with. One person-month was spent to find he most efficient usage of resources for this machine-to-machine interface.

Finally, a formal grammar and the appropriate "C" code (Appendix B) was written to transform the picture output into a QUEL statement via the compiler-compiler, YACC. The appropriate INGRES configuration was obtained through the invocation of a set of UNIX shell commands. This aspect was accomplished in three person-weeks.

4.5 Implementation Restrictions

In the retrieval mode, only logical operators and set and aggregate functions remain to be implemented. General logical AND operations (ANDing target list elements or qualification elements) are implicit in a CUPID query. For expediency, it was decided not to implement logical OR operators. Also for expediency, set and aggregate functions were omitted from the implementation. One target list element or qualification element can have, at most, two arithmetic and/or relational operators because Fortran has no facility for recursion. (In this prototype, two was deemed a reasonable number of iterations to be coded by hand in order to accomplish the needed recursion.) For the same reason, aggregates in qualifications have been left for a future exercise.

4.6 Plans for Further Implementation

Work has begun to transport the picture processing portion of CUPID from the CDC machine to the PDP configuration. PICASSO and the CUPID routines are being translated into "C" from Fortran. The data structures are being totally revised due to the change in word size. Also, the running program will likely be overlayed due to a smaller process size available in UNIX. It is estimated to take one person-year to complete this project. Once this is done, further CUPID implementation can be expedited.

Updates, as described in Chapter 2, require a simple addition to the Query Form phase. This addition will allow the user to formulate the update exactly like a retrieval and merely select the appropriate command- REPLACE, APPEND or DELETE- to initiate the update. Designed, but not as yet debugged is a macro-facility command for the Query Form phase, SAVEQ (and its counterpart RECALLQ), for naming, saving and recalling query diagrams. This facility will allow the user to save commonly used queries for easy reexpression. The "examples" shown in the Help phase use the same technique. Also designed for the Query Form phase, is a command, FININTO (finish-into) which will provide the ability to retrieve into a relation instead of retrieving back to the terminal. This parallels the QUEL "retrieve into" format. Finally, the Definition phase, whose purpose is described in Chapter Three, has also been designed.

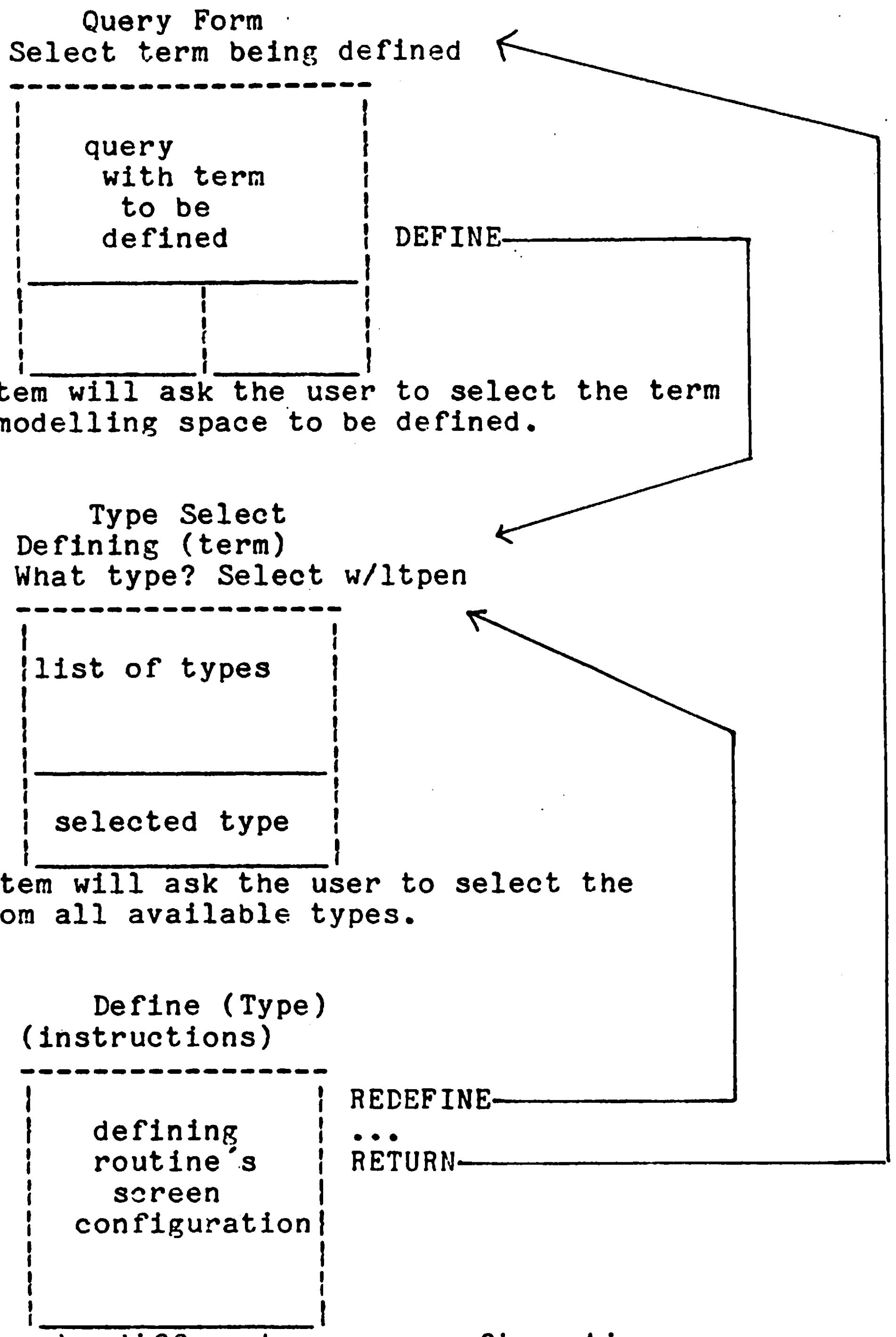
The design for implementing update commands involves replacing the present REJECT (see picture 5) command in the Query Form phase with the command UPDATE. When a user selects UPDATE, three new commands - REPLACE, APPEND, DELETE- will appear at the end of the command space. The user then draws his query, selects the appropriate update command, then selects FINISH to process the update. This procedure is purposely designed to be slightly more complicated and less automatic than the retrieval procedure as a means of protecting the user from making unwanted changes to the data base.

To save a query picture the SAVEQ command will ask the user to type in a name for the query appearing on the screen; to select a "reference" point of that diagram used primarily for placing the diagram on the screen in RECALLQ; the diagram will be stored and cataloged in a directory of saved queries (DSQ). The user will be prompted when the store operation has completed and asked to continue. The screen will not change. This will allow the user to save part or all of a query at any time while formulating it. For example, saving the basic format (all HEXes, BOXes and other symbols appropriately connected) without any relation names, domain names or constants filled in, only requires the user to fill these in without drawing the whole picture each time. RECALLQ will work in an analogous manner to draw a saved query. The user may wish to see a listing of his DSQ. Another command is necessary in the HELP phase. LISTQ would provide a list of this user's DSQ.

When the user selects the command FININTO instead of FINISH, he will be required to type in the name of a new relation and hit FININTO again. The retrieval is performed and the answer stored in the new relation. As in the QUEL equivalent, a relation with the indicated name must not already exist. This command acts like FINISH except the result is not printed on the screen. The user knows that the retrieval has been successfully performed when he sees the INGRES prompt "continue".

The Definition phase entails more than the addition of the command DEFINE to the Query Form and Finish phases. This facility requires a set of screen configurations. See Fig. 7 for the details of this set.

Fig. 7 Define Phase Screen Configurations


Query Form
Select term being defined
query with term to be defined
DEFINE
Type Select
Defining (term)
What type? Select w/ltpen
list of types
selected type
Define (Type)
(instructions)
defining routine's screen configuration
REDEFINE
...
RETURN

The system will ask the user to select the term in the modelling space to be defined.

The system will ask the user to select the type from all available types.

There may be different screen configurations for different routines.

The arrows emanate from the command and point to the phase entered by issuing the command.

The user may enter the Define Phase from either the Query Form phase or the Finish Phase. He may know that he wishes to define a new term while drawing the query (in Query Form) or he may be notified of an error caused by an undefined term (in Finish). In either case, the same procedure is invoked. The system will ask the user to select the "type" he wishes the term to be. This will produce the Select Type phase and its screen configuration. This phase is modeled after the Table Select phase. After selecting the type, the system will enter a defining mode for that type of data and proceed through a series of system-user interactions to develop a definition. Once the term is defined, the system will return to the Query Form phase where the user can continue, save, issue or edit the query.

CHAPTER 5

Human Factors Experiment

An experiment was designed to compare the two languages, QUEL and CUPID. In most previous experiments conducted to compare computer languages [REIS75,YOUN74,SHNE74,SIME73,WEIS73,SACK70] subjects' performance was tested on paper, without any human-machine interaction. This author believes that the paper-tested method is artificial and inappropriate. When teaching and testing a computer language, the environment established by the computer is extremely important. The machine's reaction to human actions and errors affects human performance. This environment can only be approximated if teaching and testing is done with paper and pencil. Moreover, in CUPID's case, it was imperative that the entire system be used. CUPID's two-dimensional language was not designed to be drawn on paper. Any attempt to immitate the functional design without a computer would be meaningless. This experiment was developed so that each language was taught and tested entirely through interactions with the computer. It was hoped that this methodology would provide more realistic results than previous experiments.

This chapter describes the design and results of this experiment. The experimental procedure is detailed along with the principles of the experimental design. Finally, the results are discussed and conclusions drawn.

5.1 Experimental Design

For each subject the following procedure was followed.

PART I

1 The subject was handed an EXPLANATION (Appendix C) page describing the experiment, its purpose, format, and some motivating information.

1a The EXPLANATION was read aloud by the experimenter.

2 Questions of any kind were answered.

3 The subject was handed one tutorial and told to perform the exercises as it directs. The same twelve queries (t1,t2,...t12) are used in each tutorial (Appendices D and E). The experimenter was nearby at all times but not in the same room. Questions concerning the machinery were answered, but questions dealing with an understanding of the language were answered only in reference to the tutorial.

4 He was asked to pose fourteen queries(q1,q2,...q14 - in Appendix F) to test what was learned during the tutorial session.

PART II

Follow the pattern of PART I with the other tutorial and omit 1a.

PART III

The questionnaire (Appendix G) was handed to and completed by the subject.

This experimental design was based on six principles.

1 Direct Interaction with the Computer

Subjects interacted directly with the system they were testing, using the same terminal (DEC GT42 graphic display). A log of all interactions was made for later analysis.

2 Self-instruction

The experiment was as self-explanatory as possible, with the subjects learning each language via a written tutorial. This was done to reduce human intervention and possible bias.

3 Completeness

An attempt was made to cover as many different aspects of the languages as possible, to compare subject reactions to and performance on simple and difficult constructs.

4 Brevity

The experiment was kept as brief as possible to maintain subject attention and interest.

5 Separate Test Days

Testing of the two languages was done on separate days, to maintain subject interest and reduce learning carryover from one language to another.

6 Random Order of Testing

The order in which the languages were presented alternat-ed from subject to subject, to remove order biasing.

5.2 Subjects

Subjects were paid volunteers. Their professio ranged from housewife to doctor to student. Twelve were o of school, five were undergraduates and five were gradua students. Subjects' ages ranged from 17 to 62. Out of th 22 subjects tested, 12 were unacquainted not only with da base systems, but with computers entirely. These 12 "naiv subjects had never taken a computer course and knew no com-puter language before this experiment. Of the remaining te subjects, seven were unacquainted with data base system but knowledgeable of computers: they had taken at least tw computer courses and knew at least one computer langua other than those tested here. The remaining three subject were "sophisticated" data base and computer users: they ha taken at least five computer courses, knew at least thre languages, and knew more than a little about some data bas system. These three categories of subjects will be referre to as naive, casual, and sophisticated.

The number of sophisticated subjects is admitted small. Their reactions were interesting though not use when testing for statistical significance. Such sophisticat-ed subjects were in very short supply. Two of the thre sophisticated subjects were INGRES/QUEL implementors an: possibly biased toward QUEL.

No attempt was made to control for educational level. Subjects varied from high school drop out(1) to Ph.D candidates(2). Four people had high school diplomas, two ha: bachelors degrees, two are pursuing masters degrees and tw: had some college credits. Educational levels for the remaining nine subjects is unknown.

5.3 Experimentation

QUEL and CUPID provide the ability to log the user anc system interactions for a single session. As a method of measuring the ease of use of the two systems, the logs of tutorial sessions are examined and compared. Since the queries(t1,...t12) posed during tutorial sessions were entirely directed by the tutorial, the number of attempts an: error statistics provide information about the simplicity of each system. Furthermore, we obtained information about the learnability of the two systems by comparing attempt an: error statistics from test-query (q1,...q14) sessions.

NOTE: All error statistics were recorded from the point i: time when the command was issued (not during the writing c drawing of the query). Therefore, one can assume that the subject was relatively confident of the query when these errors were observed.

Finally, preferences and other subjective data are obtained from the questionnaires and examined.

1 A comparison of the ease of use.

For each of the twelve queries posed during tutorial sessions, the number of errors was collected. Table 1 presents this information, for the three subject categories. Typographic errors have been aggregated out of the total error statistics.

The twelve columns represent the twelve queries and are appropriately labelled t1,t2,...t12. The rows are divided into three groups with each group being a type of information collected. These groups include: the total number of errors (any and all errors causing the query not to be processed was counted as one error for that attempt); the number of attempts (one attempt was tallied for each "\g" in QUEL and each FINISH command hit in CUPID); and the number of typographic errors (all spelling errors out of the total errors). Each of these three types has been subdivided to present the statistics by subject category. Thus, the labels, N, C, S, and T are abbreviations for Naive, Casual, Sophisticated and Total. Each space in the table may contain two entries. The statistics for the CUPID sessions are printed above the QUEL statistics. Zeros have been suppressed.

