Copyright © 1973 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 52, No. 10, December, 1973
Printed in U.S.A.

Information Management System:

The Off-The-Shelf System—A Packaged Information Management System

By L. E. HEINDEL and J. T. ROBERTO

(Manuscript received October 5, 1972)

The Off-The-Shelf System (OTSS) is a packaged information management system for hierarchical data bases. OTSS provides, without computer programming, processes to enter and alter data in such a data base, do complex retrievals of data from the data base, and specify various security mechanisms to limit access to, or alteration of, a data base. OTSS also provides a mechanism for extending the available processes on a project-by-project basis. OTSS has been implemented using MASTER LINKS and the NATURAL DIALOGUE SYSTEM.

I. INTRODUCTION

The Off-The-Shelf System (OTSS) is a packaged information management system for hierarchical data bases. Earlier work done by Sinowitz[1] was aimed at providing information retrieval capabilities for a specific hierarchical data base. OTSS was designed to operate on any hierarchical data base regardless of its structure and regardless of the data fields stored in the data base.

Retrieval processes are available to print, alter, rank, plot, distribute, compute statistics, and perform regression analysis of data. These processes are specified to OTSS in a key-word English-like language in an interactive dialogue environment. OTSS allows for simple alteration of the retrieval process from request to request by selective replacement, deletion, or addition of statements to the dialogue description of the process to be performed.

As part of the package, OTSS provides a data-base-independent security mechanism. This mechanism allows a data base administrator to restrict access or alteration of a data base and use of certain process and language facilities in OTSS on a user-by-user basis. A user can be restricted to a logical hierarchical subsection of the data base and to only certain data items stored in that subsection. The user can be further restricted to using only certain processes of OTSS such as printing, ranking, or altering processes, but not plotting or distributing.

In addition, OTSS provides the ability to extend the processing capabilities of the system by allowing a programmer to add new processes to the system on a project-by-project basis. The processes so installed are available to the project installing them, and do not become a permanent part of OTSS.

OTSS was implemented using the MASTER LINKS[2] data base management system and the NATURAL DIALOGUE SYSTEM,[3] a system for designing and implementing interactive computing languages in a dialogue environment.

II. SYSTEM DESIGN CRITERIA

As a packaged information management system, OTSS was developed to satisfy certain basic design criteria. The two primary design criteria are hierarchical independence and field independence, i.e., the structure and specific content of the data base. In establishing these as design criteria the retrieval processors, security mechanisms, and associated language specifications are written to operate on any hierarchical data base regardless of its logical structure and regardless of the data types of the fields of the data base. The system provides these capabilities by making use of the information contained in the driving tables of the data base system, MASTER LINKS, used to implement OTSS. These driving tables describe the logical structure and data fields of a given data base.

Another design criterion of OTSS was to provide the user with the ability to specify a generalized retrieval function in the sense of Ref. 2, and a comprehensive set of data base processors to operate on such functions. In general, a retrieval function is defined as a combination of data base fields, constants, and previously defined retrieval functions using the standard arithmetic, relational, and logical operators. Through a keyword-oriented language, a user of OTSS can specify an arbitrary retrieval function, and, for example, request the system to print its values, rank its values, or plot its values by the values of another retrieval function. The user can also specify a series of logical retrieval functions which are to be used to selectively delimit the search of the data base during the retrieval process.

Directed output of any retrieval process is a fourth basic design criterion. Through the retrieval language, a user can direct the output of any retrieval process to any external device including line printer, card punch, magnetic tape, disk, or console.

As a final design objective, OTSS offers the programming audience the ability to extend the processing capabilities of the system on a project-by-project basis. A programmer can write a project-dependent processor and "install" such a processor into the OTSS environment. Once installed, this processor is available only to the project installing it, and does not become a permanent part of OTSS.

In the following sections we shall describe in more detail what we mean by a hierarchical data base, examine typical uses of retrieval functions, and present the definition of the load and retrieval phases of OTSS.

III. HIERARCHICAL DATA BASES

As discussed in greater detail in Ref. 2, a *hierarchical data base* is a directed tree which is rooted at one entity (node). At each entity in the tree is stored a set of fields. Two entities belong to the same *group* if the set of fields stored at one entity is identical to the set of fields stored at the other. Two entities belonging to the same group are at the same depth in the tree (i.e., connected to the root entity of the tree by paths of the same length). Also the ancestor groups of one entity belonging to a group must be identical to the ancestor groups of any other entity belonging to the group.

Using the concept of groups, it is possible to represent the structure of a hierarchical data base as a rooted tree whose entities are the groups of the hierarchical data base placed in the tree analogously to the entities in the data base with respect to depth and connectivity. At each entity in the group tree are listed the fields stored at that group. In reference to the group tree, we say that a set of groups forms a chain if and only if for any G_i and G_j belonging to the set of groups, G_i is an ancestor or descendant of G_j. A group chain is complete if all the ancestors of every group on the chain are on the chain. Entity chains and complete entity chains are defined in an analogous way.

