A Generalized Data Distribution Model for Distributed Relational Databases

Patrick D. Yurk
Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses
Part of the Computer Sciences Commons

Recommended Citation
http://scholarworks.wmich.edu/masters_theses/1160
A GENERALIZED DATA DISTRIBUTION MODEL FOR DISTRIBUTED RELATIONAL DATABASES

by

Patrick D. Yurk

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
August, 1989
A GENERALIZED DATA DISTRIBUTION MODEL FOR DISTRIBUTED RELATIONAL DATABASES

Patrick D. Yurk, M.S.
Western Michigan University, 1989

This thesis presents a generalized model for the allocation of data in a distributed relational database. The model is applicable to a network of computers where there are different retrieval and update costs at the various sites, and different communication costs between sites.

The method used in this model is based upon clustering of similar computers within the network. Clusters are derived from grouping together sites with similar update, retrieval and communication costs. Data is then allocated to a cluster based on actual communication and processing costs.

The data distribution method between clusters uses a technique where the benefit of allocating data to a cluster is computed as the difference between the cost of not allocating the data to the cluster and the cost of allocating the data to the cluster. A heuristic approach is used to determine the cost of not allocating the data to the cluster. The algorithm will then attempt to maximize the total benefits for the clustered network.

The distribution model will determine the data allocation in polynomial time. The model achieves a near optimal allocation of data to a cluster through maximizing the benefits for the clusters in the network. An optimal assignment is achieved within each cluster.
ACKNOWLEDGEMENTS

I would like to dedicate this thesis to my wife, Debbie, whose patience and support made this endeavor possible.

I would also like to thank Dr. Dalia Motzkin for her insight and guidance throughout the course of my studies.

I would also like to recognize Dr. F. Boals and Dr. K. Williams and thank them for the valuable comments they provided.

Patrick D. Yurk
INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
A generalized data distribution model for distributed relational databases

Yurk, Patrick D., M.S.
Western Michigan University, 1989
TABLE OF CONTENTS

ACKNOWLEDGEMENTS .......................................................... ii
LIST OF FIGURES ................................................................. v
INTRODUCTION ..................................................................... 1
LITERATURE REVIEW ........................................................... 3
THE DATA DISTRIBUTION MODEL ......................................... 10
    The Method ................................................................. 10
    Assumptions .............................................................. 12
    Definitions and Parameters .......................................... 13
    Formulation ............................................................... 14
ALGORITHM DESCRIPTION .................................................. 16
    Phase I - Initial Fragment Allocation .............................. 16
    Phase II - Final Fragment Allocation ............................ 17
    Internal Cluster Allocation ......................................... 18
PERFORMANCE EVALUATION .............................................. 20
EVALUATION OF THE RESULTING SOLUTION ......................... 21
FUTURE EXTENSIONS ......................................................... 22
EXAMPLE ............................................................................. 23
    Description ............................................................. 23
    Site Descriptions ..................................................... 23
    Global Relations ...................................................... 27
    Site Applications ..................................................... 27
    Sample Data .......................................................... 29
    Site Definitions ...................................................... 30
    Transactions and Data Requirements ........................... 30
    Fragments Produced ................................................ 34

iii

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LIST OF FIGURES

1. Algorithm Flow Diagram .......................................................... 19
2. Physical Site Architecture ......................................................... 25
3. Logical Site Architecture .......................................................... 26
4. Clustered Architecture ............................................................. 39
INTRODUCTION

There has been a concentration of effort and research around the hardware configuration, architecture and control protocols for distributed database systems. There have also been considerable numbers of papers discussing the allocation of applications and files in a distributed database system. This thesis focuses attention in a different direction, that of the allocation of data in a distributed relational database system.

An algorithm is proposed that determines the allocation of horizontal fragments of a distributed relational database. This algorithm achieves a near optimal allocation of fragments to sites in the distributed system. The allocation of data fragments is determined based upon calculating the costs for updating the data, retrieving the data, communicating between sites and storing the data at the site.

