PUC

Series: Monographs in Computer Science
and Computer Applications

Nº 2/72

# A TAXONOMIC INFORMATION HANDLING SYSTEM

by

Flávio Pereira de Sousa

Computer Science Department - Rio Datacenter

Pontificia Universidade Catolica do Rio de Janeiro
Rua Marquês de São Vicente, 209 — ZC-20
Rio de Janeiro — Brasil

# A TAXONOMIC INFORMATION HANDLING SYSTEM

Flávio Pereira de Sousa

Member of The Applications
Division of Rio Datacenter

PUC/RJ

This paper will be published in the Proceedings of The Congresso Nacional de Processamento de Dados (São Paulo - Outubro - 1971)

Series Editor: Prof. A. L. Furtado February/1972

## ABSTRACT

This paper consists of a description of an information retrieval system designed for taxonometrics purposes, but with a reasonably wide range of applications.

Simple implementation and use characterize this system that can be used in batch or tele-processing by means of an English-like language and which grammar and syntax can be quickly learned.

The user can easily update the data banks,which makes the system totally independent of specialized personnel.

## 1. INTRODUCTION

Libraries, schools and other research environments are starting to face the problem of handling large amounts of information efficiently without the use of integrated information retrieval systems.

The design of an efficient system of general use doesn't look feasible for the near future, so the option has been to use systems that, although not of totally generalized use, are efficient for a large set of applications.

This system has been used with success in areas like biology and agronomy and can easily be used in other areas like the partial automation of libraries.

## 2. GENERAL CHARACTERISTICS OF THE SYSTEM

This is a subset of the system which was initially developed and programmed at University of Colorado [1] for the CDC 6400. It was next converted to the IBM 360 model 67 at Washington State University [2], where it was subject to improvements.

It requires about 200 kbytes to be compiled and about 70 kbytes to run the load module.

The data bank can be stored on tape or some faster access secondary storage device depending on system use, batch or teleprocessing.

The system has the capability of storing any compatible amount of data being the only bound, the size of the secondary storage available. The fact is due to the philosophy of the system of bringing sequentially into core, segments of data from the secondary storage to a fixed size area in the main core.

Queries to the system may contain boolean operators such as _and_, _or_ and _not_, besides allowing the use of parenthesis as in an arithmetic expression.

## 3. SYSTEM STRUCTURE

The data through which the search will be made is stored in a _data bank_ which will be made up by a set of _items_. These items are the units of information, i.e., in some subjects, an item may mean groups, populations or any other concept being studied.

The item is made up by its identification and of a sequence of informations that will characterize it. The informations are particularizations of what is named _descriptors_. It is by means of the descriptors that we can compare the characteristics of the various items in the data bank.

For example, we can consider COLOR as a descriptor, in this case, two items will be similar if they have the same color and different otherwise.

The descriptor is used as a means of comparison, this descriptor will be subject to particularizations and these will be the descriptor states. So if in the previous example, the various colors will be descriptor states of the descriptor: COLOR, through which well be able to distinguish the items.