3.1 *Structure of a Sample Hierarchical Data Base*

As an example of the structure of a hierarchical data base, consider the group tree presented in Fig. 1. This hierarchical data base is rooted at the COMPANY group as there is only one company. We shall return to this sample data base in Section IX when we discuss examples of using the OTSS retrieval language.

IV. SPECIFYING THE STRUCTURE OF A DATA BASE

OTSS is independent of the hierarchical structure of the data base and the particular fields stored in the data base. OTSS obtains all its required information about the data base from a series of driving tables. These driving tables are produced using the BUILD facility of MASTER LINKS.[2]

BUILD allows the data base designer to completely specify the logical structure of the data base. The driving tables produced by BUILD are used by OTSS to determine the correctness of data loading and retrieval requests and by MASTER LINKS to be able to enter and access data in the data base.

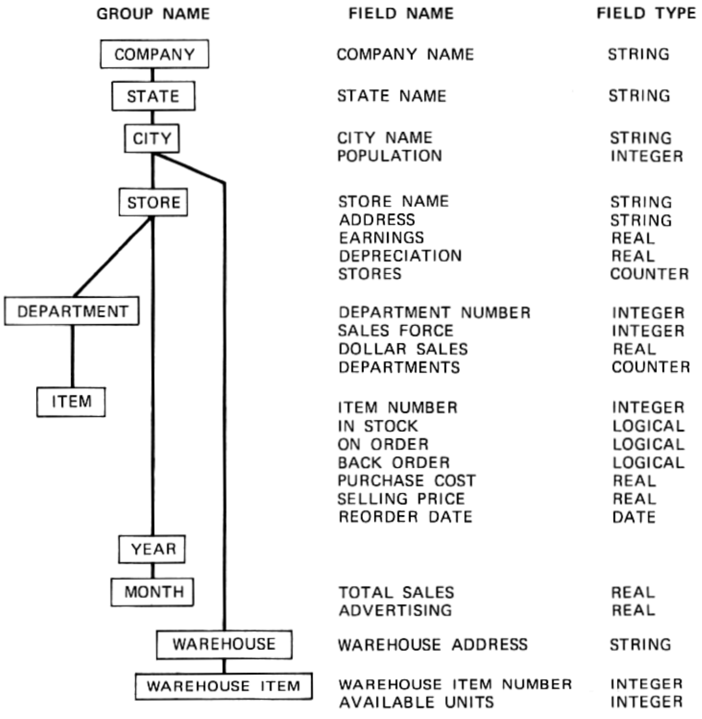
V. LOADING DATA INTO A DATA BASE

Once BUILD has been used by the data base designer to specify the logical structure of a hierarchical data base, it is ready to have data loaded into it. OTSS provides a facility, called LOAD, for bulk loading of data into the data base.

LOAD allows files of data to be sequentially loaded into a data base, i.e., LOAD does not provide for multiple concurrent updates of a data base. LOAD does provide a simple mechanism to restart a data load which was terminated abnormally due to machine failure or human error.

The file of data to be loaded using LOAD is logically divided into sections. Each section is identified by an integer number, called the *card type*, which must appear in columns 1 through 3 of each record of the section. More than one section in the data file may have the same card type. The data within a given card type section must be organized according to a specific card type definition, and it must be organized in the same manner for every section having the same card type.

The definitions of the various card types are defined once by the data base designer using the definition phase of LOAD. In a card-type definition, the designer indicates where, on the records of the specified section, the data values for particular fields can be found and the name of the entity where the data is to be loaded. Along with the field name, the user indicates which record of the section and which field (set of contiguous columns) of that record contains the value of



GROUP NAME	FIELD NAME	FIELD TYPE
COMPANY	COMPANY NAME	STRING
STATE	STATE NAME	STRING
CITY	CITY NAME	STRING
	POPULATION	INTEGER
STORE	STORE NAME	STRING
	ADDRESS	STRING
	EARNINGS	REAL
	DEPRECIATION	REAL
	STORES	COUNTER
DEPARTMENT	DEPARTMENT NUMBER	INTEGER
	SALES FORCE	INTEGER
	DOLLAR SALES	REAL
	DEPARTMENTS	COUNTER
ITEM	ITEM NUMBER	INTEGER
	IN STOCK	LOGICAL
	ON ORDER	LOGICAL
	BACK ORDER	LOGICAL
	PURCHASE COST	REAL
	SELLING PRICE	REAL
	REORDER DATE	DATE
YEAR		
MONTH	TOTAL SALES	REAL
	ADVERTISING	REAL
WAREHOUSE	WAREHOUSE ADDRESS	STRING
WAREHOUSE ITEM	WAREHOUSE ITEM NUMBER	INTEGER
	AVAILABLE UNITS	INTEGER

Fig. 1.—Structure of a sample hierarchical data base.

the field. Every section having the same card type must have the value for a specified field in the same field of the same record of the section as described in the corresponding card-type definition.

Thus the card-type definition specifies what data are contained in a section of a file of input data and where the data are to be loaded into the data base. When LOAD processes a file of data, it looks at columns 1 through 3 of the records to determine what card-type definition should be used for the section and applies the appropriate card-type definition to direct its data loading process.

