Discussed is a high-level data base management language that provides the user with a convenient and unified interface to query, update, define, and control a data base.

When the user performs an operation against the data base, he fills in an example of a solution to that operation in skeleton tables that can be associated with actual tables in the data base. The system is currently being used experimentally for various applications.

Query-by-Example: a data base language

by M. M. Zloof

Query-by-Example[1-4] is a high-level data base management language that provides a convenient and unified style to query, update, define, and control a relational data base. The philosophy of Query-by-Example is to require the user to know very little in order to get started and to minimize the number of concepts that he subsequently has to learn in order to understand and use the whole language. The language syntax is simple, yet it covers a wide variety of complex transactions. This is achieved through the use of the same operations for retrieval, manipulation, definition, and control (to the extent possible). The language operations should mimic, as much as possible, manual table manipulation, thus capturing the simplicity, symmetry and neutrality of the relational model.[5,6] The formulation of a transaction should capture the user's thought process, thereby providing freedom to formulate a transaction. The system should allow the user to create and drop tables dynamically from the data base; it must also provide the user with a dynamic capability for defining control statements and security features.

The architecture of the Query-by-Example language addresses all the requirements just mentioned. The results of various psychological studies of the language[7] show that it requires less than three hours of instruction for nonprogrammers to acquire the skill to make fairly complicated queries. Such queries would otherwise require the user to know first order predicate calculus. Other nonprocedural languages that deal with the same topic are SEQUEL[8] and QUEL.[9]

The first implementation of Query-by-Example was done by K. E. Niebuhr and S. E. Smith.[11] It is currently being used experimentally for various applications, including the management of library files, computer resources, patent files, correspondence files, and expense accounts. The formal syntax and semantics, completeness, authority specifications, and integrity specifications of the language are given in Reference 12.

The following sections introduce the Query-by-Example facilities through illustrative examples of retrieval, manipulation, and definition.

Retrieval

The query part of the language is introduced through examples on the following tables:

- EMP (NAME, SAL, MGR, DEPT),
- SALES (DEPT, ITEM),
- SUPPLY (ITEM, SUPPLIER),
- TYPE (ITEM, COLOR, SIZE),

where the EMP table specifies the name, salary, manager, and department of each employee, the SALES table is a listing of the items sold by each department, the SUPPLY table is a listing of items supplied by each supplier, and the TYPE table describes each item by color and size.

Two basic concepts are fundamental to Query-by-Example. *Programming* is done within two-dimensional skeleton tables. This is accomplished by filling in the appropriate table spaces with an example of the solution. Also, the distinction is made between a *constant element* and an *example element*. Example elements (variables) are underlined, and constant elements are not underlined.

With these two basic concepts, the user can express a wide variety of queries. Consider, for example, the TYPE table, which has ITEM, COLOR, and SIZE as column headings. As an example the user wishes to pose the following query: "Display the green items." Initially, the user is presented with a table skeleton on the screen, as shown in Figure 1.

Figure 1 A table skeleton

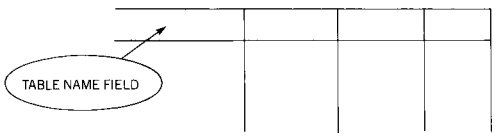


Figure 2 Column headings and entries for display of green items

TYPE	ITEM	COLOR	SIZE
	P. ROD	GREEN	

The user fills in the name of the table to be queried in the table name field; in this case, the name is TYPE. The user may now either fill in the column headings or let the system generate them automatically. (This operation is explained later.) The user next expresses the query by making the entries shown in Figure 2. The P. stands for 'print,' which indicates the desired output. ROD—which is an example element (variable)—is underlined, and represents an example of a possible answer. GREEN is a constant element that represents the required condition in the query, and is therefore not underlined.

Figure 3 Alternate formulation for display of green items.

TYPE	ITEM	COLOR	SIZE
	P.	GREEN	

The query may be paraphrased as follows: Print all items—such as ROD—where the corresponding color is GREEN. ROD need not exist in the data base. Since the example element is an arbitrary example, the user could equally correctly have specified X, 10, 11, or WATER without changing the meaning or the result of the query. Later we shall see that example elements are used to establish links between two or more rows of the same table or different tables. Where no links are necessary, one can entirely omit the example element. It is therefore equally valid to make an alternate formulation of the example query as shown in Figure 3. Thus, stand-alone P. is a default of P. 'Example Element.' In the following examples, we use either P. or P. 'Example Element' arbitrarily.

Figure 4 Example output display of green items

TYPE	ITEM
	PEN
	INK

After formulating a query, the user presses the Enter key to exhibit the answer. By using the example data base given in the Appendix, the output to the query is obtained in the form shown in Figure 4. Only the item column is displayed because the user has entered P. in the ITEM column in Figure 3. If a P. had also been entered in the SIZE column, the system would have printed both the items and their sizes.

When a user expresses a query (or any other operation) in a skeleton table, he perceives that behind this query skeleton there is a real table, with the same headings as the query table, that contains the actual data. In other words, to express a query he mimics the operation of scanning tables manually. If a user requires two or more tables to express a query, he may do so by generating additional blank skeletons (through the use of a special function key) and then filling in their headings.

