379
N81d.
No. 2439

IMPACTS OF QUERY SPECIFICATION MODE AND PROBLEM COMPLEXITY ON QUERY SPECIFICATION PRODUCTIVITY OF NOVICE USERS OF DATABASE SYSTEMS

DISSERTATION

Presented to the Graduate Council of the
North Texas State University in Partial
Fulfillment of the Requirements

For the Degree of

DOCTOR OF PHILOSOPHY

by

Wen-Jang Jih, B.S., M.C.

Denton, Texas

December, 1985

JDC

Jih, Wen-Jang, Impacts of Query Specification Mode and Problem Complexity on Query Specification Productivity of Novice Users of Database Systems. Doctor of Philosophy (Business Computer Informaton Systems). December, 1985, 185 pp., 13 tables, bibliography, 95 titles.

With the increased demand for the utilization of computerized information systems by business users, the need for investigating the impact of various user interfaces has been well recognized. It is usually assumed that providing the user with assistance in the usage of a system would significantly increase the user's productivity. There is, however, a dearth of systematic inquiry into this commonly held notion to verify its validity in a scientific fashion.

The purpose of this study is to investigate the impact of system-provided user assistance and complexity level of the problem on novice users' productivity in specifying database queries. The study is theoretical in the sense that it presents an approach adopted from research in deductive database systems to attack problems concerning user interface design. It is empirical in that it conducts an experiment in a controlled laboratory setting to collect primary data for the testing of a series of hypotheses.

The two independent variables are system-provided user assistance and problem complexity, while the dependent variable is the user's query specification productivity.

Three measures are used as separate indicators of query specification productivity: number of syntactic errors, number of semantic errors, and time required for completing a query task. Due to the lack of a well-defined metric for user assistance, the study first presents a generic classification scheme for relational query specification. Based on this classification scheme, two quantitative metrics for measuring the amount of user assistance in terms of prompts and defaults were developed. The user assistance is operationally defined with these two metrics.

Four findings emerge as significant results of the study. First, user assistance has a significant main effect on all of the three dependent measures at the 1 percent significance level. Second, problem complexity also has a significant impact on the three productivity measures at the 1 percent significance level. Third, the interaction effect of user assistance and problem complexity on the number of semantic errors and the amount of time for completion is significant at the 1 percent level. Fourth, Although this interaction effect on the number of syntactic errors is not significant at the 5 percent level, it is at the 10 percent level.

More research is needed to permit a thorough understanding of the issue of user interface design. A list of topics is suggested for future research to confirm or to modify the findings of this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

CHAPTER I

INTRODUCTION

Many database systems require users to possess a certain level of knowledge about computer language syntax. While this assumption may apply quite well to a member of the professional programming staff, it usually does not hold true for the large population of novice users. Characterized by the lack of extensive training in computing technologies, novice users are often in need of various forms of assistance to communicate effectively with a computer system. In the context of database systems, many actions have been taken to improve this user-computer interface. Examples include separation of different layers of abstractions (33), vary high level query language (11), and natural language query (21), etc. Due to the huge potential demand for easy-to-use database systems among novice users, research on the design and evaluation of easy-to-use interfaces has been one of the main activities of applied computer science.

This study addresses a particular group of novice users, functional area managers. It suggests that the pattern of a functional area manager's information need can be idenified through the interaction between the managers and the analysts. Then a collection of "interactive query patterns" can be developed based on these information requirement patterns. An interactive query pattern is a named module which, once invoked, will actively collect relevant data from the user and will execute when the data collection process is complete. By providing users with assistance in the use of a database system, these interactive query patterns may significantly improve users' productivity in specifying database queries.

To systematically investigate the effect of system's assistance in the use of database queries, a controlled experiment is conducted in the laboratory setting. The query specification mode without system assistance is used as the control group, and the mode with system assistance constitutes the experimental group. System assistance is implemented as prompts for user inputs and default (built-in) values for appropriate objects essential for a query program. Since it is likely that more complex query problems are affected more than simpler ones. problem complexity is investigated with regard to its mediating effects on the productivity measures.

The productivity meaures, in this study, include (1) the number of syntactic errors, (2) the number of semantic errors, and (3) the amount of time required to specify a database query. Other factors such as data model, data manipulation language, and on-line/off-line specification could play critical roles in the effectiveness of a database interface. These factors are controlled by using the relational data model, a logic-based query language, and on-line specification for both modes (user-assisted and non-user-assisted modes). In addition, the selection of subjects was done is such a way as to ensure that none of the subjects possess extensive experience in the use of computer.

The problem complexity in the context of database query specification is measured by Halstead's Volume (V) measure. To ensure the validity of using this measure, a set of query problems was derived from Date's text, Introduction to Database Systems (14). A query program was written for each of these problems. Volume meaures were then taken for the resulting programs. Significant differences were found to exist between the set of simple problems and that of the complex problems. For the experiment, two levels of problem complexity are employed based on the measure V.

Syntactic errors are defined to include spelling errors and grammatical errors found in the query programs written by the user. Semantic errors include several types of mistakes: wrong order of relations, incorrect names of exiting relation names, wrong attribute names, wrong operation names, and non-descriptive relation names. Any syntactic error will be detected by the system interpreter and will result in no output being returned. A semantic error will pass interpretation, but will return the incorrect output. Since the results of all syntactic errors are identical, the number of this type of error in a query is used as one of the productivity measures. For the same reason, the number of semantic errors is used as another productivity measure.

In some cases, a query is specified without any syntactic error or semantic error, but a large amount of time has been consumed for its completion. It appears that the amount of time used to complete a query specification activity constitutes a different dimension in measuring the user's productivity. This measure is calculated as the time elapsed between the point a problem is handed to the subject and the point that query is completed, regardless of its correctness.

Statement of the Problem

The general problem motivating this study is that database systems assume some computer knowledge on the user's part, hence tending to hamper wide use by the novice user. This study approaches this problem by investigating the effects of a system's assistance on the user's productivity in specifying queries. Because of the growing role of computer-aided decision making in business, successful formulation of database queries can have substantial significance in promoting the use of database systems for managerial decision making.

An array of database management systems have been developed to provide sophisticated data management capabilities in the years past. Although many database systems were initially developed to work specifically with transaction processing, many researchers have suggested that many of the capabilities of database management software can be beneficially utilized by functional area managers to support their decision-making activities (23, 24, 26). Normally, this software is general-purpose in nature (it is domain-independent); it does not possess much knowledge about users' problem space. In order to utilize the software's capabilities, the user must first learn the syntax of the system's query language. However, in the context of managerial decision making, users tend to be reluctant to devote much effort in mastering the use of the computer. Therefore, it remains a seemingly endless challenge for information system researchers to explore effective alternatives for resolving this obvious dilemma. This challenge has been observed by several researchers. For example, Eason and Damodaran maintained that,

> It is of little interest to him (the user) that the system is a technical masterpiece, or that it serves another user very well; if it serves his tasks poorly, it stands condemned as a poor system (17, p. 116).

In the introduction to their co-edited book, <u>Designing for Human-Computer Communication</u>, Sime and Coombs indicate that "the practical range of tomorrow's computer applications depends heavily upon the successful development of acceptable user interfaces (31, p. 1)." Because of the large population of novice users of the computer system, notions such as "natural communication," "user friendliness," "ease of use" have been widely discussed in both academic and practical literature. Here, the major concern is that a computer system should be designed to adapt to the novice user instead of forcing the latter to adapt to the system. As stated by Moran (27, p. 4):

> Novice and expert users generally exhibit quite different modes of behavior. The novice user usually finds himself engaged in problem-solving activity - almost everything for him is a problem - whereas the expert is skilled in interacting with the computer - it is for him a routine cognitive skill.

A similar observation is made by Coombs and Alty (12, p. 3), who noted that:

> Many people now use computers as part of their daily work, having little or no training in the technicalities of computing. This development is important for the continued growth of the industry but will necessitate a new philosophy towards the user. Whereas in the past systems have been designed with a central objective of obtaining maximum processing out of scarce and expensive hardware, the user being expected to adapt to the demands of machine-oriented software, future systems will have to take more account of users' current work practices and expertise. It is unlikely that the majority of users will wish to be extensively trained in computing, and commercial employers will certainly wish to minimize user training costs. Furthermore, inadequate user interfaces cost real money.

In general, despite the fact that much work has been done to investigate human factor aspects of the computer system, it is still difficult to make "prescriptive comments about the detailed design of the communication interface on a general basis as much depends upon the particular situation (32, p.24)." Because of the lack of a well-founded theory for the effective design of the human-computer interface, system designers have to rely on some heuristics in their endeavor. Although such design guidelines abound (2, 18), more research is still needed in the search of a normative theory which is capable of providing a systematic blueprint for the interface design. Drapper and Norman (16, p. 215) summed up the situation by saying, "The development of psychologically based, quantitative design tools is still in its infancy and much work remains to be done, both in development and in validation."

In summary, the value of a database system when used by novice users depends heavily upon the interface between the user's information requirements and system's functional capabilities. A useful system for the novice user of a database must be able to meet the user's information needs without requiring the user to expend much effort in the syntax learning activity. Several aspects of user-database interface have been explored such as data model selection, natural language vs. artificial language, and specification (nonprocedural) vs. prodedural language (20, 21, 35). Relatively little attention has been paid to the systematic investigation of possible effects of system-provided assistance on the user's query specification productivity. This study seeks to fill that gap.

Purpose of the Study

The primary objective of the study is to investigate the effect of system's assistance in forms of prompts and built-in defaults on novice users' productivity in specifying database queries. The secondary objective is to investigate the effect of problem complexity and the interaction effect of the two independent variables on the productivity measures.

A prototype relational database system is implemented for the domain of human resource management as a vehicle for data collection. The relational data model is employed as the conceptual view of the database designer as well as the external view of the end users. A logic-based programming language, Micro-Prolog, is used both as the data definition language and as the data manipulation language. The use of the same data model and the same language helps eliminate critical extraneous factors and thus allows for the isolation of the variables under investigation. The selection of the relational data model is consistent with the trend of database technology developments. It is also an attempt to build upon Ray's finding that relational model tends to provide a more natural user's model (28).

The choice of a logic programming language is motivated by research in deductive relational database systems (19). Among its most important features are nonprocedurality, modularity, and deductive capability. Nonprocedurality provides for a very high level of abstraction. Modularity allows for gradual growth of the system's model. And deductive capability efficiently facilitates implementation of interactive query patterns for the modelling of users' information requirements. Further explanation of the rationale for using the logic programming language is provided in the section on the experimental design.

Operationally, the objectives of this study can be achieved by answering the following two research questions:

(1) How does the novice user's productivity in specifying database queries differ between the use-assisted mode and non-use-assisted mode of query specification?

(2) What are the interaction effects of query specification mode and the complexity level of database query problems on novice users' query specification productivity?

Conceptual Framework of the Study

During the first decade since database technology started receiving interest from business data processing community, most database management systems (DBMSs) were aimed at the transaction and batch processing environment. The predominant concern was minimizing the computing resource required for per unit of work. The main function of database management was managing multiple files of different record types in an integrated fashion. Users are appropriately shielded from the underlying complexity of the database. As the utilization of the database technology evolved, it was recognized that the same integrated approach to data management can also be useful in the context of managerial decision making (29). With a user-friendly interface, a database system can assist functional area managers who are novices in the use of computer techonology by providing easy access to the database. As a matter of fact, a database system often constitutes a major component or subsystem in many proposed DSS design frameworks (6, 26).

This study investigates a specific approach to the design of the user-database interface for functional area managers. It stems from the observation that functional managers' information needs can be partially identified and classified into a collection of patterns. This task may be accomplished through the analysis of their functions, tasks and decisions. In summarizing the techniques for modelling users' information requirement, De and Sen (15, pp. 179, 180) contended:

> Decision analysis is characterized by its focus on decisions at the managerial level of the organization for each management-oriented applications. The approach requires identification of the critical decisions. One these critical decisions are identified, each of them is thoroughly discussed with the responsible manager and then carefully analyzed and modelled, usually by a decision flowchart. The decision flowchart indicates a set of discrete steps which the decision maker takes to make the decision.

Figure 2 depicts the conceptual framework of the study. Designing a user-system interface involves two parties - the user and the system. The user is characterized by a set of information requirements. In the case of functional area managers, the information required to support decision making is a function of the decision to be made. Some decisions are well-structured and thus can be automated. Other decisions require more human judgment before the proper action is taken. For this type of decision, a database system can be used to provide relevant information on the user's demand. An interface for this purpose should be designed with more emphasis on usability and effectiveness than on computing resource efficiency.

This study asserts that a portion of functional managers' information requirement for decision making can be analyzed and categorized into a collection of patterns. These requirement patterns constitute the user's problem space. The problem space can grow gradually along a learning curve. A collection of predefined query patterns may be designed to incorporate the prior knowledge about the user's problem space, and makes the query specification a question-answering process. As the user's experience in using the system increases, more decisions can be pinpointed to facilitate the design of a more comprehensive decision support system (DSS).

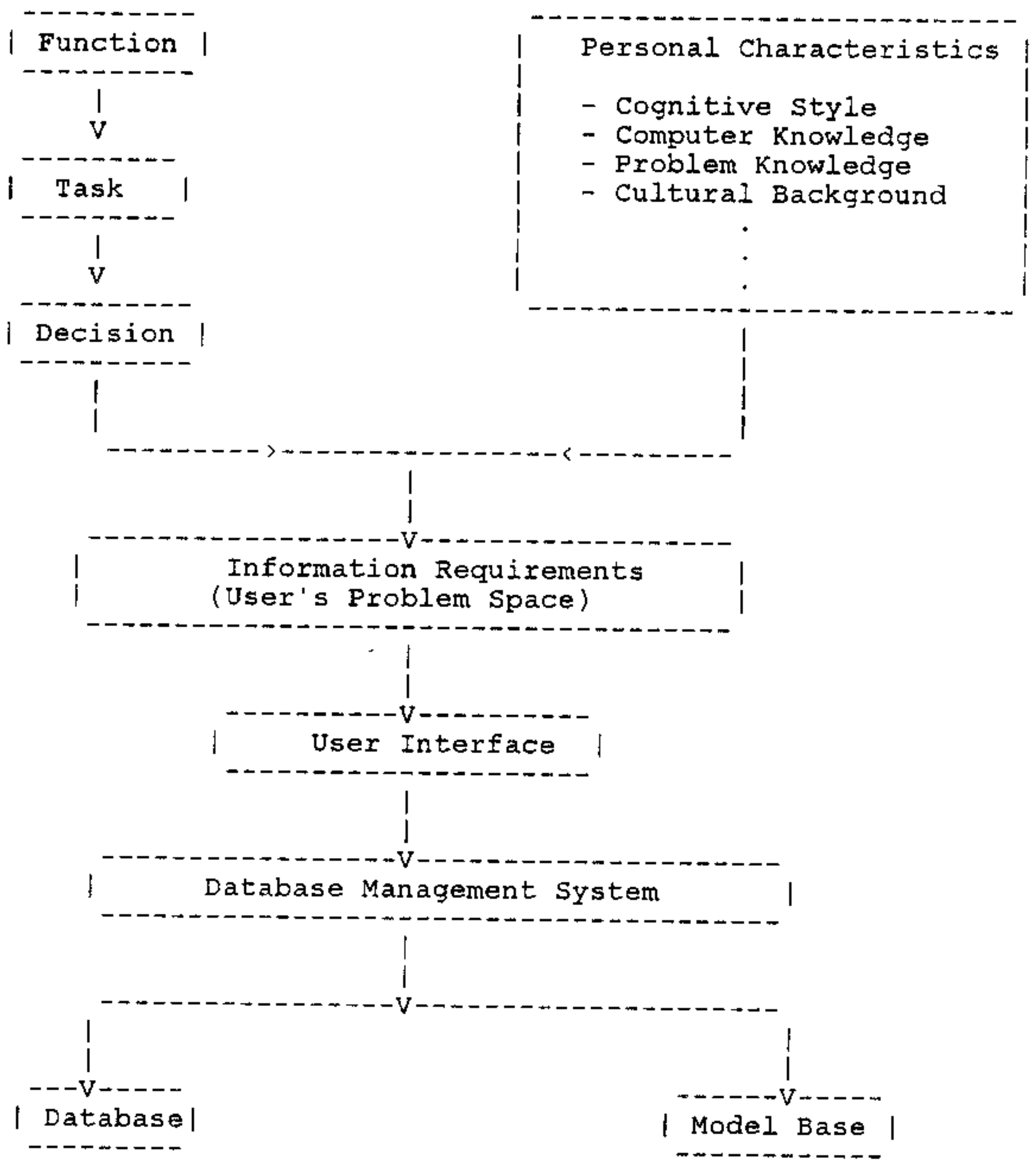


Figure 2--Conceptual Framework of the Study

In a relational database management system, a query typically consists of specifications of relation names, attribute values, and conditions in conformity with the query language's syntax. Relation name, attibute name-value pair , and condition specification constitute a relational query's basic objects. Where the system does not possess any knowledge about the user's information needs (or problem space), all these objects must be explicitly specified by the user. However, if the user's information needs can be at least partially identified in advance, a collection of interactive query modules can be developed to assist the user in specifying the required objects in the query. Prompts and defaults can, then, be employed to facilitate an easier interaction between the system and the user as perceived by the user. These interactive query modules collectively serve as the user interface when interacting with the underlying DBMS. Will this type of interface facilitate easier user-system interaction? A controlled experiment may be conducted to collect empirical evidence for answering this question.

Significance of the Study

This study investigates the effect of the assistance provided by the system in the specification of database queries. Intuitively, any such assistance should be beneficial to the user, hence worth the resource spend for its development. However, an objective justification requires a systematic inquiry. The empirical evidence collected in this study will serve to validate or to modify such a speculation.

Providing novice users, especially functional area managers, with an easy-to-use interface to facilitate more extensive use of database technologies has long been a critical issue in database research. Although ease of use may not be a predominant factor in the design of transaction processing database systems, a system which is easier to use will certainly receive a warmer welcome from novice users.

Many companies have installed database systems to establish integrated management of internal data. Functional area managers are likely to benefit from this technology through easier access to the huge pool of corporate data. Providing an easy-to-use interface is a critical function of this type of system. Knowledge and experience gained in this study will provide some useful guidance for adapting database systems to the user, rather than the other way around.

Another contribution of the study is to the growing area of deductive database (DDB) systems. A DDB uses the same representation formalism (first-order predicate logic) for the definition and manipulation of the database, including integrity contraints. This uniform representation formalism provides unique convenience useful for the study of many database problems (19). Currently, most effort has been directed to the theoretical investigation of such a logic-based approach to the design of large-scaled databases. Only a small number of experimental databases have been developed for the demonstration purpose (19). These experimental databases primarily address such problems as database definition, query optimization, and functional dependency. This study uses deductive database approach to investigate an issue concerning user interface, i.e., impact of user assistance on the user's query specification productivity. The experience obtained in this study will add to the existing body of knowledge about the application of logic in the study of database problems.

Definition of Terms

Being an interdisciplinary investigation, this study draws a number of terms from numerous fields in addition to some constructs identified by the study. This section is devoted to clarifying the meaning of these existing terms and the constructs found in the study. The following list merely serves as a summary to be referenced when necessary. The definitions of some terms will be described more thoroughly in the later chapters.

Query Pattern - A query module retrieving information to support the same category of decisions, e.g., employee promotion, training. Formally, a query pattern is characterized by a 4-tuple:

(Pattern label, Relation name, Attribute value, Condition).

When implemented with PROLOG, a query pattern is an IF-THEN inference rule. The goal of the rule serves as the label of a pattern, while the conjunctive conditions constitute a series of query statements for the retrieval of data. Entering a module name initiates the execution of these statements.

Mode of Query Specification - refers to the way queries are specified. In the interactive mode, queries are specified by the cooperation of the user and the system's assistance. System's assistance is primarily in the form of prompt and/or default. Two metrics (Interactive Volume Metric and System Default Metric) are devised to distinguish between two different modes.

Complexity of the Problem - The notion of problem complexity, in this study, is measured by Halstead's Volume metric. Complex problems require query programs with higher volume metric values, while the volume metric values of the query programs for simple problems are lower.

User Productivity in Specifying Database Queries - a construct to be measured by the number of syntactic errors, the number of semantic errors , the amount of time taken by the user to complete a query specification task.

Novice User - a type of users defined by Mayer (25, p. 123) as "users who had little or no previous experience with computers, who do not intend to become professional programmers, and who thus lack specific knowledge of computer programming."

Logic Programming Language - a programming language which basically regards any information processing as a theorem-proving process, i. e., a logic programming language is a computer realization of the first-order predicate logic.

Interactive Query Specification - a query specification mode in which database queries are specified through an interactive process between the user and the system interface. Prompts and/or defaults are used to reduce the computer knowledge requirement on the user's part. The unique feature is knowledges about the user's information needs implemented as a collection of interactive query patterns. This mode represents a "user's model plus system's model" mode of query specification.

Non-interactive Query Specification - a query specification mode in which users specify database queries without any system's assistance. The user has to have a sufficient knowledge about the query language syntax as well as the system's behavior. This mode repersents a "user's model only" mode of query specification.

CHAPTER BIBLIOGRAPHY

1. Alter, S. L., Decision Support Systems: Current Practices and Continuing Challenges , Reading, Massachusetts, Addison-Wesley Publishing Company, 1980, pp. 123 - 182.

2. Bailey, Robert W., Human Perfromance Engineering: A Guide for System Designers , New Jersey, Prentice-Hall, Inc., (1982).

3. Bateman, R. F. , "A TranBlator to Encourage User modifiable Man-Machine Dialog," in Sime, M. E., and Coombs, M. J., (Eds), Designing for Human-Computer Communication , New York, Academic Press, 1983, pp. 157 - 172.

4. Bedard, J., Gray, G. L., and Mock, T. J., "Decision Support Systems and Auditing," Center for Accounting Research, School of Accounting, University of Southern California, Working Paper No. 49, (April 1983).

5. Boies, S. J., "User Behavior on an Interactive Computer System," IBM Systems Journal , 13, No. 1, (1974), pp. 1 - 18.

6. Bonczek, R. H., Holsapple, C. W., and Whinston, A. B., Foundations of Decision Support Systems , New York, Academic Press, (1981)

7. ______________, "The Evolving Roles of Models in Decision Support Systems," Decision Sciences , Vol. 11, No. 2, (April 1980), pp. 337 - 356.

8. ______________, "A Generalized Decision Support System Using Predicate Calculus and Network Data Base Management," Operations Research , Vol. 29, No. 2, (March-April 1981), pp.263 - 281.

9. Brooks, Ruven E., "Studying Programmer Behavior Experimentally: The Problems of Proper Methodology," Communications of the ACM , Vol. 23, No. 4, (April 1980), pp.591 - 597.

10. Carlson, Eric D., Grace, Barbara F., and Sutton, Jimmy A., " Case Study of End User Requirements for Interactive Problem-Solving Systems," MIS Querterly , (March 1977), pp. 51 - 63.

11. Chamberlin, Donald D., "Relational Data-Base Management Systems," Computing Surveys , Vol. 8, No. 1, (March 1976), pp.43 - 66.

12. Coombs, M., J., and Alty, J. L., (Eds.), Computing Skills and the User Interface , New York, Academic Press, (1981).

13. Cuff, Rodney N., "On Casual Users," International Journal of Man-Machine Studies, 12, 1, (1980), pp. 163-187.

14. Date, C. J., Introduction to Database Systems, (2nd Edition), Menlo Park, California, Addison-Wesley Publishing, Co., 1981.

15. De, Prabudelha, and Sen, Arun, "A New Methodology for Database Requirements Analysis," MIS Quarterly , Vol. 8, No. 3, (September 1984), pp. 179 - 194.

16. Drapper, Stephen W., and Norman, Donald A., "Software Engineering for User Interface," Proceedings of 7th International Conference on Software Engineeing , (March 1984), pp. 214 - 220.