VI. RETRIEVALS FROM THE SAMPLE DATA BASE

A simple form of a retrieval process on the sample data base is to extract the value of a single field at all its occurrences in the data base. For instance, one might wish to extract all the values of DOLLAR SALES in the data base. A more interesting case is to extract, along with the values of DOLLAR SALES, the corresponding values of DEPARTMENT NUMBER. If the department numbers were only unique within a store, one might wish to also extract the corresponding values of STORE NAME. Notice that due to the structure of our sample hierarchical data base, there is only one value of STORE NAME for each value of DEPARTMENT NUMBER and DOLLAR SALES, but there are several values of DOLLAR SALES and DEPARTMENT NUMBER for each value of STORE NAME.

Suppose now that one were interested only in extracting the value of DOLLAR SALES for a given department within a given store. This is similar to the first example of a retrieval process, except that the retrieval process would first delimit the search of the data base to a subtree of the data base consisting of the particular STORE entity and DEPARTMENT entity. To do this there is a directory into the data base which is based on the name of an entity or a chain of entity names. Having delimited the search of the data base to the particular store and department, the retrieval process can extract the one value of DOLLAR SALES contained in the delimited data base.

Some retrieval processes combine extraction based on known entity names and the values of fields stored in the data base. One might wish to extract the value of ITEM NUMBER for those items in one particular store which have SELLING PRICE greater than $9.00. In this case, the retrieval process would first delimit the search of the data base to the entity in the STORE group with the appropriate name and to all entities which are descendants of it. The retrieval process would then search through all entities in the ITEM group in the delimited part of the data base and extract the ITEM NUMBER for those items having SELLING PRICE greater than $9.00.

So far only examples of extracting the values of simple fields stored in the data base have been discussed. It is also possible to evaluate more complex retrieval functions as part of the retrieval process. A simple example would be to extract the value of SELLING PRICE/PURCHASE COST. This retrieval process creates a new pseudo-field at the ITEM group which is then extracted. A more complex example is the summing of all the values of DOLLAR SALES within a store. To evaluate this function, the retrieval process would have to extract the value of DOLLAR SALES for every DEPARTMENT entity under each STORE entity and then add them together. This process produces a new pseudo-field at the STORE group which is then extracted. An operation which raises the level, in the hierarchy, of definition of a field or expression is called a level-raising operation.

Summing is not the only level-raising operation which can be performed. Others are minimum, maximum, and average on numerical data and any, all, and none for logical data. For instance, one might wish to extract the DEPARTMENT NAME of all departments that have their minimum ratio of SELLING PRICE to PURCHASE COST less than 1. One might also wish to extract the DEPARTMENT NAMES as above, but only including in the level-raising operation items which have a selling price greater than $9.00.

We have now seen several examples of retrieval processes. All these retrieval processes are examples of a complete retrieval process which delimits the search of a data base by entity names, accepts or rejects entities by logical conditions, and evaluates complex retrieval functions including level-raising. We have only referred to extracting data from a data base and have not said anything about what should be done with the data once extracted. This was done intentionally to divorce the retrieval process from the displaying of the extracted data.

The retrieval language provides processes to print, alter, rank, plot, distribute, compute statistics, and perform regression analysis of extracted retrieval functions by using the appropriate keyword.

VII. SPECIFICATION OF RETRIEVAL FUNCTIONS

Retrieval functions are specified by combining data base fields, constants, and previously defined retrieval functions using the standard arithmetic, relational, and logical operators. The arithmetic operators defined on numeric data and their symbols are: addition (+), subtraction (−), multiplication (*), division (/), and exponentiation (↑). The relational binary operators defined for numeric data and date data and their symbols are: equal to (=), not equal to (¬=), greater than (>), greater than or equal to (>=), less than (<), and less than or equal to (<=). Relational binary operators defined for string data and their symbols are: equal to (=) and not equal to (¬=). Logical binary operators defined for logical data and their symbols are: logical and (AND) and logical or (OR).

The unary operators available in the OTSS retrieval language are of two types: those which operate on a single value and those which operate on a set of values. Unary operators operating on a single value are: unary plus (+), unary minus (−), logarithm to the base 10 (LOG10), logarithm to the base e (LOGE), e raised to a power (EXPF), absolute value (ABSF), sine (SINF), and cosine (COSF) for numeric data and not (NOT) for logical data.

Unary operators which work on a set of values are the level-raising operators. Level-raising operators are of the following two forms:

```
                lr      field     PER     gn
or
           GLOBAL     lr      field     PER     gn,
```

where lr is any level-raising operator and gn is any group name. The field must be defined at a group which is a descendant of the group following the PER. The set of values that the level-raising operator operates on are the values of the field in entities which are descendants of an entity in the group following PER. The level-raising operator takes the set of values and computes a single value which depends on the level-raising operator. The level-raising operators for numeric values are: sum (SUM), minimum (MIN), maximum (MAX), and average (AVG). The level-raising operators for logical values are: logical any (ANY), logical all (ALL), and logical none (NO).