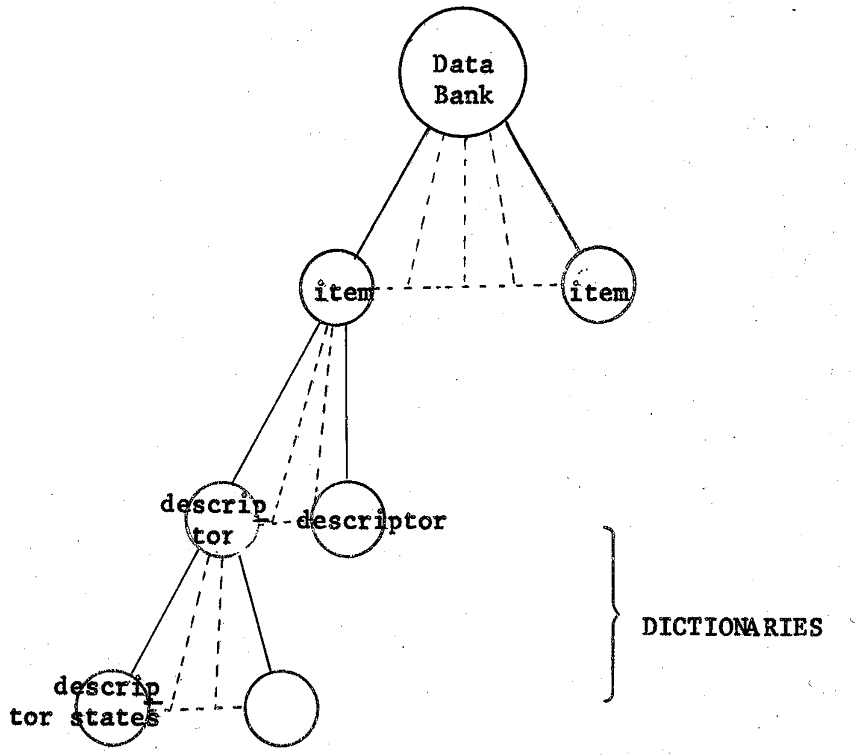


Fig.1 - Hierarquical representation of elements of the system

The set of descriptors and descriptor states will make up the dictionaries so that the user will be provided with a vocabulary, which he'll use, along with some keywords expressed in TAXIR grammar and syntax, to communicate with the system.

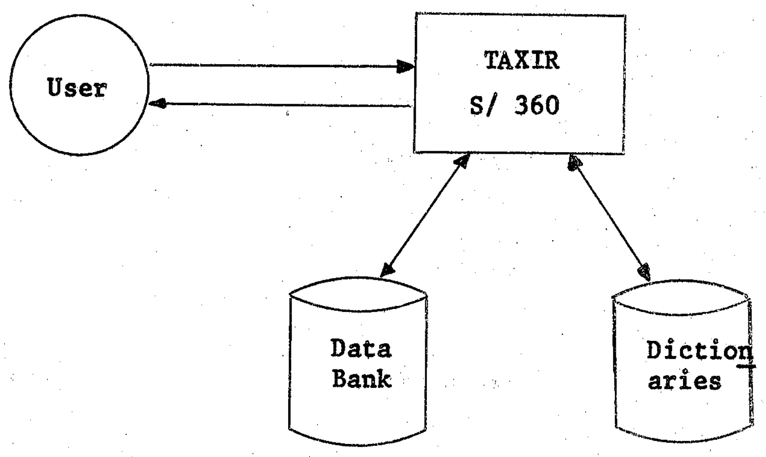


Fig. 2 - General view of the system

The vocabulary mentioned above is the <u>control vocabulary</u> which will avoid the user's asking the system questions which won't be understood by it.

Through the information in the Accessioner Module, the system will learn the order in which the data will be input, in which form it'll be input, i.e., if coded or not and how big is the variety of descriptors expected.

Summarizing the above in BNF we'll have:

```
<data bank>  ::= <item> | <item>  <data bank>
<item>  ::=  <id> , <descriptor state> ,........, <descriptor state> *
<decriptor> ::=  <name> | <name>  <descriptor>
<descriptor state> ::= <code> | <name>  |'b'| UNKNOWN
<control vocabulary> ::= <descriptor>  <descriptor state>  ,..........
                     ....., <descriptor>   <descriptor state>
<id>  ::= <unsigned integer>
<code> ::=  <letter string> |  <unsigned integer>
<unsigned integer> ::= <digit>| <unsigned  integer>   <digit>
<name> ::= <letter>  | <name> <letter>  | <name>   <digit>
<letter string> ::= <letter>  | <letter string>   <letter>
<letter> ::=  A | B |............|Z
<digit >::=  0|1|2  ........|9
```

a. The TAXIR language

The basic commands are:

I. <u>ID</u>: It's the command through which the user identifies the data bank in use. The comments included in the identification will be associated with each answer received from the system.

<u>format</u>:

ID: Any comments

For instance we can include in the comments, creation date of the data bank, where it was created, who created it, etc....

## II. Accessioner Module Statement:

This statement allows the user to inform the Accessioner Module in which way the data will be structured. Names which will be used as descriptors and ordinality of the descriptor in the set of descriptors, will be learned by this module.

Format:

```
ACCESSIONER MODULE: descriptor (sequence number,type,
                    exponent), descriptor (........),
```

Where sequence number will indicate the relative position of the descriptor, its ordinality in the set of descriptors of the items included in the data bank.

Type will indicate the way the descriptor states will be input, if by means of a name, a code or a combination of both.(NAME, CODE, BOTH).

