Query-by-example—Operations on hierarchical data bases

by MOSHÉ M. ZLOOF
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

ABSTRACT

Query-by-Example is a high level non-procedural data base language which provides the end user with a simplified unified interface for querying, updating, defining, and maintaining, the data base, as well as embedding various integrity and authority constraints. When querying the data base the user fills in, through a keyboard display, an example of a possible answer in a skeleton of the logical structure of the data base. As demonstrated in previous work, when the data base is relational, skeleton tables are used. In this paper, we show that the Query-by-Example operations are in fact independent of the structure of the data base. In particular, we demonstrate that if the view at the user interface level is hierarchical, the query is again accomplished by filling-in an example of a possible answer, but, in this case, a skeleton of the hierarchy is utilized. It is, also, shown how the user can map a relational view into a hierarchical view and vice versa, formulating queries that involve both views simultaneously. Finally, a linear version of Query-by-Example is made available for situations where a display facility is unavailable.

INTRODUCTION

Query-by-Example is a high level non-procedural data base language which provides the end user with a simplified unified interface for querying, updating, defining, and maintaining the data-base, as well as embedding various integrity and authority constraints. When querying the data base the user fills in, through a keyboard display, an example of a possible answer in a skeleton of the logical structure of the data base. Our main philosophy behind Query-by-Example was to allow the user to learn very little to get started, and keep the number of objects and concepts that has to subsequently be learned, to cover the whole language, at a minimum. And in fact, the results of various behavioral tests conducted in our labs to teach non-programmers the Query-by-Example language shows that in less than three hours the users could ask quite complicated queries as powerful as first order predicate calculus.

Initially, Query-by-Example was started as a query language for relational model of the data base as was introduced by E. F. Codd but since then it was extended to a whole data base language with facilities to update, define, and maintain the data base; however, to keep the simplicity of the language intact, most of the operations used for querying the data base are kept in use for updating, defining, and maintaining that data base, thus providing the user with only one unified interface. For that reason we did not feel it is necessary to change the name of the language for the other modes of operations, such as Define-by-Example or Update-by-Example etc., since the same Query-by-Example operations are also used for these modes.

In this paper we introduce Query-by-Example operations on a hierarchical view of the data base (view being what the user sees at the user interface level, independently of its underlying model—for example, a hierarchical view can be built on top of a relational model of the data). Here a query is again formulated by filling in an example of a possible answer through a keyboard display, but in this case a skeleton of the hierarchical structure is utilized. It is also shown how the user can map a hierarchical view of the data into a relational view and vice versa. This implies that the Query-by-Example operations are independent of the view of the data base be it relational, hierarchical, network, or any combination of the three.

The reason why the user should be given the options to view the data base in the above three different ways are as follows:

—It is sometimes more intuitive for the user to view part of his data base as a hierarchical tree (such as an organizational chart), although the underlying model may be relational.

—Although we are of the opinion, as are many others, that the relational approach which was proposed by Codd, will in the future be implemented on large-scale systems, the current large-scale implementations, however, are primarily hierarchical or network.

Therefore, given a currently standing hierarchical data base model (such as IMS) with a pro-
grammer who is already trained to work with such a hierarchical structure, it would be advantageous to be able to replace current procedural data languages (such as DL/1 or GIS) by a high level non-procedural language (such as Query-by-Example) that operates on the same hierarchical views of the data base.

While the procedural language programmer must have knowledge of the hierarchical structure and access paths of the data base in order to 'navigate' through the tree paths, the Query-by-Example user is only required to have knowledge of the data base structure. The transformation of a non-procedural Query-by-Example statement into a procedure in a lower level language is done by the system through a proper interpreter.

The last reason is that of enhancing performance. If in a given system performance is a major factor, one can enhance it by letting the user view the data base as a hierarchy. Once he is aware that certain paths are more efficient than others, the user may have some control over the performance of the system even though the language is still non-procedural.

