EDP ANALYZER

Editor and Publisher: RICHARD G. CANNING

EDP ANALYZER

OCTOBER, 1982
VOL. 20, NO. 10

RELATIONAL DATABASE SYSTEMS ARE HERE!

The reaction of some people to the title of this report will be, "I thought that relational systems had been around for several years." The reaction of others will be, "So what." But, while there have been relational systems on the market for some time, there have been many questions and doubts about their limitations and performance. Now, however, some relational systems are available that deserve your serious consideration. Relational systems are joining hierarchical and network systems as part of the database technology. (See page 16 for an executive summary of this report.)

Texas Eastern Corporation is a diversified energy company with headquarters in Houston, Texas. Operating revenues in 1981 were $4.5 billion.

Texas Eastern's data processing environment includes a Honeywell triple processor mainframe, three DEC VAX 11/780 super minis, and a number of distributed mini-computers in user areas. Business applications are programmed predominately in COBOL. The company also makes extensive use of other programming languages for creating economic forecasting and planning models.

In 1980, a study was conducted to evaluate and select a database management system (DBMS) on which to build a management information system to serve the company's exploration and production activities. This study focused on four major capabilities that were desired of the DBMS. These were: (1) data structure independence from program logic, (2) end user facilities, (3) ease of restructuring the data, and (4) adequate security features. CODASYL, hierarchical, and relational systems were evaluated according to these criteria.

The ORACLE relational DBMS, from Relational Software, Inc. of Menlo Park, California, was selected and plans were made to install it, along with Texas Eastern's first DEC VAX 11/780, in mid-1981. Since that time, Texas Eastern has acquired two additional VAX 11/780's and is running ORACLE on each of them.

ISSN 0012-7523.

For its management information system to support exploration and production, Texas Eastern purchased a customized system developed by an outside firm, and has converted the system from a HARRIS/TOTAL environment to a VAX/ORACLE environment. While this was their first system to run under ORACLE, the installation has been a lengthy process due to the complexity of the system. The staged installation of this system is still underway.

While the installation of this system was in progress, the computer services department designed an ORACLE application for monitoring project activity in the development of new systems. This management reporting system, called MARS, has been in use since April of this year. It presently provides department management with information on current and anticipated system development projects, and will eventually be expanded to include the full spectrum of functions to support computer systems project management. Texas Eastern feels that the development of MARS was accomplished significantly faster by using ORACLE than would have been possible with conventional programming languages and database technology.

Several other medium size business applications are in progress for the VAX/ORACLE computers. So far, response time for accessing the databases has not been a problem. In addition, while the ORACLE release in use at Texas Eastern does not yet have all the features that the company desires, they feel confident that Relational Software will be able to deliver those features in future releases.

All in all, Texas Eastern is well pleased with their move to this new database technology. They feel that relational database approaches, along with end user tools such as the SQL language and Query by Example, will find an important place in database technology for the 1980s.

TRW Defense Systems Group

The Defense Systems Group (DSG) of TRW Inc., located in Redondo Beach, California, specializes in electronics systems for defense and space. TRW itself, with headquarters in Cleveland, Ohio, had 1981 sales of over $5 billion and employed some 92,000 people.

DSG uses a wide variety of mainframe, mini, and micro-computers, both for in-house use and in systems that are being developed for customers. These include computers made by IBM, CDC, Burroughs, DEC, Hewlett-Packard, Prime, and Perkin-Elmer. We discussed the Group's use of a local network for connecting some of these computers together in our June 1980 issue.

In support of the information processing systems that they develop for customers—typically, defense and space systems—the people at DSG have developed a man-machine interface dialog design tool which they call FLAIR. FLAIR allows a designer to quickly create a prototype of a new system's user interface, in the early stages of a project. The prototype is developed to determine the client's needs and desires before any production coding commences.

FLAIR has been in operation since early 1981, and is being continually enhanced. (It is not for sale to other organizations, however.)

Wong and Reid (Reference 1) describe the major components of FLAIR and how it is used. A prototype developed with FLAIR allows client personnel to interact with a portion of the 'deliverable system,' via scenario simulation. The system designer uses FLAIR to create the prototype; client personnel then test out the use of the prototype and (generally) suggest changes. This activity leads to a more accurate definition of the system requirements.

For this type of use, a 'must' for FLAIR was that it be easy for system designers to use, say Wong and Reid. It itself should be a good example of an effective man-machine interface.

A point of interest: FLAIR provides for a variety of types of input to be used by the prototype systems—including digitizing graph tablets, light pens, joy sticks, keyboards, voice recognition, and others. Considerable use is made of voice recognition, we were told. It is not unusual for users to give commands to the system—say, for the display of particular graphics or maps—by voice. These commands select appropriate paths in a tree-of-action items. A graph tablet or light pen controls the cursor and a keyboard is used for entering character strings.

Wong and Reid give an example of how a user might give voice and light pen commands to a prototype system to draw a circle, using FLAIR functions: (voice) DRAW; (voice) CIRCLE; (point with light pen to the desired location for the center point); (voice) CENTER; (point with light pen to the desired location for the radius); (voice) RADIUS. And that's it!

Another approach for system input also supported by FLAIR is to use a graph tablet or light pen to select a high level instruction from a menu and then to pick instructions from submenus.

