SPECIFYING QUERIES AS RELATIONAL EXPRESSIONS

R.F. Boyce*, D.D. Chamberlin*
M.M. Hammer**, W.F. King III*

IBM Thomas J. Watson
Research Center
Yorktown Heights, N.Y.

ABSTRACT

SQUARE (Specifying Queries As Relational Expressions) is a set oriented data sublanguage for expressing queries (access, modification, insertion, and deletion) to a data base consisting of a collection of time-varying relations. The language mimics how people use relations or tables to obtain information. It does not require the sophisticated mathematical machinery of the predicate calculus (bound variables, quantifiers, etc.) in order to express simple references to tables. However, the language has been shown to be complete, i.e., any query expressible in the predicate calculus is expressible in SQUARE.

I. INTRODUCTION

In a series of papers E. F. Codd [1-5] has introduced the relational model of data which appears to be the simplest possible data structure consistent with the semantics of information and which provides a maximum degree of data independence.

Given sets S_1, S_2,...,S_n (not necessarily distinct), $R(S_1, S_2,...,S_n)$ is a relation of degree n on these n sets if it is a set of n-tuples each of whose elements has its first component from S_1, its second component from S_2, etc. In other words $R(S_1, S_2,...,S_n)$ is a subset of the Cartesian product $S_1 \times S_2 \times ... \times S_n$. In this paper we will deal only with normalized relations [1]. A relation is normalized if each of its domains is simple, i.e., no domain is itself a relation.

A normalized relation can be viewed as a table of n columns and a varying number of rows as is apparent in Figure I.

A normalized relation has the following properties:

1) Column homogeneity - in any particular column all items are of the same type;
2) All rows of the table are distinct;
3) The ordering of the rows is immaterial;
4) If distinct names are given to the columns the ordering of the columns is immaterial.

The concept of a relation has its present day analog in the notion of a file. The rows or tuples can be thought of as records. The entire data base may be viewed as a collection of time-varying relations of assorted degree upon which inserts, deletes, and updates can be made.

* Present address - IBM Research Laboratory, San Jose, California.
** Present address - M.I.T., Cambridge, Massachusetts.

EMP	NAME	SALARY	MANAGER	DEPARTMENT
	SMITH	10K	JONES	TOY
	JONES	12K	DAHL	FURNITURE
	LEE	10K	THOMAS	APPLIANCE

Figure I: Employee Relation

In addition to introducing the relational data structure, Codd has defined a language [5] which allows for the accessing or referencing of data represented relationally. This language and similar ones (COLARD [6], RIL [7]) are based on the first order predicate calculus. Queries in these languages typically require:

1) The user to define extra variables which have as values rows or portions of rows of a relation, and
2) The user to state the query using Boolean expressions, and quantifiers (universal and existential).

Knuth [8] has shown that the majority of statements in FORTRAN are rather simple. We believe this is also true of queries to a data base. SQUARE is a language which attempts to mimic how people use tables to obtain information. It does not require the sophisticated mathematical machinery of the predicate calculus (extra variables, quantifiers) in order to do relatively simple references to tables. However, it is not hard to show [9] that the SQUARE language is complete, i.e., any query expressible in the predicate calculus is expressible in SQUARE.

The user's perception of a query expressed in the predicate calculus is very different from the SQUARE perception. This is a rather illusive concept to define (section 3 treats it in detail including examples) but for introductory purposes it is sufficient to note that the calculus machinery requires the user to express the query in the form -

1) Select rows of tables
2) Apply a predicate, if true return the rows (or portions of rows)
3) Iterate.

In SQUARE the user expresses the query in the form -

1) Scan a column (or columns) of a table looking for a value (or set of values)
2) For any such values found return the corresponding element(s) of a certain column(s) in the same row.

Put another way, SQUARE enables the user to describe data selection in terms of set oriented table look-ups rather than in a row-at-a-time fashion. This capability makes possible the elimination of quantifiers and the elimination of explicit "linking terms" when the query requires the correlation of information from several tables.

Before proceeding to illustrate the key components of the subject language (section 2) we must comment on the relation of SQUARE to current data base languages, e.g., DML of DBTG [10], DL/1 of IMS [11]. In general terms both the predicate calculus languages and SQUARE are much higher level in the sense of being less procedural. These higher level languages allow the user to specify what are the properties of the data to be accessed, modified, inserted, or deleted rather than how the relevant data is to be found. Hence by moving to such higher level languages, user productivity is greatly increased.

II. DATA MANIPULATION FACILITIES

As we introduce the facilities of SQUARE, we will illustrate them by examples. The examples of this section are drawn from a data base describing the operation of a department store, as follows:

EMP (NAME, SAL, MGR, DEPT)
SALES (DEPT, ITEM, VOL)
SUPPLY (COMP, DEPT, ITEM, VOL)
LOC (DEPT, FLOOR)
CLASS (ITEM, TYPE)

The EMP relation has a row for every store employee, giving his name, salary, manager, and department. The SALES relation gives the volume (yearly count) in which each department sells each item. The SUPPLY relation gives the volume (yearly count) in which each department obtains various items from its various supplier companies. We assume that the SALES and SUPPLY relations have no zero-volume entries (e.g., if the Toy Department does not sell dresses, there is no 'TOY, DRESS, 0' entry in the SALES relation.) The LOC relation gives the floor on which each department is located, and the CLASS relation classifies the items sold into various types.