The power of two obtained through the specified exponent,will indicate the area to be reserved for the corresponding descriptor states, i.e., if one expects a total of five states for a given descriptor, then, as $2^2 < 5 < 2^3$ , 3 will be the selected exponent. For instance if the descriptor COLOR is the 9th descriptor and one expects a variety of 10 colors and the descriptor states are described by names we'll have:

ACCESSIONER MODULE: ....... , COLOR(9,NAME,4),......

because $2^3 < 10 < 2^4$, there'll be room for 16 different states of the descriptor: COLOR. This "room" will correspond to rows of the binary matrix to be described later.

<u>DEFINE ITEM</u>: This command presents the data bank to the system and associates an item to the descriptor states related to it.

format:

```
DEFINE ITEM
id, ds, ds, ds, .......... , ds*  (up to 44 ds's in
                                   the present imple
                                   mentation)
```

where ds stands for descriptor state.

For example:

```
002521, ABC, 1960, GN*
013342, XYZ, PRESENT, BU*
```

Where the first number is an identification, the names that follow are different states of a descriptor that could for instance be variety names of specimens. The years, 1960 and Present could be considered the years the specimens were collected, represented in the case by a code or by a name, so the "BOTH" case. The third descriptor state could be considered a color code where GN would stand for green and BU for blue.

## IV. PRINT CONTROL VOCABULARY:

This command asks the system to provide the user with a listing of the control vocabulary available.

format

PRINT CONTROL VOCABULARY*

the answer would have the following format.

```
1. VARIETY NAME          (NAME)                  (descriptor)

2. YEAR COLLECTED        (BOTH)                  (descriptor)
   1. 1958
   2. 1960
   3. 1961
   4. 1962
   5. 1963                                (descriptor  states)
   6. 1964
   7. 1965
   8. 1966
   9. 1967
  10. 1968
```

11. 1969
12. 1970 (descriptor states)
13. PRESENT

3. COLOR (CODE) (descriptor)
   1. WE
   2. BL
   3. YE (descriptor states)
   4. GN
   5. BU
   6. RD

V. <u>END</u>:

This command stops the execution of the program or signifies the end of data for the data bank. Its function is determined in context.

<u>format</u>

END*

In the case where the END command is detected at the end of a data stream that will make up the data bank, the system shall verify the presence of other commands eventually issued by the user . If detected after any other command, it'll be understood as end of processing.

V. Read Data Bank:

Causes the system to bring to main core a data bank previously created

format

READ DATA BANK*

or

READ DATA BANK 1* presents to the user the characteristics of the data bank in use, such as number of items in it, etc......

It is worth mentioning that this command actually brings the dictionaries to core.Only after processing and verifying the valility of the next command to the system, the data transfer from the data bank in secondary storage to core, is actually performed, being this process transparent to the user.

VII. WRITE DATA BANK:

Copies to secondary storage, a data bank (or part of it) created in main core.

format

WRITE DATA BANK*(as in the READ DATA BANK command, it has the second option).

This command not only writes to secondary storage the updating to the data bank, but also updates the dictionaries.

## VIII. NUMBER OF DESCRIPTORS:

Informs the system how many descriptors are used in the description of the data bank.

format:

NO. OF DESCRIPTORS NN*

In our example in III, we would have: NO. OF DESCRIPTORS 3*, i.e., the items being considered will be described through 3 descriptors.

## b. System's Questioning Language

The queries to the system can be written in two different ways; in the answer of the type 1 query, only the identification number of the items which meet the conditions established in the query will show while the type 2 query will provide the user with descriptor states of the descriptors mentioned in the query besides the identification number of the item.

Type 1 query format:

```
<Query> ::= QUERY <noise>  WITH <descriptor> ,<logical
            expression>*
           |QUERY <noise>  HAVE <descriptor> ,<logical
            expression>*
<noise> ::= any string of characters not containing WITH,
            HAVE,*
<logical expression> ::=  <DS> | <RO> | <DS>   LO   <DS>
                          <LE>  LO  <LE>
<RO>  ::= FROM  <DS>  TO  <DS> | FROM   <LE>  TO   <LE>
whose <DS>   - descriptor state
      <LE>   - logical expression
      <LO>   - logical operator (AND, OR, NOT)
```

It should be noticed the "<noise> " is ignored by system. Its only purpose is to improve the readability of the query.