A number of query types are now classified through the use of illustrative examples. The answers to these queries are found in the Appendix.

Figure 5 Simple retrieval

TYPE	ITEM	COLOR	SIZE
		P. WHITE	

Simple retrieval. Print out all colors. The formulation for this query is shown in Figure 5. Alternatively, WHITE may be omitted. Duplicates of colors are not repeated in this case. (A method for displaying duplicates is shown later in this paper.) In

cases where the system automatically provides these column headings, the user can (if it is desired) erase unused columns from the display. Thus the ITEM and the SIZE columns could have been erased.

Simple retrieval with ordering. Print out all colors in alphabetical order. As shown in Figure 6, AO. stands for ascending order. Similarly, DO. stands for descending order. When there is a need for a major and a minor sort, AO(1). stands for a major sort and AO(2). stands for a minor sort.

Figure 6 Simple retrieval with ordering

TYPE	ITEM	COLOR	SIZE
		P.AO. RED	

Simple retrieval with multiple prints. Print out the entire TYPE table. This operation is shown in Figure 7. Example elements could also have been used. A shorthand representation of the same query is shown in Figure 8. Here the print operator P. is applied against the entire row. Note that all systems operators, such as P., I., and AO., end with a period. If a table contains many columns, the user may print out all but a few by first erasing the unwanted columns, and then placing P. against the row that represents the remaining columns.

Figure 7 Simple retrieval with multiple prints

TYPE	ITEM	COLOR	SIZE
	P.	P.	P.

Figure 8 Shorthand simple retrieval with multiple prints

TYPE	ITEM	COLOR	SIZE
P.			

Retrieval of the table names. List the available table names in the data base. The formulation for this query is given in Figure 9. Here the print is placed in the table name field, thereby asking the system to print out all the available table names. From the example data base, the result of this query yields the following display:

EMP
SALES
SUPPLY
TYPE

Again, the example element TAB is optional.

Figure 9 Retrieval of table names

P. TAB			

Retrieval of column headings. Earlier we stated that instead of the user's filling in the column headings, the system can automatically generate them. This is done as shown in Figure 10. Find the column headings of the TYPE table. Here again, the print operator is applied against the whole row of headings, and is in fact a shorthand form for placing multiple prints in the column heading fields. Automatic generation of the column headings is useful in that it relieves the user of memorizing those headings (or looking them up in a directory), and thereby prevents typographical errors.

Figure 10 Retrieval of column headings

TYPE P.			

If the user places P. TAB P. (or P. P.) in the table name field, the system lists the data base directory, i.e., all table names and their corresponding column names.

Figure 11 Qualified retrieval

EMP	NAME	SAL	MGR	DEPT
	P.	>10000		TOY

Qualified retrieval. Print the names of the employees who work in the toy department and earn more than $10000. This is shown in Figure 11. Note the specification of the condition "more than $10000." One has the option of using any of the following inequality operators: ≠, >, >=, <, <=. If no inequality operator is used as a prefix, equality is implied. The symbol ≠ can be replaced by ¬ or ¬=.

Figure 12 Partially underlined qualified retrieval

TYPE	ITEM	COLOR	SIZE
	P. IKE	GREEN	

Partially underlined qualified retrieval. Print the green items that start with the letter I. This is found in Figure 12. The I in IKE is not underlined, and it is a constant. Therefore, the system prints all the green items that start with the letter I. The user can partially underline at the beginning, middle or end of a word, a sentence, or a paragraph, as in the example, XPAY, which means find a word, a sentence or a paragraph such that somewhere in that sentence or paragraph there exist the letters PA. Since an example element can be blank, then a word, a sentence, or a paragraph that starts or ends with the letters PA also qualifies.

The partial underline feature is useful if an entry is a sentence or text and the user wishes to search to find all examples that contain a special word or root. If, for example, the query is to find entries with the word Texas, the formulation of this query is P. X TEXAS Y.

Qualified retrieval using links. Print all the green items sold by the toy department. This is shown in Figure 13. In this case, the user displays both the TYPE table and the SALES table by generating two blank skeletons on the screen and filling them in with headings and with required entries. The significance of the example element is best illustrated in this query. Here, the same example element must be used in both tables, indicating that if an example item such as NUT is green, that same item is also sold by the toy department. Only if these conditions are met simultaneously does the item qualify as a solution. The manual equivalent is to scan the TYPE table to find a green item and then scan the SALES table to check whether that same item is also sold by the toy department. Since there is no specification of how the query is to be processed or where the scan is to start, the formulation of this query is neutral and symmetric.

Figure 13 Qualified retrieval using links

TYPE	ITEM	COLOR	SIZE
	P. NUT	GREEN	

SALES	DEPT	ITEM
	TOY	NUT

Once the concept of a linking example element is understood, the user can link any number of tables and any number of rows within a single table, as in the following examples.