A summary of the basic concepts of Query-by-Example as they apply to a relational view of the data base is presented in the second section of this paper. In addition, a linear version is introduced for situations where a display facility is unavailable. In the third section, we introduce Query-by-Example operations on a hierarchical view of the data base. Section four is a combination of the basic concepts of the second and third sections, such that the user can formulate a query concerning both relational and hierarchical views simultaneously.

It is also shown how the user can map one view into another. In the fifth section, we comment on how the user can participate in enhancing the performance of the system.

SUMMARY OF THE MAIN FEATURES OF QUERY-BY-EXAMPLE

When dealing with a relational data base, the user basically formulates his query by filling in skeleton tables with an example of a possible answer through a keyboard display. In fact, to get started on the system, the user need only distinguish between the following two entities:

1. The 'example element' (variable) which must be underlined and
2. The 'constant element' which should not be underlined.

In addition, the function denoted by 'P.' stands for 'print': the user inserts 'P.' before any data he wishes to be outputted.

A relation is basically a table, and an example of such a table is the employee relation given in Figure 1.

As in Reference 1, the concepts of Query-by-Example are introduced through illustrations of queries and their answers. The queries are drawn from the following relations which are a part of a department store data base.

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

The EMP Table specifies the name, salary, manager, and department of each employee.

The SALES Table is a listing of the items sold by departments.

The SUPPLY Table is a listing of the items supplied by suppliers.

The TYPE Table describes each item by color and size. A sample of the above data base is shown in Figure 2. At this point we are assuming that these tables are made available to the user upon calling them by name.

Actually, in a current implemented prototype, the sequence of user operations in constructing a query is as follows:

As in Reference 1, the concepts of Query-by-Example are introduced through illustrations of queries and their answers. The queries are drawn from the following relations which are a part of a department store data base.
The user is initially presented with a display with a blank skeleton table in which he enters through a keyboard, the appropriate table name in the table name field. Upon pressing a function key, the column names are automatically filled in by the system. If needed, a different function key will create additional blank skeleton tables etc. After having on the screen all the required skeletons with their column names, the user proceeds to fill in these tables with elements to satisfy the stipulation of the query. For more details, please see Reference 13.

Let us now proceed with query examples.

**Examples:**

**Q1.** Print the red items.
The user fills in the TYPE Table in the following manner.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ITEM</th>
<th>COLOR</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.PEN</td>
<td>RED</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Explanations:**

Since the query is concerned with red items, RED is a constant element and is, therefore, not underlined. On the other hand, the underlined element PEN is the example element and is entered as an example of a possible answer. Actually, a pen may not necessarily be an element of the data base and can be substituted by DRESS, WATER, or a variable X without altering the meaning of the query. One of the reasons we are using an example element is that it gives us the freedom to use an entity which is partially variable and partially constant (such as PEN, meaning: Print the red items that start with the letter P). Also, for the above simple query, one can dispose of the example element PEN altogether, entering only 'P'. The SIZE Column can be left blank or filled in with an example element.

Considering the sample data base, the answer to Q1 is:

<table>
<thead>
<tr>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIPSTICK</td>
</tr>
<tr>
<td>PENCIL</td>
</tr>
</tbody>
</table>

For cases where a display is not available, a linear version of Query-by-Example, which is a straight forward mapping of the tabular version, is also available, and Q1 in linear form will be written as follows:

**Q1.** \( \text{TYPE}(\text{ITEM} : \text{P.PEN}, \text{COLOR} : \text{RED}) \)

**Q2.** Find the department(s) that sell(s) an item(s) supplied by the supplier Parker.
Here the user fills in both the SALES and the SUPPLY Tables as follows.

<table>
<thead>
<tr>
<th>SALES</th>
<th>DEPT</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.TOY</td>
<td>PEN</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUPPLY</th>
<th>ITEM</th>
<th>SUPPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEN</td>
<td>PARKER</td>
<td></td>
</tr>
</tbody>
</table>

**Explanation:**

The example element PEN (linking variable) is included in both tables, implying if an item is sold by the department in question, that same item has to be supplied by Parker.