17. Eason, K. D., and Damodaran, L., "The Needs of the Commercial User," in Coombs, M. J., and Alty, J. L., (Eds), Computing Skills and the User Interface , New York, Academic Press, 1981, pp. 115- 142.

18. Gaines, B. R., and Shaw, M. L. G., "Dialog Engineering," in Sime, M. E. and Coombs, M. J. (Eds), Designing for Human-Computer Communication, New York, Academic Press, 1983, pp. 23 - 54.

19. Gallaire, H., Minker, J., and Nicolas, J.M., "Logic and Databases: A Deductive Approach," ACM Computing Surveys , Vol. 16, No. 2, (June 1984), pp.153 - 186.

20. Harris, L. R., "The Advantages of Natural Language Programming," in Sime, M. E., and Coombs, M. J., (Eds), Designing for Human-Computer Communication , New York, Academic Press, 1983, pp. 73 - 86.

21. Hill, I. D., "Natural Language Versus Computer Language," in Sime, M. E. , and Coombs, M. J. (Eds), Designing for Human-Computer Communication , New York, Academic Press, 1983, pp. 55 - 72.

22. James, E. B., "The User Interface: How We May Compute," in Coombs, M. J. , and Alty, J. L., (Eds), _Computing Skills and the User Interface_ , New York, Academic Press, 1981, pp. 333 - 336.

23. Loomis, Mary E. S., "The Nature of Database Management for Effective Decision Support Systems," in Holsapple, C. E. and Whinston, A. B., (Eds.), _Data Base Management: Theory and Applications_ , Armsterdam, North-Holland Publishng Co., 1983, pp. 155 - 174.

24. Martin, James, _Managing Data Base Environment_ , New Jersey, Prentice-Hall, 1983.

25. Mayer, Richard E., "The Psychology of How Novices Learn Computer Programming," _ACM Computing Surveys_ , Vol. 13, No. 1, (March 1981), pp. 121 - 141.

26. Methlie, Leif B., "Data Management for Decision Support Systems," _Data Base_ , Vol. 12, No. 1/2, (Fall 1980), pp. 40 - 46.

27. Moran, Thomas P., "An Applied Pshchology of the User," _ACM Computing Surveys_ , 13, 1, (March 1981), pp. 1-10.

28. Ray, Howard, _An Empirical Investigation of the Effects of Individual Differences and Data Models on the Ease-of-Use of Database Query Facilities by Casual Users_ , PhD Dissertation, North Texas State University (1984).

29. Reisner, Phillis, "Human Factors Studies of Database Query Languages: A Survey and Assessment," _ACM Computing Surveys_ , 13, 1, (March 1981), pp. 13-31.

30. Rich, E., "Natural-Language Interfaces," _Computer_ , Vol. 17, No. 9, (September 1984), pp. 39 - 50.

31. Sime, Max E., and Coombs, Michael J., " Introduction," in Sime, M. E., and Coombs, M. J., _Designing for Human-Computer Communication_ , New York, Academic Press, 1983, pp. 1 - 2.

32. Smith, H. T., "Human-Computer Communication," in Smith, H., and Green, T., (Eds.), _Human Interaction with Computers_ , New York, Academic Press, 1980, pp. 5 - 38.

33. Ullman, J. D., _Principles of Database Systems_, Rockville, Maryland, Computer Press, 1982.

34. Vassiliou, Y., Jarke, M., Stohr, E. A., Turner, J. A., and White, N. H., "Natural Language for Database Queries: A Laboratory Study," MIS Quarterly , (December 1983), pp.47 - 61.

35. Welty, C., and D. W. Stemple, "Human Factors Comparison of a Procedural and a Nonprocedural Query Language," ACM Transactions on Database Systems, Vol. 6, No. 4, (September 1981), pp. 626 - 649.

CHAPTER II

LITERATURE REVIEW

This study investigates impacts of system-provided user assistance on the user's productivity in specifying database queries. Essentially, a deductive database is created to facilitate the comparison of two database query specification modes in terms of various productivity measures. Relevant concepts and techniques upon which this study is built upon are works on database-user interface design and the role of logic in the study of database problems. Two specific topic of the former will be described in the first section: procedural versus nonprocedural language and user-system dialogue. The second section describes relevant theories and experimental works regarding the use of logic for the study of database problems: database definition, database manipulation, and definition of integrity constraint. The last section indicates the implications of these previous works for this study.

Database-User Interface

A database system is normally a complicated system requiring special considerations in the system architecture. A typical approach is to construct the system into multiple layers, with a mapping facility to facilitate the communication between each pair of consecutive layers. In an on-line environment, the user supplies inputs to and receives outputs from the database system. An interface of some kind is usually provided to enable the smooth communication between the user and the system. For novice users, the main function of this interface is to make the specification of database queries easier than otherwise.

The user interface has appeared in several different forms: nonprocedural query language, graphic specification format, and natural language query, etc. This study employes a nonprocedural language, PROLOG, as the implementation vehicle for the development of a particular form of user interface, question-answering. Therefore, some research results concerning relative merits of procedural versus nonprocedural languages and some guidelines for the design of system-user dialogue are reviewed here.

Procedural versus Nonprocedural Language

Welty and Stemple (12) investigated the notion of procedurality of query languages and its impact on the user's query writing performance. A quantitative metric, termed procedurality metric (PM), was constructed to be the arithmetic sum of two ratios: the ratio of the number of variable bindings to the number of permissible variable binding orderings, and the ratio of the number of operations to the number of permissible operation orderings. Here, a "permissible ordering" means "an order specified or allowed by the syntax and semantics of the language (12, p.

633)." When measured by the PM, the two query languages under investigation, SQL and TABLET, have PM values of 3.5 and 14.0, respectively. Assuming the two query languages have signigicantly different procedurality as indicated by the obvious arithmetic difference, the study went on comparing the user's query writing performance using the two languages. A controlled laboratory experiment was designed to collect data. Thirty query problems were divided into a "hard" group and an "easy" group. A test was administered immediately after the instruction, and a retention test was administered three weeks after the first test. Multivariate Analysis of Variance was used to analyze the resulting data. For the hard problems, the performance of experienced subjects is not significantly different in the first test. In the retention test, experienced subjects performed significantly better using TABLET (a procedural language) than using SQL (a nonprocedural language). For the easy problems, no significant difference in performance was found between TABLET and SQL. The overall conclusion of this study seemed to favor procedural languages for hard problems, and suggested that either a procedural or a nonprocedural language would be equally convenient for easy problems.

One weakness of the Welty and Stemple's study is that "easy" and "hard" problems seemed to be defined intuitively. No objective metric was used as a classification criterion. Consequently, replicating the study will be difficult if the same set of query problems and databases are not to be used.

User-System Dialogue

Dialogue type is one of the important characteristics of the user interface of an interactive system. Miller and Thomas (9) proposed a classification in which system-user dialogue can be viewed as differing in terms of two characteristics: 1. whether the dialogue is guided by the system or by the user, and 2. whether the user can respond freely or must choose from a set of predefined alternatives. Miller and Thomas (9, p. 524) maintained that these two dimensions are independent, and therefore, four basic types of dialogues can be identified. With the first type of dialogue, the dialogue process is guided by the system and the user is given a predefined set of alternatives to choose from. This design is most appropriate when the user is novice in the usage of the system and when the user's problem space is well-bounded. With the second type, te system guides the dialogue process and the user can respond freely. The design is necessary when the system is complicated and the user's problem space is not very well-bounded. In other words, the dialogue is essentially an ill-structured, information-gathering process. With the third type, the user guides the conversation, but the user's response is fixed. This design is appropriate when the user is at least somewhat knowledgeable about the usage of the system and when the user's problem can be well defined. The fourth type of dialogue has the user guiding the conversation and has free choices for the user. This provides maximum degree of freedom and is desirable for experienced users performing complicated tasks.

Although this classification and the suggested situations in which each of the basic dialogue types can be most benefically employed is logically sound, empirical evidence need to be collected for verification. A controlled experiment conducted in an on-line environment seems to be the most desired approach.

Logic for the Study of Database Problems

The idea of using first-order logic in clausal form as a formalizing vehicle has been applied in a variety of fields. Recent research accomplishments in specialized knowledge-based systems have prompted interest in applying logic more entensively in investigating database problems. The main convenience of this logic-based approach derives from the fact that logic "provides not only a conceptual framework for formalizing various database concepts, but also (in the form of logic programming) a tool for implementing them (4, pp. 102, 103)." This section briefly reviews the application of logic in three critical aspects of database problems: database definition, database manipulation, and integrity constraints.

Logic for Database Definition

Among the three main data models (relational, hierarchical, and network), logic has been linked mostly with the relational data model. A relational database defined with first-order logic (termed deductive relational database, or shortly as deductive database, DDB) retains all the useful features of the traditional relational database as developed by Codd (1). Since the only representation formalism for both database definition and manipulation is formal logic, it can make use of the inference power offered by formal logic.

The portion of formal logic which has been found most helpful as related to database research is first-order predicate calculus. Its primitive symbols set include: parentheses, variables, constants, predicate symbols, logical connectors, and quantifiers. A <u>term</u> is defined recursively to be a constant, a variable, an n-ary function with terms as arguments. If P is an n-ary predicate symbol, and t1, t2, ..., tn are terms, then P(t1, t2, ..., tn) is called an <u>atomic formula</u>. An atomic formula, the negation of an atomic formula, or a set of atomic formula and/or their negations makes a <u>well-formed formula</u>. Well-formed formulas constitute useful building blocks of deductive databases. A special form of well-formed formulas, called a Horn Clause, is employed as the implementation basis by the logic-based language, PROLOG.

An easy way to conceptualize a DDB is to regard it as a set of atomic formulas with all constants as arguments and a set of general laws governing the generation of virtual (or derived) data. The atomic formulas represent the facts about the universe of discourse. When general laws are invoked, new facts can be derived from existing facts. A citical design consideration of DDB, therefore, concerns the relative amount of atomic formulas which depict the world model explicitly and general laws modelling the real world in an implicit fashion.

Since the language used to define databases is executable, there is no need to separate tasks of requirement specification and database definition. The design cycle is shortened, thus facilitating faster feedback and closer interaction between the user and the designer.

Logic for Database Manipulation

According to Gallaire (7, p.249), Logic has many advantages over other query languages when related to relational databases. An obvious advantage is the nonprocedurality of the logic-based language. Constucting a query with Logic involves more of what information are wanted than of how the requested information are to be obtained. At this moment, there is still no consistent conclusion on the relative merits of nonprocedural versus procedural languages. It appears that for novice users, nonprocedural language tends to provide a more convenient vehicle for constructing queries.

Another unique feature of logic-based query language concerns the notion of "relational completeness" as defined by Codd (3). According to Codd, a query language is said to be relationally complete if it has the same expressive capability as relational calculus. However, using relational calculus as the yardstick of expressive power has its restrictions. Jarke and Koch (8, p. 117) contended that relational completeness should only be considered as a minimum requirement with respect to a query language's expressive power. At least two significant operations can not be expressed in a single relational calculus expression: transitive relationship and aggregation. An example of a query involving transitive relationship is, "Find the names of all employees reporting to manager Smith at any level." Many aggregations commonly encountered in daily operations can not be expressed in pure relational calculus. Although various aggregation functions have been added to most existing query languages, the problem with transitive relationship can be more effectively resolved by logic through its recursive capability.

Warren (10) studied the issue of improving query processing efficiency for interactive relational database queries expressed in logic. As he stated, the premise of his study is, "the first order logic formulation of a query lends itself to transformation which can improve the efficiency, corresponding to what is usually called query optimization (22, p. 272)." His query planning startegy essentially keeps track of the relative amount of pattern-matching associated with a query. Most restricted predicates, i.e., those involving the least amount of pattern-matching operations, are evaluated prior to the evaluation of less restricted predicates. This rearrangement of the sequence of predicates to be evaluated can generate query processing efficiency comparable with some well-enhanced existing systems such as INGRESS and System R (10, p. 279).

Logic and Integrity Constraints

Integrity constraints refer to the rules concerning valid states of databases. Two classes of integrity constraints are often discussed in database literatures: type constraints and dependency constraints (7, p. 251). Type constraints ensure that all data values fall into appropriate domains. Dependency constraints maintain various forms of data dependency among data items, such as functional dependency, multivalued dependency, etc. While type constraints ensure consistency of database instances, dependency constraints serve as critical guidelines for logical database design.

Logic does not distinguish the form of type constraints from that of general laws generating virtual data. Both type constraints and general laws are expressed in clausal forms with some predicates at both sides of the implicational connector. They also may be specified explicitly in query programs. This feature allows much flexibility in determining binding times for particular applications and users.

The significance of various data dependencies in the design of well-behaved logical data models have been widely documented. It has been demonstrated that the following guidelines implied by dependency constraints when designing a data model can most effectively avoid undesired anomalies associated with ongoing manipulations of the database (11). Gallaire (7, p.256) showed that these dependency constraints can be expressed in the form of first-order predicate calculus. Therefore, a theorem-prover can be used to generate a set of relations of desired normal forms. This feature may significantly improve the productivity of logical database design efforts.

Implications of Previous Research for this Study

This study was motivated by previous research in database-user interface and deductive databases in several ways. First, the relative merit of procedural and nonprocedural languages is still an issue of great controversy. While intuition suggests that a nonprocedural language might be easier to use than a procedural language, some empirical evidence has been found to support the opposite at least to some extent. Particularly related to this controversy is the effect of complexity level of the involved problems. In order to scientifically examine this effect, this study uses Halstead's Volume metric as a quantitative measurement for the degree of problem complexity. A validation procedure was followed to ensure the validity of using this metric in this context.

Second, As evidenced by frequent emphasis found in both academic and professional trade literature, providing novice users with an easy to use interface constitutes a critical success factor for most application systems. Various forms of assistance have been presented by research as well as by commercial products in an attempt to improve ease of use. However, there is still a dearth of systematic investigation of possible benefits due to the existence of system usage assistance from the perspective of user productivity. It is expected that empirical evidence collected in a scientific manner would provide better understanding of this critical component of system design.

Third, this study is supported by a useful tool used by deductive database researchers: logic programming language. Although many other languages are available for investigating effects of system assistance on users' productivity in specifying queries, a logic programming language is used for a number of reasons, including those cited in the above section. For summary, some of them are listed below:

1. The same language can be used for both data definition and data manipulation.
2. Fact assertions and inference rules are of the same basic forms: they are all Horn clauses. Modification of the database system components (database and knowledge-based interface) is relatively easy to perform.
3. As a predicate calculus language, a logic programming language satisfies Codd's notion of relational completeness by definition. In addition, some critical

features not supported by relational calculus, such as transitive relationship and aggregate operation, are available in the logic programming language. More specifically, transitive relationships can be implemented by recursion, and aggregate operations can be implemented by user-defined functions in forms of inference rules.

4. The high modularity of a system developed in a logic programming language facilitates prototyping strategy for system development.

The fourth motivation from previous researches stems from the observation that most research focused on development of individual systems. The literature survey has not found any research addressing possible effects of problem complexity on human performance measures. It is reasonable for one to speculate that problems of different complexity levels may benefit to different extents from system-provided assistance.

CHAPTER BIBLIOGRAPHY

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM , Vol. 13, no. 6, (June 1970), pp. 377 - 387.

2. Codd, E. F., "Relational Completeness of Data Base Sublanguage," In Data Base Systems , edited by R. Rustin, New York, Prentice-Hall, 1972, pp. 65 - 98.

3. Codd, E. F., "Relational Database: A Practical Foundation for Productivity," Communications of the ACM , Vol. 25, No. 2, (February 1982), pp. 109 - 117.

4. Dahl, V., "On Database Development through Logic," ACM Trasactions on Database Systems , Vol. 7, No. 1, (March 1982), pp. 102 -123.

5. Ford, Nelson, "Decision Support Systems and Expert Systems: A Comparison," Working Paper, Department of Management, Auburn University, (1985).

6. Gallaire, H., J. Minker, and J.-M. Nicholas, "Logic and Databases: A Deductive Approach," ACM Computing Surveys , Vol. 16, No. 2, (June 1984), pp. 153 - 186.

7. Gallaire, H., "Impacts of Logic on Data Bases," Proceedings of International Conference on Very Large Databases , 1981, pp. 248 - 259.

8. Jarke, M., and J. Koch, "Query Optimization in Database Systems," ACM Computing Surveys , Vol. 16, No. 2, (June 1984), pp. 111 - 152.

9. Miller, L. A., and J. C. Thomas, JR., "Behavioral Issues in the Use of Interactive Systems," International Journal of Man-Machine Studies, (1977), 9, pp. 509 - 536.

10. Warren, D. H. D., "Efficient Processing of Interactive Relational Database Queries Expressed in Logic," Proceedings of International Conference on Very Large Database , 1981, pp. 272 - 281.

11. Ullman, J. D., Principles of Database Systems , Rockville, Maryland, Computer Science Press, 1982.

12. Welty, C., and D. W. Stemple, "Human Factors Comparison of a Procedural and a Nonprocedural Query Language," ACM Transactions on Database Systems, Vol. 6, No. 4, (September 1981), pp. 626 - 649.

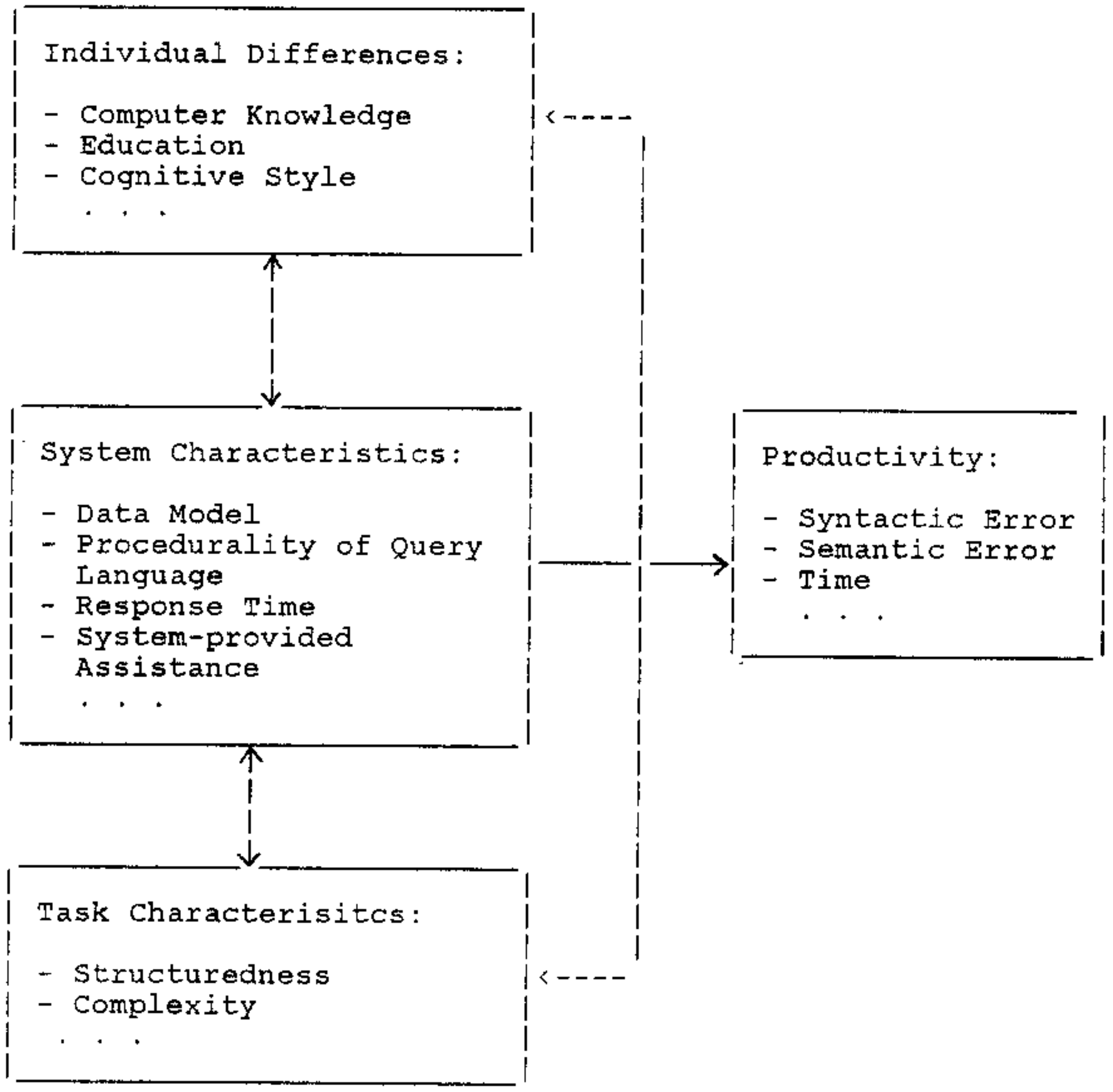
CHAPTER III

RESEARCH METHODOLOGY

This research is both theoretical and empirical. It's theoretical in the sense that it suggests and experiments with an idea of using deductive database in investigating problems concerning the user-system interface. It's empirical since it conducts a controlled experiment to collect empirical data and tested a set of hypotheses regarding possible impacts of the proposed idea. This section primarily describes methodogical issues of this experiment. Specifically, issues described in this section include: theoretical constructs to be investigated, research variables and the way these variables are measured, hypotheses to be tested, general experiment design, design of the experimental deductive database system, and the data collection and analysis techniques. Due to the exploratory nature of the experimental system, a substantial portion of this section is devoted to a detailed description of this system. Critical topics related to the experimental database system design are: architecture of the system, conceptual model of the database, implementation design of the database, and logical access maps of the interactive query patterns.

Major Factors Affecting Query Specification Productivity

The user's productivity in specifying queries is often affected by several factors. As indicated by Ray (5), these dimensions include: environmental context, system features, and individual differences. In an investigation of procedurality of query languages, Welty (11) found that, for easy problems, a procedural language results in better performance in query writing. This finding is in contrast to a popularly held conjecture that a nonprocedural language should always be easier to use than a procedural language. One may also suspect that the complexity level of problems may have some bearing on the productivity of query specification, given the same environment, system features, and psychological characteristics. Figure 2 diagrammatically summarizes these critical factors. Each of the three classes of factors has a main effect on the productivity measures. For example, one with sufficient knowledge of a system tends to perform better than the other without much system knowledge; a relational data model may provide a more natural view of the database, thus allowing for higher productivity. In addition to the main effects, these factors may interact and affect the productivity measures in more subtle ways. Welty's finding that a nonprocedural language seems to be more appropriate for simple problems, whereas a procedural language may be better for complex ones is an obvious example.



Note: Each class of factors has effects on the productivity measures. And different classes of factors may interact and generate some subtle impacts. It is worth noting that, while the definitions of syntactic errors and semantic errors depend upon the implementation language, they are also determined by the characteristics of the associated task domain.

Figure 2--Major Factors Affecting Query Specification Productivity.

Research Variables and Measurements

This study investigates the effect of system-provided assistance on novice users' query specification productivity. It also explores the role of problem complexity in this respect. Therefore, the independent variables involved in the experiment are system assistance and problem complexity. Three productivity measures are used as separate measures of the dependent variable, productivity in specifying queries. Table I provides a summary list of research variables followed by detailed explanations for each variable.

TABLE I

RESEARCH VARIABLES LIST

INDEPENDENT VARIABLES	DEPENDENT VARIALES
System-provided User Assistance	Query Specification Productivity
Problem Complexity	- Syntactic Errors - Semantic Errors - Time of Completion

Note: The three dependent measures are treated separately in this study.

Independent Variables

Operational Definition of System-Provided User-Assistance.-- The major independent variable investigated by this study is system-provided assistance when interfacing with a database system. Although many reports regarding this feature can be found both in the academic literature and in commercial products, a generic metric for the measurement of the "amount" of assistance has not been proposed. In order to conduct the investigation in a systematic fashion, such a measurement scheme is needed to facilitate experimental manipulation.