With this kind of flexible and powerful prototyping system available, the people at DSG saw the need for providing FLAIR with database management facilities. They considered the leading types of DBMS, such as hierarchical, network, secondary index, and relational. They decided that relational database management was the 'wave of the future' and that products now on the market offered a practical solution for database applications. So they chose two products for use with FLAIR—INGRES, from Relational Technology, Inc. of Berkeley, California, and the IDM 500 database machine from Britton Lee Inc. of Los Gatos, California. Relational database management, they felt, offered them more flexibility in adding, deleting, or changing data fields and records and more flexibility in making use of relations among data items. And in defense and space systems, such unforeseen change seems to be the norm, not the exception.

INGRES is a software product that runs on DEC VAX machines. It includes on-line features (such as the QUEL query language and end user utilities) and the EQUEL language for embedding INGRES commands in programs written in other languages. It will be described in somewhat more detail later in this report.

The Britton Lee IDM (Intelligent Database Machine) is a backend machine which off-loads the entire relational database job from the host computer. It can interface to any of a variety of mainframes, minis, or micros, but interfacing software is needed for each type of machine. The IDM, too, will be described in more detail shortly.

INGRES is not seen by the system designers who use FLAIR. It is buried deep within the system, we were told. But it does provide these designers with an easy and friendly way to define the data, store that data, and then retrieve it in almost any manner that the system designer (or client) desires. FLAIR translates the user's commands into INGRES query and retrieval commands. For the prototype systems developed with FLAIR, files tend to be small in size, and do not tax the capacity of INGRES. The actual production systems can have much larger databases, and the choice of the DBMS to be used depends on numerous factors, including the particular computer that is to be used.

For client systems where very large databases will occur and where relational database management may be desired, DSG obtained the IDM 500. FLAIR might have to handle databases hundreds of millions of characters in size, and the IDM 500 probably would be used for this, we were told. But, in addition, the people at DSG want to get experience with backend database machines, looking forward to the day when they will have to provide database management for files measured in the billions (even hundreds of billions) of characters in size.

In general, the people at DSG see relational database technology as playing an increasingly important role. It is the most flexible database management system, they feel, and is the one best able to handle unforeseen changes in requirements. It is good at handling complex, dynamic data that has many inter-relationships. Also, they see software systems, such as INGRES, and the backend hardware/software systems, such as the IDM 500, as being more complementary than competitive. Each has advantages in certain situations, depending on the host computer and the database size.

Also, they see database management becoming hybrid in nature, in two different ways. One, database management must be able to handle data of a variety of basic types—formatted data, unformatted text, digitized maps and pictures, matrices, and so on. Each type of data will require its own type of database management. Second, within the category of formatted data, there are times when a hierarchical or network struc-

ture is desired, as opposed to the 'flat files' of relational systems. So DBMS of the future probably will have to accommodate all three of these structures.

As they say at TRW's Defense Systems Group, relational database management appears to be the way of the future. So that is why they have incorporated it in FLAIR, their man-machine interface test bed.

Developments in DBMS

Relational database management systems have been evolving since the relational data model was first proposed by Dr. Edgar F. Codd of IBM in 1970. Perhaps more than any other approach to database management, they have triggered a debate on which data model is conceptually 'best.' A data model is the method by which data is structured, to represent the real world, and the way that data is accessed.

There are three major types of data models in use today. The *hierarchical* model structures data in parent and child relationships—components of an organization, for example. In this approach, a data item can have only one parent. It is represented by IBM's IMS and Intel's System 2000. Another major type is the *network* structure, where each data item can have more than one parent. Manufacturing assembly parts lists illustrate this structure, where the same part can be used in more than one assembly—and, in fact, might be used in different quantities in each such assembly. This approach is represented by the CODASYL-type database management systems, such as Cullinane's IDMS.

In both the hierarchical and network models, the data relationships are explicitly stated, generally by pointers stored with the data. These pointers provide the means by which the user's program accesses the desired data records.

The third approach is the *relational* model, where the relationships among data items are *not* expressly stated by pointers. Instead, it is up to the DBMS to find the related items, based on the values of specified data fields. Thus, all the employees of a certain department are found by the department number in the employee records.

There are some other popular DBMS that do not fit neatly into these three categories. Cincom's TOTAL, for instance, comes close to being in the network category. Software ag's ADABAS is a secondary index system that has some similarities to both hierarchical and relational systems. Some other systems are 'almost' relational. And still others claim to support two types of data models—say, network and relational.

But what is the relational model? And how does it differ from these other models, as far as users are concerned?

To set the stage for answering these questions, it will help if we first discuss how DBMS in general are evolving, by way of the three-level database concept.

Three-level databases

One of the easiest-to-understand discussions of today's database technology that we have seen is by James Bradley (Reference 2). He describes the work of the Standards, Planning and Requirements Committee of the American National Standards Institute (ANSI/SPARC) in the mid-1970s. The main concept that came out of the work of this committee was the *three-level* database, he says—and it was the research on relational database technology that led to this concept. Interestingly, one of the prime movers on this committee was Charles Bachman, whose ideas were so influential in developing the network approach.

It should be noted, as Bradley says, that most of today's major DBMS have either adopted this three-level concept or are close to it. So the ideas of ANSI/SPARC have had a significant influence already.