Example elements linked in the same table. Find the names and salaries of the employees who earn more than Lewis. This is shown in Figure 14. If Lewis earns S1, as an example, then what are the name(s) of employees who earn more than S1? The order of the rows is immaterial: the user is not forced to structure the query in any one specific way. Instead, there is freedom to formulate the query according to one's thought process.

Figure 14 Example elements linked in the same table

EMP	NAME	SAL	
	P.	P. > S1	
	LEWIS	S1	

Another feature of Query-by-Example is the ability to make a complex query by augmenting an existing simpler one. This is an augmentation of the previous example by an additional condition and is illustrated in Figure 15 as follows. Find the names and salaries of the employees who earn more than Lewis and work in a department that sells pens.

Figure 15 Augmentation of the example in Figure 14

EMP	NAME	SAL	MGR	DEPT
	P.	P. > S1		TOY
	LEWIS	S1		

SALES	DEPT	ITEM
	TOY	PEN

Another query is to find the names of the employees who earn more than their managers, as shown in Figure 16. This query may be paraphrased as follows. Print the names of the employees whose manager may be JONES (as an example) and who earn more than S1 (as an example) such that JONES earns S1. Here the same example element is used to link the manager in the first row to the name in the second row, and the same example element is used to compare the salaries. Again, the order of the rows is, of course, immaterial. On receiving the answer, the user has the option of going back and modifying or expanding the old query. If, for example, the user is not sure whether the last query is correct, i.e., whether the question was correctly posed, it is possible to prefix every entry with the print operator, thus getting the employees' names, their salaries and their managers' names in the first row with these same managers' names and their salaries in the second row. In this way the user can verify the accuracy of the query.

Figure 16 Two links in the same table

EMP	NAME	SAL	MGR	DEPT
	P.	> S1	JONES	
	JONES	S1		

Retrieval using a negation. Print the departments that sell an item not supplied by the Pencraft Company. This query is shown in Figure 17. Here the not (¬) operator is applied against the entire query expression in the SUPPLY table. This query may be paraphrased as follows. Print department names for items INK such that it is not the case that PENCRAFT supplies INK. In other words, the system is to look for (INK, PENCRAFT) throughout the entire table, and only if it does not find that entry is the corresponding department printed. This query is different from the following one.

Figure 17 Retrieval using a negation

SALES	DEPT	ITEM
	P.	INK

SUPPLY	ITEM	SUPPLIER
¬	INK	PENCRAFT

Retrieval using a negation. Print the departments that sell items supplied by a supplier other than the Pencraft Company. This

query is illustrated by Figure 18. Here the system retrieves data in the SUPPLY table with suppliers different from Pencraft, and then retrieves the relevant departments. Note that (INK, PENCRAFT) might also exist.

Figure 18 Retrieval using a negation

SALES	DEPT	ITEM
	P.	INK

SUPPLY	ITEM	SUPPLIER
	INK	¬ PENCRAFT

Retrieval of collected output from multiple tables. Print out each department with its corresponding suppliers. Since the output must be a new table, the user must generate a third table skeleton, and fill it in with examples mapped from the two existing tables that satisfy the stipulation of the query. Since it is a user-created table—and, therefore, does not correspond to stored data—the user can fill in the required descriptive headings or leave them blank. This is shown in Figure 19.

Figure 19 Retrieval of collected output from multiple tables

ZZZ	THING	XXX
	P. TOY	P. IBM

SALES	DEPT	ITEM
	TOY	INK

SUPPLY	ITEM	SUPPLIER
	INK	IBM

The use of ZZZ, THING, and XXX as headings illustrates the fact that the user may choose his own headings. Only the linking elements TOY and IBM are transferred from the other two tables to the output table. INK in both the SALES and the SUPPLY tables indicates that TOY is a department supplied with INK by IBM. The order of the tables is immaterial, i.e., the output table need not be displayed first.

If the user does not generate a third table, but rather prefixes TOY and IBM with P. in the original tables, the output is two separate tables, one containing the departments and one the suppliers. Thus, the association between the departments and their corresponding suppliers has been lost.

Consider the following table: EMP1 (NAME, SAL, COMMISSION).

Arithmetic operation. For each employee, list name and salary plus commission. Here again the user creates an output table, and performs the desired arithmetic operation in it, as shown in Figure 20. In this case, the user creates the table OUTPUT and

calls the heading EARNINGS. The arithmetic expression (S1 + S2) sums the salary and commission of each employee. Any arithmetic expression is valid as an entry in a table.

Figure 20 Arithmetic operations

OUTPUT	NAME	EARNINGS
	P. JONES	P. (S1 + S2)

EMP 1	NAME	SAL	COMMISSION
	JONES	S1	S2

Retrieval using a condition box. In Query-by-Example there are two two-dimensional objects. The first is the two-dimensional table skeleton that has been described. The second is the *condition box*, which is a box with the heading CONDITIONS. A blank condition box may be displayed at any time the user desires. A condition box is used to express one or more desired conditions difficult to express in the tables.

