Improving the Human Factors Aspect of Database Interactions

BEN SHNEIDERMAN
University of Maryland

The widespread dissemination of computer and information systems to nontechnically trained individuals requires a new approach to the design and development of database interfaces. This paper provides the motivational background for controlled psychological experimentation in exploring the person/machine interface. Frameworks for the reductionist approach are given, research methods discussed, research issues presented, and a small experiment is offered as an example of what can be accomplished. This experiment is a comparison of natural and artificial language query facilities. Although subjects posed approximately equal numbers of valid queries with either facility, natural language users made significantly more invalid queries which could not be answered from the database that was described.

Key Words and Phrases: human factors, database systems, data models, query languages, natural language interfaces, psychology, experimentation
CR Categories: 4.33, 4.6, 3.72

1. INTRODUCTION

As questions of technical feasibility and performance of database systems are resolved, increased attention is being paid to human factors. There is widespread recognition that future systems will be commercially viable only if the user interface is in harmony with user skills and task requirements. Management increasingly focuses on human factors, but technical professionals have shown little predilection to go beyond introspection and their own experience. Unfortunately, the background of a systems or language designer may be profoundly different from the background of the intended users. Even if this were not the case, casual introspection hardly seems an adequate basis to develop costly and widely used computer and information systems.

The programming language community has begun to take a more psychologically oriented approach to studying programmer behavior and utilization of language facilities [1–6]. Research in this area is leading to improved guidelines

This work was partially supported by the National Science Foundation under Grants MCS-76-03142 and MCS-77-18641.

Author's address: Department of Information Systems Management, University of Maryland, College Park, MD 20742.

for use of currently available programming languages, suggestions for development of future languages, techniques for evaluating program quality, and recommendations for program development methodologies. Related work is being done on programmer aptitude, programmer ability, and team behavior. This research is leading to fundamental theoretical models of programmer behavior which may provide a basis for making predictions and improvements. Research in decision support systems is germane but psychological results are only beginning to appear. Developments in specific applications areas such as library information retrieval, hospital information systems, or airlines reservation are also useful. Studies in database system usage may draw on these areas but the unique issues require a fresh formulation.

This paper presents frameworks for discussing human factors research, potential research methodologies, and specific issues for evaluation.

2. FRAMEWORKS FOR DISCUSSING DATABASE USAGE

The wide variety of applications and users of database systems may produce confusion in the designer's mind and result in a system which is optimized for only a limited subset of the application domain. The ensuing categories may help organize the design process, aid in evaluation of user facilities, and suggest directions for experimental research. These categories are not entirely independent and the sections in each category represent discrete partitions of a continuum. An attempt has been made to be thorough, but no claim for completeness is made.

2.1 Functions

Functions are the operations that users wish to perform on the database. An item may be field, record, collection of records, file, or the entire database. Primary transaction-oriented functions include:

(1) *Insertion* of one or more items. This operation typically includes the specification of the keys and related data.
(2) *Deletion* of one or more items. This operation typically requires the user to specify a key which is then located in the database. If the record is found, it is deleted and may trigger other changes to the database.
(3) *Retrieval* of information from the database. The user provides a query and the database system returns required information, possibly storing it for later use. A retrieval may be as complex as the copying of the entire database.

Ancillary functions include:

(4) *Locking and unlocking* of items to provide for integrity during concurrent processing.
(5) *Privacy check* to ensure that the user is permitted to perform the function requested.
(6) *Data definition* to create a schema or subschema. This includes description of items and relationships in the database.
(7) *Utility functions* include database administration operations such as an

initial load, physical reorganization, logical restructuring, data translation, performance statistics collection, and data validation.

2.2 Tasks

Tasks are components in the performance of the previously mentioned functions. To accomplish one of the functions, the user must perform one or more of these tasks:

(1) *Learning* the syntax and semantics of the function specification. A typical goal is to reduce the time of learning for a function. Database facilities which are easy to learn to use may be convenient for only a limited subset of the functions. The traditional tradeoffs in programming languages of ease of learning and power of expression apply here. For example, Basic is believed to be relatively easy to learn but has weak control and data structure facilities.

(2) *Composition* of the syntax required to perform a function. Composition includes writing a program or a query, formulating a natural language query, or even responding to a menu selection frame on an interactive terminal. We hope that facilities which are easy to learn are also easy to compose with, but this is not always the case. An easy to learn facility may be so limited that composing useful functions is difficult.

(3) *Comprehension* of function syntax composed by someone else. It is often necessary to read syntax composed by others for learning or other purposes. Easily composed syntax may not be easy to comprehend. Comprehension is often a component of other tasks.

(4) *Debugging* of syntax or semantics written by oneself or others to correct errors. Debugging requires comprehension and composition ability but includes other complex cognitive skills. Database application programs may be debugged using traditional programming techniques, but natural language, menu selection, and query language programs will require novel debugging strategies. The central problem will be to provide users with feedback to help them determine whether the semantics of the function they invoke correspond with their intentions.

(5) *Modification* of a function written by oneself or others. Existing database queries will often be the basis of new queries. This task requires composition and comprehension skills.

This categorization is taken directly from previous work on programming languages, but seems to be appropriate. Learning and composition may be more important in database applications, but the full range of tasks will be required. New techniques for debugging database requests in query languages appear to be an attractive area for research.

2.3 Interaction Modes

Interaction modes are the facilities for accessing the database system. A wide range of available modes are designed to accommodate the variety of users who may require database interaction.

These modes include:

(1) *Host language embedding* of new syntactic forms into well-established programming languages such as Cobol, PL/I, or Fortran. The embedding may be by the simple invocation of subprograms, which requires no alteration of syntax or special translators. Alternatively, new syntax may be added, requiring a preprocessor or modified translator. Most contemporary database management systems which require application programmer intervention use this method. Implementation is relatively easy and training is reduced. Unfortunately, developing applications can be time consuming.

(2) *Self-contained language* which provides all the facilities for performing the available functions. This method allows developers to create an elegant language of their own design without the constraints of older programming language translators. Self-contained languages require learning of new syntactic and semantic forms which are generally easy to use but not as powerful as a traditional programming language such as Cobol.

(3) *Computer-directed* function specification by menu selection, fill-in-the-blank, or parametric requests. With minimal training, an inexperienced or unskilled person can respond to multiple choice questions or requests for parameters. Syntax learning and typing can be avoided, but the range of options available may be small and this mode may be time consuming. The relatively small range of options facilitates testing and can lead to relatively few errors when the system is released.

(4) *Natural language* interfaces which eliminate the need for syntax learning. This attractive possibility has enticed numerous artificial intelligence researchers to work on question answering systems: The February 1977 issue of the ACM Special Interest Group on Artificial Intelligence has reports from no less than 52 projects on natural language interaction with databases. Disadvantages may include long clarification dialog, high CPU overhead, and an illusion of unlimited machine intelligence which inhibits careful thinking on the user's part.