In this paper we do not deal with the data description language. The questions of unique names, comparability of domains, units, authorization, etc., are not described. For a discussion of these issues, see [6].

We now proceed to describe the syntax of a relational expression, i.e., an expression which evaluates to a relation. The simplest form of relational expression is called a "mapping", and is illustrated by Q1.

Q1. Find the names of employees in the Toy Department.

$$\underset{\text{NAME}}{\text{EMP}}\ (\text{'TOY'})_{\text{DEPT}}$$

A mapping consists of a relation name (EMP), a domain name (DEPT), a range name (NAME), and an argument ('TOY'). The value of the mapping is the set of values in the range column of the named relation whose associated values in the domain column match the argument. This mapping evaluates to a unary relation (in this case, a list of names.) Mapping emulates the way in which people use tables. In this example, to find the names of employees in the Toy Department, a person might look down the DEPT column of the EMP relation, finding 'TOY' entries and making a list of the corresponding NAME entries.

In terms of the first order predicate calculus the notion of mapping can be defined as:

$$R_{B}^{A}(S) \equiv \{\nu[B] : \nu \in R \wedge \nu[A] = S\}.$$

The argument of a mapping may be either a single value (e.g., 'TOY') or a set of values. If the argument is a set, the mapping returns all those range-values whose corresponding domain-values match <u>any</u> element of the argument. Formally, if the argument S is a set of individual values s_i,

$$R_{B}^{A}(S) \equiv \bigcup_{i} R_{B}^{A}(s_i)$$

For this reason the mapping is generally called a disjunctive mapping. For simplicity the term mapping in this paper always refers to a disjunctive mapping.

In certain instances (e.g., with built-in functions SUM, COUNT, etc.) the set-theoretic notions of the disjunctive mapping, which eliminates duplicates from the range set of returned values leads to undesirable side-effects. Consequently, a special type of mapping, denoted by a prime symbol on the relation name, which does not remove duplicates is defined. For example,

Q2. Find the average salary of employees in the Shoe Department.

$$\text{AVG}\ (\underset{\text{SAL}}{\text{EMP}}{}'_{\text{DEPT}}\ (\text{'SHOE'}))$$

Mappings may be "composed" by applying one mapping to the result of another, as illustrated by Q3.

Q3. Find those items sold by departments on the second floor.

$$\underset{\text{ITEM}}{\text{SALES}} \circ \underset{\text{DEPT}}{\text{LOC}}\,('2')$$

The floor '2' is first mapped to the departments located there, and then to the items which they sell. The range of the inner mapping must be compatible with the domain of the outer mapping, but they need not be identical, as illustrated by Q4.

Q4. Find the salary of Anderson's manager.

$$\underset{\text{SAL}}{\text{EMP}} \circ \underset{\text{MGR}}{\text{EMP}}\,('ANDERSON')$$

Q3 is repeated in Section III in order to demonstrate the different perception of the query that is required in order to answer the query in a predicate calculus-like language.

The next important building block of relational expressions is called a free variable. A relational expression containing a free variable takes the following form:

free-variable-list : test

On the left side of the colon are listed the free variables to be used in the query and the relations to which they belong. Each free variable represents a row of a relation. Free variables may be given arbitrary names provided they do not conflict with the names of relations. On the right side of the colon is a logical test which may be true or false for each set of values of the free variables. The value of the expression is the set of free-variable values for which the test is true. A subscripted free variable represents a particular field-value from the row represented by the free variable. For example:

Q5. Find the names of employees who make more than their managers.

$$x_{\text{NAME}} \in \text{EMP} : x_{\text{SAL}} > \text{EMP}_{\text{SAL}}(x_{\text{MGR}})_{\text{NAME}}$$

The following types of operators are permissible in tests:

numeric comparisons:	$= \neq > \geq < \leq$
set comparisons:	$= \neq \supset \supseteq \subset \subseteq$
arithmetic operators:	+ - x /
set operators:	$\cup \cap$ -
logical connectives:	$\wedge$ $\vee$
parentheses for grouping:	( )
built-in functions:	SUM, COUNT, AVG, MAX, MIN, etc.

The following example constructs a binary relation:

Q6. List the name and salary of all managers who manage more than ten employees.

$$x_{\text{NAME, SAL}} \in \text{EMP} : \text{COUNT}(\text{EMP}_{\text{NAME}}(x_{\text{MGR}})_{\text{NAME}}) > 10$$

The free variable is introduced into queries where it becomes necessary to correlate information pertaining to a specific row in a table with another row or set of rows from some table. Consequently, this variable is introduced only for queries that are more complex than simple selection. As can be seen in Section III, all queries regardless of complexity require free variables in predicate calculus based languages.