System-provided user assistance has many different forms, such as menu-selection, command macro, help facility, and graphic prompts. Therefore, a generic measurement scheme for the relative amount of use assistance must be independent of the forms of assistance. In the context of database query, there are some unique characteristics helpful for this purpose. Typically, the way the user views a database is primarily through data models. An interface for end users must inevitably be closely linked to the data model supported by the database management system. It can be expected that the basic conceptual constructs of the data model would determine, to a great extent, the format of such a measurement scheme.

In the relational database, data are stored in a set of two-dimensional tables, called relations. Each relation has a number of columns, called attributes, and a dynamic, time-varying number of rows, called tuples. The set of possible values for a given attribute is called its domain. In the literature, a formal definition is usually based on the notion of domians. Given a collection of domains D1, D2, ..., Dn, a relation of degree n is defined as any subset of the cartesian product of these domains. Therefore, a relation can be viewed as a set of ordered tuples <d1 d2, ..., dn> such that d1 belongs to D1, d2 belongs to D2, ..., and dn belongs to Dn. Here, n denotes the number of attributes constituting a relation. A relational database is manipulated by query programs written in specially defined query languages. Basically, the definition of a relational query language is supported either by relational algebra or by relational calculus. A query program specifies which attributes are to be extracted from what relations subject to certain conditions on the values of those desired attributes. The relational algebra takes one or two relations as operands and produces one relation as a result. While the relational algebra manipulates the relational database at the row (or tuple) level, the relational calculus manipulates the relational database at the relation level. Instead of having to create a number of intermediary relations as in the case of relational algebra, the relational calculus specifies only the list of attributes, the relations from which those attribute are drawn, and the predicates which have to be satisfied. From the logical point of view, the same fundamental objects are specified in an algebra-based as well as a calculus-based query program. These fundamental objects are: 1). relation name, 2). attribute name, and 3). condition regarding attribute values. The requirement for the specification of these objects remains the same regardless of syntactical differences between lanuages.

A generic measurement scheme for use assistance provided by the system may be developed based on whether or not a particular object is assisted. There are at least two levels involved in this concern. In a measurement scheme at the first level, assistance for the specification of an object is either existent or not existent. A measurement scheme of the second order takes into account different forms of assistance for the specification of each object. An even higher-order measurement scheme can be contructed incorporating amount of assistance associated with each form of assistance for the corresponding object. Generally, the degree of sophistication of a measurement model depends on the purpose that the model is developed for. For this study, a first-order measurement is sufficient for the experiment which was designed to test the hypotheses of this study. However, the significance of a more sophisticated, more comprehensive, second-order metric is not excluded in any sense.

Table II illustrates how a user interface may be classified based on the existence or non-existence of the assistance for specifying an object. In addition to the three fundamental objects (relation name, attibute name, and condition operator) described in the previous paragraph, this classification scheme includes a higher-level object, the label of the query pattern.

TABLE II

A CLASSIFICATION SCHEME FOR RELATIONAL QUERY SPECIFICATION

		OBJECTS			
		Query Pattern Label	Relation Name	Attribute Name	Conditional Operator
	1	+	+	+	+
	2	+	+	+	-
	3	+	+	-	+
C	4	+	-	+	+
A	5	-	+	+	+
T	6	+	+	-	-
E	7	+	-	-	+
G	8	+	-	+	-
O	9	-	-	+	+
R	10	-	+	-	+
Y	11	-	+	+	-
	12	+	-	-	-
	13	-	+	-	-
	14	-	-	+	-
	15	-	-	-	+
	16	-	-	-	-

Note: A "+" indicates presence of system assistance as to the specification of a particular object, while a "-" indicates absence of such assistance.

Several points about this classification scheme must be noted before proceeding to describe the metrics based on it. First, this scheme is not intended to classify existing commercial products. As a matter of fact, it only considers conceptual contructs of the relational data model. No technical details regarding system implementation have been considered. In other words, it is primarily a conceptual tool without consideration of technological feasibility. This lack of correspondence with existing technology is desirable if a model is to extend its life beyond the scope of current state-of-the-art.

Second, it is clear that more assistance is available when more objects are included. Consider the two extremes, for instance, in this classification scheme. While Category Sixteen provides no assistance for any of the objects, Category One furnishes assistance for all the objects. Although the relative "amount" of assistance is difficult to distinguish for the categories between these two extremes, it can be comfortably asserted that Category One represents more assistance than Category Sixteen.

Third, the label of a query pattern represents a meaningful name of a predefined query program which is similar to a procedure or subroutine name of, for example, COBOL or PL1 programs. In a command-driven interface, entering such a label will invoke the execution of the associated query program. Since labels or query pattern names are easier to remember and to specify, they represent abstractions at a very high level. With the aid of a template or a well-designed menu prompt, the user does not need to have much knowledge of the underlying system and still can successfully interface with it.

The major independent variable, user assistance, can be further operationalized by developing two related, quantitative metrics. Essentially, this study employs an experimental database system which provides use assistance in forms of prompts and defaults. Thus, the amount of user assistance may be expressed as a function of prompts and defaults. This indicator for the amount of user assistance is termed Interactive Metric (IM) in this study. Since prompt and default are basically two different forms of system assistance, a metric is defined for each. A metric, termed Interactive Volume Metric (I¢C), is suggested for measuring relative amount of system-provided users assistance in the form of prompts. Another metric, termed System Default Metric (SDM), may be used to measure this assistance in the form of defaults. The definition of IVM and SDM together with the meanings of involved parameters are presented as follows.

$$\text{Interactive Volume Metric (IVM)} = \left(\sum_{i=1}^{k} (ni/\ Ni) \right) / k$$

Where

n1: Average no. of prompts for query pattern labels;

n2: Average no. of prompts for relation names;

n3: Average no. of prompts for attribute values;

n4: Average no. of prompts for conditional operators;

N1: Average no of query patterns to be specified;

N2: Average no. of relations to be referenced;

N3: Average no. of attributes to be bound;

N4: Average no. of conditions to be specified;

$ni <= Ni$

k: 1 - 4

Boundary Conditions of the metric:

1. $n1 = n2 = n3 = n4 = 0 \;\; \text{--->} \;\; IVM = 0$

   (No prompts are provided for any one of the objects.)

2. $n1 = N1$, $n2 = N2$, $n3 = N3$, $n4 = N4$ $\;\; \text{--->} \;\; IVM = 1$

   (A comprehensive menu is provided for all query pattern labels in addition to the prompts for all of the other objects.)

$$\text{System Default Metric (SDM)} = \left(\sum_{i=1}^{k} (ni/\ Ni) \right) / k$$

Where

n1: Average no. of defaults for query pattern labels;

n2: Average no. of defaults for relation names;

n3: Average no. of defaults for attribute values;

n4: Average no. of defaults for conditional operators;

N1: Average no. of query patterns to be specified;

N2: Average no. of relations to be referenced;

N3: Average no. of attributes to be bound;

N4: Average no. of conditions to be specified;

ni <= Ni

k: 1 - 4

Boundary Conditions of the metric:

1. n1 = n2 = n3 = n4 = 0 ---> SDM = 0

   (No defaults are provided for any one of the objects.)

2. n1 = N1
   n2 = N2
   n3 = N3 ---> SDM = 1
   n4 = N4

   (A natural language description of the problem is parsed, and the query program formulation process is entirely transparent to the user.)

In this study, the <u>user assistance</u> is therefore operationally defined as

> The user assistance is a variable concerning system-provided use assistance in forms of prompts and/or defaults, the amounts of which are measured by IVM and SDM, respectively.

<u>Operational Definition of Problem Complexity</u>.-- The complexity of a problem is usually measured indirectly by measuring the complexity of the corresponding program. This approach presumably implies that there is a proportional relationship between problem complexity and program complexity. In the field of software engineering, the measurement of program complexity has long been a critical research issue. Many metrics have been proposed. However, many researchers have noted that the software science proposed by Halstead (2) represents "the most encompassing quantitative relevant theory (in learning and using query languages) (9, p. 173)." Other studies (8) have found that Halstead's metrics are more suitable for simple programs than complex ones. Since query programs in this study are written in a predicate calculus language, they can be considered as simple programs as opposed to equivalent ones written in procedural query languages. It seems that Halstead's metrics may be used for distinguishing simple problems from complex ones.

In order to ensure the validity of this choice, a validation process was performed on a particular metric, Volume (V) measure. This validation involved three steps. At the first step, a collection of database query problems were established based on a list of query problems found in Date's book (2, pp. 141, 142). At the second step, a query program was written for each of these problems in the predicate calculus language, Micro-Prolog, and V measures were taken for these programs based on a criteria of assignment of operators and operands. The third step performed a pooled t test on the two groups of problems (programs), and found a significant difference between the V values of simple problems and those of complex ones. In an attempt to examine possible effects of different assignment of operators and operands, a sensitivity analysis was conducted for several different assignments. This last step found no sensitive effect was caused by different assignments of operators and operands. As a consequence, the choice of Halstead's V measure has been justified. This metric is summarized as follows.

$$\text{Volume (V)} = (N1 + N2)\ \text{Log2}\ (n1 + n2)$$

Where

N1:	gross occurrences of operators
N2:	gross occurrences of operands
n1:	number of unique operators used
n2:	number of unique operands used
Log2:	Logarithm function based on 2

The validation process with the list of database query problems and the associated programs are described more clearly in Appendix A. In summary, the operational definition of _Problem Complexity_ is

> The complexity levels of database query problems are measured by the Halstead's Volume (V) metric as applied to the associated query programs written in Micro-Prolog.

Dependent Variables

Number of Syntactic Errors.-- In a query specification process, a syntactic error occurs when a spelling error is committed and/or a grammatical rule is violated. In the experimental system of this study, a syntactic error will cause the query specification process either to fail or to behave abnormally. The main reason for committing syntactic errors is unfamiliarity with the experimental query interface. Theoretically, users tend to make fewer syntactic errors with an easier-to-use interface. A good user interface should be able to improve users' productivity in specifying queries. As one of the productivity measures, syntactic errors in the query specification process were counted, and the number of these errors served as a dependent measure in the data analysis process.

Number of Semantic Errors.-- Semantic errors refer to incorrect specifications passing interpreter's syntactic check but resulting in incorrect outputs to be retrieved.

A semantic error can be caused by insufficient knowledge of the problem. In most cases, it arises from deficient mapping from the function of the query interface facility to the formulation of the problem solving solution. Therefore, in this sense, familiarity with the interface syntax represents "superficial" knowledge, whereas mastering the mapping procedure requires "deep" knowledge of the query interface facility. A good interface for end users should not only facilitate avoidance of syntactic errors, but also lead to minimal semantic errors.

Identification and categorization of possible semantic errors depends heavily upon the implementation of the interface. Based on the framework proposed in the previous section, several classes of semantic errors are listed as follows.

1. Incorrect order of relation names: With a nonprocedural language, most relation names can be specified in any order. However, this freedom of reference order applies to database relations only. In the predicate calculus language, conditional operators also appear in the same form as database relations. These predicate relations must be specified after the associated database relations. The reason for this is that binding of variables must occur before these variales can be manipulated in any way.
2. Incorrect relation names: This error refers to a case in which a relation is specified where it should not

appear. This error will cause wrong relations to be searched, and thus resulting in retrieval irregularities.

3. Incorrect attribute names or values: Attribute names function as variables, and will be bound to any values of the attributes. Attribute values are constants, and are bound to identical values of the attributes only. Specifying incorrect attribute names or values causes incorrect tuples being selected.
4. Incorrect conditional or Boolean operators: Conditional and Boolean operators allow specifications of predicates which restrict the data to be retrieved from the database. All query facilities have a set of predefined conditional and Boolean operators to support a variety of query requirements. Some facilities provide an easy way of specifying conditions correctly, while others require more knowledge of the system.
5. Non-descriptive relation names: When a relation other than any of the existing ones is specified, the system's search for the relation will fail. No output will be returned in this case. Since this error is concerned more with knowledge of domain databases than with the query interface, it is treated as a semantic error. Another reason for this treatment is because a program with this type of error will successfully pass the syntax check of the query processor.
6. Non-descriptive attribute names and condition operators:

A critical requirement of the experimental system is that all attributes must be given either a unique variable name or a correct constant. Depending on the specific situation, specifying a non-descriptive attribute name may produce unpredictable query results.

Although there might be some complicated relationships existing between different types of semantic errors, this study considers all types of semantic errors being equivalent due to their same consequence. The numbers of all types of semantic errors are summarized. This sum of the numbers of semantic errors is used as a dependent measure. It is assumed that a better query interface facility should lead to fewer semantic errors being committed in the query specification process.

Time of completion.-- The third dependent measure of this study is the amount of time required to complete a query specification activity. This variable was measured from the instant the user received a query problem and started to read the problem to the moment the query specification activity was complete. The time elapsed between these two points includes several components. The first component is the time a user spends for reading and comprehending the problem. The second component is the time used by a user when thinking about how to formulate a solution with the features provided by the query facility. The third component involves user typing time. The fourth component is the system response time. It is worth noting that these components are not necessarily mutually exclusive with each other. For example, some users may be typing while thinking. Since the main purpose of this study is to investigate the impact of different user assistance modes on users' query specificaton productivity. No attempt needs to be made to deal with individual time components. The gross time as represented by number of minutes was used in the data collection, and statistical analysis process.

Research Hypotheses

In order to answer the two research questions laid out in the first chapter, two sets of hypotheses are established for statistical testing. These hypotheses are devised to test the main effects as well as the interaction effects of the independent variables (User Assistance and Problem Complexity) on the dependent measures (Number of Syntactic Errors, Number of Semantic Errors, and Time to Completion). Each hypothesis is stated below in the null form. The rejection of a null hypothesis implies significant impacts of the investigated independent variable on the associated dependent measure.

H1a - The number of syntactic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance or the problem complexity.

H1b - The number of semantic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance or the problem complexity.

H1c - The amount of time required by novice users to complete a database query is the same regardless of the mode of use assistance or the problem complexity.

H2a - The number of syntactic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance.

H2b - The number of semantic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance.

H2c - The amount of time required by novice users to complete a query specification task is the same regardless of the mode of use assistance.

H3a - The number of syntactic errors committed by novice users in specifying database queries is the same regarless the type of problems encountered.

H3b - The number of semantic errors committed by novice users in specifying database queries is the same regardless of the type of problems encountered.

H3c - The amount of time required by novice users in specifying database queries to complete a query specification task is the same regardless of the type of problems encountered.

H4a - For simple problems, the number of syntactic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance.

H4b - For simple problems, the number of semantic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance.

H4c - For simple problems, the amount of time required by novice users to complete a query specification task is the same regardless of the mode of user assistance.

H5a - For complex problems, the number of syntactic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance.

H5b - For complex problems, the number of semantic errors committed by novice users in specifying databse queries is the same regardless of the mode of use assistance.

H5c - For complex problems, the amount of time required by novice users to complete a query specification task is the same regardless of the mode of use assistance.

Database query specification is a human-computer interaction activity. In order for this communication to proceed effectively and efficiently, either the user has to have sufficient knowledge about how to use the system or the system has to know much about the user's information requirement. Typically, functional managers possess a substantial body of knowledge about the problems in the domains of their interest. However, many managers tend to be less fluent in their use of computer language required to communicate with the computer. Therefore, incorporating knowledge about users' information requirements seems to be a feasible approach to facilitate easier communicaton between this type of users and the computer. Hypotheses H1a - H2c were devised to test this idea.

Complexity level of the problem has long been considered a key variable in the dynamics of human-computer interaction. For example, Welty (11) found no significant difference between writing queries in a procedural language and writing queries in a nonprocedural language for easy problems. However, it was found that procedural language is more appropriate for difficult problems. It seems to be a reasonable assumption that user assistance provided by a system may have different impacts for simple problems and complex ones. Empirical evidence can be obtained by testing Hypotheses H3a - H5c.

Experiment Design

This study is not only concerned with the main effects of use assistance mode and problem complexity, but also the interaction effect of these two variables. A factorial design is appropriate for this purpose. According to Montgomery (5, p. 124), the rationale for using a factorial design includes the following:

1. A factorial design is necessary when interaction effects

may be present.

2. The factorial design is more efficient than one-factor-at-a-time experiment design.
3. Factorial designs allow effects of a factor to be estimated at several levels of the other factors, yielding conclusions that are valid over a range of experimental conclusions.

The specific design of this study is depicted by Figure 3.

	Simple Problems	Complex Problems
User-Assisted Mode		
Non-User-Assisted Mode		

Figure 3--Experiment Design (2 X 2) of the Study

With this design, User Assistance Mode has two treatment levels. Both modes are provided with an interface facility which translates a more English-like query statement into the internal representation of the experimental system. Beyond this front-end support, Non-User-Assisted Mode provides no assistance for specifying database queries. In the non-assistance mode, users essentially have to construct a query program to satisfy the infromation retrieval requirement as indicated by each of a set of problems. The IVM and SDM values of this query specification mode is zero. The other mode (User-Assisted Mode) offers defaults for relations , operational predicates and some attribute values. Prompts are provided for users to enter the other attribute values to complete the query specification. The SDM value is 0.75, and the IVM is 0.25. The sum, i. e., IM, is 1.00 for this mode.

Therefore, query specification basically involves a question-answering process. Given a problem description, users select and enter a relevant command to initiate this question-answering process. When all the questions have been answered, the query program is complete and is executed. The command format of the user-assisted mode is the same as that of the non-user-assisted mode. The fact that both query specification modes have the same language syntax assures that language syntax will not constitute noise in the experiment.

Problem Complexity also has two treatment levels. Each level has four replications: four simple problems and four complex problems. The four problems at each level are semantically equivelent; only a slight difference exists in their information requirements. The volume measures of the four problems is 77.7, while the volume measures of the four complex ones range between 164.2 and 211.8. The significant different volume measures of the two sets of problems was intended to make the treatment strong enough so that the cause-effect relationship can be examined clearly. The query programs associated with all the problems are presented in Appendix 5 together with the way their volume values are calculated. A sensitivity analysis has been performed for different assignments regarding operators and operands. No significant impact of different assignments on the volume values was found. Appendix 1 presents more fully this validation process.

Data Collection

This study conducted an experiment in a controlled laboratory setting. An experimental database system is developed based on a PROLOG interpreter. This experimental system consists primarily of two databases, a set of high-level operators, and a collection of interactive query patterns implemented as inference rules. The system and the procedure in which the experiment was conducted are described in this section.

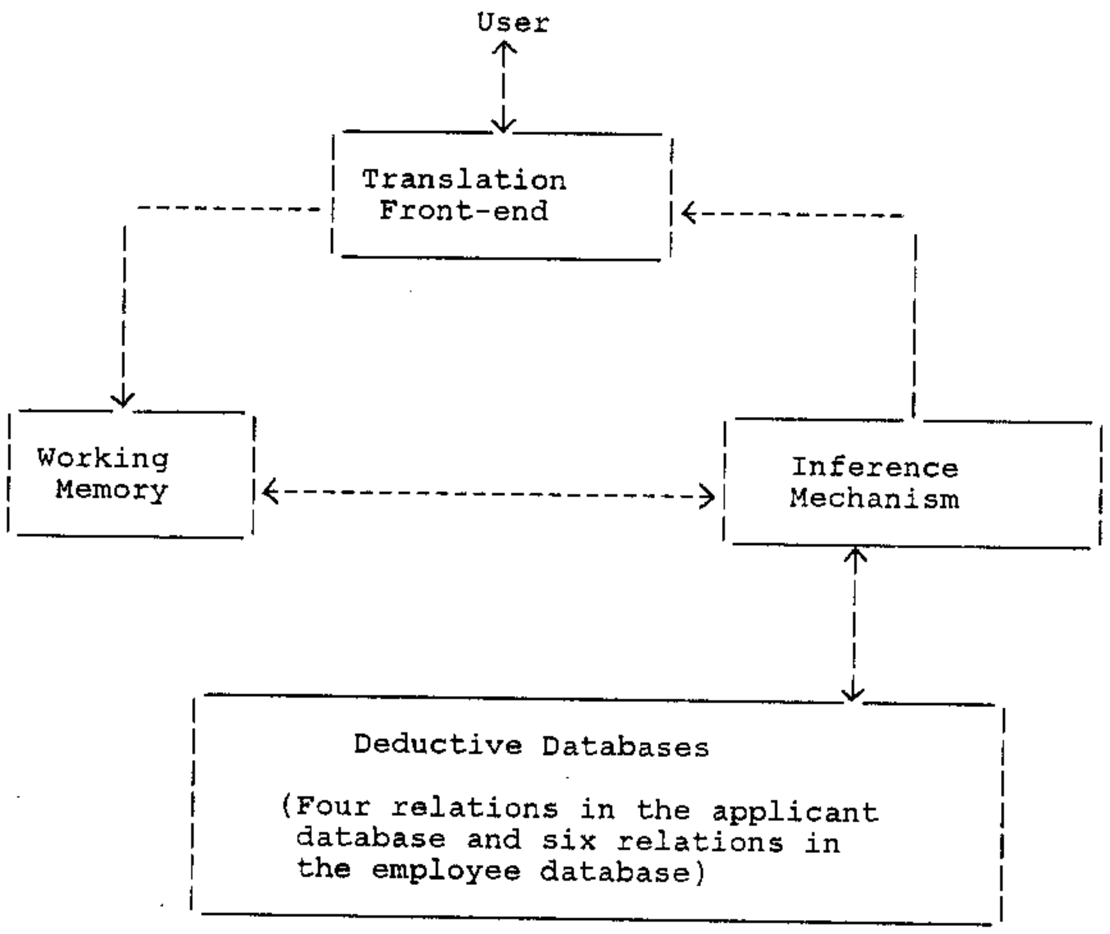
Design of the Experimental Database System

The experimental database system was developed based on a PROLOG interpreter running under MS-DOS, version 2.1, called micro-Prolog. Therefore, this is a deductive database system. Major components consist of a modified module of the micro-Prolog's language preprocessor, two application databases, and a collection of interactive query patterns.

System Architectures-- Both query specification modes have the following common components: translation front-end (a language preprocessor), inference mechanism, and two deductive databases. In addition to these common components, the user-assisted mode has an interactive query pattern module. The ways these two modes interact with the user are described as follows.