(5) *Human intermediary* to assist users in formulating function requests. By providing a human intermediary to help interpret requests, the user is freed from syntactic concerns and is aided in query formulation. The extra cost, the opportunity for misinterpretation, and the possibility that the intermediary acts as a barrier may be disadvantageous.

2.4 Retrieval Response Types

Since a strong emphasis is placed on the retrieval function in database systems, a finer categorization is appropriate:

(1) *Simple verification* of the presence or absence of a specified item, or that the value of a field is acceptable. This kind of operation is the basis for credit card verification and similar systems. The response may be provided by a simple flashing light or a brief indicator on a terminal.

(2) *Single record* retrieval when a key is provided. Inventory, airline reservation, insurance policy, or similar systems operate by providing users with a prescribed set of fields when a primary key field is presented.

(3) *Record collection* retrieval when a key or Boolean predicate is provided. Typically the user is looking for the set of primary key or records which

satisfy some criterion such as employees whose age is above 30 or who work for the sales department. Library information retrieval systems operate this way.

(4) *Total report* listing of all information in a file. This response type is the more traditional batch-oriented production job, but it may be invoked from an interactive terminal. In this case performance is a critical issue since execution time may be measured in hours. Output formats for total reports probably would differ from single record formats.

This crude categorization does not reveal the complexity of many transactions which have multiple components. A banking transaction may include a simple verification of the existence of the account, a check on the balance, an update, and a printout of the transaction summary.

2.5 Query Features

For the same reason that retrieval response types were delineated, query features need a finer categorization. The following is based on Reisner's [7] list:

(1) *Simple mapping* returns data values when a known data value for another field is supplied. An example would be: find the names of employees in department 50. Comparison operations such as equal to, less than, or greater than may be included in the specification of the simple mapping.

(2) *Selection* of all the data values associated with a specified key value, for example: Give the entire record for the employee whose name is John Jones.

(3) *Projection* in the relational model is an entire column or domain or a relation. In general this is the query for finding all the domain values for a domain, for example: Print the names of all employees.

(4) *Boolean queries* are those which permit AND/OR/NOT connectives such as: Find the names of employees who work for Smith and are not in department 50.

(5) *Set operation* queries are those which permit intersection, union symmetric difference, or other set operators. Boolean queries can be converted to set operation queries.

(6) *Built-in functions* provide special-purpose library functions to aid in question formulation. A common set of these include MAXIMUM, MINIMUM, AVERAGE, COUNT, and SUM, for example: Print the average salary in department 50.

The query types described thus far are an easy to learn subset which has been used in our pilot studies and Reisner's work. She found that even nonprogrammer subjects scored better than 70 percent correct in composing these query types using SEQUEL. Programmers scored better than 80 percent correct.

More complex query features include:

(7) *Combination* queries are the result of using the output of one query as the input for another. Reisner uses the term "composition," but this paper has already used this term for another purpose. An example would be: Find the names of all departments which have more than 30 employees and then print the department managers' names.

(8) *Grouping* of items with a common domain value, such as department

number, for example: Print the names of departments where the average salary is greater than $15,000. The employees must be grouped by departments before the averaging can take place. SEQUEL uses the GROUP BY statement, while Furtado and Kerschberg [8] developed an algebra of quotient relations to facilitate these kind of queries.

(9) *Universal quantification* corresponding to the "for all" concept of the first-order predicate calculus. This operation is difficult for most users to comprehend and work with. The ambiguous use of the word "all" in English and the subtlety of set equivalence and set containment contribute to the difficulties [9]. Date's [10] presentation in the relational calculus and his discussion of division in the relational algebra provide additional perspectives on this feature.

The query features discussed thus far are available in most query languages that have been designed for the binary, tree, and relational models of data.

Codd defines relational completeness of a query language as the property of having the descriptive power of the first-order predicate calculus [11]. Relational completeness has been used as a primitive measuring rod for the selective power of a query language. Two problems come to mind with this yardstick in evaluating the human factors aspect of a language:

(1) Many queries that can be written with a relationally complete language are extremely difficult to compose or comprehend. Few people claim to have a thorough understanding of first-order predicate calculus.

(2) Many common, useful, simple to understand, and potentially easy to express queries are outside the bounds of relational completeness. For example, in a table of distances between adjacent cities, finding the shortest path between two remote cities is not included in relational completeness. Similarly, in a table of employees and their managers, finding the names of all the employees that a given individual manages at all levels, is not a relationally complete query.

As an alternative to relational completeness, we need a taxonomy of queries which orders queries from simple to complex. Reisner and others have argued for a level structured or layered query facility which allows users to compose simple queries and gradually increase their capacity for composing more complex queries. Reisner's feature list and her theoretical linguistic model are a beginning but much research remains.

2.6 User Types

Diversity of user types may require a spectrum of interaction modes. It seems unlikely that one mode will accommodate all user types satisfactorily, nor should one interaction mode be considered optimal for all users. The users should be further subdivided according to their problem domain knowledge. The following categorization offers generic groupings.

(1) *Nontrained intermittent* users who infrequently access the database. This is the proverbial "casual user" which Codd [12] describes as the most rapidly expanding class of users. These people will have no syntactic knowledge and

may have little knowledge of the organization of the data. They may be expected to understand the application domain such as a library index or airline schedule, but their ability to pose reasonable questions may be shallow. A system to allow users to get airline schedule information from a terminal should require minimum skills: typing ability cannot be expected and there may even be anxiety over the use of a computer terminal. The best interaction mode is probably computer directed or the use of a human intermediary.

(2) *Skilled frequent* users who make daily use of the database. These users may be willing to learn a simple syntax for performing functions, but they are more interested in their own work than in programming computers. The easier the system is for them to use, the more frequent will be their use. This category includes the skilled secretary, lawyer, engineer, or manager.

(3) *Professional database* users whose main role is to provide access to the database. They will apply their long experience and accept substantial training. Professional users will be concerned with efficiency and the quality of their work.

3. RESEARCH METHODS

The research methods of traditional experimental psychology and human factors will have to be adapted to the complex cognitive tasks required for interacting with database systems. The high variance in subject performance encountered in programming language experimentation (where students with similar educational backgrounds or professionals at the same job level in an organization had 30 to 1 ratios in performance) will certainly be repeated. Increasing the size of the subject population will help in combatting this problem, but we are limited by the fact that specialized skills, which few people have and take long to acquire, may be necessary for some experiments. Experiments with facilities for nontrained intermittent users will be easier to accomplish because the subject pool is vast. Professional users are highly paid and employers are reluctant to permit them to participate in time-consuming experiments unless sufficient benefits can be guaranteed. As in most psychological and human factors research, the bulk of subjects will probably be university undergraduates who will be compelled to participate for course requirements or additional credit. As much as possible, the intended user population for the facility should be recruited for the research.