If the level-raising operator is not preceded by the literal GLOBAL, any entity restrictions that have been applied to all groups between the group following PER down to and including the group of the field are evaluated and entities and their descendants are rejected for which the entity restriction is not satisfied. An entity restriction is a retrieval function whose type is logical. An entity in the group of the retrieval function is accepted or rejected if the logical function evaluates to TRUE or FALSE respectively. The remaining set of entities at the group of the field are then combined using the level-raising operator. If the literal GLOBAL is present, all entity restrictions below the group following PER are ignored.

Unary operators can be nested without parentheses and are evaluated from right to left. Parentheses can be used to cause evaluation of a retrieval function to occur in other than the normal order of evaluation.

The groups of all fields in a retrieval function must form a group chain. A retrieval function has a definition group associated with it. The definition group of a retrieval function is the group of maximum depth of the groups of fields not operated upon by a level-raising operator and the groups given in level-raising operators. In this manner, retrieval functions are made to be single-valued for each entity in the definition group.

VIII. THE RETRIEVAL PROCESS

The retrieval process used by OTSS can be described by considering the steps involved to evaluate any one retrieval function for all entities at which it is defined in a subset of the data base. To describe the retrieval process, let us assume we wish to evaluate a retrieval function, f, defined at some group, G, at every entity of G in a subset of the data base. The steps of the retrieval process are as follows:

Step 1: Entity Selection Based on Entity Chains

(a) Delimit the search of the data base by constructing an access tree based on any specified entity chains.

Step 2: Entity Selection Based on Entity Restrictions

(a) Start at the root of the access tree constructed in Step 1.
(b) Select the next entity in a depth-first, left-to-right manner. If all entities have been selected, the retrieval process is finished.
(c) If an entity restriction has been placed on entities in the group of the entity obtained by Step 2b, apply it. If the result is "reject," reject the entity and all its descendants and go back to Step 2b. If f is defined at the group of the entity, evaluate it using Step 3. Upon completion of the evaluation or if f is not defined at the entity, go to Step 2b.

Step 3: Function Evaluation

(a) If f is a field, retrieve its value; or if f is a constant, use its value.
(b) If f is a level-raised field without the GLOBAL prefix, apply all entity restrictions to entities in all groups from G down to and including the group of the level-raised field and ignore all entities and their descendants for which the entity restriction is not satisfied. Perform Step 3 on each remaining entity of the group of the field and combine the results according to the appropriate level-raising operator.
(c) If f is a level-raised field with the GLOBAL prefix, perform Step 3 on all descendant entities at the group of the field and combine the results according to the appropriate level-raising operator.
(d) Combine values obtained in Steps 3a, 3b, and 3c using appropriate operators.

The above described retrieval process can be expanded to evaluate several different retrieval functions during one pass through the data base. It can be seen that all the retrieval processes in Section VI can be formulated in terms of the general retrieval process given above.

Hence the user of OTSS need only learn how to specify the retrieval process and the form of output desired. Detailed algorithms for implementing the retrieval process are described in Appendix A.

IX. LANGUAGE EXAMPLES

Having presented the descriptions of retrieval functions and the retrieval process, let us proceed to examine several examples of the OTSS retrieval language. The OTSS retrieval language is a keyword-oriented, English-like language which provides the necessary input to the retrieval process and means of specifying the output format. The language contains FOR and IN statements which are used to specify subtree delimiting of the search of the data base; WHEN statements for specifying logical entity restrictions; LET statements for specifying retrieval functions; and various output specification statements such as PRINT, RANK, PLOT, etc. A complete description of these and other auxiliary statements are described in Appendix B.

To begin our examples, let us print the names of all the departments in the store at 19 Fifth Ave., New York City, New York. The following statements accomplish this:

PRINT DEPARTMENT NAME: FOR 19 FIFTH AVE, NEW YORK CITY, NEW YORK: GO:

Now to print only those departments with dollar sales greater than \$5000 we need only enter the following statements:

WHEN DEPARTMENT HAS DOLLAR SALES $>$ 5000: GO:

as OTSS remembers the last occurrence of each keyword statement entered.

To print the department which has the highest dollar sales in the store at 19 Fifth Ave., we would enter the statements:

WHEN DEPARTMENT HAS DOLLAR SALES = GLOBAL MAX DOLLAR SALES PER STORE: GO:

Suppose we now wished to print the ratio of dollar sales of those departments having dollar sales less than \$5000 to the dollar sales of the entire store. We would enter the statements:

PRINT DOLLAR SALES PER STORE/GLOBAL DOLLAR SALES PER STORE: WHEN DEPARTMENT HAS DOLLAR SALES $<$ 5000: GO:

If we wished to do the above request for all stores, not just the one at 19 Fifth Ave., we would enter:

```
DELETE FOR: GO:
```