TABLE 1 Tabulated Data From Tutorial Sessions

		t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12
Total Number of Errors	N	4 1	0 7	3 8	5 7	9 2	5 13	0 1	2 5	0 0	2 9	2 5	1 6
	C	5 0	0 3	1 2	3 6	0 5	2 3	0 1	3 7	1 4	2 9	1 10	0 7
	S	2 0	0 1	0 0	1 4	0 2	0 0	0 0	2 5	1 2	0 4	1 2	1 2
	T	11 1	0 11	4 10	10 17	9 9	7 16	0 2	7 17	2 6	4 22	4 17	2 15
Number of Attempts	N	16 13	12 19	15 20	18 19	21 14	17 25	12 13	14 17	12 12	15 21	14 17	13 18
	C	12 7	7 10	8 9	10 13	7 12	9 10	7 8	10 14	8 11	9 16	8 17	7 14
	S	5 3	3 4	3 3	4 7	3 5	3 3	3 3	6 8	4 5	3 7	4 5	4 5
	T	33 23	22 33	26 32	32 39	31 31	29 38	22 24	30 39	24 28	27 44	26 39	24 37
Number of Typo. Errors	N	 1	 5	 7	 7	 2	 9		1 2		 9	 5	1 3
	C	 0	 3	 1	 4	 4	 3	 1	1 6	 4	 4	 9	1 7
	S		 1		 2	 2			1 5	 2	 2	 2	 1
	T	 1	 9	 8	 13	 8	 12	 1	3 13	 6	 15	 16	2 11

N="naive", C="casual", S="sophisticated", T=total.
t1...t12 denote the twelve queries posed.

CUPID results - upper left
QUEL results - lower right

The most significant (By the Student's t-Distribution: t=5.07 which indicates that the probability of the null hypothesis (i.e. no difference in the usability existed) is less than .001. Hereafter, this probability will be denoted by p). measurement of usability was the number of errors encountered during each subject's session with the tutorial. The results shown in Table 1 indicate that there were more than twice as many errors in the QUEL testing as encountered during CUPID sessions. These results were most striking in the more difficult constructs (t6, t10, t11 and t12).

2 A comparison of learnability.

Results of subjects' interactions during attempts to pose the fourteen test queries, produced Table 2. The columns of Table 2 represent the fourteen test queries q1,q2,...q14. The rows are grouped by the type of information collected. Each query was analyzed for correctness, number of attempts, and total number of errors, by subject category. The correctness statistic is a number from 0 to the total number in that subject category. If all naive subjects posed the query correctly at some time, a 12 would appear in the row headed by "N". The number of attempts is tallied as in Table 1, as is the number of errors. (Note that more than one error could be recorded for a given attempt.) These groups were again subdivided into subject categories as discussed above.

TABLE 2 Tabulated Data From Test-query Sessions

		q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14
Number Correct	N	12	12	10	11	10	12	11	12	11	12	11	10	11	10
		12	10	10	9	7	12	9	12	9	10	9	6	11	8
	C	7	7	7	6	6	7	5	7	6	7	6	6	5	6
		7	7	5	4	4	7	6	6	4	7	6	5	5	4
	S	3	3	3	3	3	3	2	3	3	3	3	3	3	3
		3	3	3	3	3	3	3	3	2	3	3	2	3	2
	T	22	22	20	20	19	22	18	22	20	22	20	19	19	18
		22	20	18	16	14	22	19	21	16	21	18	13	20	14
Number of Attempts	N	12	13	16	14	18	12	22	12	19	13	18	19	15	22
		12	15	19	14	14	14	14	12	15	12	14	12	13	14
	C	7	7	10	7	10	7	9	7	12	8	7	15	7	15
		8	7	8	7	9	7	9	7	9	8	8	12	9	9
	S	3	3	4	3	5	3	3	3	4	3	3	4	4	3
		4	4	3	4	5	3	4	3	4	3	3	3	4	4
	T	22	23	30	24	33	22	33	22	35	24	28	38	26	40
		24	26	30	25	28	24	27	22	18	23	25	27	26	2[illegible]
Number of Errors	N			6	6	3		3		8	1	6	3	1	12
			4	6	7	7	2	9		7	1	5	14	1	9
	C		1	3		1		2		6	1	1	4	2	8
		1		1	3	8		3	1	5	2	2	7	3	4
	S		1	1				1		1			1	1	
		1	2		1	3		2		2			3	2	1
	T		1	10	6	4		6		15	2	7	8	4	20
		2	6	7	11	18	2	14	1	14	3	7	24	6	14

N="naive", C="casual", S="sophisticated", T=total.
q1...q14 represents the fourteen test queries posed.

In Table 2 the number of correct answers, the number of attempts, and the number of errors encountered while the subject was posing the fourteen test queries gave some indication of the learnability of the two languages.

a. Number correct

The majority of subjects answered more of the fourteen test queries correctly in CUPID than in QUEL. Using the Student's t-Distribution, this statistic was found to be significant for $p<.001$. The naive subjects answered 13% more queries correctly, while the casual users performed (answered correctly) 12% better. The sophisticated subjects did 5% better in CUPID.

b. Number of tries

There were more attempts ($t=3.6 > t(.05)=3.01$) made in CUPID, especially when difficulties were encountered (queries q5,q6,q9,q12,q14). Although the subjects were instructed to pose a query a maximum of three times in both languages, 20% of the subjects exceeded three attempts in CUPID while 3% of the subjects exceeded three attempts in QUEL.

c. Number of errors

There were 20% more errors in queries issued with QUEL than those issued with CUPID. This fact was found to be significant ($p< .01$). Naive and casual subjects had less trouble formulating more difficult queries (queries q3, q4, q5, q7, q9, ,q12, q14) in CUPID . The simpler queries were almost all done correctly in one or two tries.

3 Types of errors

Errors were further divided into three broad type categories: syntax , semantic and typographic. Due to the large quantity of information, it was necessary to use a table for each language. Tables 3A and 3B present the information recorded during CUPID and QUEL sessions. Each table follows the same format: columns represent the fourteen test queries (q1,...q14) and rows are grouped into number of syntax errors, number of semantic errors and number of typographic errors. These groups are subdivided into subject categories (N, C, S, and T). Entries in the syntactic and semantic groups include a number to indicate the quantity and a two-letter abbreviation, as discussed below, to further specify the errors.

In QUEL, syntax errors were categorized as:
PE - parenthesis error
MK - misplaced keyword
PP - other punctuation problems

CUPID's "syntax" errors included:
MS - misconnected symbols
MN - misplaced names or constants
FS - forgotten symbols

Since the semantics of the two languages are the same, the three following categories of semantic errors applied to both QUEL and CUPID.

TV - misunderstanding the tuple variable construct
L - misunderstanding linking between relations
AG - misunderstanding the aggregate construct

TABLE 3A Types of Errors During CUPID Test-query Sessions

		q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14
Syntax Errors	N			2fs	1ms 1mn	1ms 1fs		3ms		2ms 1fs	1fs	2ms 1mn	2fs	1fs	2ms 3mn 1fs
	C		1ms			1ms		2ms		3ms	1fs	1ms	2ms	2ms	2ms 2mn 2fs
	S			1fs				1fs		2fs			1fs	1fs	
	T		1ms	3fs	1ms 1mn	2ms 1fs		5ms 1fs		5ms 3fs	2fs	3ms 1mn	2ms 1fs	2ms 2fs	4ms 5mn 3fs
Semantic Errors	N			4ag	2l	1l				4l		3l	1l		4l
	C			3ag						3l			2l		2l
	S														
	T			7ag	2l	1l				7l		3l	3l		6l
Typo. Errors	N									1					
	C							1		1					
	S														
	T														

N="naive", C="casual", S="sophisticated", T=total
q1...q14 represent the fourteen test queries posed.
ms=misconnected symbol; mn=misplaced name; fs=forgotten symbol; tv=tuple variable problem; l=linking problem; ag=aggregate problem.

TABLE 3B Types of Errors During QUEL Test-query Sessions

		q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	q11	q12	q13	q14
Syntax Errors	N		2pe	2pe 1pp	2pe	2pe		1pe 2pp		1pe		1pe 1pp	2pe		
	C				1pe	1pe 1pp			1pe						
	S					1pp		1pp					1pp		1pe
	T		2pe	2pe 1pp	3pe	3pe 2pp		1pe 3pp	1pe	1pe		1pe 1pp	2pe 1pp		1pe
Semantic Errors	N			1ag	11	1tv 11		2tv		2tv 21	1ag	1tv 21	3tv 51		2tv 21 2ag
	C			1ag	11	1tv 31				1tv 11	1tv	11	1tv 31	2tv	11 1ag
	S					11		1tv 11			1tv 1ag		11		
	T			2ag	21	2tv 51		3tv 11		3tv 31		1tv 31	4tv 91	2tv	2tv 31 3ag
Typo. Errors	N		2	2	4	3	2	4		2			4	1	3
	C	1			1	4		3		3	1	1	3	1	2
	S	1	2		1	1				2			1	2	
	T	2	4	2	6	8	2	7		7	1	1	8	4	5

N="naive", C="casual", S="sophisticated", T=total.
q1...q14 represent the fourteen test queries posed.
pe=parenthesis error; mk=misplaced keyword; pp=other
punctuation problem; tv=tuple variable problem;
l=linking problem; ag=aggregate problem.

The types of syntactic errors are not comparable, but provide information about each language separately. The majority (68%) of these errors in QUEL fell under the category of 'parenthesis error'; while the majority (64%) of these errors in CUPID were classified as 'misconnections'.

The tuple-variable type of error occurred only in QUEL; 48% of all errors in QUEL came from this misunderstanding. Naive subjects who said in the questionnaire that they excell in English and related subjects had the greatest difficulty learning this concept (97% of these errors). The two naive subjects who had the greatest difficulty with QUEL (even the simpler queries) did not seem to grasp this concept.

Linking terms appeared to be easier to learn (queries q4, q5, q9, q11, q12, q14) in CUPID. Although 50% of these errors occurred in CUPID, more (98 correct attempts) of the queries concerned were finally phrased correctly in CUPID. Only 77 correct queries were posed in QUEL sessions for the same five queries. This result was found to be significant at the .01 level.

The aggregate construct was a little more difficult to learn in CUPID (70% of the errors occurred during CUPID interactions). The errors involving aggregates made during QUEL sessions were mainly (90%) due to to misunderstanding or forgetting to name the domain in which the aggregate answer was to be placed.

All but three of the total 60 typographic errors occurred during sessions with QUEL.

4 Subjective results

To measure subjective reactions to both systems, questionnaires were analyzed, producing the information shown in Table 4. In this case, the columns represent the subject categories and are labelled accordingly, NAIVE, CASUAL and SOPHISTICATE. The rows represent the following groups (and subgroups) of information.

a. Preference

Subjects were asked which language they preferred(CUPID or QUEL), and what they liked and disliked about each language. Their preferences are quantified in the first two rows of Table 4, but discussion of further reactions is deferred to a later section.

b. Computer background.

Subjects were asked to list the number of computer science courses taken and languages learned. They were also asked to state whether they knew nothing, a little, or a lot about data bases before the experiment. This information appears in rows labelled "# courses", "# languages", and "knowdb", respectively. The last subgroup ("knowdb") presents the number of subjects who knew "a little" or more about data base systems.

c. Aptitude

They were requested to state in which of three areas they excell: English and related subjects, all subjects, or math and related sciences. The corresponding subgroups in this category are "English", "All", and "Math".

d. Personal background.
The subject's age and sex were requested and recorded in rows of the same names.

e. Opinion of experiment.
Detailed opinions of the clarity of the experiment and each tutorial were requested. Table 4 presents this information in rows labelled: "Clear ex.", to show how many people found the experiment's instructions clear; "Clear CUP", to display the number of subjects who thought the CUPID tutorial was understandable; "Clear QUEL", to show the number who found the QUEL tutorial clear; and "GT42", to indicate the number of subjects who had difficulties with the hardware.

TABLE 4

	NAIVE	CASUAL	SOPHISTICATE
CUPID	9	6	1
QUEL	3	1	2
# courses	0	1.5(avg)	4.3(avg)
# language	0	1.0(avg)	4.2(avg)
knowdb	0	0	3(well)
English	8	3	0
All	4	2	2
Math	0	2	1
Age	31.2(avg)	20.6(avg)	22(avg)
Sex	4M 8F	6M 1F	2M 1F
Clear ex.	12	7	3
Clear CUP.	10	7	3
Clear QUEL	10	7	3
GT42	11	7	3

Table 4 collates subjects' opinions, reactions and feelings. Seventy-three percent of the subjects preferred CUPID to QUEL. Using a Chi-Square analysis, this statistic produces a $X^2=3.2$, significant at the 10% level). Everyone found the experiment clear and most said the tutorials were easy to follow (99%). Most everyone, including the three people who had used CRT terminals before, found fault with the hardware equipment, specifically the GT42 terminal ("lightpen lousy", "flakey machine", "sometimes too sensitive, sometimes not sensitive enough").

5.4 Discussion

A measure of usability was obtained from subjects' performance during tutorial sessions and their stated preferences on the questionnaires. Since the tutorials presented the subject with a step by step procedure and each exact interaction necessary, reasons for errors should be few. The fact that fewer errors were encountered during CUPID sessions and 73% preferred CUPID, indicates that CUPID is easier to use.

Subjects' interactions during sessions posing the fourteen test queries gave a measure of learnability. While most of these results (Tables 2, 3A and 3B) speak for themselves, there are several additional points to be made.

1 The statistics for the number-of-attempt (greater than 3 attempts per query happened more often in CUPID than in QUEL) may indicate either that the subjects did not understand the instructions; or that they were having such "fun" they did not count their attempts; or that they learned the language better from their mistakes.

2 The analysis of syntactic error provides no comparison information between the languages. However, it does indicate areas of possible redesign to make each language easier to learn.

3 The number of tuple-variable errors in QUEL suggests that the algebraic overtones of the tuple-variable construct in QUEL is difficult to accept or understand. Possibly, CUPID's subtler handling of the tuple variable and blatent handling of the linking mechanism makes linking relations easier.

4 The primary confusion expressing aggregates in CUPID seemed to be designating the target list of the aggregate while the aggregate was itself part of a target list. Several subjects commented on the lack of detailed explanation of aggregates in the tutorial which may account for some of the difficulty there.

5 The CUPID typographic errors were probably due to a particular PICASSO restriction (character strings must be less than 10 characters long) which was noted in the tutorial.

6 It is possible to attribute some of the typographic errors made in QUEL to unfamiliarity with the specific terminal's keyboard (which is different from a standard typewriter). However, the QUEL statement appears plainly on the screen before the subject issues the query as does the CUPID picture. Mistakes should be caught at that time. The fact that more errors are made (or fewer errors are noticed) in QUEL suggests that CUPID, the graphic language, is either different enough to cause the subject to be extra careful or visual enough to reduce error production and/or

to make errors more obvious.