Example:

QUERY which specimens HAVE variety name, ABC or XYZ AND ( NOT (year of collection, FROM 1960 to 1970) OR color, GN)*

where we have:

| | |
|---|---|
| which specimens: | <noise> |
| variety name...: | <descriptor> |

ABC OR XYZ AND (NOT (year of collection,FROM 1960 TO 1970) OR color, GN)*..: <LE>

where:

|  |  |  |
|---|---|---|
| year of collection | : | <descriptor> |
| FROM 1960 TO 1970 | : | < RO > |
| color | : | <descriptor> |
| GN | : | <descriptor state> |

Type 2 Query format:

```
<QUERY> ::= QUERY <noise> : <descriptor>  FOR <noise> WITH
            <descriptor>  <LE> *
            QUERY <noise> : <descriptor list> FOR  <noise>
            HAVE  <descriptor> <LE> *
<descriptor list> ::= <descriptor> | <descriptor list>   ,
                      <descriptor>
```

Example:

```
QUERY list : year of collection, variety name   FOR
             specimens which HAVE color, YE*
```

where we have:

|  |  |  |
|---|---|---|
| list | : | <noise> |
| year of collection,variety name | : | <descriptor list> |
| specimens which | : | <noise> |
| color | : | <descriptor> |
| YE | : | <descriptor state> |

C. Errors:

The system has the capability of detecting the errors made by the user and provide him with messages informing the error code. By means of a table the user can determine the cause of the error and take the appro priate action.

## 4. PHILOSOPHY OF THE SYSTEM:

The system makes use of a series of tables to store the descriptor states and indices to the rows of the binary matrix* which contains the information about the items.

This matrix has in each of its rows, binary information about the descriptor states, i.e., for a given data bank, one row of the matrix will have for, instance, information about the year of items included in the data-bank.

For example, if the year being considered is 1960 ( a descriptor state of the descriptor above mentioned), the row of the matrix will have a 1(one) in the columns corresponding to the items collected during that year, a zero otherwise. This can be easily visualized in Fig. 3 below.

* the data-bank is stored in binary matrix form.

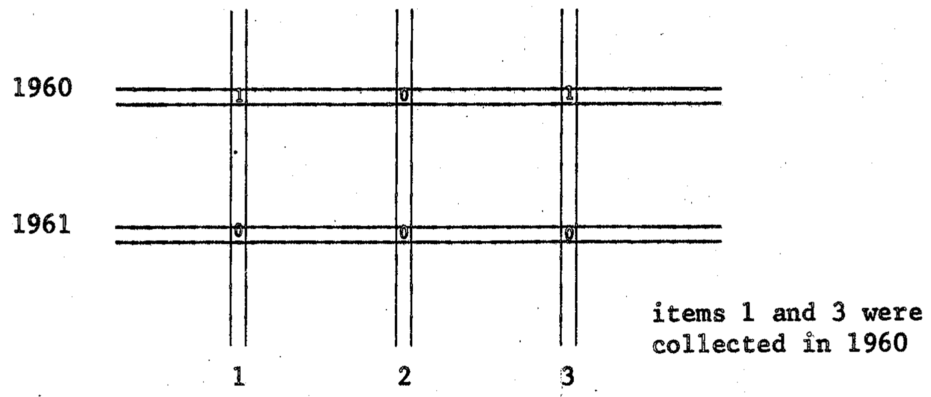


Fig. 3 - Information as stored in the binary matrix

Where items 1, and 3 will satisfy the condition, i.e., they were collected during the year of 1960.

As previously seen, the system learns which are the descriptors that will be describing the items in the data bank , through the Accessioner Module. These descriptors will be kept in a table that will have pointers to another table which will be filled with the descriptor states.

The linking among the tables and the binary matrix (da̲ ta bank) can be more easily visualized through the Fig. 4 below.