Another important concept is that of projection. If a relation-name appears subscripted by one or more column-names, it represents the set of unique tuples of values occurring in those columns of the relation. For example, $\text{SUPPLY}_{\text{ITEM}}$ is the set of all item-values in the SUPPLY relation. This feature is useful in constructing expressions like the following:

Q7. Find those companies, each of which supplies every item.

$$x_{COMP} \in SUPPLY : {}_{ITEM}SUPPLY_{COMP}(x_{COMP}) = {}_{ITEM}SUPPLY$$

Note that equality here is set equality.

We will now discuss some extensions to the concept of mapping. A mapping may specify more than one domain field in which case each domain field must be compatible with its respective argument. If an argument is a set then the value of the domain field must match some element of the set. This facility is useful in dealing with n-ary associations. For example:

Q8. Find the volume of guns sold by the Toy Department.

$$VOL\ {}_{DEPT,\ ITEM}SALES('TOY', 'GUN')$$

Similarly, a mapping may specify more than one range field, in which case it returns tuples of values from the fields specified.

When one of the numeric comparison operators $\neq$, $<$, $\leq$, $>$, $\geq$, is used as a prefix to the argument of a mapping, the argument effectively becomes the set of all values which compare by the given operator with the given argument. This type of mapping often avoids the use of a free variable, as illustrated in Q9.

Q9. List the names and managers of employees in the Shoe Department with a salary greater than 10000.

$$NAME,\ MGR\ {}_{DEPT,\ SAL}EMP('SHOE', > '10000')$$

The numeric comparison operators $>$, $\geq$, $<$, $\leq$, may also be extended so that a number may be compared to a set. This is done by placing the word SOME or ALL on the side(s) of the comparison operator which is a set. For example, X > ALL Y is true if the number X is greater than all elements of the set Y, and Y ALL < SOME Z is true if all elements of Y are less than some element of Z. This facility is useful in queries like the following:

Q10. Find the names of those employees who make more than any employee in the Shoe Department.

$$x_{NAME} \in EMP : x_{SAL} > ALL\ {}_{DEPT}EMP_{SAL}('SHOE')$$

In understanding Q10, it is important to remember that the free variable x represents a row of the EMP relation. If the test (which uses the SAL value of the row) is true, the NAME value of the row is returned. All rows of the relation are tested in this way, and duplicate values are eliminated from the returned set.

It should be noted that the functions of ALL and SOME could be accomplished equally well by the built-in functions MAX and MIN. In fact, definitions of the modifiers ALL and SOME are given by the following table, which specifies how any modifier may be replaced by a built-in function:

o represents a comparison operator

	$>, \geq$	$<, \leq$
SOME o	MAX o	MIN o
ALL o	MIN o	MAX o
o SOME	o MIN	o MAX
o ALL	o MAX	o MIN

Examples:

$$X > ALL\ Y \equiv X > MAX\ (Y)$$
$$Y\ SOME < ALL\ Z \equiv MIN\ (Y) < MIN\ (Z)$$

Another language feature which is occasionally useful is a special type of mapping called a conjunctive mapping. As in a disjunctive mapping, a relation name, domain, range, and argument are specified, but the domain name is underlined to denote the conjunctive mapping. The conjunctive mapping differs from a disjunctive mapping only when the argument is a set. In this case, the conjunctive mapping returns the set of range values whose corresponding domain values match all elements of the argument set. Formally, we write the following definitions for a disjunctive mapping and a conjunctive mapping on a set S of values s_i:

$$\overset{R}{B}\ \overset{}{A}\ (S) \equiv \bigcup_i\ \overset{R}{B}\ \overset{}{A}\ (s_i)$$

$$\overset{R}{B}\ \overset{\underline{A}}{}\ (S) \equiv \bigcap_i\ \overset{R}{B}\ \overset{}{A}\ (s_i)$$

As an example of the use of a conjunctive mapping, we might express Q7 as follows, eliminating the free variable:

$$\text{COMP}\ \overset{\text{SUPPLY}}{\underline{\text{ITEM}}}\ (\ \overset{}{\text{ITEM}}\ \text{SUPPLY})$$

In the case of a mapping with more than one domain, each of the domains may participate conjunctively or disjunctively in the mapping; those domains which participate conjunctively are underlined. This is illustrated by Q11.

Q11. Find companies, each of which supplies every item of type A to some department on the second floor.

$$\text{COMP}\ \overset{\text{SUPPLY}}{\text{DEPT},\ \underline{\text{ITEM}}}\ (\ \overset{\text{LOC}}{\text{DEPT}}\ \overset{('2'),}{\text{FLOOR}}\ \overset{\text{CLASS}}{\text{ITEM}}\ \overset{('A'))}{\text{TYPE}}$$

The formal definition of a mapping in which some domains participate conjunctively is as follows (before applying the definition, permute the domains so that the conjunctive domains are on the right):

If $S_1 = \{s_{i_1}\}$, $S_2 = \{s_{i_2}\}$, etc., then

$$\overset{R}{B}\ \overset{}{A_1 \ldots A_k\ A_{\underline{k+1}} \ldots \underline{A_n}}\ (S_1\ S_2 \ldots S_n) \equiv$$

$$\bigcup_{i_1} \ldots \bigcup_{i_k}\ \bigcap_{i_{k+1}} \ldots \bigcap_{i_n}\ \overset{R}{B}\ \overset{}{A_1 \ldots A_n}\ (s_{i_1}\ s_{i_2} \ldots s_{i_n}).$$

This concludes our discussion of the basic accessing facilities of SQUARE. Additional notions of assignment, returning values from functions computed on data, insert, delete, and update are described in detail elsewhere [9].

