EDP ANALYZER
Editor and Publisher: RICHARD G. CANNING

EDP ANALYZER

SEPTEMBER, 1982
VOL. 20, NO. 9

QUERY SYSTEMS FOR END USERS

As we pointed out last month, training end users on office systems could become a large 'hidden' cost. A major determinant of whether users will need a little or a lot of training is the 'language' they must use in order to communicate with these systems. This portion of a computer system has been called the man-machine interface, and more recently, the end-user interface. Clearly, most conventional computer languages aren't suitable for end users. This month we discuss one type of end-user interface—one that is used to query data files—with emphasis on natural language systems and multi-tiered systems.

Filene's, a department store chain headquartered in Boston, Massachusetts, is a part of the giant Federated Department Stores nationwide chain. Filene's has twelve 'upstairs' units and six 'basement' store units in New England and employs about 7,000 people. Last year, they had sales over $350 million.

Filene's is really two stores in one, with traditional ready-to-wear fashions and accessories in the 'upstairs store,' and 'off-price' quality merchandise and off-season purchases in the 'basement store.' Moreover, the retail industry is a very fast-paced one, where the managers (the buyers) in the different departments—in both upstairs and basement stores—need to continually monitor sales and inventory and react as quickly as possible to new trends. As the vice president for management information systems at Filene's told us, the information needs of these buyers are continually changing, so there is little chance that data processing can keep up with their day-to-day requests.

In their search for sales and inventory information, most of the buyers' questions start out, "What were our....?" The information is usually available in a database file, if only the buyers can get at it. For this reason, Filene's has been looking for a user-friendly query system that its merchandising executives, buyers, and staff people could use to enter their own queries on-line to the corporate IBM 4331-2 computer.

ISSN 0012-7523.

In late 1980, they saw a demonstration of the Intellect™ query system from Artificial Intelligence Corporation (AIC) of Waltham, Massachusetts (Reference 1). Intellect is a natural language query system that allows users to type free-form English sentences for their queries. The system interprets the meanings of the words in the query based on: its knowledge of the English language, the fields in the file(s) being accessed, and its specially designed dictionary (called a 'lexicon' by AIC). Intellect then creates a structured query to be processed against the appropriate file(s). The query can contain arithmetic expressions, pronouns that refer back to the previous query, full sentences, partial sentences, or nouns only.

Intellect looks at all possible interpretations of the query and discards those that do not make any sense. If Intellect cannot 'understand' a portion of the query, it displays that portion and asks the user to clarify it. Perhaps a word was misspelled; after the user re-spells that word correctly, Intellect interprets the query. If a portion of the query can have several possible interpretations, Intellect presents the several possibilities and asks the user to choose the correct one; for example, the term 'New York' could be a city or a state. Further, Intellect can be used in an interactive or a batch mode.

Filene's liked Intellect because it could be used with their existing files, because it required very little user training, and because it appeared to be able to handle a wide variety of queries. They decided to start using it in a very controlled fashion. They chose a 'back office' operation with well-defined data—personnel records—and created a system to be used initially by only six employees for benefits administration.

To use Intellect, Filene's first needed to design the dictionary for the employee benefits group. That involved finding out the terms (and their meanings) that these employees use in their work. One AIC employee and the personnel manager spent about fifteen working days, spread out over several months, defining the terms in this dictionary. Meanwhile, another employee, who would eventually be maintaining this particular dictionary as well as creating new ones, attended two one-week courses on Intellect.

AIC also suggested a few changes that Filene's might want to make in their file structures to speed up query processing for the personnel file. The personnel file used for queries is a derived file; it is updated once a week from their payroll file.

Records in the personnel file consist of 120 fields, for each of the 7,000 active and 8,000 inactive employees. Also, records for seasonal and terminated employees are kept on file for two years. With a file that large, response time for queries depends mainly on the indexing scheme used, we were told. If the indexing scheme can divide the file into pertinent segments, then less of the file needs to be searched for each query. For instance, if the question is phrased "How many *active* employees...?" only 7000 records need to be searched. Filene's indexes this personnel file by employee status (active/inactive), the store the employee works in, and the employee name.

Filene's has placed a CRT terminal and a printer in the personnel office for use with Intellect. On the first day of use, last August, the number of pending requests—from the personnel department to data processing—dropped from 34 requests to 20 requests. And this trend has continued; now almost all queries are entered by the Intellect users. So Filene's feels they are on the right track for off-loading some data processing work to end users.

Use of Intellect has speeded up numerous 'typical' types of personnel work. For example, it used to take two days to gather all of the needed information for a person who was retiring from the company, because some of the information was stored in filing cabinets while other information was in computerized systems, such as payroll. Using Intellect, obtaining the computerized information now takes less than a minute.

The first project was very successful, so Filene's is expanding their use of Intellect. They have acquired IDMS, a DBMS from Cullinane Database Systems Inc. of Westwood, Massachusetts, which works with Intellect as a front end. They have hired a data base administrator. And

they are working on two new 'mainline' systems that will merge Intellect and IDMS.

One of these new systems is a merchandising information system that will be used by buyers and merchandising management to monitor sales and inventory content. The data is being stored under IDMS and the system will eventually contain 100,000 style records and be used by hundreds of employees. This system is currently being made operational.

The other new system involves financial data. It will be used, for example, to plan and control the expenses of each work center. This system will be installed later this year.