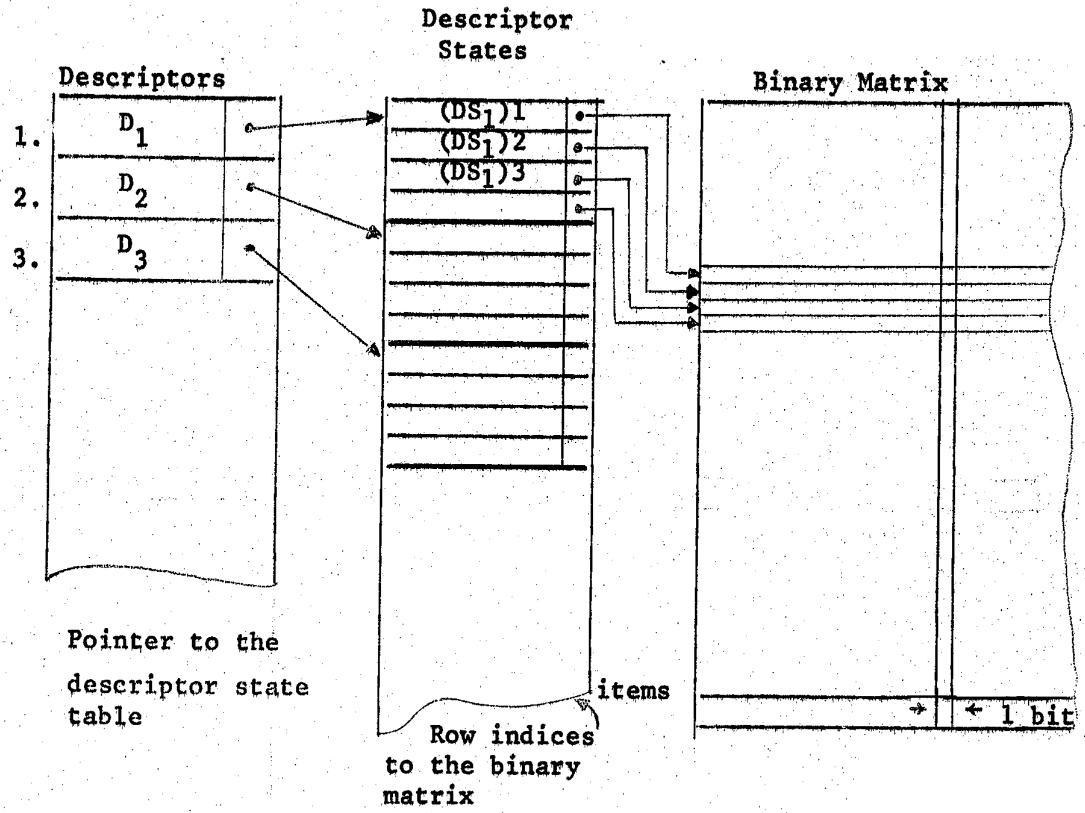


Fig. 4 - Linking among the tables and matrix

When a query is issued to the system, the processing of such query will convert it to polish notation and by using a push down stack (PDS), the descriptor states will be then transformed in row indices of the binary matrix after the searches are performed in the dictionaries.