III. COMPARISON WITH PREDICATE CALCULUS BASED LANGUAGES

In this section we illustrate the difference in perception between queries expressed in the calculus and those expressed in SQUARE. As we have already mentioned, the ALPHA language [5], COLARD [6], and RIL [7] are examples of relational languages based on the first order predicate calculus. They permit the description of sets of data but require the description to be in terms of tests on individual rows of the relations in question. This assumes a certain degree of mathematical sophistication on the part of the programmer.

In the predicate calculus Q1 is expressed as follows:

$$\{v\ [\text{NAME}] \in \text{EMP} : v\ [\text{DEPT}] = \text{'TOY'}\}$$

where v is a variable which ranges over rows of EMP and v[X] is the projection of v on the (set of) domain(s) X. Even for this simple query the user must invent a variable to be used as a cursor for selection of rows.

Q3 shows how this notation is extended for functional composition.

Q3: $\{v_0 [ITEM] \in SALES : \exists(v_1 \in LOC) [(v_1 [FLOOR] = '2') \wedge (v_1 [DEPT] = v_0 [DEPT])]\}$

Here the distinctions between the programmer's perception of the languages becomes clearer. In Section II we saw that the SQUARE programmer could view this query as a simple combination of table look-ups. The calculus programmer must be concerned with:

1) Setting up two variables, v_0 and v_1, to sequence through each table;
2) The notion of existential quantifier and bound variable;
3) The explicit linking term, "v_1 [DEPT] = v_0 [DEPT]", which describes the interrelationship between the variables;
4) The actual matching criteria to be satisfied for membership in the set.

As the queries become more complex the differences between the languages become greater. More variables and linking terms are required in the calculus and the management of quantifiers becomes more complex.

Of course, we do not suggest that really complex queries are simple to express in SQUARE; rather we stress the relative difference between the two approaches and perceptions. As an example of a complex query, we express Q11 in the predicate calculus:

$\{v [COMP] \in SUPPLY : \exists(\ell \in LOC) [(v [DEPT] = \ell[DEPT]) \wedge (\ell[FLOOR] = '2')]$

$\wedge \forall (c \in CLASS)[(c[TYPE] = 'A') \Rightarrow \exists(s \in SUPPLY)$

$((s[COMP] =_v [COMP]) \wedge (s[DEPT] =_v [DEPT]) \wedge (s[ITEM] = c[ITEM]))]\}$.

IV. CONCLUSIONS

This paper has presented the data accessing portion of a data sublanguage based on the relational model of data. This query facility corresponds to the way people use tables. The language does not require the user to have the mathematical sophistication demanded by the previous languages based on the first order predicate calculus. The user describes the relevant data to be accessed by set expressions rather than by row-at-a-time iteration. Consequently, the queries are more concise, use fewer temporary variables, and do not require the quantifiers of the predicate calculus.

ACKNOWLEDGEMENT

The authors wish to thank L. Y. Liu, B. M. Leavenworth, E. F. Codd and P. L. Fehder for their useful discussions.

REFERENCES

1. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks", Comm. ACM, vol. 13, no. 6 (June 1970), pp. 377-387.
2. E. F. Codd, "Further Normalization of the Data Base Relational Model", Courant Computer Science Symposia, vol. 6, Data Base Systems, Prentice-Hall, New York, May 1971.
3. E. F. Codd, "Relational Completeness of Data Base Sublanguages", Courant Computer Science Symposia, vol. 6, Data Base Systems, Prentice-Hall, New York, May 1971.
4. E. F. Codd, "Normalized Data Base Structure: A Brief Tutorial", Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, November 1971.

5. E. F. Codd, "A Data Base Sublanguage Founded on the Relational Calculus", Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, November 1971.
6. G. Bracchi, et. al., "A Language for a Relational Data Base Management System", Proc. of the Sixth Ann. Princeton Conf. on Infor. Sci. and Systems, March 1972, pp. 84-92.
7. P. L. Fehder, "The Representation - Independent Language", IBM Technical Report, RJ 1121, November 1972.
8. D. E. Knuth, "An Empirical Study of FORTRAN Programs", Software - Practice and Experience, vol. 1, no. 2 (April 1971), pp. 105-133.
9. R. F. Boyce, et. al., "Specifying Queries As Relational Expressions: SQUARE", IBM Technical Report, September 1973.
10. CODASYL Data Base Task Group Report, April 1971.
11. Information Management System /360, Application Description Manual H20-0524-1, IBM Corp., White Plains, N.Y., July 1968.