The merchandising information system has presented several new challenges to Filene's data processing management. First, the user group is not homogeneous, so the terms or phrases to be used in the queries cannot be anticipated easily. Therefore, they had a cross-section of expected users define the terms for the dictionary. Filene's has chosen to index this file by style, number, vendor, class, retail store location, and department. They are beginning a controlled trial of the system this month, and they plan to expand the system to all users next month.

As a second challenge, this system has very marked peaks and valleys of use. Mondays are a peak usage day, because buyers want to know what their top selling items were for the previous week. Wednesdays and Thursdays are very low usage days. To help smooth out this cyclical use, the data processing department has defined a 'Monday-morning report' which they prepare on Sunday night in a batch mode using Intellect, for delivery Monday morning. For each department, the report will list their ten best selling items for the previous week.

Filene's is also taking advantage of the Intellect and IDMS combination in the two newer systems. First, the files used by Intellect will be automatically updated; the merchandising system files, for instance, will be updated twice a week. Second, Intellect will be able to search more than one file to answer a query. Thus the system will to be able to answer the question: "What is the profitability of vendor XYZ?" To answer this question, the system will search one financial file for purchases from that vendor, a merchandise file for sales of that vendor's merchandise, and a second financial file for the cost of markdowns to that vendor's merchandise. This will give truer profitability figures for lines of merchandise, Filene's told us.

And third, Filene's plans to use Intellect to support a hierarchy of queries. For example, at one level a buyer may want to know his or her best selling items for the past week. At a higher level, the division manager may want to know this same information, but for all of the buyers in his or her division. Filene's will make such consolidated ad hoc requests possible from a single Intellect query.

Filene's is pleased that Intellect is helping them off-load work to end users and, at the same time, provide more timely information to employees.

Chevron Oil Field Research Company

Chevron Oil Field Research Company is a wholly-owned subsidiary of Standard Oil of California and performs research for oil and gas exploration for the parent company. Located in La Habra, California, a suburb of Los Angeles, the company has 600 employees, most of whom are technical researchers.

Data processing support for these researchers is provided on two IBM mainframes (a 370/168 and a 3033). Up until a couple of years ago, the company was small enough that they had only a minimal amount of business-type programming—so there are no application programmers in the data processing department. However, there are two programmers and a supervisor in the general services department. These people provide business-type programming using the MARK IV system from Informatics General Corporation, of Woodland Hills, California.

Chevron Oil Field Research has been using MARK IV since 1969, as their primary business programming language, as well as for processing query and report requests. Many of the researchers do not want to write their own MARK IV programs, so the programmers created some special user interface programs. These programs use menus and question-and-answer formats to help the researchers enter their requests. Even so, a good number of the requests were complex

enough that they had to be written and maintained by the programmers.

Whereas during most of the 1970s there had not been much need for business-type programming, as the company grew in size, things gradually changed. The demand for business-type programming increased. Also, maintenance of the customized query and report programs accelerated, to the point where two years ago the programmers were spending 60 percent of their time on this task.

Early last year, Informatics asked Chevron Oil Field Research if they would like to be a test site for a new product—INFORMATICOM®. It includes both (1) a stand-alone work-station that employs a micro-computer and (2) a friendly software front-end, INFORM/DMS, for entering MARK IV requests. The work-station has 64k bytes of memory, a terminal and keyboard, dual floppy disk drives, and a printer. A hard disk unit is optional. And, up to eight terminals can be connected to share common databases. The price ranges from $20,000 to $37,000.

INFORMATICOM includes two languages, each with multiple input formats. INFORM/DMS helps end users enter their own query and report requests. These requests can be processed on the work-station itself, when local files are involved. Or the requests can be passed on to the appropriate mainframe for processing under MARK IV—in which case, the job control language and communications functions, and the logging on and off the mainframe, are performed automatically for the user. The other language is the MARK IV Support System, with levels of input formats for end users as well as programmers. It includes the basic form of INFORM/DMS and provides additional MARK IV facilities for complex report generation.

Chevron Oil Field Research liked the idea of being a test site for this new product. They saw it as a possible solution to their business programming problems. They also believed it might help them train new MARK IV programmers. Further, a larger share of the query and report requests could be entered by the end users themselves.

And then there was the problem of 'consistency of data'—and they thought that INFORMATICOM could help on this, too. The company keeps records of 'cores' that have been made in field drillings—that is, records of the soils and rocks that were encountered. They keep these records not only because of government requirements but also for possible use by future researchers. The data in these records has not been as consistent as the company would like. For instance, the word 'California' was spelled 19 different ways in these records. With INFORMATICOM, they felt that on-the-spot data validation would reduce such data inconsistencies.

The first test work station was installed in May 1981, for use by the three MARK IV programmers. At first it was used mainly to speed up maintenance work, but increasingly they are using it for creating new programs.

The second test work station was installed in August 1981, in the purchase and stores department, for use by the manager and several clerks. It is being used for maintaining inventory files, for such things as furniture, laboratory equipment, oil samples, and the 'cores' just mentioned. The clerical people in the department have learned to enter their own queries against these inventory files.

In addition, the department has set up some smaller files on floppy disks, for either local or mainframe processing. In some instances, these files are transmitted to one of the mainframes, for processing under MARK IV. After the processing has been completed, the files are erased from the mainframe's storage—providing an added level of data security for such data.

Almost immediately after the installation of these two test work stations, the company saw that they did, indeed, help ease the programming and data consistency problems—and decided to purchase the two units.

In all, Chevron Oil Field Research is glad they agreed to test the INFORMATICOM system. It is providing them with a programming work station for their MARK IV programmers and another one for the purchase and stores department. It is enabling them to transfer some MARK IV work to end users, both researchers and clerical staff members. And it is helping them train new MARK IV programmers. So the trial worked out just about as the company hoped it would.