In linear form Q2 will be written as follows:

**Q2.** \( \text{SALES}(\text{DEPT} : \text{P.TOY}, \text{ITEM} : \text{PEN}) \)

**SUPPLY (ITEM : PEN, SUPPLIER : PARKER) **

If the user wishes to retrieve information from two or more tables in the form of a new relation (table), he first sets up a skeleton of the required table by specifying its attributes, and then fills in this skeleton table with linking elements. The assignment of a name to the new relation being outputted is optional. This is illustrated in the next query.

**Q3.** List the departments and their corresponding suppliers.

<table>
<thead>
<tr>
<th>SALES</th>
<th>DEPT</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOY</td>
<td>PEN</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUPPLY</th>
<th>ITEM</th>
<th>SUPPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEN</td>
<td>BIC</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPT</th>
<th>SUPPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.TOY</td>
<td>P.BIC</td>
</tr>
</tbody>
</table>

**Explanation:**

Here the user groups the two columns DEPT and SUPPLIER to form a skeleton of a table (which may or may not be identified by name). The example element PEN appearing in both the SALES and the SUPPLY tables indicates a natural join on the common domain ITEM.

**Q3 in linear form:**

**Q3.** \( \text{SALES}(\text{DEPT} : \text{TOY}, \text{ITEM} : \text{PEN}) \)

**SUPPLY (ITEM : PEN, SUPPLIER : BIC) **

**(DEPT : P.TOY, SUPPLIER : P.BIC) **

If the user wishes to save an intermediate relation, which he created from columns of existing ones, for use in subsequent queries, he must assign a name to this relation and precede it by the command 'SAVE.'.

**Q4.** Create an intermediate table DEPT-SUPP such that it contains all the departments and their corresponding suppliers.
SALES DEPT ITEM
TOY PEN

SUPPLY ITEM SUPPLIER
PEN BIC

SAVE DEPT-SUPP DEPT SUPPLIER
TOY BIC

Explanation:

The ‘SAVE.’ command will save this intermediate relation DEPT-SUPP. The user can now call this table by name and query it in the usual manner. A ‘P.’ in front of TOY & BIC would cause the system to print out the data (in addition to saving it), which would be the same as the table outputted in Q3. A second command, ‘REMOVE.’, is used in the same manner as ‘SAVE.’ to dispose of intermediate relations when no longer required.

Q4 in linear version:
Q4. SALES (DEPT:TOY,ITEM:PEN)
SUPPLY (ITEM:PEN, SUPPLIER:BIC)
SAVE.DEPT-SUPP (DEPT:TOY,SUPPLIER:BIC)

Q5. Find the name(s) of any employee(s) who earns more than his (their) manager(s).

EMP NAME SAL MGR DEPT
P. JONES >10K HARRIS
HARRIS 10K

ANS: NAME
LEWIS HOFFMAN

Explanation:

If HARRIS is an example of such a manager and if HARRIS earns 10K (as an example) then JONES is an example of an employee who earns more than 10K (indicated by the comparison operator “greater than”, or “>”), and therefore more than his manager. It should be noted that the order of the rows is immaterial.

The above examples briefly demonstrate the concepts of Query-by-Example. For more details and for reformulation of the queries in predicate calculus, see Reference 1.

Q5 in linear version:
Q5. EMP (NAME:P.JONES,SAL:>10K,MGR:HARRIS)
EMP (NAME:HARRIS,SAL:10K)

* The ‘SAVE.’ & ‘REMOVE.’ commands were not included in the original version of Query-by-Example. They will be added to the revised version which will be published soon.

OPERATIONS ON HIERARCHICAL STRUCTURES

In this section we assume that the user is familiar with the hierarchical data base model and views the data as a collection of logical hierarchical structures. He formulates his queries by filling in skeletons of the structure of the logical data base through a keyboard display with an example of a possible answer. These skeletons are initially displayed on the screen by entering their name, in much the same manner that skeleton tables are displayed in the case of a relational view.

The Query-by-Example operations are again introduced through illustrative examples, and a linear version of these examples is given for cases where a display facility is unavailable.

Let us consider a sample hierarchical data base taken from “An Introduction to Data Base Systems” by C. Date. Figure 3 illustrates the various data base record types of an educational system whose function is to run a number of training courses. Each course is offered at a number of different locations.

