AD-A039 783 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
AN ANALYSIS OF DATA BASE QUERY LANGUAGES.(U)
MAR 77 D E LOUGH, A D BURNS

UNCLASSIFIED NL

| OF |
AD
A039783

END
DATE
FILMED
6-77

(2)

FC

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN ANALYSIS OF DATA BASE QUERY LANGUAGES

by

Dennis Elliot Lough
and
Allen Dale Burns

March 1977

Thesis Advisor: L. V. Rich

Approved for public release; distribution unlimited.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	(9)
4. TITLE (and Subtitle) (6) An Analysis of Data Base Query Languages	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; March 1977
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Dennis Elliot Lough Allen Dale Burns	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California	(11) 12. REPORT DATE Mar[illegible] 1977 (12) 13. NUMBER OF PAGES 62
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940	15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

251 450

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

query languages
data sublanguage
data base
procedural
non-procedural
level completeness
learnability

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

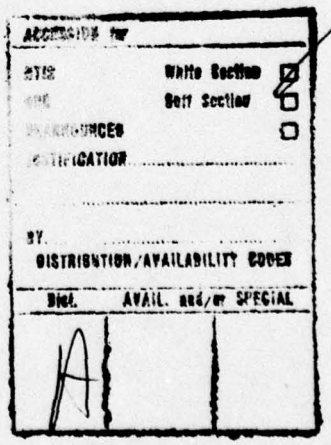
An abundance of Data Base Management Systems and Query Languages already exist, not to mention those which have been, and continue to be proposed. Most Data Base Management System surveys focus on the type of model used to represent the data, methods of access, protection, etc. This paper acquaints the EDP manager with the fundamental differences among the more significant query languages, with emphasis on those → (cont.)

DD FORM 1 JAN 73 1473 (Page 1)
EDITION OF 1 NOV 68 IS OBSOLETE
S/N 0102-014-6601 |

6 08

20. (cont.)

characteristics which should be considered when choosing a query language. The term query language as used here has been expanded to include the entire user interface to the data base, and encompass both data sublanguages and stand-alone query languages.



Approved for public release; distribution unlimited

AN ANALYSIS OF DATA BASE QUERY LANGUAGES

by

Dennis Elliot Lough
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1965

Allen Dale Burns
Lieutenant, United States Navy
B.S., University of Maryland, 1970

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1977

Authors: Dennis E. Lough

Allen D. Burns

Approved by: Lyle V. Rich
Thesis Advisor

Chas R. Kodres
Second Reader

E. R. Gibson
Chairman, Department of Computer Science

D. F. Howard
Dean of Information and Policy Sciences

ABSTRACT

An abundance of Data Base Management Systems and Query Languages already exist, not to mention those which have been, and continue to be proposed. Most Data Base Management System surveys focus on the type of model used to represent the data, methods of access, protection, etc. This paper acquaints the EDP manager with the fundamental differences among the more significant query languages with emphasis on those characteristics which should be considered when choosing a query language. The term query language as used here has been expanded to include the entire user interface to the data base, and encompasses both data sublanguages and stand-alone query languages.

TABLE OF CONTENTS

LIST OF FIGURES

I. BACKGROUND

A. THE DATA BASE - WHAT IS IT?

A data base is not just a file system. While a data base could include a file system, it is much broader in scope. In general, an automated file system is a continuous group of fixed length records, sequentially ordered, which are accessed through card readers, tape units, and usually slow-speed rotating storage devices.

The data base came of age with the advent of fast, relatively inexpensive random access devices. A data file previously tied to and used with a specific application program was often unavailable to other users. This meant that for each new application a new program would be written necessitating a new data file relevant to that application. This led to much duplication of data, which, when combined with infrequent and inconsistent updating methods, produced a predictably large proliferation of redundant, often outdated, data files.

Martin [Ref. 1] provides the following definition of a data base in contrast to traditional file structures:

"A data base may be defined as a collection of interrelated data stored together with as little redundancy as possible to serve one or more applications in an optimal fashion; the data are stored so that they are independent of programs which must use the data; a common and controlled

approach is used in adding new data and in modifying and retrieving existing data within the data base."

Summarizing, a data base, as compared to a file system, reduces data redundancy, proliferation, and inconsistencies, permits shared access, and provides improved data integrity and comprehensive data protection.

B. BRIEF HISTORY

Prior to the age of the computer, data was stored and controlled in some form of clerical ledger. Thus manual extraction of information was severely restricted by labor costs and the output capacity per clerk. Additionally, such systems were subject to a high error rate and were typically redundant.

Of the early attempts at integrating information systems, the one most often mentioned is a project developed at the Mitre Corporation for the U.S. Air Force Electronics System Division. The outgrowth of the project was the Advanced Data Management System (ADAM), significant for its external data definition facility, which allowed different data base applications to use a common retrieval system.

Early data base systems employed exclusively low-level query languages. As Data Base Management System (DBMS) technology has developed, there has been a parallel development of query languages, not unlike the evolution of high-level programming languages during the development of modern computer systems.

Fry and Sibley [Ref. 2] cite three significant families of systems developed in the first decade of DBMS technology:

the formatted file/GIS family originated at the David Taylor Model Basin around 1958; the Bachman/IDS family, an Integrated Data Store facility developed at General Electric which was noted for its random access storage and high-level data manipulation language; and the Postley/MARK IV family for the IBM System/360.

The Data Base Task Group (DBTG), a CODASYL programming language committee formed to extend COBOL to operate in a database environment, made reports in 1969 [Ref. 3], 1971 [Ref. 4], and 1973 [Ref. 5]. These reports generally approached the data management question on the basis of using two separate languages: the Data Definition (Description) Language (DDL) and the Data Manipulation Language (DML).

The DBTG reports marked what is commonly regarded as the beginning of the second generation of Data Base Management Systems. Noteworthy examples of query languages in the CODASYL/DBTG family include DMS 1100 (UNIVAC 1110 series) and IDMS (IBM System/360). Paralleling the growth of DBTG systems, the relational model, first proposed by Codd [Ref. 6] in 1970, began to receive widespread attention and has been the subject of a great deal of academic research and debate.

C. RECENT DEVELOPMENTS

Considering the dominance of IBM in the data processing field, it is hardly surprising that most commercially available data base management software systems today run on IBM equipment. These include: Information Management System/360 (IMS/360), released by IBM in 1969; ADABAS, released by Software AG in 1970; IDMS (Cullinane

Corporation, 1973); System 2000 (MRI Systems, 1970); and TOTAL (Cincom Systems, 1971).

While the substance of the commercial market today remains in the realm of the CODASYL/IMS network/hierarchical system approach to DBMS, significant effort in recent years has been devoted to relational data bases. A notable effort in this area is the Interactive Graphics and Retrieval System (INGRES), a PDP-11/40 based hardware configuration installed and running on top of the UNIX Operating System at the University of California, Berkeley [Refs. 7, 8]. Another major effort is System R [Refs. 9, 10], a relational implementation developed at the IBM San Jose Research Lab. System R runs on an IBM/370 using VM/370 and provides a complete data base management capability.

D. MAN-MACHINE

Almost the entire thrust of recent DBMS proposals appears to have been in the direction of relational models underlying a non-procedural query language interface. Furthermore, each new proposal would seem to be designed for a more casual class of user than its predecessor. This is in an effort to make the machine perform ever more of the thought processes and improve the efficiency of interaction with the user. Obviously, as the machine assumes a greater role in the interaction, the user's efficiency increases and the costs both in processor time and software escalate. It would seem that there must come a point when it will be recognized that even the most casual user will always be required to possess at least a minimal, rudimentary knowledge of the environment in which his queries are to be formulated.

In the next chapter, the terminology associated with query languages and data bases is presented. Chapter III presents proposed criteria for measuring and selecting a query language. Chapter IV will present the characteristics of some of the better known, more widely used, or more interesting query languages. The final chapter provides some further guidance for selecting a query language, discusses some implications of the current trends in DBMS and examines some of the prospects for future systems.