And suppose finally we wanted to do the same request for only those stores whose dollar sales are greater than \$1,000,000 but less than \$5,000,000. We would enter:

```
WHEN STORE HAS GLOBAL DOLLAR SALES PER STORE
> 1000000 AND GLOBAL DOLLAR SALES PER STORE
< 5000000: GO:
```

To save a small amount of typing, we could have entered the following:

```
LET X = GLOBAL DOLLAR SALES PER STORE: WHEN
X > 1000000 AND X < 5000000: GO:
```

X. REPORT FACILITY

The REPORT statement is the general interface to extend the processes available in the OTSS retrieval language on a project-by-project basis. One can write a process in FORTRAN which can be installed into OTSS to be referenced by some report name using the REPORT statement. The syntax of the REPORT statement is the keyword "REPORT" followed by a report name optionally followed by a list of retrieval functions separated by commas. If the process requires retrieval functions to be passed into it as parameters, they follow the report name in a manner analogous to the retrieval function list in the PRINT statement. The process so installed is available to the project installing it, and does not become a permanent part of OTSS.

XI. SYSTEM SECURITY

The SECURITY statement is a command in the language which enables a data base administrator to define, interrogate, and remove security information for his user audience. The types of security which are available to a data base administrator are environmental, language, and data base. These security mechanisms may be used by the administrator to restrict access or alteration of his data base and use of facilities in the retrieval language on a user-by-user basis.

11.1 *Environmental Security*

The first type of security which may be specified is the environmental security. Environmental security is used by the administrator to specify legal users of the system. This security mechanism uses two pieces of information: a sign-on key passed into the system and password information input by the user. The data base administrator specifies valid combinations of sign-on keys and optional password information for each potential user of the system. When a user initially enters the retrieval environment, the system will interrogate the sign-on key typed in by the user to see if it is valid. If the sign-on key is not in the list of valid sign-on keys provided for by the administrator, the system will terminate the session. A valid sign-on key will cause the system to prompt the user for the password information (if this sign-on key requires a password). If the password is incorrect, the session is terminated.

11.2 *Language Security*

The data base administrator has the ability to limit use of certain facilities in the retrieval language. This type of security is called language security. In order to specify language security the administrator defines one or more statement restriction classes. A statement restriction class is a set of statements in the retrieval language which is *not* available to a set of users of a data base. The administrator would then indicate, on a user-by-user basis, which statement restriction class pertains to each user. If a user attempts to use a statement in the retrieval language which is a member of his statement restriction class, the system will output a message indicating that the statement in question is not available for use by him.

11.3 *Data Base Security*

The final type of security provided for by the system is data base security. Data base security can be subdivided into two parts: field security and access tree security. Field security is specified in a manner similar to the language security specification. The data base administrator defines one or more field restriction classes. A field restriction class is a set of fields in the data base which is *not* accessible to one or more users of the system. A field restriction class may be restricted on a read/write basis or on a write basis only. After defining the sets of field restriction classes, the administrator indicates, on a user-by-user basis, which restriction class pertains to each user. If a user attempts to retrieve or modify the value of a field which is a member of his field restriction class, the system will output a message indicating that the field in question is not available for access or alteration.

Access tree security is used to restrict users to a logical hierarchical subsection of a data base. For each potential user of his system, the data base administrator may specify a corresponding USER statement. The USER statement has the same form as the FOR and IN statements described in Appendix B, and is used to delimit the search of the data base.

When a user initially enters the retrieval environment, after passing the environmental security phase, the USER statement corresponding to that user will be processed to delimit the search of the data base. If the user tries to access a portion of the data base outside of the logical hierarchical subsection specified in the USER statement, the system will output a message indicating this as an illegal action. Note that the USER statement cannot be modified or deleted by a user.

XII. CONCLUSION

OTSS is an information management system designed to be independent of the structure or content of any specific hierarchical data base. OTSS provides a simple keyword-oriented, English-like language for specifying the retrieval of values of complex retrieval functions and the alteration of data in a data base. In addition, OTSS provides a means of loading data into a data base and specifying various forms of security on the data base and the use of statements in the language.

APPENDIX A

The retrieval process (RP) has as its inputs a set of complete access lists, a set of retrieval functions, and a set of entity restriction functions. These inputs completely specify the semantics of the retrieval process.

RP applies the algorithm TB (Tree Building) to construct a subtree of the data base over which the retrieval process will be performed. RP applies the algorithm GTP (Group Tree Pruning) to make up a list, R, whose entry for each group, g, is "referenced" if g is the definition group of a retrieval function or an ancestor of the definition group of a retrieval function.

RP uses the algorithm ACTION to create a list, A, of selector directions for each group in the hierarchy. The entry in A for group, g, is "down" if it is an ancestor of the definition group of a retrieval function and is "right" otherwise.

After having applied TB, GTP, and ACTION, RP proceeds to select the entities of the subtree (using the algorithm GEN) in a left-to-right, depth-first manner. The algorithm GEN only returns entities to RP for which a retrieval function may have to be evaluated or for which an entity restriction function may need to be evaluated and which is the group of a retrieval function or an ancestor of a group of a retrieval function. Hence, in this manner, no extraneous entities are selected.