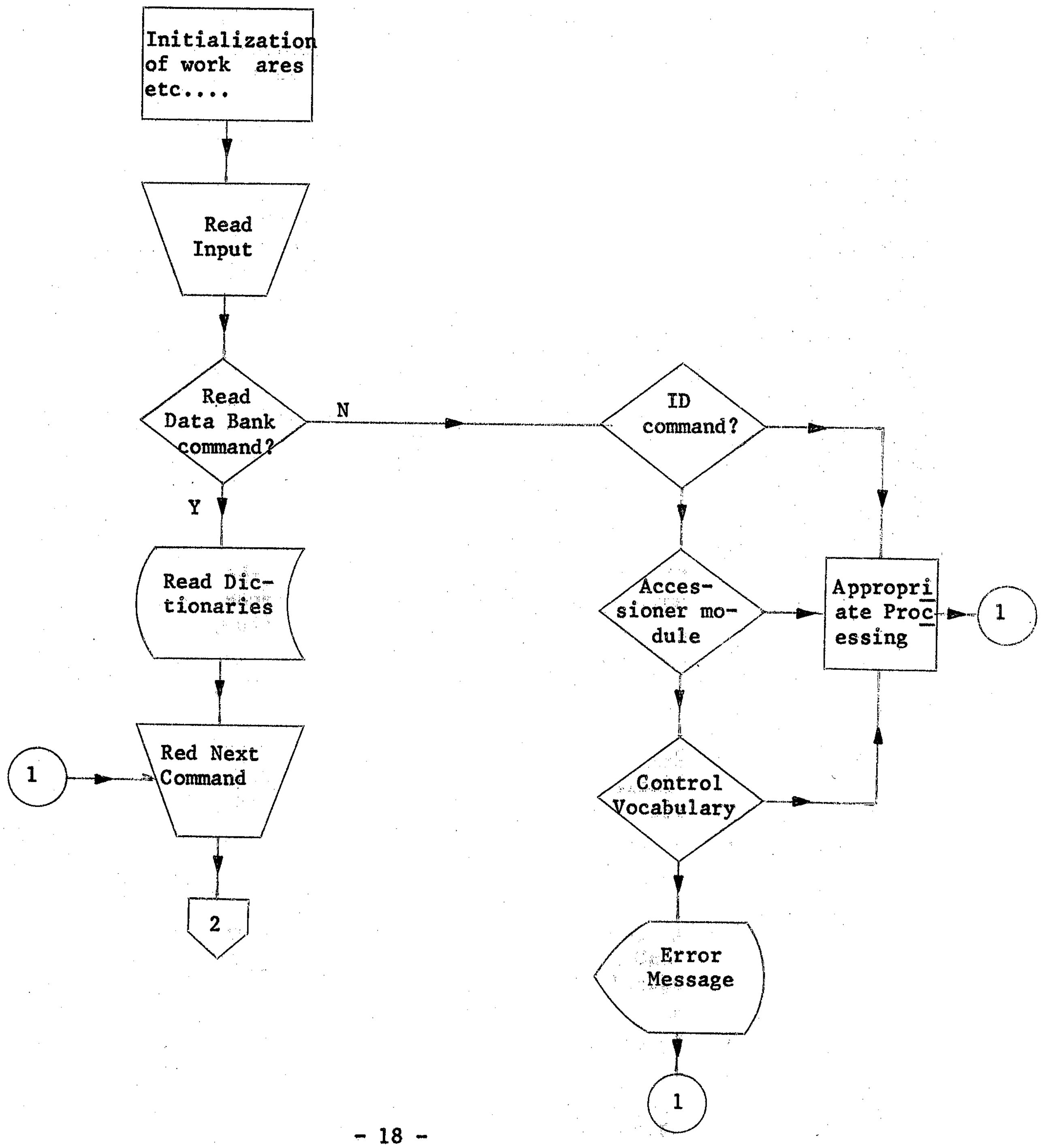
In the type 1 query, where the descriptor states which will characterize the item listed in the answer are included, a table look-up on the descriptor state table, will provide the row index to the binary matrix which contains the information about the state.

For the type 2 query, in initial look-up must be made on the descriptor table. Next, a sequential search on the descriptor state table is made to get the row indices of the binary matrix. In this case, the table indices must be saved until the end of the search, so that the states related to the selected items can be retrieved later.