II. TERMINOLOGY

A. DATA BASE MODELS

Much effort has been expended in comparing the three logical methods of organizing data by DBMS's. The advantages and disadvantages of each are well established. While the models are not the central issue here, a few words will be devoted to discussion of them in order to provide a framework from which to explore alternative methods of comparison. Martin [Ref. 1] and Date [Ref. 11] provide additional material on data base models.

1. Network

In the network approach, typified by systems of the CODASYL/DBTG family, record occurrences are represented as nodes of a network, chained together by named, directed arcs. The arcs present logical links between the entities which can be traversed in the specified direction in order to navigate through the data base.

2. Hierarchy

A restricted form of the network approach is the hierarchical model in which record occurrences are represented as nodes of a tree in a strictly owner-member (or more traditionally parent-child) relationship.

3. Relational

In the relational model data is viewed as a group of tables or flat files (relations). Each table is composed of rows (tuples). The order of the columns (attributes) within tables is of no significance and no hierarchical or graphic relationship exists among the tables containing the data.

B. DATA INDEPENDENCE

General users of a data base do not want to be and should not have to be concerned with data base implementation details such as access methods, character representation, or a host of other physical implementation and operating system particulars. All they need is a "view" of the data that will allow them to formulate queries and manipulate data. These users desire an "independence" from implementation details.

These details of access method, character representation, floating-point and integer representation, pointers, and record blocking are referred to as the physical structure of a data base. Freedom from the storage and access details gives the user "physical data independence". What the user is provided in the place of a physical view is a "logical view" of the data. Furthermore, it is often advantageous to provide different users with individually tailored logical views of the data. To meet this need and to give the system added flexibility, the following general approach is normally taken.

A system logical view of the data, termed a schema, is

defined. For hierarchical and network based systems the schema describes the relationship between record types and specifies the contents of record fields (data items). Similarly, it describes the structure of the relations in relational systems. Subschemas are then defined which give each user, or group of users their own logical view of the data. Thus, users are provided with "logical data independence".

A system that provides true physical data independence would allow the physical storage and access details to be changed without affecting the logical structure of the data (schemas and subschemas). True logical data independence exists only when logical changes can be made to the data base without significantly affecting the programs which access it [Ref. 2]. With one recent exception, Apple [Ref. 12], which is discussed in Chapter IV, logical data independence is currently more of a goal of a data base than a characteristic.

C. QUERY LANGUAGE DEFINITION

A data definition facility must exist to translate the schema and subschemas into a form usable by the data base system. A data manipulation facility is required to allow data in the system to be deleted, changed, and manipulated. The data definition facility, or Data Definition Language (DDL), describes the details and content of the schema and subschema tc the system. The data definition language may be a separate language available only to the Data Base Administrator (DBA), or it may be an extension of an application programming language or query language. There may, in fact, be two DDL's: one to define the schema and another to define subschemas. Alternatively, portions of

the data definition facility used for defining subschemas may exist as an extension to a query language; for example, the DEFINE VIEW statement of "embedded" SEQUEL allows the user to create a view against which he may issue queries or define other views [Ref. 9].

The data manipulation language in addition to having a query capability, characteristically provides the facility to update, create, and remove data base entities. Other operators typically include COUNT, SUM, MAX, MIN, AVG, relational operators, boolean operators, and inclusion operators. Like the data definition facility, the data manipulation facility may be an extension of a host application programming language; in such cases it is referred to as a data sublanguage (DSL) and is said to be "embedded" in the host language. The data manipulation facility may also exist as a stand-alone query language through which the user interacts directly with the DBMS; some authors [Ref. 10] limit the use of the term "query language" to languages of the stand alone variety. No distinction will be made here between a DSL and a query language except to point out that the latter is generally, though not necessarily, less procedural (this term will be defined later). The term query language will be used to refer to both, and is simply defined as the user interface to the data base.

In evaluating general purpose programming languages, consideration is normally given to the following: syntactic clarity, data structures, control structures, operators, efficiency of program execution, and, more recently, efficiency of program design, efficiency of problem solution, and compatability with top-down programming techniques [Ref. 13]. To use these same characteristics to examine query languages amounts to looking at query

languages only through the eyes of a programmer. Application programmers would probably be quite content with the "procedural" query languages selected using these criteria; however, everyone who wishes to interface directly with a data base is not a programmer.

III. USER-QUERY LANGUAGE COMPATIBILITY

Having recognized the importance of identifying the various user groups, this chapter defines five classes of user. The differences among these classes and the changing relative importance of them establish the need for measures of query languages. Two quantitative and three qualitative measures are proposed.

A. CLASSES OF USERS

Codd [Ref. 14] divides the users of data bases into five classes.

1. System Analysts/DBA's Staff

The system analysts are responsible for maintaining the data base management system, a function which includes creating or altering logical views of the data.

2. Application Programmmers

The application programmer serves as the middle-man for most of today's data processing needs; his function should be limited to designing and optimizing frequently executed routine queries or those queries that are inappropriate for more non-procedural query languages (more will be said about this later).

3. On-line Job-trained User

This group includes bank tellers and insurance company clerks who use the data base to answer routine queries on a random basis. The needs of this group are structured in nature, allowing most of their queries to be formalized. An example might be, "What is the balance of James M. Simpson's checking account?"

4. Researchers

This class of users is quite diverse but their queries could probably be characterized as being ad hoc and requiring aggregate results. Users in this group are most likely willing to wait a few hours or even days for an answer.

5. Casual User

Some authors [Ref. 15] seem willing to extend this term to include almost anyone; it is not unlikely that by the 1990's this may well be justified. However, at present a practical need is seen to limit this term to users such as managers, lawyers, analysts, accountants, and planners. These people need the information in the data base to help them make decisions but prefer not to encounter the expense or experience the delay in going through a third party, perhaps an application programmer, to process their queries.

Not only is there a varied group who use, or would like to interact with a data base, but the distribution of

interactions by the five classes is changing. In the early seventies, most data base interactions were by application programmers. In the next two decades the number of interactions by this group is expected to be less significant; a corresponding increase in the significance of interactions by on-line job-trained users seems imminent. But, by far, the casual user class seems to be the emerging dominant force.

Why is the role of the casual user on the rise? An increasingly large number of people have recognized the value of the data base and the lost opportunity costs that result from not being able to interface with it at a level in which the machine properly compliments man's decision making. In many instances availability of the proper query language could eliminate the need to go through the application programmer to answer a query, which seems particularly desirable in an era of increasing manpower costs. In many data base systems, such as military command and control systems, having the ability to answer queries as close as possible to the level at which critical decisions are made can be of particular importance. Some commercial vendors view developing the casual user market as a matter of survival [Ref. 16].

Note that the need to cater to those at the casual end of the user spectrum in no way implies that the languages that are used comfortably by the more computer oriented users do not and will not continue to play an important role in data base system interface.

B. THE NEED FOR MEASURES

What are the differences which exist among the classes

of user that affect the type of query language with which they would be comfortable? There are certainly differences in the understanding of how a computer works, internal data representations, operating systems, and programming experience. The latter is of particular importance since non-programmers may not be acquainted with such notions as data structures, looping, branching, and program efficiency. There are quite likely differing levels of mathematical sophistication. Differences in the number of times a user interacts with a data base and the interval between interactions are significant as an indication of the amount of time and effort a user can devote to learning a query language. The infrequent user may experience difficulty retaining syntactic details of some query languages.

In short, these differences point to the need for query languages compatible with users of varying qualifications and varying needs. This may imply having several query languages running concurrently on a data base. The more casual users do not desire to know and should not be required to learn the structure of the underlying data model, access methods, or programming control structures. Query languages giving users freedom from these details will require the machine to play a greater role in the man-machine symbiosis.

C. MEASURES OF QUERY LANGUAGES

With the need well established, qualitative and quantitative measures of query languages are presented.