QUESTIONS

Jack Minker:

You seem to be attacking the predicate calculus yet you claim your system is complete with respect to the first order predicate calculus. That is you can put any expression into a representation in the first order predicate calculus. Aren't you sort of hiding the variables, i.e. when you have "department" in there what you really mean is a variable and the only difference is that the first order predicate calculus makes this explicit?

Boyce:

Ignoring the problem of quantifiers, which is a primary problem, the secondary problem with respect to the variables is to make them implicit. We believe that is a good saving because, with respect to the user, to do simple types of queries the user need not come up with a programming language definition of variables having the particular correspondence.

Jack Minker:

You really are not attacking the first order predicate calculus, but it seemed in your talk that you were. What you are doing is making it a little more convenient to the user by hiding the variables from him.

Boyce:

Yes, I would say that is a fair summary. We believe that the first order predicate calculus gives us a complete system but one that is hard to use. What we have sought is to come up with a mathematically equivalent system but one that is easier to use.

Richard Nance:

With respect to Jack Minker's question, I believe that your concern with the predicate calculus is that it might force the user into forms that seem unnatural and that seemed contradictory to one of the goals you mention, which is to have something natural in terms of the user. Is that an accurate assessment?

Boyce: Yes.

Jack Minker:

Wouldn't it be more natural to go to a natural language, rather than a form that is less natural but similar to the predicate calculus?

Boyce:

Yes, if you believe that in the natural language one can make an unambiguous statement of a query. It seems to us that there is a great problem with that from both the user and system points of view. Since Codd has been mentioned, one of the things he is working on now is a system that will accept a natural (English) form of input in an interactive mode and constantly operate on it with interaction by the user to formulate on unambiguous query statement. From the experiments I have seen, this appears to be a very slow query formulation process.

Lawrence Robertson:

I am unclear as to how actually the user to going to express himself. If you say: "Find the employees in the shoe department." Is he going to use the abstract form or something else?

Boyce:

Basically there are two notations in the system; one is the mathematical one you have seen here, which is two dimensional (one of its major drawbacks), and the other is a keyword English syntax. In the second, for example, you would say something like "Select name from employee where department equal toy." I did not present that syntax today. It was a development that came along later.

George Weinberger:

If you are translating from that natural language into SQUARE and from there into the predicate calculus, would it not be equivalent to go directly to the predicate calculus notation?

Boyce:

The English sentences that we have presented have been for explanatory purposes only. We do not intend for that to be the input to the system.

George Weinberger:

What is it that makes the two part translation necessary?

Boyce:

The predicate calculus notation sets variables that run as cursors through each of these rows. For example, a query that takes two variables is like a double DO loop. If you take the set type of expression that I have shown on the board, it takes two linear scans. That requires 2n operations in SQUARE while requiring n^2 in the calculus. So our implementation will not go through the calculus.

George Weinberger:

But the calculus does not necessarily imply that you are going to search twice, that is only a way of describing it. What can internally happen depends on the implementation.

Boyce:

That is right, but we already have a more natural way of describing it in SQUARE; so it seems needless to go through the predicate calculus notation.

Esther Lee:

I am interested in what kind of search you plan to be able to do. For example with a large data base and perhaps a large number of employees, can you do something like searching for all employees whose salaries are within $10,000 of a stated value?

Boyce:

The prototype system that we are building is very heavily index oriented. We use hashed lookups into large indexes. Ideally, we would like to have some type of associative memories. One of the later papers is going to talk about associative disks; I will buy one.

Leo Bellew:

Why did you go to the two dimensional form, what is gained?

Boyce:

There are concepts of functions in the predicate calculus as contrasted to the relational calculus proposed by Codd. We stole that notation as a starting point, and it seemed more natural to use the two dimensions to break things up. There is actually no inherent reason. We could as easily have chosen another notation.

DISCUSSION SESSION

Boyce:

When you are looking at a pattern matching type of situation, i.e., looking for patterns of words that match patterns of words in documents, then the English language is perhaps sufficiently precise to do something meaningful. But I can ask a nice, well-defined query in English, e.g., "What parts have been assigned to San Jose?" and I cause all sorts of problems. I have "quantity on hand" and "quantity on order" for parts; which do I mean? When I consider "assigned to San Jose", I have warehouses in San Jose and I have departments in San Jose; which do I mean? It seems obvious to me that in this case the English language is very ambiguous. The second point I would like to address is what is the difference between that which we are doing and that which Dr. Waksman is doing. While he didn't emphasize it in his paper, Dr. Waksman's system has included a theorem proving type of capability. He can handle inferential type queries while we work only with the actual information stored or simple functions of that information. Those are the two comments I would like to make to invigorate some type of discussion.

Hammer:

As another of the authors of the paper I would like to begin with a particular point; a question that might apply to our work as well as others: "Why are you doing it when there are lots of systems doing the same thing? Rather than reinventing the wheel you should be looking for solutions to the various problems that remain unsolved." The answer to this I think lies in the reply that there are a lot of things that can be done but not economically. By economics I think of both computer usage and human usage. As Winograd has said, it is the human usuage that is becoming more expensive relatively. So we are concentrating on a language that reduces the human cost.