Level 1 is the storage level, says Bradley; this is the way the data is physically stored on (say) disk. A data record consists of its data fields plus some implementation data—generally, pointers and flag fields. The problem of overflow of record storage areas occurs for almost any method of direct access, so a flag is needed to identify a record as being an overflow one and a pointer is needed for pointing to the next overflow record. The DBMS simply follows this chain of pointers until it finds the desired record. The end user, of course, need not be concerned with these pointers and flags; they are for use by the DBMS only.

Level 2 is the conceptual level. The data fields are the same as at Level 1, but without the implementation data (flag and pointer fields). The data records at this level, then, are logical records.

Level 3 is the external level, says Bradley—and this data is what an application program sees. Level 3 records contain only selected fields from the Level 2 records.

To illustrate the principles involved, a simple Level 1 data record might have fields A, B, and C, plus a flag field and a pointer field. At Level 2, the flag and pointer would be stripped off, leaving only the A, B and C fields. Then, at Level 3, one application program might be given only fields A and B, while another might be given only A and C.

There are numerous advantages to this three-level concept. The organization of Level 1 data can be changed—say, from calculated addresses ('hashing') to index sequential access—with no effect on Levels 2 and 3. Or a new field can be added to a Level 1 record. While Level 2 also must handle this new field, Level 3 records that do not use this new field will have no change at all.

(Traditionally, says Bradley, to add a new field requires changing the existing application programs that use that file. What happens instead is that a new file is set up that incorporates the new field. So data files proliferate.)

Now to return to the questions: What is the relational model, and how does it differ from the hierarchical and network models?

What are relational systems?

The first point to make is that the relational model fits this three level concept. The hierarchical and network models may or may not fit, depending on how they are implemented. Also the relational model stores and accesses data much differently from the other two models.

Codd, in his 1981 ACM Turing Award paper (Reference 3), discusses his ideas of what constitutes a true and fully relational system. In brief, and using data processing terminology, he says such a system must have (a) a structural part consisting of 'flat' tables, where the columns represent fields and the rows represent records, but there is no notion of one column succeeding another or of one row succeeding another, (b) a collection of operators (such as select, join, etc.) for retrieving, deriving or modifying data in those structures, and (c) a collection of general integrity rules. So a relational system is more than a collection of flat tables.

While there are numerous features that differentiate a 'true' relational model from a network or hierarchical model, we will single out three: (1) data structures, (2) operators (or commands), and (3) user views.

Data structures. In a relational model, data is represented in the form of flat tables—where 'flat' means that there are no repeating groups. As mentioned, the rows of a table represent the different records in the file, and the columns represent the different data fields in those records. Each record has a fixed length and a fixed format. In addition, there should be no duplicate records. These conditions hold true for the Level 1 stored data, Level 2 logical data, and Level 3 user data.

(Putting the data into this form is called 'normalizing' it. Codd proposed three forms of normalizing; this has since been expanded to five forms. A discussion of normalization, while important, is beyond the scope of this paper.)

To compare the relational model with (say) the CODASYL network model, Bradley says (in Reference 2) that CODASYL records can be formed in two distinct ways: (a) as variable length records, with repeating groups, using the COBOL OCCURS clause, or (b) as owner-member sets of fixed length, fixed format records. Using (b), one obtains records that are similar to those in relational systems—and this is the best way to design *any* database, he says.

So relational systems *require* files of flat records, and records which are not inter-related by pointers. Network and hierarchical systems can have flat records, if set up that way, but such records are not required. Moreover, the latter two models generally designate data relationships by pointers.

Operators. A relational model is *more* than this simple data structure. Another feature of relational systems is the existence of powerful operators (commands) by which the user tells *what*

the system is to do, not *how* it should be done. With relational systems, the user does not have to 'navigate' through a database, by following pointers, seeking the desired information.

All that the user deals with are tables. The results of operations on tables are themselves tables. There is only one set of operators in a relational system, as opposed to other types of DBMS where one set of operators is used to manipulate the data and another set is used to manipulate the relationships. In a relational system, changing the data in a table changes the relationships.

Furthermore, said Codd in a discussion with us, the operators should at least provide the facilities of selecting specified columns of tables, of selecting specified rows of tables, and *joining* the rows of one table to the rows of another, where specified criteria are met.

Ferris believes that this 'join' function is perhaps the key aspect of relational systems. For instance, see his comments in References 5a and 5c.

It is the join that allows a Level 3 user table to be created by selecting and joining specified columns and rows *from two or more Level 2 tables.* Many of today's DBMS can create Level 3 *records* by selecting specified fields from Level 2 records. Relational systems go beyond this; they can create Level 3 user *tables* by both *selecting and joining* data from multiple tables.

While there are some join-like features in other types of DBMS, we have not heard of a true join function in a non-relational system.

User views. In Reference 4, Codd makes the point that a relational system should provide user view tables, which are made up of selected rows and columns that meet specified criteria, from one or more other tables. For example, a manager might be given a user view that allows him/her to see certain data fields in the records of employees under him/her—but not all the data on those employees, and none of the data on other employees. This type of facility is useful in many ways, including data security.

Thus Level 3 tables represent 'user views.' As indicated, no other type of DBMS yet equals relational systems in providing this function.

At the moment, relational systems are 'in,' in the sense that numerous suppliers are claiming that their systems are relational, for sales purposes. This discussion has touched only on some highlight aspects of relational systems. Answering the question of whether a system is truly relational or not is a bit tricky and is beyond the scope of this report. Readers interested in reading more on this question are referred to References 3, 4, and (for instance) 5a, 5b and 5c.