1. Quantitative Measures

Level and completeness are considered quantitative measures. It is not essential that an evaluator apply these measures directly to a language being evaluated; it is, however, recommended that the concepts embodied in them be at least subjectively applied. References 17 and 18 discuss other measures of "software physics".

a. Level

Level [Ref. 17] is a quantitative measure of the amount of decision making that goes into the formation of a program to solve a problem. The number of decisions required to solve a given problem can be profoundly affected by the language being used. The user may be forced to make many decisions concerning syntax, delimiters, operators, etc., which are of little or no significance to the problem itself. Specifically, level is a mathematically derived value, between zero and one, based on the number of operators and operands used in the most efficient solution of a problem in a language.

As an example (though perhaps extreme and certainly a misuse of COBOL) of the difference in level, consider the following problem: program in COBOL and in APL, the matrix multiplication of matrices A and B. The result in APL is A+.xB. The amount of code generated to solve the same problem in COBOL is obviously much greater. In this example APL would have a level close to one while COBOL would have a level close to zero.

An evaluator may want to generate some benchmark

algorithm to actually determine the level of languages under consideration. Those languages that consistently have low levels will generally prove to be procedural in nature, have lower query design efficiency, hold the user responsible for exception or error checking, depend on the user for insuring efficiency of execution; and are thus less suited to the casual user. The opposite characteristics are generally true for languages yielding high values for level.

One word of caution is in order. Languages that yield a high value for level do not necessarily relieve the user of the responsibility for execution efficiency. APL is a case in point; the order of operation may have a profound effect on execution efficiency.

b. Completeness

Completeness refers to the selection capability of a query language, independent of any host language in which it may be embedded. A complete query language allows an authorized user to extract any data item that is semantically contained within a data base. Actually completeness is not a quantitative measure, but is included here because of its theoretical basis in mathematics. Codd [Ref. 19] established the basis for completeness of relational algebra and relational calculus. Thus the completeness for any language based on relational algebra or relational calculus may be established by determining if it permits the expression of any query expressible in the relational calculus. Recent work [Ref. 20] suggesting the equivalence of the three data models makes it appear reasonable to attempt to determine completeness for languages designed for hierarchical and network models. At the very least, an intuitive judgment of completeness should be obtained.

2. Qualitative Measures

Mathematical sophistication, learnability, and procedurality are proposed and discussed as qualitative measures of a query language.

a. Mathematical Sophistication

Mathematical sophistication is a subjective measure of the degree to which a language requires a user to be familiar with mathematical concepts, terminology, and symbology.

There is much to be said for languages that have a strong theoretical foundation in mathematics being used as target languages for user-oriented source languages. Relational algebra, discussed later, is often used for this purpose. The degree to which the relational algebra operators (projection, restriction, etc.), used to manipulate data, are visible in the source language is an indication of the amount of mathematical background required by the user. While terms such as restriction and projection are not well known, the actions accomplished by them are certainly more natural than that of an algorithm which uses a sequence of operations acting on one element at a time to select data. However, when these operations are "visible" at the user level, the user must mentally go through the mathematical operations necessary to extract the data. Is it reasonable for a casual user, such as a lawyer, to go through a mathematical process to formulate a query? Should he really have to do more than describe what he wants?

The presence of mathematical terms such as

"range" and symbols such as $\subset, \subseteq, \wedge, \vee, \sim$ may be other indicators of the amount of mathematical sophistication required to use a language.

Languages requiring little or no mathematical sophistication will tend to be non-procedural.

b. Learnability

In looking at the learnability of a language, one should be interested in the time and effort required to learn a workable subset to answer simple queries; this working set should meet the needs of many users. The restrictions and exceptions that must be mastered to compose moderately complex queries may be a better indication of learnability. The ability of casual users to retain what they have learned between infrequent uses must also be considered.

Human factors studies even more rigorous than those conducted on SQUARE and SEQUEL [Ref. 21] and Query by Example [Ref. 22] may be necessary to accurately identify elements that affect learnability. These studies support the notions that simple concepts should be used and that those concepts which differ semantically should also differ syntactically to avoid causing confusion. Reference 21 found that two different uses of the term WHERE were confusing; this ambiguity was eliminated in a later version of the language [Ref. 9]. Similarly Ref. 22 found that subjects made mistakes on one-fourth of the occasions when they needed to choose between the COUNT, SUM, COMPACT-COUNT, and AVERAGE operators. It was also noted that a significant percentage of users had trouble with the universal quantification constructs - for all, and there exists. A

smaller number even had difficulty with the relational operators <, >, ≤, ≥.

Languages that require less mathematical background and are non-procedural tend to be more easily learned by a broad class of users. Ideally a human factors study should be conducted on the specific group for which a language is intended.

The ability of a user to recall what he has learned can be enhanced if the user is given an explicit format with which to formulate his query. For example, in SEQUEL [Ref. 16] queries are structured as:

```
SELECT  <what>
    FROM    <relation name>
    WHERE   <condition>.
```

The user is less likely to recall the operators necessary to construct a query in languages containing more freedom of form. It is possible for an implementation, particularly an interactive one, to assist the user by providing prompts (such as query formats on a CRT screen) or by providing menus. Interactive input devices, such as lightpens, tablets, and cursors, that allow the user to specify his requirements in the most natural way may help appreciably.

c. Procedurality

Several of the measures described above have made reference to the procedurality or non-procedurality of a language. Many of these measures have a close correlation to the degree of procedurality of a language. Languages fall along a procedural -- non-procedural spectrum just as users span a broad spectrum of backgrounds and needs. This

spectrum cannot be described in absolute terms. Languages fall along this spectrum based on the following factors, several of which may apply in varying degrees.

(1) User Specified Access Path

Languages that require the user to provide a detailed access path to locate the desired data are highly procedural. In such languages the user must have a thorough knowledge of the underlying logical data organization. In hierarchical models the user must literally start at the root node and walk down the tree, node-by-node, until he reaches the required record type. In network models there may be several possible access paths (network models allow records to have more than one parent). Since there is no guarantee the user will pick the optimum access path, the efficiency of the query's execution may be adversely affected. Languages with this characteristic will have a low level and generally require that queries be composed by application programmers. The efficiency of problem solution is, therefore, low. Additionally, the DBA has less freedom in restructuring the underlying schema; doing so might require the rewriting of user programs that run on a recurring basis. In relational models different relations do not have parent-member relationships. Therefore, specifying the logical access path is not characteristic of relational query languages.

(2) User Conduct Element-by-element Search

Languages that require the user to program detailed handling of each record or tuple characteristically have go to, branching, and looping control structures. Once the user arrives at a node (in a hierarchical model) he must check each record element-by-element to determine if it meets the search criteria. The user is also responsible for

handling exceptions and error conditions. Naturally, languages requiring element-by-element search are quite procedural, have a low level, and are meant for users with programming experience.

(3) User Specified Logical Order of Operations

In languages exhibiting the preceeding two characteristics the user specifies the sequence in which operators are executed. However, this trait is identified separately because there are languages in which the user is not required to specify the access path or process records (tuples) individually but which do require him to logically apply operators in a specified sequence to locate data. Languages using terse relational algebra notation have this requirement. The term logical was used because the system may optimize the actual order of execution; if not, the user affects efficiency. Languages falling on this part of the spectrum are not limited to application programmers, but generally do require a significant degree of mathematical sophistication.

(4) User Affects Order of Execution

Language implementations exist in which the user does not actually specify the order of operations, but where he can significantly affect it. This feature will be discussed further in the examples presented in Chapter IV.

(5) User Knowledge of Data Base Content

At the extreme procedural end of the spectrum the user must have detailed knowledge of access paths. Toward the center of the spectrum relational languages still require knowledge of what attributes are in what relations; this knowledge must be used to "navigate

across relational boundaries" [Ref. 12]. Network and hierarchical languages in the center require knowledge of what data item is in specific files. At the non-procedural end the user is required only to know attribute or data item names. Some language implementations accept predefined synonyms or derivatives of actual data names. Such implementations are said to be "user-friendly."

Many attempts have been made to give adjectives to languages along the spectrum. Very procedural languages in which the user must have detailed knowledge of the data base in order to specify an element-by-element path through the data base are called "navigational" languages [Ref. 23]. Languages along the center of the spectrum, generally exhibiting factors three and four, are "prescriptive" languages. Languages at the non-procedural end, simply requiring the user to state attribute names, are "descriptive". Languages requiring no knowledge of the data base are termed "open-ended". The special class of query languages which allow the user to specify queries using tables, forms, and geometric images are also described using these adjectives.