The three fundamental paradigms for research are introspection, field studies, and controlled experiments. This author feels that all three are useful, but that controlled experiments must ultimately be the basis for the most profound theoretical and practical conclusions.

3.1 Introspection

This research method depends on an individual's sensitivity to the cognitive skills required while performing one or more of the five tasks: learning, composition, comprehension, debugging, and modification. Subjects may be asked to make subjective judgments about the ease of use of an interaction mode or a query

feature, or make comparisons. These judgments are highly influenced by training and often do not correlate well with performance metrics. Subjects may perceive a task as being easy, but do poorly at the task.

Protocol analysis, in which subjects are asked to verbalize their procedures in performing a task, is an intriguing and popular technique. Analyzing the protocol can be a tedious task and it is hard to demonstrate the generality of any conclusions because of extreme intersubject variability and because analyzing lengthy protocols from numerous subjects is demanding. Still, intriguing insights may be gained from working with expert users who are sensitive to their performance.

3.2 Field Studies

Field studies or case studies are an attempt to study practitioners in a realistic environment. The goal is to evaluate actual performance in a precise manner, with minimal interference to standard practice. Not all the variables can be carefully controlled, but unexpected events or insights may reward the careful observer/experimenter.

In programming language work, researchers captured the job stream of submitted jobs on a particular day and analyzed the types and frequencies of statement usage [13], error patterns [14, 15], and performance data such as timing or number of lines of code written. Similar strategies could be applied in database usage to capture information about function requests, response types, or query features. These data would be helpful in developing new facilities, improving available languages, or revising training procedures.

Field studies are appropriate for investigating complex issues such as an entire data management system, interaction modes, or a management strategy for team organization. These issues are not amenable to straightforward controlled experimentation but some conclusions can be drawn from a field study. Critics complain about the lack of controls and the possible influence of external factors such as organizational morale, individual motivation, and personality differences. Results are not always generalizable or replicable, but important insights or suggestions for controlled experimentation can be derived from field studies. A final criticism is that field studies tend to measure current practice rather than the improvement obtainable from new strategies.

3.3 Controlled Experimentation

Controlled experimentation depends on a reductionist approach which minimizes uncontrolled bias. A small number of factors, say one to four, are chosen as critical to the performance of a task. These factors are varied while all other factors are kept constant, if possible. If varying a factor results in a statistically significant difference in performance, then suggestions for practical implementation can be developed. The factors which the experimenter varies are the *independent variables* and the performance measures are the *dependent variables.*

The advantage of controlled experimentation is that the results are generalizable and replicable. Critics argue that controlled experiments too often have a narrow focus and produce trivial results. Those favoring the reductionist strategy claim that each result is like a small tile contributing to an emerging mosaic of

user behavior. A sound theoretical foundation, produced from controlled experimentation on fundamental factors can lead to predictions in novel situations and recommendations for future database management systems design.

A typical experiment might be designed to compare the ability of a range of user types (nontrained intermittent, skilled frequent, and professional) to comprehend retrieval queries requiring set operations in a computer-directed or self-contained language. The two experimental factors are the user type and the interaction mode. The three levels in the user type and the two levels of interaction mode make this a "3 by 2" or "3 × 2" experiment requiring six groups. If ten subjects were selected for each group then a total of 60 subjects would be required. A comprehension test would be administered to each group and a statistical analysis would be performed to determine if there were statistically significant differences among the groups. Statistical significance implies that the results were probably not a result of chance occurrences. Typical levels of statistical significance demanded are 5 percent or 1 percent. A difference in mean scores is not sufficient in controlled experiments; statistical significance must be demonstrated, in this case by the use of a technique called *analysis of variance.*

A major problem in the design of this hypothetical experiment is that it is extremely difficult to ensure that the subjects using the computer-directed technique are similar to the subjects using the self-contained language. Programming language research has demonstrated an enormous variation in subject performance, even for those subjects with the same job titles, experience, or training. This variation often obscures any experimental factor. Questionnaires or pretests have been insufficient to screen out these individual differences, nor does increasing the group size seem effective. The skills being tested are complex and difficult to measure. To deal with this problem, researchers have begun to move to *within subject* designs rather than *between group* designs. In a within subject design, a subject takes two tests and a comparison is made between the two test scores: subjects compete against themselves, not against other subjects whose background or ability may differ. For the proposed experiment each subject would perform the comprehension task using the computer-directed and the self-contained language interaction modes.

The problem with this method is that it may make a big difference which mode is tested first. The obvious response to this problem is to *counterbalance* the ordering of modes by presenting half the subjects with the computer-directed mode first and the other half with the self-contained language first. A statistical analysis should be made of the *order effect* to see if it is significant. The more similar the tasks, the more pronounced the order effect. This remedy has the advantage that fewer subjects are needed, but the disadvantage that each subject must spend more time. This basic two factor within subjects design can be adapted to a wide variety of issues. A simple one factor, or multiple factor designs can be used as well. The more factors tested, the more complex the design and administration, and the greater potential for *main effects* to be obscured.

For a guide to designing experiments see [16] and [17].

3.4 Measurement Techniques

Quantifying human performance in these complex cognitive tasks is a challenge. A central problem in this area is developing adequate techniques for measuring

learning, composition, comprehension, debugging, and modification. Again the experience with programming language experimentation will be a guide and will be most appropriate when the host language embedded interactions mode is studied.

If we define learning as the increase in the ability to perform a task, then before and after tests seem fitting. Such pretests and posttests are the traditional tools of educational psychologists. The contents of these tests would be items measuring one of the four remaining tasks.

Composition of database functions is a complex skill which has multiple components such as problem comprehension, program design, and program coding. We will assume that debugging is a separable task. We are interested in a person's ability to compose a function specification and can simply require subjects to perform a number of composition tasks. In database accessing, most functions can be specified quickly when common high-level database manipulation languages are used. For host language embedded systems, the composition task may require substantially more time. Grading the work can be difficult if partial credit is allowed and therefore duplicate grading is advised to improve reliability. Time to completion has been used but is unreliable since the fastest workers may not be the most accurate. If incorrect results can be returned for reworking, then the time to correct completion can be used. The number of errors might also be recorded. A multiple choice test in which subjects are asked to choose proper syntax, rather than generate it, is a possible alternative, but is subject to criticism, because it is not the normal mode of composition.