Interacting with a system

While there are differences among relational systems in the way the user interacts with them, there are also striking similarities. This is due, no doubt, to the extensive definition of a relational system that Codd gave in his 1970 paper. He provided the model from which the subsequent systems have been developed.

IBM's SQL illustrates the features of a relational language. An SQL retrieval command has the following three main components: (1) the verb SELECT, followed by the name(s) of the field(s) to be selected; (2) the qualification, indicating which tables are involved; and (3) the criteria to be used in the selection (the WHERE clause).

If it is desired to obtain the name of the employee whose number is 12345, from the employee file, the entry would be: SELECT ENAME FROM EMP WHERE EMPNO = 12345. (Actually, with SQL, the select, from, and where clauses would be on separate lines, and the entry would be ended with a semicolon.)

A join query is a bit (but not much) more complex. Assume a listing is desired of all employees over age 60 and the department and floor on which they work. Assume also that the desired data is to be obtained from two files, the employee file and the department file. The entry would be: SELECT ENAME,DEPTNO,FLOOR FROM EMP,DEPT WHERE EMPAGE 60 AND EMP.DEPTNO = DEPT.DEPTNO.

The next step up the ladder is a nested query. An example might be to list all the employees who have the same job title as Jones. The entry would be: SELECT ENAME FROM EMP WHERE TITLE = SELECT TITLE FROM EMP WHERE ENAME = 'JONES'. In this example, the initial WHERE clause involves a retrieval; Jones' job title must first be obtained, after which it is used for re-

trieving all other employees with the same job title.

A user view can be defined in much the same way. The format is: DEFINE VIEW (view name) SELECT (fields to be incorporated in the view) FROM (tables from which these fields are to be obtained) WHERE (criteria for making the selection).

There is much more that could be described, but these examples should give some idea of what user interaction with a relational DBMS is like. For a somewhat longer discussion of the command structure of three leading systems (SQL/DS, ORACLE, INGRES), see Dieckmann (Reference 6).

A structured query language such as SQL is not too complex, from an end user's point of view, but it is still something that executives, managers, and professional staff members will have to learn. Nested queries are likely to cause the most problems for these users.

But, you might ask, cannot something closer to natural language be used, to make a relational system even easier for these types of end users? We discussed the subject of natural language query systems last month. Some natural query languages are available—but they have their problems, too. It is not too likely that you will soon see a natural language query system for relational databases.

Relational data management systems

Besides data*base* management, today's relational systems are offering many more features that, together, we have been calling 'data management.' These features include the ability to create input screen formats and procedures, report writing, a query language, word processing, interface to high level programming languages, and such.

In many of our 1981 reports, we discussed how these data management systems are being used—by programmers and by end users—to get new applications up and running rapidly. We ourselves have been using one such system for several years and can attest that the concept is not just theory; it really works.

And we must say, we were well impressed with the way these relational systems performed the data management functions. From what we have observed, it is an easy matter to state most of the queries in the relational language. It is easy, too, to define user views, whereby a user is given access only to specified fields of specified records. In short, the user interface of these systems is indeed friendly.

So we foresee these relational systems advancing the state of the art in end user programming, developing new applications by prototyping, and the other such uses that we discussed last year.

How will they be used?

What is the likely role of relational systems? Will they replace existing DBMS or not? Will they be used primarily for handling end user queries, or will they also be used in high transaction volume applications with large files? To what extent will they be used to provide higher productivity in the development of new applications (which was the main theme of Codd's Turing Award paper, Reference 3)?

For some partial answers to these questions, consider the marketing strategies (as we interpret them) of four leading suppliers of relational systems.

IBM SQL/DS. IBM's first relational product was announced in January 1981, and deliveries began during the first quarter of this year. However, IBM had gained extensive experience with a prototype system, System R, that was tested by several IBM customers over a period of time.

We will touch briefly on what IBM has announced about the marketing of SQL/DS (which means 'Structured Query Language/Data System' and where the SQL is usually pronounced "Sequel").

First, SQL/DS has been released so far only for IBM's intermediate range of computers operating under DOS/VSE.

Secondly, IBM says they see SQL/DS as complementing, not competing with, IMS and DL/1. For instance, an extract function is available to run under DL/1, to extract desired data from production databases. The extracted data is formatted for previously defined SQL/DS tables and can then be manipulated by SQL/DS. So this use seems to be mainly for management information purposes.

It is possible to update both DL/1 and SQL/DS databases in parallel, with the provision that if one of these updates fails to complete, the other is cancelled.

So IBM, in their marketing strategy, does not see SQL/DS competing with or displacing existing DBMS such as IMS or DL/1. Nor do they seem to see it being used in high volume production work. Instead, they appear to expect it to be used for management information purposes and for handling smaller, lower activity databases.

For more information on SQL/DS, see Reference 4, or your local IBM office.

ORACLE, offered by Relational Software, Inc., has been designed to be 'plug compatible' with IBM's SQL/DS. IBM published the SQL data language and RSI has adopted that language. So ORACLE, too, will run on IBM's intermediate range of computers under DOS/VSE.

But, in addition, ORACLE will run on DEC PDP-11/23 (and above) computers, under the RSX-11M, IAS or UNIX operating systems. It also runs on all models of the DEC VAX computer under the VMS or UNIX operating systems.