The following discussion is presented to provide a fee for a comparison of the friendliness of the two languages. The information was obtained from answers to the questionnaire.

When asked which language was preferred, 73% preferred CUPID (75% naive, 86% casual, 33% sophisticates). There is a relationship between those who preferred CUPID and subjects who excel in English (84% excel in English) as well as a correspondence between CUPID devotees and subjects who know fewer than two computer languages (87% knew no language). Those subjects who excel in the mathematical areas and sciences and who have experience with computer science(more than two languages) preferred QUEL (90%).

It is not surprising that people with training in formal computer languages should state a preference for QUEL. However, half of these people performed better(more correct, fewer errors) in CUPID.

In general, subjects who liked QUEL, liked it because it was more "mathmatical", "logical", and "easier to state". They didn't like "formulating the query", 'tuple variables', and having to specify range statements. The naive subjects did not like typing in the queries, while the casual and sophisticate subjects did.

CUPID qualities which were generally liked include being "easier to visualize", "easier to formulate", "more fun to do", and "easier to follow". Negative comments include: "slower to draw than to type", "slow editing mechanism", and "not complete".

5.5 Concluding Remarks

Of the subject pool tested, there was a decided preference for the graphic query language, CUPID. Four out of five subjects used CUPID easier than the query language QUEL; four out of seven subjects learned CUPID better(fewer errors) than QUEL; three out of four subjects liked CUPID better.

There is much more work to be done in the testing of human- machine interfaces. While this experiment was designed to compare two specific language systems, it also provides some important guidelines for future experimentation of this nature. Certainly the actual use of the two systems in question gives more accurate reports on usability and learnability. The variety of subject backgrounds gives a broad spectrum of opinion. We hope to continue human factors tests involving these two systems to investigate timing considerations, the importance of subjects' education levels, and further comparisons of more refined versions of the languages.

Perhaps this experiment lends some credence to the old, familiar saying that a picture is worth 10,000 words.

CHAPTER 6

Conclusion

This dissertation has described the design, implementation and testing of a picture query language system, CUPID. In this chapter the highlights of this work will be summarized and directions for future research will be indicated.

6.1 The CUPID Language

In chapters 2 and 3 this thesis discussed the attributes of the picture query language. In an attempt to make this casual user interface easy to learn and easy to use, a straightforward syntax and a flexible definition capability was designed.

The semantics of the language was designed to be one-to-one to the query language QUEL, while the syntax of CUPID omits the use of explicit punctuation and keywords. Instead, the user diagrams his query with system provided symbols and relation- domain names on a CRT graphic device with a lightpen. Although the picture representation was judiciously chosen, the potential for imaginative alternatives provides an area for further research and experimentation.

In describing the definition capability, an algorithm for handling type discrepancies was presented. Also, an algorithm for detecting and defining previously undefined constants was detailed. In conjunction with this process, the relational storage structure for representing user defined terms was described. Since each unique "type" of data may require its own unique defining routine(s), it is expected that these defining routines present an interesting field for future work in the design and development of various heuristic and other artificial intelligence techniques.

6.2 Implementation

The present implementation was detailed in Chapter 4. Also included in that chapter were implementation designs for several aspects of the CUPID system (including the definition capability) that are not presently implemented due to the unusual hardware configuration. It is expected that once CUPID is converted to run entirely on the PDP-11, further implementation will be expedited.

6.3 Experimentation

In Chapter 5 the design and results of a psychological experiment devised to compare CUPID to the keyword-oriented query language, QUEL was described. In testing human reactions to a human-machine interface, it was extremely important to conduct the experiment using the computer and the systems in question. The results indicated that even totally naive subjects learned CUPID easily and 73% of all subjects preferred CUPID to QUEL.

In general, human factors experimentation may be a fruitful area for further research. Specifically, with respect to CUPID, further investigations which could prove very useful include:

1 comparing timing considerations
2 comparing more advanced versions of the two systems
3 correlating aptitude, preference and correctness
4 correlating education level, preference and correctness
5 performing protocol analyses with the two systems

6.4 Implications

This project demonstrates a new approach to query languages and provides a working prototype. Its implications may be far reaching, affecting:

(a) specialized application areas
(b) the programming language field
(c) the graphics industry

A pictorial interface may significantly enhance usability of the computer in such application areas as report generators, computer aided teaching programs, and computer aided design facilities.

There is no need to limit picture languages to data base management systems. Pictorial general purpose computer languages can be developed, especially for the non-programmer.

The concept that information can be expressed easily and precisely by a picture language will provide incentive for further graphics develpment. As picture interfaces become more popular than keyboard interfaces, better and less expensive graphics hardware and software will be needed.

In conclusion, this work may be considered a feasibility study of a pictorial query language system. As such, it has shown that such a system can be designed and implemented in a reasonable amount of time (approximately one person-year). Moreover, the system has proven, under comparative testing, to be a plausible human-machine interface. Even in it present primitive implementation, CUPID is relatively easy to use, easy to learn and appealing to casual users. It remains for future research to pursue this concept and investigate further additions and refinements in order to provide further casual user pictorial interface designs.

Appendix A - Formal Syntax

The following is a formal description of CUPID's retrieval mode syntax. Although the update mode has the same syntax, it utilizes different commands to denote the various operations.

Notational meanings:

All closed figures contain names, constants or CUPID provided items. Constants are keyboard entries.

Each terminal symbol has a connecting point, designated by '.'; other terms use the connecting point of their highest precedence binary operator. While only one connection point is shown in the following definitions, aesthetic reasons cause us to use more than one point in implementation.

The special symbols devised to express picture positioning include:

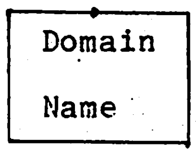
a) $a \downarrow b$: connected from a to b by a connecting line (connector)

b) $a \Rightarrow b$: a is to the left of b and touches it directly

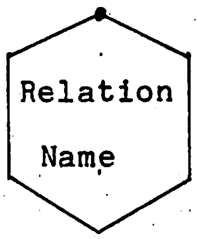
c) $a \rightsquigarrow b$: a and b are anywhere in the query

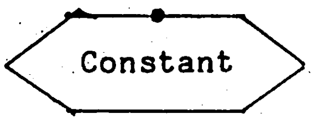
d) $a \overset{\sim}{\rightsquigarrow} b$: a is positioned to the left of b and not connected by lines, but by contiguous picture parts.

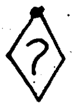
The Grammar

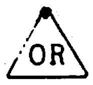
box: 

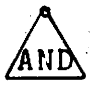
boxes: box ¦ box boxes

hex: 

const: 

qbox: 

lbop: 

¦  luop: NOT

rop: EQ ¦ NE ¦ GT ¦ LT ¦

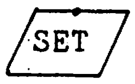
GE ¦ LE

bop: $\boxed{+}$ ¦ $\boxed{-}$ ¦ $\boxed{*}$ ¦ $\boxed{/}$

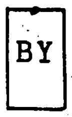
uop: ⊕ ¦ ⊖

log: $\boxed{\text{LOG}}$

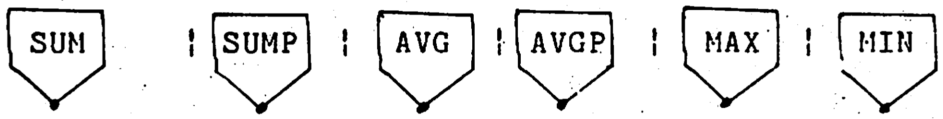
set:



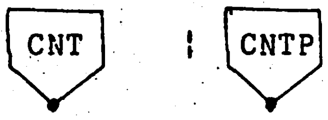
by:



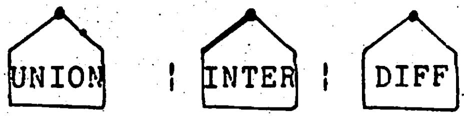
aop:



aop2:



sop:



```
tl1:      box ↓ qbox     ¦   a_fcn ↓ qbox
tl:       tl1      ¦  tl1⇝tl
qual:     luop↓ qual   ¦  qual↓ lbop↓ clause   ¦
          clause
clause: a_fcn↓ rop↓ a_fcn   ¦  a_fcn↓ rop↓ a_clause   ¦
          set_clause   ¦  luop ↓clause
a_clause:         a_fcn ↓rop ↓a_fcn
a_fcn:  attrib_fcn   ¦  aggr_fcn   ¦  ag2_fcn   ¦
          a_fcn↓ bop↓ a_fcn   ¦  log↓ const↓ a_fcn   ¦
          uop ↓a_fcn
attrib_fcn:  const   ¦  box
aggr_fcn1:  aop↓ asblob   ¦  aop ↓sblob
sblob:  set ↓set_fcn
asblob:  tl⟿qual
aggr_fcn:  hex⇒boxes ↓ by↓ aggr_fcn1   ¦   aggr_fcn1
ag2_fcn1:  aop2↓asblob
ag2_fcn:  hex⇒boxes ↓by ↓ag2_fcn
attrib:  box
set_clause:  set_fcn↓ rop↓ set_fcn
set_fcn:  set_fcn↓ sop↓ set_fcn   ¦  set_fcn1   ¦
           set ↓sblob
a_seq:  a_fcn ↓a_seq   ¦  a_fcn
set_fcn1:  set↓ asblob   ¦  hex⇒a_seq↓ by↓ set↓ asblob
que:  hex⟿tl   ¦  hex⟿tl⟿qual
```

Appendix B - Implemented Grammar

```
%term HEX       RETRIEVE         QBOX     BOX      NCONST
%term WHERE     CONS     GT      EQ       LT       NE
%term MINUS     TIMES    DIV     MAX      MIN      COUNT
%term PRINT     SUM      WHERE_AGG        AND
%term   SCONST  PLUS     AVG
%left MAX
%left TIMES
%{
int nlines, linect, fnam, output, ncons, nn, aopcnt ;
char buf[135][40];

struct qpid {
        char *w;
        int tk;
        };

struct qpid token[] {
        "BOX",          BOX,
        "HEX",          HEX,
        "QBOX",         QBOX,
        "RETRIEVE",     RETRIEVE,
        "CONS",         CONS,
        "WHERE",        WHERE,
        "WHERE_AGG",    WHERE_AGG,
        "AND",          AND,
        "GT",           GT,
        "EQ",           EQ,
        "LT",           LT,
        "NE",           NE,
        "PLUS",         PLUS,
        "MINUS",        MINUS,
        "TIMES",        TIMES,
        "DIV",          DIV,
        "MAX",          MAX,
        "MUM",          MIN,
        "CNT",          COUNT,
        "AVG",          AVG,
        "SUM",          SUM,
        "PRINT",        PRINT,
        "SCONST",       SCONST,
        "NCONST",       NCONST,
        0
        };
%}
%%
prog:           prog stmt ¦ stmt;
stmt:           range ¦ retrieve ¦ print;
range:          HEX = {
```

```
                    printf(" range of ");
                    printf(" %s ",&buf[linect][20]);
                    printf(" is ");
                    printf(" %s ",&buf[linect][10]);
                    printf("\n");
                            };
print:              PRINT HEX = {
                    printf(" print %s\n",&buf[linect][10]);
                            };
xretrieve:          RETRIEVE = {
                            printf(" retrieve( ");

                            };
retrieve:                   xretrieve tl qual ;
tl:                 tlelm = {
                            } |
                    tl  comma tlelm = {
                            };
tlelm:              QBOX a_fcn ;
attrib:             BOX = {
                    if(buf[linect][30] != '0')
                    printf(" x = ");
                    printf(" %s . %s",&buf[linect][20],
                    &buf[linect][10]);

                            };

null:               =printf(" )\n");
cons:               NCONST = {
                            for(nn=0; nn<10  ;nn++) {
                            if(buf[linect][nn+10] == ' ')
                            buf[linect][nn+10] = '\0' ;
                                    }
                    if(atoi(&buf[linect][10],&ncons) != 0)
                    syserr("\ninteger conversion error\n");
                            printf(" %d ",ncons);
                            }|
                    SCONST = {
                    if(buf[linect][10]!='"')printf("\"");
                    for(nn=0;nn<10;nn++) {
                            if(buf[linect][nn+10]=='\0' ||
                       buf[linect][nn+10]==' ')break;
                            if(buf[linect][nn+10]>='A' &&
                               buf[linect][nn+10]<='Z')
                                    buf[linect][nn+10]=
                             buf[linect][nn+10]+'a'-'A';
                                                }
                            write(1,&buf[linect][10],nn);
                            printf("\"");
                            };
qual:               where qualif | null;
qual_agg:           where_agg clause           |
                    qual_agg and1  where_agg clause  ;
where_agg:          WHERE_AGG  =  {
```

```
                            if(aopcnt == 0){
                                    printf("\n where ");
                                    aopcnt = 1;
                                          }
                            }
xaop:               AVG ¦ MAX ¦ MIN ¦ SUM ¦ COUNT;
aop:                xaop = {
                    aopcnt = 0;
                    if(buf[linect][0]=='C')
                    printf(" c = count(");
                            else
                    printf(" %c =%s( ",buf[linect][0],
                            &buf[linect][0]);
                            }
qualif:              clause          %prec MAX ¦
                    qualif and clause        %prec TIMES;
where:              WHERE = {
                            printf(" ) \nwhere ");
                            };
a_fcn:              a_fcn1 ¦
                    agg_fcn;
a_fcn1:             attrib_fcn ¦
                    a_fcn1 bop a_fcn;
agg_fcn:            agg1_fcn =printf(" )\n");
agg1_fcn:           aop a_fcn qual_agg       %prec MAX ¦
                    aop a_fcn;
attrib_fcn:         attrib ¦
                            cons;
bop:                PLUS ={ printf(" + "); } ¦
                    MINUS ={ printf(" - ");  } ¦
                    TIMES = { printf(" * ");  } ¦
                    DIV  ={ printf(" // "); };
clause:             a_fcn rop a_fcn ;
rop:                GT= { printf(" > ");  } ¦
                    EQ= { printf(" = ");  } ¦
                    LT= { printf(" < ");  } ¦
                    NE= { printf(" != ");  };
comma:              =printf(" , ");
and:                %prec TIMES     =printf(" and \n");
and1:               AND =printf(" \n and");
%%
# include "/mnt/nancy/jim/yylex.c"
# include "/mnt/nancy/jim/main.c"
```

Appendix C - Explanation for experiment

EXPLANATION

The purpose of this experiment is to obtain your reactions to two different ways we communicate with a computer. One of the ways is called QUEL and the other is known as CUPID. You can think of them as different languages you might use to obtain information stored in a computer(this information is often called a data base).