Get the names of the employees whose salaries are greater than the sum of those of Jones and Nelson, as shown in Figure 21. Of course, this simple condition could have been expressed by replacing S1 by > (S2 + S3) in the first row of the EMP table.

Figure 21 Use of condition box

EMP	NAME	SAL
	P. JONES NELSON	S1 S2 S3

CONDITIONS
S1 > (S2 + S3)

An equality in a condition box is an equality condition and should not be confused with an *assignment statement.* Assignment statements imply a procedure, and Query-by-Example is a nonprocedural language. Thus, assignment statements are not allowed. The expression W = M + N, for example, can also be stated as M + N = W or M = W − N. The expression M = M + 1 is always false. Different expressions in a condition box are entered on separate lines, but all must hold simultaneously, i.e., all conditions in a condition box are ANDed together.

Retrieval using AND and OR. In Query-by-Example, the AND and OR operations are expressed implicitly. In most of the examples given so far, we ANDed conditions together either by writing more than two entries in the same row (qualified retrieval) or by linking different rows with the same example element (qualified retrieval using links). Queries in Figures 22

Figure 22 Implicit AND operation

EMP	NAME	SAL
	P. JONES JONES JONES	>10000 <15000 ¬13000

Figure 23 Implicit OR operation

EMP	NAME	SAL
	P. JONES	10000
	P. LEWIS	13000
	P. HENRY	16000

and 23 demonstrate the AND and OR operations as used implicitly. The queries in Figures 24 and 25 are reformulations of these queries using a condition box.

Print the names of employees whose salary is between \$10000 and \$15000, provided it is not \$13000, as shown in Figure 22. The use of the same example element JONES in all three rows implies that these three conditions are ANDed on the employee JONES.

Figure 24 AND operation using condition box

EMP	NAME	SAL
	P.	S1

CONDITIONS
S1 = (>10000 & <15000 & ¬13000)

Print the names of employees whose salary is either \$10000 or \$13000 or \$16000. This is illustrated in Figure 23. Different example elements are used in each row, so that the three lines express independent queries. The output is the union of the three sets of answers. (In this example, the P.'s alone would have been sufficient.)

Reformulating AND and OR operations using a condition box. Figure 24 illustrates the formulation of the AND operation using a condition box; Figure 25 shows the formulation of the OR operation. The ampersand & represents the AND operation and the bar | represents the OR.

Figure 25 OR operation using condition box

EMP	NAME	SAL
	P.	S1

CONDITIONS
S1 = (10000 | 13000 | 16000)

The user can also specify AND or OR on ordered pairs of entries, as in the following example. Print the names of employees whose salary and department is either (\$10000, Toy) or (\$20000, Hardware). The salaries and departments are ordered pairs that are to be ORed as shown in Figure 26.

Figure 26 OR operation on ordered pairs

EMP	NAME	SAL	DEPT
	P.	S1	D1

CONDITIONS
(S1, D1) = ((10000, TOY) | (20000, HARDWARE))

Note that up to this point with the concepts of the example element, constant element, print operator, and their use within the table skeletons and condition box, the user can express quite complicated queries. These cover a wide variety of relational data base operations such as projections, selections, joins, projection of joins, union, difference, intersection, and arithmetic operations.

Retrieval using built-in functions and the ALL. operator. There are six built-in functions in the Query-by-Example language, which are the following: CNT. (count), SUM., AVG. (average), MAX. (maximum), MIN. (minimum), and UN. (unique). The function UN. can be attached to the function CNT., SUM., or AVG. Thus CNT. UN. means count only the unique values, etc. The following examples illustrate their use.

Figure 27 Use of the function CNT.

EMP	NAME	
	P. CNT. ALL. JONES	

Count the total number of employees. This is shown in Figure 27. The expression ALL.JONES represents a *multiset* (a set that retains duplicates) of all names in the EMP table, and the CNT. function counts this set. Here again, one can use P.CNT.ALL., omitting the example element JONES.

Count the total number of departments in the SALES table. This is illustrated in Figure 28. Since there are duplicate departments in the DEPT column (in which DEPT is not a key), the function UN. is attached to eliminate duplicates. ALL. does not automatically eliminate duplicates since it is a multiset. Note that if one enters P. ALL.TOY, a printout of all the departments and their duplicates is obtained.

Figure 28 Use of the functions CNT. and Un.

SALES	DEPT	ITEM
	P. CNT. UN. ALL. TOY	

Qualified retrieval using a built-in function. Print the sum of the salaries in the Toy Department. This is shown in Figure 29. ALL.S1 is the multiset of all salaries that match TOY, i.e., the multiset of all salaries in the Toy Department. Thus, if all fifty employees have $12 000 as their salary, the printout is fifty times $12 000.

Figure 29 Qualified retrieval using built-in function

EMP	NAME	SAL	DEPT
		P. SUM. ALL. S1	TOY

Retrieval with grouping. For each department, print the name and the sum of the employees' salaries. This is shown in Figure 30. The grouping is accomplished by double underlining TOY for an explicit 'group-by' operator. This query can be paraphrased as follows: for each department TOY (as an example), sum all the salaries matching it.