The algorithm allows for different processing costs at the various sites and different costs for communication between the various sites in the distributed network of computers. This makes the algorithm especially applicable to a network of dissimilar computers across a widely varied geographic area.

A heuristic approach is utilized in order to achieve the efficiency of being able to complete the determination of a near optimal solution in polynomial time. An example is described to clarify the allocation model and show its applicability to a real life situation. The model has also been implemented in the form of a PASCAL program (see Appendix A) executing on a personal computer.

This model is an extension to the general case of earlier work by Motzkin (1987a, 1988) and Motzkin and Ivey (1987b, 1987c). Concepts...
for the model and the algorithm, as suggested by Motzkin, have been incorporated in this thesis. Formerly, a homogeneous distributed system configuration was assumed where the costs for processing and communication were nearly the same across all sites of the network. This generalized algorithm is a major extension that allows application to a broad spectrum of real life distributed database situations.

The input requirements are manageable, yet provide accurate cost estimation. The model allocates fragments of files rather than complete files. Thus, each site is assigned only the data fragments that are used by the applications executing at the site. Replication of the data is allowed, thus a more efficient utilization of the data is achieved.
LITERATURE REVIEW

In this section a review is made of several recent papers in the area of distributed relational database systems. A brief overview is given of each paper and the relevance of each paper to this thesis is described.

A paper by Baba, You and Hevner (1987) develops a top-down design methodology for a multi-processor database machine using data flow analysis. The model finds the best way to distribute the database function on the processors in order to reduce CPU bottlenecks and improve response time. The methodology combines the data flow model with various cost computations. An interesting point in the cost computation is the assumption that the cost of data transfer is proportional only to the amount of data to be transmitted. The focus of the paper is on a methodology to determine an optimum distribution of the database functions across the various processors of a distributed database machine. The paper does not address the allocation of data fragments to sites in the network.

A paper by Borzemski and Kasprzak (1987) demonstrates a branch and bound algorithm to obtain an optimal network communication design for a distributed database system. The goal is to achieve the requirement of supporting communicating between all pairs of sites in the DDBMS. This is done using a connected and undirected graph with capacity and cost associated with each link of the graph. All alternative links between all pairs of sites are evaluated and the optimal link between each pair of sites is selected. The paper demonstrates the connectivity between sites of a DDBMS network but does not address the distribution and allocation of data between sites.

A paper by Ceri and Pelagatti (1982) examines the advantages of making
all applications in a distributed computer system local applications. This is done by locally allocating all the resources needed to support the applications at each site. Cost summation formulas are developed to compare the cost of allocating the resources locally as compared to allocating the resources on a distributed basis. The model uses a complicated maximum flow-minimum cut graph theory based method of determining optimal allocation. The model uses similar cost summation formulas to those used in this thesis, however a very different solution method is applied. Also, the suggestion is made to allocate all resources so that all applications become local applications. This is only possible in terms of data retrievals, updating of data still remains as a global application.

The paper by Ceri, Pernici and Wiedrhold (1987) is a good survey of existing design methodologies. The paper provides a summary of vertical fragmentation of a distributed relational database. It also outlines both the top-down and bottom-up approaches to distributed design. A good example of an airline reservation system for applications, requirements, sites and fragment allocation is described. Suggested are several other references for additional information. The paper does not address the optimal allocation of fragments based on costs.

A paper by Irani and Khabbaz (1982) describes a very similar model to that in the paper by Borzemski and Kasprzak. A graph theory based algorithm with a Greedy heuristic is employed to determine the best design for a distributed network with communication cost and reliability constraints. Two points are of particular interest in this paper. First, it is mentioned that an especially serious problem is network disconnection. If some of the copies of a file are unavailable for update due to a temporarily
disconnected node, then update to that file cannot be completed. Second, it is shown that in some cases it is less costly to have more communication channels in the network when the cost of reliability and file availability are taken into account. The paper does not address the problem of determining the content of the data files.

The paper by Jain (1987) addressed the problem of selecting the class of computer, processing speed, primary and secondary storage requirements and allocation of data files at each node in a distributed system. A nonlinear goal programming model was presented and a modified pattern search heuristic was utilized. The paper does not address the allocation of tuples in a distributed relational database.