In contrast to the procedural languages, the non-procedural languages require little knowledge of the data base or its underlying model; require less source code (higher level); allow the system to determine access paths; allow information to be addressed by content (as opposed to location); require less mathematical sophistication; free the user from error handling and execution efficiency considerations; are more efficient in total problem solution; and are easier to learn and retain. They are <u>generally</u>, though not necessarily, interactive. They also require another layer of software and thus <u>may</u> require more machine time. The extra layer of software gives the DBA

more freedom in changing the logical structure of the data base.

It is important to note that every non-procedural query language must be supported by an underlying procedural language. And it is reasonable to use the criteria presented in Chapter One for selecting general purpose programming languages as a starting point for evaluating a non-procedural query language.

IV. THE QUERY LANGUAGE SPECTRUM

A sampling of query languages is presented in order to examine the characteristics discussed in Chapter III. No attempt has been made to limit the sampling to commercially available languages since languages were selected on the basis of their ability to demonstrate various characteristics. The first two examples do not conform syntactically to any actual languages, but are used to represent a large group of navigational languages.

The sample query used throughout the examples is applied to the data base in Figures 1 and 2. The sample data base was extracted from Ref. 10. The sample query,

Q1: LIST THE ELECTION YEARS IN WHICH A REPUBLICAN FROM CALIFORNIA WAS ELECTED.

is solved in all examples.

A. PROCEDURAL - NETWORK

The following example uses Figure 2 and is representative of languages such as IMS [Ref. 24], CODASYL/DBTG [Ref. 4], and IDMS [Ref. 25]. This language models a network language, but a hierarchical procedural language would employ similar constructs. A typical query for Q1 might be formulated as follows:

ELECTIONS-WON

YEAR	WINNER-NAME	WINNER-VOTES
1952	Eisenhower	442
1956	Eisenhower	447
1960	Kennedy	303
1964	Johnson	486
1968	Nixon	301
1972	Nixon	520

PRESIDENTS

NAME	PARTY	HOME-STATE
Eisenhower	Republican	Texas
Kennedy	Democrat	Mass
Jchnson	Democrat	Texas
Nixon	Republican	Calif

Figure 1 - SAMPLE DATA BASE - RELATIONAL

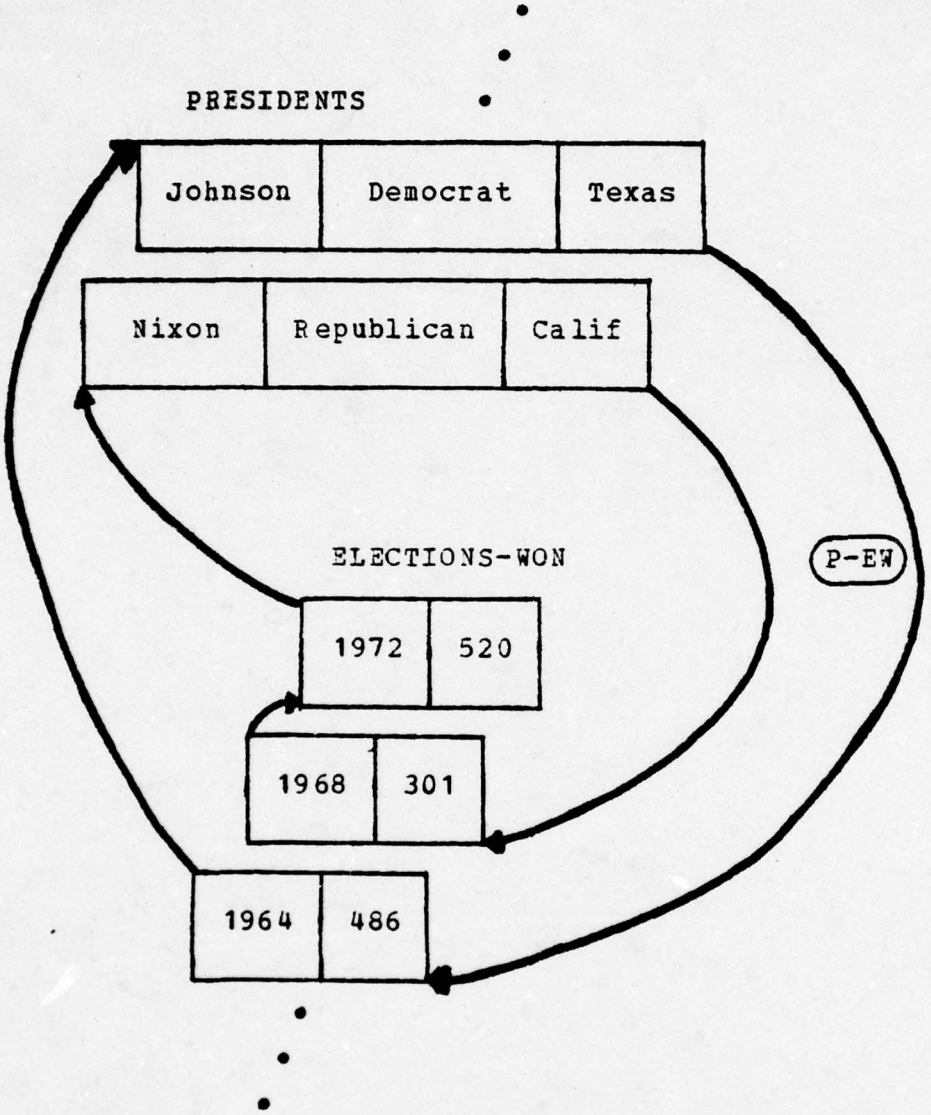


Figure 2 - SAMPLE DATA BASE - NETWORK

```
Q1:     MOVE "Republican" TO PARTY AND "Calif" TO
            HOME-STATE IN PRESIDENT
        FIND PRESIDENT RECORD
        if failure go to error
LOOP1:  FIND  MEMBER OF P-EW SET
        if none go to error
        PRINT YEAR
LOOP2:  FIND  NEXT MEMBER OF P-EW SET
        if none go to LOOP3
        PRINT YEAR
        go to LOOP2
LOOP3:  FIND  NEXT PRESIDENT
        if none "done"
        go to LOOP1
```

The query resembles a program in a simple general purpose programming language. Answering a query in this language amounts to writing a program that "navigates" through the data element-by-element. Indeed the language was designed to be used by programmers. It has low level and requires detailed knowledge of the data base.

B. PROCEDURAL - RELATIONAL

This example, using Figure 1, was included to demonstrate that low level relational languages such as XRM [Ref. 26] and GAMMA-0 [Ref. 27] have characteristics in common with network/hierarchical procedural languages. Those minor differences which do exist are not significant.

```
Q1:  FIND FIRST PRESIDENTS TUPLE
          WHERE PARTY = "Republican"
               AND HOME-STATE = "Calif"
          if failure; return "no such president"
LOOP1:    save  name
          FIND ELECTIONS-WON TUPLE WHERE
               WINNER-NAME = saved name
               if failure; return "president exists that did
                    not win election"
LOOP2:         PRINT YEAR
               FIND NEXT ELECTION-WON TUPLE WHERE
                    WINNER-NAME = saved name
                    if none:  go to LOOP3
               go to LOOP2
LOOP3:    FIND  NEXT PRESIDENTS TUPLE WHERE PARTY =
               "Republican" AND HOME-STATE = "Calif"
               if none "done"
          go to LOOP1
```

Although there is no access path to specify, the programmer must still use the knowledge that the President's name was common to two relations in order to navigate across relational boundaries and extract YEAR.