Figure 30 Retrieval with grouping

EMP	NAME	SAL	DEPT
		P. SUM. ALL. S1	P. TOY

Get the departments that have more than three employees. This is shown in Figure 31. Built-in functions only operate on set expressions, so they must be followed by the ALL. operator or a bracketed set expression. Thus CNT.INK results in an error message.

Figure 31 A condition on the set of items

EMP	NAME	DEPT
	ALL. JONES	P. TOY

CONDITIONS
CNT. ALL. JONES >3

Retrieval involving 'set links' using ALL. Get the names of the departments each of which sells at least all the green items. This is shown in Figure 32. The expression ALL.INK in the TYPE table represents a multiset (i.e., a set that retains duplicates) of all green-colored items. Thus ALL.INK is a multiset of all the green items. The row in the SALES table means that we are looking for departments such as TOY that sell this set and possibly more. The asterisk indicates that there may be additional items not in the set.

Figure 32 Retrieval involving "set links"

SALES	DEPT	ITEM
	P. TOY	[ALL. INK *]

TYPE	ITEM	COLOR
	ALL. INK	GREEN

Get the names of the departments such that all the items of each department have to be green. This is shown in Figure 33. In this example, ALL.INK in the SALES table represents the multiset of all the items that match the department TOY. The asterisk in the TYPE table indicates that although ALL.INK must be green, there may also be additional green items.

Figure 33 Retrieval involving "set links"

SALES	DEPT	ITEM
	P. TOY	ALL. INK

TYPE	ITEM	COLOR
	[ALL. INK *]	GREEN

Get the names of the departments each of which sells all the green items, and nothing more. This is shown in Figure 34. Here the sets on both tables are the same, which means no additional items that are not green nor any different green items sold in a different department.

Figure 34 Retrieval involving "set links"

SALES	DEPT	ITEM
	P. TOY	ALL. INK

TYPE	ITEM	COLOR
	ALL. INK	GREEN

Get the names of departments each of which sells all the items supplied by the Hardware Department and possibly more. Exclude the Hardware Department itself from the output. This is shown in Figure 35.

Figure 35 Retrieval involving "set links" with a condition box

SALES	DEPT	ITEM
	P. TOY	[ALL. INK *]
	HARDWARE	ALL. INK

CONDITIONS
TOY ¬ = HARDWARE

Although a simple link is achieved by using the same example element in two or more entries, in a set link one has to distinguish between set equality and set containment. The former is achieved by identical set link as in Figure 34, whereas the latter is achieved by the use of an asterisk.

A general bracket expression may contain one or more sets and any number of single example or constant elements, with or without a single asterisk. For example, the following bracketed entry is valid:

Whereas the entry

is not valid because it does not contain a set.

Insertions, deletions, and update

Insertions (I.) deletions (D.), and updates (U.) are done in the same style as the query operations. The only major difference is the use of I., D., and U. in place of the P. used in query expressions. This section is broken down into the following two parts: simple insertions, deletions, and updates, and those that are query-dependent. The term "simple" implies operations that involve constant elements only.

Insert into the employee table a new employee of the Toy Department named Jones, whose salary is $10000, and whose manager is Henry. This is shown in Figure 36. As P. is used against a whole row, the I. also applies to the whole row. A blank entry during the insertion operation is interpreted as a null, with the restriction that null entries are not allowed in the primary key field(s), as in the NAME field in Figure 36. This is consistent with the Relational Model[5,6] in that a primary key field must uniquely identify the record (tuple). (The specification of a primary key field is shown later in Figure 45.)

Figure 36 Simple insertion

EMP	NAME	SAL	MGR	DEPT
I.	JONES	10000	HENRY	TOY

Simple deletion. Delete all information about employees in the Toy Department. This is shown in Figure 37. All records having Toy as a Department entry are deleted in this case. It is, of course, optional to fill in the NAME, SAL, MGR with example elements.

Figure 37 Simple deletion

EMP	NAME	SAL	MGR	DEPT
D.				TOY

Simple update. Update Henry's salary to $50000. This is shown in Figure 38. Query-by-Example does not allow the user to update the primary key field(s). The expression in Figure 38 updates Henry's salary to $50000 regardless of its old value. The primary key field(s) must be specified to ensure uniqueness. Blank fields imply that no updating is required. If, however, the user wants to update a field to a null value, the user must either enter NULL or a special user-defined symbol such as is described later in this paper.

Figure 38 Simple update

EMP	NAME	SAL	MGR	DEPT
U.	HENRY	50000		

Figure 39 Query-dependent insertion

EMP	NAME	SAL	MGR	DEPT
I.	HENRY	S1	LEE	TOY
	LEWIS	S1		

Query-dependent insertion. Insert into the employee table an employee in the Toy Department whose name is Henry, whose manager's name is Lee, and whose salary is the same as that of Lewis. This is shown in Figure 39. This is a query-dependent insertion because the system must first query the data base to find Lewis's salary and then duplicate it for Henry. Although the sequence of the rows is immaterial, there is an implicit ordering in the execution. The insertion cannot be completed until after executing the query.