A particular problem arises when a composition task is to be tested using a natural language interaction mode: How can we present a problem without using the natural language formulation which would be precisely the solution? It has been suggested that subjects be given the answers which the database would produce and require subjects to produce the questions. This is not altogether ludicrous, but is unrealistic because several questions might generate the same answers and users rarely work this way. A more reasonable solution seems to be to present subjects with a situation and a database and ask them to compose natural language queries to respond to the situation. For example, using a university database we required subjects to write queries which would assess the quality of education in various departments. We expected queries like: List the professors and their rank by department, or how many students were in each section of a course? The queries may be rated on their appropriateness in responding to the situation and whether the database contained sufficient data to respond to the query.

Measuring comprehension seems to be easier than composition, but the definition of comprehension is elusive. In programming language experiments, it is possible for a subject to comprehend the high-level function of a program but not understand the low-level details of how the program operates, and vice versa. In database accessing the interactions tend to be smaller discrete entities which are more amenable to testing with multiple choice or fill-in-the-blank type question. Subjects can be presented with functions and asked to execute them against a database. If the function is written in a self-contained language, the subject can be asked to write a natural language interpretation. Time to correct completion and subjective measures of difficulty can be used.

The debugging task can be studied by providing subjects with an incorrect function specification and requiring them to repair the errors. Since syntactic errors will probably be caught by the system, semantic bugs are the main object of study. A function specification in natural language can be used when interaction modes other than natural language are being studied. For natural language debugging the situational method described earlier can be used. Debugging research should be particularly interesting, since the main problem in database access debugging will be the determination that a semantic error has occurred: syntactically correct functions will produce a reasonable event, how are users to know that there is a bug?

Modification studies will be similar to composition experiments, except that the original function will be provided. Since most interactions are short, modification may be viewed as an original composition.

The high variability in subjects dictates collection of background data. Statistics revealing the months of system experience, programming background, knowledge of problem domain, or other significant variables should be collected for *correlational* or *covariance analyses.*

4. RESEARCH ISSUES

This section presents a number of popular issues which would be candidates for experimental study.

4.1 Natural Language Versus Artificial Language

Substantial effort has been put into developing systems for natural language interactions with computers. Impressive systems such as Weizenbaum's Eliza [18], Winograd's SHRDLU [19], and Woods' Lunar Sciences System [20] were all predecessors for the 52 projects, reported on in the ACM Special Interest Group on Artificial Intelligence Bulletin (February 1977), whose goal was to develop natural language interfaces for database systems. These researchers argue that since most users are competent in using natural language, it would be the ideal language for database interaction. Training would be eliminated and users would have no inhibition to using the computer.

The limited number of critics (Montgomery [21], Hill [22], and Shneiderman [23]) argue that natural language interfaces may not be preferable in every situation. Just because users know natural language syntax does not ensure that they know the semantics of database interaction or the semantics of the information stored in the database. These limitations may lead to several problems;

(1) Unrealistic expectations of the computer's power. Users might pose questions such as: How can I improve profits? or Is the defendant guilty? These questions involve value judgments and complex ideas which computers cannot and probably should not be relied upon to answer (Weizenbaum [24]).

(2) Attempts to request information which is not contained in the database, thus wasting time and effort, while increasing frustration. Natural language users may not be aware of the contents and semantics of the database. In a corporate database it may be reasonable to inquire about departmental

average salaries but comparisons with industry wide salaries may be inappropriate.

(3) By allowing users to use natural language without training we allow the ambiguities of English syntax to pollute the query process, driving developers to design long and tedious clarification dialogs. This clarification dialog will have to take place even for sophisticated users who are careful in their selection of words. Particularly annoying are difficulties with existential and universal quantification.

(4) Typical users may not be aware of the semantics of question asking. Although they may know English syntax, they may not have thought of what kind of questions could be answered by a database system. By teaching users a concise and precise artificial language we are also teaching the semantics of question asking. Having the tool of a well-learned query language may enable users to compose complex queries which might not have occurred to them otherwise.

(5) Finally, the overhead of creating and maintaining a natural language interface will always be larger than for a concise query language or a menu selection process.

These criticisms do not imply that natural language inferfaces are useless, only that their domain of application may be less broad than has been suggested.

Research experiments which compare natural language usage with artificial languages can be performed even though the on-line systems are not available. Paper and pencil experiments are less than optimal in this case, but they will provide useful evidence until workable on-line facilities are developed.

4.2 Specification Versus Procedural Languages

Database query languages provide a new domain for the controversy between specification languages, which describe the goal, and procedural languages, which describe a process for arriving at the goal [25]. This classification is not discrete: a language may fit in the continuum between the extremes described here. Specification languages are usually viewed as being of higher level and having shorter length than procedural languages. In database querying, specification languages may be more appealing since the database functions are brief and lend themselves to specification.

The relational algebra is seen as being more procedurally oriented than the relational calculus [26] or calculus-based languages such as QUEL [27], SEQUEL II [28], FORAL [29, 30], and Query-by-example [31]. Commercial query languages like those for System 2000 [32] or Model 204 [33] blend procedural and specification concepts.

All of these languages are more specification oriented than host embedded systems which usually require programmed logic control and record at a time processing.

The psychological foundations of this issue are intriguing. Are there cognitive style variables which might indicate which method is perferable for certain subjects or tasks? Could composition be easier with procedural languages but comprehension be easier with specification languages? Paper and pencil studies seem appropriate here since only semantic issues are involved and syntactic or terminal interface problems are secondary.

4.3 Linear Keyword Versus Two-Dimensional Positional Languages

Some proposals for database query languages, such as QUEL [27] or SEQUEL II [28] have been keyword oriented, basing their structure on traditional programming language design. Other proposals have sought to include a two-dimensional notation in which the position of items was critical: SQUARE [34], Query-by-example [31], CUPID [35], and FORAL LP [30]. In the latter class of languages, very few keywords are used and a graphics support system may be required.

Supporters of keyword-oriented languages argue that the keywords help in learning and query composition, by associating query semantics with familiar terminology. Supporters of two-dimensional positional query languages claim that confusion can be reduced by using positional notation or special shapes to indicate components of queries.

In Query-by-example, users are provided with a screen display of a relation skeleton. Queries are composed by filling in columns with literals or underlined examples. A keyboard controlled cursor makes placement of items easy. In CUPID, lightpen touches enable users to move shapes and generate diagrams which represent a query. In FORAL LP the user employs a lightpen to select operations and data elements displayed as a binary network on the screen.

The fundamental psychological issue of keyword use versus abstract shape notation is unstudied. For each of the five tasks, does English language knowledge confuse or facilitate users? An early pilot study of ours suggests that high SAT verbal nonprogrammers preferred the keyword-based approach of SEQUEL, while high SAT math nonprogrammers preferred the mathlike positional notation of Query-by-example. Users who emphasize right brain visual intuitive thinking may have different preferences from those who prefer left brain verbal deductive thinking.

4.4 Hardware Factors

The design of user interactive teminals may critically affect user performance variables such as fatigue, anxiety, motivation, and satisfaction. Traditional human factors research has concentrated on these performance variables issues, but contemporary work is necessary with on-line transaction-oriented database access.