End user interfaces

It is perhaps inevitable that people will communicate with computers using spoken natural language in the future. Many science fiction writers, for example, portray humans communicating with robots and other machines via spoken language. In addition, these writers imagine that such communications will be two-way conversations, with the machines having voice recognition, voice response, and intelligence capabilities.

The field is not yet there, of course, but the query systems that accompany today's database management systems could be considered 'primitive' forerunners of these 'ideal conversational machines.' Research is progressing on many fronts—voice recognition, voice response, artificial intelligence, and natural language processing. Then, too, there is research in combining these disciplines—for example, combining a free-form natural language processor with a voice recognition system.

That is the future. And, until recently, most information systems executives have not needed to think about such futuristic notions, because computers were used mainly by computer professionals who would put up with cryptic, demanding man-machine interfaces. But office systems are now causing this interface area to receive much more attention. 'User friendly' is the catch-word, because these systems will not be used by full-time computer professionals. They will be used by office employees as part of their job. Providing a marvelous new tool is no longer sufficient; it must also have an end user interface which is comfortable, rememberable, helpful, forgiving, *and* efficient.

The term 'end user interface' really applies to any piece of hardware or software that a user deals with when communicating with a computer. It could include a keyboard, light pen, touch screen, any device that controls cursor movement, touch pad, end user language, or system response language. In this report, we will concentrate on the *software* aspects of the man-machine interface.

The following discussion is based partly on ideas presented by Jaime Carbonell of Carnegie-Mellon (Reference 3) and Richard Marcus and J. Francis Reintjes of MIT (Reference 4). In their papers, these researchers discuss desirable features of end user interfaces. They are working on natural language systems and query systems for textual databases, respectively.

Evaluating an end-user interface. When evaluating an end user interface of, say, a specific office system, users and evaluators should ask these questions:

Is it comfortable? This question can also be stated, "Is it natural?" This means, are the human's actions appropriate for the intended applications? For instance, reaching out to touch a touch-sensitive screen may be natural for selecting items from a menu, but very unnatural and uncomfortable for selecting items from a long sequence of menus. Typing may be comfortable for a professional when he or she is entering text or performing an analysis, but a touch-sensitive screen may be more natural for retrieving mail from an electronic mailbox. So the interface needs to be suited to the task.

Is it rememberable? End users who make infrequent use of a system will need procedures that they can remember. This means that both the instructions, that the user must give the system to perform a task, and their sequence should be both logical and meaningful. For example, requiring a user to enter a password to get onto a shared system is logical, but requiring the user to remember and enter a *meaningless* password, such as AL943BZ7, is not.

Is it helpful? Infrequent users will need help when using a computer system—not just the first time they use it, but perhaps every time they use it. As we discussed last month, the system's interface needs to provide some sort of: (1) beginner training, to get one started, (2) in-depth training, to teach users how to perform more sophisticated operations, and (3) refresher training, to aid a user's memory. An important point here is that these on-line training options should be user-initiated; that is, they should only appear when the user asks for them. And the user's request for help should initiate an *appropriate* response from the system. It would be best if the user could choose how much information he or she wants from the system's help function, rather

than always being given either the most detailed or the most basic explanation.

Another aspect of helpfulness is feed-back from the system, such as when it displays its interpretation of the user's request. It may even ask the user if it has made the correct translation. Such feed-back can assure the user that the system is answering the question that was *really* intended. Also, it may teach the user a more concise way of stating the request. Attempting to deal with ambiguous query statements is a necessary, and important, trait of natural language query systems.

Is it forgiving? A forgiving system is one that protects users from inadvertently initiating 'wrong' actions, and makes it easy to correct things when they do make mistakes. Some systems validate all user requests to be sure that they do not contain mis-spelled words, incorrect punctuation, and so on. Other systems double-check user requests which can cause expensive or irreversible actions—such as by asking the user, "Do you really want to abandon this updated file?" And still others allow users to undo the previous action(s).

Forgiveness is an important characteristic that most computer systems have not had. This is why many end users are very uncomfortable with computers; they believe that wrong actions on their part can 'destroy' the system or programs or data.

Is it efficient? Obviously, the more software that is in a system to provide the above four features, the slower the system may run. There is a definite tradeoff here. Yet efficiency is relative. Even a very fast system, when it is performing a long task, can appear inefficient to a new user, who thinks computers do everything instantaneously. Thus, systems which keep users informed about what they are doing are preferred to those which leave the user wondering: "Is anything happening? Has the system gone down?"

These are some of the desirable features of end-user interfaces. We suspect that in future systems, more of the resources will be used in the end-user interface than has been true in the past. These interfaces can apply to any operation a user might want to perform. In an office system, this could be reading electronic mail, adding an appointment to a calendar, creating a report on-line, querying a database, and so on.

To put some reasonable bounds around our discussion in this report, we have chosen to look at end user query systems. And within that subject area, we will discuss mainly *data* databases, as opposed to *textual* databases—although some concepts are applicable to both.

Types of end user query systems

We have encountered four types of end user interfaces for query systems: menus, forms, command languages, and free-form natural language systems. These approaches are not mutually exclusive; they can be combined. The newer systems that we have seen do combine and/or draw upon the desirable features of each approach.

Menus are easy to use. The user sees what options are available and selects one. So menus are good for inexperienced or infrequent users. However, for experienced and frequent users, they can be slow and restrictive, because they often require the user to 'wade through' numerous menus to perform the desired task.