Figure 40 Query-dependent deletion

EMP	NAME	SAL	MGR	DEPT
D.				D1

SALES	DEPT	ITEM
	D1	PEN

Query-dependent deletion. Delete all employees who work in departments that sell pens. This is shown in Figure 40. Here the filling in of the NAME, SAL, and MGR columns with example elements is optional.

Query-dependent updates. It is often desirable to update an entry with a value relative to its old value. This is done implicitly. Raise the salaries of the employees in the Toy Department by ten percent. This is shown in Figure 41. The salary expression in the row that contains the U. is the value after the update, and the second row retrieves the old salary values. This operation can be paraphrased as follows: retrieve a record that has Toy as a department, find the salary S1 and update that salary to 1.1 times S1.

Figure 41 Query-dependent updates

EMP	NAME	SAL	MGR	DEPT
U.		1.1 * S1 S1		TOY

Since an output of a relation or table in a relational data base is itself a table, the user can operate directly on that table. As an example, after retrieving as output all employees in the Toy Department, the user may delete any specific record by placing a D. against that record and pressing the ENTER key.

Table creation

In the Query-by-Example language, the creation of tables is done in the same style as the previous operations. A table is defined by using skeletons with constant and example elements. The user has facilities that create a new table, extend an existing table, create a *snapshot* table—that is, one that contains collected data from various tables at a particular point in time—or create a dynamic *view* of data collected from various tables. All such operations are done implicitly, without using special key words as was required in an earlier version of the language.[2]

Figure 42 Creation of table headings

I. EMP I.	NAME	SAL	MGR	DEPT

Creation of a new table. Create a new table with table name EMP and column headings: NAME, SAL, MGR, DEPT. Starting with a blank skeleton on the screen (as though one were formulating a query), the user fills in the headings by inserting the field names, as shown in Figure 42. The I. on the right of EMP refers to the whole row of column headings.

Certain system row attributes (key words) are utilized to specify the data types, sizes, domains, and keys, etc. To aid the user, these attributes are attached to all skeletons, and the user need not insert them. For example, the user can ask for the names of the row attribute fields, as shown in Figure 43. This operation creates a new table and then requests display of all the row attributes in the system. The resultant output is as shown in the lower part of Figure 44. The user then defines the attributes of the table by filling in the corresponding rows, as shown in Figure 45. Since the row attributes are already imbedded in the table, the user is not required to enter an I. on the left of these attributes; no harm results, however, if the I.'s are entered.

Figure 43 Retrieval of row attribute names

I. EMP I.	NAME	SAL	MGR	DEPT
P.XX				

Figure 44 Display of row attribute names

EMP	NAME	SAL	MGR	DEPT
TYPE				
LENGTH				
KEY				
DOMAIN				
SYS NULL				

Figure 45 Definition of row attributes

EMP		NAME	SAL	MGR	DEPT
TYPE	I.	CHAR	FLOAT	CHAR	CHAR
LENGTH	I.	20	8	20	12
KEY	I.	K	NK	NK	NK
DOMAIN	I.	NAMES	MONEY	NAMES	DEPARTMENTS
SYS NULL	I.		—	—	—

Row attributes of tables are described as follows:

- TYPE specifies the data entry type, such as CHAR, FLOAT, FIXED, etc.
- LENGTH specifies the length of that field. (The default is the length of the column heading.)
- KEY specifies the fields that are to be considered as primary keys. (K stands for Key and NK represents Nonkey.) As stated earlier in this paper, primary key fields may not contain null values, nor may they contain duplicate entries. The specification of the key fields ensures that these constraints are maintained. In addition, the system prevents the user from updating key fields.
- DOMAIN specifies the name of the underlying domain, the value set from which data elements are drawn. For example, the data in the columns NAME and MGR are both drawn from the underlying domain of NAMES. The specification of the DOMAIN is useful when the user needs to know which columns are drawn from the same underlying domain. The column heading names are not always sufficient for this. For

example, the NAME and MGR columns in the EMP table are drawn from the same underlying domain. Thus the elements of NAME and MGR may be linked.

SYS NULL (System Null) specifies an optional symbol to be used in the system as null. In this example, the symbol (—) is used.

Having specified all or part of the entries in Figure 45, the user can enter data in the same skeleton table. After the data have been inserted, queries may then be formulated against the data definition directory, as well as against the data in the same skeleton.

Figure 46 Retrieval of the row attribute data

EMP	NAME	SAL	MGR	DEPT
P.XX P.				

Table expansion. The owner of a table may expand an existing table in the same fashion that the original table was created. For example, add a column to be labeled COMMISSION to the EMP table. The user first queries the EMP table to retrieve all the data concerning the row attributes as shown in Figure 46. The answer to this query is the whole table directory previously defined. The user inserts a new column name and new entries in the directory as shown in Figure 47. If data are currently present in that table, the COMMISSION column is considered to have null data until the user updates the table.