A paper by Jain and Dutta (1986) develops a multi-criteria decision making model which supports the human designer. The model allows for inclusion of multiple opposing factors that are traded off against each other to help achieve a satisfactory design of a distributed computer system. This is done by formulating the design into a vector minimization problem and then using an exchange heuristic. The model results in a solution table showing the values of such variables as cost, response time, and availability. This will show, for example, how the cost will be increased if higher availability and lower response times are desired. The paper does not address the allocation of data in the network.

The paper by Jain (1985) describes an interactive system that has been developed in FORTRAN on a UNIVAC computer. The paper gives a sample session and simple example. The objective of the interactive program is to give the designer of a distributed computer system a tool to assist in the task of choosing the best design of the system. Parameters taken

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
into consideration include availability required, response times required, classes of computers, costs of storage, costs of communication, etc. The interactive procedure will determine a solution as to what classes of computers, amount of storage, and what files should be selected for each site in the network. The procedure uses a two step process. First an initial satisfactory solution is determined. Then, an optimizing routine attempts to improve the solution. The paper does not address the allocation of tuples in the network.

The paper by Motzkin and Ivey (1987b) describes a process to segment and fragment the global relations of a distributed relational database. The fragments are then allocated to sites such that optimization of the local and global costs is achieved. Algorithms for segmenting, fragmenting and allocating the fragments of the global relations are described. The allocation of fragments is based on maximizing the benefit of allocating the fragment to specific sites. The benefit is computed as the difference in the cost of not allocating the fragment and the cost of allocating the segment to a given site. The cost of allocating a fragment is the sum of the costs for space, local retrievals, local updates and updates sent from other sites. The cost of not allocating the fragment is the sum of local retrievals from remote sites. A fundamental assumption in the allocation model is that the owning site pays for all local applications retrievals and for all updates issued by remote sites. A average costs of communication, retrieval and update are also assumed. The initial allocation is adjusted to take into account space constraints at each site. When reallocation is required, the sites with the largest benefits for allocation are selected.
Motzkin and Ivey (1987c) describes an automated tool for the design of distributed relational databases. It uses previous work by Motzkin and Ivey as the foundation for the fragment allocation algorithms and extends that work to incorporate user's reliability requirements. This is accomplished through a two step process of first fragmenting and allocating the global relations and secondly revising the allocation to achieve the required reliability.

A paper by Motzkin (1987a) integrates the work done in previous papers by Motzkin and Ivey into an allocation model that includes both reliability and space constraints.

Another paper by Motzkin (1988) further extends the work done previously by Motzkin and Ivey by revising the initial data allocation based on statistics of transaction workload and network throughput that is accumulated over time by empirical means. The reallocation of data is only done if it is determined that the expected cost savings due to the revision is greater than the cost of the reallocation process itself.

The paper by Mukkamala, Bruell and Shultz (1988) describes three parts of the design of distributed database systems. This includes the allocation of data in the system, the load-balancing of the system and the design evaluation of the system. An iterative heuristic approach is employed to achieve a near-optimal design. The data allocation scheme used is a tree-structured transaction model that takes into account the CPU and I/O times, and the amount of information flow between nodes in the tree. The first step in the allocation process is to partition the data items into groups via a non-linear zero-one integer programming technique with minimization of the sum of group interactions. The benefit of grouping
pairs of data items is determined as the reduction in communication costs as a result of grouping the data items together at a site. The second step in the allocation process is to allocate the groups that were formed in the first step. The allocation of groups is also represented as a non-linear zero-one integer programming problem with the objective of minimizing the total communication cost. The next two parts of the design, the load-balancing and design evaluation are also described. One point made in this paper is that allocation of data items is the primary factor in achieving performance of the database.

