Operational software for restructuring network databases*

by DONALD E. SWARTWOUT, MARK E. DEPPE and JAMES P. FRY

The University of Michigan, Ann Arbor, Michigan

ABSTRACT

A high-level "access path" approach to database restructuring is described and contrasted with the "elementary operations" approach taken by most restructuring systems. With the elementary operations approach, restructuring is viewed as a sequence of basic or "primitive" operations which manipulate a source database in order to convert it into a target database. In the access path approach, restructuring is seen as the process of accessing a body of information represented by the source data, and constructing the target database representation of the same information. While the elementary operations approach is useful for restructuring hierarchical databases, it does not generalize well for networks. The access path approach is better-suited to the complex structures possible in network databases.

The access path approach permits the specification of complex restructuring transformations in terms of application-oriented concepts such as access strategies and selection criteria. A non-procedural Network Restructuring Language (NRL) based on this approach is presented, and an example of its use in restructuring is given. The architecture of an NRL-driven Restructurer for network databases is described.

INTRODUCTION

Due to the fact that the database design process is essentially an opaque art, an important tool needed by a Database Administrator is a generalized reorganization facility. With such a facility the designer can adapt existing database structure to conform to new information or processing requirements. For example, a new entity or relationship could be added to the database structure to satisfy a new information requirement, or a new access path could be added to satisfy a processing requirement.

Although some reorganization schemes exist in database management systems, they deal primarily with the physical organization of data, e.g., performing garbage collection, changing the blocking factors, and to some extent, modifying the accessing mechanism. To our knowledge no generalized reorganization facility exists in any database management system and, in particular, we are unaware of a restructuring facility which would provide the capability to change the logical structure of the database.

In the current state of the art, the reorganization facility has developed only within the context of a Data Translator. As described in References 2 and 6, a Data Translator is a generalized software system that accepts as input a source database, descriptions of the source (input), target (output) databases and the mapping between the source and target logical structures. Its output is the reorganized target database produced automatically by the generalized software from the input description. The execution of the translator invokes three major modules—Reader, Restructurer, and Writer. While the function of the Reader and Writer modules can be classified primarily as physical reorganization or reformatting, the primary purpose of the Restructurer module is to logically reorganize the database or restructure it. Birss and Fry described the physical reformatting capabilities of the Reader and Writer through the logical restructuring capabilities which were embodied in several prototype implementations. In this paper we focus on the development of a generalized restructuring capability for the network class of logical structures.

What is restructuring?

The purpose of restructuring is to transform the logical structure of a database in response to new information or processing requirements. Navathe and Fry developed a categorization of restructuring operations for the hierarchical class of logical structures. Three fundamental logical structure modifications were defined—Naming, Relation, and Combining—which were refined into several lower level restructuring operations. Figure 1 illustrates a few of these hierarchical restructuring operations. The first part, Figure 1a depicts a hierarchical relationship among PRESIDENTS, SPOUSES and CHILDREN. The uppermost box...
or record type,** PRESIDENTS, contains the names of the presidents of the United States. The second level record type, SPOUSES, is related to PRESIDENTS through the set type ‘Married to’ and contains the names of all the spouses for each president. Thus, the set is a one-to-many mapping over two record types. In a similar fashion, the third level record type, CHILDREN, is related to SPOUSES and contains the names of the children born to each SPOUSE.

One restructuring operation is partitioning in which the occurrences of a record type are divided into two or more distinct record types based upon the value of one or more data items. This operation is illustrated by the source and target logical structure of Figure 1 where the source record type CHILDREN has been partitioned into two record types, SONS and DAUGHTERS, based on the data item ‘sex.’ Notice that the data item ‘sex’ has been eliminated in the logical structure of Figure 1, since this information is now represented by the target logical structure. The dual restructuring operation of partitioning is merging. The merging operation consolidates occurrences from two or more record types into a single record type, often adding a data item to the record type to preserve information previously represented by the logical structure. Merging is illustrated by the source logical structure and the target logical structure of Figure 1, where the occurrences of the two record types, SONS and DAUGHTERS, have been merged into a single record type, CHILDREN. Note that the terms ‘source’ and ‘target’ are relative and depend on the direction of the restructuring transformation. Consequently, Figure 1 represents the source logical structure for the partitioning example, and the target database for the