Figure 47 Defining a new column in a table

EMP	NAME	SAL	MGR	DEPT	I. COMMISSION
TYPE	CHAR	FLOAT	CHAR	CHAR	I. FLOAT
LENGTH	20	8	20	12	I. 8
KEY	K	NK	NK	NK	I. NK
DOMAIN	NAMES	MONEY	NAMES	DEPARTMENTS	I. MONEY
SYS. NULL		—	—	—	I. —

Update of the table directory. The operator U. updates the table directory, be it the headings or the key word specifications, in the same fashion as I. For example, if we update the table name from EMP to EMP1, we prefix the table name with the U. operator and type in the new name on top of the old; pressing the ENTER key then updates the table name.

Deletion of information in the table directory and dropping a table. Here the D. operator is used to delete directory entries in the same way in which it is used to delete normal data entry. Thus placing a D. against a record that specifies the key and nonkey fields causes the deletion of those key specifications.

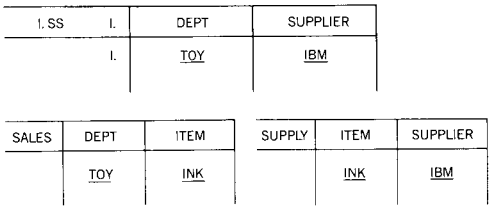
To drop a whole column from the data base, the user prefixes that column name with a D. This is actually a shorthand for deleting both the entire data in that column and the column name.

Similarly, prefixing a table name with a D. is a shorthand for deleting the entries, the directory information, the column headings, and the table name.

Creating a snapshot. Previously we showed how the user can obtain a new table that consists of data from other tables. This table was displayed and not stored by the system. The user may store that table by creating a heading for it (as though one were creating a new table).

Redo the example illustrated in Figure 19 but store the resultant table with table name SS and column headings DEPT, SUPPLIER. This is shown in Figure 48. Table SS is a snapshot of the data stored in the tables at the time of its creation. Because this is now a table in the system, the user may insert, delete, or update it at will.

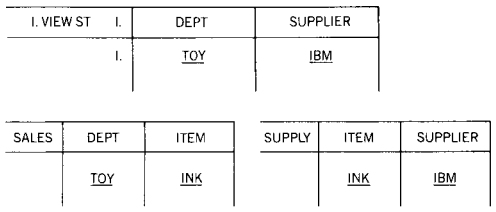
Figure 48 Creating a snapshot



Creating a view. Often the user wishes to create one table from several others, and further wants the new table to reflect changes in the base tables dynamically. Such a table is called a *view*, and changes made to the underlying tables are reflected in the view. To create a view, the user has to prefix the table name with the keyword VIEW, thus distinguishing it from a snapshot.

Redo the previous example but instead of a snapshot create a view named ST, as shown in Figure 49. The data in a view are not physically stored. Rather, the system stores pointers to its parts and the data materialize when the user queries this view for output.

Figure 49 Creating a view



Concluding remarks

This paper has presented an overview of retrieval, manipulation, and definition by the Query-by-Example language. One of the unique features of this language is that very few concepts need be learned for one to start using the system.

Also, as opposed to English-like query languages by which the user has to conform to the phrase structure of the language, the Query-by-Example user may enter any expression as an entry as long as it is syntactically correct. In other words, since the entries are bound to the table skeleton, the user can only specify admissible queries. In an English-like query language, on the other hand, the user may formulate a query that does not conform to the phrase structure of the language.

Since the sequence of filling in the tables is immaterial, the user enjoys many degrees of freedom in formulating a transaction.

It has been shown that the user can build up a query by augmenting it in a piecemeal fashion—each time adding a new condition—thereby making a smooth transition to difficult queries.

Retrieval, manipulation, and definition are all similarly accomplished with minimum new syntax.

Appendix

The following tables give the data base that is used for the examples in this paper:

EMP	NAME	SALARY	MGR	DEPT
	JONES	8000	SMITH	HOUSEHOLD
	ANDERSON	6000	MURPHY	TOY
	MORGAN	10000	LEE	COSMETICS
	LEWIS	12000	LONG	STATIONERY
	NELSON	6000	MURPHY	TOY
	HOFFMAN	16000	MORGAN	COSMETICS
	LONG	7000	MORGAN	COSMETICS
	MURPHY	8000	SMITH	HOUSEHOLD
	SMITH	12000	HOFFMAN	STATIONERY
	HENRY	9000	SMITH	TOY

SALES	DEPARTMENT	ITEM
	STATIONERY	DISH
	HOUSEHOLD	PEN
	STATIONERY	PENCIL
	COSMETICS	LIPSTICK
	TOY	PEN
	TOY	PENCIL
	TOY	INK
	COSMETICS	PERFUME
	STATIONERY	INK
	HOUSEHOLD	DISH
	STATIONERY	PEN
	HARDWARE	INK

SUPPLY	ITEM	SUPPLIER
	PEN	PENCRAFT
	PENCIL	FLIC
	INK	PENCRAFT
	PERFUME	BEAUTEX
	INK	FLIC
	DISH	CHEMCO
	LIPSTICK	BEAUTEX
	DISH	FLIC
	PEN	BEAUTEX
	PENCIL	PENCRAFT