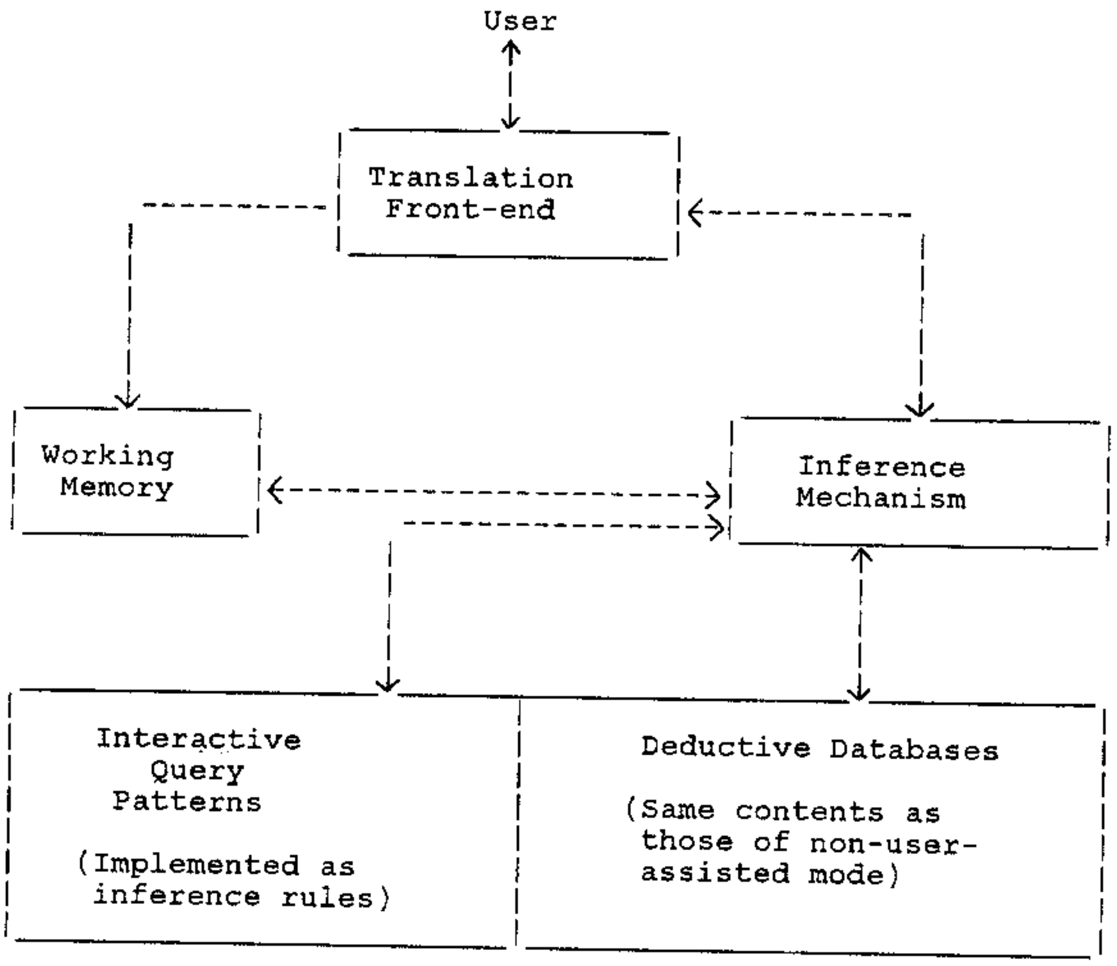
In the non-user-assisted mode, the user enters a query program in an easier-to-read format. This program basically states WHAT attribute values are to be extracted from WHAT relations subject to WHAT conditions. There is no specification in the program as to HOW the database should be navigated. This user-entered program will be translated into the internal representation by the translation front end and stored in the working memory. This front-end is essentially a version of a limited natural language processor coming with the interpreter diskette refined to fit this study's need. The internal representation of the program is in the format of a list. The inference mechanism evaluates the program using a depth-first, backward-chaining control strategy. The fundamental operation during this reasoning process is pattern-matching. When a pattern is found in the deductive database which matches the pattern of the subgoal, binding occurs and the value is returned as the side effect of the evaluation process. Evaluation of the program terminates when no (more) relations in the database can be found to match a subgoal of the program. Figure 4 illustrates this user-system interaction behavior.

In the user-assisted mode, the user selects and enters a command based on his understanding of the problem. The commands are implemented in the same language as that for non-user-assisted mode. As a result, the format of the commands is identical to that of query programs. This choice excludes likely noise arising from effects of differnent query languages. When the command is executed, an interactive query pattern is invoked. By going through a question-answering process, the user supplies data essential for the specification of a query under the assistance of the system. This interaction process is illustrated by Figure 5.


User
Translation Front-end
Working Memory
Inference Mechanism
Deductive Databases
(Four relations in the applicant database and six relations in the employee database)

Note: The user enters an English-like query program which is translated into the internal representation and is stored in the working memory. Inference Mechanism searches the databases when evaluating the program by pattern-matching and backtracking.

Figure 4--Architecture of the Experimental Database System for the Non-User-Assisted Mode.


User
Translation Front-end
Working Memory
Inference Mechanism
Interactive Query Patterns
(Implemented as inference rules)
Deductive Databases
(Same contents as those of non-user-assisted mode)

Note: Given a problem, the user selects and enters a command to invoke an interactive query pattern. Through a system-controlled question-answering process, the user supplies appropriate data to complete a query specification.

Figure 5--Architecture of the Experimental Database System for the User-Assisted Mode

Hierarchy of Database Views.-- As an integrated collection of data resources, a database typically has various users interfacing with it. These users play different roles in the life cycle of the database. For example, database administrators design, implement, and maintain the whole database. Application programmers develop applications which obtain inputs from the database. End users extract from the database information pertaining to their jobs. Each class of users holds a unique view of a database. The ANSI/SPARC architecture basically proposes a hierarchy of views for a database. Each view is built on top of the next lower one to represent a next higher abstraction of the original database. For example, the internal view regards the database as a set of storage and search strategies. The conceptual view treats the database from the standpoint of inter-records relationships. And the external view represents a logical window through which a user sees the portion of a database which is related to his scope of responsibility. A major function of the external view is provision of data independence; the user is shielded from the complexity of the original design. This notion can be extended even further: multiple levels of external views can be developed to provide simpler views. The use-assisted mode of this study represents an implementation of a second-level external view (Figure 6).

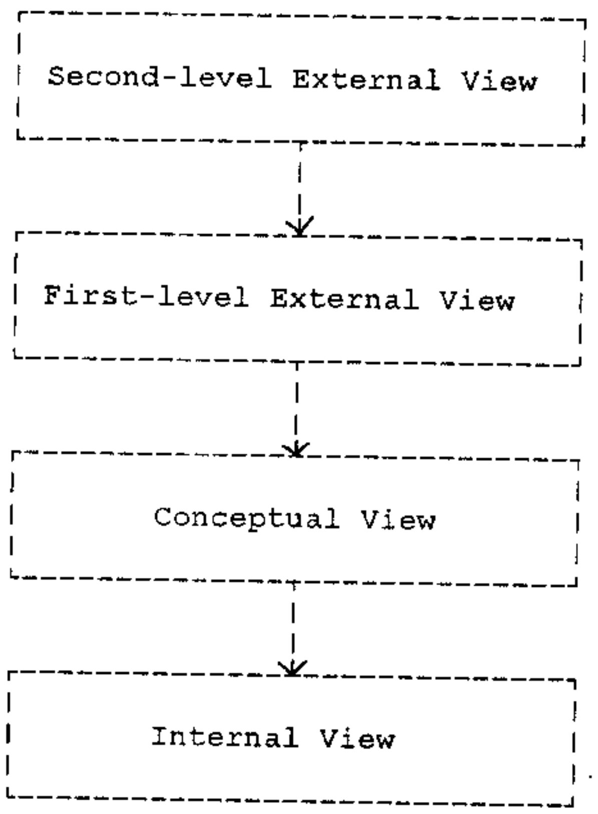


Figure 6--Hierarchy of Database Views

Conceptual Model of the Databases.-- A conceptual data model provides a comprehensive logical view of a database, and should serve as a foundation for all applications. This study involves two subject databases: Applicant database and Employee database. The Applicant database contains information about job applicants, whereas the employee database contains information about existing employees of a hypothetical, management information systems department. These conceptual models are presented as follows.

Applicant Database:

1. Applicant-info(SS# Name Address Phone-no Degree Major Position-wanted Expected-annual-pay)

2. Experience-applicant(SS# Skill-name Number-of-years)

Employee Database:

1. Employee-info(Employee-ID Name Address Phone-no major Department Title Date-in Communication-grade)

2. Experience-employee(Employee-ID Skill Number-of-years)

3. Project-completion(Employee-ID Current-project Date-of-completion)

A commonly used convention is adopted here for the depiction of a relation: a relation name followed by a list of attribute names which are specified within a pair of parentheses. Primary keys are underlined. Secondary keys are not shown in the following models. From the perspective of the theory of functional dependency, all of the relations are in the third normal form.

Implementation models of the databases.-- Since a conceptual data model is typically independent of any database management software, some modification is often needed to take into account various constraints when the database is implemented. These constraints include DBMS characteristics, application processing requirements, user characteristics, etc. In this study, two relations (Applicant-info and Employee-info) seem to be containing too many attributes. Breaking them down can also offer more flexibility for possible future growth of the system. Shown below are the implementational (operational) models of the experimental databases derived from the conceptual models.

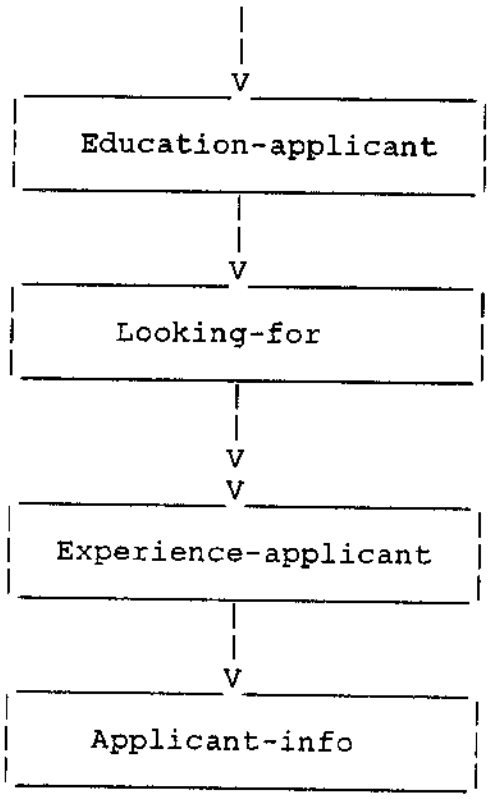
Applicant Database:

1. Applicant(SS# Name Address Phone-no)
2. Education-applicant(SS# Degree Major)
3. Experience-applicant(SS# Skill Number-of-years)
4. Looking-for(SS# Position-wanted Expected-pay)

Employee Database:

5. Employee(Employee-ID Name Address Phone-no)
6. Education(Employee-ID Degree Major)
7. Job-title(Employee-ID Department Title Date-in)
8. Experience(Employee-ID Skill Number-of-years)
9. Communication-skill(Employee-ID Grade)
10. Project-status(Employee-ID Current-project Date-of-completion)

Logical Access Maps for the Interactive Query Patterns.-- A logical access map identifies the sequence in which the relations are accessed. This technique was proposed and explained by Martin (4, pp. 361 - 374), and has been claimed to well suit application specification needs for database environments. From a logical access map, a database action diagram can be drawn in a straightforward way. In a nonprocedural language, developing an application requires only minimal translation from the database action diagram to the source code. In the course of developing the experimental system, this study also finds logical access maps a convenient conceptualization tool for the design and implementation of interactive query patterns. Figure 7 illustrates an example of these logical access maps.



Note: This query pattern is designed for extracting information from the Applicant database. The single arrow indicates that the relation is specified only once, while the double arrow indicates the relation is specified multiple times. However, the inference mechanism's backtracking feature may return multiple values as the result of pattern-matching

Figure 7--Logical Access Maps of the Interactive Query Patterns

Experiment Procedure.-- This study used two sections of a fundamental course offered by the Department of Business Computer Information Systems at North Texas State University as experimental subjects. Fifty students were randomly divided into a control group and an experimental group. These students are characterized by a lack of extensive experience in the use of computer, hence can be considered a convenient sample of the population addressed by this study, novice users. The experiment was made a mandatory , graded assignment so that students would pay serious attention to their performance.

Two databases were created for a personnel management domain. For the user-assisted mode, a collection of interactive query patterns was developed. There were eight database query problems to be solved. Four problems concerned selecting from the applicant database job applicants qualified for a particular job opening, while the other four were related to project team assignment and the employee database was involved. Among these eight problems, four were simple problems, whereas the other four were complex. The complexity levels were measured by Halstead's Volume metric. The two classes of problems were randomly mixed when presented to the subjects. This arrangement is an attempt to mimic the real world situation where various kinds of problems arrive in an unpredictable fashion. A functional manager must be able to match a computing tool's functional capability with the problem's semantic requirements. A user-friendly tool should be able to provide aid in this respect.

The experiment was conducted over a period of five days. Two to four subjects participated in the experiment in each session. When arriving at the scheduled date and time, the subject had no idea about the details of the experiment. After being given a brief but clear introduction, the subject was given a tutorial , written instruction on the experimental system and example problems to be solved (Appendices B, C, D, and E). The subjects were encouraged to ask any questions when reading the instructions. There was no limitation on the amount of time used for reading the instructions. Actual lengths ranged anywhere between fifteen minutes and ninety minutes. At the end of reading, the subjects indicated that they had felt comfortable about the system and the problems.

Problems for all subjects were in the same sequence, and all problems were given to a subject at a time. To ensure the correctness of the time recorded, the administrator of the experiment stood nearby to double check the time they started and finished each problem. When the subject inadvertently ran into a long loop, or when the keyboard stuck, the administer solved the problem immediately so that the experimental result wouldn't be contaminated by irrelevant time measurements. The entire problem-solving process was conducted on-line for both query specification modes (Appendices G and H provide an outline of system operating procedures for both modes). A printer was used to capture both users' input and system's output to facilitate data analysis. Conducting the experiment for both experimental group and controlled group on-line helped to eliminate possible discrepancies in user performance which may exist between the on-line and off-line environments.

Data Analysis

All users' inputs and system's outputs recorded by the printer were analyzed according to the criteria established and described in the measurement section. In order to encode the experimental data in an organized manner, a data sheet was prepared. This data sheet includes subjects' demographical data, as well as dependent measures. Figure 8 shows its format.

EXPERIMENT DATA SHEET

1. Subject Name: __________ Sex: _____ Age: _____
   Major: _________ Computer Courses Completed: __________
   Software Use Experience: ____________________

2. Query Specification Mode: __________

3. Number of Syntactic Errors:

Spelling Error:	__	__	__	__	__	__	__	__
Grammatical Violation:	__	__	__	__	__	__	__	__
Total Syntactic Errors:	__	__	__	__	__	__	__	__

4. Number of Semantic Errors:

Incorrect order of relations:	__	__	__	__	__	__	__	__
Incorrect Relation Names:	__	__	__	__	__	__	__	__
Incorrect attribute Names	__	__	__	__	__	__	__	__
Incorrect conditional or boolean operators:	__	__	__	__	__	__	__	__
Non-descritive relation:	__	__	__	__	__	__	__	__
Non-descriptive attribute names and condition operator:	__	__	__	__	__	__	__	__
Total Semantic Errors:	__	__	__	__	__	__	__	__

5. Time of Completion: (minutes)

Time of Completion: (minutes)	__	__	__	__	__	__	__	__

Figure 8--Experiment Data Sheet Used for Grading

Due to some unique syntax of the implementation language, micro-Prolog, several notes need to be made regarding the classification of errors found in the query specification. First, grammatical errors include the following cases:

1. The key word, and, is missing.
2. Parentheses are not balanced.
3. Given a correct relation, the number of variables and constants together are not identical to what it is supposed to be.

4. Separating blanks are missing.

Second, since there is a hierarchical relationship between certain error types (wrong attributes are always specified if relations are not correct, for example), the grading process only counted the error at the highest level to avoid overemphasis on some user-made errors. This practice was performed for both modes to maintain essential consistency. Third, any other errors not exactly identified by the list of semantic error types were assigned to the error type items closest to the actual errors. For example, extra relation(s) specified was regarded as wrong relation(s). Since only total number of all semantic errors is used as a dependent measure, the results will not be affected by minor deviations from a truly exhaustive list of semantic errors. This confidence is further enforced by the consistency of practices for both modes.

The resulting data were analyzed by Analysis of Variance (ANOVA) procedure of the Statistical Analysis System (SAS), Release 5.08, running under OS/MVS. Since each dependent measure was to be analyzed separately, three ANOVA runs were performed for the three dependent measures with simple and complex problems togethers. Then, another three ANOVA analyses were performed with either simple or complex problems removed. This procedure resulted in nine sets of ANOVA test results.

Limitations of the Study

Limitations to the external validity of this study can be caused by the use of student subjects, hypothetical sample databases and problems, single data model, and single implementation language. Findings of the study will not be overgeneralized for these reasons. Another source of limitation may come from a drawback found in most laboratory experiments: relatively weak treatments. This study uses two very distinct levels for both independent variables, in an attempt to significantly increase the strengths of the treatments. When resource is sufficient, real world subjects can be used to either confirm or modify the findings of this study with problems derived directly from the real world. However, due to these limitations, all findings will not be overly generalized.

CHAPTER BIBLIOGRAPHY

1. Bonczek, R. H., C. W. Holsapple, and A. B. Whinston, " A Generalized Decision Support System Using Predicate Calculus and Network Data Base Management," Operations Research , Vol. 29, No. 2, (March-April 1981), pp. 263 - 281.

2. Date, C. J., Introduction to Database Systems, (2nd Edition), Menlo Park, California, Addison-Wesley Publishing, Co., 1981.

3. Halstead, M., Elements of Software Science , New York, Elsevier Computer Science Library, 1977.

4. Martin, James, Managing Data Base Environment , Englewood, New Jersey, Prentice-Hall, 1983.

5. Montgomery, D. C., Design and Analysis of Experiments , Englewood, New Jersey, Prentice-Hall, 1976.

6. Ray, Howard, An Empirical Investigation of the Effects of Individual Differences and Data Models on the Ease-of-Use of Database Query Facilities by Casual Users , PhD Dissertation, North Texas State University, 1984.

7. Reisner, Phllis, "Human Factors Studies of Database Query Languages: A Survey and Assessment," Computing Surveys , 13, 1, (March 1981), pp. 13 - 31.

8. Shneiderman, Ben, Software Psychology , Cambridge, Massachusetts, Winthrop Publisers, Inc., 1980.

9. Thomas, J. C., "Psychological Issues on the Design of Database Query Languages," in Designing for Human-Computer Communication , edited by Sime, M. E., and M. J. Coombs, New York, Academic Press, 1983, pp. 173 - 208.

10. Ullman, J. D., Principles of Database Systems , Rockville, Maryland, Computer Science Press, 1982.

11. Welty, C., and D. W. Stemple, "Human Factors Comparison of a Procedural and a Nonprocedural Query Language," ACM Transactions on Database Systems , Vol. 6, No. 4, (September 1981), pp. 626 - 649.

CHAPTER IV

RESULTS AND ANALYSIS

This study involves two independent variables (User Assistance Query Mode and Problem Complexity) and three dependent variables (Number of Syntactic Errors, Number of Semantic Errors, and Time of Completion). In order to analyze the independent (main) as well as the interaction effects of the two independent variables on a dependent variable, a factorial analysis of variance was used. This chapter first presents the distribution of some characteristics of the subjects. Then, the results of statistical analysis are reviewed from the perspective of the hypotheses established in Chapter III.

Demographical Distribution of the Subjects

Most students participating in this study are majoring in business. As shown in Table III, the major area covers a wide range. The range of age in the use-assisted query specification mode (the experiment group) is 20 - 39, with the mean of 23. In the non-use-assisted mode (the control group), this range is 21 - 34, with the mean of 24. Thirteen females and twelve males are in the experiment group, while the control group has fourteen females and eleven males. The relatively lower age of the experiment subjects than the real world counterpart constitutes one of the critical factors limiting the exernal validity of the study.

TABLE III

DISTRIBUTION OF SUBJECTS' MAJORING AREAS

Major Areas	Number of Subjects		
	Total	User-Assisted Mode	Non-User-Assisted Mode
Marketing	9	3	6
Finance	7	2	5
General Business	6	3	3
Accounting	5	4	1
Management	5	1	4
Production Management and Organizational Behavior	4	2	2
Business Computer Information Systems	4	3	1
Undecided	3	2	1
Personnel and Industrial Relations	2	2	0
Political Science	2	1	1
Computer Science	2	1	1
Real Estate	1	1	0

The majority of the subjects (95 percent) have completed only an introductory computer course in the past five years. Only the two subjects majoring in computer science have had more computer-related courses. The same percentage of subjects had no prior experience in using any application software. None of them had any experience in using database management software. In general, this sample group is characterized by the same minimal level of computer knowledge as that of the population addressed by the study.

Overall Effects of Use Assistance Mode and Problem Complexity

The objective of the study is to investigate possible effects arising from each independent variable as well as from the interaction of the use assistance mode and problem complexity. The first attempt of the analysis is to examine the overall effect of these two independent variables as a whole. If the overall effect is not significant, it is still possible that one of the variables has a significant effect, but that effect has been obscured by its interaction with another variable. On the other hand, if there is a significant overall effect, this overall effect can be broken down to several components to provide more information about the sources of the effect.

Hypothesis H1a states: The number of syntactic errors committed by the novice users in specifying database queries is the same regardless of the modes of use assistance and the complexity levels of the problems. This null hypothesis has been rejected by the F test at a significance level of 1 percent (Table IV).

Hypothesis H1b states: The number of semantic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance and the complexity levels of the problems. The result of analysis of variance rejects this null hypothesis at a significance level of 1 percent (Table V).

Hypothesis H1c states: The amount of time required by novice users in specifying database queries is the same regardless of the modes of use assistance and the complexity levels of the problems. At the significance level of 1 percent, this null hypothesis is rejected (Table VI). It is therefore obvious that users' productivity in specifying database queries is affected by the use assistance provided by the system and the complexity levels. Together, these two factors contribute to the significant difference of productivity measures in some way. The remainder of this chapter examines the main effects as well as the interaction effect of these two factors.

TABLE IV

ANALYSIS OF VARIANCE FOR THE NUMBER OF SYNTACTIC ERRORS WITH BOTH SIMPLE AND COMPLEX PROBLEMS INCLUDED

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F
MODEL	3	68.76750000	22.92250000	18.91	0.0001
ERROR	396	479.93000000	1.21194444		
CORRECTED TOTAL	399	548.69750000			

SOURCE	DF	ANOVA SS	F VALUE	PR > F
QRYMODE	1	39.06250000	32.23	0.0001
PROBLEM	1	25.50250000	21.04	0.0001
QRYMODE* PROBLEM	1	4.20250000	3.47	0.0633

TABLE V

ANALYSIS OF VARIANCE FOR THE NUMBER OF SEMANTIC ERRORS WITH BOTH SIMPLE AND COMPLEX PROBLEMS INCLUDED

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F
MODEL	3	101.01000000	33.67000000	19.25	0.0001
ERROR	396	692.50000000	1.74873737		
CORRECTED TOTAL	399	793.51000000			
SOURCE	DF	ANOVA SS	F VALUE	PR > F	
QRYMODE	1	62.41000000	35.69	0.0001	
PROBLEM	1	27.04000000	15.46	0.0001	
QRYMODE* PROBLEM	1	11.56000000	6.61	0.0105	

TABLE VI

ANALYSIS OF VARIANCE FOR THE TIME OF COMPLETION WITH BOTH SIMPLE AND COMPLEX PROBLEMS INCLUDED PROBLEMS ARE INCLUDED

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F
MODEL	3	2687.96187500	895.98729167	35.00	0.0001
ERROR	396	10137.14750000	25.59885732		
CORRECTED TOTAL	399	12825.10937500			
SOURCE	DF	ANOVA SS	F VALUE	PR > F	
QRYMODE	1	1574.10562500	61.49	0.0001	
PROBLEM	1	883.57562500	34.52	0.0001	
QRYMODE* PROBLEM	1	230.28062500	9.00	0.0029	

Main Effect of Use Assistance Mode on the Number of Syntactic Errors

The number of syntactic errors committed by the user in the query specification process reflects the structural complexity of an interface. An interface which is easier to use tends to lead to fewer syntactic errors. As an attempt to investigate the effect of use assistance provided by the system in this regard, Hypothesis H2a states: The number of syntactic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance. This hypothesis concerns the main effect of Use Assistance when both simple and complex problems are present. The test result shown in Table IV (F value = 32.23) rejects the hypothesis at the significance level of 1 percent. It clearly indicates that significant difference of the number of syntactic errors exists between the two use assistance modes. Another analysis results shows that the average number of syntactic errors made in the non-use-assisted mode is 0.965, while that in the use-assisted mode is 0.34. This result represents an empirical evidence that the use-assisted mode results in higher user productivity marked by fewer syntactic errors.

In the case that only simple problems or only complex problems are present, what would be the main effect of the use-assisted mode on the number of syntactic errors? Hypotheses H4a and H5a are devised to investigate this issue. H4a states: For simple problems, the number of syntactic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance. H5a states: For complex problems, the number of syntactic errors committed by novice users is the same regardless of the mode of use assistance. Tables VII and VIII show that both null hypotheses are rejected at the significance level of 1 percent. Further analysis shows that , for simple problems, the average number of syntactic errors per problem is 0.61 in the non-use-assisted mode, and 0.19 in the use-assisted mode. For complex problems, this average is 1.32 in the non-use-assisted mode, and 0.49 in the use-assisted mode. It is clear that Use-Assisted Mode consistently results in fewer syntactic errors regardless the complexity level of the problems encountered by the users.