The experiment consists of 3 parts:

1. You will be given a tutorial for one of the languages and asked to follow it and interact with the machine as it tells you. After completing the tutorial you will be asked to apply the language to some queries.

2. You will be given the tutorial for the other language and asked to proceed as in 1.

3. You will be asked to complete a brief questionnaire.

Try to follow each tutorial by yourself. If you do not understand something or need help please call Nancy immediately(dial 2-5649 or 2-7520).

During some aspects of both languages you will be typing in letters and numbers and other characters. Erasing is different in the two languages:

In QUEL -

to erase a single character, type #

to erase the whole line, type @

In CUPID -

to erase a single character, hit the rub out key

to erase a whole line, hit the lf (linefeed) key

Now you are ready to begin. The information you have stored in the computer is shown on the wall to the right of the terminal. The tutorials deal with the two tables labelled PARTS RELATION and SUPP RELATION. Their meanings will be explained in the tutorials. The other two tables - EMPL and DEPT - are tables that a department store administrator might find interesting. The EMPL table contains the following information: the employee number (numb), the name (name), the salary in dollars(sal), the manager's number (mgr), the employee's birthdate year (bdate), the starting date-year(sdate). The DEPT information contains the department number (numb), the name (name), the store number (store), the floor number (floor), and the manager's number (mgr).

For the tutorial --

Imagine, if you will, that you are the head of an automotive store. You have information stored about the parts you sell in the table PARTS and information concerning where and how you obtain these parts in SUPP. Assume the information is stored as you see it listed. Each of the queries is meant ot obtain some portion of the information. To grasp what we are attempting, think about how you might have to get the information if it were stored in a file cabinet - one table per drawer and one row of a table per sheet of paper in the drawer.

Appendix D - QUEL Tutorial

This document contains an introduction to the data base management system, INGRES, and in particular stresses its user language QUEL. It is meant to be read while interacting with the INGRES system at a terminal.

Please proceed by typing the lines underlined. A cr (carriage return) must be entered after every line typed.

One's first encounter with INGRES is to type the UNIX shell command

```
quelrun data-base-name
```

which has the effect of turning you over to INGRES for subsequent interactions. The actual sequence is:

```
%quelrun nancy
```

[you type - quelrun nancy and cr(carriage return)]

Here, we have entered INGRES and specified that we are interested in the data base "nancy" which will have in it the tables of interest to this document. After a momentary pause the following will be returned to your terminal.

```
INGRES vers 4.0 login Tue Aug 26 20:02:34 1975
    Set  operators,  Aggregate  functions  and X.ALL are not
```

implemented.

```
go
```

The first three lines constitute the current "dayfile" which gives relevant information on the status of INGRES. The statement "go" indicates INGRES is waiting for your input.

Now type

```
help "help"
\g
```

"help" is an INGRES command which can deluge you with information about the system. In this case, you will receive the page from the INGRES reference manual which describes the help command. "\g" is a statement to INGRES to execute the "help" command without waiting for additional input from the terminal. The response from INGRES is:

```
query formulation complete

 HELP(X)                 4/22/75          HELP(X)


     NAME
  help - get information about how to use INGRES

     SYNOPSIS
          HELP ["item-in-question"]

     DESCRIPTION
```

HELP may be used to obtain information about any sectio: of this manual, the content of the current dat: base, or a specific relation in the data base, dependin- on the item- in-question. Omission of that argumen: is functionally equivalent to HELP "help" .The othe- legal forms are as follow:

HELP "section" - Produces a copy of the specified sec- tion of the INGRES Programmer's Manual, and prints i: on the standard output device.

HELP "" - Gives information about all relations tha: exist in the current database.

HELP "relname" - Gives information about the specifie: relation, but in greater detail than would HELP "" .

EXAMPLE

```
HELP
HELP "quel"
HELP ""
HELP "empl"
```

SEE ALSO

DIAGNOSTICS

Unknown name - The item-in-question could not be recognized.

BUGS

Alphabetics appearing within the item-in-question must be lower-case to be recognized.

```
continue
```

The final line contains the word "continue". This indicates INGRES is ready to listen to you again.

At this point it is important for you to realize that INGRES maintains a workspace in which you formulate your interactions. This workspace is desirable so that you can correct spelling errors and other mistakes which you may from time to time make without having to type in your entire interaction again.

At the present time your workspace contains

```
help "help"
```

If you type in "\g" once more, INGRES will simply execute your workspace which will give you a second printout of what you have just seen above.

In order to clear out our workspace we use the command "\r"as follows:

```
\r
go
```

Our workspace now is empty. It is still possible to type in

"\g" as follows. However, it has no effect.

```
\g
query formulation complete
continue
```

We will now try to exercise the "retrieve" command and will do so on the data that now follows. To print the contents of any relation (or table if you are more comfortable with that terminology), simply type:

```
print relation-name
```

If we type help"" we can obtain a list of relations in the data base ndemo. One relation from this list is called "parts". We can print this relation as follows:

Q1:

```
print parts
\g


query formulation complete


parts relation

|pnum  |pname               |color   |wgt   |qoh   |
|--------------------------------------------------|
|     1|antifreez           |pink    |    10|     1|
|     2|wrench              |gray    |    20|    32|
|     3|tires               |black   |   685|     2|
|     4|ash-tray            |black   |   450|     4|
|      5|oil                |gray     |      1|   250|
|      6|chamois            |yellow   |    578|     3|
|      7|ornament           |white    |     15|    95|
|      8|seatcover          |blue     |     19|    15|
|      9|race-strp          |white    |      2|   350|
|     10|wash-solv          |clear    |      0|   143|
|     11|jacks              |gray     |    327|     0|
|     12|chrome             |gray     |    427|     0|
|     13|tape-play          |black    |    107|     0|
|     14|radio              |black    |    147|     0|
continue
```

Notice that the "parts" relation has information about the components in a hypothetical auto parts store. Each row of this table (or tuple in this relation) contains information on a given part including its part number (pnum), its part name (pname), its color, its weight (wgt), and the quantity that are on hand (qoh).

Using a "retrieve" command we will be able to obtain portions of this table which are of interest to us. (There is almost no limit on how large the tables can be; these examples are done on small ones so that this tutorial does not become too large. In fact, the actual limit on the size of a table is approximately 30,000,000 bytes for those who are interested.)

To obtain information, we must first tell INGRES what table it is that we wish to interogate. One way to do this might be the command

```
I WANT TO TALK ABOUT parts
```

Although this is natural to the beginner, INGRES makes you do something slightly more complicated. This added complexity is necessary so that one does not get into trouble with more complicated interactions.

The statement required in INGRES is

range of variable-name is relation-name

The variable-name is indicated to be a surrogate for the relation name specified. We can declare p to be this surrogate for "parts" as follows:

```
\r
go
range of p is parts
```

Notice that we first cleared our workspace so that the whole parts relation would not be printed again.

Now, we can add a "retrieve" command which can be the following

```
retrieve p.pname
```

Q2: The interpretation is that we wish to obtain the pname column of the relation specified by the variable "p".

In order to ensure that we have typed our interaction correctly we may use the special command "\p". This will simply print the contents of our workspace as follows:

```
\p
```

```
range of p is parts
retrieve p.pname
```

Since it appears to be a correct query we can execute it by the "\g" command as follows:

```
\g
query formulation complete
PERIOD = `.´ : line 3, syntax error
continue
```

Unfortunately, we have made a syntax error. What is more unfortunate is that INGRES is not always overly helpful in showing us what it is.

The problem with this interaction is an arbitrary convention in INGRES that whatever you wish to retrieve must be enclosed in "( )" . We will correct our mistake by retyping the query as follows:

```
\r
go
range of p is parts
retrieve (p.pname)
\g
query formulation complete
|pname               |
|--------------------|
|antifreez           |
|wrench              |
|tires               |
|ash-tray              |
|oil                   |
|chamois               |
|ornament              |
|seatcover             |
|race-strp             |
|wash-solv             |
|jacks                 |
|chrome                |
|tape-play             |
|radio                 |
continue
```

Everything has now worked out all right and we have obtained the column of the parts table which contains the names of the parts.

We can retrieve more than one column at once by simply indicating a sequence of column names separated by a comma. Hence we could obtain part names and colors as follows.

Q3:

```
\r
go
range of p is parts
retrieve (p.pname, p.color)
\g
query formulation complete
|pname                |color   |
|------------------------------|
|antifreez            |pink    |
|wrench               |gray    |
|tires                |black   |
|ash-tray             |black   |
|oil                  |gray    |
|chamois              |yellow  |
|ornament             |white   |
|seatcover            |blue    |
|race-strp            |white    |
|wash-solv            |clear    |
|jacks                |gray     |
|chrome               |gray     |
|tape-play            |black    |
|radio                |black    |
continue
```

Notice in the printout each column contains the name of the column so we do not get confused. Sometimes we require more complex results than simply the names of columns. Suppose, for example, we require the computation "1000-qoh". In other words, we wish to know for each part how many less than 1000 we possess. This can be stated as follows:

```
\r
go
range of pa is parts
retrieve (p.pname, computation=1000-p.qoh)
\g
query formulation complete

|pname                |comput|
|----------------------------|
|antifreez            |   999|
|wrench               |   968|
|tires                |   998|
|ash-tray             |   996|
|oil                  |   750|
|chamois              |   997|
|ornament             |   905|
|seatcover            |   985|
|race-strp            |   650|
|wash-solv            |   857|
|jacks                |  1000|
|chrome               |  1000|
|tape-play            |  1000|
|radio                |  1000|
continue
```

Note that the heading on your printout is the first six characters of the name "computation" which we have given to the computed quantity "1000-qoh".

In order for INGRES to accept computed quantities you must always give them a name. This is simply done by picking a name and putting it to the left of an equals sign in the retieval.

It is important that you spell correctly any column names which you use in an interaction, since INGRES has no spelling correcter at the present time.

Note lastly that you need not put interactions on three lines as we have been doing. It is usually wise to space your interactions so they are as readable as possible.

So far we have produced interactions which give us columns of the "parts" relation. We now indicate how to obtain only portions of columns. The basic mechanism is a "where" clause which can be added onto the end of the interactions we have been doing. If we wanted to see pnames and colors for those parts whose color is pink we would do the following:

Q4:

```
\r
go
```

```
range of p is parts
retrieve (p.pname, p.color) where p.color = "pink"
\g
query formulation complete
|pname               |color  |
|----------------------------|
|antifreez           |pink   |
continue
```

The "where" clause limits the number of rows which are examined to only those which satisfy the qualification given i.e. to those which satisfy "p.color="pink". Only antifreez has this property so it is the only entry in the output.

We are now to the point where we are typing enough information so that errors in typing are likely. It is very annoying to have to reset the workspace and try again every time an error is encountered. Two mechanisms are supported in INGRES to help with this problem.

1) INGRES accepts the symbol # to mean "backspace". Consequently, one can simply backspace and retype errors which occur. One can backspace as many times as one wishes; INGRES will continue to back up until it reaches the beginning of the current line. Subsequent backspaces will have no effect. If a line has become so garbled that the user wishes to simply erase it and start typing again one can use the symbol @ which means "erase the whole line"

2) More complicated corrections are often necessary than can be done easily using mechanism 1). These are supported by

calling on the features of the UNIX program called the editor. A tutorial on the editor is available in the UNIX programmer's manual. Here, we will simply discuss two features of this program. Since it is a very powerful program, the serious INGRES programmer would be wise to study that tutorial in more detail than the few exerpts we present here.

Suppose we type in an incorrect query as follows:

```
\r
go
ranhe of p is perts
retrieve p.pname
where p.pcolor = "pink="
```

This query has many errors and we might do better to start over, but for the exercise we will use the editor which we obtain by typing \e as follows:

```
\e
>>ed
```

The statement ">>ed" says now we are in the hands of the UNIX editor and our workspace has been sent to it.

We can sequence through our program by typing a line number followed by a carriage return i.e.

```
1
```

```
ranhe of p is perts
2
retrieve p.pname
3
where p.pcolor = "pink="
1
ranhe of p is perts
2
retrieve p.pname
3
where p.pcolor = "pink="
```

We have now looked at each line twice and are ready to fix each one.

We do this with a substitute command. This has the form:

```
s/this character string/that character string/
```

The editor goes through the current line of our command and finds the first instance of "this character string" and replaces it with "that character string". In this way we can find offending portions of our interaction and fix them.

First we do it for line 1.

```
1
ranhe of p is perts
s/ranhe/range/
s/perts/parts/
```

1

```
range of p is parts
```

After two substitutions, everything is fine.

Notice that you only need to specify enough of "this character string" so that the editor can correctly make the substitution.

Also, if you simply put a "p" after the last "/" , the current line will be automatically printed.

Notice lastly, that # and @ work the same way in the editor as in INGRES.

We now proceed to fix the rest of our statement without further comments.

2

```
retrieve p.pname
s/p/(p/
s/me/me)/p
retrieve (p.pname)
```

3

```
where p.pcolor = "pink="
s/pc/c/
s/k=/k/p
where p.color = "pink"
```

We have now fixed all lines and use the command "w" to send the corrected statement back to INGRES as follows:

```
w
```

We now issue a "q" command to quit the editor and return to INGRES as follows:

```
q
<<monitor
```

The echo "<<monitor" is to remind you that you have returned to INGRES.

It is usuallly wise to make sure INGRES got your corrected interaction back from the editor correctly by typing "\p" i.e.

```
\p
range of p is parts
retrieve (p.pname)
where p.color = "pink"
```

A "\g" will now execute the corrected command.

```
\g
query formulation complete
|pname                |
|---------------------|
|antifreez            |
continue
```

The operators "not", "and" and "or" are supported in INGRES. Users may simply use the operators remembering only to put a space on either side of them. It is sometimes essential to remember that the precedence of boolean operators is "not" then "and" then "or". Users who wish to alter this precedence (or who do not remember it) may use parentheses to precisely specify their meaning. The following interaction gives an example of multiple boolean operators.