In a paper by Raghuram, Morgan, Bosworth, Stiles and Wheeler (1987) a model is developed for the data distribution and network topology of a distributed database system. The system is formulated into a multi-criteria non-linear programming problem. The problem is solved using a heuristic branch and bound algorithm. The model is intended to determine the system with the best combination of cost and response time. Cost components considered include communication line costs, computer costs, storage and database maintenance costs, costs of queries and cost of updates. An interesting point made in the paper is that "if there is only one copy [of the database] then query traffic on the network is maximum and update synchronization traffic is minimum. If it is replicated at every node the query traffic is minimum and the update synchronization traffic is maximum." The paper also categorizes transactions into two types, queries and updates. Another point made is that the channel delay is considerable when compared to the other delays so that all but the channel delay can be neglected and also that the delay is dependent on the particular route. The heuristic function used is that of an iterative present best solution algorithm that
starts with a given initial solution. The paper makes several relevant points but does not address the allocation of fragments to sites.

In a paper by Su (1983) described is a network of microcomputers referred to as MICRONET. This is a very specialized network consisting of identical PDP11/03 microcomputers connected through a common network BUS. This restricts the network to a very small geographic area (within the same facility). The paper describes the inner workings of the various RDBMS functions as implemented in the MICRONET system. The paper does not address data allocation nor does it address a general purpose network that spans a large (perhaps global) geographic area consisting of a variety of different types and capacity computers.
THE DATA DISTRIBUTION MODEL

The Method

This thesis develops a method for allocation of the data in a distributed relational database system for a network of dissimilar computers. A new data distribution model has been developed and the method used is described as follows.

Individual processing sites are grouped into clusters. Sites with similar processing and communications costs are assigned to the same cluster.

Fragments of the global data relations are defined by relational predicates which identify the tuples of the global relations that are included in the respective fragments. This fragmentation is determined based upon which applications need to use the particular data tuples (Ceri & Pelagatti, 1984; Motzkin 1987a, 1988; Motzkin and Ivey 1987b, 1987c). A fragment may be replicated at more than one cluster as well as at one or more sites within a cluster. The algorithm determines which clusters are to be allocated the fragment and which sites within a cluster are to be assigned a copy of the fragment.

It is assumed that the site which has been allocated a copy of a fragment pays for the costs associated with having its own copy of the fragment. This includes the costs for all local updates (updates done by applications executing at the site), costs for all remote updates (updates done by applications executing at other sites), the cost of local retrievals and the cost of communication for update or retrieval by remote sites. The owning site also pays for the cost of storage space for the fragment.
Costs of communications between clusters are computed as if clusters were single nodes, i.e. as if communication with the cluster from outside is handled by one site within the cluster.

Introduced is the term potential benefit (PB). The potential benefit of a fragment to a site is the difference between the frequency of local retrievals and the frequency of remote updates. Local retrievals will generally cost a site less than remote retrievals while updates are a maintenance "penalty" that a site pays when it owns a fragment. It is likely that the actual benefit of assigning a fragment (F) to a site (S) is positive when PB(F,S) is high.

The cost of allocating (CA) a fragment to a cluster is calculated as the cost of allocating the fragment to the site with the highest potential benefit value of all sites in the cluster. This also becomes the potential benefit value of the cluster.

In order to compute the cost of not allocating (CN) a fragment, it is necessary to determine where the fragment will be allocated. If a fragment is not allocated to the cluster then it is determined at which other cluster the data has been allocated. The heuristic used here assumes that the fragment will always be allocated to the cluster that has the highest potential benefit value of all clusters. The model then determines additional clusters to which each fragment is assigned.

The model has two phases. During the first phase the fragments are derived and initially allocated to each site that has applications that use them. It is during the first phase that the clusters are determined. A pair of sites belong to the same cluster if and only if they have similar processing and communications costs. During the first phase the fragments
are marked as being permanently allocated to the clusters that have the highest potential benefit value for the fragment.

During the second phase an approximate cost of not allocating the fragments to the clusters is computed and the benefit of allocating the fragments to the clusters is computed as the difference between the approximate cost of not allocating and the cost of allocating the fragment. The algorithm cancels assignments of fragments at all clusters that do not have a positive benefit value. Since the benefit value is an approximate value, near optimal assignment of fragments to clusters is achieved. After a fragment is assigned to a cluster, then the fragment will be assigned to sites within the cluster in the same manner as in Motzkin (1987a, 1988) and Motzkin and Ivey (1987b, 1987c).