C. RELATIONAL ALGEBRA

These languages and the one used in the following example are based in set theory. They were proposed by Codd when he first introduced the relational model [Ref. 6] and later when he further defined a relational algebra and relational calculus [Refs. 19, 28]. Languages based on relational algebra include MACAIMS [Ref. 29], IS/1 [Ref. 30], and RDMS [Ref. 31]. The query Q1, using Figure

1, is presented in the relational algebra followed by a simple description of the operators used. A more complete description of relational operators is provided in Ref. 10.

```
#(0)(%[(1=3)](ELECTIONS-WON*(#(0)(%[(1="Republican"
    & 2="Calif")]PRESIDENTS))))
```

Processing begins at the innermost nested level. Domains are numbered left to right beginning with zero.

RESTRICTION(%) of a relation amounts to selecting all tuples (rows) that meet the conditions defined by the restriction of the relation specified. Thus

```
%[(1="Republican" & 2="Calif")] PRESIDENTS
```

selects all tuples from the relation PRESIDENTS in which the attribute (column) number "1" has the value "Republican" and attribute number "2" has the value "Calif". The result is a new relation, A, which serves as the operand for the next operator. A has the value:

NAME	PARTY	HOME-STATE
Nixon	Republican	Calif

PROJECTION(#) extracts specified attributes. The result is a new relation. Thus #(0)A selects the name attribute from

A yielding B:

NAME
Nixon

PRODUCT(*) concatenates its left and right arguments. If N and M are the cardinalities of the relation, the result is a new relation having N x M tuples. ELECTIONS-WON * B yields C:

YEAR	WINNER-NAME	WINNER-VOTES	NAME
1952	Eisenhower	442	Nixon
1956	Eisenhower	447	Nixon
1960	Kennedy	303	Nixon
1964	Johnson	486	Nixon
1968	Nixon	301	Nixon
1972	Nixon	520	Nixon

To this result a restriction is applied in which WINNER-NAME = NAME to eliminate ambiguous information such as the tuple

"1952, Eisenhower, 442, Nixon". The result is:

YEAR	WINNER-NAME	WINNER-VOTES	NAME
1968	Nixon	301	Nixon
1972	Nixon	520	Nixon

A projection yields the final answer:

YEAR
1968
1972

Actual implementations might use a more descriptive syntax in order to improve readability and learnability.

The significant difference between this and the two preceeding examples is that the user specified what he wanted in terms of sets not in terms of individual records. The underlying system took care of locating the relations, doing element-by-element processing, including error handling and storing intermediate results (perhaps virtually). The language is not navigational; it is essentially prescriptive. However, the user still had to specify the operations to be performed and the order in which they were to be performed. Some implementations may optimize the query by (1) finding an equivalent re-ordering

of the operators, (2) performing some operations virtually, (3) using a different combination of operators, (4) using a different set of relations and attributes, or (5) a combination of the above. Obviously, in many cases the user determines or affects the execution efficiency of his query.

The relational algebra does not require the programming skills of the first two examples, but the mathematically unsophisticated user might find it cumbersome, at least in its terse form.

D. RELATIONAL CALCULUS

In the relational calculus the user specifies what he wants using logical operators. The system maps the query into an equivalent expression in the relational algebra. Relational calculus may be thought of as restrictions followed by projections. Q1 stated in the relational calculus uses Figure 1:

$$\{x: \exists ((y,\text{Republican},\text{Calif}) \in \text{Presidents} \wedge (x,y,z) \in \text{ELECTIONS-WON})\}$$

The query specifies <u>what</u> is wanted, x; where it comes <u>from</u>, ELECTIONS-WON; and with what qualifications. Paraphrasing, Select x from (x,y,z), an element of ($\in$) ELECTIONS-WON, where z has any value and ($\wedge$) y is taken from (y,Republican,Calif), an element of President. X, y, and z are variable; Republican and Calif are constants.

The relational calculus is prescriptive. The user relies on his knowledge of what attributes are in what relations to prescribe a solution to his query. It is the system's responsibility to determine the exact sequence of

operations to be used. As with the relational algebra, it is still possible that using some decomposition algorithms the user may have a profound influence on execution efficiency. This is particularly true in implementations that use join as one of the operators in the underlying algebra (Pecherer [Ref. 32] has shown that certain subsets of the relational algebra are sufficient).

As discussed earlier many users have difficulty with the universal and existential qualifiers as well as set notation. This could limit the number of users who would feel comfortable with this language.

E. QUEL

QUEL [Ref. 33] is typical of languages which are based on the relational calculus. Other such languages include ALPHA [Ref. 28], COLARD [Ref. 34], and RIL [Ref. 35]. These languages do not require the user to apply quantifiers directly.

QUEL is actually a query language as well as a DSL which is embedded in the "C" language and the UNIX operating system using the procedural language EQUEL [Ref. 33].

The solution to Q1, using Figure 1, is presented:

```
RANGE OF E IS ELECTIONS-WON
RANGE OF P IS PRESIDENTS
RETRIEVE E.YEAR
     WHERE E.WINNER-NAME = P.NAME
          WHERE P.PARTY = "Republican" AND
                P.HOME-STATE  = "Calif"
```

The RANGE statements specify the relations used in the RETRIEVE and WHERE statements. The requirement for the user to have detailed knowledge of relational structure is reflected in the block structure of the language. Here again the user is prescribing a means to navigate across relational boundaries.

F. CUPID

References 36 and 37 describe CUPID (Casual User Pictorial Interface Design), which as its name implies is a picture-oriented query language. It is intended for the "casual" user. CUPID contains a high-level, menu-type sublanguage which is the front-end to INGRES, the relational data base system supporting QUEL. Additionally, CUPID offers a user definition facility which allows the system to "learn" new concepts. CUPID is presented here due to its graphic nature, which places it in the "special" category of query languages.

Figure 3 illustrates how Q1 might appear on a cathode ray tube device. An English language approximation of the query as it is depicted here, would be, "Select and save Name from Presidents relation where Party equals Republican and Home-state equals Calif; select and output Year from Elections-won relation where Winner-name equals saved Name." The graphic diagram is drawn as a result of user inputs from a keyboard.

In order for the individual graphic symbols to be displayed, the user must select from a menu of shapes available, those symbols which are necessary to formulate his query, and then through interactive queuing, properly position each symbol. It would seem that this would, for

all practical purposes, necessitate the user having sketched his intended query, or at least have it well-formed in his own mind prior to commencing "construction". There are unique shapes available in the menu to represent each of the following entities: relation names, domain names, relational operators, arithmetic operators, logical operators, constants, q-boxes(designates the target list of requested data), and a special symbol to enclose aggregate operations.

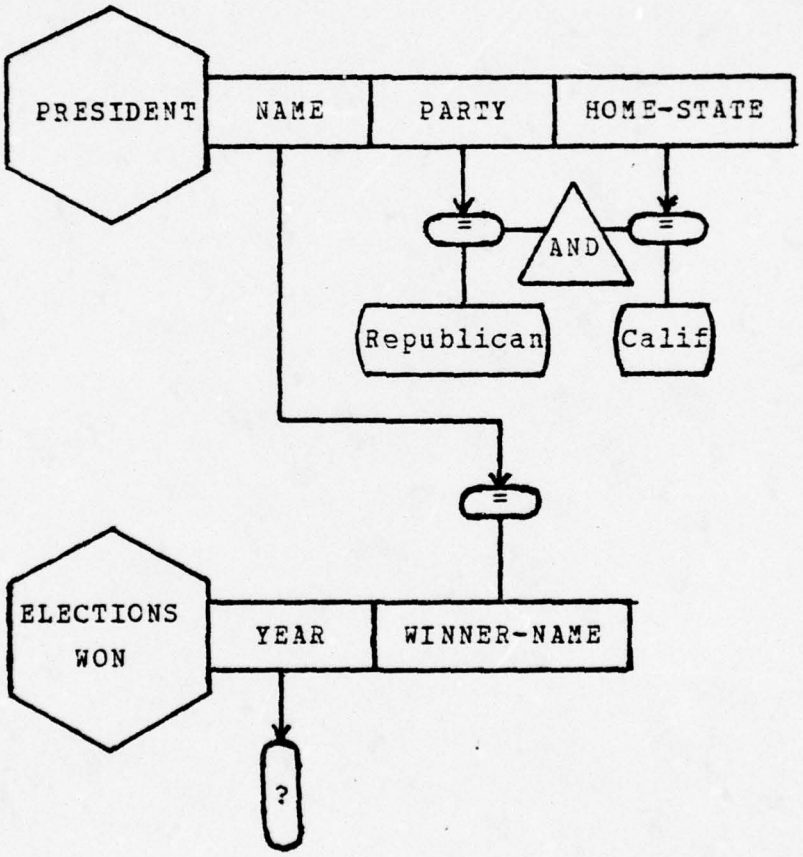


Figure 3 - CUPID QUERY

While this query appears fairly straightforward, some additional comments are in order. CUPID is prescriptive since the user must provide a graphical "prescription" for