TABLE VII

ANALYSIS OF VARIANCE FOR THE NUMBER OF SYNTACTIC ERRORS FOR SIMPLE PROBLEMS

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F
MODEL	1	8.82000000	8.82000000	13.95	0.0002
ERROR	198	125.18000000	0.63222222		
CORRECTED TOTAL	199	134.00000000			
SOURCE	DF	ANOVA SS	F VALUE	PR > F	
QRYMODE	1	8.82000000	13.95	0.0002	
PROBLEM	0	0.00000000	.	.	
QRYMODE* PROBLEM	0	0.00000000	.	.	

TABLE VIII

ANALYSIS OF VARIANCE FOR THE NUMBER OF SYNTACTIC ERRORS ERRORS FOR COMPLEX PROBLEMS

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F
MODEL	1	34.44500000	34.44500000	19.23	0.0001
ERROR	198	354.75000000	1.79166667		
CORRECTED TOTAL	199	389.19500000			
SOURCE	DF	ANOVA SS	F VALUE	PR > F	
QRYMODE	1	34.44500000	19.23	0.0001	
PROBLEM	0	0.00000000	.	.	
QRYMODE* PROBLEM	0	0.00000000	.	.	

Main Effect of Use Assistance Mode on the Number of Semantic Errors

A computer interface facility such as the one used in this study is merely a tool assisting the user in problem solving activities. The user must be able to map the functional capability provided by the interface to a specifc implementation of the associated query activity. This study uses the number of semantic errors committed by the user to reflect the relative ease of use of the interface in terms of this conceptualization aid. Would different use assistance modes lead to different productivity as measured by the number of semantic errors found in the query specification process? Hypotheses H2b, H4b, and H5b are devised to investigate this issue.

Hypothesis H2b states: The number of semantic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance. This hypothesis is tested with both simple and complex problems included. As shown in Table V, the F value of Use Assistance Mode is 35.69, rejecting H2b at the significance level 1 percent. Analysis also shows that the average number of semantic errors per problem is 1.18 in the non-use-assisted mode , and 0.39 in the use-assisted mode. Thus, regardless of complexity level of the problems, the use-assisted mode tends to result in fewer semantic errors.

H4b and H5b attempt to investigate the effect of the use assistance on the number of semantic errors with simple and complex problems treated separately. H4b states: For simple problems, the number of semantic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance. H5b states: For complex problems, the number of semantic errors committed by novice users in specifying database queries is the same regardless of the mode of use assistance. The analysis results shown in Tables IX and X reject both null hypotheses at a significance level of 1 percent. Analysis of data also reveals that the average number of semantic errors per simple problem is 0.75 in the non-use-assisted mode, and 0.30 in the use-assisted mode. For complex problems, this average is 1.61 for the non-use-assisted mode, and 0.48 for the use-assisted mode. These consistent results provide strong evidence that the use-assisted mode does tend to lead to higher user productivity indicated by fewer semantic errors found in the query specification process.

TABLE IX

ANALYSIS OF VARIANCE FOR THE NUMBER OF SEMANTIC ERRORS FOR SIMPLE PROBLEMS

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F
MODEL	1	10.12500000	10.12500000	7.96	0.0053
ERROR	198	251.75000000	1.27146465		
CORRECTED TOTAL	199	261.87500000			

SOURCE	DF	ANOVA SS	F VALUE	PR > F
QRYMODE	1	10.12500000	7.96	0.0053
PROBLEM	0	0.00000000	.	.
QRYMODE* PROBLEM	0	0.00000000	.	.

TABLE X

ANALYSIS OF VARIANCE FOR THE NUMBER OF SEMANTIC ERRORS FOR COMPLEX PROBLEMS

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F
MODEL	1	63.84500000	63.84500000	28.68	0.0001
ERROR	198	440.75000000	2.22601010		
CORRECTED TOTAL	199	504.59500000			

SOURCE	DF	ANOVA SS	F VALUE	PR > F
QRYMODE	1	63.84500000	28.68	0.0001
PROBLEM	0	0.00000000	.	.
QRYMODE* PROBLEM	0	0.00000000	.	.

Main Effect of Use Assistance on the Amount of Time Required to Complete a Query

Users' time is precious. Users tend to resist using a computing tool if much time is required for the use of it. This study investigates possible effects of use assistance provided by the system on the amount of time for completing a query by Hypotheses H2c, H4c, and H5c. H2c examines the issue with both simple and complex problems included, while H4c and H5c deal with simple problems and complex problems, respectively.

Hypothesis H2c states: The amount of time required by novice users to complete a query specification task is the same regardless of the mode of use assistance. Hypothesis H4c states: For simple problems, the amount of time required by novice users to complete a query specification task is the same regardless of the mode of use assistance. Hypothesis 5c states: For complex problems, the amount of time required by novice users to complete a query specification task is the same regardless of the mode of use assistance. According the test results shown in Table VI, Table XI, and Table XII, F value of Use-Assistance Mode is 61.49 when both simple and complex problems are present, 19.22 when only simple problems are present, and 42.27 when complex problems only are present. These high F values reject the three hypotheses at a significance level of 1 percent.

The average amount of time required to complete a query, regardless of the complexity level of the problems, is 8.8 minutes with the non-use-assisted mode, and 4.8 minutes with the use-assisted mode. With the use-assisted mode, each simple problem requires, on the average, 4.1 minutes, while each complex problem requires 5.5 minutes. With the non-use-assisted mode, each simple problem requires 6.5 minutes, and each complex problem requires approximately 11 minutes. A consistent pattern seems to be clear: the use-assisted mode requires less time on the user's part to complete a query specification activity. If the assumption that the user interface requiring less time for the user to complete a query is preferred by the user is correct, then incorporating appropriate use assistance in the system would be an effective approach.

The fact that null hypotheses H1a - H1c, H2a - H2c, and H5a - H5c are all rejected by F test of ANOVA at a very significant level suggests an interesting finding. Although the three dependent measures used in this study represent different dimensions of user's productivity in specifying database queries, they appear to be affected by the use assistance of the system in the same manner. This implies that perhaps one of them can be more economically used as the sole indicator of user's productivity in query specification.

TABLE XI

ANALYSIS OF VARIANCE FOR THE TIME OF COMPLETION FOR SIMPLE PROBLEMS

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F
MODEL	1	300.12500000	300.12500000	19.22	0.0001
ERROR	198	3091.47000000	15.61348485		
CORRECTED TOTAL	199	3391.59500000			

SOURCE	DF	ANOVA SS	F VALUE	PR > F
QRYMODE	1	300.12500000	19.22	0.0001
PROBLEM	0	0.00000000	.	.
QRYMODE* PROBLEM	0	0.00000000	.	.

TABLE XII

ANALYSIS OF VARIANCE FOR THE TIME OF COMPLETION FOR COMPLEX PROBLEMS

SOURCE	DF	SUM OF SQUARES	MEAN SQUARE	F VALUE	PR > F
MODEL	1	1504.26125000	1504.2612500	42.27	0.0001
ERROR	198	7045.67750000	35.5842298		
CORRECTED TOTAL	199	8549.93875000			

SOURCE	DF	ANOVA SS	F VALUE	PR > F
QRYMODE	1	1504.26125000	42.27	0.0001
PROBLEM	0	0.00000000	.	.
QRYMODE* PROBLEM	0	0.00000000	.	.

Main Effect of Problem Complexity on the Number of Syntactic Errors, the Number of Semantic Errors, and the Amount of Time Required to Complete a Query

The analysis of variance also reveals the main effect of problem complexity on the three dependent measures. Although the main concern of this study is with the impact of use assistance, knowledge regarding the effect of problem complexity can supplement our understanding of this impact. Therefore, hypotheses H3a - c were established to facilitate this investigation.

Hypothesis H3a states: The number of syntactic errors committed by novice users in specifying database queries is the same regardless of the type of problems encountered. Hypothesis 3b states: The number of semantic errors committed by novice users in specifying database queries is the same regardless of the type of problems encountered. Hypothesis H3c states: The amount of time required by noivce users to complete a database query specification task is the same regardless of the type of problems encountered. The F values associated with these three hypotheses are 21.04, 15.46, and 34.52 (see Table IV, Table V, and Table VI). All the three null hypotCeses are rejected at the significance level of 1 percent.

When examining the average values of each dependent measure in terms of complexity levels of problems, a consistent pattern is identified. With the non-use-assisted mode, the average number of syntactic errors for simple problems is 0.61, the average number of semantic errors 0.75, the average amount of time required for completing a query 6.5 minutes. With the same mode, for complex problems, the average number of syntactic errors is 1.32, the average number of semantic errors 1.61, the average amount of time for completion approximately 11 minutes. These statistics are in contrast to those with the use-assisted mode. For simple problems, the averages are 0.19, 0.30, 4.07, respectively. For complex problems, the averages are 0.49, 0.48, and 5.5 minutes. With the use-assisted mode, users tend to perform query specification tasks with higher productivity.

This empirical evidence supports the common sense regarding the impact of problem complexity: more complex problems require more effort to solve. Nevertheless, it serves to reinforce the finding that use assistance does provide a significant contribution to improving users' productivity in query specification effort over a wide spectrum of problem complexity. This consistent pattern also adds to the validity of Halstead's Volume metric as a complexity measure.

Interaction Effect of User Assistance Mode and Problem Complexity on the Number of Semantic Errors, and the Amount of Time Required to Complete a Query

Table V shows that the interaction effect of Use Assistance Mode and Problem Complexity appears not to be ignorable. The F value of 6.61 indicates that the interaction effect is significant at slightly below the 1 percent level. The same situation is found in Table VI where the interaction effect on the amount of time required for the completion of a query appears to be significant at the 1 percent level. There is clearly some dynamic relationship existing between the use assistance mode and the problem complexity. Further analysis is necessary to arrive at sound interpretation of this phenomenon.

Referring to Figure 9, with the non-use-assisted mode, the average number of semantic errors for each complex problem is 1.61, for each simple problem 0.75. With the use-assisted mode, these averages are 0.48 and 0.30, respectively. When linking the two points representing complex problems, and the two points for simple problems, the two line segment are clearly not parallel to each other. This is a graphical illustration of significant interaction effect.

Figure 10 shows that the distance between the two points representing the amount of time of completion with the non-use-assisted is much wider than that between the two points representing the use-assisted mode. With the non-use-assisted mode, the average time for completing each complex problem is 11 minutes, simple problem 6.5 minutes. With the use-assisted mode, the averages are 5.5 minutes and 4.1 minutes, respectively. A significant interaction effect is graphically illustrated.

The interaction effects observed in these two cases can be explained by two possible reasons. First, in the case of number of semantic errors, it is possible that the average number of semantic errors has been so low that reducing this measure very much is not likely. Second, in both cases, perhaps complex problems do benefit more than simple ones from the use assistance provided by the system. It is also possible that both reasons hold. In any case, it seems to be a reasonable conclusion that use assistance does provide a useful aid in dealing with complexity associated with the problems encountered by the users.

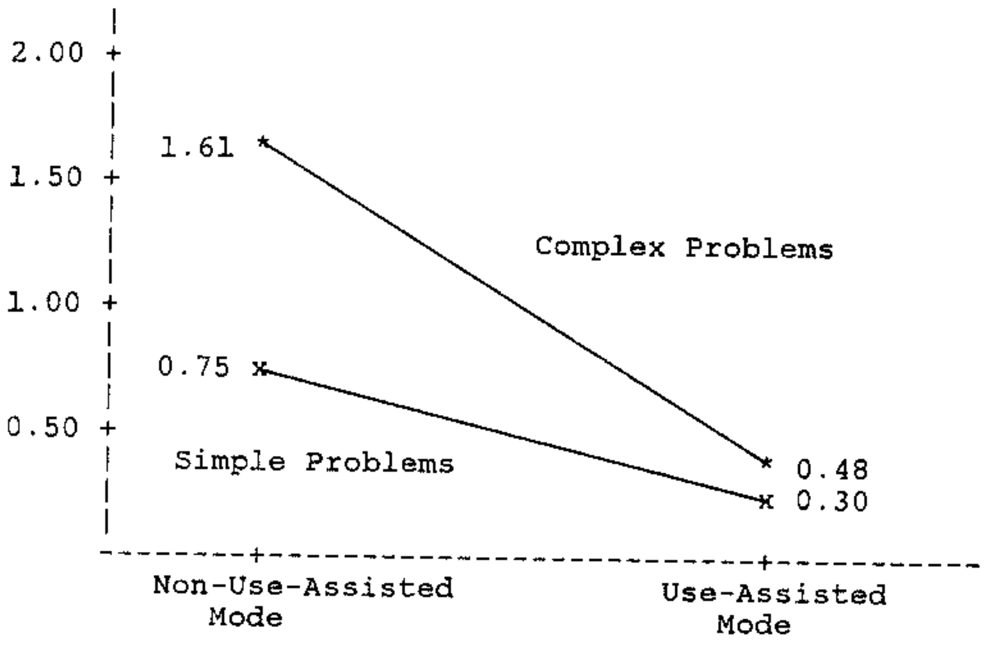


Figure 9--Interaction Effect of Use Assistance and Problem Complexity on the Number of Semantic Errors

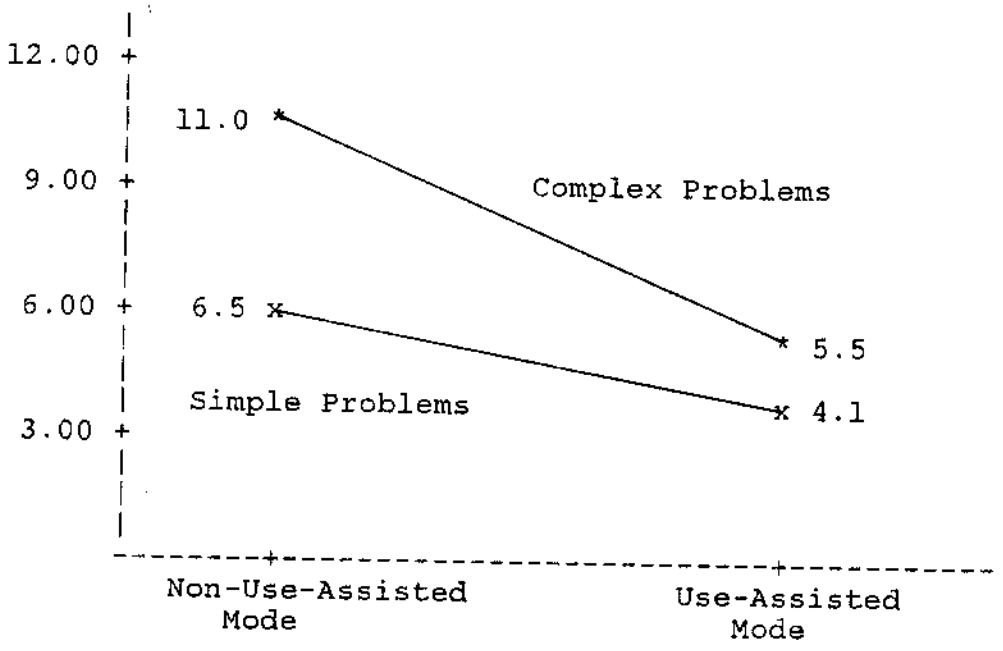


Figure 10--Interaction Effect of Use Assistance and Problem Complexity on the Amount of Time Required to Complete a Query

Interaction Effect of Use Assistance Mode and Problem Complexity on the Number of Syntactic Errors

When plotted on the same scale as that for the number of semantic errors, the interaction on the number of syntactic errors does not appear to be so significant. Table IV shows that the F value for this interaction effect is 3.47. This means that no significant interaction effect exists at the significance level of 0.01 and 0.05. If the significance level of 0.10 (or even 0.07) is used, this interaction effect will be regarded as significant. A conservative conclusion is that there may be some interaction effect on this dependent measure, the number of syntactic errors. This conclusion is illustrated in Figure 11.

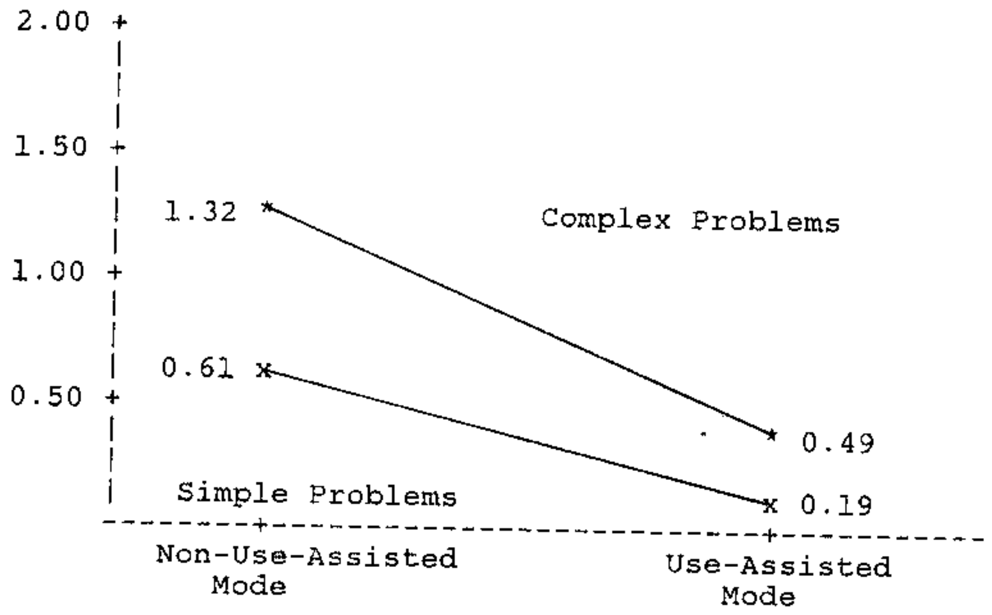


Figure 11--Interaction effect of Use Assistance and Problem Complexity on the number of syntactic errors

Summary of the Results of Hypotheses Testing

With Analysis of Variance, it has been found that all the null hypotheses established in the study have been rejected at the 1% significance level. The critical F value, the calculated F value, and the probability of rejecting the true null hypothesis associated with each hypothesis are listed in the following summary table for ease of reading (Table XIII).

TABLE XIII

SUMMARY OF RESULTS OF HYPOTHESES TESTING

Hypothesis	Critical F Value (Alpha = 0.01)	Calculated F Value	PR*
H1a	3.78	18.91	0.0001
H1b	3.78	19.25	0.0001
H1c	3.78	35.00	0.0001
H2a	6.63	32.23	0.0001
H2b	6.63	35.69	0.0001
H2c	6.63	61.49	0.0001
H3a	6.63	21.04	0.0001
H3b	6.63	15.46	0.0001
H3c	6.63	34.52	0.0001
H4a	6.76	13.95	0.0002
H4b	6.76	7.96	0.0053
H4c	6.76	19.22	0.0001
H5a	6.76	19.23	0.0001
H5b	6.76	28.68	0.0001
H5c	6.76	42.27	0.0001

* Probability of rejecting the hypothesis while it is actually true.

CHAPTER BIBLIOGRAPHY

1. Cochran, W. G., and G. M. Cox, Experimental Designs , Second Edition, New York, Wiley, 1957.

2. Kerlinger, Fred N., Foundations of Behavioral Research , New York, Holt, Rinehart and Winston, Inc., 1973.

3. SAS User's Guide , Cary, North Carolina, SAS Institute Inc., 1985.

CHAPTER V

CONCLUSIONS

User/system interface is a critical success factor of a database system aimed at providing functional manager users easy access to the database. This class of users is characterized a rich body of knowledge about their problem domains and a lack of knowledge which would permit fluent manipulation of the computer system. In order to present an interface which would enhance users' productivity in specifying database queries, this study investigates an idea derived from research in a number of fields. Generally, this study suggests that the design of database query interface for this class of users may take advantage of users' prior knowledge of information requirements. When incorporating this knowledge into the system at system design time, a collection of interactive query patterns is created. A question-answering facility is recommended because of the need for flexibility: users' inputs are bound at the run time.

This chapter first summarizes the critical findings of this study. Then, the concept of abstraction and a theory of limitation of memory capacity are employed to link the research findings to the theories of information systems. The last section presents a list of suggestions for further research.

Summary of Critical Research Findings

An experiment was conducted in a controlled lab setting to collect empirical evidence for testing a set of hypotheses. The result of Analysis of Variance on the data exhibits several findings:

1. The use assistance mode has a significant main effect on the number of syntactic errors, the number of semantic errors, as well as the amount of time required for completing a query specification task. The significance level is 1 percent.
2. The complexity level of problems has a signifcant main effect on all of these three productivity measures at the significance level of 1 percent.
3. The interaction effect of the use assistance mode and the problem complexity on the number of semantic errors and the amount of time for completion is significant at the significance level of 1 percent.
4. This interaction effect on the number of syntactic errors is significant at the 10 percent level. However, it is not significant at the 5 percent level.

Since both query interface modes are implemented in the same very high level language, PROLOG, complex problems and simple problems probably derive similar degree of assistance from the system. However, in the case of the number of semantic errors and the amount of time for completion, complex problems appear to benefit more from the system-provided use assistance. This implies that users' burden for conceptualization when mapping the problem requirements into a query has been significantly relieved by the system. This result supports the notion that users' problem knowledge can be incorporated into a system to make the system interface more intelligent.

Theoretical Explanation of Research Findings

The advantage of incorporating users' knowledge about their problem domains into the system can be explained in terms of the concept of abstraction and a theory of short term memory. Miller (2) suggested that the capacity of human's short term memory in processing information lies within a range of seven plus or minus two. If a system requires too many constructs to be mastered by the user at the same time, productivity tends to suffer. In this study, the use-assisted mode requires less constructs to be remembered by the user. The comlexity of query formulation has been broken into several portions. The user enters a command based on the problem requirement, and then provides answers to system-prompted questions.

On the other hand, the non-use-assisted mode requires the user to specify the query program from scratch. The user must be able to put a number of "pieces" together according to the syntactical rules. More constructs must be kept in the short term memory for this task. In other word, more complexity must be dealted with in the the non-use-assisted mode. This difference of complexity users have to deal with with two different modes accounts for at least part of the main effect of the use assistance mode.

Another perspective to view the results of this study is through the concept of abstraction. Wirth (5, p. 1) stated that "our important mental tool for coping with complexity is abstraction". No widely accepted definition of abstraction has been proposed. Usually, it means a model which ignores irrelevant details. In the database system design, an internal model represents an abstraction at the next higher level than physical organization of the stored database. The conceptual model is a logical view of the database, thus more abstract than the internal level. The external view ignores much of the complexity of the conceptual model, and presents the portion of the datbase relevant to the user. If desired, multiple levels of the external view can be constructed to make a database system easier to use. This concept has been proposed for the design of model-based decision support systems (or model management systems). For example, Dolk and Konsynski (1) derive a concept of model abstraction as a special instance of the frame construct in the artificial intelligence. Model management systems are characterized as frame-systems. Their motivation is to treat a model management system as an analog of a database management system. Wang and Courtney (4) use the concept of abstraction as the conceptual foundation for the development of generalized decision support systems. A generalized decision support system thus developed integrates the capabilities of data management, model management, and knowledge base management.

The use-assisted mode represents an interface at a more abstract level than the non-use-assisted mode: much complexity of the syntactical details has been hidden from the user. The results of this study support the notion that abstraction is a useful mental tool for interfacing with computer systems in general, and with database systems in specific.