The size of the display screen, brightness of the display, glare, flicker, contrast, typefont size, typefont design, graphics or color capability, and physical placement may all affect users. Keyboard design issues such as tactile or audio feedback, placement of keys, angle of keys, and placement of keyboard, are also significant variables. Ancillary equipment such as special-purpose keypads, lightpens, sonic pens, touch sensitive screens, mouse-controlled cursors, or joysticks may facilitate interactive usage.

An underlying issue is the response time of the system. Long delays are usually disruptive and disturbing, but the variance of response time may be as critical as the mean response time [36, 37]. If the variance of response time is small, users incorporate the waiting into their work patterns by preplanning future queries or attending to other functions, but if the variance is large, users must maintain a continued high level of awareness and become anxious if response time grows. Performance and satisfaction may actually improve if responses are delayed so as

to minimize variance. Another tactic may be to inform users of the estimated waiting time if the response is to be more than a few seconds.

4.5 Menu Selection, Fill-In, and Parametric User Interfaces

Three easy to implement, computer-directed, user interface modes that are frequently used are:

(1) *Menu selection.* The terminal screen is filled with a set of numbered choices.
(2) *Fill-in-the-blank.* The user provides a word, number, or phrase response to a line of text.
(3) *Parametric.* The user provides a formatted set of numbers, codes, or words in response to a prepared line of text.

Menu selection requires little or no user training and has the advantage that users may be informed about additional system features. A succession of menu selections can be used to produce a tree search. Choosing the terms in the menus, the number of items in each menu, and the sequencing of menus requires careful planning so as to minimize user error. A simple exit from the menu sequence, the opportunity to return to previous menus, and help frames should be provided. With a high-speed communication line, menu selection does not lead to boredom and can be an effective method. Users should be allowed to respond to a menu as soon as it appears or as soon as their choice appears.

Fill-in-the-blank questions require users to be aware of response formats, but lengthy displays of menus are avoided. Some training may be required but with experience users quickly become proficient. Exit, backup, and "help" facilities should be provided.

Parametric systems require still more training, but usage is extremely fast and user satisfaction is increased because users feel more in control. Airline reservation systems, which require the flight number and date plus a function code, are successful applications of this mode. Training requirements are increased, but not severely. More complex error-handling modules are required, but help frames which list the set of choices or commands are usually easy to prepare. When an error occurs with a novice user, the system could default to a slower menu selection approach.

Experimental tests need to be conducted to clarify the applications which are most suitable for these three modes. How many choices are optimal for a single menu? How many parameters can users be expected to master if their usage is infrequent? What kind of frustrations are encountered in each of these three modes? How does variation in response time affect user satisfaction? These and other questions seem suitable topics for experimental comparison.

4.6 Schema Design

Once a data model has been selected for an application, the database designer/analyst must create the schema. Although machine efficiency issues may intervene, every effort should be made to provide the easiest to use schema.

In the relational model, since joining or linking of relations increases the complexity of query formulation, it is preferable to have higher degree relations which may not be in third or Boyce-Codd normal form. Unfortunately, first or second normal form relations exhibit update anomalies [38] and may obscure the

semantics of the data. These two constraints produce an optimization question which can be resolved by experimental study. Groups of users could be given a relational database in various stages of normalization and be required to perform database access tasks.

In the tree structured model of data, many-to-many relationships can be expressed by building two separate trees which are logically paired or by having redundant fields in segments. No evidence has been presented about the relative merits of these two methods in terms of user comprehension. Two separate trees seem more confusing but redundant entries are more confusing if updates are required.

Another problem that arises in tree structures is depth versus breadth tradeoffs. Although this problem involves machine performance considerations, the human factors component is important. An organizational division may be parent to departments which are parents to employees, or a division may be parent to employees directly with department included as a field of the employee segment. These and other design considerations should be experimentally studied.

Network models which require currency pointer maintenance present additional difficulties for programmers. Record types which have more than one owner record type and cyclic schema structures are particularly confusing to novices.

The binary relational model [39] has an elegant and simple basis, but the complexity of schema diagrams can lead to confusion. Studies need to be performed to assess ease of use of binary relations which do not have the convenience of a grouping structure such as a record or a segment.

In the record-oriented models, redundancy of data plays an important role in implementation efficiency and in user ease. Redundancy may facilitate query tasks, but complicate insertion and deletion.

Since variable names are critical in conveying the semantics of the data, experiments should be performed with the intended user population to ensure that the proper meaning has been conveyed. Even domain values, such as job titles or student grades (e.g., not everyone may be familiar with each of the grade codes: A,B,C,D,F,I,W), should be tested to ensure user comprehension.

4.7 Data Model Selection

The heated debates of the past few years over the relative merits of proposed data models, have cooled down and observers are more concerned with the incorporation of schema-oriented database systems in realistic environments. Supporters of each model have included features from their competitors and the relative merits of each model are becoming somewhat clearer.

McGee's [40] paper gives a set of criteria for evaluating data models which is based largely on human factors considerations, including the following:

Simplicity. A model should have the smallest possible number of structure types, composition rules, and attributes.

Elegance. A model should be as simple as possible for a given direct modeling capability.

Picturability. Model structures should be displayable in pictorial form.

Modeling directness. A model should not provide equivalent direct modeling techniques.

Overlap with coresident models. A model should mesh smoothly with other coresident models.

Partitionability. A model should have structures which facilitate the administrative partitioning of data.

Nonconflicting terminology. A model should use terminology which does not conflict with established terminology.

These ill-defined qualitative criteria are a useful starting point, but few guidelines are offered for measuring the simplicity or elegance of a data model. Picturability appears to be a useful attribute, but how can we be sure? Other visually based schemes such as detailed flowcharts have lately come under fire [4]. Another problem with these criteria is that we may get conflicting impressions from different users of the relative simplicity of two data models. In short, we need more precise, replicable, and generalizable results which can be obtained from controlled experiments.

Tree structured data models appear to be most successful when the data are perceived to have a natural tree structure, but is cumbersome otherwise. The relational model is elegant, but critics have complained that it is too "syntactic" and that models which can represent more semantic information are preferred [41]. Network or data-structure-set models permit sophisticated structures to be described, but the concommitant complexity and the use of currency pointers increases the difficulty of usage. These subjective impressions culled from the literature and comments from colleagues need to be clarified and verified. I suspect that no model will emerge as the best, but that several data models will be necessary to accommodate the variety of users.

5. RESEARCH DIRECTIONS

Initial steps have been taken in controlled experimental research on human factors in database systems. The relational query languages SQUARE and SEQUEL have been studied by Reisner and her colleagues [43, 7] using programmers and nonprogrammers. Thomas and Gould [44] have done a detailed study of Query-by-example using nonprogrammers and Ascher and Gould studied an IQF-like query facility [45].