—For each course: course number (unique), course title, course description, details of prerequisite courses (if any), and details of all offerings (past and planned)
—For each prerequisite course for a given course: course number and title
—For each offering of a given course: date, location, format (for example, duration, full-time or half-time), details of all teachers, and details of all students
—For each teacher of a given offering: employee number and name
—For each student on a given offering: employee number, name, and grade

As shown in Figure 3, there are five types of segment: COURSE, PREREQ, OFFERING, TEACHER, and STUDENT, each one consisting of the field types indicated. COURSE is the root segment type; the others are dependent segment types. Each dependent

Figure 3—The education data base record type
has a parent—for example, the parent of TEACHER (and STUDENT) is OFFERING. Similarly, each parent has at least one child—COURSE, for example, has two.

Figure 4 illustrates specific instances (occurrences) of the data base. For example, COURSE M23 has three OFFERING occurrences, in Oslo, Dublin and Madrid. Madrid's OFFERING, in turn, has three STUDENT and one TEACHER occurrences.

Figure 4—Sample occurrence for the education data base

We now proceed with examples of queries on this data base, some of which are also taken from Reference 15.

**Q6. Get the dates of all the OFFERING occurrences where the location is Stockholm.**

The user fills in the skeleton of the logical data base as follows:

```
COURSE
COURSE #  TITLE  DESCRIPTION

OFFERING
DATE  LOCATION  FORMAT

STUDENT
NAME  GRADE
```

*Explanation:*

Since there are no conditions on the COURSE fields, they are either left blank or filled in with example elements (underlined); the same is true of the DATE and the FORMAT fields in the OFFERING segment. STOCKHOLM is, therefore, the only specified constant element. The function ‘P.’ under the DATE field indicates that we want the dates to be printed out.

Linear version:

**Q6.** COURSE

COURSE;OFFERING(DATE:P.,LOCATION: STOCKHOLM)

**Explanation:**

The segment name COURSE in the first row specifies the root of the logical data base. The segment name COURSE in the second row followed by a semicolon indicates that COURSE is a parent (not necessarily the immediate parent) of the OFFERING segment. The condition Stockholm Location and the function ‘P.’ are specified in the parentheses. It should be pointed out that since the system has the knowledge of the logical data base, in this case it is sufficient to simply put the query in the following short form.

OFFERING(DATE:P.,LOCATION:STOCKHOLM)

The system will be able to trace the OFFERING segment to its parent segment. We originally specified the COURSE segment for clarity, so that the user will get the feel for the hierarchical tree.

**Q7. List the names of all the students who achieved grade A where the offering location is Stockholm.**

```
COURSE
DESCRIPTION
LOCATION
FORMAT

STUDENT
NAME  GRADE
```

Linear version:

**Q7.** COURSE

COURSE;OFFERING,LOCATION: STOCKHOLM)

OFFERING;STUDENT(NAME:P.ED,GRADE: A)

It should be noted that the sequence of the rows is immaterial. Furthermore, as the above examples illustrate, in this language, whether in the graphical or in the linear form, the user simply has to state what he requires to be retrieved; he does not have to be concerned with particular hierarchical sequences or hierarchical paths of the data base. In a procedural language, on the other hand, the user has to be concerned with hierarchical sequences of the data base, so that he can 'navigate' through the tree paths. As an example, let us reformulate this query in a simplified version of the DL/1 Language, which is the data sublanguage for IMS.
Q7 in DL/1 is classified as a sequential retrieval with a conditional segment search argument (SSA).

GU COURSE
OFFERING
,LOCATION = 'STOCKHOLM'
STUDENT (GRADE = 'A')
NSA GN STUDENT (GRADE = 'A')
go to NSA

Here the user has to be familiar with the IMS search paths in order to structure the query accordingly. In the cases where the search is forward, the formulation of the query is relatively easy; however, it becomes much more complex when a backward search path is required. One can say that here "the user is forced to devote time and effort to solving problems which are introduced by the model and are not intrinsic to the questions being asked."15

Q8. Get the teacher's name of an offering (of any course) attended by student 183009.