Whenever an entity is returned to RP by GEN, RP determines if there is an entity restriction function to be evaluated at this entity. If there is, it is evaluated. If the entity restriction function evaluates to false, the action input to GEN is set to "right" and GEN is applied to select the next entity.

Should there be no entity restriction function to be evaluated, or should it evaluate to true, RP examines the list of retrieval functions to be evaluated and evaluates those defined at the group of the current entity. RP then iterates the whole procedure until all the entities on the subtree have been generated.

More formally the following algorithms define the functions of RP, TB, GTP, ACTION, and GEN.

Retrieval Process (RP)

Input: Complete entity access lists: $e_1, e_2, \cdots, e_n$.
Retrieval functions: $f_1^{g_1}, f_2^{g_2}, \cdots, f_n^{g_n}$.
Entity restriction functions: $b_1^{g_1}, b_2^{g_2}, \cdots, b_n^{g_n}$.

Output: Values of retrieval functions: $f_1^{g_1}, f_2^{g_2}, \cdots, f_n^{g_n}$.

Step 1: $T \leftarrow TB(e_1, e_2, \cdots, e_n)$ which builds a tree, T, the subtree of the data base that the retrieval process is to be applied to.

Step 2: $R \leftarrow GTP(f_1^{g_1}, f_2^{g_2}, \cdots, f_n^{g_n})$ which builds a list, *R*, containing one entry for each group in the hierarchy. The entry for a group is marked "referenced" if the group is one of the g or one of its ancestors and is marked "unreferenced" otherwise.

Step 3: $A \leftarrow ACTION(f_1^{g_1}, f_2^{g_2}, \cdots, f_n^{g_n})$ which builds a list, A, containing one entry for each group in the hierarchy. The entry for a group is marked "down" if the tree traversal action to be performed is "down" and is "right" otherwise.

Step 4: CE $\leftarrow$ root entity of the retrieval tree, T.

Step 5: Examine the list of entity restriction functions to see if any $b_i^{g_i}$ is defined at the group of the current entity, CE. If none, go to Step 7.

Step 6: Evaluate $b_i^{g_i}$. If the value is FALSE, go to Step 11.

Step 7: Any more $f_i^{g_i}$ to be evaluated? If none, go to Step 9.

```
Step  8: Evaluate $f_i^{g_i}$, output result and go to Step 7.
Step  9: If there are no more entities on T to be generated, then exit.
Step 10: CE ← GEN(T, R, A($g_i$), CE) which generates the next
         entity on T, then go to Step 5.
Step 11: If there are no more entities on T to be generated, then exit.
Step 12: CE ← GEN(T, R, "right," CE), then go to Step 5.
```

Tree Building (*TB*)

Input: Complete entity access lists: $e_1, e_2, \cdots, e_n$.
Output: The retrieval tree, T_1.

```
Step  1: $T_1$ ← null tree.
Step  2: If there are no more $e_i$, go to Step 5.
Step  3: Construct tree, $T_2$, consisting of the entities on the access
         list, $e_i$.
Step  4: $T_1$ ← $T_1$ union $T_2$, go to Step 2.
Step  5: Make $T_2$ a copy of $T_1$.
Step  6: If there are no more unexamined entities on $T_2$, exit.
Step  7: e ← next unexamined entity on $T_2$.
Step  8: Examine each group which is a descendent of the group of e
         to see if there exists an entity on $T_2$ which is a descendent
         of e. For each group in which this is not true, put all entities
         in the data base on $T_1$ which are descendents of e. Go to
         Step 6.
```

Group Tree Pruning (*GTP*)

Input: Retrieval functions: $f_1^{g_1}, f_2^{g_2}, \cdots, f_n^{g_n}$.
Output: A list, R, containing one entry for each group in the hierarchy. The entry for a group is marked "referenced" if the group is one of the g_i or an ancestor of one of the g_i and is marked "unreferenced" otherwise.

```
Step  1: Initialize the entries in R for each group in the hierarchy to
         "unreferenced."
Step  2: If there are no more $f_i^{g_i}$, exit.
Step  3: g ← $g_i$.
Step  4: If the entry in R for group g is "referenced," go to Step 2.
Step  5: Set the entry in R for group g to "referenced."
Step  6: If g is the root group, go to Step 2.
Step  7: g ← father of g; go to Step 4.
```

Generating Action (ACTION)

```
Input:   Retrieval function: $f_1^{g_1}$, $f_2^{g_2}$, $\cdots$, $f_n^{g_n}$.
Output:  A list, A, containing one entry for each group in the hierarchy.
         The entry for a group is marked "down" if the tree traversal
         is down, and "right" otherwise.

Step 1: Initialize the entries in A for each group in the hierarchy to
        "right."
Step 2: If there are no more $f_1^{g_i}$, then exit.
Step 3: $g \leftarrow g_i$.
Step 4: If the entry in A for group g is "down," go to Step 2.
Step 5: If g is the root group, go to Step 2.
Step 6: $g \leftarrow$ father of g.
Step 7: If the entry in A for group g is "down," go to Step 2.
Step 8: Set entry in A for group g to "down," go to Step 5.
```