Durding, Becker, and Gould [46] performed an intriguing experiment using word organization problems to test the ability of nonprogrammers to use a variety of data structures.

Thomas provides an excellent survey of these experiments and outlines psychological issues in database management [47]. Lochovsky and Tsichritzis have compared usage of three data models by programmers for three specific problems [48, 49]. Brosey and Shneiderman compared the hierarchic and relational models independently of query facilities, by giving comprehension and memorization tasks to novice programmers [50]. Recently Greenblatt and Waxman compared three languages for the relational model: Query-by-example, SEQUEL, and a relational algebra variant [51].

A good beginning has been made but much work remains. Successful research can have a dramatic impact on future systems and will influence the acceptability of information systems by the general public.

6. NATURAL VERSUS ARTIFICIAL QUERY LANGUAGE EXPERIMENT

A small experiment was run to demonstrate the research methods described in Section 3, and to investigate the question raised in Section 4.1 concerning the relative advantages of natural and artificial query languages. This experiment was not intended to be conclusive, but merely an example of the design, administration, analysis, and conclusion components of an experiment.

Recent results from Small and Weldon [52] raised further doubts about the advantages of natural language query facilities. In that experiment, 20 subjects were required to compose queries in natural language English and a subset of SEQUEL. Answers were marked on sample databases with some notation to indicate the origin of the answers and subjects were required to compose the queries at interactive computer terminals. Experimental aides in a separate room played the role of the natural language or subset-SEQUEL processor and provided error messages as needed. Users of natural language had to follow the tabular patterns, but had syntactic and naming freedom. Each subject performed in both interaction modes and counterbalanced orderings were used in this repeated measure design. Harmonic times to first response and to correct response both indicated that subset-SEQUEL was superior. Those using SEQUEL in the latter half of the experiment were the highest scoring group.

Small and Weldon conclude that:

> The common assumption that ordinary, everyday English is the ideal way to communicate with computers is not supported by the present results. Subjects were not reliably more accurate using English than using SEQUEL. They were reliably faster using SEQUEL, suggesting that the structured language is easier to use.

We felt that requiring subjects to provide queries in this constrained format did not measure a subject's capacity to formulate queries to resolve problems. Secondly, requiring subjects to understand the patterns of table usage did not represent true natural language usage. To resolve these two problems we decided to offer subjects a situation problem where they had to formulate questions on their own. Natural language users were told about a department store employee database and were asked to pose questions to help them decide which department to work in. Subset-SEQUEL users were given brief training, a seven item comprehension test, and were told to pose questions in subset-SEQUEL. The criterion of success was the number of relevant queries that each subject asked in each mode.

6.1 Procedure

(1) *Subjects.* The subjects were 22 University of Maryland students enrolled in an undergraduate Cobol programming and information systems course, some of whom may have had previous programming experience. Subjects were tested in the eighth week of a 15-week course. A standard Experimental Consent Agreement used by the Human Subject Committee of the Department of Information Systems Management was circulated and signed by each participant.

(2) *Materials.*

(a) SEQUEL Experiment cover sheet describing the sequence of events

to be followed in learning SEQUEL, taking the comprehension test, and doing the situation problem.

(b) SEQUEL Instruction booklet consisting of four double-spaced pages with SEQUEL sample questions and answers. Single-table databases were assumed, thus eliminating the need for the FROM clause, and only simple mappings, AND/OR logic and five arithmetic functions (SUM, COUNT, AVG, MAX, MIN) were shown.

(c) Comprehension questions included three SEQUEL samples which subjects were to execute against the given database and four English queries which had to be translated into SEQUEL.

(d) Situation Problem (SEQUEL) contained the following instructions:

> You are considering taking a job in one of the departments of the example table on Page 1. You can work in any department you wish. You can review and compare departments by querying the database in SEQUEL. Write all queries you might possibly ask in making your decision. DO NOT RETURN TO PREVIOUS SECTION FOR REFERRAL. NOTE: This section will be scored on both the number of questions you formulate and the relevance of each question. (Put the question down even if you feel your syntax is incorrect.)

(e) Situation Problem (English) contained the following instruction:

> You are considering taking a job in one of the departments of a department store. You can work in any department you wish. In order to help you with your decision of which department to work in, you can review and compare departments by asking questions about the department store's employees. You may obtain information concerning the employee's: names, salaries, managers, departments employees work in, years of employment, and age of employees. Write all questions you might possibly ask in making your decision. DO NOT RETURN TO A PREVIOUS SECTION FOR REFERRAL.
>
> NOTE: This section will be scored on both the number of questions you formulate and the relevance of each question.

Two forms of the test booklet were constructed from these materials: NAT-SEQ was ordered *eabcd* and SEQ-NAT was ordered *abcde*. Half the subjects received NAT-SEQ booklets and the other half received the SEQ-NAT booklets. Fifteen minutes were allowed for studying the SEQUEL Instruction booklet and for each of the situation problems. Ten minutes was allowed for the SEQUEL Comprehension questions.

(3) *Grading.* SEQUEL comprehension scores were graded as being correct or incorrect, but some freedom was given in syntactic form. In any case, these scores were not intended for analysis, but merely to assure that the subjects had learned some SEQUEL in the 15-minute training period. None of the subjects scored less than a three out of seven and the average score was 3.95.

The Situation problem results were graded as invalid or valid. Valid queries had to be answerable from the database and relevant to the task of deciding which department to work in. If subjects repeated question templates such as: What is the maximum salary in the TOY department? What is the maximum salary in the SHOE department? only one point was assigned for the entire group. Such patterns appeared in both SEQUEL and English forms since the GROUP BY feature was not taught. Minor spelling or syntactic errors were accepted in both forms, as long as the intent was clear.

6.2 Results

Tables I–III present the detailed results. *T*-tests showed no significant differences, even at the 0.10 level, between valid English and valid SEQUEL queries. The order effect was not significant, however, the number of invalid queries did differ significantly ($p<0.01$) between the English and SEQUEL groups. The order effect for invalid queries was also significant ($p<0.01$): the NAT-SEQ group had more invalid queries than the SEQ-NAT group.

Since the invalid queries provided the significant differences, an informal review of the kinds of invalid queries was undertaken. Those using English often let their imagination go and came up with interesting and relevant questions which could not be answered from the database. Typical examples include:

Are the managers lenient concerning tardiness and absences?
Do people like working in (the) department?