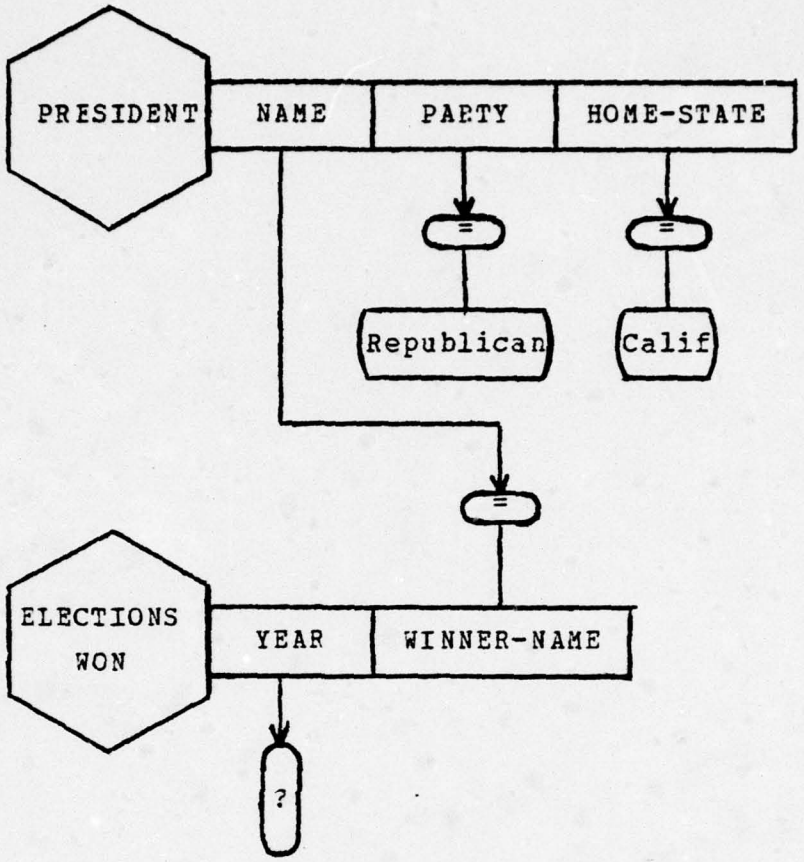


Figure 4 - AMBIGUOUS QUERY IN CUPID

answering a query. It would indeed be difficult to assess the learnability of this language without a much more thorough examination. Although, one small study reported favorable results [Ref. 37]. It can, however, be observed that the menu-selection feature would most surely stimulate the infrequent user's recall. The language requires a fairly high-level of mathematical sophistication; even the most simple queries generally cannot be formulated without boolean operators. Furthermore, it is not clear (at least to the casual user) whether the query depicted in Figure 4 would yield the same results as that in Figure 3; a substantial understanding of the formal syntax is also required. Considering these factors, it would seem that the

language would be more appropriate for the on-line job-trained user or perhaps the research category of user. As for the level and completeness of CUPID, it would be expected that both are very near QUEL, as the pictorial queries of CUPID are compiled into that prescriptive query language.

An additional, unique feature of CUPID is its definition capability. There are provisions for both user definitions and "learning" (through global definition tables). As an example, Figure 5 depicts how a user might formulate the query Q2: "List the names of minority presidents." The vocabulary definition algorithms would ideally resolve both problems presented by this query: (1) define "minority-president" and (2) resolve the apparent misplacement of "minority-president", an unknown value for the winner-name domain. The user would eventually be required, through a queuing sequence, to provide the maximum number of votes which would qualify a member of the winner-name domain as a minority president.

Q2:

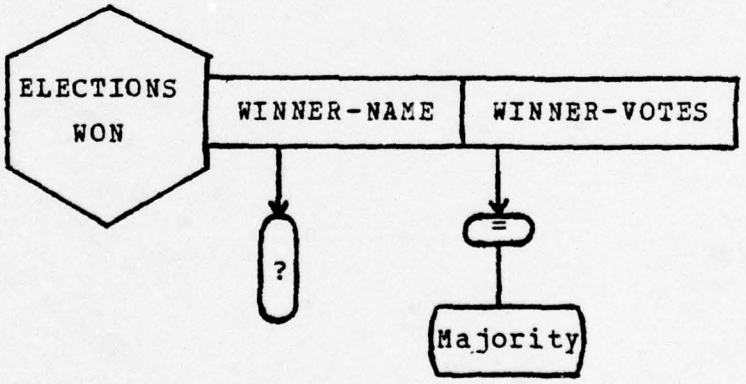


Figure 5 - USER-FRIENDLY QUERY IN CUPID

With some practical experience, it appears that CUPID would be an interesting, tidy method of expressing simple queries, and almost fun to use. On the other hand, more

complicated queries, such as the examples depicted in Ref. 36, become quite cumbersome and difficult to follow. Nonetheless, its designer should be commended for her unique efforts in the area of pictorial (graphical) queries, her success with user vocabulary definition, and the contribution to reduction of typographic and spelling errors in English-language text input by use of the "menu" facility.

G. SEQUEL

SEQUEL [Ref. 24], an outgrowth of SQUARE [Ref. 38], is typical of what Chamberlin [Ref. 10] calls "mapping-oriented languages". Mapping-oriented languages specify queries by defining a mapping between the desired result, which is a relation, and relations which are known to exist in the data base. SEQUEL was originally intended for interactive problem solving by non-computer specialists. System R [Ref. 31] implements SEQUEL both as a stand-alone language and as a DSL callable from application programming languages. The SEQUEL syntax resembles that of QUEL. Both are block structured and use WHERE statements. The SEQUEL solution of Q1 using Figure 1 is:

```
SELECT YEAR
    FROM   ELECTIONS-WON
    WHERE  WINNER-NAME =
        SELECT NAME
            FROM   PRESIDENTS
            WHERE  PARTY = "Republican"
                   AND HOME-STATE = "Calif"
```

Most of the comments about QUEL are applicable here. SEQUEL is prescriptive. The syntax of SEQUEL (SELECT, FROM

vice RANGE, RETRIEVE) seems more natural and more learnable than that of QUEL (see Ref. 21 for human factors study of SEQUEL). The mathematical sophistication required for simple queries is quite low. More complex queries may require use of the set operators - union, intersection, and set difference between intermediate mappings. For example, use the relations in Figure 1 to answer Q3:

Q3: FIND THE NAMES OF PRESIDENTS BORN IN TEXAS WHO RECEIVED MORE THAN 400 WINNER-VOTES.

An appropriate query might be:

```
SELECT WINNER-NAME
     FROM   ELECTIONS-WON
     WHERE  WINNER-VOTE > 400
∩
SELECT NAME
     FROM   PRESIDENTS
     WHERE  HOME-STATE = "Texas"
```

Some users may not have the prerequisite background for the proper use of the set operators.

H. QUERY BY EXAMPLE

Query by Example [Refs. 39, 40] is intended to serve the needs of the non-programming casual user with little mathematical background. It is presented here as an example of a category of query languages generally referred to as "forms". MARK IV [Ref. 41], intended for batch usage, was an early attempt at the forms approach. Another significant example of this approach has been presented by the CODASYL End User Facility Task Group [Ref. 42], which is attempting to define a "language" to emulate the naturalness of

manually extracting data from "forms" which are familiar to the user.