Tree Generation (GEN)

```
Input:   Retrieval tree, T; the list R of referenced and unreferenced
         groups; and the action, A, either "right" or "down," and the
         current entity, CE.
Output:  CE the next entity to be processed by the retrieval process.

Step 1: If the action, A, is "right," go to Step 4.
Step 2: Find the leftmost entity on T, LME, which is a descendent
        of CE and for which the entry for the group of CE in list R
        is marked "referenced." If there are none, go to Step 4.
Step 3: CE $\leftarrow$ LME, then exit.
Step 4: If CE has no brother entity to the right, go to Step 6.
Step 5: CE $\leftarrow$ next brother of CE to the right, then exit.
Step 6: Find the leftmost group, g, which is on the same level as the
        group of CE for which the entry in the list R is marked
        "referenced" and for which there exists an entity on T which
        has not previously been processed. If none exists, go to
        Step 8.
Step 7: CE $\leftarrow$ leftmost entity of group, g, which has not yet been
        generated; then exit.
Step 8: CE $\leftarrow$ father of CE; go to Step 4.
```

APPENDIX B

OTSS Retrieval Language Statements

To make the description of the OTSS retrieval language statements more readable, the following notations are used:

(a) Capitals and special symbols are literals in the language.

(b) Lower case include:

f	—any retrieval function
str	—any non-null string of alphanumeric characters
num	—any number
gn	—the name of a group
null	—a null character string
stmt	—the name of a statement in the retrieval language.

(c) Square brackets imply that the constructs within the brackets are alternatives starting from the top line down. One item from the vertical list of alternatives must be selected.

$$\text{LET str} = \text{f}:$$

The LET statement is used to create additional fields which are not stored in the data base. The field so created may be used in any other statement or retrieval function.

$$\begin{bmatrix}\text{FOR}\\ \text{IN}\end{bmatrix} \text{e-list}_1;\ \text{e-list}_2;\ \cdots\ \text{e-list}_n:$$

The notation, e-list, indicates a list of entity names separated by commas. Each e-list represents an entity chain. The combination of all entity chains specifies an access tree. This access tree is used to select the subset of the data base over which the retrieval search will take place.

$$\text{WHEN} \begin{bmatrix}\text{gn HAS}\\ \text{null}\end{bmatrix} \text{f}:$$

The WHEN statement specifies an entity restriction function defined at the group, gn, which delimits the search of the data base during the retrieval process. If a WHEN condition is defined for a group, retrieval will take place from a entity within that group only if the entity restriction function evaluates to TRUE. If the entity restriction evaluates to FALSE, the entity (and all its descendents) will be ignored during the retrieval process.

$$\text{PRINT f}_1,\ \text{f}_2,\ \cdots\ \text{f}_n:$$

The PRINT statement specifies a tabular printout of the values of the individual retrieval function.

$$\text{DISTRIBUTE f}_1 \text{ BY f}_2:$$

The DISTRIBUTE statement specifies a tabular histogram with f_1

as the ordinate and f_2 as the abscissa.

$$\text{BETWEEN num}_1 \begin{bmatrix} \text{TO} \\ \text{AND} \end{bmatrix} \text{num}_2 \begin{bmatrix} \text{IN STEPS OF num}_3 \\ \text{ISO num}_3 \\ \text{null} \end{bmatrix} :$$

The BETWEEN statement specifies the range and cell intervals to be used by the DISTRIBUTE process.

CUMULATIVELY:

The CUMULATIVELY statement may be used with the DISTRIBUTE process to alter the distribution to produce cumulative values in each of the defined cells.

CHART:

The CHART statement may be used with the DISTRIBUTE process to specify bar chart output.

RANK f AT gn:

The RANK statement specifies to rank in descending order (largest to smallest) the individual values of f within each entity of the group gn, and displays the results in tabular form.

INVERSELY:

The INVERSELY statement specifies to the RANK process to invert the order of the RANK output.

$$\text{KEEPING} \begin{bmatrix} \text{THE} \\ \text{null} \end{bmatrix} \begin{bmatrix} \text{LARGEST} \\ \text{HIGHEST} \\ \text{SMALLEST} \\ \text{LOWEST} \end{bmatrix} \text{num}:$$

The KEEPING statement is used to specify to the RANK process to rank the "num" largest or smallest values of the rank function at each entity in the rank group.

CARRYING ALONG $f_1, f_2, \cdots f_n$:

The CARRYING statement may be used to specify to the RANK process to "carry along" the values of other retrieval functions and have them displayed as part of the RANK output.