Stanley Su:

Regarding the host language that you are proposing, it seems that this special type of notation is to allow the user to submit a query that meets his knowledge of the data base and his need for information. You imply a negativism about the predicate calculus, yet this formal form required for your language seems to be restrictive.

Boyce:

I think there is a basic conceptual difference between the way a query is viewed in the predicate calculus and in SQUARE. SQUARE is basically a set-oriented language. For example, we begin with something we know, i.e., a floor, and then determine all departments on this floor. Then we might determine all sales items linked with our departments. Consequently, we have mapped from a set of items into another set of items. In no case have we chosen an individual row and applied a predicate to that row. If we were to express that query in the calculus, we would have two free variables, each one having the nature of going through rows of the table one at a time and applying the predicate to see if the item in the rows satisfied the predicate. Nowhere do we work with a single row at a time; all of our processing involves examination of sets.

Hammer:

Implementation-wise it is quite a bit different. You are going to look into the location table for rows with item 4 having value 2 and from this obtain a set of departments. You then go through this set of departments in the index of the sales table on departments to get a set of rows and then come back to take out the item fields. If you were strictly to implement the predicate calculus, you would end up with a double DO loop. This is a completely well defined formal system, and I see no reason to go to a predicate calculus by formulation. Furthermore, I can implement it in a more straightforward manner.

Stanley Su:

It seems to me that you still perform the same operations as you do in the calculus.

Boyce:

The calculus requires n^2 operations and this requires only 2n.

Stanley Su:

If you give me a description of what you are doing, I think that I can map it into the calculus in such a way as to take less than n^2 operation.

Hammer:

Let me answer the question in a slightly different way. What you are saying might be true. We do not have excess power over the calculus. It is as powerful as anything that we can do. We have proved that the SQUARE language is equivalent to the predicate calculus in that we can express any language in SQUARE that can be expressed in any query. We believe that our notations are more straight forward and more natural than that of the predicate calculus. We believe that SQUARE's advantage lies in its expressive style rather than its expressive power.

William A. Zimmerman:

Related to this, it seems to me that what you have is sort of an APL of sequences on n-tuples. Consequently, the difference that we are talking about is almost the same as the difference between APL and FORTRAN in juggling arrays.

Boyce:

I think that is a very good characterization.

William A. Zimmerman:

It seems to me what you really want, as in APL, is to be quite expressive about what kind of data selection and data reduction that you can do in terms of algebra or boolean algebra.

Boyce:

We tried to emphasize in the paper the ease of use of the simpler concepts as opposed to things programmers would like to have. Our technical report addresses the more complicated things that we can do.

Bernard Plagman:

Another thing you might emphasize is the difference between predicate calculus approach to relations between the rows and your approach. The predicate calculus works with the rows and you are working with the columns. Could you invert the tables and do it the same way? Perhaps you get a different look at the information in the tables, but you are doing the same thing. It seems to be a question of the content of the tables and the way you construct them. The real difference, as Hammer mentioned, is the syntactical approach in the way you form the query. If you take the row approach, there is no way to get a handle on sets of rows at one time. If you take the columnar approach, there are ways to talk about sets of values in columns. I think your characterization, subject to the one clarification I made, is quite correct.

Frank Manola:

Earlier in your discussion, you commented that this type of language would be for one type of user while programmers might prefer a language with more control. Would you comment on your expectations as to whether programmers would like to use a language with the capability of SQUARE in their own work space and then doing their own searching procedures?

Boyce:

I have no doubt that programmers can easily adapt to this type of notation and be content with it from a conceptual point of view. The argument we get is that programmers deal with current technology and are more efficiency oriented. How well a system such as this can perform, given the constraints of the technology over the next two to five years, is still an open question. We are still working on the efficiency question.

George Weinberger:

In perspective, relating the work of Salton to your work, you are both providing a front end approach to the problem of searching for material. His approach with the natural language is at a higher level,

while your approach with SQUARE is at a lower level. Neither of you are at extremes of what you want to do, but you are both addressing the same problems with different approaches. His system seems further out than yours and in some sense he could develop his approach and translate it into yours.

Boyce:

Let me make one remark, and say that he has basically a pattern matching type system. He "burdens" the user with pulling away that information he really does not want. He made the statement that something like 40% of the information returned using his approach is irrelevant and should be thrown away. We have a much higher percentage of return information; on the other hand we have a harder specification. I see that you do not agree with that.

George Weinberger:

He throws it away before he gets to your system, and that is all.

Boyce:

No, I think he returns it to the user, who then throws away all that he doesn't want.

Bernard Plagman:

I think he throws away that in which he is not interested before he comes to your system. Then he goes into your system to search for that in which he is interested.

Unidentified Questioner:

I think there is a basic difference in the problem that Salton is solving and the problem that we are solving in that Salton's problem is non-deterministic. We are trying to invent a language that can be used by a user with a clear understanding of that which he wants.

Bernard Plagman:

If he knows what is in the system.

Unidentified Questioner:

A necessary part of the system that uses our language is a catalog, well defined data, and the user can examine the catalog to determine that which he wants.