Forms display a fill-in-the-blanks format for querying and data entry. Like menus they help guide infrequent users. And the newer tabular form languages (such as IBM's Query By Example) allow users to ask complex queries in simple ways.

Command languages are not so easy for novice or infrequent users to use, because they must first learn each command in the language—what operations it performs, when it can be used, and how it needs to be stated (perhaps with variables attached). So command-based languages require more user training. But for experienced users, command languages are often preferred, because they allow users to rapidly and concisely specify what they want the system to do.

Natural language systems are also well-suited for novice and infrequent users, because they generally require little or *no* training. The user simply tells the system what is to be done, using his or her own terms and in any sequence. However, natural language interfaces are currently limited to databases with 'restricted domains,' where most of the terms have only one meaning.

Thus they can be used with some data files, but not with most textual files.

Also, natural language systems do not define the scope of the system for the user by listing what can be done—and either explicitly or implicitly identifying what *cannot* be done. Thus, these systems are not inherently instructional. And they are costly to create. Yet, with these shortcomings, many still see natural language as having the ideal characteristics for an end user interface.

With that background, we now look more closely at two types of query systems that are available on the market today. The first is the free-form natural language query system. The second is the multi-tiered system, which combines menus, command languages, and other techniques to make the interface 'comfortable' for novices as well as experienced users.

Natural language query systems

It has long appeared desirable to have a computer system that could understand one or more commonly spoken languages, such as English, rather than requiring humans to learn new computer languages. The field of artificial intelligence has been working on this problem for many years, and we now see renewed interest in the subject. As yet, however, there are only a few truly free-form natural language query systems on the market. Systems that call themselves 'English-like' are *structured* query systems, not natural language systems.

But natural language query systems have negative as well as positive features. Let's look at some of their pros and cons.

A natural language query system is easy to use, say proponents, because users can state queries in their own, familiar terms. As Harris points out (Reference 5), a user can get an answer to a query without having to know much about the query system or the structure of the file being accessed. Proponents of structured query systems cannot make this claim, he says. This aspect comes in handy when database structure changes are made, say Templeton and Kameny (Reference 6), because the users do not need to know in which records the particular data fields are located. They may not even realize that a data structure change has been made. Such changes are handled by the query dictionary, which translates the user's query into the structured query language used for searching (and which is transparent to the user).

Ease of use is one of the main benefits of natural language systems; however, this ease of use does lead to a 'problem': unrealistic expectations. These systems appear to be able to 'understand' anything; they seem almost magical. Thus users assume they can get an answer to any question, so they can (and do) ask questions outside of the scope of the application—questions that cannot possibly be answered. So users need to understand the scope of the application when using a natural language query system. Otherwise, they may become mistrusting of the system, when it does not answer some of their seemingly simple questions. The people at Artificial Intelligence Corporation told us they spend about one hour with new Intellect users to help them understand the scope of the database and to practice making queries.

One common criticism of any natural language is that it is easy to create ambiguous queries. Critics of natural language systems say that users cannot be as precise in their complicated queries. When speaking, people use pauses and voice inflections to combine words to avoid ambiguities. But with written queries in natural language this is not so easy.

Most proponents of natural language concede that *structured* query systems allow experienced users to state the conjunctions in a query more concisely. Inexperienced users often use the conjunctions 'and' and 'or' loosely. 'And' may (incorrectly) be used to mean 'either A or B' or (correctly) 'only where both A and B occur.' Templeton and Kameny give an example: 'Who are the minority and handicapped applicants with FORTRAN and COBOL experience?' This sentence could have four meanings—does it mean only people who are both handicapped and minority or either handicapped or minority people? Likewise, does it mean only people who know *both* FORTRAN and COBOL?

But structured query systems aren't the full answer to ambiguities, either. Harris notes that just because a structured query system has no

ambiguities in the language itself, does not imply that users cannot use it in an ambiguous manner. After looking at a few examples of queries in a structured language, users may believe they understand the formal language. Unfortunately, there is a difference between understanding queries written by others and generating queries yourself. There can be a great deal of subtlety in using structured query systems, such as mis-interpreting 'and' and 'or,' as just described.

The question of dealing with ambiguities is a serious one, because the user wants the system to answer the 'real' question he is asking—which very likely uses ambiguous terms. Both structured query and natural language query systems should attempt to handle ambiguities. Ideally, the system would recognize all possible multiple meanings and ask the user to clarify the meaning. Realistically, this is not yet possible. But the more a system tries to deal with multiple meanings of words, the more valuable it is to the users; otherwise, the system will answer the 'wrong' question without the user being aware of it.

Another common criticism is that, although a natural language is comfortable for speaking, it can be rather tedious for typing because it is too wordy and verbose. Harris counters this argument by saying that natural language query systems allow users to phrase queries in many ways (and still get the same answer). And he believes there is often a more concise wording for a natural language query than for a structured query. His example: "California salesmen" is shorter than "job = salesmen, state = California." As users gain experience, they learn to type only the necessary words, thus making queries in an abbreviated natural language.

There are other concerns about natural language systems. Templeton and Kameny found that truly naive users prefer the guidance of a form. Many fewer options can be presented on a form, but if the user is doing a routine type of task with a standard set of options, a form is preferred.

For general maintenance of the database, some types of query systems are not appropriate, Templeton believes. Maintenance may or may not be considered a job of the query system, but it is a necessary related job. In some applications, users must intermix queries and updates. Natural language and structured query systems do not work well here, because it is too easy to leave out important fields of information when entering the data. Forms are more appropriate for this function.