The portability of ORACLE does not stop there, however. Version 3, which is being released about the time we are going to press, has been completely written in the "C" programming language. Hence, it can run on any 16-bit (or higher) computer with a C compiler and at least 256k bytes of main memory. RSI markets a C compiler for the IBM 370, 4300 and 30XX computers, to run under most of IBM's operating systems. As we go to press, RSI has shipped their first IBM production versions of ORACLE, to run on 4300 and 30XX computers under VM/CMS.

Also, ORACLE will run on micro-computers that use the Motorola MC68000 processor. Over 40 suppliers of this type of micro have talked with RSI about putting ORACLE on their computers, and at least one instance of this is expected to be on the market before the end of this year.

RSI foresees some instances of ORACLE being used in high transaction volume environments, with large data files. The system uses re-entrant code, so that multiple concurrent batch and online updates and queries are possible. Thus ORACLE might well compete with, and possibly replace, existing DBMS, if the customer so desires.

ORACLE provides a 'cluster' feature whereby two tables can be combined in a nested manner, such as a department record that is followed by the records of the employees in that department, followed by the next department record and then its employee records, and so on. Where this type of structure can help speed processing, ORACLE can provide that structure. While making the processing of transactions faster, such a step probably would make the processing of some ad hoc queries slower. The query user need not be aware of this cluster structure, however.

RSI also has developed ORATOR, which includes (a) an interactive application development facility, (b) a report writer, (c) an interactive query facility, (d) a text editor with word processing capabilities, and (e) a high level programming language.

So RSI sees ORACLE and ORATOR being used for new database applications, as well as sometimes replacing existing DBMS applications, on mainframe, mini, and even on micro-computers. These systems will be used both for management information purposes and for production updating of files. They will be used for developing new applications rapidly. Also, if a customer wants to start out with ORACLE and later change over to IBM's SQL/DS, no changes should be needed in the customer's application programs.

For more information on ORACLE and ORATOR, see Reference 7.

INGRES, offered by Relational Technology, Inc., is a commercial product that emerged from Project INGRES at the University of California, Berkeley. The project, which was begun in 1974, developed INGRES to run on the DEC PDP-11 under the UNIX operating system. Since it was developed with public funds, the University's INGRES has been available at low cost, and some 125 sites are using it. But the University could not justify enhancing and supporting it in the manner that users desired.

RTI was formed in 1980, and staffed with people from the university project and from industry, to extend and support INGRES as a product. For the first environment, the company chose to have their version of INGRES run on DEC VAX

computers operating under the VMS operating system.

Most customers are using INGRES for new applications, so that conversions from existing DBMS are not involved. RTI's emphasis has been on providing an application development system that uses INGRES, aimed at greatly reducing the number of lines of code that programmers must write to get applications up and running.

To this end, INGRES includes a utility for creating screens, a report writer, a forms-based query system (Query by Forms, available for 140 models of terminals), a text editor with some word processing capabilities, and an embedded query language with interfaces for several programming languages—C, FORTRAN, PASCAL, COBOL, BASIC, and, if desired, APL and PL/1. EQUEL, the embedded query language which is used in programs written in one of these languages, employs the same commands as INGRES' interactive QUEL query language. A pre-processor searches for these EQUEL commands and converts them into appropriate CALL statements for the programming language being used. So programmers need only learn one way to use INGRES.

With these utilities available, the people at RTI feel that some users may choose to convert existing DBMS applications; the new versions of the applications might well be developed so rapidly with these utilities that the old code will just be discarded. One reason they might choose to do this is to get the advantages that INGRES can offer them. For instance, one advantage is in the area of data security; end users can be granted access for update or for just retrieval, at the individual field level, for a specific terminal number, for specific times of day, and for specific commands that the user wishes to execute.

Also scheduled for release this year are two other extensions—a graphics capability whereby users can get graphics output with no programming needed, and a forms-based application development system.

For more information on INGRES, see Reference 8.

Intelligent Database Machine (IDM), from Britton Lee, Inc., is a back-end processor and storage system that has a complete database management system which uses the relational data model. The company was formed in 1978 to develop an intelligent database controller that would perform high speed searching of a database. Soon this goal was changed to that of a intelligent database machine that included a relational DBMS.

The IDM consists of: (a) an optimized high speed disk controller that overlaps the seeks and writes, and (b) a uniquely designed relational database processor. Also under development is (c) a parallel accelerator function that will allow high speed searching at disk data transfer rates. So the IDM is a specialized processor, designed to perform its functions at high speed—up to ten times faster than a general purpose back-end computer, running a regular DBMS software package, could perform the same functions.

Why use a back-end processor of this type? Britton Lee sees several reasons. Among them is the potential of higher speed. Another is the ability to take a significant workload off the host and free a good amount of storage space in the host's main memory. Not only is the DBMS code removed from the host but also a large amount of memory is made available that otherwise would be needed for holding the tables that have been retrieved and are being worked on. Another advantage is that this approach can provide a centralized database, as the IDM is designed to simultaneously support a variety of host computers. Still another is that it can provide a high level interface for the hosts—some of which can even be micro-computers or intelligent terminals.

(In fact, we witnessed an example of this, where a Z80-type micro, with 64k bytes of memory and running CP/M, entered queries and received responses from the IDM.)

The relational database system used in the IDM has similarities to both SQL/DS and to INGRES. Some of the key people came from the INGRES project at the University of California, and they also took advantage of IBM's published material on System R, the predecessor of SQL/DS.