Assumptions

During the course of the algorithm, the following assumptions are made.

Transactions that update more than one fragment can be broken into composite transactions that update only one fragment.

The various costs (cost of space, cost of unit retrieval, cost of unit update, cost of unit communication) have been converted to a common unit (bytes, blocks, etc.) either manually or by an automated front end conversion process prior to being used in this model. This simplifies the calculations while it does not detract from the generalization of the model.

The physical allocation of tuples is done by the RDBMS based upon the fragment assignment.
Definitions and Parameters

During the course of the model the following definitions and parameters are used.

\[ \text{FREQRS}(T_j, F, S_i) = \text{frequency of retrieval issued by transaction } T_j \text{ to fragment } F \text{ at site } S_i \text{ (0 when } T_j \text{ is not using anything from } F) \]

\[ \text{FREQRC}(T_j, F, C_i) = \text{frequency of retrieval issued by transaction } T_j \text{ to fragment } F \text{ at cluster } C_i \text{ (0 when } T_j \text{ is not using anything from } F) \]

\[ \text{FREQUS}(T_j, F, S_i) = \text{frequency of updates issued by transaction } T_j \text{ to fragment } F \text{ at site } S_i \text{ (0 when } T_j \text{ is not using anything from } F) \]

\[ \text{FREQUC}(T_j, F, C_i) = \text{frequency of updates issued by transaction } T_j \text{ to fragment } F \text{ at cluster } C_i \text{ (0 when } T_j \text{ is not using anything from } F) \]

\[ \text{PBI}(F, C_i) = \text{Potential benefit indicator of fragment } F \text{ at cluster } C_i \]

\[ \text{REL}(F) = \text{the relation that defines fragment } F \]

\[ \text{FREQR}(T_j) = \text{the number of unit retrievals required by transaction } T_j \text{ during a unit of time} \]

\[ \text{FREQU}(T_j) = \text{the number of unit updates required by transaction } T_j \text{ during a unit of time} \]

\[ \text{CSPS}(F, S_i) = \text{cost of space of fragment } F \text{ at site } S_i \]

\[ \text{CSPC}(F, C_i) = \text{cost of space of fragment } F \text{ at cluster } C_i \]

\[ \text{CRS}(S_i) = \text{cost of unit retrieval at site } S_i \]

\[ \text{CRC}(C_i) = \text{cost of unit retrieval at cluster } C_i \]

\[ \text{CUS}(S_i) = \text{cost of unit update at site } S_i \]

\[ \text{CUC}(C_i) = \text{cost of unit update at cluster } C_i \]

\[ \text{CCS}(S_i, S_k) = \text{cost of unit communication between site } S_i \text{ and site } S_k \]

\[ \text{CCC}(C_i, C_k) = \text{cost of unit communication between cluster } C_i \text{ and cluster } C_k \]

\[ \text{CA}(F, C_i) = \text{cost of allocating fragment } F \text{ to cluster } C_i \]
RCN(F,Cj,Ct) = relative cost of not allocating fragment F to cluster Cj with respect to cluster Ct

RB(F,Cj,Ct) = relative benefit of allocating fragment F to cluster Cj with respect to Ct

Formulation

Also during the course of the algorithm, the following list of formulas are used.

\[ \text{FREQRC(Tj,F,Cj)} = \sum_{Sk \text{ in Cj}} \text{FREQRS(Tj,F,Sk)} \]

\[ \text{FREQUC(Tj,F,Cj)} = \sum_{Sk \text{ in Cj}} \text{FREQUS(Tj,F,Sk)} \]

\[ \text{CSPC(F,Cj)} = \frac{\text{AVG}(\text{CSPS(F,Sk)})}{\text{Sk in Cj}} \]

\[ \text{CRC(Cj)} = \frac{\text{AVG}(\text{CRS(Sk)})}{\text{Sk in Cj}} \]

\[ \text{CUC(Cj)} = \frac{\text{AVG}(\text{CUS(Sk)})}{\text{Sk in Cj}} \]