** We follow the terminology of DBTG.*
† For a more complete description of the Presidential Database see Reference 10.
merging example. Notice the item type "sex" has been added to contain the information previously represented semantically by the source logical structure of Figure 1. Another form of restructuring involves the compression of two or more hierarchical levels into one. This example is illustrated by the source and target logical structure of Figure 1 where the two source records, SPOUSES and CHILDREN, have been compressed into a single record type, SPOUSE-CHILDREN. On the record occurrence level, this operation would be accomplished by replicating the associated SPOUSE record occurrence for every CHILD record occurrence. The dual operation, expansion, expands one level of hierarchy into two or more by factoring out selected data items. This procedure is illustrated in Figure 1 where the SPOUSE-CHILDREN record type has been factored into two levels.

Network restructuring

Although a complete categorization of such operations has been established for the hierarchical class of structures, restructuring operations are not nearly as easy to classify in the network class of logical structures due primarily to the variety and complexity of the structures involved. For example, an important network restructuring operation, changing the implementation of many-to-many relationships, is illustrated by the restructuring transform of Figure 2. The source logical structure represents information on students, their class standing, all courses taken by each student, and the final grade received in each course. The target logical structure represents exactly the same information except that the association between students, course, and grades is achieved using a LINK record type. This example is typical of the restructuring operations which may be posed in a network environment, those which involve several record and set types.

Another example, which might enhance processing, would be the addition of indexing sets or record types. This could be achieved by migrating the data item "class" to the record type CLASS-STANDING, which would serve to segment the students into graduate, undergraduate, special, foreign exchange, etc. (Figure 3). Notice that this operation is actually expansion performed on a hierarchical substructure of the target structure of Figure 3, namely the single record type STUDENTS.

Yet another important network restructuring capability is the ability of the restructuring process to extract not only data which exists explicitly in the source database, but also that which exists implicitly, i.e., the information which may be inferred from the actual source data. An example of the difference between implicit and explicit information is described in the hypothetical restructuring transform of Figure 4.

The example describes a source database containing two record types, PERSONS and LINK. The PERSONS record type contains name and sex item types, while the LINK record type contains no data but provides relationship information in conjunction with sets PARENTS and CHILDREN. The target structure also contains the PERSONS record type, and construction of target PERSONS records involves the extraction of data explicitly resident in the source file PERSONS record type. In contrast, target record types PARENTS and GRANDFATHERS do not correspond directly to any source file record type; they contain information which is represented implicitly in the source file. Needless to say, this transformation cannot be described by operations on hierarchical substructures.

In the next section, we review the previous work in developing hierarchical restructuring capabilities through specification of elementary operations. The third section comprises a discussion of the access path approach necessary to achieve a network restructuring capability, and the fourth section describes a Network Restructuring Language based on this approach. The implementation of the Restructurer is discussed in the fifth section and the paper concludes with our observations on building and using restructurers.

THE ELEMENTARY OPERATIONS APPROACH

An interesting analogy exists between restructuring systems and high-level query systems. In fact, one can consider a query as a restricted restructuring transformation in which the target is not a database but some other form of
Efficient, easy-to-use query languages for the hierarchical class of logical structures have been built through the use of elementary operations and have been in existence for some time. These systems allow a user to specify a sequence of elementary operations on hierarchical structures which will accomplish his query. Such languages are easy to learn and use since they employ a few rather simple operations to perform most queries to hierarchical structures.

Several research efforts have addressed the problems of restructuring and restructuring language specification from the elementary operations point of view. CONVERT, a high-level translation language, provides a generalized restructuring capability for hierarchical structures. The approach is based upon the concept of a "data form" in conjunction with a set of restructuring functions called "form operations." Shoshani takes a similar approach to restructuring whereby a set of "conversion functions" is used to specify restructuring operations.