TYPE	ITEM	COLOR	SIZE
	DISH	WHITE	M
	LIPSTICK	RED	L
	PERFUME	WHITE	L
	PEN	GREEN	S
	PENCIL	BLUE	M
	INK	GREEN	L
	INK	BLUE	S
	PENCIL	RED	L
	PENCIL	BLUE	L

The following are the answers to the example queries in the text:

Figure 5

TYPE	COLOR
	WHITE
	RED
	GREEN
	BLUE

Figure 6

TYPE	COLOR
	BLUE
	GREEN
	RED
	WHITE

Figure 7

TYPE	ITEM	COLOR	SIZE
	DISH	WHITE	M
	LIPSTICK	RED	L
	⋮	⋮	⋮

Figure 10

TYPE	ITEM	COLOR	SIZE

Figure 11

EMP	NAME

Figure 12

TYPE	ITEM
	INK

Figure 13

TYPE	ITEM
	PEN
	INK

Figure 14

EMP	NAME	SAL
	HOFFMAN	16000

Figure 15

EMP	NAME	SAL

Figure 16

EMP	NAME
	LEWIS
	HOFFMAN

Figure 17

SALES	DEPT
	STATIONERY
	COSMETICS
	HOUSEHOLD

Figure 18

SALES	DEPT
	STATIONERY
	HOUSEHOLD
	COSMETICS
	TOY
	HARDWARE

Figure 19

ZZZ	THING	XXX
	STATIONERY	FLIC
	STATIONERY	CHEMCO
	HOUSEHOLD	PENCRAFT
	STATIONERY	PENCRAFT
	⋮	⋮

Figure 20 This query is not part of this data base.

Figure 21

EMP	NAME
	HOFFMAN

Figure 22

EMP	NAME
	LEWIS SMITH

Figure 23

EMP	NAME
	MORGAN HOFFMAN

Figure 26

SALES	DEPT

Figure 27

EMP	NAME CNT.
	10

Figure 28

SALES	DEPT CNT.
	5

Figure 29

EMP	SAL SUM.
	21000

Figure 30

EMP	SAL SUM.	DEPT
	16000 21000 33000 24000	HOUSEHOLD TOY COSMETICS STATIONERY

Figure 31

SALES	DEPT
	STATIONERY

Figure 32

SALES	DEPT
	STATIONERY TOY

Figure 33

SALES	DEPT
	HARDWARE

Figure 34

SALES	DEPT

Figure 35

SALES	DEPT
	STATIONERY TOY

ACKNOWLEDGMENTS

The author is indebted to S. P. deJong and K. E. Niebuhr for their helpful suggestions throughout the development of Query-by-Example. The author also wishes to thank S. E. Smith, R. R. Jones, and R. J. Byrd for their helpful discussions.

CITED REFERENCES

1. M. M. Zloof, "Query by Example," *AFIPS Conference Proceedings, National Computer Conference* **44,** 431–438 (1975).
2. M. M. Zloof, "Query by Example, The Invocation and Definition of Tables and Forms," *Proceedings of The International Conference on Very Large Data Bases*, Boston, Massachusetts, September 22–24, 1975, pp. 1–24.
3. M. M. Zloof, *Query-by-Example: Operations on the Transitive Closure*, Research Report RC 5526, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1975.
4. M. M. Zloof, "Query-by-Example: Operation on Hierarchical Data Bases," *AFIPS Conference Proceedings, National Computer Conference* **45,** 845–853 (1976).
5. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM* **13,** No. 6, 377–387 (1970).

6. E. F. Codd, "Further Normalization of the Data Base Relational Model," *Courant Computer Science Symposia Vol. 6, Data Base Systems*, Prentice-Hall, Inc., New York, NY (1971).
7. J. C. Thomas and J. D. Gould, "A Psychological Study of Query by Example," *Proceedings of the National Computer Conference* **44,** 439–445 (1975).
8. D. D. Chamberlin *et al.*, "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control," *IBM Journal of Research and Development* **20,** 560–575 (1976).
9. G. D. Held, M. R. Stonebraker, and E. Wong, "INGERS: A Relational Data Base System," *Proceedings of the National Computer Conference* **44,** (1975).
10. C. J. Date, *An Introduction to Data Base Systems*, Addison-Wesley Publishing Co., Inc., Reading, MA (second edition, 1977).
11. K. E. Niebuhr and S. E. Smith, "Implementation of Query-by-Example on VM/370," Research Report, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, in preparation.
12. M. M. Zloof, "Query-by-Example: A Data Base Management Language," IBM Research Report available upon request from the author, IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

GENERAL REFERENCES

1. M. M. Zloof and S. P. deJong, "The system for business automation (SBA): Programming Language," *Communications of the ACM* **20,** No. 6, 385–396 (1977).

2. S. P. deJong and M. M. Zloof, "Application design within the system for business automation (SBA)," *Proceedings of the Twelfth Design Automation Conference*, Boston, Massachusetts, June, 1975, pp. 69–76.

Reprint Order No. G321-5056.