Another concern is the amount of time a programmer or database administrator must spend setting up the dictionary of user terms and meanings. A natural language application will be more or less fluent depending upon the amount of time spent developing the dictionary. Artificial Intelligence Corporation generally spends about 15 working days to produce an application dictionary. This time includes information gathering, dictionary development, on-site tests, and final delivery. Since this task takes quite a bit of time, Templeton and Kameny believe that natural language query systems are not appropriate for systems with a short life or a small number of expert users.

We also talked with Marjorie Templeton, who is leading a natural language query system project at System Development Corporation; we discussed their EUFID system in our August 1979 issue. She described the types of things designers and users would like natural language query systems to do, but which are not yet possible. This is not to negate the progress made so far, but to point out where these query systems may not yet be appropriate.

People who are designing natural language query systems would like them to understand all types of queries and ambiguities, and catch mistakes. Today's systems do understand a lot, and can recognize that they do not understand a query statement. But they generally cannot give the user much help in correcting the query, except to point out the portions they do not understand.

Processing a query can be slow on a large file or when the query requires access to many files. While this is a problem with any query system, it seems to be more of a problem with a natural language system, says Templeton, because the users may not understand the structure of the da-

tabase and therefore are more likely to ask 'expensive' queries.

Since users are allowed to be imprecise, a problem arises when using a field which contains the same type of information in two different files. This is the 'restricted domain' problem mentioned earlier. And the mis-understanding may occur either with the system or with the user. Templeton told us that there are levels of sophistication in dealing with this problem, as with dealing with ambiguities. For example, a problem occurs with the query: "List the social security number of Jones," when the data file contains names of employees, former employees, and (say) consultants—all of whom have a social security number listed. Upon seeing the query answered, the user may incorrectly assume all of the names listed are current employees.

Since natural language systems seem to 'understand' synonyms, people often assume they also understand partial matches, and this is generally not the case. The system will not select the record for "Paul Smith" if the user asks for "P. Smith." The user must specify the person's name exactly as it is stored in the database for the record to be retrieved. In the worst case, the same name may be stored differently in different files. It is often recommended that users ask for all people named Smith, say, and then choose the desired record from the list, rather than expect the system to find the one correct record.

There are only a few truly free-form natural language query systems available today. As the field of artificial intelligence solves more of the problems with understanding natural language, more of these systems are likely to appear. Then machines, not humans, will perform the role of the interpreter.

Multi-tiered systems

One of the concerns about end user query systems is that they should be equally friendly to new users as well as to experienced users. Thus they should guide a novice user through the query process yet also allow an experienced user to rapidly perform his query. In addition, they should allow users to gradually learn about the capabilities of the system, so that users can advance as far and as fast as they wish.

One of the techniques being used to provide such diverse facilities is the 'multi-level' or 'multi-tiered' query system. With such a query front-end, users have their choice of how they want to interact with the computer. At the beginner level, the computer guides the user; at the advanced level, the user directs the computer. Generally, there are two major levels, the question-and-answer or menu level, and the user-guided command level. Following are three examples of multi-tiered query systems.

An end user budgeting system. Palme (Reference 7) describes a multi-tiered query system developed at the Swedish National Defense Research Institute. It was designed to be used by local and regional government employees for budgeting purposes. Users would access local databases using any one of several levels of the query language. The query language was designed to support beginner as well as advanced users.

A prototype of the query front-end was written by one programmer in two months time. It contained eight levels and was used by a few employees. Based on their experiences, a full production system was created, which contains the first five levels. The levels proceed from most basic to most advanced.

Level 1: Dialog via questions and menus. The first level is for users who have no experience with the system. These new users are led along by menus and questions, to which they need only select an item from the menu, or answer a question.

Level 2: Help facility. If a user does not understand what the computer expects him or her to do, he/she can type a question mark and the system will display a short explanation of possible responses. Palme explains that this simple feature teaches a new user two important lessons: (1) how to issue some commands to the system, and (2) how to interrupt the system whenever desired.

Also part of this level is the general help facility, with its own menu, which can be invoked at any time by typing two question marks (??). Thus a user can interrupt a conversation and browse through the help facility, and then be returned to the original dialog.

Level 3: Other user commands for interruption. Palme explains that there are times when a user wants to interrupt the dialog sequence for some other reason. Perhaps he wants a list of existing files because he has forgotten their names. Or perhaps he feels that the number being entered needs a special explanation. To meet these needs, the system provides such features as a 'footnote' command. When invoked, the system allows the user to perform this supplemental activity and then return to the primary activity.

Level 4: Parameters to commands. Building upon the previous two levels, in which users can issue commands to the system, this level allows users to add variables or delimiters to those commands. For example, the user may only want to see the list of files for certain departments. By adding these names, the user can limit the listing that is prepared. The system is designed so that novice users do not need to know about such delimiters until they want to; the system contains default values that automatically are used whenever a parameter is left off. At the same time, experienced users can speed up (and tailor) the processing by providing parameters.

Level 5: Command-driven user interface. Once a user enters the command mode (the most basic one being the help mode) he or she can stay in the command mode and not return to the computer-led mode. Experienced users can thus use the system entirely in the command-driven mode, if desired. In this mode, the user quickly proceeds through the same sequence of actions, by entering many commands in sequence on one line. If the system does not receive enough information to act, it asks for the missing information. If, for example, the user only enters "make budget," the system might ask "give department name." So the system automatically takes over when it needs more information.