Suggestions for Future Research

A number of research issues have been identified in the course of this study. These issues are mostly concerned with extending the scope of this study such that the findings of the study may be validated or modified to be applied to broader domains. Specifically, future research may be directed to investigate issues such as the following:

1. Different problem domains may be used.

   This study uses personnel management in an industry of information systems as the application domain. It is implied that the findings will be applied to any other domains as well. However, before empirical evidence is found, this assumption is still subject to scientific verification. A study addressing this issue has different problem domains as an independent variable. This variable's main effect as well as its interaction with other independent variables may be investigated in the similar fashion as this study.

2. A different data model may be used.
   Although there is research supporting the usage of the relational data model as being more natural to end users, many firms are currently using database management software which support hierarchical or network data model. These data models make explicit the access path to the desired records, and therefore, more constructs must be dealt with by the user. Would the data model be a significant factors in determining users' query productivity? Empirical evidence is needed to answer this question.
3. It is more appropriate to conduct the experiment with real world users.
   As any finding will be applied to the real world practice eventually, using real world users as experiment subjects would generate results with higher external validity. Sufficient attention must be paid to the control of irrelevant factors in this case. Appropriate sample size must be assured to justify the use of analysis techniques.
4. Users' individual difference may be a critical factor.
   As identified by many previous researchers, individual difference of the user often constitutes a significant factor determining productivity measures. Further research may be conducted using users with different

experience in the use of computer, or different psychological patterns in collecting and processing information (cogintive style). If these factors are found to have significant impacts on the productivity level, they should be input into the design process of user-system interface.

5. Other assistance types may be investigated.

   The question-answering process is invoked by the command entered by the user of this study. Other assistance types are feasible in this context. Particularly, a menu-driven may be most beneficial to the domain where maximal prior knowledge of information requirements may be identified. As a menu-driven interface places less memory burden on the user, productivity may be improved as a result of reduced complexity associated with the use of the system. More research must be conducted to accumulate knowledge in this area.

6. Benefit of system-provided assistance must be traded off with the associated cost.

   As revealed by this study, incorporating use assistance in the system interface facilitates easier interaction between the novice user and the system. However, designing and implementing the use assistance requires consumption of various kinds of cost, especially human cost associated with the interaction between the user and the interface designer. A relevant research issue at this conjunction is how much assistance is optimal considering both benefit and cost. Some case studies may have to be conducted before a more comprehensive, analytical approach is taken to attack this problem.

It is very likely that stages of the learning process also play a significant role in determining the relationships investigated by this study. A longitudinal study may help to reveal possible effects of the user's learning curve. This study speculates that the user-assisted mode would lead to higher productivity because of lower load being placed on the user's memory capacity. A rigorous design involving several distinct stages on the learning curve is needed to verify this statement.

In addition to those empirical inquiries suggested above, research effort can also be beneficially directed to making several conceptual models more sophisticated. For example, the classification scheme for relational query specification as presented in this study treats each cell on the table as consisting of a binary value (0 or 1). A useful extension may be made to consider multiple values or even a continuous distribution for some cells.

CHAPTER BIBLIOGRAPHY

1. Dolk, D. R., and B. R. Konsynski, "Knowledge Representation for Model Management," IEEE Transactions on Software Engineering , Vol. SE-10, No. 6, (November 1984), pp. 619 - 628.

2. Miller, G. A., "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," Psychological Review , 63, (1956), pp. 81 - 97.

3. Shneiderman, B., Software Psychology: Human Factors in Computer and Information Systems , Cambridge, Massachusetts, Winthrop Publishers, Inc., 1980.

4. Wang, M. S.- Y., and J. F. Courtney, Jr., "A Conceptual Architecture for Generalized Decision Support System Software," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-14, No. 5, (September/October 1984), pp. 701 - 711.

5. Zimmer, J. A., Abstraction for Programmers, New York, McGraw-Hill Bool Company, 1985.

APPENDIX A

VALIDATION OF HALSTEAD'S VOLUME METRIC (V)

1. Halstead's Volume metric has been used to measure relative complexity of programs written in different languages as well as programs in the same language using different algorithms for the same problem (Mynatt 1984). As the other software metrics proposed by Halstead (1977), this metric is a function of the numbers of operators and operands in the program. The formula is

$$V = (N1 + N2) \operatorname{Log2} (n1 + n2)$$

Where
- N1 = the total number of ocurrences of all operators in a program,
- N2 = the total number of occurrences of all operands in a program,
- n1 = the number of distinct operators in a program,
- n2 = the number of distinct operands in a program.

Log2: Logarithm function based on 2

2. As an implementation language for first-order predicate calaulus, PROLOG has been recommended as an appropriate vehicle for the database research (Gallaire et al. 1984). The fundamental operation of a PROLOG program, when viewed by the programmer, are pattern-matching and backtracking. Therefore, relation names and operations are in the uniform form; they are all predicates. For example, in Micro-Prolog, the relation

```
Project(Project-no Project-name City)
```

is represented as Project(x y z). The operation that prints the values of Project-no is represented by PP(x). After converted into internal representation, the former is (Project x y z), and the latter becomes (PP x); they are all lists. It is this simplicity that this study assumes Halstead's volume metric can be used to measure the complexity of query problems.

3. VALIDATION

In Date's book (1981 pp.141, 142), he has a list of query problems which approximately range from easiest through most complex. Typically, easy problems involve less relations than complex ones. In order to have some means of validating the proposal of using Halstead's volume metric in the context of PROLOG programs, the same relational database as presented in Date's book is assumed. A set of 10 easy problems and a set of 7 complex problems are then developed based on Date's list. Some of the problems in both sets are drawn directly from the list, while some are similar to Date's problems in complexity.

The database, the problems, and some Micro-Prolog programs for solving the problems are described as follows. The number within the parenthesis after each problem is the Volume measure.

DATABASE (in third normal form):

```
Supplier(supplier-no supplier-name status city)
Part(part-no part-name color weight city)
Project(project-no project-name city)
SPJ(supplier-no part-no project-no quantity)
```

SIMPLE PROBLEMS:

1. Get full details of all projects.
2. Get full details of all suppliers.
3. Get full details of all parts.
4. Get supplier numbers for suppliers in London.
5. Get project numbers for projects in London.
6. Get suppliers names for suppliers in London.
7. Get supplier numbers for suppliers who supply Project J1.
8. Get part numbers for parts used by Project J1.
9. Get part numbers for parts supplied by supplier S1 for Project J1.
10. Get supplier names for suppliers who supply some parts to Project J1.

COMPLEX PROBLEMS:

1. Get part numbers for parts supplied to any project in London by a supplier in London.
2. Get project numbers for projects using any part from London supplied by a supplier in London.
3. Get supplier names for suppliers in London who supply red parts to some projects in London.
4. Get all <city part-no city> triples such that a supplier in the first city supplies the specified part to a project in the second city.
5. Get part names for parts supplied to all projects in in London.
6. Get project names for projects in London using red parts supplied by suppliers in London.
7. Get all <supplier-name part-name> pairs such that a supplier in London supplies some parts to a project in Paris.

EXAMPLE MICRO-PROLOG PROGRAMS AND V CALCULATIONS:

Simple Problem #5:

```
Which(x Supplier(x y z London) & PP(x))
```

N1: 6   N2: 6   n1: 4   n2: 4

V = 12 Log2 8

Complex Problem #7:

```
Which((x y) Supplier(x1 x2 x3 London) &
            SPJ(x1 y1 y2 Z5) &
            Project(y2 y Paris) &
```

```
                    PP(x y))

        N1: 11  N2:18  n1:4  n2:13

        V = 29 Log2 17

Operators: Which, (), &, operation predicate
Operands: relation predicate, variable, constant
```

Volume values are obtained for all the programs using the above convention. Two-sample T test is then used to test the hypothesis that complex measures for both sets of problems are actually equal in the statistical sense. An output from MINITAB is shown as follows.

```
-------------------------------------------------------
Simple Problems  N: 10  Mean = 51.086  St. Dev. = 15.9
Complex Problems N:  7  Mean = 129.37  St. Dev. = 24.8

    Approx. degree of freedom: 9

A 95.00 percent C. I. for mu1 - mu2 is
         (-102.3751,  -54.1967)

Test of mu1 = mu2 vs. mu1 N. E. mu2
T = -7.354

The test is significant at 0.0000
-------------------------------------------------------
```

The result of statistical testing shows that Halstead's V metric is able to distinguish two sets of database query problems.

4. SENSITIVITY ANALYSIS:

In order to assure the convention adopted to count the operators and the operands does not constitute a major factor in the calculation of V measures, two other different conventions which are likely to be adopted by other researchers are used and tested. First, relation predicates are treated as operators instead of operands. Then, operation predicates are treated as operands instead of operators. The same result has been obtained in all cases. This phenomenon rules out the possibility of getting significantly different results due to inconsistent subjective judgments regarding the classification of operators and operands.

APPENDIX B

Subject Information Sheet

(Informations provided herein will be used by this research only. No disclosure will be made for any other purpose.)

1. Name: ______________________________

2. Sex (M: Male, F: Female): ____ 3. Age: ____

4. Major: ________________________

5. I have completed the following computer courses in the past five years:

   ____

   ____

6. I can use the following software(s) to complete my work without much assistance from somebody else:

   ____

   ____

7. I am currently taking BCIS 361 Section- _______.

APPENDIX C

Instruction for Non-User-Assisted Mode

I. <u>ABOUT THE DATABASES</u>

This experiment involves two personnel databases: the AAplicant Database and the Employee Database. The Applicant Database contains various information about the individuals applying for data processing positions: SS#, Name, Address, Phone-no, Terminating Degree, Degree Major Area, Skill(s), Number of years associated with each skill, Position Wanted, and Expected Annual Pay. The Employee Database contains informations about current employees of a company in the industry of computer information system design. Data items are Employee-ID, Name, Address, Phone-no, Date of Entry, etc.

All the data items are divided into a number of groups. Each group is given a name. Another term for such a data group is <u>relation</u> . For example, four data items - SS#, Name, Address, and Phone-no - are parts of a group named Applicant. The format of relation Applicant is:

Applicant (<u>SS#</u> Name Address Phone-no)

It indicates that the first data item within the relation Applicant is an applicant's SS#, the second one is Name, and so on. <u>SS#</u> is underlined, meaning it can be used to uniquely identify the rest of the data items in the same relation, i.e. Name, Address, Phone-no.

The names of all the relations together with the data items contained in each relation are listed below. You will need to keep refering back to this list at the later part of the experiment.

A. <u>Applicant</u> <u>Database</u>:

1. Applicant(<u>SS#</u> Name Address Phone-no)
2. Education-applicant(<u>SS#</u> Degree Major)
3. Experience-applicant(<u>SS#</u> <u>Skill</u> Number-of-year)
4. Looking-for(<u>SS#</u> Position Expected-annual-pay)

B. <u>Employee</u> <u>Database</u>:

5. Employee(<u>Employee-ID</u> Name Address Phone-no)
6. Education(<u>Employee-ID</u> Degree Major)
7. Job-title(<u>Employee-ID</u> Department Title Date-in)
8. Experience(<u>Employee-ID</u> <u>Skill</u> Number-of-year)
9. Communication-skill(<u>Employee-ID</u> Grade)
10. Project-status(<u>Employee-ID</u> Current-project Date-of-completion)

<u>Explanation</u>:

1. Date-in is the date on which the employee entered the company. Its format in the database is (mm dd yy).
2. Date-of-completion is the date on which the project the employee is currently working on will be complete. The format is also (mm dd yy).
3. Grade in the relation Communication-skill ranges from 1 to 5. 5 means excellent. 3 means fair. 1 means poor.
4. Education: 1 - bachelor, 2 - master, 3 - PhD.

II. HOW TO INSTRUCT THE COMPUTER TO FIND WANTED INFORMATION?

Instructing the computer to find wanted information, in this experiment, is a straightforward task. It invloves some simple rules and the specification of WHAT is wanted using these rules. This section uses examples to explain how a query specification program is constructed. Go through these examples. Keep the pages for database definitions at hand because you may need to refer to them from time to time.

Example Problem 1:

An information center consultant is to be recruited. Appropriate applicants must have at least one year of experience in IBM-PC. Print the names and the phone numbers of the qualified applicants.

Example Command 1:

```
&Find((x1 x2)  Experience-applicant(x IBM-PC x3) and
      x3 at-least 1 and Applicant(x x1 x4 x2))    <return>
```

Explanation:

1. This problem concerns recruiting qualified applicants for a position opening. Therefore, Applicant Database is to be used.
2. The data items pertaining to the problem are 1). Skill, 2). Number of years for the skill, 3). Name, and 4). Phone number. These data items can be found in relations

Experience-applicant and Applicant.

3. In the above program,

```
Experience-applicant(x IBM-PC x3)
```

instructs the computer to search through all occurrences (Data Items) of the relation.

```
Experience-applicant(SS#  Skill  Number-of-year)
```

and to select the ones with IBM-PC as the value of Skill. That is, all the applicants having experience of IBM-PC are selected. Their SS#'s (represented by x) and Number-of-year's (represented by x3) are identified and passed on to the next element of the program.

4. The second element,

```
x3 at-least 1
```

will cause only those applicants with at least ONE year of experience being kept.

5. The third and last element in the above program,

```
Applicant(x x1 x4 x2)
```

will accept the values of SS# values of remaining applicants, i.e. those with at least one year of experience in IBM-PC, and uses x1 to represent the values of Name, x4 for values of Address, and x2 for values of Phone-no.

6. The &. in the beginning of the sample is a prompt symbol of the system. So the first entry of the program is

```
Find((x1 x2)  .............)
```

which indicates that the values of x1 and x2 are to be printed as a result of executing the above three elements. The rule requires that Find begins with capital F and that a pair of parentheses is needed for the entire part of the program after Find. A pair of parentheses is also needed to encompass x1 and x2.

7. The x, x1, x2, x3, and x4 in the program are variables. A variable representing a data item should be unique. Different data items must use different variables.
8. The IBM-PC in this program is called a Constant. A constant should be specified exactly as indicated in the problem descripiton.
9. There is at least one space between (x1 x2) and the first element following it, Experience-applicant in this case. At least one space is required before and after "and". Data items are also separated by at least one space.
10. When entering the program into the computer, keep typing until the entire program is complete before the ‹return› key is pressed. DON'T PRESS ‹return› KEY BEFORE THE PROGRAM IS FINISHED. If indentation is desired, use space bar for this purpose. Use ‹back space› key to move the cursor back to the beginning of the mistake and type in the correct line when necessary. DON'T USE ARROW KEY(S) FOR MOVING AROUND THE CURSOR.
    Pressing ‹return› key will cause the program to be executed.

<u>Example</u> <u>Problem</u> <u>2</u>:

A new division is looking for a well-trained applicant to fill a Data Administrator position for its database group. Qualified applicants must have at least a master degree in "Information Systems", at least two years of experience in "data modelling", and at least two years of experience in IMS-DB-DC. Obtain a listing of qualified applicants' names and phone numbers.

Example Command 2:

```
&Find((x1 x2)  Education-applicant(x x3 "Information Systems")
      and X3 at-least 2 and Experience-applicant(x IMS-DB-DC x4)
      and x4 at-least 2 and Experience-applicant(x
       "data modelling" x5)
      and x5 at-least 2 and Applicant(x x1 x6 x2))    <return>
```

Explanation:

1. The Applicant Database is to be used, since this is a recruiting problem.
2. Pertinent data items (Degree, Major, Skill, Number-of-year, Name, Phone-no) can be found in relations Education-applicant, Experience-applicant, and Applicant. Therefore, these are the relations to be referenced in the program.
3. The program begins with a Find and a set of data items to be printed (x1 and x2). The last element of the program

   Applicant(x x1 x6 x2)

shows the x1 represents Name and x2 represents Phone-no.

4. Education-applicant(x x3 "Information Systems") will search for all applicants with "Information Systems" as the major area. x is used to represent SS#'s and x3 used to represent Degree of these selected applicants. Note that "Information systems" is quoted. When a multiple-word constant is specified, it has to be quoted. A hyphened word is considered a single word which does not needed to be quoted.
5. x3 at-least 2 will screen away all the applicants except those with at least a master degree.
6. Experience-applicant(x "data modelling" x4) will screen away the applicants without experience of "data modelling" from the remaining applicants passed from the preceeding relation.
7. x4 at-least 2 specifies that the qualified applicants's "data modelling" experience must be at least two years.
8. When the next two elements

   Experience-applicant(x IMS-DB-DC x5) and x5 at-least 2

   are evaluated by the computer, only the applicants who have at least a master degree in "Information Systems', at least two years of experience in "data modelling", and at least two years of experience in IMS-DB-DC are selected. These remaining applicants' SS#'s (represented by x throughout the program) are then passed on to the last element.

9. As Example Program 1, the last element,

```
Applicant(x x1 x6 x2)
```

   instructs the computer to search for the applicants whose SS# match the SS#'s passed from the preceeding relations. It also uses x1, x6, and x2 to represent Name, Address, and Phone-no, respectively.

10. When the entire program has been typed in, press <return> key to have the program executed.

As a summary of the above two examples, a query program begins with a key word

Find

followed by a pair of parenthese. Within these parentheses are two components: answer pattern and relation series. The answer pattern specifies the data items which values are to be displayed, for example, Names and Phone-no's. The relation series specify the relation(s) containing the data items involved in the problem as well as the condition(s) to be satisfied. When the relations and condition expressions are connected by the key word

and

there must be at least one space before and after the "and". When the entire program has been entered into the computer, pressing <return> key will cause the program being evaluated. The output, if it exists, will be printed. The system prompt symbol &. will appear when the execution of the program is through.

Occasionally, when the <return> key is pressed, instead of evaluating your program, the computer displays a dot and a number. That simply means exactly that many more right parenthesis needs to be entered to make a complete program. For example, if the system displays a .2 after you pressed <return> key, your response would be to enter two parentheses, i.e.)), and then press <return> key. JUST REMEMBER THAT THE NUMBER OF RIGHT PARENTHESES MUST BE IDENTICAL TO THE NUMBER OF LEFT PARENTHESES.

You must choose a unique variable for each desired data item and each data item of a particular relation which is not relevant to the problem. You can choose anything from x, x1 to x50, and y, y1 to y50. For example, to refer to relation Applicant, you can specify

Applicant(x x1 x2 x3)

where x represents SS#, x1 represents Name, x2 represents Address, and x3 represents Ph#. In other words, the positions of variables determine which data items these variables stand for. For another example, to refer to the relation Education-applicant with "information Systems" as the value of data item Major, you can specify

Education-applicant(x x5 "Information Systems")

Where x and x5 represent SS# and Degree, respectively. Example Problems 3 ,4 and 5 are examples of problems concerning project team assignment. Read through these three problems and the associated query programs. The rules are the same as the previous two examples. Then you will have a better idea on how to construct query programs for the ten problems to be handed to you.

<u>Example Problem 3</u>:

A programmer/analyst trainee is needed for a project development team. This team member will be selected from the existing employee. Qualification includes at least a bachelor degree, and entering the company no later than June 1, 84.

<u>Example Command 3</u>:

```
&Find((x1 x2) Education(x x3 x4) and x3 at-least 1 and
       Job-title(x x5 x6 y) and y no-later-than (06 01 84)
       and Employee(x x1 x7 x2))                  <return>
```

Explanation:

In addition to the rules explained in the previous two examples, some more comments are made below,

1. This is a project team assignment problem, hence the Employee Database is used.
2. Some data items are included in the relevant relations but are not involved in any condition. For example, the problem does not mention what major area is required. This irrelevant data item is also represented by a variable. That's it. But no condition expression will ever include this irrelevant data item.

3. The condition expression

   y no-later-than (06 01 84)

   specifies that all qualified personnel must enter the company no later than the date indicated. Throughout this experiment, whenever a date is concerned, always use this format: (mm dd yy).

<u>Example Problem 4</u>:

Two systems analysts are to be selected from the existing staff for a project team. These two staffs must have at least two years of experience in ADR-DATACOM-DB, at least two years of experience in "manufacturing applications", and will be available by August 1, 85. Obtain a listing of all qualified employees.

<u>Example Command 4</u>:

```
&Find((x1 x2)  Experience(x ADR-DATACOM-DB x3) and
              x3 at-least 2 and
              Experience(x "manufacturing applications" x4)
              and x4 at-least 2 and
              Project-status(x x5 x6) and
              x6 no-later-than (08 01 85) and
              Employee(x x1 y x2))                <return>
```

All rules associated with example are the same as what have been explained in the previous examples.

<u>Example Problem 5</u>:

A "Programmer" trainee is to be recruited from the applicants who have at least a bachelor degree and expected annual pay no higher than 30000. Get a list of names and phone numbers.

<u>Example Command 5</u>:

```
&Find((x1 x2)  Education-applicant(x y1 x3) and
              y1 at-least 1 and
              Looking-for(x y2 x4) and
              x4 no-higher-than 30000 and
              Applicant(x x1 y x2))                <return>
```

<u>Explanation</u>:

x4 stands for Expected-annual-pay in this program. The condition

```
        x4 no-higher-than 30000
```

specifies that the applicant's expected annual pay should not exceed 30000. NO COMMA MAY BE INSERTED INTO A NUMERIC VALUE.

* TO INDENT YOUR PROGRAM LIKE THIS EXAMPLE, PRESSING SPACE BAR UNTIL THE CURSOR GETS TO THE APPROPRIATE POSITION.

APPENDIX D

Instruction for User-Assisted Mode

I. <u>ABOUT THE DATABASES</u>

This experiment involves two personnel databases: the Applicant Database and the Employee Database. The Applicant Database contains various information about the individuals applying for data processing positions: SS#, Name, Address, Phone-no, Terminating Degree, Degree Major Area, Skill(s), Number of years associated with each skill, Position Wanted, and Expected Annual Pay. The Employee Database contains informations about current employees of a company in the industry of computer information system design. Data items are Employee-ID, Name, Address, Phone-no, Date of Entry, etc.

All the data items are divided into a number of groups. Each group is given a name. Another term for such a data group is <u>relation</u> . For example, four data items - SS#, Name, Address, and Phone-no - are parts of a group named Applicant. The format of relation Applicant is:

Applicant (<u>SS#</u> Name Address Phone-no)

It indicates that the first data item within the relation Applicant is an applicant's SS#, the second one is Name, and so on. <u>SS#</u> is underlined, meaning it can be used to uniquely identify the rest of the data items in the same relation, i.e. Name, Address, Phone-no.

The names of all the relations together with the data items contained in each relation are listed below. Go through these definitions before proceeding further.

A. Applicant Database:

1. Applicant(SS# Name Address Phone-no)
2. Education-applicant(SS# Degree Major)
3. Experience-applicant(SS# Skill Number-of-year)
4. Looking-for(SS# Position Expected-annual-pay)

B. Employee Database:

5. Employee(Employee-ID Name Address Phone-no)
6. Education(Employee-ID Degree Major)
7. Job-title(Employee-ID Department Title Date-in)
8. Experience(Employee-ID Skill Number-of-year)
9. Communication-skill(Employee-ID Grade)
10. Project-status(Employee-ID Current-project Date-of-completion)

Explanation:

1. Date-in is the date on which the employee entered the company. Its format in the database is (mm dd yy).
2. Date-of-completion is the date on which the project the employee is currently working on will be complete. The format is also (mm dd yy).
3. Grade in the relation Communication-skill ranges from 1 to 5. 5 means excellent. 3 means fair. 1 means poor.
4. Education: 1 - bachelor, 2 - master, 3 - PhD.

HOW TO RETRIEVE INFORMATION FROM DATABASES

1. General Description:

Retrieving information from the databases involves a question-answering process. A collection of conversational programs have been designed and implemented in the computer. After realizing what information is to requested, you then select and enter an appropriate command to initiate this question-answering process. Each of these commands is named to reflect some specific characteristics of the problems to be resolved. YOU ARE TO JUDGE WHICH COMMAND SHOULD BE SELECTED BASED ON PROBLEM CHARACTERISTICS.

Once a command has been entered, you are advised to read the displayed message carefully, and follow the instruction for data entry before typing in the response. Mistakes can be corrected ONLY before the <return> key is pressed. Once the <return> is pressed, any user input will be executed as such.