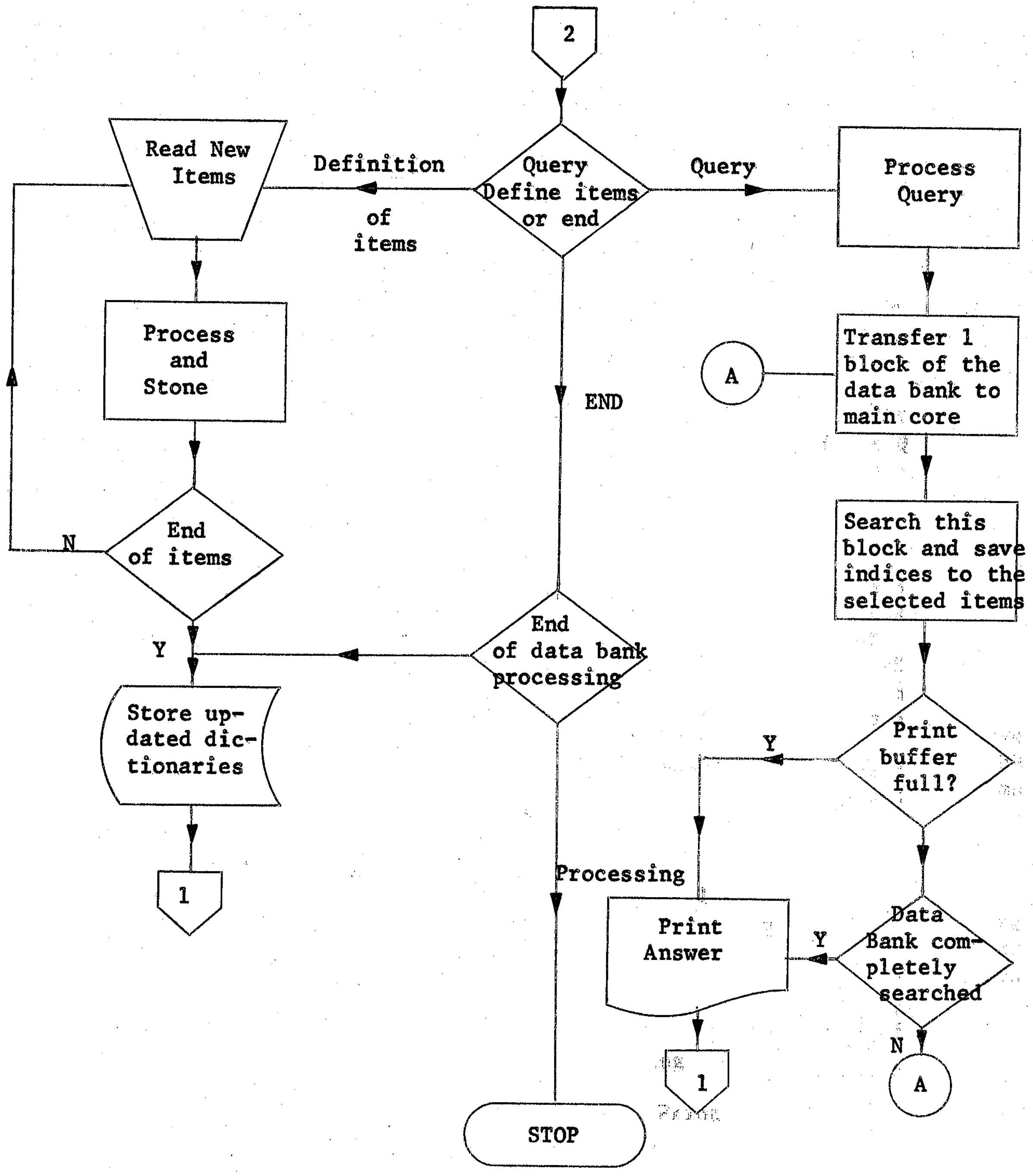
After the row indices to the binary matrix are obtained, logical operations such as AND, OR and NOT are performed among these lines according to the contents of the PDS where the query previously processed is kept.

The items selected after performing the operations mentioned, are kept in a binary form,in an extra row of the binary matrix which will be swept later for obtaining the final answer.

## 5. GENERAL FLOWCHART OF THE SYSTEM



GENERAL FLOWCHART OF THE SYSTEM


2
Query Define items or end
Definition of items
Query
Read New Items
Process and Stone
End of items
N
Y
Store up-dated dic-tionaries
1
END
End of data bank processing
Processing
STOP
Process Query
Transfer 1 block of the data bank to main core
A
Search this block and save indices to the selected items
Print buffer full?
Y
Data Bank com-pletely searched
Y
N
Print Answer
1
A

## 6. DATA BANK UPDATE

The updating of the data bank is achieved through the DEFINE ITEM statement, in the same way that we create the data bank. The updating of the dictionaries is performed automatically by the system, as new items are added to the data bank, i.e., new descrip tor states are included in the descriptor state table.

## 7. CONCLUSION

We presented some examples for taxonomic applications for the described system, other applications such as the partial au tomaton of libraries can be easily achieved [3], considering for instance: author, title, publisher edition, volume, etc......., as descriptors of items which will be books. As previously mentioned, the system in its present version has capability of handling up to 44 descriptors.

The batch mode which characterizes the way the described system works, leads to a response time, which is dependent not only on the structure of the system itself but also on the computing sys tem load and the level of priority assigned to these types of jobs.

This system is implemented in various universities in the United States and countries like Israel, Italy and Germany have shown interest in the implementation of the system in their universities.

## BIBLIOGRAPHY

1. Rodgers, D. J.; Brill, Bob; Esterbrook, George - TAXIR - Taximetrics laboratory - University of Colorado (*), 1968

2. Konzak, C. F.; Walden, W. E.; Souza, F.P. - Problems and Progress in the Management of Information on Genetic Resources - in print.

3. Dutton, R.; Massara, M.; Walden, W. E. - TAXIR, an information Retrieval Program - Time Slicer - Washington State University, V.3, N.4 - September, 1969

(*) Potential users of this system are suggested to consult the Taximetrics Laboratory about the data analysis.