These five levels of the front-end are in use by over 200 users in Sweden, and Palme reports that the front-end is used as it was designed to be used. Beginners use the computer-guided levels at first but then move to the user-guided command levels after a few weeks of experience.

The prototype system had three additional, more advanced levels. They were not implemented on the production system for cost and efficiency reasons, as we discuss below. These three levels would actually turn the query system into a high level programming language.

Level 6: Saving series of commands. Experienced users often find that they want to perform the same operations again and again. This level allowed a user to create a 'program' by listing these commands, and then give the program a name, which, when called, would execute the program.

Level 7: Parameters to user-defined commands. Taking the programming idea a bit further, the prototype system allowed users to create a series of commands and include a variable name, so that these programs could be used for variations of the same task. The system would automatically ask for the value of the variable each time the program was run.

Level 8: Model language. This level is an extension of the two previous levels. Using it, users could define their own commands to perform certain operations—that is, they could build subroutines and give each one a name. Facilities needed to implement this level include a text editor, an interpreter (but not a compiler), control statements, and special functions for creating menus and validating query input.

In reference 7, Palme discusses the programming problems encountered on this project. As users moved from one level to the next, the transition proved to be not very smooth. There was a sharp delineation between the menu-driven and the command-driven levels. Palme believes it is important to give users both facilities at once, because a user may want to be in the command mode for often-used features, but switch to the menu-driven mode for seldom-used features.

A number of problems were caused because the designers wanted to give users as much freedom as possible. For example, since the system allows users to interrupt an action to perform another action, the user could continually interrupt actions, causing many unfinished actions. And when the system tried to move backward through these, the user might not understand what it was doing. To avoid this 'unfriendliness,' the system returns to the original action only if a few supplemental actions are taken. If several

supplemental actions are performed, the system returns the user to the top level of the program.

Everything considered, though, Palme says the production system is successful, and is providing useful experience for future work.

An end user graphics system. A similar approach was used by Mozeico (Reference 8) at Tektronix to provide an end-user interface to a business graphics system, as mentioned in our Commentary last month. The system was designed for a desk-top system on which users could create line, bar, scatter, and pie charts.

The purpose of the interface was to allow users to progress easily from beginners to experienced users while employing the system to produce useful graphs. The designers added training aids wherever possible within the levels of the interface.

The original design of the system was based on five levels.

Level 1: Easy Graph mode. By pressing the function key labeled 'Easy Graph,' the user initiates a question-and-answer session for creating a graph. Most of the questions are multiple choice, and a default value is available for each. During the session, the system presents graphics on the screen to illustrate the choices, where possible.

An interesting feature of this level is the computer feedback or 'echoes' to the user's choices. For example, if a user chooses to give the x-axis of the chart the title "TIME IN SECONDS", the system responds with: "The Easy Graphing command corresponding to your previous response is X-TITLE 'TIME IN SECONDS'". Thus, users who want to learn the system's command language, can do so by studying these 'echoes.'

Level 2: Tutorial. By pressing the function key labeled 'Tutorial,' a user can read the on-line tutorial. It is divided into four chapters, with each 'page' generally presenting one new idea.

Level 3: Initial command level. Inexperienced users who want to begin to create graphs with options not available in the Easy Graph mode can begin to use the command language at this level. There are extensive error messages and a help facility at this level, initiated by pushing the key labeled 'Help.'

Level 4: Intermediate command level. At this level, the user is given added capabilities, such as being able to use all 15 function keys, add parameters to commands, and save data and graphs for future use.

Level 5: Advanced command level. The most advanced level provides the user with an editor, which he or she can use to create and modify sequences of commands. These sequences can be stored on a floppy disk and re-run later. The system leads the user through defining the needed variables and then performs the processing.

In the implementation, because of cost reasons, the entry points for levels 3, 4, and 5 were combined. Even so, Mozeico states that the system has been used by both inexperienced and advanced users, and that inexperienced users who used the Easy Graph facility have been able to advance to the command level. He notes that level 3—where the user first uses the command language—could have been made easier to use if it contained some menus.

Mozeico also points out that the major drawback of the multi-tiered query system is cost. He sees it really being most cost effective for systems where the users will have widely differing levels of experience and ability. But he feels that the teach-while-using method helps even the most intimidated users master the system.

Query systems on micros. In our research for this issue, we also looked into the database management systems available on *micro*-computers, hoping to find an example of how a company was using such a system for querying.

What we found, in general, was that the DBMS systems available today for micros were not being used for sophisticated querying. Perhaps the main reason for this is the rather small file sizes involved with floppy disk storage. But with the growing use of hard disks, with capacities in the tens of millions of characters, this situation probably will change.

In fact, our issue next month will discuss advances in relational database technology. One of the points that came up several times during our research for that issue was that relational DBMS will soon be appearing on some micro-computers that use the Motorola MC68000 processor. So sophisticated querying on files stored on micros

themselves will be a reality before too many more months.

Of course, today a micro-computer can be used as a front-end for a sophisticated querying system, as illustrated by the INFORMATICOM system discussed earlier. That system is used for entering and validating complex queries, that will later be processed on a host computer, and for the printing out of the results. In addition, it is used for storing and searching smaller or sensitive files locally, for emulating an IBM 3275 terminal, and for acting as a personal office computer.

Conclusion

With the increasing interest in end user systems comes the increased importance of the end user interface. Many of the query interfaces on the market today are not very well designed, because they require users to state their queries in a 'programming-like' language. For infrequent users, this type of interface is not very user-friendly. And, it increases the 'hidden' training costs of using the system. We see query systems becoming much more friendly, with the computer taking on more of the man-to-machine interpretation tasks.