Q5:

```
\r
go
range of p is parts
retrieve (p.pname)
where (p.color="pink" or  p.color = "gray" ) and p.pnum < 10
\g
query formulation complete
|pname               |
|--------------------|
|antifreez           |
|wrench              |
|oil                 |
continue
```

Three points should be carefully noted about the above interaction:

1) Character strings must be enclosed in double quote marks while numbers may be typed with no special delimiters.

2) Note the operator "<" in the above interaction. Valid relational operators include:

```
= (equals to)
< (less than)
> (greater than)
<= (less than or equal to)
>= (greater than or equal to)
!= (not equal to)
```

3) There is no limit to the complexity of the expressions which can be constructed using relational and boolean expressions.

We now do one last example concerning arithmetic operators in QUEL. This example finds the pnames and qoh of parts whose total weight (wgt times qoh) is less than 1000.

Q6:

```
\r
go
range of p is parts
retrieve (p.pname, p.qoh)where  p.wgt*p.qoh < 1000
\g
query formulation complete
```

```
|pname                    |qoh    |
|---------------------------------|
|antifreez                |      1|
|wrench                   |     32|
|oil                      |    250|
|seatcover                |     15|
|race-strp                |    350|
|wash-solv                |    143|
|jacks                    |      0|
|chrome                   |      0|
|tape-play                |      0|
|radio                    |      0|
continue
```

It should be noted that arithmetic operators can be used in the qualification portion of an interaction as well as in the portion indicating the desired information. Valid arithmetic operators include:

+ (addition)

- (subtraction)

* (multiplication)

/ (floating point division)

** (exponentiation)

mod (integer division)

It should also be noted that any user can save any result of an interaction by simply specifying the name of a relation into which the answer should be placed. The following suggests an equivalent way of obtaining the previous result. First a relation is created with the answer then the print command is used to display the result.

```
\r
```

```
go

range of p is parts

retrieve into local(p.pname, p.qoh )where p.wgt*p.qoh < 1000

\g

query formulation complete

continue

\r

go

print local

\g

query formulation complete

local relation
|pname                |qoh   |
|----------------------------|
|antifreez            |     1|
|chrome               |     0|
|jacks                |     0|
|oil                  |   250|
|race-strp            |   350|
|radio                |     0|
|seatcover            |    15|
|tape-play            |     0|
|wash-solv            |   143|
|wrench               |    32|
continue
```

Notice that local remains as a relation in the data base and may be used in any future interactions by simply declaring a range variable for it.

We turn now to interactions which involve more than one relation at a time. It is in these interactions that QUEL is especially useful because of its ability to connect information in different relations.

First we print a second relation that will be used in the sequel.

Q7:

```
\r
go
print supp\g
query formulation complete

supp relation
|snum  |pnum  |jnum  |shdt     |quan  |
|-------------------------------------|
|     8|     1|  1003|74-12-31|      1|
|     8|     1|  1004|75-01-15|      1|
|     8|     1|  1007|76-02-01|      1|
|     8|     2|  1003|74-12-29|    128|
|     8|     2|  1004|75-01-15|    256|
|     8|     2|  1007|76-02-01|   1024|
|     8|     6|  1003|74-12-25|      2|
|     8|     6|  1004|75-02-01|      4|
|     8|     8|  1004|74-12-20|      5|
|     8|     9|  1004|74-12-31|    500|
|     8|    11|  1004|75-01-01|      2|
|     8|    11|  1007|76-02-01|      3|
|     8|    12|  1004|75-04-31|      3|
|     8|    12|  1007|76-02-01|      2|
|     9|     5|  1004|75-02-05|    400|
|    20|     5|  1001|75-01-10|     20|
|    20|     5|  1002|75-01-10|     75|
|    41|     5|  1003|75-01-02|     50|
|    62|     3|  1002|74-06-18|      3|
|    67|     4|  1005|75-07-01|      1|
|    67|     5|  1005|75-07-31|     20|
|   122|     7|  1003|75-02-01|    144|
|   122|     7|  1004|75-02-01|     48|
|   122|     9|  1004|75-02-01|    144|
|   131|     8|  1001|75-03-15|      2|
|   131|     8|  1002|75-03-15|      1|
|   131|     8|  1004|74-11-22|      4|
|   131|     9|  1001|75-04-31|    200|
|   131|     9|  1002|75-03-31|    100|
|   440|     6|  1001|74-10-10|      2|
|   475|     1|  1001|73-12-31|      1|
|   475|     1|  1002|74-07-01|      1|
|   475|     2|  1001|73-12-31|     32|
|   475|     2|  1002|74-05-31|     32|
|   475|      3|  1001|73-12-31|      2|
|   475|      4|  1002|74-05-31|      1|
continue
```

This relation gives information on conditions under which the hypothetical auto parts store can buy more parts. It indicates the supplier number (snum) from whom each part (pnum) is available, the quantity (quan) in which it can be ordered, the date (shdt) such an order could be shipped and the job number (jnum) to which such an order could be charged. Notice that the column pnum appears in both the parts relation and this relation. Using this information we can "connect" the two relations. For example, we might want to know the supplier numbers of suppliers who sell antifreez.

One way to proceed is to interrogate the parts relation to find the part number of antifreez as follows:

Q8:

```
\r
go
range of p is parts
retrieve (p.pnum) where p.pname = "antifreez"
\g
```

The answer returned is:

```
query formulation complete
|pnum  |
|------|
|     1|
continue
```

Hence, part number 1 is antifreez. Then we could interrogate the supply relation seeking the suppliers of part number 1 as follows:

Q9:

```
\r
go
range of s is supp
retrieve (s.snum) where s.pnum =1
\g
query formulation complete
|snum  |
|------|
|     8|
|     8|
|     8|
|   475|
|   475|
continue
```

Notice that suppliers 8 and 475 supply antifreez.

Notice also that suppliers 8 and 475 are repeated more than once. Because of the internal way that INGRES is organized, much faster response time can be supported if the "answer" is printed on the terminal with duplicate values sometimes present. In this case, the user must look at the response and note the duplications. On the other hand, should the user wish the system to detect and delete the duplicates, the user need only retrieve his answer into a temporary relation and then print that relation. The instructions are the following:

```
\r
go
range of s is supp
retrieve into newsupp(s.snum) where s.pnum = 1
print newsupp
\g
query formulation complete

newsupp relation

|snum  |
|------|
|     8|
|   475|
continue
```

In any case, it is rather inconvenient to have to issue two retrieve commands to get the information we require.

What is even more inconvenient is the necessity of obtaining the first output, namely the number 1, and then manually substituting this into the second query. It would have been extremely inconvenient if antifreez had had several part numbers; we would have had to substitute them all.

Whenever you are in doubt concerning the meaning of a query with more than one variable in it, always think of the two step process described above and you will not go wrong. With this in mind, convince yourself that the correct answer (with duplicates) to our interaction above can also be found using the following code.

Q10:

```
\r
go
range of s is supp
range of p is parts
retrieve (s.snum) where s.pnum=p.pnum  and p.pname="antifreez"
\g
```

So far in this document we have considered how to retrieve portions of a relation (or relations) that are of interest. The examples have indicated the power of QUEL for retrieval purposes. The only feature which has not yet been considered is aggregation.

We now illustrate the use of this construct in two examples. The following command finds the number of part names from the parts relation which are black.

Q11:

```
\r
go
range of p is parts
retrieve (total= count(p.pname where p.color = "black"))
\g

query formulation complete
|total |
|------|
|     4|
continue
```

Notice that the 'where' clause here is inside the parentheses surrounding the target list. This is because we are using the 'where' clause to modify the aggregate COUNT in the target list. If we wished a further qualification on the entire query, we would have another 'where' clause at the end.

The next command finds the sum of quantities of part number 2 able to be supplied before October 1, 1975.

Q12:

```
\r
go
range of s is supp
retrieve (s = sum(s.quan where s.pnum=2 and s.shdt<"75-10-1"))
\g
query formulation complete
|s     |
|------|
|   448|
continue
```

The following points should be noted about aggregates:

a) aggregates have the form

agg-op(target-list where qualification).

agg-op can be

```
    min
    max
    count
    sum
    avg  (sum/count)
```

The target list is the quantity for which the aggregate is desired using those tuples which satisfy the qualification.

b) There is no limit on the number of variables which can appear in an aggregate.

c) Aggregates can be nested, i.e. the target list and qualification may themselves contain aggregates.

d) The "QUEL" section of the reference manual indicates certain illegal aggregations. For example, avg is only allowed for quantities which are numeric. An attempt to find the average of a quantity that is alphanumeric (for example pname) will result in an error.

e) An aggregate can appear anywhere in a QUEL interaction.

Another command which proves useful is the exit command which is "\q", i.e.

```
\r
go
\q
```

This command will type a friendly greeting on your terminal and return you to the care of UNIX for any further processing you may wish to do. The current greeting is the following:

```
query formulation complete
INGRES vers 4.0 logout
Tue Aug 26 20:05:22 1975
goodbye - come again
```

The only other way to "bail out" of INGRES is to hit the "rub out" key. This should only be used in emergency (for example to abort a printout which is much too long). It has the effect of returning you directly to UNIX.

Appendix E - CUPID Tutorial

This document contains an introduction to the data base management system, INGRES, and in particular stresses one of its user language CUPID, the Casual User Pictorial Interface Design. You will be retrieving information from a database via query-diagrams. To "draw" such a diagram, you merely select the appropriate symbols from a ´menu´ of pre-drawn symbols ; connect them by either juxtoposition or connector lines; fill in the table and column names through more ´menu´ selection; and type in constant value when needed.

This manual is meant to be read while interacting with the INGRES system at a DEC GT42 graphics terminal. It will be assumed that you are sitting in front of an already invoked version of CUPID. If so, a welcoming message should appear on the screen (the invoking procedure will be described in another document).

GRAPHIC TERMINOLOGY

Some graphic terminology is appropriate at this time. The lightpen (abbreviated -LTPEN) is the silver-colored pen-like instrument hanging to the right of the screen. CUPID is designed to necissitate a minimal amount of typing (or key-board interaction); therefore, most of your actions will utilize the lightpen. The word HIT is used to mean a lightpen selection. There are two types of HITs:

1 Touching the item on the screen with the tip of the lightpen. Since the lightpen is light-sensitive, you can select a word, line, or symbol in this way.

2 Placing the lightpen on the tracking-cross(8-pointed star-like figure on the screen) and dragging the tracking-cross(t-c) to the place on the screen you wish to select, then depressing the CTRL key and while holding that key down, type an "a"(CTRL-a).

REMEMBER: You can use either method 1 or 2 to select a lighted part of the screen; but, you must use method 2 to select a blank position.

Both of these operations will result in a beep from the machine and the appearance of a small cross very near the hit position which indicates the hit was registered. As soon as you hear the beep, pull the lightpen away from the screen (if you are using the t-c, you may retract the LTPEN as soon as the t-c is in place, even before depressing the CTRL-a).

Please notice the screen configuration. There is an area circumscribed with a large rectangle (the welcome message is inside). Most of your actions will take place within the rectangle, instructions will appear above the rectangle, and control commands will appear to the right of the rectangle. You will be hitting commands also.

LTPEN Exercise

Now try out the lightpen. Touch LTPEN to the screen (gently) where you see the word HIT 1. It will blink if your hit was accurate. If it doesn't blink try again or drag the t-c on top of HIT 1 and type CTRL-a. When it blinks, drag the t-c to a blank portion of the screen (within the rectangle) and depress CTRL-a. Now you should see HIT 2 and you can try to hit it, make it blink, then place a new target elsewhere. When either you or IT have had enough, hit the command HELP.

NOTE: If you happen to make a double hit (your new target will overwrite the blinking one) just proceed as though the old one were not there. But be aware of the possibility of double hits due to the lightpens sensitivity. If this annoys you, turn the brightness (knob at the upper right) down.

HELP

Now you see before you the screen set-up for drawing your queries. You will be selecting names at the bottom left portion of the screen(name-space); you will be selecting symbols(the BOXes and HEXes ,etc.) from the bottom right(symbol-space); and you will be putting them together into a query in the top half of the screen(query-space).

We shall work with only a few of the help commands Hit ALL to see information about all tables in the database ("nancy" is this database). When you have finished reading the information, hit RETURN to get back into the help-mode.

Now hit EXAMPLE to see some example queries. Page through several examples with the MORE command.

Note the form of the query in CUPID. Hexagons (containing table names) are abutted to the left of boxes (containing column names). The vertical diamonds with "?" inside are connected by lines to the items (target items) you wish to see displayed; horizontal hexagons are reserved for typing in values. The operators (arithmetic-squares, relational-horizontal diamonds) are straight forward. Aggregation operators (pentagons) are explained in more detail later. Please note that only the HEXes and BOXes are juxtoposed, all other symbols are connected via specific lines (connectors).

When you are finished perusing the examples, hit RETURN to return to the HELP mode.

Once back in the HELP mode, hit CONTINUE to proceed into the query formulation phase.

Query formulation consists of two phases:

1) Table Selection

2) Query Drawing

TABLE SELECTION

You start the query formulation by selecting the tables you wish to use. Hit "parts" and "supp". The selected table names will appear at the bottom of the screen as you select them. If you get any names other than the two desired (or multiple copies), hit the command REMOVE, then hit the table name to be removed in the selected area at the bottom of the screen (it will be scratched out). You must hit the command REMOVE for each name you wish to delete.

Hit CONTINUE to proceed to Query Drawing.

QUERY DRAWING

The commands that you see on the right side of the screen are:

CONNECTOR: For each connecting line between symbols, hit CONNECTOR, then hit one symbol's attacher point(i.e. the short line extending from the symbol), then the second symbol's attacher point. If you wish to go around a symbol, or merely draw a "curved" connector, continue making hits as you wish the line to "curve". To draw 2 different connectors(i.e. disconnected lines), you must hit the command CONNECTOR before beginning the second line.

ERASE: This routine will redraw the query (eliminating the bottom half of the screen). It will ask what you wish to erase (Connector, Name, or Symbol); you hit the appropriate word on the right; the screen is redrawn; you hit the item to be erased:

for connectors- hit the first drawn endpoint

for names- hit the lower left point of the first character

(NAME refers to any string of characters- i.e. table or column names or constant values)

for symbols- HEX and BOX, hit upper left corner; all others, hit the center of the symbol;

the item hit should blink(keep trying until it does); then hit the command ERASE and the screen will be redrawn without the erased item; either hit RETURN to return to complete the query or hit one of the three types of items to be erased next.