In Query by Example the user formulates his query by displaying blank tables on the CRT, naming the tables and their columns, and filling in the columns to illustrate the query to be answered. The queries are then translated into relational calculus for processing.

The Query by Example solution to Q1 follows:

ELECTIONS-WON	YEAR	WINNER-NAME	WINNER-VOTES
	P.1948	Wilson	

PRESIDENTS	NAME	PARTY	HOME-STATE
	Wilson	Republican	Calif

Republican and Calif are "constant elements" and are not underlined. WILSON and 1948 are variables, termed "example elements", and are designated as such by underlining. Example elements need not be actual elements in the data base. "P." specifies that the element is to be retrieved and printed.

This language is classified as "prescriptive" because the user must prescribe the means to navigate across two "tables" in a manner not unlike that required by SEQUEL. Reference 22 presents a human factors study of Query by Example. As discussed earlier, universal quantification presented a problem for some users. Q4, taken from [Ref. 39], is a query requiring universal quantification.

Q4: FIND THE NAMES OF SUPPLIERS WHO SUPPLY A JOB LOCATED IN NEW YORK WITH ALL PARTS OF TYPE A.

SUPPLY	SUPPLIER	PART-NAME	JOB-NAME
	P.Name	[ALL Rod] •	Bulb

PART	PART-NAME	TYPE
	ALL Rod	A

JOB	JOB-NAME	LOCATION
	Bulb	New York

ALL Rod means all PART-NAMEs of TYPE A. The dot '•' indicates that a SUPPLIER may supply more than parts of TYPE A to a job in New York.

In comparing this query with that for relational calculus, note how the target language shows through. The

problem with universal quantification was also detected in the evaluation of SEQUEL [Ref. 21]. This suggests that the level of mathematical sophistication required for a language may be easily under-estimated, and that the casual user's facility of the language may be limited.

I. APPLE

APPLE [Ref. 12] is the language used by a developing system which allows the user to specify queries using only attribute names. The formulation of Q1 follows:

```
SELECT YEAR WHERE PARTY = "Republican" AND
                  HOME-STATE = "Calif"
```

This language is truly "descriptive". The user need not "navigate" or give a "prescription" for navigating across relational boundaries; he need not even know what boundaries exist, let alone know what attributes transcend their boundaries. All that is required is a knowledge of attribute names. The system determines access paths and identifies the operators necessary to answer the query. APPLE currently has some implementation problems relating to the solution of ambiguities within queries, but, when satisfactorily resolved, could present the casual user with a language that is truly simple to understand and use.

J. NATURAL LANGUAGE

The idea of using natural English as a query language is not new or unique [Ref. 43]. In fact, Simmons [Ref. 44] reviewed fifteen experimental question-answering systems more than ten years ago.

Due to the inherent complexities of the English language, it is not surprising that while significant progress has been made, the field remains a virtual frontier. Martin [Ref. 45] has stated, "The art of devising dialogues between men and machines should be regarded as a new form of literacy. As yet the majority of practitioners of this art are unquestionably illiterate."

Since it has, in some sense, evolved as an efficient medium for communications interchange between men, it would seem that English must surely be the ultimate man-machine communications medium. The motivation is simply an extension of the previously discussed trend in query languages: making the machine do more so that man expends less effort in interacting with the machine. The solution is not quite as simply stated. Almost every attempt at employing natural English as a query language has actually utilized an extremely restricted subset of the language.

Instead of limiting the system to a subset of the English language, another approach which is relatively easy to implement is the computer-initiated dialogue. The user may experience the sensation of communicating with the machine in a very natural manner, when in reality the machine forces the use of a restricted subset of the language by initiating the dialogue and requiring the user to respond in an unambiguous manner which can be clearly understood by the machine.

To implement a genuine, user initiated dialogue system remains a challenge for the following reasons:

1. Typographic and Spelling Errors

Free-form input opens a Pandora's box of potential problems with respect to typographic and spelling errors.

Some query languages, such as CUPID have reduced the significance of this problem by the implementation of a menu.

2. Syntactic and Lexical Ambiguities

Many grammatically correct English language requests are inherently ambiguous. For instance, "List students receiving diplomas by sections," can easily be interpreted at least two different ways; the "by sections" phrase can be applied to modify (1) the verb "list" or (2) the phrase "receiving diplomas". Codd [Ref.15] has proposed a method of "RENDEZVOUS with the casual user" which does much to resolve this problem by having the machine introduce clarification dialogue limited to the machine-comprehensible subset of the natural language, and by periodically restating the user's query.

3. Large Vocabulary

Many languages (including CUPID, RENDEZVOUS, and IQF) now employ features which allow the user to define individual terms to suit his own needs.

There can be no doubt that natural English query languages will eventually come into widespread usage. In the interim, however, it is not unreasonable to expect the casual, but interested, user to familiarize himself with the environment of the DBMS and one of the currently available prescriptive query languages. Surely a clerk who has used tape-output adding machines for his entire career would be quite bewildered the first time he attempted to use a modern hand-held, reverse-polish notation calculator. But once acquainted with the machine, and its increased computational power, it would probably be difficult to persuade him to return to his former environment.

V. CONCLUSIONS

A. SUMMARY

The different classes of data base users were identified. Differences among these users were characterized which pointed to the need for measures with which to evaluate query languages. Quantitative and qualitative measures were proposed. Procedurality was shown to be a good overall measure of a language.

Summarizing, procedural languages (1) are intended for experienced programmers, (2) require more source code, (3) decrease programmer efficiency, (4) give less logical data independence, and (5) increase machine efficiency (provided queries are well designed). The more non-procedural languages (1) may be used by a broader spectrum of users, (2) require less source code, (3) increase programmer productivity, (4) provide increased logical data independence, and (5) decrease total problem solution time. Thus non-procedural languages increase human productivity at the expense of machine time due to the additional layers of software.

It is evident that the user must be a prime consideration in selection of a query language. The more casual the user the more non-procedural the language must be. "However, there will, no doubt, always be users whose interaction rates are so high, whose types of interactions are limited and whose data structures change slowly enough that they will rationally prefer a procedural system"

[Ref. 20]. The final choice of language(s) must be made within the context of other realities. One may be limited to existing hardware in which case increased machine time may not be available or the desired software interface may not exist.

Currently, the choice of commercial languages is limited to those which are designed for network and hierarchical models; it may be some time before a relational model is commercially available. The importance of the underlying model is likely to diminish in light of Stonebraker and Held's suggestion that the hierarchical and relational models are special cases of the network model [Ref. 20], and Date's proposed architecture for a single high-level language which supports all three data models [Ref. 46].

Reference 20 suggests another factor that should not be overlooked -- non-procedural languages are inappropriate for certain queries. For example, consider the query:

Q5: FIND the President receiving the second highest WINNER-VOTES.

It is unlikely that a non-procedural language processor would handle Q5 efficiently.

All things considered, most general data base implementations will require a mix of query languages to meet the needs of its various users, while a single query language may suffice for some special purpose data bases.

B. THE FUTURE

Two factors will impact on the DBMS of the future. First, the machine's role in the man-machine symbiosis will

expand beyond providing data which man analyzes and uses to make decisions. The machine itself will be programmed to access the data base and conduct increasingly higher levels of analysis to assist man in making complex decisions. A thorough discussion of "decision support systems" is found in Ref. 47 and an experimental system, GMIS, is the subject of Ref. 48.

Secondly, the arrival of mass storage devices on the market and the expected arrival of reasonably priced associative memory hardware will undoubtedly have a profound impact on the future of DBMS's and query languages. Mass storage devices make it practical to store large volumes of information on-line. Associative memory will surely serve as a catalyst for the development of the Data Base Machine (DBM).

The DBM will allow data base management functions, <u>at all levels</u>, to be separated from the traditional operating system. The DBM will not be dependent on the operating system's access methods, nor will it rely on the operating system for I/O control. It will be a specialized machine within a computer system which provides services to application processes. The DBM will be able to run asynchronously with the central processing unit and will make available data stored on secondary storage devices. Efficiency of query processing will improve because the DBM will have singular control over access, integrity, and protection [Ref. 49]. Freedom from the operating system and the use of associative memory (which will allow logical storage to be closer in form to physical storage) will further increase efficiency and decrease the machine "costs" of catering to the casual user.