COURSE

<table>
<thead>
<tr>
<th>COURSE #</th>
<th>TITLE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
</table>

OFFERING

| DATE | LOCATION | FORMAT |

TEACHER

<table>
<thead>
<tr>
<th>EMP#</th>
<th>NAME</th>
</tr>
</thead>
</table>
P.ED  |

STUDENT

<table>
<thead>
<tr>
<th>EMP#</th>
<th>NAME</th>
<th>GRADE</th>
</tr>
</thead>
</table>
183009|

Explanation:

Here the STUDENT occurrence (EMP# 183009) and the required TEACHER occurrence must be linked to the same OFFERING occurrence. This link is indicated by a line connecting the three segments and can be thought of as having two relations, OFFERING-STUDENT relation and OFFERING-TEACHER relation, linked by example elements as follows.

OFFERING

<table>
<thead>
<tr>
<th>DATE</th>
<th>LOCATION</th>
<th>FORMAT</th>
<th>EMP#</th>
<th>NAME</th>
</tr>
</thead>
</table>

TEACHER

<table>
<thead>
<tr>
<th>EMP#</th>
<th>NAME</th>
</tr>
</thead>
</table>
P.ED  |

STUDENT

<table>
<thead>
<tr>
<th>EMP#</th>
<th>NAME</th>
<th>GRADE</th>
</tr>
</thead>
</table>
183009|

Note that the x y z links indicate that the TEACHER and the STUDENT have the same parent OFFERING.

Q8 in DL/1 is classified as use of command code F query.

GU COURSE
NO GN OFFERING
GNP STUDENT (EMP# = '183009')
if not found go to NO
GNP TEACHER*F

Explanation:

In IMS, certain command codes are introduced to give the language more flexibility for such cases as when the user wants to search the path backwards. Q8 is an example of such a case. If we used GET NEXT WITHIN PARENT (GNP) TEACHER (without the code "*F"), the answer would be "segment not found", since the search would be in a forward direction and teachers precede students in the hierarchical sequence. What is required is a means of stepping backwards under the current parent and this is accomplished by the code "*F".

When we wish to relate segment occurrences to two or more occurrences of the same parent we have to qualify these parents accordingly. This is illustrated in the next example. Consider the following data base.

MGR

EMP

NAME |
| SAL |

Q9. List the employees' names who earn more than their managers (same as Q5.).

MGR

EMP

NAME |
| SAL |
P. JONES >10K

NAME |
| SAL |
SMITH 10K

Explanation:

Here a separate skeleton is used for each occurrence of the MGR. The line connecting SMITH and JONES indicates that SMITH is JONES' manager. In order
to find SMITH's salary we have to search for SMITH in an occurrence of the EMP segment; however, SMITH in the EMP segment has a manager of his own (say LEWIS) as indicated by the connecting line.

Linear version:

Q9. MGR (NAME :SMITH)
MGR (NAME :SMITH) ; EMP (NAME :P.JONES, SAL : >10K)
MGR (NAME :LEWIS)
MGR (NAME :LEWIS) ; EMP (NAME :SMITH, SAL : 10K)

Explanation:

The manager qualified by the name SMITH is the parent of the employee whose name we wish to be printed out, and the manager qualified by the name LEWIS is the parent of the employee SMITH.

Note that here we cannot dispose of parent qualifications, without which the system will not be able to relate an employee to his manager.

The next query is an example of relating two different Logical data bases. Consider the following two data bases.

Q10. List the department names that sell items supplied by Parker (same as Q2.).

Linear version:

Q10. DEPT (NAME :P.TOY)
DEPT; ITEM (NAME :PEN)
SUPPLIER (NAME :PARKER)
SUPPLIER; ITEM (NAME :PEN)

MAPPINGS

In this section we demonstrate how the user can map a hierarchical (H) view to a relational (R) view and vice versa. An example of a query involving both views simultaneously is given.

a. Mapping a hierarchical to a relational view

Consider the following data base.

Q12. Convert the above data base to a relation called EMP1 and save it.