HELP: The HELP command allows one to obtain help while drawing a query.

REJECT: To reject an item (either a name or symbol) after selecting it but before placing it in the query space (after placing you must ERASE), hit REJECT; then the item to be rejected; then proceed.

CONSTANT: To place a constant in a CONS (horizontal hex), hit CONSTANT, then type the value and a carriage return; then place the value inside the CONS-hex; then proceed.

FINISH; Hit FINISH to process query. A new screen configuration will appear in anticipation of the answer to your query. After hitting FINISH either

1. the query will be processed and the answer will appear

2. error messages will tell you the type of pictorial syntax error

you made

3. you may be asked to "define" the constant value used The three commands you will be hitting are: ALTER to change some aspect(s) of the query just drawn; REDRAW to draw a totally new diagram; and DEFINE to enter the define phase for defining a constant value (to be discussed in another document).

REDRAW: The REDRAW command will clear the query space to allow you to draw a new query.

QUIT: Hit this command to return to the first screen configuration of CUPID. This will allow you to redo the TABLE SELECTION phase or exit entirely.

Before beginning to issue queries there are a few points of procedure.

1 HEXagons must be placed in the query space before BOXes which must be next to the HEXes. When placing a BOX next to a HEX or another BOX, point to the upper right

corner of the existing item.

2 HEXes and BOXes and CONSes must be on the picture before selecting and placing either names or constantsto be put in them.

3 A symbol must be placed in the query space before drawing a connector to it. Connectors must be drawn to and from attacher points. There are from 2 to 4 short lines which jut out of each symbol; the free end of these lines are the attacher points.

Several examples follow.

In this first query, you will be guided in a step by step manner, however, in all other queries, the completed picture will be shown to you and you will be expected to reproduce it in any manner.

Q1: Retrieve the entire parts table.

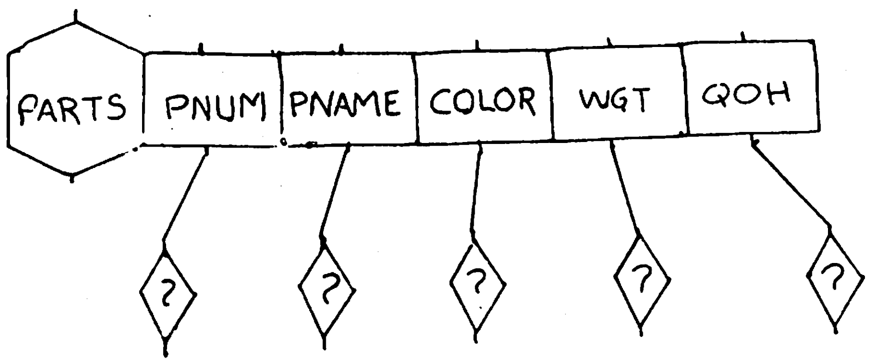
First select a HEX with the LTPEN by either method described earlier The HEX will flash. Now bring the t-c to a clear part of the query space and type CTRL-a. The HEX should appear there. (If this doesn't happen, try again -or hit the ERASE command, then hit the RETURN command without erasing --this may help. Electronic problems sometimes make this messy. Do not get frustrated.)

Select a BOX similarly. It will flash; bring the t-c up to the upper right corner of the HEX; type CTRL-a; the BOX should now appear beside the HEX. Repeat this process until you have 5 BOXes attached together and to the right of the single HEX.

Now fill in the table and columns names. Point the LTPEN at the table name- PARTS; it will flash; bring the t-c to point inside the HEX; type CTRL-a; the name-PARTS should then appear inside the HEX. If this doesnot happen try moving it by moving the t-c and typing CTRL-a again. Continue placing all of the column names in BOXes, one to a BOX.

Select the "?"-symbol to indicate which columns you wish to see. Bring the t-c to a place in the top half of the screen convenient to connect the "?" from the first BOX; type CTRL-a. Proceed selecting and placing until you have 5 "?"´s on the screen

Connect one BOX to one "?" by hitting the command CONNECTOR; hit one endpt of the connector (an attacher point of either the BOX or "?") The line should appear to connect the symbols completely. Repeat this procedure 5 times. Your picture should look like:



Now hit the command FINISH.

The result should be:

```
|pnum  |pname                |color    |wgt    |qoh    |
|------------------------------------------------------|
|     1|antifreez            |pink     |     10|      1|
|     2|wrench               |gray     |     20|     32|
|     3|tires                |black    |    685|      2|
|     4|ash-tray             |black    |    450|      4|
|     5|oil                  |gray     |      1|    250|
|     6|chamois              |yellow   |    578|      3|
|     7|ornament             |white    |     15|     95|
|     8|seatcover            |blue     |     19|     15|
|     9|race-strp            |white    |      2|    350|
|    10|wash-solv            |clear    |      0|    143|
|    11|jacks                |gray     |    327|      0|
|    12|chrome               |gray     |    427|      0|
|    13|tape-play            |black    |    107|      0|
|    14|radio                |black    |    147|      0|
continue
```

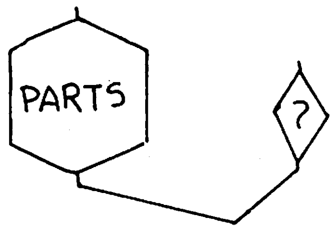
NOTE: The word continue and a beep will always follow the response. Please wait until this occurs before drawing your next query.

Notice that the "parts" table has information about the components in a hypothetical auto parts supply store. Each row of this table (or tuple in this relation) contains information on a given part including its part number (pnum), its part name (pname), its color, its weight (wgt), and the quantity that are on hand (qoh).

Now you are ready to draw the next query. Hit ALTER to correct, change or add to the last query drawn; hit REDRAW to draw a picture from scratch; hit QUIT to get to the first screen configuration (this will allow you to proceed from the beginning and step through the TABLE SELECTION phase again). The DEFINE command presently acts like the QUIT command.

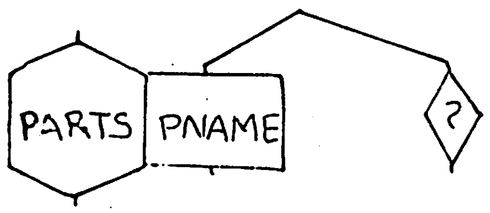
Hit REDRAW.

Q1.5 A short-hand diagram for that query is:



Using a CUPID picture we will be able to obtain portions of this table which are of interest to us. (There is almost no limit on how large the tables can be; these examples are done on small ones so that this tutorial does not become too large. In fact, the actual limit on the size of a table is approximately 30,000,000 bytes for those who are interested.)

Q2: Retrieve PNAMEs out of the PARTS table.

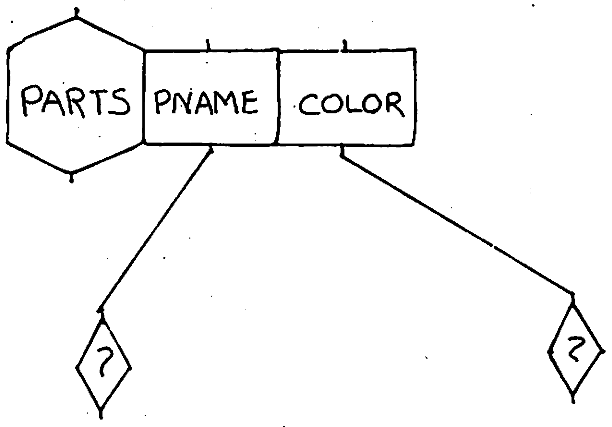


```
|pname               |
|--------------------|
|antifreez           |
|wrench              |
|tires               |
|ash-tray            |
|oil                 |
|chamois             |
|ornament            |
|seatcover           |
|race-strp           |
|wash-solv           |
|jacks               |
|chrome              |
|tape-play           |
|radio               |
continue
```

Everything has worked correctly and we have obtained the column of the parts table which contains the names of the parts.

We can retrieve more than one column at once(as in Q1) by simply indicating a sequence of boxes containing column names with attached "?" diamonds. At this point you may either hit the command ALTER or REDRAW.

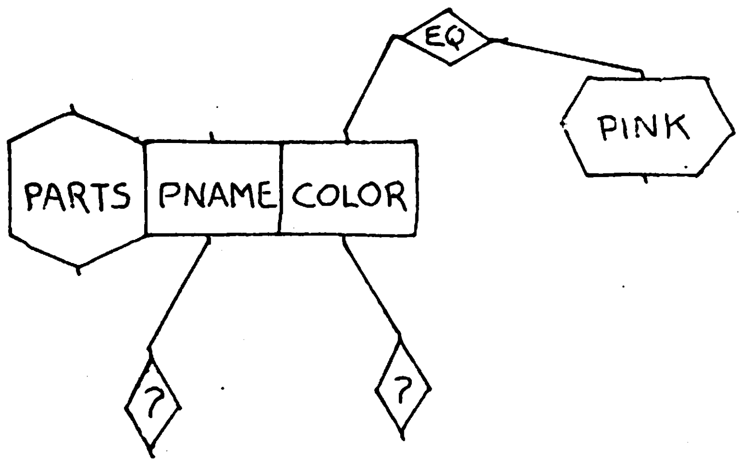
Q3: Thus, to obtain part names and colors we draw:



```
|pname                    |color   |
|--------------------------------- |
|antifreez                |pink    |
|wrench                   |gray    |
|tires                    |black   |
|ash-tray                 |black   |
|oil                      |gray    |
|chamois                  |yellow  |
|ornament                 |white   |
|seatcover                |blue    |
|race-strp                |white   |
|wash-solv                |clear   |
|jacks                    |gray    |
|chrome                   |gray    |
|tape-play                |black   |
|radio                    |black   |
continue
```

So far we have produced interactions which give us columns of the "parts" table. We now indicate how to obtain only portions of columns. The basic mechanism is to connect appropriate limiting operators and any constant values to the BOXes targeted with "?"-symbols.

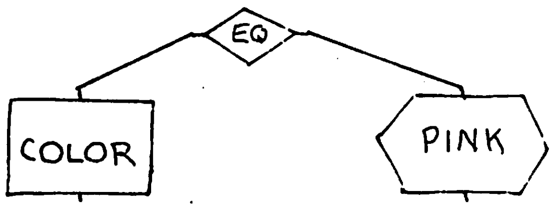
Q4: If we wanted the previous query only performed for those parts whose color is pink we would do the following:



An explanation of how to place the word - pink - is in the following NOTE on the next page.

```
|pname                      |color  |
|-----------------------------------|
|antifreez                  |pink   |
continue
```

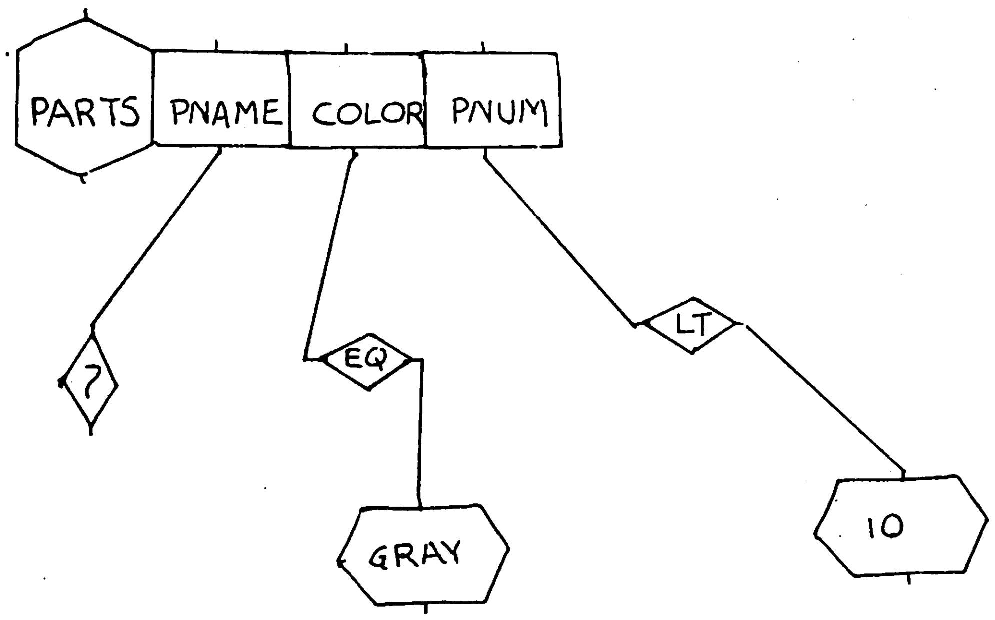
These limiting operators reduce the number of rows which are examined to only those which satisfy the qualification given i.e. to those which satisfy the part of the query not singled out with "?" -or:



Only antifreez has this property so it is the only entry in the output.

NOTE: To place the word 'pink' in the CONS-box; hit the command CONSTANT; type - pink - and a 'cr'(carriage return); now place the t-c inside the CONS and depress CTRL-a. The first letter of every constant value must be entirely in the symbol.

Q5: Obtain the pnames of parts which are gray and whose pnum is less than 10.



```
|pname               |
|--------------------|
|wrench              |
|oil                 |
continue
```

Three points should be carefully noted about the above interaction:

1) Note the relational operator LT in the above interaction. Valid relational operators include:

EQ (equals to)

LT (less than)

GT (greater than)

LT EQ (less than or equal to-not implemented)

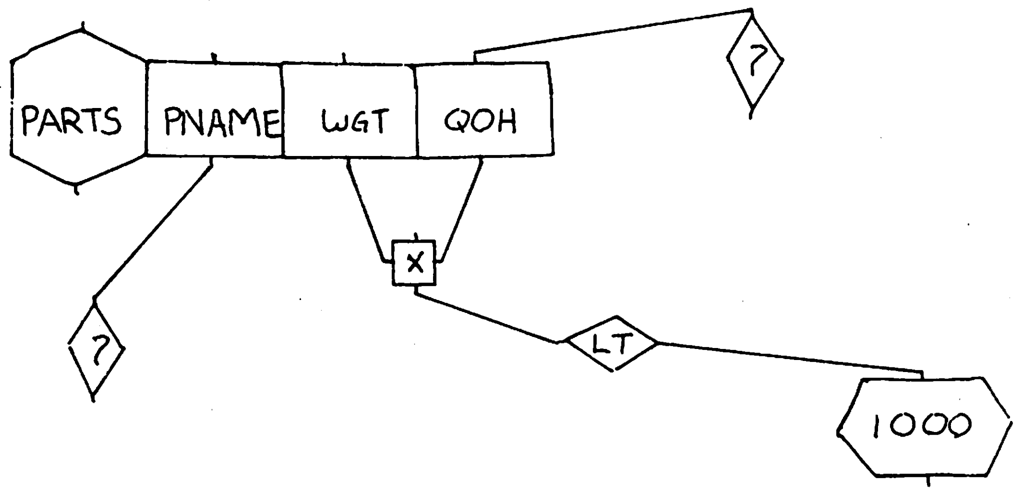
GT EQ (greater than or equal to-not implemented)

NE (not equal to)

2) There is no limit to the complexity of the expressions which can be consructed using relational and boolean expressions, theoretically since the present implementation is restricted.