Table I. Mean Number of Queries Posed in Situational Problem with Standard Deviations in Parentheses

	Subset-SEQUEL		Natural language		Combined	
	Valid	Invalid	Valid	Invalid	Valid	Invalid
NAT-SEQ	2.54 (1.29)	0.36 (0.91)	2.45 (1.21)	2.90 (2.16)	2.49 (1.22)	1.64 (1.61)
SEQ-NAT	3.54 (2.11)	0.09 (0.30)	2.64 (3.41)	0.90 (1.81)	3.09 (2.76)	0.49 (1.26)
Combined	3.04 (1.70)	0.23 (0.66)	2.54 (2.49)	1.91 (1.94)	2.79 (2.95)	1.07 (2.02)

Table II. NAT-SEQ Raw Data, Mean Scores, and Standard Deviations

	NAT-SEQ				
	Natural language situation		Comprehension questions correct	SEQUEL situation	
Subject No.	Valid	Invalid		Valid	Invalid
1	2	2	3	3	1
2	3	0	5	2	0
3	3	1	5	5	0
4	2	6	3	3	0
5	0	5	3	3	0
6	3	0	6	3	0
7	2	4	6	0	0
8	4	3	4	1	0
9	4	3	4	3	0
10	1	2	6	3	0
11	3	6	3	3	3
Total	27	32	48	28	4
Mean	2.45	2.90	4.36	2.54	0.36
Standard deviation	1.21	2.16	1.22	1.29	0.91

Table III. SEQ-NAT Raw Data, Mean Scores, and Standard Deviations

	SEQ-NAT				
	Natural language situation		Comprehension questions correct	SEQUEL situation	
Subject No.	Valid	Invalid		Valid	Invalid
1	0	1	5	6	1
2	8	0	7	8	0
3	3	0	5	3	0
4	1	0	6	2	0
5	1	0	6	2	0
6	3	0	7	5	0
7	0	0	5	2	0
8	0	4	5	2	0
9	0	4	5	2	0
10	3	0	6	4	0
11	1	0	5	1	0
Total	29	10	62	39	1
Mean	2.64	0.90	5.64	3.54	0.09
Standard deviation	3.41	1.31	1.03	2.11	0.30

What is the starting salary for each department?
What is the personality of the managers?
What type of clientele does the department cater to?
How often are raises awarded?

6.3 Discussion

The fact that there were not significant differences in the number of valid queries in the natural language and subset-SEQUEL groups can be used to support advocates of natural language facilities or precise concise artificial languages. Adherents of artificial languages might argue that with only 15 minutes of training in SEQUEL, the performance is impressive and that with additional training, SEQUEL users should be able to surpass natural language users. Learning additional features of the SEQUEL language should also improve performance.

Supporters of natural language front ends might complain that the SEQUEL training period and the comprehension test helped subjects by providing examples to follow. They might also complain that natural language users were not given a chance to become familiar with the application domain.

Future experiments will have to be directed to determine the importance of familiarity with the application domain, familiarity with the data items stored in the computer, amount of prolog needed to prepare natural language users to deal with the computer, capacity of the system and the user to produce effective clarification dialogs, and the importance of typing skill for communication at a terminal.

The significant differences on the invalid query tally do support the reservations about natural languages use made in Section 4.1 of this paper. Natural language users were far more likely to pose unanswerable queries. Only 3 of the 22 subjects posed invalid queries during their SEQUEL sessions while 12 of the same 22

subjects posed invalid queries during their natural language sessions. Nine out of the eleven who had natural language first made invalid queries in natural language, but only three out of eleven subjects who had natural language second made invalid queries in natural language. These results suggest that the structuring during SEQUEL use was learned and applied during the natural language session.

These results should not be interpreted as a condemnation of natural language usage, but as an aid in determining which applications are suitable for natural language front ends and what training users should be given. User knowledge of the application domain seems to be critical: without this prerequisite, natural language usage would be extremely difficult. Secondly, user knowledge of the structure of the data in the computer and what each item means appears to be vital. Finally, experience in asking questions against a specific database is probably helpful. Thus the ideal candidate for natural language usage may be the experienced frequent users of a manual information system, but these users are likely to appreciate the simplicity, brevity, and precision of a structured query language. The casual user with little knowledge of the application area, understanding of the data structure, and experience in posing queries may find natural language facilities more confusing. Realistic applications for natural language would be situations where people have familiarity with the application area, data structure and queries, but are infrequent users. Typical situations that fit this description include library card catalogs, airline schedules, or banking transactions. More research is necessary to support these hypotheses.

This simple one factor counterbalanced within subjects experiment raises more questions than it answers, but this fits well with our goal of provoking further experimental research. The results must be replicated under a variety of conditions before any conclusions or recommendations can be made.

ACKNOWLEDGMENT

Kevin Storms carried out the experiment described in Section 6. The referees' comments were helpful in improving the presentation. The University of Maryland Computer Science Center provided support for computer services.

REFERENCES

1. GANNON, J.D., AND HORNING, J.J. The impact of language design on the production of reliable software. *IEEE Trans. Software Eng. SE-1,* 2 (1975).
2. GANNON, J.D. An experimental evaluation of data type conventions. *Comm. ACM 20,* 8 (Aug. 1977), 584–595.
3. SHNEIDERMAN, B. Exploratory experiments in programmer behavior. *Int. J. Computer and Inform. Sci. 5,* 2 (June 1976), 123–143.
4. SHNEIDERMAN, B., MAYER, R., MCKAY, D., AND HELLER, P. Experimental investigations of the utility of detailed flowcharts in programming. *Comm. ACM 20,* 6 (June 1977), 373–381.
5. SHNEIDERMAN, B. Measuring computer program quality and comprehension. *Int. J. Man-Machine Studies 9* (1977).
6. WEISSMAN, L. A methodology for studying the psychological complexity of computer programs. Ph.D. Th., U. of Toronto, Toronto, Ont., Canada, 1974.
7. REISNER, P. Use of psychological experimentation as an aid to development of a query language. *IEEE Trans. Software Eng. SE-3,* 3 (1977), 218–229.
8. FURTADO, A.L., AND KERSCHBERG, L. An algebra of quotient relations. Proc. ACM SIGMOD Int. Conf. Manage. of Data (1977), pp. 1–8.