Linear version:

Q12. MGR (NAME :ED)
MGR; EMP (NAME :TOM, SAL :10K)
SAVE.EMP1 (NAME :TOM, SAL :10K, MGR-NAME :ED)
If we just want a printout in the form of a relation, without being concerned with saving it, we fill in the following skeleton.

<table>
<thead>
<tr>
<th>NAME</th>
<th>SAL</th>
<th>MGR-NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>P. TOM</td>
<td>$10K</td>
<td>P. ED</td>
</tr>
</tbody>
</table>

Linear version:

(NAME: P. TOM, SAL: $10K, MGR-NAME: P. ED)

This is consistent with the examples in Q3 and Q4.

b. Mapping a relational view into a hierarchy

(R \rightarrow H)

Consider the SALES relation in Figure 2.

Q13. Construct and save a hierarchical view from the SALES relation, where the root segment is DEPT.

Linear version:

Q13. SALES (DEPT: TOY, ITEM: PEN)

SALES SAVE.DEPT

SAVE.DEPT (NAME: TOY)

DEPT; ITEM (NAME: PEN)

Consider the following data base which combines hierarchical and relational views.

<table>
<thead>
<tr>
<th>MGR</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JONES</td>
</tr>
</tbody>
</table>

SALES | DEPT | ITEM |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TOY</td>
<td>PEN</td>
<td></td>
</tr>
</tbody>
</table>

EMP | NAME | SAL | DEPT |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P. ED</td>
<td></td>
<td>TOY</td>
<td></td>
</tr>
</tbody>
</table>

The following query is formulated using both views simultaneously.

Q14. Find the employees who are managed by Jones and work in a department that sells pens.

Linear version:

Q14. MGR (NAME: JONES)

MGR; EMP (NAME: P. ED, DEPT: TOY)

SALES (DEPT: TOY, ITEM: PEN)

PERFORMANCE CONSIDERATIONS

Where the performance of the system is a major factor, and it is particularly so when the data base is large, one can improve the performance by making the user cognizant of the fact that certain paths are more efficient than others; this will probably encourage him to use a more efficient approach. For example, take the TYPE relation that has ITEM and COLOR as column names and consider the following two queries.

Q15. Find the red items.

Q16. Find the colors of ink.

The above two queries are completely symmetric (which is, of course, one of the merits of the relational approach).

We now view the above data base as hierarchical with the ITEM segment as the root.
Q15 and Q16 will look as follows.

<table>
<thead>
<tr>
<th>Q15</th>
<th>ITEM</th>
<th>Q16</th>
<th>ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NAME</td>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td>F.</td>
<td></td>
<td>F.</td>
<td></td>
</tr>
<tr>
<td>RED</td>
<td></td>
<td>RED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COLOR</td>
<td></td>
<td>COLOR</td>
</tr>
</tbody>
</table>

If the user is told that queries involving an "upward" search path, such as Q15, are less efficient than queries requiring a "downward" search path, such as Q16, he will then try to keep the number of queries formulated in the form of Q15 to a minimum. If many queries inevitably involve the same upward path, he may improve the performance by constructing another logical data base which is the inverse of the first, thus diverting the search to a downward path.

This is just a simple example of how the user can participate in the performance. It remains to be seen whether this concept can be generalized to a practical size data base, involving queries of a practical complexity.

CONCLUSIONS AND REMARKS

In this paper we demonstrated that Query-by-Example operations are independent of the view of the data base.

The feature of printing-out part of the data (such as, 'only one element' or 'the first five') is yet to be embedded in Query-by-Example. At present the 'P.' operator means 'print all the elements'. It may be possible to specify the number of elements to be printed out following the 'P.'. Example: 'P.(1)PEN' will print out one item, and 'P.(5)PEN' will print the first five items, etc.

The feature of ordering the outputted set is also not as yet included in Query-by-Example.

ACKNOWLEDGMENTS

To Peter deJong I am most grateful for his helpful discussions, suggestions, and interest throughout the preparation of this paper.

I also wish to thank my wife Rosy for correcting the manuscript and for many helpful suggestions.

REFERENCES