Bernard Plagman:

In his system he has the catalog and the dictionary, and the dictionary scans the system for the user.

George Weinberger:

Your system in a sense translates to still a lower level, and the question is where should the user actually come in. That depends on the applications that he wishes to address.

Boyce:

Also, the analogies made with respect to the programmer are that he might wish to come in at a lower level.

Stewart A. Schuster:

I think that one of the most important things is that Salton's system is dealing with individual items whereas the relational system is dealing more with sets. More importantly, the relational system is dealing with relations between two sets or two data bases which is the most important problem in data base management. This might be characterized as the cross-reference problem whereas Salton is dealing with relations among one set rather than many relations.

Leo Bellew:

Two problems arise here: (1) the decision problem and (2) the deadlock problem. How do you people handle those two problems since they seem to be crucial ones?

Boyce:

Let me begin with the deadlock problem. This is one small part of a big data base architecture project. The concept consists of three front ends: the first is for the casual user, which is very English like, the second is for an intimate intermediary user, and the third is for the programmer.

All interfaces translate to a system/logical interface. This might be DBTG. Each user above the system/logical interface uses the system as his own private system. At the system/logical level we treat the problem of concurrent users. At this level we have done work on the locking and deadlock level. Locking seems to be a fundamentally different problem in the data base environment than in the operating systems environment. One distinctive difference is that a particular record in a data base environment can be accessed to many different names; whereas, a printer is a printer is a printer in an operating system environment. Another distinction is that in a data base environment, after you access a resource and return it to the system, it is now in a different state. When you return a printer to the operating system, it is still a printer; whereas when you return an employee to the data base, he may no longer be an employee of department X but an employee of department Y. I think it is a very interesting problem, and we handle it at the lower level of the system/logical interface. Would you repeat the second question?

Leo Bellew:

It had to do with logical consistency in whether the system can detect such a thing and how it would treat it?

Boyce:

I'm sure that there could be a logical inconsistency that would grind the system to a halt just as with the FORTRAN program, one can write a legitimate looking program that does not execute properly.

Harold Feinleib:

Returning to the deadlock problem, do you have a way where you can allow the casual user to update base and maintain integrity with respect to locking or do you require the programmee interface, which knows what it is doing, to maintain the locking?

Boyce:

We require the programmee interface to do the locking, and the person at this particular level will not know anything about locking. The current system that we are developing does not prevent locking, and the transmitter must decide the minimum set to be locked and lock it.

Chamberlin:

Let me expand on this a little. We propose to have several user interfaces. At the SQUARE interface we feel the user should not be concerned with locking. The user is basically nonprocedural, and locking is a procedure. However, we can have several users at that level simultaneously accessing that system and the locking will be taken care of for them. The basic problem in translation has to do with isolating something that is a transaction. Finding the minimum necessary unit to lock in order to process that transaction is difficult. We must lock those units in queue transactions in such a way that they will not interfere with each other. Our basic approach is that nothing should remain locked in the system while the user is sitting there thinking. Consequently, we are defining a transaction as that which the user submits and the system can process as a unit.

Bernard Plagman:

(This question, which could not be understood, was related to some system which might be similar to the authors').

Chamberlin:

Our work is not identical to theirs but there is some overlap.

Tom McMullan:

Do you plan to allow the users to improve on modifications to SQUARE?

Boyce:

Yes, we have a set of built-in functions to do common kinds of arithmetic computations, e.g., min, max, etc. We intend that it should be an extensible set of functions although we have not yet developed a particular syntax program.

Stanley Su:

As I understand it, one of the reasons for using something like the predicate calculus is so that you can apply some known theorem proving procedures to arrive at logical deductions and so forth. Can your system utilizing the SQUARE language apply some kind of logical deduction procedures?

Boyce:

We have gone absolutely nowhere in that area. To me, it is the next interesting area. The question/answering systems are based on deduction and theorem proving capabilities, and they prove to be very slow, even for simple queries. It is my hope to develop a more powerful system, but we have yet to do anything in this area.

Stanley Su:

It seems to me that if you want to embody the theorem proving capability, then you must map the language that you have into some form of the predicate calculus.

Boyce:

I have just the opposite opinion--that you will map the predicate calculus into SQUARE.

William A. Zimmerman:

Can one of your columns in relations be a set of relations.

Boyce:

No, you need an argument specification.

Zimmerman:

Could we have a relation such as your sales relationship to a department?

Boyce:

No, we deal with normalized specifications.

Esther Lee:

Is this just for an inquiry system or are you thinking in terms of extending into the interface programming work with other higher level programming languages, e..g., COBOL, FORTRAN, etc.?

Boyce:

We are thinking about doing that. As a matter of fact our prototype is embedded in PL/1. You can write any combination of PL/1 statements in this. But we have not defined what we might call an ideal interface between existing programming languages.

Esther Lee:

How does this system work in a multiple data base environment in such a case where one data item might be identified as "station" in one data base and "installation" in another data base? How would you handle this problem?

Boyce:

Isn't this simply a synonym capability? Is it more complicated than that?