2. Specific Instruction:

There are two kinds of problems in this experiment, six for each. The first kind of problems concerns retrieving the names and the phone numbers of job applicants who are qualified for a job opening. The second kind of problems concerns retrieving the names and the phone numbers of existing employees who satisfy the requirements for a project development team appointment. These twelve problems have been mixed in order. However, each problem was described in a specific manner so that you will be able to tell what kind of problem each one is as you read through the problem description.

For the recruiting problems, you are to select and enter a command of the following format:

```
Find((x1 x2)  Recruiting-n(x1 x2))
```

where n must be a positive interger indicating the number of experience(s) involved. For example, if a job requires no prior experience of any kind, then the command to be entered will be

```
Find((x1 x2)  Recruiting-0(x1 x2))
```

On the other hand, if the job requires three different experiences, then you will enter

```
Find((x1 x2)  Recruiting-3(x1 x2))
```

For the project assignment problems, the command has the following format:

```
Find((x1 x2) Assignment-n(x1 x2))
```

where n also indicates the number of experience(s) involved.

A &. symbol on the screen means the computer is ready for you to enter a command. When entering a command, key words such as Find, Recruiting, and Assignment must begin with the capital letter. Upper case and lower case letters are treated as different characters. Within the command, AT LEAST ONE SPACE IS NEEDED BETWEEN X1 AND X2. THERE ALSO HAS TO BE AT LEAST ONE SPACE SEPARATING THE FIRST (X1 X2) AND EITHER Recruiting OR Assignment. No other space is needed elsewhere. This format requirement is illustrated below,

```
&.Find((x1 x2) Recruiting-1(x1 x2))
          |   |                |
           space             space
```

(One experience is assumed.)

Several examples are provided in the rest of this instruction. Be sure that you read through these examples. They will help you solve the problems to be handed to you. For each of the following problem, system's prompt is indicated by a header (SYSTEM:), while the input you are supposed to enter is indicated by the header (USER:). EVERY USER INPUT MUST BE BE FOLLOWED BY PRESSING THE <return> KEY.

Example Problem 1:

An information center consultant is to be recruited. Appropriate applicants must have at least one year of experience in IBM-PC. Print the names and the phone numbers of the qualified applicants.

Interaction Session for Example Problem 1:

```
&Find((x1 x2) Recruiting-1(x1 x2))          <return>
```

(This command is selected because this problem concerns recruiting activity and one experience is involved.)

```
SYSTEM: Enter the minimum degree required as follows,
            1 for bachelor,
            2 for master,
            3 for PhD, or
            0 if the degree requirement is nCt mentioned
              in the problem:
USER:0        <return>


SYSTEM: What's the major area?
        Or simply enter an X if no specific area of major
        is specified:
USER: X         <return>


SYSTEM: What's the maximum annual pay that should not
        by exceeded? Enter a large number such as 1000000
        if not mentioned. No special character such as
        $ or , may be included in the amount:
USER: 1000000       <return>


SYSTEM: Enter the experience required:
USER: IBM-PC    <return>

SYSTEM: Enter the minimal number of year(s):
USER: 1     <return>

      Output is displayed and session ends.
```

Example Problem 2:

A new division is recruiting a well-trained applicant to fill a Data Administrator position for its database group. Qualified applicants must have at least a master degree in "Information Systems", at least two years of experience in "data modelling", and at least two years of experience in IMS-DB-DC. Obtain a listing of qualified applicants names and phone numbers.

<u>Interaction Session for Example Problem 2</u>:

```
&Find((x1 x2)   Recruiting-2(x1 x2))        <return>

(This command is selected because this is a recruiting
 problem and two experiences are required in this problem.)


SYSTEM: Enter minimum degree required as follows,
             1 for bachelor,
             2 for master,
             3 for PhD, or
             0 if no degree requirement is specified:
USER: 2        <return>


SYSTEM: Enter required major area, or simply enter X if no
        specific area is required. Multiple words must be
        put within a pair of quotes:
USER: "Information Systems"      <return>


SYSTEM: Enter maximal amount of annual pay that should not
        be exceeded. Enter a large number such as 1000000
        if no maximal level of allowed annual pay is
        specified. No special character such as $ or ,
        may be included in the amount:
USER: 1000000        <return>


SYSTEM: Enter the first experience required:
```

```
USER: IMS-DB-DC          <return>


SYSTEM: Enter minimal number of years:
USER: 2             <return>



SYSTEM: Enter the second experience required:
USER: "data modelling"

SYSTEM: Enter minimal number of years:
USER: 2         <return>

      Output is displayed and session ends.
```

Example Problem 3:

A programmer/analyst trainee is wanted by a project development team. This team member will be selected from the existing employee. Qualification includes at least a bachelor degree, and entering the company no later than June 1, 84.

<u>Interaction Session for Example Problem 3</u>:

```
&Find((x1 x2)     Assignment-0(x1 x2))       <return>
```

(This command is selected because this problem concerns project team assignment and no experience requirement is specified in the problem.)

```
SYSTEM: Enter minimum degree required as follows,
            1 for bachelor,
            2 for master,
            3 for PhD, or
            0 if no degree requirement is specified:
USER: 1       <return>


SYSTEM: Enter required major area, or simply enter X if
        no specific area is required. Multiple words must
        be put within a pair of quotes:
USER: X      <return>


SYSTEM: Enter the current position title of the qualified
        candidates. Enter an X if the required current
        title is not mentioned:
USER: X      <return>


SYSTEM: Enter the latest date before which qualified
        candidates must enter the company in order to
        be considered for the new assignment. The format
        of your input must be (mm dd yy). If no deadline
        of entry is mentioned, enter (99 99 99) as your
        response:
USER: (06 01 84)     <return>
```

```
SYSTEM: Enter the required minimum grade for communication
        skill. Enter 0 if the communication skill grade
        is not mentioned:
USER: 0       <return>


SYSTEM: Enter the deadline date the personnel must become
        available in order to be considered for the new
        assignment. If the requiremet is not mentioned,
        enter (99 99 99). The format of your input should
        be (mm dd yy):
USER: (99 99 99)     <return>
```

Output is displayed and session ends.

Example Problem 4:

Two systems analysts are to be selected from the existing staff for a project team. These two people must have at least two years of experience in ADR-DATACOM-DB, at least two years of experience in "manufacturing applications", and will be available by August 1, 85. Obtain a listing of all qualified employees.

Interactive Session for Example Problem 4:

```
&Find((x1 x2)   Assignment-2(x1 x2))      <return>

(This command is selected because this is a project team
assignment problem and two experiences are required.)

SYSTEM: Enter minimum degree required as follows,
             1 for bachelor,
             2 for master,
             3 for PhD, or
             0 if no degree requirement is specified:
USER: 0        <return>


SYSTEM: Enter required major area, or simply enter X if no
        specific major area is required. Multiple word must
        be put within a pair of quotes:
USER: X           <return>


SYSTEM: Enter the current position title of qualified
        candidates. Or enter X if the required current
        position title is not mentioned.
USER: X           <return>


SYSTEM: Enter the latest date before which personnel
        must have entered the company ir order to be
        considered for the new assignment. The format of
        your input is (mm dd yy).
        If no deadline of entry is mentioned, enter
        (99 99 99) as your response:
USER: (99 99 99)           <return>
```

```
SYSTEM: Enter the first experience required:
USER: ADR-DATACOM-DB       <return>


SYSTEM: Enter minimal number of years:
USER: 2          <return>


SYSTEM: Enter the second experience required:
USER: "manufacturing applications"


SYSTEM: Enter minimal number of years:
USER: 2       <return>


SYSTEM: Enter the required minimum grade for
        communication skill. Enter 0 if the requirement
        for communication skill grade is not mentioned:
USER: 0       <return>


SYSTEM: Enter the deadline date the personnel must become
        available in order to be considered for the new
        assignment. If the requirement is not mentioned,
        enter (99 99 99).
        The format of your input is (mm dd yy):
USER: (08 01 85)       <return>
```

Output is displayed and session ends.

Example Problem 5:

A programmer trained is to be recruited from the applicants who have at least a bachelor degree and expected annual pay no higher than 30000. Get a list of names and phone numbers.

Interaction Session for Example Problem 5:

```
&Find((x1 x2)   Recruiting-0(x1 x2))

(This command is selected because this is a recruiting
problem and no experience requirement is mentioned.)

SYSTEM: Enter minimum degree required as follows,
            1 for bachelor,
            2 for master,
            3 for PhD,  or
            0 if no degree requirement is mentioned:
USER: 1         <return>


SYSTEM: Enter requirement major area, or simply enter X
        if no specific major area is required.
        Multiple words must be put within a pair of quotes:
USER: X         <return>


SYSTEM: Enter maximal amount of annual pay that should
        not be exceeded. Enter a large number such as
        1000000 if the allowed maximum pay is not mentioned
        No special character such as $ or , may be included
        in the amount:
USER: 30000         <return>


    Output is displayed and session ends.
```

APPENDIX E

Query Problems Used in the Experiment

Problem #1:

An entry level programmer is to be recruited. The only qualification is at least a bachelor degree of any major area. Obtain a list of the names and the phone numbers of qualified applicants.

Query Program:

```
    Find((x1 x2)   Education-applicant(x x3 x4)
                   and x3 at-least 1
                   and Applicant(x x1 x5 x2))
```

Calculation of Volume Measure:

Operator	Gross Occurrence		Unique Count
Find	1		1
()	4		1
and	2		1
Operation Predicate	1		1
Total (N1)	8	(n1)	4
Operand			
Data Relation	2		2
Variable	10		6
Constant	1		1
Total (N2)	13	(n2)	9

$$V = (N1 + N2) \text{ Log2 } (n1 + n2)$$

$$= (8 + 13) \text{ Log2 } (4 + 9)$$

$$= 77.709$$

Problem #2:

A project development team is to be formed. The project leader will be an existing employee with the following qualifications:

1. Current job title is "Project Leader",
2. At least two years of experience in the design of "payroll processing" (skill name: "payroll processing").
3. At least three years of experience in IMS-DB-DC (skill name: IMS-DB-DC).

Obtain a listing of the qualified employee's names and phone numbers.

Query Program:

```
     Find((x1 x2)   Job-title(x x3 "Project Leader" x4)
                and Experience(x "payroll processing" x5)
                and x5 at-least 2
                and Experience(x IMS-DB-DC x6)
                and x6 at-least 3
                and Employee(x x1 x7 x2))
```

Calculation of Volume Measure (Problem #2):

Operator	Gross Occurrence		Unique Count
Find	1		1
()	6		1
and	5		1
Operation Predicate	2		1
Total (N1)	14	(n1)	4
Operand			
Data Relation	4		3
Variable	15		8
Constant	5		5
Total (N2)	24	(n2)	16

$$V = (N1 + N2) \text{ Log2 } (n1 + n2)$$

$$= (14 + 24) \text{ Log2 } (4 + 6)$$

$$= 164.233$$

Problem #3:

A junior programmer is wanted for a project development team. The qualified employee must have at least one year of COBOL experience. Find the names and the phone numbers of qualified employees. (Skill name is COBOL.)

Query Program:

```
Find((x1 x2)   Experience(x COBOL x3) and
               x3 at-least 1 and
               Employee(x x1 x4 x2))
```

Calculation of Volume Measure:

Operator	Gross Occurrence		Unique Count
Find	1		1
()	4		1
and	2		1
Operation Predicate	1		1
Total (N1)	8	(n1)	4
Operand			
Data Relation	2		2
Variable	10		6
Constant	1		1
Total (N2)	13	(n2)	9

$$V = (N1 + N2) \text{ Log2 } (n1 + n2)$$

$$= (8 + 13) \text{ Log2 } (4 + 9)$$

$$= 77.709$$

Problem #4:

A senior systems analyst is to be recruited. Job requirements include:

1. At least a bachelor degree in "Information Systems",
2. At least three years of experience in "systems analysis" (skill name: "systems analysis").
3. Expected annual pay no higher than 50000.

Get a listing of qualified applicants' names and phone numbers.

Query Program:

```
     Find((x1 x2)
          Education-applicant(x x3 "Information Systems")
          and x3 at-least 1   and
          Experience-applicant(x "systems analysis" x4)
          and x4 at-least 3   and
          Looking-for(x x5 x6)  and
          x6 no-higher-than 50000  and
          Applicant(x x1 x7 x2))
```

Calculation of Volume Measure (Problem #4):

Operator	Gross Occurrence		Unique Count
Find	1		1
()	6		1
and	6		1
Operation Predicate	3		2
Total (N1)	16	(n1)	5
Operand			
Data Relation	4		4
Variable	16		8
Constant	5		5
Total (N2)	25	(n2)	17

$$V = (N1 + N2) \text{ Log2 } (n1 + n2)$$

$$= (16 + 25) \text{ Log2 } (5 + 17)$$

$$= 182.837$$

Problem #5:

A part time contract programmer is to be recruited. Applicants must have at least three years of COBOL experience. Get a listing of qualified applicants' names and phone numbers. (Skill name is COBOL.)

Query Program:

```
    Find((x1 x2)   Experience-applicant(x COBOL x3)
                   and x3 at-least 3
                   and Applicant(x x1 x4 x2))
```

Calculation of Volume Measure:

Operator	Gross Occurrence		Unique Count
Find	1		1
()	4		1
and	2		1
Operation Predicate	1		1
Total (N1)	8	(n1)	4
Operand			
Data Relation	2		2
Variable	9		5
Constant	2		2
Total (N2)	13	(n2)	9

$$V = (N1 + N2) \text{ Log2 } (n1 + n2)$$

$$= (8 + 13) \text{ Log2 } (4 + 9)$$

$$= 77.709$$

Problem #6:

A programmer/analyst is needed by a project developement team. This person must be currently a programmer/analyst (position title: "Programmer/Analyst"). The experience requirement includes: at least three years of COBOL (skill name: COBOL), at least three years of manufacturing applications design (skill name: "manufactuirng applications"), and at least three years of MVS-TSO (skill name: MVS-TSO). Print the names and the phone numbers of employees who are qualified for this position.

Query Program:

```
     Find((x1 x2)   Job-title(x x3 "Programmer/Analyst x4)
                and Experience(x COBOL x5)
                and x5 at-least 3
                and Experience(x "manufacturing
                    applications" x6)
                and x6 at-least 3
                and Experience(x MVS-TSO x7)
                and x7 at-least 3 and
                    Employee(x x1 x8 x2))
```

Calculation of Volume Measure (Problem #6):

Operator	Gross Occurrence		Unique Count
Find	1		1
()	7		1
and	7		1
Operation Predicate	3		1
Total (N1)	18	(n1)	4
Operand			
Data Relation	5		3
Variable	18		9
Constant	7		5
Total (N2)	30	(n2)	17

$$V = (N1 + N2) \text{ Log2 } (n1 + n2)$$

$$= (18 + 30) \text{ Log2 } (4 + 17)$$

$$= 210.831$$

Problem #7:

A senior programmer/analyst is to be recruited. Job requirements include:

1. At least a bachelor degree in "Information Systems",
2. At least three years of experience in COBOL (skill name: COBOL).
3. At least two years of experience in MVS-TSO (skill name: MVS-TSO).

Find the names and phone numbers of all qualified applicants.

Query Program:

```
     Find((x1 x2)
          Education-applicant(x x3 "Information Systems")
          and x3 at-least 1   and
          Experience-applicant(x COBOL x4)
          and x4 at-least 3   and
          Experience-applicant(x MVS-TSO x5)
          and x5 at-least 2   and
          Applicant(x x1 x6 x2))
```

Calculation of Volume Measure (Problem #7):

Operator	Gross Occurrence		Unique Count
Find	1		1
()	6		1
and	6		1
Operation Predicate	3		1
Total (N1)	16	(n1)	4
Operand			
Data Relation	4		3
Variable	15		7
Constant	6		6
Total (N2)	25	(n2)	16

$$V = (N1 + N2) \text{ Log2 } (n1 + n2)$$

$$= (16 + 25) \text{ Log2 } (4 + 16)$$

$$= 177.199$$

Problem #8:

A programmer trainee is wanted by a newly-established project development team. The only requirement is at least a bachelor degree in "Information Systems". Get a listing of qualified employees' names and phone numbers.

Query Program:

```
Find((x1 x2)   Education(x x3 "Information Systems")
               and x3 at-least 1
               and Employee(x x1 x4 x2))
```

Calculation of Volume Measure:

Operator	Gross Occurrence		Unique Count
Find	1		1
()	4		1
and	2		1
Operation Predicate	1		1
Total (N1)	8	(n1)	4
Operand			
Data Relation	2		2
Variable	9		5
Constant	2		2
Total (N2)	13	(n2)	9

$$V = (N1 + N2) \text{ Log2 } (n1 + n2)$$

$$= (8 + 13) \text{ Log2 } (4 + 9)$$

$$= 77.709$$

APPENDIX F

Program Listing of Interactive Query Patterns

I. Operation Predicates:

/* With the primitive features of Micro-Prolog as building blocks, this study defines some higher level predicates which make the system easier to use. These operation predicates are available for both user-assisted and non-user-assisted modes. Furthermore, these operation predicates are named in such a way that the user knows the meaning of the predicate simply by its name. */

```
X at-least Y if X EQ Y
X at-least Y if Y LESS X
X no-higher-than Y if X EQ Y
X no-higher-than Y if X LESS Y
X Find Y if X which Y
(X Y Z) after (x y z) if z LESS Z
(X Y Z) after (x y Z) if x LESS X
(X Y Z) after (X x Z) if x LESS Y
(X Y Z) no-later-than (x y z) if (x y z) after (X Y Z)
(X Y Z) no-later-than (x y z) if
                              X EQ x and
                              Y EQ y and
                              Z EQ z
```

II. RECRUITING PROBLEMS:

/* The following relations are used to generate prompts for attribute values and to capture those values entered by the user. The user inputs will be passed along to formulate a complete query. */

```
((Experience-input-3 X Y Z)
 (space 10)
 (P Enter the third experience required:)
 (R Y)
 (PP)
 (PP)
 (space 10)
 (P Enter minimal number of years:)
 (R Z)
 (blank-line 5)


((Experience-input-2 X Y Z)
 (space 10)
 (P Enter the second experience required:)
 (R Y)
 (PP) (PP)
 (space 10)
 (P Enter minimal number of years:)
 (R Z)
 (blank-line 5))


((Experience-input-1 X Y Z)
 (blank-line 5)
 (space 10)
 (P Enter the experience required:)
 (R Y)
 (PP) (PP)
 (space 10)
 (P Enter the minimal number of years:)
 (R Z)
 (blank-line 5))


((Looking-for-input X Y Z)
 (space 10)
 (PP Enter maximal amount of annual pay that should
     not be exceeded.)
 (space 10)
 (PP Enter any number larger than 1000000 if no maximal
     level of allowed annual pay is specified.)
 (space 10)
 (PP No special characters such as $ or , may be present
  in the amount:)
 (R Z)
 (blank-line 5))


((Education-input X Y Z)
 (blank-line 20)
 (space 10)
 (PP Enter minimum degree required as follows:
```

```
 (space 15)
 (PP 1 for bachelor)
 (space 15)
 (PP 2 for master)
 (space 15)
 (PP 3 for PhD, or)
 (space 15)
 (P 0 if no degree requirement is specified:)
 (R Y)
 (PP) (PP) (space 10)
 (PP Enter required major area, or simply enter an x if
     no specific area is required. Multiple words must
     be put within a pair of quotes:)
 (R Z))
```

/* This relation provides an error message when the user inadvertently misses the number specifying the number of experience required in the command. A help message is also provided to remind the user of the correct command. */

```
((Recruiting X Y)
 (P "                ")
 (PP Recruinting (X Y) is NOT defined.)
 (P "                ")
 (PP Enter:)
 (P "                ")
 (PP Recruiting-n (X Y))
 (P "                ")
 (PP where n is 0, 1, 2, 3, ...))
```

/* Recruitng-0, Recruiting-1, Recruiting-1, Recruiting-2, and Recruiting-3 define the main bodies of the interactive query patterns. To illustrate the various ways of the query pattern definition, Recruiting-0 and Recruiting-1 incorporate all of the fundamental relations, while Recruiting-2 and Recruiting-3 use predefined higher level relations as building blocks for prompts generation and user inputs storage. */

```
((Recruiting-0 X Y)
 (blank-line 20)
 (space 10)
 (PP Enter minimum degree required as follows,)
 (space 15)
 (PP 1 for bachelor,)
 (space 15)
 (PP 2 for master,)
 (space 15)
 (PP 3 for PhD, or)
 (space 15)
 (PP 0 if the degree requirement is not mentioned:)
 (R Z)
 (PP) (space 10)
 (PP Enter required major area, or simply enter an x if)
 (space 10)
 (PP no specific major area is required.)
 (space 10)
 (P Multile words must be put within a pair of quotes:)
 (R y)
 (blank-line 10) (space 10)
 (PP Enter maximal amount of annual pay that should not
     be exceeded.)
 (space 10)
 (PP Enter any number larger than 1000000 if not mentioned.)
 (space 10)
 (PP No special character such as $ or , may)
 (space 10)
 (P be included in the amount:)
 (R z)
 (blank-line 20)
 (space 10)
 (P Name)
 (space 5)
 (P Phone-no)
 (PP)
 (Looking-for X1 Y1 Z1)
 (no-higher-than Z1 z)
 (Education-applicant X1 x1 y1)
 (at-least x1 Z)
 (EQ y1 y)
 (Applicant X1 X z1 Y))


((Recruiting-1 X Y)
 (blank-line 20)
 (space 10)
 (PP Enter the minimum degree required as follows,)
 (space 15)
 (PP 1 for bachelor,)
 (space 15)
 (PP 2 for master,)
 (space 15)
 (PP 3 for PhD, or)
 (space 15)
 (P 0 if the degree requirement is not mentioned in the
      problem:)
 (R Z)
 (blank-line 5)
 (space 10)
 (PP Or simply enter an x if no specific)
 (space 10)
 (P area of major is spefified:)
 (Y y)
 (blank-line 5)
 (space 10)
 (PP What's the maximum annual pay that should not
     be exceeded?)
 (space 10)
 (PP Enter any number larger than 1000000 if not mentioned.)
 (space 10)
 (PP No special character such as $ or , may be)
 (space 10)
 (P included in the amount:)
 (R z)
 (blank-line 5)
 (space 10)
 (P Enter the experience required:)
 (R X1)
 (space 10)
 (P Enter the minimum number of years:)
 (R Y1)
 (blank-line 20)
 (Looking-for Z1 x1 y1)
 (no-higher-than y1 z)
 (Education-applicant Z1 y2 Z2)
 (EQ Y2 y)
 (Experience-applicant Z1 y2 Z2)
 (EQ Y2 X1)
 (at-least Z2 Y1)
 (Applicant Z1 X x2 Y))


((Recruiting-2 X Y)
 (Education-input Z x y)
 (Looking-for-input Z z X1)
 (Experience-input-1 Z Y1 Z1)
 (Experience-input-2 Z x1 y1)
 (blank-line 10)
 (space 10)
 (P Name)
 (space 5)
 (P Phone-no)
 (PP)
 (Looking-for Z z1 X2)
 (no-higher-than X2 X1)
 (Education-applicant Z x2 y2)
 (EQ Z2 y)
 (at-least y2 z1)
 (Experience-applicant Z z2 X3)
 (EQ z2 x1)
 (at-least X3 y1)
 (Applicant Z X Y3 Y))
.

((Recruiting-3 X Y)
 (Education-input Z x y)
 (Looking-for-input Z z X1)
 (Experience-input-1 Z Y1 Z1)
 (Experience-input-2 Z x1 y1)
 (Experience-input-3 Z z1 X2)
 (blank-line 15)
 (space 10)
 (P Name)
 (space 5)
 (P Phone-no)
 (PP)
 (Looking-for Z z Y2)
 (no-higher-than Y2 X1)
 (Education-applicant Z Z2 x2)
 (at-least Z2 x)
 (EQ x2 y)
 (Experience-applicant Z y2 z2)
 (EQ y2 Y1)
 (at-least Z1 z2)
 (Experience-applicant Z X3 Y3)
 (EQ X3 x1)
 (at-least y1 Y3)
 (Experience-applicant Z Z3 x3)
 (EQ Z3 z1)
 (at-least X2 x3)
 (Applicant Z X Z3 Y))
```