9. THOMAS, J.C. Quantifiers and question-asking. IBM Res. Rep. RC 5866, IMB T.J. Watson Res. Ctr., Yorktown Heights, N.Y., 1976.
10. DATE, C.J. *An Introduction to Database Systems.* Addison-Wesley, Reading, Mass., 2nd ed., 1977.
11. CODD, E.F. Relational completeness of data base sublanguages. In *Data Base Systems,* R. Rustin, Ed., Prentice-Hall, Englewood Cliffs, N.J., 1971.
12. CODD, E.F. Seven steps to rendezvous with the casual user. In *Data Base Management,* J. W. Klimbie and K. L. Koffeman, Eds., North-Holland Pub. Co., Amsterdam, 1974, pp. 179–199.
13. KNUTH, D.E. An empirical study of FORTRAN programs. *Software—Practice and Experience 1* (1972), 105–133.
14. LITECKY, C.R., AND DAVIS, G.D. A study of errors, error-proneness and error diagnosis in Cobol. *Comm. ACM, 19,* 1 (Jan. 1976), 33–37.
15. YOUNGS, E. A. Human factors in programming. *Int. J. Man-Machine Studies 6,* 3 (1974).
16. EDWARDS, A. L. *Experimental Design in Psychological Research.* Holt Reinhart and Winston, New York, 1968.
17. CRONBACH, L.J. *Essentials of Psychological Testing.* Harper and Row, New York, 3rd ed., 1970.
18. WEIZENBAUM, J. Eliza—a computer program for the study of natural language communication between man and machine. *Comm. ACM 9,* 1 (Jan. 1966), 36–45.
19. WINOGRAD, T. *Understanding Natural Language.* Academic Press, New York, 1972.
20. WOODS, W.A., KAPLAN, R.M., AND NASH-WEBBER, B. The lunar sciences natural language information system. Bolt Beranek and Newman, Cambridge, Mass., June 1972.
21. MONTGOMERY, C.A. Is natural language an unnatural query language? *Proc. ACM Nat. Conf.,* New York, 1972, pp. 1075–1078.
22. HILL, I.D. Wouldn't it be nice if we could write computer programs in ordinary English—or would it? *Honeywell Comptr. J. 6,* 2 (1972), 76–83.
23. SHNEIDERMAN, B., Ed. *Database Management Systems,* Inform. Tech. Ser., Vol. 1, AFIPS Press, Montvale, N.J., 1976, pp. 59–61.
24. WEIZENBAUM, J. *Computer Power, Human Reason.* W. H. Freeman, San Francisco, 1976.
25. LEAVENWORTH, B.M., AND SAMMET, J.E. An overview of nonprocedural languages, SIGPLAN Notices (ACM) *9,* (April 1974), 1–12.
26. CODD, E.F. A relational model of data for large shared data banks. *Comm. ACM 13,* 6 (June 1970), 377–387.
27. HELD, G.D., STONEBRAKER, M.R., AND WONG, E. INGRES: A relational database system. Proc. AFIPS 1975 NCC, Vol. 44, AFIPS Press, Montvale, N.J., pp. 409–416.
28. CHAMBERLIN, D.D., et al. SEQUEL 2: A unified approach to data definition, manipulation, and control. *IBM J. Res. and Develop. 20,* 6 (Nov. 1976), 560–574.
29. SENKO, M.E. The DDL in the context of a multilevel structured description: DIAM II with FORAL. *Data Base Description,* Proc. IFIP-TC-2 Working Conf., Wepion, Belgium, Jan. 1975, B.C.M. Douque and G. M. Nissen, Eds., North-Holland Pub. Co., Amsterdam, 1975, pp. 239–258.
30. SENKO, M.E. DIAM II with FORAL LP: Making pointed queries with light pen. Information Processing 77, North-Holland Pub. Co., Amsterdam, 1977, pp. 635–640.
31. ZLOOF, M.M. Query by example. Proc. AFIPS 1975 NCC, Vol. 44, AFIPS Press, Montvale, NJ., pp. 431–437.
32. System 2000 Reference Manual. MRI Systems Corp., Austin, Tex., 1977, pp. 635–640, 1973.
33. Model 204 User Language Reference Manual. Computer Corp. of America, Cambridge, Mass., 1977.
34. BOYCE, R.F., CHAMBERLIN, D.D., KING III, W. F., AND HAMMER, M.M. Specifying queries as relational expressions: SQUARE. Proc. ACM SIGPLAN-SIGIR Interface Meeting, Gaithersburg, Md., Nov. 1973.
35. MCDONALD, N., AND STONEBRAKER, M. CUPID: The friendly query language. Proc. ACM Pacific Conf., San Francisco, April 1975.
36. MILLER, R.B. Response time in man-computer conversational transactions. Proc. AFIPS 1968 SJCC, Vol. 33, AFIPS Press, Montvale, N.J., pp. 267–277.
37. MILLER, L. A study in man-machine interaction. Proc. AFIPS 1977 NCC, Vol. 46, AFIPS Press, Montvale, N.J., pp. 409–421.
38. HEATH, I.J. Unacceptable file operations in a relational data base. ACM-SIGMOD Proc., 1972.
39. SENKO, M.E., ALTMAN, E.B., ASTRAHAN, M.M., AND FEHDER, P.L. Data structures and accessing in database systems, *IBM Syst. J. 12,* 1 (1973), 30–93.

40. McGee, W.C. On user criteria for data model evaluation. *ACM Trans. Database Syst. 1,* 4(Dec. 1976), 370–387.
41. Chen, P. The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst. 1,* 1 (March 1976), 9–36.
42. Kerschberg, L., Ozkarahan, E.A., and Pacheco, J.E.S. A synthetic English query language for a relational associative processor. Proc. Second Int. Conf. Software Eng., San Francisco, 1976, pp. 505–519.
43. Reisner, P., Boyce, R.F., and Chamberlin, D.D. Human factors evaluation of two data base query languages: SQUARE and SEQUEL. Proc. AFIPS 1975 NCC, Vol. 44, AFIPS Press, Montvale, N.J., pp. 447–452.
44. Thomas, J.C., and Gould, J.D. A psychological study of query by example. Proc. 1975 National Computer Conference, AFIPS Press, Montvale, N.J.
45. Gould, J.D., and Ascher, R.N. Use of an IQF-like query language by non-programmers. IBM Res. Rep. RC 5279, IBM T.J. Watson Res Ctr., Yorktown Heights, N.Y., Feb. 1975.
46. Durding, B.M., Becker, C.A., and Gould, J.D. Data organization. *Human Factors 19,* 1 (1977) 1–14.
47. Thomas, J.C. Psychological issues in database management. *Proc. Third Int. Conf. Very Large Data Bases,* Tokyo, 1977.
48. Lochovsky, F., and Tsichritzis, D. User performance considerations in DBMS selection. Proc. ACM-SIGMOD Int. Conf. Manage. of Data, 1977, pp. 128–134.
49. Lochovsky, F. Database management system user performance Ph.D. Diss. U. of Toronto, Toronto, Ont., Canada, 1978.
50. Brosey, M., and Shneiderman, B. Two experimental comparisons of relational and hierarchical database models. *Int. J. Man-Machine Studies* (to appear).
51. Greenblatt, D., and Waxman, J. A study of three database query languages. *Databases: Improving Usability and Responsiveness,* B. Shneiderman, Ed., Academic Press, New York, 1978.
52. Small, D.W., and Weldon, L.J. The efficiency of retrieving information from computers using natural and structured query languages. Rep. SAI-78-655-WA, Science Applications, Arlington, Va., Sept. 1977.

Received January 1978; revised July 1978