3) All of the relational operators except EQ and NE are ordered operators. This means that whatever is connected on the left is the first operand and the item connectec on the right is the second.

Q6: We now do an example concerning arithmetic operators in CUPID. This example finds pname and qoh of parts whose total weight (wgt times qoh) is less than 1000.



```
|pname                   |qoh  |
|----------------------------- |
|antifreez               |    1|
|wrench                  |   32|
|oil                     |  250|
|seatcover               |   15|
|race-strp               |  350|
|wash-solv               |  143|
|jacks                   |    0|
|chrome                  |    0|
|tape-play               |    0|
|radio                   |    0|
continue
```

It should be noted that arithmetic operators can be used in the target list ("?") portion of an interaction as well as in the qualification portion

Note also that any operations (even a "?"-symbol) you may wish done to the result of an arithmetic operation is 'hung' off of the operator [as in the picture part meaning less than 1000].

Valid arithmetic operators include:

$\boxed{+}$ (addition)

$\boxed{-}$ (subtraction)

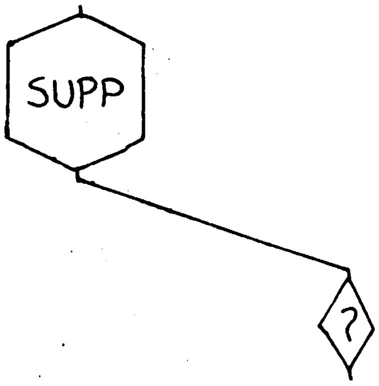
$\boxed{\times}$ (multiplication)

$\boxed{/}$ (division)

$\boxed{**}$ (exponentiation)--not implemented

We turn now to interactions which involve more than one table at a time. It is in these interactions that CUPID is especially useful because of its ability to connect information in different tables.

Q7: First we print a second table that will be used in the sequel.



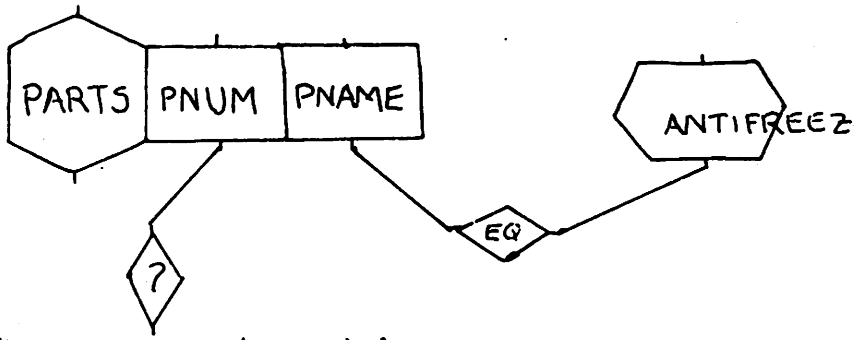
supp relation

snum	pnum	jnum	shdt	quan
8	1	1003	74-12-31	1
8	1	1004	75-01-15	1
8	1	1007	76-02-01	1
8	2	1003	74-12-29	128
8	2	1004	75-01-15	256
8	2	1007	76-02-01	1024
8	6	1003	74-12-25	2
8	6	1004	75-02-01	4
8	8	1004	74-12-20	5
8	9	1004	74-12-31	500
8	11	1004	75-01-01	2
8	11	1007	76-02-01	3
8	12	1004	75-04-31	3
8	12	1007	76-02-01	2
9	5	1004	75-02-05	400
20	5	1001	75-01-10	20
20	5	1002	75-01-10	75
41	5	1003	75-01-02	50
62	3	1002	74-06-18	3
67	4	1005	75-07-01	1
67	5	1005	75-07-31	20
122	7	1003	75-02-01	144
122	7	1004	75-02-01	48
122	9	1004	75-02-01	144
131	8	1001	75-03-15	2
131	8	1002	75-03-15	1
131	8	1004	74-11-22	4
131	9	1001	75-04-31	200
131	9	1002	75-03-31	100
440	6	1001	74-10-10	2
475	1	1001	73-12-31	1
475	1	1002	74-07-01	1
475	2	1001	73-12-31	32
475	2	1002	74-05-31	32
475	3	1001	73-12-31	2
475	4	1002	74-05-31	1

continue

This table gives information on conditions under which the hypothetical auto parts store can buy more parts. It indicates the supplier number (snum) from whom each part (pnum) is available, the quantity (quan) in which it can be ordered, the date (shdt) such an order could be shipped and the job number (jnum) to which such an order could be charged. Notice that the column pnum appears in both the parts table and this table. Using this information we can "connect" the two tables. For example, we might want to know the supplier numbers of suppliers who sell antifreeze.

08: One way to proceed is to interrogate the parts table to find the part number of antifreeze as follows:

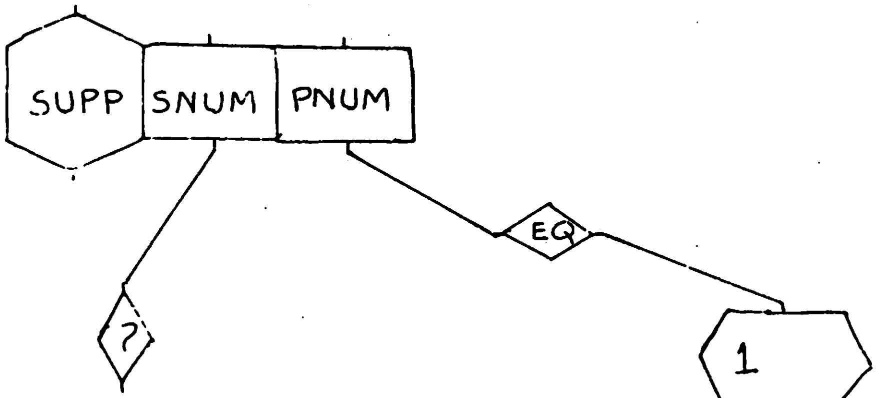


The answer returned is:

```
|pnum   |
|-------|
|      1|
continue
```

Hence, part number 1 is antifreeze.

09: Then we could interrogate the supply table seeking the



suppliers of part number 1.

```
|snum  |
|------|
|     8|
|     8|
|     8|
|   475|
|   475|
continue
```

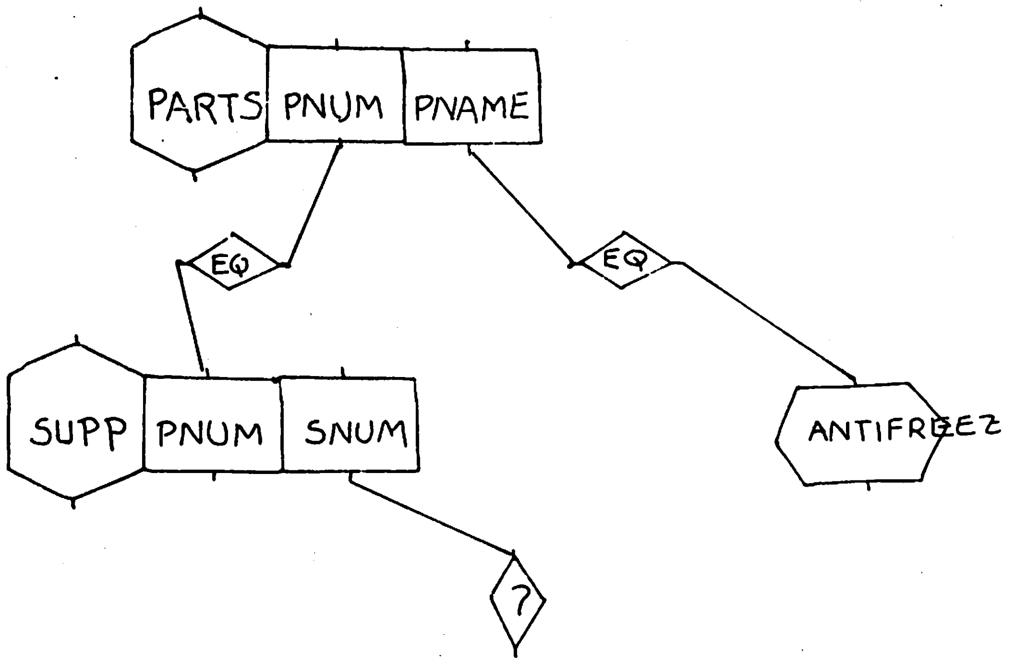
Notice that suppliers 8, 475 supply antifreeze.

Notice also that suppliers 8 and 475 are repeated more than once. Because of the internal way that INGRES is organized, much faster response time can be supported if the "answer" is printed on the terminal with duplicate values sometimes present. In this case, the user must look at the response and note the duplications. On the other hand, should the user wish the system to detect and delete the duplicates, the user need only retrieve his answer into a temporary relation and then print that relation. At this time the appropriate protocol for CUPID has not been implemented. We must live with duplicates.

It is rather inconvenient to have to issue two query formulations to get the information we require.

What is even more inconvenient is the necessity of obtaining the first output, namely the number 1, and then manually substituting this into the second query. It would have been extremely inconvenient if the antifreeze had had several part numbers; we would have had to substitute them all.

Q10: The following indicates one way around this inconvenience.



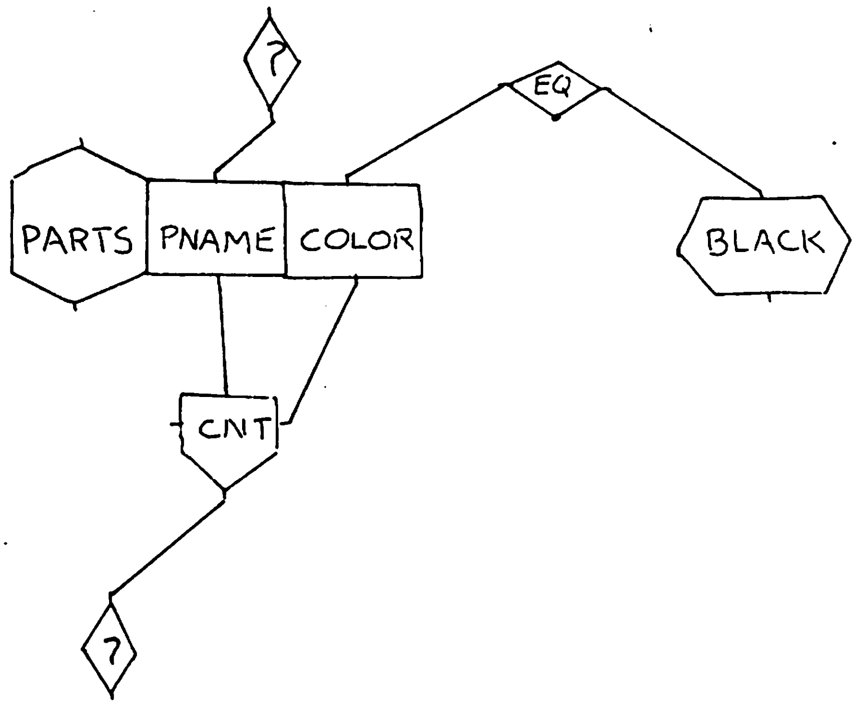
Resulting in the following:

```
|snum  |
|------|
|     8|
|     8|
|     8|
|   475|
|   475|
continue
```

So far in this document we have considered how to retrieve portions of a table (or tables) that are of interest. The examples have indicated the power of CUPID for retrieval purposes. The only feature which has not yet been considered is aggregation.

We now illustrate the use of this construct in two examples.

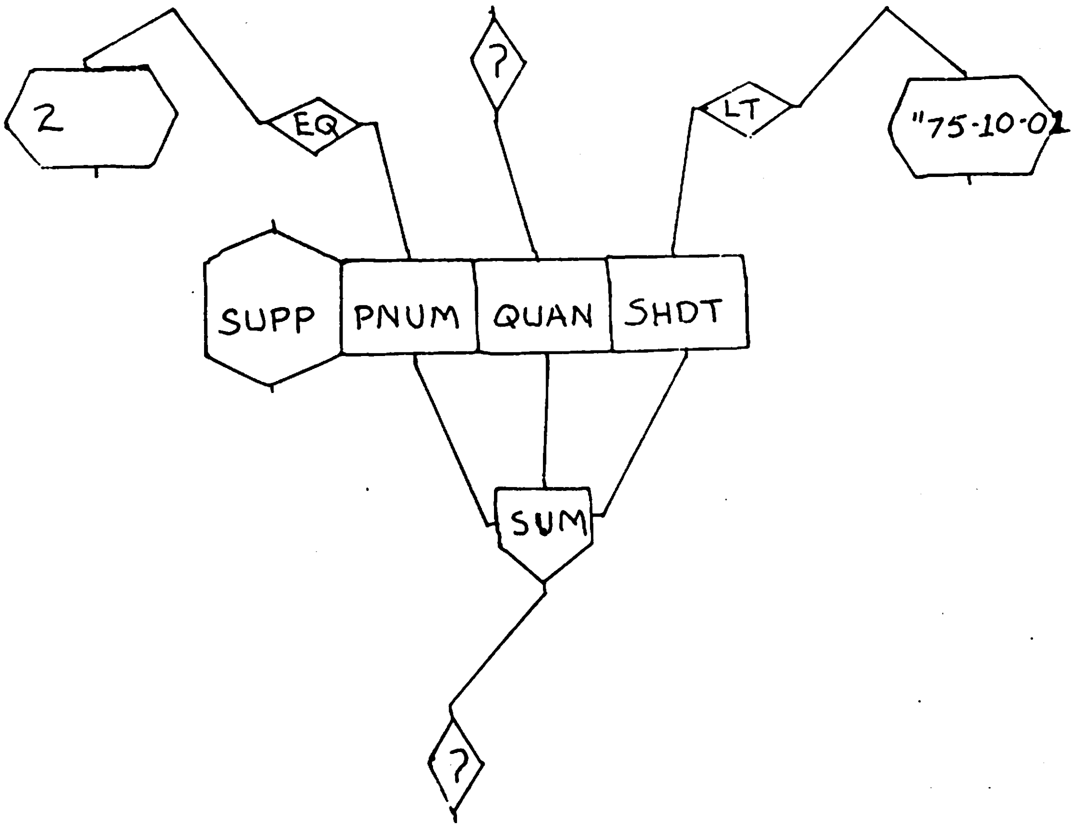
Q11: The following command finds the number of part names from the parts table which are black.



```
|c      |
|-------|
|      4|
continue
```

You should note that there are two "?"-symbols and only one column of result. This is due to information necessary to perform aggregation. Each aggregate operator needs a targeted ("?"-symbol attached) BOX. The aggregate may also need some untargetted #BOXes (known as qualification). Notice the line connecting the aggregate operator CNT to the box containing COLOR. This indicates that the qualification of 'color = black' modifies the aggregate and not the whole query(if the connector where not there). Without that connecting line, we would have obtained the count of PNAMEs for each time the color of an item is black - or the result would have shown 14 listed 4 times. You might like to try this query also. If so, just erase the connector and reissue the query.