Communications site (Sm) for the cluster is selected such that the following is satisfied:

\[ \text{FREQRS(Tp,F,Sm)} = \max_{p} \sum_{Sk \text{ in Cj}} \text{FREQRS(Tp,F,Sk)} \]

\[ \text{PBI(F,Cj)} = \sum_{Tj at Cj} \text{FREQRC(Tj,F,Cj)} - \sum_{Tj not at Cj} \text{FREQUC(Tj,F,Cj)} \]

\[ \text{CA(F,Cj)} = \sum_{j,Tj at Cj} \text{CRC(Cj)} \times \text{FREQRC(Tj,F,Cj)} + \sum_{j,Tj at Cj} \text{CUC(Cj)} \times \text{FREQUC(Tj,F,Cj)} + \text{CSPC(F,Cj)} + \sum_{j,Tj at Ck \ w i t h \ k \neq i} \text{CUC(Cj)} \times \text{FREQUC(Tj,F,Ck)} + \sum_{j,Tj at Ck \ w i t h \ k \neq i} \text{SUM}(\text{FREQUC(Tj,F,Ck)} \times \text{CCC(F,Cp,Ck)}) \]
\[ RCN(F,C_i,C_m) = \sum_{j,T_j \text{ at } C_l} CRC(C_m) \times FREQRC(T_j,F,C_i) \]

\[ + \sum_{j,T_j \text{ at } C_l} FREQRC(T_j,F,C_i) \times CCC(F,C_i,C_m) \]

\[ RB(F,C_i,C_m) = RCN(F,C_i,C_m) - CA(F,C_i) \]
ALGORITHM DESCRIPTION

Phase I - Initial Fragment Allocation

Accept the input from the user defining the global relations and tuples contained in the global database relations. Also accept input as to the applications that will execute at each of the various sites in the network and the processing and communication costs at and between those sites. This is done through a user friendly interface as described in previous works (Motzkin & Ivey, 1987b).

Determine the fragments using the method described in previous works (Ceri & Pelagatti, 1984; Motzkin et al., 1987a 1987b, 1987c, 1988). Fragments must be pairwise disjoint and satisfy the condition that each fragment is completely required by the transaction using the fragment or not required at all. The principle here being that there is no unnecessary data to support the transactions at a given site.

Determine the clusters and which sites belong to each cluster. This is done by looking at each of the defined sites in turn and building groups (clusters) of sites that have similar costs for processing and communication. Sites are considered to be similar if they have processing costs that are within a given threshold of one another. The threshold value is arbitrary and can be supplied as an input parameter to the algorithm. Likewise, a threshold value is supplied for communications costs. All pairs of sites contained in the cluster must have communication costs that do not exceed the threshold.

Initially allocate fragments to all clusters that have applications which use the fragment. This is to be considered a temporary assignment that
may be canceled later in the algorithm.

Compute the potential benefit (PB) of allocating a fragment to a cluster as described above. Permanently allocate each fragment to the cluster with the highest potential benefit for that fragment.

**Phase II - Final Fragment Allocation**

For each fragment, process all clusters in the order of least communication cost relative to the cluster indicated as permanent allocation of the fragment in Phase I.

For each cluster that is not indicated for permanent allocation of the fragment, compute the relative cost of not allocating the fragment to the cluster using the costs of communication and cost of retrieval relative to each of the other clusters to which the fragment has been permanently allocated.

For each of the clusters, compute the relative benefit of allocating the fragment to the cluster as the difference between the relative cost of not allocating the fragment to the cluster and the cost of allocating the fragment to the cluster.

If the relative benefit of allocating the fragment to the cluster is less than zero, then cancel the allocation of the fragment to the cluster.

If all of the benefits of allocating the fragment to the cluster are greater than or equal to zero, then set the indicator to mark the fragment as being permanently allocated to the cluster.