In the elementary operations approach, the source database is considered to be a collection of data in a specific logical structure and format, while the target database is viewed as essentially the same data, but in a different logical structure and format. Restructuring, therefore, is considered to be the process of manipulating the source data to conform to the target logical structure and corresponding format. Consequently, restructuring research using this approach turns toward the development of low-level or "primitive" operations which transform occurrences of one logical structure to another. One advantage of such an approach is that the architecture of the restructuring software system is greatly simplified; it defaults to a set of low-level subroutines which correspond directly to the elementary operations. Thus, a restructuring specification...
consists of a sequence of primitives which may be directly converted to a sequence of subroutine calls which perform the actual restructuring. Unfortunately, the user is not shielded from any aspects of the restructuring; he must thoroughly understand the function of each operation in order to be able to use it. Consequently, the language, although high-level, still requires the user to treat restructuring essentially as a sequence of low-level steps.

Several other problems arise when the elementary operations approach is applied to restructuring network databases. In general, elementary operations are designed to operate on small, logical substructures consisting of one or two record types and a set, to produce a new, logical substructure. Due to the limited number of operations which may be defined, only a finite number of source substructures can be valid candidates for restructuring. The more complex the structure, the greater the probability that it will contain substructures which are not valid candidates for the set of available operations. Consequently, the restructuring capabilities of elementary operations decrease as the complexity of the structure increases. For this reason, elementary operations are not particularly well-suited to describing restructuring transformation upon complex network logical structures. In addition, a complex restructuring transformation (assuming that it may be performed by the elementary operations) will require the specification of a complex sequence of elementary operations which is difficult to analyze and to understand.

Another class of restructuring transformations that is difficult to accomplish with elementary operations is the extraction of implicit information. We have pointed out that such transformations (as in the example of Figure 4) cannot usually be described by a sequence of operations on hierarchical substructures. Furthermore, although they may be describable by sequences of elementary operations on network substructures, such descriptions are generally long and complicated. There is also reason to believe that, given time, designers of network databases will produce enough intricate methods of storing information implicitly to exhaust any set of elementary operations.

Following our analogy, it has been found to be extremely difficult to generalize the elementary operations of high-level query systems to network databases. The elementary operations approach to developing a query system for network databases tends to be less powerful and much more cumbersome than its hierarchical counterparts. In general, they suffer from the same problems cited above—limited allowable input structures, overly complex specifications, and difficulties with link records and other implicit information storage techniques. The high-level access path approach to restructuring grew out of attempts to develop a restructuring strategy more suited to network databases.

THE ACCESS PATH APPROACH

Following the current trend in host language database systems which process the network structure databases, we chose the high-level “access path” approach to restructuring. The source database is viewed as a body of information, some of which is represented explicitly by data, and some of which is represented implicitly, i.e., may be inferred from the data. Similarly, the target database is considered to contain a subset of information represented by the source; this data is created from information provided by the source data. Executing a restructuring transformation, then, is simply the process of traversing the source database to obtain the information needed to create the target database, and storing it according to the target logical structure. There are numerous consequences of this approach. For one, research in restructuring specification turns toward the development of restructuring specification languages based on the concepts of access strategies and selection criteria. Since this area is closely related to query language development, certain concepts from previous research in this area may be utilized. A second consequence of the access path approach is manifest in the actual restructuring algorithm development. Restructuring technology turns toward the development of algorithms which efficiently and exhaustively access the source database and perform tests upon the data, following externally specified access strategies and test criteria. This is a radical change from the elementary operations approach which tends to develop low-level subroutines.

The most important consequence of this approach is that it produces powerful generalized network restructuring capabilities. Since restructuring is viewed as an operation which accesses information (rather than manipulates data), the approach is unaffected by the logical structure of the database. This independence from the logical structure insures that essentially any database is a valid candidate for restructuring, regardless of the complexity of the logical structure (hierarchical, network, etc.). Furthermore, source and target logical structures need not even remotely resemble each other since the target is derived from information provided by the source rather than from the source structure itself. Also (unlike the elementary operations approach), implicit information may be extracted and restructured as easily as explicit information (as in Figure 4). Thus, explicit information may become implicit and vice versa.