Q12: The next command finds the sum of quantities of part number 2 to be supplied before October 1, 1975.



NOTE: Due to the special type of information in column SHDT (dates) you must type "75-10-01 with the first character being " .

```
|s      |
|-------|
|    448|
continue
```

The following points should be noted about aggregates:

a) aggregate operators include:

min	minimum
max	maximum
cnt	count
sum	sum
avg	average(sum/count)

b) There are illegal aggregations. For example, avg is only allowed for quantities which are numeric. An attempt to find the average of a quantity that is alphanumeric (for example pname) will result in an error.

c) An aggregate can appear anywhere in a CUPID interaction.

d) Aggregate operators may be (1) attached at one end only - meaning any qualification does not refer to the aggregate; or (2) attached to boxes at several points - meaning the box not marked with a "?" starts the qualification referring to the aggregate only.

e) The resulting column heading is a CUPID generated heading using letters of the aggregate operator's name.

Appendix F - Test queries

Now you are ready to test your skills. Try to formulate the following queries and retrieve the appropriate information.

Try your best, but do not worry if you can not do some. Give yourself a maximum of three attempts at each query, then go on to the next. Remember, this is not a test of you---we are testing the systems.

The two tables are briefly described in the EXPLANATION page.

1 List the whole EMPL table.

2 Display the NAMEs from EMPL table whose SALs are greater than 10000.

3 Retrieve the COUNT of NAMEs in the EMPL table.

4 Get the NAMEs out of EMPL who are in a DEPT in the DEPT table
which is on FLOOR= 2 (link via MGR).

5 Find the MGR out of EMPL whose NAME is associated with a SAL greater than 10000.

6 List all DEPTs whose FLOOR is less than 3.

7 Find the SAL of Choy,W. after multiplying it by 2 .

8 Show DEPTs and their respective FLOORs.

9 Get people who work for Thomas,T.

10 Find the maximum salary .

11 Find the floor where Evans,M. works .

12 List those employees who make more than their managers.

13 Where are earrings sold?

14 How many people work in store 7 ?

Appendix G - Questionnaire

QUESTIONNAIRE

1 Which language did you like better?

QUEL_____ CUPID_____

2 Was QUEL's tutorial clear?

Yes_____ No_____

If NO, please detail:

3 Was CUPID's tutorial clear?

Yes_____ No_____

If NO, please detail:

4 Were the instructions for the experiment clear?

Yes_____ No_____

5 Have you ever used this kind of terminal (CRT) before?

Yes_____ No_____

6 Did you have any difficulties with this terminal?

Yes_____ No_____

If yes explain further

7 What aspects of QUEL did you like-

What aspects of QUEL did you dislike-

8 What aspects of CUPID did you like-

What aspects of CUPID did you dislike-

9 Circle the area in which you excel (based on SAT scores, grades, or some comparable measure).

ENGLISH and related subjects

ALL subjects

MATH and related sciences

10 How long have you lived in U. S.

All your life_____ No. of years______

11 How old are you? ________ yrs.

12 How many computer science courses have you taken?

number of courses________

13 How much do you know about data base systems?

none_____ a little______ know INGRES well______ (or other)

14 What programming languages are you familiar with?

none _____ list _________

____

____

____

15 Sex: female_______

male _______

Appendix H - Login procedures

The following two procedures detail the steps involved to bring up both parts of the CUPID system. They need not be done in this order, but both parts must be in operation before any queries are issued. Whatever follows type: is to be typed and followed by a "cr" [carriage return]. That following ltpen: is to be invoked by taking a hit with the lightpen. All else are system responses. Comments by this author are bracketted by [].

1 The UNIX-INGRES language processor.

```
[log into UNIX]

type:  cupid nancy   [or-"cupidrun nancy" to get a log]

Your database is nancy

go

[please wait until the "go" before proceeding]
```

2 The PTSS-PICASSO picture processor.

```
[log onto the CDC machine]

OK-^EDIT

type: ^load,logcup
```

```
LOAD COMPLETE, ENTERING ^EDIT

OK-^EDIT

type:  ^run

BEGIN EDIT

type:  cupid;r

NOW TYPE <SEMICOLON>G TO COMPILE AND EXECUTE

type:  ;g

[be patient, this may take a few minutes]

PICASSO  screen configuration will appear

ltpen:  hit  USER COM   [lower right]

[now you should see the CUPID welcoming message]
```

The following two procedures will exit you from both parts of the CUPID system. The conventions are the same as above.

1 The UNIX-INGRES language processor.

```
type:  "rub out"   [hit the key marked "rub out"]

interrupted

%
```

[you are now out of INGRES and in UNIX]

type: "CTRL-d" [depress the key marked "CTRL" and while holding it down, type a d]

[you are now off the UNIX-INGRES system entirely]

2 The PTSS-PICASSO picture processor.

[if you do not see the CUPID welcoming message on the screen]

ltpen: hit QUIT [until you see the welcoming message]

ltpen: hit QUIT

PICASSO screen configuration appears

ltpen: hit FINISHED

FINISHED screen configuration appears

type: skip

BEGIN EDIT

type: quit;r

NOW TYPE <SEMICOLON>G TO TERMINATE JOB

type: ;g

References

BOYC73 Boyce, R. & et. al., "Specifying Queries as Relational Expressions: SQUARE," IBM Research, San Jose, Ca., RJ 1291, Oct. 1973.

CHAM74 Chamberlin, D. & Boyce, R., "SEQUEL: A Structured English Query Language," Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1974.

CODD70 Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," CACM, Vol. 13 No. 6, pp. 377-387, June, 1970.

CODD71 Codd, E.F., "A Data Base Sublanguage Founded on the Relational Calculus," Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, CA, Nov. 1971.

CODD72 Codd, E.F., "Relational Completeness of Data Base Sublanguages," Courant Computer Science Symposium 6, May 1972.

CODD74 Codd, E.F. & Date, C.J., "Interactive Support for Non-Programmers, The Relational and Network Approaches," Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1974.

CODD74a Codd, E.F., "Seven Steps to Rendevous with the Casual User," Proc. IFIP TC-2 Working Conference on Data Base Management Systems, Cargese, Corsica, Apr. 1974.

COLE69 Coles, L.S., "An On-Line Question-Answering System with Natural Language and Pictorial Input," Proc. ACM 23rd Natl. Conf., 1969.

DATE74 Date, C.J. & Codd, E.F., "The Relational and Network Approaches: Comparison of the Aplication Programming Interfaces," Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, Mich., May 1974.

EVAN69 Evangilisti, C.J. & Morse, S.P., "Graphical Modelling using Contextually Implied Functions," personal communication.

HELD75 Held, G.D. & Stonebraker, M., "Storage Structures and Access Methods in the Relational Data Base Management System, INGRES," Proc. ACM-Pacific 75 Conf., Apr. 1975.

HELD75a Held, G.D. & Stonebraker, M. & Wong, E., "INGRES - A Relational Data Base Management System," Proc. 1975 NCC, AFIPS Press, 1975.

HOLM75 Holmes, H. H., "Graphics Modeling Techniques in Computer Aided Design," Ph.D. Thesis, EECS Dept., University of California, Berkeley, Dec., 1975.

JOHN74 Johnson, S.C., "YACC, Yet Another Compiler-Compiler," UNIX Programmer's Manual, Bell Telephone Labs, Murray Hill, N.J., July 1974.

LAWR74 Lawrence Berkeley Laboratory Computing Facility, "BKY - Users Introduction," Internal Documentation, LBL, April 26, 1974.

MART73 Martin, J. , "Design of Man-Computer Dialogues," Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.

MCDO74 McDonald, N. & Stonebraker, M. & Wong, E., "Preliminary Specification of INGRES," University of California, Electronics Research Laboratory, Memorandum No. M435-436, April 1974.

QUIL66 Quillian, M.R., "Semantic Memory," Ph.D. Thesis, Carnegie-Mellon Univ., Pittsburgh, Pa., Feb., 1966.

REIS75 Reisner, P. & Boyce, R.F. & Chamberlin, D.D. , "Human factors," Human factors evaluation of two data base query languages-Square and Sequel

RITC74 Ritchie, D.M., "C Reference Manual," UNIX Programmer's Manual, Bell Telephone Labs, Murray Hill, N.J. July 1974.

RITC74a Ritchie, D. & Thompson, K., "UNIX Programmer's Manual," Bell Telephone Labs, Murray Hill, N.J. June 1975

SACK70 Sackman, H. , "Man-Computer Problem Solving," Auerbach Publishers, Princeton, New Jersey, 1970.

SHNE74 Shneiderman, B. & Ho, Mao-Hsia, "Two Exploratory Experiments in Program Comprehension," Technical Report No. 27, Computer Science Dept., Indiana University, 1974.

SIME73 Sime, M.E. & Green, T.R.G. & Guest, D.J. , "Psychological Evaluation of Two Conditional Constructions

Used in Programming Languages ," International Journal of Man-Machine Studies, 1973, vol. 5, 105-113.

STON74 Stonebraker, M. & Wong, E., "Access Control in a Relational Data Base Management System by Query Modification," Proc. 1974 ACM National Conference, San Diego, Ca., Nov. 1974

STON75a Stonebraker, M.R., "Getting Started in INGRES - A Tutorial," University of California, Berkeley, ERL Mem. No. ERL-M518, Apr. 1975.

WALS71 Walsh, W.J., "CSMP User's Manual," (unpublished), Univ. of Cal., Berkeley, 1971.

WEIS73 Weissman, L., "Psychological Complexity of Computer Programs: An Initial Experiment," Technical Report CSRG-26, Computer Systems Research Group, University of Toronto, Toronto, Canada, 1973.

WINO71 Winograd, T., "Procedures as a Representation for Data in a Computer Program for Understanding Natural Language," Revised Ph.D. Dissertation, M.I.T., Jan., 1971.

WONG75 Wong, E., "Decomposition - Query Processing in INGRES," Private Communication, May 1975.

WOOD66 Woods, W.A., "Semantic Interpretation of English Questions on a Structured Data Base," Rep. NSF-17, 1967, Computer Lab, Harvard Univ., Cambridge, Mass., Aug., 1966.

YOUN74 Young, E.A. , "Human Errors in Programming," International Journal of Man-Machine Studies, 1974, vol. 6, 361-376.

ZLOO75 Zloof, M.M., "Query by Example," Proc. 1975 NCC, pp. 431-438, AFIPS Press, May 1975.

ZOOK75 Zook, W. et. al., "INGRES - Reference Manual," University of California, Berkeley, ERL Mem. No. ERL-M519, Apr. 1975.

Related Bibliography

APTE70 Apter, M.J., The Computer Simulation of Behavior, Harper and Row, New York, 1970.

CODD71b Codd, E.F., "Normalized Data Base Structures: A Brief Tutorial," Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, CA, Nov. 1971.

DATE75 Date, C.J., An Introduction to Database Systems, Addison-Wesley Publishing Company, Inc., Reading, Mass., 1975.

EARL70 EArley, J., "Toward an Understanding of Data Structures," Proc. 1970 ACM-SIGFIDET Workshop

EARL73a Earley, J., "Relational Level Data Structures for Programming Languages," Computer Science Dept., Univ. of Calif., Berkeley, MArch, 1973.

ENGL72 debate, "English as a Query Language," debate, Proc. ACM Nat. Conf., New York, 1972.

HUMA70 journal, "Human Factors," Vol. 12, No. 2., pp. 165-214, 1970.

KLIN73 Klinger, A., "Natural Language, Linguistic Processing, and Speech Understanding: Recent Research and Future Goals," R-1377-ARPA, Dec., 1973.

KNUT68 Knuth, D.E., The Art of Computer Programming, Vols. 1 and 3, Addison-Wesley, Reading, Mass., 1968.

MACR73 Macri, P., "BUDS: Berkeley Urban Data System," ERL

Tech. Memo M412, University of Cal., Berkeley, Nov., 1973.

NIJS72 Nijssen, G.M., "Common Data Base Languages," Data Base of SIGBDF, Vol. 4, No. 4, Winter, 1972.

NILS71 Nilsson, N.J., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill Book Co., New York, 1971

OLLE69 Olle, T.W. (chm), "The Large Data Base, Its Organization and User Interface," Data Base of SIGBDF, Vol. 1, No. 3, Fall, 1969.

ROTH72 Rothnie, J.B., "The Design of Generalized Data Management Systems," Ph.D. Dissertation, Dept. of Civil Engr., M.I.T., 1972.

ROTH72a Rothman, S. & Mossmann, C., Computers and Society, Science Research AssociAtes, Inc., Chicago, 1972.

SLAG71 Slagle, J.R., Artificial Intelligence: The Heuristic Programming Approach, McGraw-Hill Book Co., New York, 1971.

WEIN71 Weinberg, G.M., The Psychology of Computer Programming, New York, Van Nostrand Reinhold Company, 1971.