Britton Lee offers four models of the IDM—the 200, 300, 500, and 600. For instance, the 200, with 1/2 megabyte of memory, can process in the order of 10 transactions per second; with 1 megabyte, the rate goes to roughly 16 per second.

And the 500 can have up to 5.5 megabytes of memory, and is aimed at the super-mini market. On-line disk storage capacities can be up to 2.5 billion bytes on the 200 and 11 billion bytes on the 500.

The 200 and 500 IDMs perform only some of the functions involved; each host must perform others. For instance, a host must translate a query into the IDM's internal form; it could be a query expressed in Britton Lee's IDL language, or IBM's SQL, or RTI's QUEL, or other. The host must provide the report writing functions, data entry functions, pre-compiling, database administration utilities, and drivers. The IDM, on the other hand, performs the database management, optimizes the access path selection, provides concurrency control, transaction management, security, audit logs, crash recovery, and the dumping and loading of data.

The 300 and 600 currently interface with the DEC VAX 11/750 and 11/780 machines. They differ from the 200 and 500 in that they perform the above-listed host functions.

Britton Lee sees their prime market as other computer system suppliers (Original Equipment Manufacturers, OEM's) who want to offer relational database management. Some customizing is needed to tailor the IDM to the particular computer(s) it is to work with, and Britton Lee expects that the OEM's can provide much of this customizing. Software is needed for communicating with the end user, for translating the user's commands to IDM internal form, for sending the translated commands to the IDM, for receiving the results back from the IDM, and for formatting those results and displaying them to the user. At present, Britton Lee has interfaces for DEC PDP-11 and VAX computers running under the VMS and UNIX operating systems, and is working on interfaces for other makes and models.

For more information on the IDM, see Reference 9.

How will they be used?. These are only four of the numerous relational systems that are on the market. Others include Hewlett-Packard's RELATE, GTE'S RELSTORE, and Logica's RAPPORT. Then there are some on the market that are *almost* relational; for instance, Reference 5d lists over 100 DBMS, of all types and sizes. In their advertising, many of these companies state that their products are relational. As mentioned earlier, it is not always an easy matter to determine if a DBMS is really a relational one or not.

But how will the relational systems be used? Based on the marketing strategies of the four firms just discussed, it seems most likely that relational systems will soon be used for: (a) new applications that can benefit from this technology, and (b) using data extracted from production databases, for answering management queries and producing management reports, particularly of an ad hoc nature. It is not likely that they will be used soon to replace many existing DBMS applications.

What about performance? Won't relational systems always be inherently slower than hierarchical or network systems, and hence limited to smaller transaction volumes and smaller databases? That is the next question to address.

The performance question

One point has been stressed over and over again by suppliers of relational systems—both in articles in the trade press and in conversations. That is, there are no theoretical reasons for relational systems to have poorer performance than the hierarchical or network systems.

Today's relational systems use conventional methods for accessing data—calculating addresses for direct access, the use of index sequential organization, the use of multiple secondary indexes, and so on. They do not require associative memories.

But differences in performance *do* exist. Some of today's hierarchical and network systems can handle more transactions per second than today's relational systems. Why is that?

Dr. Michael Stonebraker, of Relational Technology, Inc. and the University of California, Berkeley, offers the following explanation. "The early versions of any complex software system are almost always slow; this has been true of relational systems," he says. "The developers have to learn how to tune them, to improve performance. Some tuning techniques used with current DBMS technology can probably be adapted for tuning relational systems. Other tuning tech-

niques will have to be developed, such as moving some of the overhead from execution time to compile time."

Dr. Robert Epstein of Britton Lee, Inc. agrees. "Whenever you get general and complete, you get a lot of overhead. The relational model has a lot of overhead. It hasn't been around long, so not much tuning has yet been done on it. But there is nothing inherently less efficient about it."

Lawrence Ellison, president of Relational Software Inc., says, "It is the join operation that has been the major slow-down function in relational systems. As developers learn to overcome this problem (and we think our 'clustering' feature is a solution), there is no reason why relational systems cannot be as fast as network or hierarchical ones."

But Stonebraker cautions, "Considering that current DBMS have a number of years head start over relational systems, and are continually being tuned, it is not clear that relational systems will ever catch them on performance. But clearly relational systems will both catch and exceed *today's* performance of the current DBMS."

In support of his views, Stonebraker offered the following statistics on the performance of INGRES. Relational Technology's version 1.2 of INGRES ran five times faster than the University of California's version of the system—due, to a good extent, to the University project's concern more with function than with performance. Then RTI's version 1.3 ran 35% faster than version 1.2. The latest version, 2.0 (which has just been released), runs 50% faster than version 1.3.

"Furthermore, if content addressable disk storage is developed, a relational system can exploit such a development, while a DBMS technology that is based on pointers probably cannot," continued Stonebraker. In that case, relational systems might become *faster* than hierarchical or network systems.

What about performance? The upshot seems to be that today's relational technology can handle quite large databases (hundreds of millions of bytes) and transaction rates of about 10 to 15 transactions per second, for simple transactions (with the higher rates possibly requiring a database machine such as the Britton Lee IDM, under today's technology). With large databases, intelligent database administration is essential; if a relational DBMS defaults to a sequential search of the database, a large amount of computer resources can be wasted.