Finally, the system is inherently simple. Assuming that the user has some familiarity with databases and applications, a restructuring specification language based on application-oriented concepts such as access strategies and selection criteria should be easily understood. Furthermore, since all restructuring operations are expressed as information accessing problems, confusion does not significantly increase as the complexity of the restructuring transformation increases. An algorithm designed to carry out this process is also simple and therefore, straightforward and dependable. All restructuring is performed in an identical sequence of steps, regardless of the particular transformation: (1) exhaustively access the source data according to the access specifications, (2) test data based on selection criteria, and (3) create target record occurrences containing the relevant data. For a more complete discussion of the theoretical foundations of this approach, see Reference 13.
NETWORK RESTRUCTURING LANGUAGE

Architecture

The architecture of the Network Restructuring Language is based upon the high-level specification of access paths. The major components of the language describe access strategies and selection criteria. Access strategies are described using an access path statement which specifies the traversal scheme required to obtain data for each target record type. The selection criteria establishes the source (and, indirectly the target) data requirements. It would be beyond the scope of this paper to describe the NRL in great detail. A complete language specification may be found in Reference 14. However, the major points of the language will be described.

The NRL is essentially a block-structured language in which each “block” contains a single target record statement. There is one target record statement for each record type in the target database. This structure reflects the access path approach for the following reason: each target record type is considered to represent a certain quantity of information which may be obtained from the source database; consequently, each target record description contains the specifications for the source database accessing scheme as well as the selection criteria. The second level of structure in the NRL is the target set statement (see Figure 5 for the NRL structure). Each target record statement includes one or more target set statements which identify the sets of which the target record type is a member. The third level of NRL structure is the access path statement. The access path statement specifies exactly how the source logical structure is to be traversed in order to obtain the information necessary to create an occurrence of the target record type. There may be many individual access path statements—one or more for each target set statement—since the information which contributes to the target record type may come from several different source record types.

TARGET RECORD STATEMENT
TARGET SET STATEMENT
ACCESS PATH STATEMENT
NEW TARGET ITEM STATEMENT
SOURCE RECORD STATEMENT
ITEM QUALIFICATION STATEMENT
ITEM ASSIGNMENT STATEMENT

SOURCE RECORD STATEMENT

ACCESS PATH STATEMENT

TARGET SET STATEMENT

TARGET RECORD STATEMENT


Figure 5—Structure of NRL specification

There are two types of NRL statements at the fourth level of structure: the new target item statement, and the source record statement. There may be zero or more new target item statements for each access path statement. Such a statement indicates a target item which receives a constant value each time a target record occurrence is constructed using the specified access path. Since access paths indicate structure in the source database, the new item statement is useful when information represented semantically in the source structure is converted to actual data values in the target structure.

The second NRL statement at the fourth level of structure is the source record statement. The source record statement identifies the source record(s) on the access path which will be used in the creation of a target record occurrence, to obtain item values and/or test the source data. There may be one or more source record statements for each access path statement since data may be obtained or examined from several different source record types. Furthermore, each source record type may be optionally assigned an index number. The index number is used to identify uniquely a source record within an access path which “cycles” or “loops” back on itself such that the same source record type appears more than once. Finally, there are two statement types at the fifth and final level of NRL structure; they are called the item qualification statement, and the item assignment statement. Item qualification statements are used to establish the selection criteria used in testing the source data. Consequently, one may specify a constant value to be compared against a source item value which will determine whether or not a target record occurrence is to be created using the current set of source data. The item assignment statement is used actually to obtain the source item values and assign them to the proper target items.

It should be noted that the NRL does not explicitly describe the logical structure of either the source or target database. The Michigan Data Translator obtains appropriate descriptions of the source and target databases independently of the restructuring specifications. The NRL, therefore, describes only the information relevant to the actual transformation from source to target, and assumes that the logical structures of the databases have been previously defined and are available to the Restructurer.