C. FURTHER RESEARCH

Several areas were discussed having potential for future research which could lead to further improvement in the user interface. Human factors studies are needed to determine in general what language characteristics are compatible with each user class; where feasible, studies of specific user/language combinations should be conducted. Additional research in software engineering could result in a more complete set of quantitative measures of languages.

LIST OF REFERENCES

1. Martin, J. T., Computer Data Base Organization, Prentice-Hall, 1975.

2. Fry, J. P. and Sibley, E. H., "Evolution of Data-Base Management Systems," Computing Surveys, v. 8,n. 1, 7-42, March 1976.

3. Data Base Task Group of CODASYL Programming Language Commitee, Report, October 1969.

4. Data Base Task Group of CODASYL Programming Language Commitee, Report, April 1971.

5. CODASYL, CODASYL Data Description Language Journal of Development , U. S. Government Printing Office, June 1973.

6. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, v. 13,n. 6, p. 377-397, June 1970.

7. Held, G. and Stonebraker, M., "Storage Structures and Access Methods in the Relational Data Base Management System Ingres", Proceedings of the Pacific Regional Conference on Data: Its Use, Organization and Management, p. 26-33, April 1973.

8. Stonebraker, M. and Wong, E., "INGRES - A Relational Data Base System," University of California, Berkeley, MEM #ERL-M472, November 1974.

9. Astrahan, M. M. and others, "System R: A Relational Approach to Data-Base Management," IBM Research Report RJ1738, February 1976.

10. Chamberlin, D. D., "Relational Data-Base Management Systems," Computing Surveys, v. 8,n. 1, p. 43-66, March 1976.

11. Date, C. J., An Introduction to Database Systems, Addison-Wesley, 1976.

12. Carlson, C. R. and Kaplan, R. S., "A Generalized Access Path Model and Its Application to a Relational Data Base System," Proceedings of the ACM-SIGMOD International Conference , p. 143-155, June 1976.

13. Pratt, T. W., Programming Languages: Design and Implementation, Prentice-Hall, 1975.

14. Codd, E. F., "Recent Investigations in Relational Data Base Systems," Information Processing 74: Proceedings of the IFIP Congress 74, p. 1017-1021, 1974.

15. Codd, E. F., "Seven Steps to Rendezvous with the Causal User," Proceedings of the IFIP Working Conference on Data Base Management, p.169-179, 1974.

16. Chamberlin, D. D. and Boyce, R. F., "SEQUEL: A Structured English Query Language," Proceedings of the ACM-SIGMOD Workshop on Data Description, Access, and Control, p. 249-264, May 1974.

17. Halstead, M. H., "Software Physics Comparison of a Sample Program in DSL ALPHA and COBOL," IBM Research Report RJ1460, October 1974.

18. Halstead, M. H., "Software Physics: Basic Principles," IBM Research Report RJ1582, May 1975.

19. Codd, E. F., "Relational Completeness of Data Dase Sublanguages," Courant Computer Science Symposium 6: Data Base Systems, p. 65-98, May 1971.

20. Stonebraker, M. and Held, G., "Network, Hiearchies and Relations in Data Base Management Systems," Proceedings of the ACM Pacific Regional Conference, p. 1-9, April 1975.

21. Reisner, P., Boyce, R. F., and Chamberlin, D. D., "Human Factors Evaluation of Two Data Base Query Languages: SQUARE and SEQUEL," Proceedings of the AFIPS National Computer Conference, p. 447-452, May 1975.

22. Thomas, J. C. and Gould, J. D., "A Psychological Study of Query by Example," Proceedings of the AIPS National Computer Conference, p. 439-445, May 1975.

23. Bachman, C. W., "The Programmer as Navigator," Communications of the ACM, v. 16,n. 11, p. 653-658, November 1973.

24. IBM, IMS/360 Applications Description Manual, GH20-0765 IBM Corporation.

25. B. F. Goodrich, IDMS COBOL Data Manipulation Language.

26. Lorie, R. A., "XRM -- An Extended (n-ary) Relational Memory," IBM Scientific Center Report G320-2096, January 1974.

27. Bjorner, D. and others, "The GAMMA ZERO N-ary Relational Data Base Interface: Specifications of Objects And Operations," IBM Research Report RJ1200, April 1973.

28. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," Proceedings of the ACM-SIGFIDET Workshop on Data Description, Acess, and Control, p. 35-67, November 1971.

29. Goldstein, R. C. and Strnad,A. L., "The MACAIMS Data Management System," Proceedings of the ACM-SIGFIDET Workshop on Data Description, Acess, and Control, p.201-229, November 1970.

30. Todd, S. J. P., "Peterlee Relational Vehicle PRTV, a Technical Overview," IBM Scientific Centre Report UKSC0075 July 1975.

31. Whitney, V. K. M., "RDMS: A Relational Data Management System," Proceedings of the Fourth International Symposium on Computer and Information Sciences, December 1972.

32. Pecherer, R. M., "Efficient Retrieval in Relational Data Base Systems," University of California, Berkeley, MEM #ERL-M547, October 1975.

33. Allman, E. and Stonebraker, M. "Embedding a Relational Data Sublanguage in a General Purpose Programming Language," University of California, Berkeley, MEM #ERL-M504, March 1975.

34. Bracci, G., Fedeli, A., and Paolini, P., "A Language for a Relational Data Base," Sixth Annual Princeton Conference on Information Sciences and Systems, March 1972.

35. Fehder, P. L., "The Representation-Independent Language," IBM Research Reports RJ1121 and RJ1251, November 1972 and July 1973.

36. McDonald, N. and Stonebraker, M., "CUPID -- The Friendly Query Language," University of Califcrnia, Berkeley, MEM #ERL-M487, October 1974.

37. McDonald, N., "Cupid: A Graphics Oriented Facility for Support of Non-Programmer Interactions with a Data Base," University of California, Berkeley, MEM #ERL-M563, November 1975.

38. Boyce, R. F. and others, "Specifing Queries as Relational Expressions," Proceedings of the IFIP Working Conference on Data Base Management, p.169-177, 1974.

39. Zloof, M. M., "Query by Example," IBM Research Report RJ4917, July 1974.

40. Zloof, M. M., "Query By Example: the Invocation and Definition of Tables and Forms," Proceedings Of the International Conference on Very Large Data Bases, p. 1-24, September 1975.

41. MARK IV Systems Company, MARK IV File Management System Manual.

42. CODASYL, "A Progress Report on the Activities of the CODASYL End Users Facility Task Group," FDT Bulletin of ACM-SIGMOD, v. 8,n. 1, 1976.

43. Montgomery, C. A., "Is Natural Language An Unnatural Query Language?," Proceedings of the ACM National Conference, p. 1075-1078, 1972.

44. Simmons, R. F., "Natural Language Question-Answering Systems: 1969," Communications the ACM, v. 13,n. 1, p. 15-30, January 1970.

45. Martin, J. T., Principles of Data Base Management, Prentice-Hall, 1976.

46. Date, C. J., "An Architecture for High-Level Database Extensions," Proceedings of the ACM-SIGMOD International Conference, p. 101-121, June 1976.

47. Donovan, J. J., "Database System Approach to Management Decision Support," ACM Transactions on Database Systems, v. 1,n. 4, p. 344-369, December 1976.

48. Donovan, J. J. and Jacoby, H. D., "GMIS: An Experimental System for Data Management and Analysis," M.I.T. Working Paper #MIT-EL-75-0011WP, September 1975.

49. Everest G. C., "The Futures of Database Management," Proceddings of the ACM-SIGMOD Workshop on Data Description Access and Control, p. 445-462, May 1974.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexanderia, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	LT Lyle V. Rich, SC, USN, Code 52Rs Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	LCDR Dennis E. Lough, USN Patrol Squadron Seventeen FPO San Fransisco 96601	1
6.	LT Allen Dale Burns, USN Bureau of Naval Personnel (PERS 211) Washington, D.C. 20370	1