But most of the suppliers that we talked with say that initial uses of relational DBMS are likely to be for new, smaller applications. In such an environment, they feel that their products are competitive with the hierarchical and network systems.

Relational systems on micros

Why put a relational DBMS on a micro-computer? Both RSI and RTI told us that they had been visited by more than 40 suppliers who have MC68000-based systems under development. Both expected to see at least one such computer with a relational DBMS on the market by the end of this year.

(Note that there are some DBMS on the market for 8-bit micros that are claimed to be relational. However, they are not in the same league as the systems we are discussing here, in functions performed or the size of the database handled.)

Won't the price of a complete relational system be too high for use on a micro? The answer appears to be No. For mainframes and minis, the prices of the relational systems discussed in this report range from about $30,000 to over $100,000. While the price of a complete relational DBMS for a single-user micro has not been announced, it is likely to be under $1,000, and perhaps in the $500 range, we were told. Of course, a mini or a mainframe can serve multiple users, while each single-user micro would be expected to have its own purchased package.

But even if it is affordable, why do it? The reason seems to be that users will want to extract data from production files, load it on to their micro-computer work stations, and then 'play' with the data, seeking answers to problems. Also, a relational system would make it convenient to 'join' such internal data with data obtained from outside sources, to do market forecasting, sales forecasting, share-of-market analyses, production forecasting, financial analyses, budgeting, and so on. Ellison of RSI believes

that relational systems will be favored for such applications because they are so simple to use.

Why not perform these analyses on minis or mainframes; why use micros? The answer here seems to be, we gather, that many executives, managers, and professional staff members probably will feel more comfortable using their own work stations instead of using terminals tied to larger computers. Many of their applications will involve small files and will be used infrequently; it will be convenient to store such programs and data on floppy disk, to be saved offline until they are needed. Many small files would tend to clutter up the directories of the mainframes and minis. Also, data security can be higher when the data is under the complete control of the end user—since mini and mainframe operating systems can be penetrated.

So do not be surprised if you see complete relational systems offered on micro-computers, particularly those that employ the MC68000. And don't be surprised if executives, managers, and professional staff members prefer to use this type of work station for their problem solving, instead of using terminals tied to larger computers.

Conclusion

This report has dealt with relational systems almost to the exclusion of other types of database management systems. One reason for this is that we have discussed the use of these other database technologies in numerous past issues. Another reason is that we think these relational systems have finally reached the point where they can take their place alongside these other types of DBMS.

But we do not wish to give the impression that we now think the relational technology will displace the hierarchical, network, and secondary index technologies. As far as can be determined now, that will not happen.

Practical relational DBMS are here. They have many interesting features to offer. We believe that they deserve your serious consideration, particularly for new applications that involve medium-size databases.

REFERENCES

1. Wong, P.C.S. and E. R. Reid, "FLAIR—User Interface Dialog Design Tool," *ACM SIGGRAPH '82 Proceedings*, Association for Computing Machinery (ACM Order Dept., P.O. Box 64145, Baltimore, Maryland 21264); Order No. 428820; price $32.
2. Bradley, James, "The Elusive Relation," *Computerworld* (375 Cochituate Road, Framingham, Mass. 01701), March 8, 1982; In Depth section, pages 1-16; price $1.25. (This material was based largely on the author's book *File and Data Base Techniques*, Holt, Rinehart & Winston, 1982, and partly from his book *Introduction to Data Base Management in Business*, to appear early next year.)
3. Codd, E. F., "Relational Database: A Practical Foundation for Productivity," *Communications of the ACM* (ACM Order Dept., address above); February 1982; p. 109-117; price $6.
4. Codd, E.F., "SQL/DS, What It Means," *Computerworld* (address above), February 23, 1981; In Depth section, pages 27-30; price $1.25.
5. *Software News* (5 Kane Industrial Drive, Hudson, Mass. 01749); single issue price, $3.00:
   a) Ferris, David, "Will a real relational DBMS please stand up?" September 7, 1981, page 49.
   b) Plyter, Norman, Letter to the Editor, claiming Henco's INFO is a relational system; October 5, 1981.
   c) Ferris, David, Letter to Editor, replying to Plyter's letter and saying that INFO's 'Relate' function is not equivalent to a relational 'join;' December 7, 1981.
   d) "A sampling of database management systems," February 2, 1982, page 30.
6. Dieckmann, E. M., "Three Relational DBMS," *Datamation* (666 Fifth Avenue, New York, N.Y. 10103); September 1981; p. 137-142; price $4.
7. For more information on ORACLE and ORATOR, contact Relational Software Inc., 3000 Sand Hill Road, Menlo Park, Calif. 94025; tel. (415) 854-7350.
8. For more information on INGRES, contact Relational Technology Inc., 2855 Telegraph Avenue, Suite 515, Berkeley, Calif. 94705; tel. (415) 845-1700.
9. For more information on the IDM, contact Britton Lee, Inc., 90 Albright Way, Los Gatos, Calif. 95030; tel. (408) 378-7000.

COMMENTARY

WHAT PROBLEMS WILL 'END USER SYSTEMS' RAISE?

Friends of ours recently related to us several concerns they have, or have heard expressed by some information systems executives, about problems that end user systems may create.

It appears to us that the particular problems mentioned are not likely to be very troublesome, although at first hearing they sound serious.

Here are our views on those problems.

Problem: "End users will soon create a 'mess' of hard-to-maintain applications. They will then just want to get rid of this mess by turning it all over to the information systems department."