An NRL restructuring example

The NRL example has been chosen to illustrate the capability of the language to specify restructuring capabilities which may not be readily classified in terms of hierarchical structures or elementary operations. The example involves the extraction of implicit information from a network structure and is taken from Figure 4. This example is reproduced in Figure 6 for convenience. The NRL required to accomplish the necessary transformation is documented in Figure 7. The form of the NRL has been simplified slightly for clarity. The block structure of the language may be easily observed. Because there are three target record
The NRL description for the second record type, GRANDFATHERS, begins with statement #8. As before, there is only one target set statement (#9) because the record type is a member of only one set (GRANDSET). Statements #10, #11, and #12 specify the source access path to be used to obtain the desired information. This access strategy may be summarized as follows. The LINK record type in the source models the relation between parents and children. Given a particular PERSONS record occurrence, one may access all of his/her parents by using the set types PARENTS and CHILDREN in conjunction with the LINK record type. One similarly obtains all grandparents of a person by accessing all parents of parents. The access path statement on lines #10, #11, and #12 describes the essence of this strategy: use set PEOPLE to the PERSONS record type, then set PARENTS to the LINK record type, then set CHILDREN back to the PERSONS record type, then set PARENTS back to the LINK record type, and finally, set CHILDREN back to the PERSONS record type. Notice that the source record type PERSONS is used three times in the access path specification: once as a person, once as a parent, and once as a grandparent. Consequently, the source record statement (#13) must indicate which source group PERSONS is to be used. Since the PERSONS record which represents grandparents is the fifth record on the access path, INDEX=5 makes the necessary distinction. Statement #14 is an item qualification statement which is used to select only male grandparents to yield grandfathers, the desired target record information. Statement #15 assigns the value in the source item type “name” to the target item type “name.”

It should be clear that the target record type PARENTS
is created in a manner similar to that of GRANDFA­
HERS, except that the accessing strategy is simpler and no selection criterion is required.

OPERATIONAL SOFTWARE

There are basically two approaches to implementing the modules of a data translator: generative and interpretive. In the generative approach each module generates an object program which, when executed, performs the desired function of the data translator. The advantage of this approach is that efficient machine code may be generated for each application or function. This efficiency may be irrelevant, however, since transformations are rarely executed more than once. The disadvantage of this approach is that two phases (code generation and module execution) are required to arrive at the final result.

The interpretive approach consists of a table-driven pro­
gram that is applicable to all potential transformations. The disadvantage of this approach is the possible inefficiency of a general purpose interpreter. However, the program is easier to understand and debug.

Restructurer design decisions

Overall, the basic design objective of the Michigan Data Translator is to provide an operational software system for demonstrating, testing, and validating research results on generalized data translators. The primary design goal for the Restructurer module is the incorporation of general network restructuring capabilities. Due to the prototype nature of the task, and to reduce the implementation time and enhance the experimentation and validation effort, the simplest and most straightforward algorithm was selected. Consequently, little emphasis was placed on execution efficiency. Efficiency development was left to future versions of the Michigan Data Translator.

The decision to use an internal DBMS as an implementa­
tion tool was made early in the design process. It reflects the need for an environment in which system tables are shared by translator modules, and change often in size and complexity. In addition, since the Michigan Data Translator was developed in a research environment, the table designs themselves were subject to change. Therefore, some degree of centralization, integrity control, and data independence for system tables was clearly necessary. The translator uses ADBMS, a DBTG-type DBMS developed at the University of Michigan by the ISDOS Project. All databases, except the source and target databases, are ADBMS databases. They are manipulated by ADBMS “verbs” which take the form of subroutine calls.

Another design decision to provide generality and to ease the complexity of the Restructurer’s implementation re­
sulted in the development and use of an internal data form. As summarized earlier, the Reader module in the Michigan Data Translator accesses the source database and converts it into the internal format called the source Re­

structurer Internal Form—source RIF. In addition to main­
taining the logical structure of the source database, the RIF database has additional system access sets to every record type which facilitates fast access by the Restructurer.

Restructurer algorithm

The function of the Restructurer algorithm is to provide all the restructuring capabilities discussed in the first part of this paper. Since the Reader and Writer modules isolate the Restructurer from the source and target databases, the Restructurer need be tailored only to handle RIF databases. The Restructurer, then, is essentially on ADBMS-specific data translator and is directed by the contents of the NRL tables. The basic cycle of the Restructurer is divided into five phases as indicated in Figure 8.

During the first phase, Target Control, the Restructurer determines the next step in its traversal of the target logical structure. That is, a target set/record pair is selected for construction, as is a current access path to direct the construction process. The paths that the Restructurer takes in traversing the target database are called “construction paths.” They are defined implicitly by the TARGET SET and TARGET RECORD statements supplied by the user in his NRL specification.

During the Source Accessing phase, the Restructurer
determines the next step in its traversal of the source RIF, as indicated by the current access path. In essence, an access path is a continuous list of source relations which indicates the location in the source of information required to create a target instance.