Perform the steps above until all fragments and clusters have been processed.
Internal Cluster Allocation

For each of the clusters determine the internal allocation of fragments to sites within the cluster using the allocation model described in previous works (Motzkin 1987a, 1988; Motzkin and Ivey 1987b, 1987c).

Lastly, output the allocation solution and terminate the algorithm. See Figure 1 for a flow diagram showing the above algorithm.
Phase I

Determine pairwise disjoint fragments of relations such that each fragment is completely required by the transaction or not at all.

Define clusters of sites satisfying the following: Each site belongs to exactly one cluster. A pair of sites belong to the same cluster if and only if they have similar processing costs. A group of sites is in the same cluster if and only if there is a similar communication cost between every pair of sites in the group.

Initially allocate each fragment to all sites where it is used. Define these allocations as temporary allocations.

For each fragment compute the potential benefit indicator (PBI) for each cluster that contains sites that have been allocated the fragment.

For each fragment permanently allocate the fragment to the cluster with the highest potential benefit indication.

Phase II

Denote all fragments as unprocessed. Let $M$ be the index of the first unprocessed fragment.

Sort temporary clusters of $F_M$ in ascending cost of communication order relative to the permanent site.

Denote $RCH(F, C_1, C_2) = \text{relative cost of not allocating } F \text{ to } C_1 \text{ with respect to } C_2$

$RB(F, C_1, C_2) = \text{relative benefit of allocating } F \text{ to } C_1 \text{ with respect to } C_2$

$CA(F, C_1) = \text{cost of allocating } F \text{ to } C_1$

Set $T$ = the index of the first temporary cluster of $F_M$

Set $P = \text{index of first permanent site of } F_M$

Compute $RB(F, C_1, C_2) = RCH(F, C_1, C_2) - CA(F, C_1)$

Set $P = \text{next permanent cluster of } F_M$

$RB(F, C_1, C_2) < 0$

Cancel allocation of $F_M$ to $C_1$

Set $T = \text{next temporary cluster of } F_M$

$T$ becomes a permanent cluster of $F_M$

Set $T = \text{next temporary cluster of } F_M$

Determine internal cluster fragment allocation.

Output final data distribution.

Figure 1. Algorithm Flow Diagram

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
PERFORMANCE EVALUATION

This model differs from previous models in the computation of relative benefit. The additional part of this algorithm has a computing time of $O(F \cdot S^2)$ where: $F =$ number of fragments, $S =$ number of sites.

Since the fragments are disjoint, $F$ is bounded by the number of records, $N$, in the database. Let $M = \text{MAX}(N,S)$ then the added part of the model has a computing time of $O(M^3)$.

Since the previous model has a computing time of $O(M^2 \log(M))$, this entire model has a computing time of $O(M^3)$. 

20
EVALUATION OF THE RESULTING SOLUTION

The frequency of retrieval is the primary parameter in the model. In most real life applications the frequency of update will be low by comparison to the frequency of retrieval. In the normal case where the frequency of update is low, allocation is primarily dependent on the frequency of retrieval. This would result in the model allocating the data fragments to most of the clusters that have applications that use the fragments.

In cases where the frequency of update is high, the model will likely allocate the fragments to only a few clusters. In this case, the fragments would likely be allocated to clusters that have the highest frequency of updates.
FUTURE EXTENSIONS

One future extension might be to make use a different two phased approach. In the first phase, the cost of not allocating is computed based on average communication and local retrieval costs. In the second phase, re-compute the cost of not allocating based upon knowing to which clusters the fragment was allocated in the first phase. Then reallocate to reflect the second computations of the cost of not allocating.

Another extension of the model would be to add an adaptive algorithm which uses statistics on workload and throughput collected by empirical means to potentially reallocate the data fragments.

Also, the model could be extended to incorporate space and reliability constraints when determining the allocation of fragments in the network.
EXAMPLE

Description

In order to illustrate the application of the allocation model to a real life situation the following example is given. This example consists of a distributed relational database management system for the information needs of the sales staff of an international company.