PLOT $f_1, f_2, \cdots, f_{n-1}$ BY f_n:

The PLOT statement specifies an X-Y point plot with the values of

f_1 through f_{n-1} as the ordinate and f_n as the abscissa.

$$\begin{bmatrix}\text{X-AXIS}\\ \text{Y-AXIS}\end{bmatrix} \begin{bmatrix}\text{BETWEEN}\\ \text{FROM}\\ \text{null}\end{bmatrix} \text{num}_1 \begin{bmatrix}\text{AND}\\ \text{TO}\end{bmatrix} \text{num}_2\text{:}$$

The X-AXIS and Y-AXIS statements must be used to specify to the PLOT process to specify the origins and ranges of the independent and dependent fields.

$$\text{STATISTICS } f_1, f_2, \cdots f_n\text{:}$$

The STATISTICS statement requests a set of standard statistics to be produced for each function and the results pιinted in tabular form.

$$\text{REGRESS } f_1 \text{ BY } f_2, f_3, \cdots f_n\text{:}$$

The REGRESS statement specifies to perform a multiple linear regression analysis of the function f_1 (dependent field) by the functions f_2 through f_n (independent fields).

$$\text{ALTER field TO f:}$$

The ALTER statement is used to permanently change the value of the field to the value of f.

$$\text{INTERACTION} \begin{bmatrix}\text{BRIEF}\\ \text{DETAIL}\\ \text{VERIFY}\end{bmatrix}\text{:}$$

The INTERACTION statement specifies to the ALTER process a level of verification required by the user when altering a datum value.

$$\text{REPORT str} \begin{bmatrix}, f_1, f_2, \cdots f_n\\ \text{null}\end{bmatrix}\text{:}$$

The REPORT statement specifies to pass control to an application dependent process identified by the string, str.

$$\text{TITLE str}_1\text{! str}_2\text{ !}\cdots\text{! str}_n\text{:}$$

The TITLE statement specifies to any process to print lines of text centered at the top of the output of the process.

$$\text{PLACES num:}$$

The PLACES statement is used to control the number of decimal places displayed for real-valued functions.

$$\text{OUTPUT TO str:}$$

The OUTPUT statement specifies to any process to direct its output to a specific output device.

$$\text{DELETE} \begin{bmatrix} \text{WHEN FOR gn}_1\text{, gn}_2\text{, } \cdots \text{gn}_n \\ \text{stmt} \\ \text{ALL} \end{bmatrix} :$$

The DELETE statement specifies to remove a statement or set of statements which are currently active.

$$\text{DEFINE var}_1\text{, var}_2\text{, } \cdots \text{var}_n :$$

where var_1 through var_n are the names of created fields previously defined through the use of the LET statement. The DEFINE statement will *permanently* save the name and definition of each of the created fields mentioned in the DEFINE list.

$$\text{UNDEFINE field}_1\text{, field}_2\text{, } \cdots \text{field}_n :$$

where field_1 through field_n are the names of fields which were permanently created through the use of the DEFINE command. The UNDEFINE statement specifies to remove the names of the permanently created fields from the list of all possible fields accessible through the retrieval language.

DETAIL:

The DETAIL statement specifies to automatically recap the current state of the dialogue when a process is executed.

RECAP:

The RECAP statement specifies to display the current state of the dialogue.

INPUT FROM str:

The INPUT statement causes OTSS to accept input from a previously prepared file identified by str.

$$\text{SAVE} \begin{bmatrix} \text{IN} \\ \text{null} \end{bmatrix} \text{str} \begin{bmatrix} \text{WITH GO} \\ \text{null} \end{bmatrix} :$$

When the SAVE statement is given by the user, the system writes the current state of the dialogue on to the file identified by str.

DATABASE str:

where str is the name of another data base. The DATABASE statement allows the user to switch from one data base to another from within OTSS.

ERASE str:

The ERASE statement causes the disk file identified by str to be erased.

VOCABULARY:

The VOCABULARY statement specifies to print out the entire list of keywords and their associated synonyms available in the OTSS retrieval language.

RETURN:

The RETURN statement is used to return control to the operating system level.

STOP:

The STOP statement will disconnect the user from the time-sharing system.

$$\text{SECURITY} \left\{ \begin{array}{ll} \text{CREATE} & \left[\begin{array}{l} \text{str} \\ \text{null} \end{array}\right] \\ \text{INTERROGATE} & \left[\begin{array}{l} \text{ALL} \\ \text{user}_1,\ \text{user}_2,\ \cdots \text{user}_n \end{array}\right] \\ \text{REMOVE} & \left[\begin{array}{l} \text{ALL} \\ \text{user}_1,\ \text{user}_2,\ \cdots \text{user}_n \end{array}\right] \end{array} \right\} :$$

The SECURITY statement is used by a data base administrator to define, interrogate, and remove security information for his user audience.

GO:

The GO statement causes the last-mentioned process to be executed.

REFERENCES

1. Sinowitz, N. R., "DATAPLUS—a Language for Real-time Information Retrieval from a Hierarchical Data Base," AFIPS Conf. Proc., *32*, 1968.
2. Gibson, T. A., and Stockhausen, P. F., "MASTER LINKS—A Hierarchical Data System," B.S.T.J., this issue, pp. 1691–1724.
3. Puerling, B. W., and Roberto, J. T., "The Natural Dialogue System," B.S.T.J., this issue, pp. 1725–1741.