The purpose of Source Accessing is to exhaustively search for every potential target record occurrence that could be created from the records on the source access path. When a new record occurrence is found, the Restructurer enters the Qualification phase. If none is found, that is, when the access path has been exhaustively examined, the Restructurer returns to the Target Control phase.

It is during the Qualification phase that the Restructurer implements the user's "selection criteria." The source record occurrences on the current source access path are retrieved and all specific items to be qualified are tested against the values specified in the NRL. If all of the occurrences pass, the Restructurer enters the Construction phase. Otherwise, the Restructurer returns to the Source Accessing phase to find another access path.

During the Construction phase, source record occurrences along the current access path are retrieved. Data items are extracted, conversions are performed on them as necessary, and they are stored into a new target record occurrence.

The Linking phase is the most complex phase in the Restructurer cycle. Basically, its job is to connect the newly created target record occurrence on all appropriate target sets. This is no simple task because:

1. at any given moment, all potential parent record occurrences may not exist.
2. a record occurrence may be incomplete because it may obtain partial data from each of several access paths.

During Linking, the Restructurer relies on user-specified primary key items for determining whether instances are unique. Each time a target record occurrence is created, the Restructurer must check all the records in the record type and compare keys with the newly created record. As may be obvious, this is not efficient, but is necessary given this particular implementation. Upon completion of this phase, control is returned to the Target Control phase.

Restructurer implementation

The Restructurer was written primarily in ANSI FORTRAN to simplify coding and maximize program portability. In addition, some routines were coded in assembler language to perform specialized functions not easily accomplished in FORTRAN.

Approximately 5000 lines of FORTRAN code were required to implement the prototype Restructurer. It ran in 60K words on a Honeywell H6000, and required approximately 10,000 CPU seconds to construct a target RIF database containing 2500 record occurrences, an average of 4 seconds/record. Unfortunately, the Restructurer's execution time was found to be proportional to the square of the number of target record occurrences, and an average of 1500 seconds/record was projected for a target database of 150,000 record occurrences.

SUMMARY, PROBLEMS, AND CONCLUSIONS

Two approaches to the construction of restructuring languages and software for network databases have been presented. The elementary operations approach, although effective with hierarchical databases, does not extend well to network databases. The access path approach, in which all source structures are treated in the same way, is more naturally suited to complex network transformations, and powerful restructuring capabilities are obtained.

A Network Restructuring Language (NRL) based on the high-level specification of access paths was developed. It is essentially non-procedural, and its basic constructs—access paths and selection criteria—are familiar to users of network databases. These factors help to make it generally user-friendly.

The architecture of an operational NRL-driven Restructurer was discussed. The internal DBMS used as an implementation tool proved valuable in the construction of the Restructurer. It allows last-minute design changes to be incorporated easily and provides a facility for tuning internal data management.

Experience with the Restructurer has revealed two major problems. First, as noted above, execution efficiency was not a primary concern in its design. It was expected that the prototype Restructurer would be slow but practical to use, and that efficiency enhancements would be built into subsequent versions. Unfortunately, execution time was proportional to the square of the number of the target record instances. The core of the problem was the algorithm chosen for the Linking phase of restructuring, which led to a very large (approximately 3000 FORTRAN statements), very slow Linker module.

Secondly, although the construction of target record instances was easily specified in the NRL, the means by which target sets were established was quite opaque. In fact, it required a user writing NRL descriptions to understand certain features of the Restructurer algorithm. This difficulty compromised the NRL’s claim to non-procedurality and user-friendliness.

Both of these problems have been corrected in a second version of the NRL and the Restructurer. The basic NRL structure remains unchanged although access paths are no longer required to be strictly linear—they may be tree-structured—and sets are explicitly established in a way that completely hides the restructuring algorithm from the user. Changes in the Restructurer’s algorithm and enhancements to ADBMS, which maintains the source and target databases in their internal forms, have lowered execution time to a linear function of the size of the target database (with what appears to be an acceptable slope). The reader is referred to Reference 13 for theoretical details of the
We feel justified in concluding that, using the access path approach, it is possible to build practical, general-purpose restructuring specification languages and restructuring software for network databases. However, since restructuring involves the exhaustive traversal of the source database—with some or all of the source data accessed many times—as well as the complete construction of a target database, it is an inherently slow process, and serious attention must be paid to a restructuring system’s execution efficiency.

REFERENCES