/* The following relations define synonyms for the previously defined commands. Instead of capitalizing each command, the uesr may enter lower case comands. */

```
((recruiting-0 X Y)
 (Recruiting-0 X Y))
((recruiting-1 X Y)
 (Recruiting-1 X Y))
((recruiting-2 X Y)
 (Recruiting-2 X Y))
((recruiting-3 X Y)
 (Recruiting-3 X Y))
```

III. Project Team Assignment Problems:

/* The following relations generate prompts and capture user inputs. Texts of prompts are listed in a more readable format for more natural presentation. When implemented, the format presented for the Recruiting problems must be used.
*/

```
((Job-title-input X Y Z)
 (blank-line 5)
 (space 10)
 (PP Enter the position title of interest, or simply
     enter an x if the required current title is not
     mentioned:)
 (R Y)
 (PP) (PP) (space 10)
 (PP Enter the latest date before which personnel must enter
     the organization in order to be considered for the
     new assignment. The format of your input is (mm dd yy).
     If no deadline of entry is mentioned, enter (99 99 99)
     as your response.)
 (R Z))


((Communication-skill-input X Y)
 (blank-line 5)
 (space 10)
 (PP Enter the required minimum grade for communication
     skill. Or enter 0 if the communication skill grade is
     not mentioned:)
 (R Y))


((Project-status-input X Y)
 (blank-line 5)
 (space 10)
 (PP Enter the deadline date the personnel must become
     available in order to be considered for the new
     assignment. The format of your response should
     be (mm dd yy). Enter (99 99 99) if the requirement
     is not mentioned.)
 (R Y)
 (blank-line 5))
```

/* The following relations define the main bodies of the interactive query patterns for the Project Team Assignment Problems. Predefined building blocks are used to generate prompts and to capture user inputs. The variables presented herein are generated by the interpreter. The user may use a different set of variables for ease of manipulation. */

```
((Assignment-0 X Y)
 (Education-input Z x y)
 (Job-title-input Z z X1)
 (Communication-skill-input Z Y1)
 (Project-status-input Z Z1)
 (space 10)
 (P Name) (space 5)
 (P Phone-no) (PP)
 (Education Z x1 y1)
 (at-least x1 x)
 (EQ y1 y)
 (Job-title Z z1 X2 Y2)
 (EQ X2 z)
 (no-later-than Y2 X1)
 (Communication-skill Z Z2)
 (at-least Z2 Y1)
 (Project-status Z x2 y2)
 (no-later-than y2 Z1)
 (Employee Z X z2 Y))


((Assignment-1 X Y)
 (Education-input Z x y)
 (Job-title-input Z z X1)
 (blank-line 5)
 (Experience-input Z Y1 Z1)
 (Communication-skill-input Z x1)
 (Project-status-input Z y1)
 (PP) (space 10)
 (P Name)
 (space 5)
 (P Phone-no)
 (PP)
 (Education Z z1 X2)
 (EQ Z2 z)
 (no-later-than x2 X1)
 (Experience Z y2 z2)
 (EQ y2 Y1)
 (at-least z2 Z1)
 (Communication-skill Z X3)
 (at-least X3 x1)
 (Project-status Z Y3 Z3)
 (no-later-than Z3 y1)
 (Employee Z X x3 Y))


((Assignment-2 X Y)
 (Education-input Z x y)
 (Job-title-input Z z X1)
 (Experience-input-1 Z Y1 Z1)
 (Experience-input-2 Z x1 y1)
 (Communication-skill-input Z z1)
 (Project-status-input Z X2)
 (PP) (space 10)
 (P Name)
 (space 5)
 (P Phone-no) (PP)
 (Education Z Y2 Z2)
 (at-least Y2 x)
 (EQ Z2 y)
 (Job-title Z X2 Y2 z20
 (EQ Y2 z)
 (no-later-than z2 X1)
 (Experience Z X3 Y3)
 (EQ X3 Y1)
 (at-least Y3 Z1)
 (Experience Z Z3 x3)
 (EQ Z3 x1)
 (at-least x3 y1)
 (Communication-skill Z y3)
 (at-least y3 z1)
 (Project-status Z z3 X4)
 (no-later-than X4 X2)
 (Employee Z X Y4 Y))


((Assignment-3 X Y)
 (Education-input Z x y)
 (Job-title-input Z z X1)
 (Experience-input-1 Z Y1 Z1)
 (Experience-input-2 Z x1 y1)
 (Experience-input-3 Z z1 x2)
 (Communication-skill-input Z y2)
 (Project-status-input Z Z2)
 (PP) (space 10)
 (P Name)
 (space 5)
 (P Phone-no)
 (PP)
 (Education Z x2 y2)
 (at-least x2 x)
 (EQ y2 y)
 (Job-title Z z2 X3 Y3)
 (EQ X3 z)
 (no-late-than Y3 X1)
 (Experience Z Z3 x3)
 (EQ Z3 Y1)
 (at-least x3 Z1)
 (Experience Z y3 z3)
 (EQ y3 y1)
 (at-least z3 y1)
 (Experience Z X4 Y4)
 (EQ X4 z1)
 (at-least Y4 x2)
 (Communication-skill Z Z4)
 (at-least Z4 Y2)
 (Project-status Z x4 y4)
 (no-later-than y4 Z2)
 (Employee Z X z4 Y))
```

/* The following relations define synonyms. */

```
((assignment-0 X Y)
 (Assignment-0 X Y))
((assignment-1 X Y)
 (Assignment-1 X Y))
((assignment-2 X Y)
 (Assignment-2 X Y))
((assignment-3 X Y)
 (Assignment-3 X Y))
```

/* The following function extend the primitive spacing feature of the interpreter and provide building blocks for screen formating. */

```
((blank-line 5) (PP) (PP) (PP) (PP) (PP))
((blank 5) (blank-line 5))
((blank-line 10) (blank-line 5) (blank-line 5))
((blank-line 20) (blank-line 10) (blank-line 10))
((space 5) (P "     "))
((space 10) (space 5) (space 5))
((space 15) (space 10) (space 5))
((space 20) (space 10) (space 10))
```

APPENDIX G

System Operating Procedure for the Non-User-Assisted Mode

The following procedure is to be followed in the experiment:

1. Boot the system with the MS-DOS diskette.

2. Remove the MS-DOS diskette. Insert the PROLOG diskette.

3. Type PROLOG LOAD SIMPLE then hit <return> key.

4. When in &. type load db then hit <return>.

5. When in &. again, the system is ready for the user to specify his/her query program.

6. PRESS <control> AND P SIMULTANEOUSLY before handing out the first problem.

7. Record the time each problem is handed to the subject.

8. Record another time when the output from the system is displayed, or when the subject has given up that problem.

9. In case the subject runs into an infinite loop, press <control> and C simultaneously to get out. When this situation does occur, the system must be reloaded. To proceed, enter

```
prolog load simple
```

and then enter

```
load db
```

10. Enter QT. to exit the system and return to MS-DOS.

APPENDIX H

System Operating Procedure for the User-Assisted Mode

The following procedure is to be followed in the experiment:

1. Boot the system with the MS-DOS diskette.

2. Remove the MS-DOS diskette. Insert the PROLOG diskette.

3. Type PROLOG LOAD SIMPLE then hit <return> key.

4. When in &. type load db then hit <return>.

Then, type load r and hit <return>.

5. When in &. again, the system is ready for the user to specify his/her query program.

6. PRESS <control> AND P SIMULTANEOUSLY before handing out the first problem.

7. Record the time each problem is handed to the subject.

8. Record another time when the output from the system is displayed, or when the subject has given up that problem.

9. In case the subject runs into an infinite loop, press <control> and C simultaneously to get out. When this situation does occur, the system must be reloaded. To proceed, enter

```
prolog load simple
```

and then enter

```
load db  <return>  and  load r  <return>.
```

10. Enter QT. to exit the system and return control to MS-DOS.

BIBLIOGRAPHY

Books

Bailey, Robert W., Human Perfromance Engineering: A Guide for System Designers , New Jersey, Prentice-Hall, Inc., 1982.

Bonczek, R. H., Holsapple, C. W., and Whinston, A. B., Foundations of Decision Support Systems , New York, Academic Press, 1981.

Chang, C. L., and Lee, R. C., Symbolic Logic and Mechanical Theorem Proving , New York, Academic Press, 1973.

Clark, K. L., and Tarnlund, S.-A., (Eds.), Logic Programming New York, Academic Press, 1982.

Clocksin, W. F., and Mellish, C. S., Programming in Prolog , New York, Springer-Verlag, 1981.

Coombs, M., J., and Alty, J. L., (Eds.), Computing Skills and the User Interface , New York, Academic Press, 1981.

Date, C. J., Introduction to Database Systems , (3rd Edition), Menlo Park, California, Addison-Wesley Publishing Co., 1981.

Halstead, M., Elements of Software Science , New York, Elsevier Computer Science Library, 1977.

Keen, Peter G. W., and Scott Morton, M., Decision Support Systems: An Organizational Perspective , Massachusetts, Addison-Wesley, 1978.

Machlup, F., and Mansfield, U., (Eds.), The Study of Information - Interdisciplinary Messages , New York, John Wiley & Sons, 1983.

Martin, James, Managing Data Base Environment, New Jersey, Prentice-Hall, 1983.

____________, Design of Man-Computer Dialogues , New Jersey, Prentice-Hall, 1973.

Montgomery, Douglas C., Design and Analysis of Experiments , New Jersey, Prentice-Hall, 1980.

Ray, Howard, An Empirical Investigation of the Effects of Individual Differences and Data Models on the Ease-of-Use of Database Query Facilities by Casual Users , PhD Dissertation, North Texas State University, 1984.

Ryan, T. A., Joiner, B. L., and Ryan, B. F., MINITAB - Student Handbook , Boston, Massachusetts, Duxbury Press, 1976.

Shneiderman, Ben, Software Psychology , Cambridge, Massachusetts, Winthrop Publishers, Inc., 1980.

Teorey, Toby J., and Fry, James P., Design of Database Structures , Englewood Cliff, New Jersey, Prentice-Hall, Inc., 1982.

Ullman, J. D., Principles of Database Systems, Rockville, Maryland, Computer Press, 1982.

Articles

Bateman, R. F., "A Translator to Encourage User modifiable Man-Machine Dialog," in Sime, M. E., and Coombs, M. J., (Eds), Designing for Human-Computer Communication ,Academic Press, NY.

Bedard, J., Gray, G. L., and Mock, T. J., "Decision Support Systems and Auditing," Center for Accounting Research, School of Accounting, University of Southern California, Working Paper No. 49, (April 1983).

Benbasat, Izak, and Taylor, Ronald N., "An Experimental Investigation of Some MIS Design Variables," MIS Quarterly , (March 1977), pp.37 - 49.

Boies, S. J., "User Behavior on an Interactive Computer System," IBM Systems Journal , 13, No. 1, (1974), pp. 1 - 18.

Bonzcek, R. H., Holsapple, C. W., and Whinston, A., "The Evolving Roles of Models in Decision Support Systems," Decision Sciences , Vol. 11, No. 2, (April 1980), pp. 337 - 356.

________________, "A Generalized Decision Support System Using Predicate Calculus and Network Data Base Management," Operations Research , Vol. 29, No. 2, (March-April 1981), pp.263 - 281.

Brooks, Ruven E., "Studying Programmer Behavior Experimentally: The Problems of Proper Methodology," Communications of the ACM , Vol. 23, No. 4, (April 1980), pp.591 - 597.

Carlson, Eric D., Grace, Barbara F., and Sutton, Jimmy A., " Case Study of End User Requirements for Interactive Problem-Solving Systems," MIS Querterly , (March 1977), pp. 51 - 63.

Chamberlin, Donald D., "Relational Data-Base Management Systems," ACM Computing Surveys , Vol. 8, No. 1, (March 1976), pp.43 - 66.

Chen, E. T., "Program Complexity and Programmer Productivity," IEEE Transactions on Software Engineering , Vol. SE-3, (1978), pp.187 - 194.

Christensen, K., Fitsos, G. P., and Smith, C. P., "A Prospective on Software Science," IBM Systems Journal , Vol. 20, No. 4, pp.372 - 387.

Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM , Vol. 13, No. 6, (June 1970).

Codd, E. F., "Relational Database: A Practical Foundation for Productivity," Communications of the ACM , Vol. 25, No. 2, (February 1982), pp.109 - 117.

Cuff, Rodney N., "On Casual Users," International Journal of Man-Machine Studies, 12, 1, (1980), pp. 163-187.

Codd, E. F., "Seven Steps to Rendezvous with the Casual User," Data Base Management, 1, 1, (1974), pp. 179-199.

Curtis, Bill, "Measurement and Experimentation in Software Engineering," Proceedings of the IEEE , Vol. 68, No. 9, (September 1980), pp.628 - 641.

Curtis, B., Sheppard, P., Borst, M. A., and Love, T., "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics," IEEE Transactions on Software Engineering , Vol. SE-5, (1979), pp.96 - 104.

Dahl, Veronica, "On Database Systems Development Through Logic," ACM Transactions on Database Systems , Vol. 7, No. 1, (March 1982), pp.102 - 123.

Dickson, Gray W., James A Senn, and Norman L Chervany, "Research in MIS: The Minnesota Experiments," Management Science, 23, 9, (1977), pp. 913-923.

Dolk, D. R. and B. R. Konsynski, "Knowledge Representation for Model Management Systems," IEEE Transactions on Software Engineering , Vol. SE-10, No. 6, (November 1984), pp. 619 - 628.

Eason, K. D., and Damodaran, L., "The Needs of the Commercial User," in Coombs, M. J., and Alty, J. L., (Eds), Computing Skills and the User Interface , New York, Academic Press, 1981.

Embley, David W., and Nagy, George, "Behavioral Aspects of Text Editors, " ACM Computing Surveys , Vol. 13, No. 1, (March 1981), pp. 33 - 70.

Enrenreich, S. L., "Query Language: Design Recommendations Derived from the Human Factors Literature," Human Factors, 23, 6, (1981), p. 709.

Fitter, M. J., and Cruickshank, P. L., "Doctors Using Computers: A Case Study," in Sime, M. E., and Coombs, M. J., (Eds), Designing for Human-Computer Communication , Academic Press, NY.

Gaines, B. R., and Shaw, M. L. G., "Dialog Engineering," in Sime, M. E. and Coombs, M. J. (Eds), Designing for Human-Computer Communication ,New York, Academic Press, 1983.

Gallaire, H., "Impacts of Logic on Data Bases," Proceedings of International Conference on Very Large Data Bases , (1981), pp.248 - 259.

Gallaire, H., Minker, J., and Nicolas, J.M., "Logic and Databases: A Deductive Approach," ACM Computing Surveys , Vol. 16, No. 2, (June 1984), pp.153 - 186.

Gingras, Lin and Ephraim R. Mclean, "Designers and Users of Information Systems: A Study in Differing Profiles," Proceedings of the Third International Conference on Information Systems , 1, 1, (December, 1982), pp. 169-181.

Grant, J., and Minker, J., "Answering Queries in Indefinite Databases and the Null Value Problems," Tech. Rep. 1374, Computer Science Dept., University of Maryland, College Park, (1983).

Harris, L. R., "User Oriented Data Base Query with the ROBOT Natural Language Query System," International Journal of Man-Machine Studies , 9, (1977), pp. 697-713.

Harris, L. R., "The Advantages of Natural Language Programming," in Sime, M. E., and Coombs, M. J., (Eds), Designing for Human-Computer Communication ,New York, Academic Press, 1983.

Henderson, John C., and Paul C. Nutt, "The Influence of Decision Style on Decision Making Behavior," Management Science , 26, 4, (April, 1980), pp. 371-386.

Hill, I. D., "Natural Language Versus Computer Language," in Sime, M. E. , and Coombs, M. J. (Eds), Designing for Human-Computer Communication , New York, Academic Press, 1983, pp. 55 - 72.

James, E. B., "The User Interface: How We May Compute," in Coombs, M. J. , and Alty, J. L., (Eds), Computing Skills and the User Interface , New York, Academic Press, 1981.

Jarke, M., and Koch, J, "Query Optimization in Database Systems," ACM Computing Surveys , Vol. 16, No. 2, (June 1984), pp. 111 - 152.

Keen, P.G.W., "Interactive Computer Systems for Managers," Sloan Management Review , 1, 1, (Fall 1976), pp. 1-17.

Keen, Peter G. W., "Adaptive Design for Decision Support Systems," Data Base , Vol. 12, No. 1/2, (Fall 1980), pp. 15 - 25.

Kelly, J. F., "An Iterative Design Methodology for User-Friendly Natural Language Office Information Applications," ACM Transactions on Office Information Systems , Vol. 2, No. 1, (March 1984), pp. 27 - 41.

Kennedy, T. C. S., "Some Behavioural Factors Affecting the Training of Naive Users of an Interactive Computer System," International Journal of Man-Machine Studies , 7, (1975), pp.817 - 834.

Loomis, Mary E. S., "The Nature of Database Management for Effective Decision Support Systems," in Holsapple, C. E. and Whinston, A. B., (Eds.), Data Base Management: Theory and Applications , (1983), pp. 155 - 174.

Lucas, Henry C. Jr., "A Descriptive Model of Information Systems in the Context of the Organization," Database , (Winter 1973), pp. 27 - 39.

Lucas, Henry C. Jr., and Nielsen, N. R., "The Impact of the Mode of Information on Learning and Performance," Management Science , 26, 11, (1980), pp. 982 - 993.

Mason, R. O., and Mitroff, I. I., "A Program for Research on Management Information Systems," Management Science , 19, 1, (1973), pp. 475-487.

Mayer, Richard E., "The Psychology of How Novices Learn Computer Programming," ACM Computing Surveys , Vol. 13, No. 1, (March 1981), pp. 121 - 141.

Methlie, Leif B., "Data Management for Decision Support Systems," Data Base , Vol. 12, No. 1/2, (Fall 1980), pp. 40 - 46.

Miller, Lance A., and Thomas, John C., "Behavioral Issues in the Use of Interactive Systems," International Journal of Man-Machine Studies , Vol. 9, (1977), pp. 509 - 536.

Moran, Thomas P., "An Applied Pshchology of the User," ACM Computing Surveys , 13, 1, (March 1981), pp. 1-10.

Moher, Thomas, and Schneider, G. Michael, "Methods for Improving Controlled Experimentation in Software Engineering," Proceedings of the Fifth International Conference on Software Engineering , (1981), pp. 224 - 233.

Mozeico, Howard, "A Human/Computer Interface to Accomodate User Learning Stages," Computing Practices , 25, 2, (February 1982), pp. 100-104.

Mynatt, Barbee T., "The Effect of Semantic Complexity on the Comprehension of Program Modules," International Journal of Man-Machine Studies , (August 1984), 21, pp. 91-103.

Nickerson, Raymond S., "Why Interacitve Computer Systems Are Sometimes Not Used By People Who Might Benefit from Them," International Journal of Man-Machine Studies , 15, 1, (1981), pp. 470-480.

Nolan, Richard L., and Wetherbe, James C., "Toward a Comprehensive Framework for MIS Research," MIS Quarterly , (June 1980), pp.1 - 19.

Palme, J., "A Human-Computer Inbterface Encouraging User Growth," in Sime, M. E., and Coombs, M. J., (Eds), Designing for Human-Computer Communication , New York, Academic Press, 1983, pp. 139 - 156.

Rasmussen, J., "The Human as a Systems Component," in Smith, H., and Green, T., (Eds.), Human Interaction with Computers , New York, Academic Press, 1980, pp. 67 - 96.

Reising, John M., Ward, Sharon L., and Rolek, Evan P., "Some Thoughts on Improving Experiments," Human Factors , Vol. 19, No. 3, (1977), pp.221 - 226.

Reisner, Phyllis, "Use of Psychological Experimentation as an Aid to Development of a Query Language," IEEE Transaction on Software Engineering , 3, 1, (1977), pp. 218-229.

Reisner, Phyllis, "Human Factors Studies of Database Query Languages: A Survey and Assessment," ACM Computing Surveys , 13, 1, (march 1981), pp. 13-31.

Rich, E., "Natural-Language Interfaces," Computer , Vol. 17, No. 9, (September 1984), pp. 39 - 50.

Rockart, J., and Flannery, L., "The Management of End User Computing," Communications of the ACM , Vol. 26, No. 19, (October 1983), pp. 776 - 784.

Roland, Ronald J., "A Model of Organizational Variables for DSS," Data Base , (1981), pp. 63 - 72.

Robinson, J. A., "A Machine-Oriented Logic Based on the Resolution Principle," Journal of the ACM , Vol. 12, No. 1, (January 1965), pp. 23 - 41.

Rouse, William B., "Human-Computer Interaction in the Control of Dynamic Systems," ACM Computing Surveys , Vol. 13, No. 1, (March 1981), pp.71 - 100.

Scott Morton, M., and Huff, S., "The Impact of Computers on Planning and Decision-Making," in Smith, H., and Green, T., (Eds.), Human Interaction with Computers , New York, Academic Press, 1981, pp.177 - 202.

Sheil, B. A., "The Psychological Study of Programming," ACM Computing Surveys , Vol. 13, No. 1, (March 1981), pp.101 - 120.

Shneiderman, B, and Mayer, R., "Syntactic/Semantic Interactions in Programming Behavior: A Model and Experimental Results," International Journal of Computer and Information Sciences , 7, (1979), pp. 219 - 239.

Shneiderman, Ben, "Improving the Human Factors Aspect of Database Interactions," ACM Transactions on Database Systems , 3, 4, (December 1978), pp. 417-439.

Smith, H. T., "Human-Computer Communication," in Smith, H., and Green, T., (Eds.), Human Interaction with Computers , New York, Academic Press, 1980, pp. 5 - 38.

Thadhani, A. J., "Interactive User Productivity," IBM Systems Journal , 20, No. 4, (1981), pp. 407 - 423.

Thomas, J. C., "Psychological Issues on the Design of Database Query Languages," in Sime, M. E., and Coombs, M. J., (Eds), Designing for Human-Computer Communication , New York, Academic Press, 1983, pp. 173 - 208.

Thomas, J. C., and Carroll, J. M., "Human Factors in Communication," IBM Systems Journal , Vol. 20, No. 2, (1981), pp. 237 - 263.

Van Emden, M. H., and Kowalski, R. A., "The Semantics of Predicate Logic as a Programming Language," Journal of the ACM , Vol. 23, No. 4, (October 1976), pp. 733 - 742.

Vassiliou, Y., Jarke, M., Stohr, E. A., Turner, J. A., and White, N. H., "Natural Language for Database Queries: A Laboratory Study," MIS Quarterly , (December 1983), pp.47 - 61.

Wang, Michael S. Y. and James F. Courtney, JR., "A Conceptual Architecture for Generalized Decision Support System Software," IEEE Transactions on Systems, Man, and Cybernetics , Vol. SMC-14, No. 5, (September/October 1984), pp. 701 - 711.

Warren, David H. D., "Efficient Processing of Interactice Relational Database Queries Expressed in Logic," Proceedings of International Conference on Very Large Data Bases , (1981), pp.272 - 281.

Welty, C., and D. W. Stemple, "Human Factors Comnparison of a Procedural and a Nonprocedural Query Language, " ACM Transactions on Database Systems, Vol. 6, No. 4, (September 1981), pp. 626 - 649.

Wiederhold, G., "Databases," IEEE Computer , Vol. 17, No. 10, (October 1984), pp. 211 - 223.

Zloof, M. M., "The Query-by-Example Concept for User-Oriented Business Systems," in Sime, M. E., and Coombs, M, I., (Eds), Desinging for Human-Computer Communication , New York, Academic Press, 1983.