REFERENCES

1. For more information on Intellect, contact Artificial Intelligence Corporation, 200 Fifth Ave., Waltham, Massachusetts 02254.
2. For more information on INFORMATICOM, contact Informatics General Corp., 21031 Ventura Blvd., Woodland Hills, California 91367.
3. Carbonell, J. G., "Natural language interfaces to software systems," *Office Automation Conference '82 Digest*, AFIPS Press (1815 North Lynn St., Suite 800, Arlington, Virginia 22209), March 1982, pp. 511-517; price $34.00.
4. Marcus, R. S. and J. F. Reintjes, "A translating computer interface for end-user operation of heterogeneous retrieval systems. I. Design," *Journal of the American Society for Information Science*, ASIS (Washington, D.C.), July 1981, pp. 287-303.
5. Harris, L.R., "Communicating with computers in natural languages—Future promises," Session No. 8.6 at the 1981 AFIPS National Computer Conference held in Chicago, Illinois May 4-7. There is no paper of this presentation in the Proceedings; however a cassette tape of the session can be ordered from On-the-Spot Duplicators, 7224 Valjean Ave., Van Nuys, California 91406. Price is $8.00.
6. Templeton, M. and I. Kameny, "Comparing the use of English, forms, and query language for data base usage," *20th Annual Technical Symposium of the Washington D.C. Chapter of the ACM*, Washington D.C. Chapter of ACM (P.O. Box 6228, Washington, D.C. 20015), June 1981, pp. 117-121; price $10.
7. Palme, J. *A human computer interface encouraging user growth*, Swedish National Defense Research Institute (Rapportcentralen, FOA 1, S-104 50 Stockholm 80, Sweden), Report No. C-10073-M3(E5,H9), April 1981, 20 pages; price $5.00, which does not include postage.
8. Mozeico, H. "A human/computer interface to accommodate user learning stages," *Communications of the ACM*, ACM (11 West 42nd St., New York, New York 10036), February 1982, pp. 100-104; price $6.00.

Prepared by:
Barbara C. McNurlin
Associate Editor

The trade press has been printing new product announcements about 'relational database management systems' for several years. But, upon investigation, it has turned out that most of these systems have had rather severe limitations, such as slow performance or file size constraints. Now, however, some relational DBMS *are on the market that perform well and handle reasonably large files; we think that they deserve your consideration, as we will discuss next month.*

COMMENTARY

CHASING THE PAPERLESS DOCUMENT

by Mary J. Culnan, School of Library and Information Studies, University of California, Berkeley

One major impact of office automation is the replacement of traditional paper files by electronic document storage. These electronic files include *personal files*, consisting of letters and memos, as well as *'corporate memory'*, consisting of reports, procedure manuals, correspondence files, and so on. In addition, there has been an explosive growth of *commercial time sharing database services*, which provide access both to full-text information and bibliographic citations on a wide range of topics. For example, Lockheed's Dialog Information Retrieval Service provides access to over 100 databases. These services are useful for strategic planners, lawyers, marketing researchers, and others who rely on external information.

For the end user of such systems, it may be difficult to know exactly what information is available in a particular file, or how to phrase a request so as to retrieve only wanted information. The main difficulty in querying these files, as opposed to data files, results from the differences between data and documents.

In the majority of typical data files, individual data elements are structured and standardized. Numeric codes are used to represent most entities, such as products or departments. A single spelling is used for names. The key used to uniquely identify a single record is usually part of the record, and this key is usually meaningful and likely to be known to the user. A group of related records may be retrieved by using one or more of the standard data elements.

Documents, however, consist of unstructured text rather than structured data. Most organizations create and use many different types of documents, each with different attributes or 'fields' which might be used for retrieval. For example, suppose a manager wants to retrieve all correspondence *from* a given customer. If a search is performed strictly on the text of the documents, the results may also include correspondence *to* or *about* the customer. Further, in creating documents, people use different words to express the same idea. Unless documents are indexed using standard subject terms, a user will have difficulty locating a group of related documents, because it is impossible to know what common words occur in *all* the relevant documents. Finally, there may be nothing in a document that uniquely identifies it; therefore, a key must be assigned. This key is likely to have no meaning to the user who, instead, may need to combine many descriptors, such as creation date, author, or subject in order to retrieve this single, known document. All of these factors can complicate life for the end user who is chasing the paperless document.

Currently, a number of options exist for querying in-house files of textual information. Most commercial office systems offer some form of retrieval ca-

pability, ranging from a 'file folder' approach to full text retrieval. Some systems allow the user to specify fields to be used for retrieval. The majority of these current systems are fairly primitive, and we should expect to see more advanced office systems as user organizations gain more experience with electronic document retrieval.

There are also a number of specialized micro-computer-based systems designed specifically for document retrieval. One is the PRIMATE System from the Institute for Scientific Information in Philadelphia, Pennsylvania. Another is the STAR System from Cuadra Associates in Santa Monica, California. While these systems perform well for specific applications, they may be incompatible with the hardware or software that an organization is using to perform other office-related tasks.

Commercial database services provide sophisticated query languages. These query languages are, however, likely to be perceived as 'unfriendly' by end users who use them infrequently. In order to facilitate access to these database services by the casual user, the Franklin Institute Research Laboratories in Philadelphia has developed the IIDA (Individualized Instruction for Data Access) System. IIDA serves as an 'intermediary' between the end user and the commercial service. The program only intervenes when the end user appears to be having difficulties, i.e. when it sees excessive syntax errors, thrashing, excessive null search results, unnecessary repetition of commands, and so on. IIDA can also be used as a training device for novice users.