Claimed cause. "This problem centers on three points. (1) End user usage is expected to grow 30% a year, compounded, which includes the applications that users will develop themselves. (2) They will make use of a variety of personal office computers (Apples, Radio Shack, IBM, Xerox, and so on) that have little or no compatibility with each other. (3) They will use incompatible programming languages and non-standard data definitions, and will produce little or no documentation of their systems. In short, they will repeat most of the mistakes that early users of computers did—and then dump everything on us."

Our view. While overall usage of computing power by end users may grow at 30% per year, individual users are unlikely to do anything like that, at least for any length of time. Most end users will use their computers to help them do their jobs. They will have a limited amount of time for using their computers, probably in the order of only a few hours a day at most.

We expect that each person's usage will follow an S-shaped growth curve. It will start slowly, then increase rapidly for a period of time as the person sees new ways to use the computer, and then taper off as the limit of the person's available time is approached.

Also, most end users will not want to do much of their own programming, using a conventional programming language. It takes time to learn how to program, and this knowledge is forgotten all too quickly if used infrequently. And programming takes time away from the user's main job.

If a user has a personal office computer, particularly one that uses the CP/M operating system, then a large variety of useful, inexpensive packages are available. Most end users will *greatly* prefer to purchase packages over trying to develop comparable programs. Packages require no development work and no maintenance by the user.

If end users within a company have purchased a variety of personal office computers, and do not use a standard operating system such as CP/M, they may well ask the information systems department for help in exchanging programs and data files among each other. Company policies can help prevent

such incompatibilities by strongly urging the use of a common operating system.

If, instead of a personal office computer, an end user is using a mainframe or mini, in an 'Information Center' type of environment, then available packages are much more limited. It is more likely that the user may have to program something him/herself. But, of course, the language(s) that will be used will be standard to that organization. Again, the user will have only limited time for such work.

Regardless of the computer used, there is always the problem of storage 'clutter,' from no-longer-used programs and data. If the clutter is stored on off-line floppy disks, the economic impact probably is not as great as if stored in on-line disk storage.

In short, we do not see users of personal office computers developing numerous 'poor' applications that they get tired of maintaining and want to turn over to the information systems department.

Problem: "End users will inadvertently 'mess up' the company's data files."

Claimed cause: "End users will input data from their computers directly to company files, without proper validation. They will store all kinds of data in their local files, about which data administration knows nothing, and this data will surface in undesired ways. Also, they will retrieve data from company files, store it for a while, change it, and then put it back into the company files, causing loss of integrity."

Our view. There is no more reason for end users to enter or change data in company files when using their own computers than when they have terminals tied to the company's mainframe. Company policies and disciplinary action can curb any such tendencies.

The types of information that we see stored on a personal office computer include the person's appointment calendar, notes on past events (diary), tickler file, correspondence, drafts of bodies of text in preparation, data extracted (with authorization) from company files, data applying to the person's area of responsibility, and perhaps purchased application programs and purchased outside data. Data administration would have little concern with most such information.

Data would be extracted from company files for analysis purposes, or for preparing graphical output, or such. It should be dated with the date of extraction, and should not be put back later into the company files.

Again, the problem of clutter arises. And again, clutter from the storage of strictly local data on floppy disks would seem to be less troublesome than if on central hard disk storage.

We are not dismissing the possible problems that end user computers can raise. We just feel that the particular problems mentioned above will not be as severe as some people seem to believe. But it is an intriguing subject area, and we expect to return to it.

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
10. Relational Database Systems Are Here!	G,C,H

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.

EXECUTIVE SUMMARY

After being 'in the wings' for several years, relational database management systems are now finally 'on stage' and taking their positions alongside hierarchical, network, and other types of DBMS.

Even as recently as about one year ago, users who had investigated relational systems had concerns about their slow performance and limited file size capability. Also, there have been systems on the market for several years that their developers have called 'relational'—for use on mainframes, minis, and even on 8-bit micro-computers. Most of these have not met the criteria for 'true' relational systems, though.

But 'true' relational systems are now in use that can handle multiple transactions per second and can manage file sizes of tens (and probably hundreds) of millions of characters. Currently, such systems are available for use on some mainframes (mostly IBM) and minis (mostly DEC). Shortly, they will be offered on micros that use the Motorola MC68000 processor.

At present, IBM is marketing their new SQL/DS relational system as a supplement to, not a replacement for, their DL/1 DBMS. Other relational system suppliers also see their systems now being used primarily for new, medium size applications. However, they do not rule out the possibility of their systems being used to replace some existing DBMS applications—and they cite numerous reasons that users may choose to do so.

And what are those reasons? Mainly, they center on a relational system's friendly user interface—friendlier (say the suppliers) than the user interfaces of most other types of DBMS. From our observation of several of these systems in use, and in talking with some users, we agree that their user interfaces are indeed powerful and easy to operate.

In addition, these relational systems have some interesting features, such as 'user views' (see text) that define what data items each user is authorized to access. When coupled with other end user facilities, such as screen and report program generators and query languages, powerful data management systems result. Many types of new applications can be set up quickly with these tools. When released soon on micro-computers, the benefits of relational systems will be available to organizations of all sizes, small to large.

We do not foresee relational systems obsoleting the other types of DBMS. But we do feel that they have moved from the development stage, and small application stage, into systems that can be of interest to many user organizations.