Chamberlin:

This goes back to the question of data definition and cataloging, which came up once before. The material presented here deals only with the data manipulation capabilities of the language. In addition, there is another interface, which might be called a data definition capability, where the users declare the view of data that they wish to have materialized. The underlying data base in some sense is the union of all the user's views. Different users can view the same data in quite different ways, e.g., using different names, and even in some cases different users may view the data as structurally different. A query submitted by a user will be translated considering his particular view, and the result will be translated into the underlying system language.

Esther Lee:

So you do have a translator that sits between systems?

Chamberlin:

We consider it to be a single system with a single underlying data base.

Diane Smith:

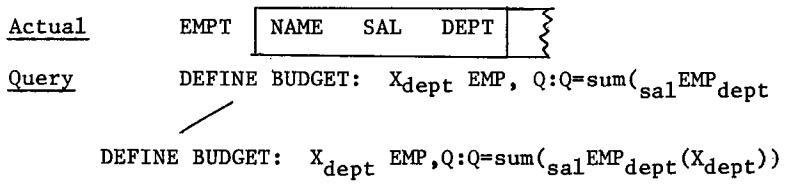
With the ability to have differing users' views of essentially a single system data base, what kind of restructuring capabilities are you planning for resolving differences?

Boyce:

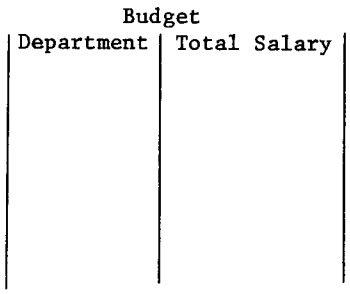
Currently, the system has a set of basic relations. You can formulate queries on top of this. The queries form a network or tree structure. The query is given a name. You ask another query in terms of this meta-relation or meta-structure, and the system, acting in a macro processor type mode, expands the query in terms of the basic relations.

Chamberlin:

Suppose in our data base we have the following relation: An employee has a tag with name, salary and department, as keys in the record. There might also be others. A particular user, who is not interested in employees, wants a particular view that gives a total budget for wages for each department. He says, "I wish to define user view, whose name will be BUDGET". This will appear as follows:



This appears to be a query expressed in the SQUARE language. However, rather than being a one-time query, it has the nature of a definition. This user then can believe he has the following kind of table existing:



And he can issue queries against such a relation.

William Zimmerman:

Are you going to have some provision for retaining such tables based on their usefulness?

Boyce: At the moment no, but it would be useful.

Jack Heller: It may not be useful, for you may have a horrible maintenance problem.

Boyce:

It is a question of frequency of reference versus frequency of updating. We are assuming that it is bad, but we do not have good evidence to support that view.

Esther Lee: Do you have currently or do you plan any backing of queries placed on the system?

Boyce:

We have the notion of a transaction, and you can have a sequence of assignment statements essentially inside a transaction. Hopefully, we will be able to optimize over the entire transaction. We have some algorithms that can do some of that, and there are some things we do not know how to accomplish.

Paul King:

Did I understand you to say that you avoid the deadlock problem by...(unintelligible)...of all the resources available during the process.

Boyce:

No. It is a preemption type scheme, and you assume that you do not have to predeclare everything. It would take too long to explain that in this meeting.

Unidentified Respondent:

I could probably characterize it in 25 words or less. If the user submits a query in SQUARE, the system will decide that set of records, the set of rows, that need to be locked in order to process that query to completion. There may be other queries going through locking records also. If two queries collide, one will have to go to "sleep" and the other will have to preempt one of the records locked by the "sleeping" one. The query that did the preemption will eventually complete its set of records, process them to completion and release them. It's basically a record oriented preemption scheme.

Paul King:

But you are not in fact doing anything in ...(unintelligible)...allocation.

Boyce:

Yes, we are. We have no idea as to how that will perform. If there are many intersections of queries, we have a bad scheme. If there are few intersections, then we have a fairly good scheme in terms of maximizing concurrency.

Chamberlin:

There is some external evidence, a publication by Jack Shemer in last year's SIGFIDET conference, to indicate that the number of deadlocks in data base systems are relatively rare.

George Weinberger:

In the storage interface problem, you claim an optimal storage structure selection. Are all forms stored the same way and how is the selection made?

Boyce:

That is work in progress; it is an extension of the b-tree work of Bayer and others. We really have nothing to say about it at this point since our prototype system does not contain this.

George Weinberger:

Are b-trees the only type of storage structure to be allowed?

Chamberlin:

In addition to choosing the structure in which the data itself is stored, we are also interested in accessing things such as indexes, setting different types of indexes, and setting ways of measuring the traffic against relations automatically.

Leo Bellew: Do you use any of the Data Base Task Group's conceptions?

Boyce: I would say no.

Robert A. Gaskill: I assume that you allow any degree of sparseness in your tables, is that correct?

Boyce: Do you mean no values?

Robert A. Gaskill: Yes.

Boyce: Yes, we do.

Robert A. Gaskill: How do you query on the basis of the absence of a value?

Boyce: There is a specific no value symbol.