While the differences between data and text outlined above may necessitate separate system development efforts, common impediments to the use of both types of systems merit a shared effort for information retrieval. Both text and data files continue to proliferate, making it difficult for a user to select the appropriate system or file which contains the desired information. Once the appropriate system is identified, a user may be faced with different sign-on procedures and query languages for each system. The development of a friendly interface between a user and a variety of systems will facilitate retrieval of both data and text.

In conclusion, while office automation may cause organizations to face the problems associated with the 'paperless chase' for the first time, many graduate library schools have more than a decade of experience with automated document retrieval systems. Major breakthroughs in office systems are unlikely, however, until theory is advanced by the experiences of user organizations. Companies now implementing office systems might contact the library school at a nearby university, to see if any of the faculty are interested in collaborating on a field research study at their organization.

SUBJECTS COVERED BY EDP ANALYZER IN PRIOR YEARS

1979 (Volume 17)

Number	*Coverage*
1. The Analysis of User Needs	H
2. The Production of Better Software	H
3. Program Design Techniques	H
4. How to Prepare for the Coming Changes	K
5. Computer Support for Managers	C,A,D
6. What Information Do Managers Need?	C,H
7. The Security of Managers' Information	L,C,A
8. Tools for Building an EIS	C
9. How to Use Advanced Technology	K,B,D
10. Programming Work-Stations	H,B
11. Stand-alone Programming Work-Stations	H,B
12. Progress Toward System Integrity	L,H

1980 (Volume 18)

Number	*Coverage*
1. Managing the Computer Workload	I
2. How Companies are Preparing for Change	K
3. Introducing Advanced Technology	K
4. Risk Assessment for Distributed Systems	L,E,A
5. An Update on Corporate EFT	M
6. In Your Future:Local Computer Networks	F,B
7. Quantitative Methods for Capacity Planning	I
8. Finding Qualified EDP Personnel	J
9. Various Paths to Electronic Mail	D,M
10. Tools for Building Distributed Systems	E,B,F
11. Educating Executives on New Technology	K
12. Get Ready for Managerial Work-Stations	C,A,B

1981 (Volume 19)

Number	*Coverage*
1. The Coming Impact of New Technology	K,A,B
2. Energy Management Systems	M
3. DBMS for Mini-Computers	G,B
4. The Challenge of "Increased Productivity"	J,K,A
5. "Programming" by End Users	C,H,B,G
6. Supporting End User Programming	C,H,B,K
7. A New View of Data Dictionaries	G,B
8. Easing the Software Maintenance Burden	H,B,G
9. Developing Systems by Prototyping	H,B,G
10. Application System Design Aids	H
11. A New Approach to Local Networks	F,K
12. Portable Software for Small Machines	B,H

1982 (Volume 20)

Number	*Coverage*
1. Practical Office Automation	A,B,C,K
2. Computer Graphics for Business	K,C,B
3. Interesting Decision Support Systems	C,B,H,A
4. Can Tele-communications Replace Travel?	A,F,J,L
5. The Human Side of Office Automation	A,J,K
6. Some Users Want Their Own Computers	B,C,K
7. Using Minis and Micros	B,E
8. Training for End Users	C,B,K,J
9. Query Systems for End Users	C,B

Coverage code:

A Office automation
B Using minis & micros
C Managerial uses of computers
D Computer message systems
E Distributed systems
F Data communications
G Data management and database
H Analysis, design, programming
I Computer operations
J Personnel
K Introducing new technology
L Security, privacy, integrity
M New application areas

(List of subjects prior to 1978 sent upon request)

Prices: For a one-year subscription, the U.S. price is $66. For Canada and Mexico, the price is $66 *in U.S. dollars,* for surface delivery, and $73 for air mail delivery. For all other countries, the price is $78, including AIR MAIL delivery.

Back issue prices: $7 per copy for the U.S., Canada, and Mexico; $8 per copy for all other coutries, sent via AIR MAIL.

Reduced prices are in effect for multiple copy subscriptions, multiple year subcriptions, and for larger quantities of a back issue. Write for details. Agency orders are limited to single copy subscriptions for one-, two-, and three-years only.

Please include payment with order. For U.S. subscribers, you can use your Visa or MasterCard charge card; include your card name, number, and card expiration date on your order.

For payments from outside the U.S., in order to obtain the above prices, take your choice of three options: (1) use an international money order, (2) pay in U.S. dollars with a check drawn on a bank in the U.S., or (3) use any of the following charge cards: Visa, MasterCard, Eurocard, Access Card, Standard Bank/Kaart, Union Card International, or Diamond Card International. Please be sure to include your card name, number, and card expiration date on your order.

Editorial: Richard G. Canning, Editor and Publisher; Barbara McNurlin, Associate Editor. While the contents of this report are based on the best information available to us, we cannot guarantee them.

Missing Issues: Please report the non-receipt of an issue within one month of normal receiving date; missing issues requested after this time will be supplied at the regular back-issue price.

Address: Canning Publications, Inc., 925 Anza Avenue, Vista, California 92083. Phone: (714) 724-3233, 724-5900.

Microfilm: EDP Analyzer is available in microform, from University Microfilms International, Dept. P.R., (1) 300 North Zeeb Road, Ann Arbor, Mich. 48106, or (2) 30-32 Mortimer Street, London WIN 7RA, U.K.