Site Descriptions

Three of the field sales offices (S1, S2, S3) use an IBM PS2 microcomputer for execution of local field office applications. One field office (S4) uses a DEC VAX for execution of local field office applications. These field offices are located at various geographic locations both domestically and internationally.

There are also regional sales offices (S5, S6) that centralize administrative functions for a larger geographic area. Each of the regional offices uses a DEC VAX computer for execution of regional administrative applications. The regional centers also serve as a major node for communication with the field offices in the region.

The international sales organization (S7) has a data center with a DEC VAX computer. The domestic sales organization (S8) is the main data center and uses an IBM 3090 computer servicing the needs of the domestic sales organization and the company as a whole. The domestic and international computer centers also serve as the major node for communication with their respective regional computer centers. In addition, the respective divisional sites are the manufacturing facilities which produce
certain products for sale worldwide. See Figure 2 for a graphical representation of the physical network.

The model when applied to this example views the network as fully connected with each site being able to communicate with every other site in the system. In practice this may be accomplished by passing communication through more than one node in the network in order to communicate with any given pair of sites. This is reflected in the communication costs that are provided between each pair of sites in the system. See figure 3 for a graphical representation of the logical network architecture.
Figure 2. Physical Site Architecture
Figure 3. Logical Site Architecture
Global Relations

The following are the global relations in third normal form that are used by this example distributed relational database system.

SALES - sales figures.

\[
\text{CUST\#, PROD\#, REP\#, DATE, QTY, UNIT\_PRICE}
\]

PRODUCTS - manufacturing cost information.

\[
\text{PROD\#, DIV\#, COST}
\]

CUSTOMER - customer information.

\[
\text{CUST\#, NAME, ADDRESS}
\]

ORG - organizational staff assignment information.

\[
\text{DIV\#, REG\#, AREA\#, REP\#}
\]

Where: CUST\# is the customer identification number.

\[
\text{PROD\# is the product identification number.}
\]

\[
\text{REP\# is the sales representative identification number.}
\]

\[
\text{DIV\# is the division identification number.}
\]

\[
\text{REG\# is the region identification number.}
\]

\[
\text{AREA\# is the field office identification number.}
\]

Site Applications

The field offices perform the following list of applications:

1. Sales orders for each sales representative (application A1 at site S1, application A4 at site S2, application A7 at site S3, application A10 at site S4).

2. Establishing prices for specific customers and products (application A2 at site S1, application A5 at site S2, application A8 at site S3, application A11 at site S4).
3. Maintain lists of customer names and addresses for customers to which the representatives in the field sales area sell products. (application A3 at site S1, application A6 at site S2, application A9 at site S3, application A12 at site S4).

The regional offices perform the following list of applications:

1. Consolidated sales orders for the field offices in the region (application A13 at site S5, application A15 at site S6).

2. Monthly sales summaries for the region (application A14 at site S5, application A16 at site S6).

The domestic and international data centers perform the following list of applications:

1. Manufacturing and shipping of sales orders for products produced by the division (application A17 at site S7, application A20 at site S8).

2. Establishing product costs for products produced by the division (application A18 at site S7, application A21 at site S8).

3. Monthly sales summaries for the division (application A19 at site S7, application A22 at site S8).

One additional application at the domestic divisional data center:

Organization assignment (application A23 at site S8).
Sample Data

The following is used as input to the model. These are the tuples in the relations of the example.

### SALES RELATION

<table>
<thead>
<tr>
<th>CUST#</th>
<th>PROD#</th>
<th>REP#</th>
<th>DATE</th>
<th>QTY</th>
<th>UNIT_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>10</td>
<td>100</td>
<td>4/30</td>
<td>5</td>
<td>1.50</td>
</tr>
<tr>
<td>1001</td>
<td>11</td>
<td>200</td>
<td>4/30</td>
<td>100</td>
<td>2.00</td>
</tr>
<tr>
<td>1003</td>
<td>10</td>
<td>100</td>
<td>5/11</td>
<td>5</td>
<td>1.50</td>
</tr>
<tr>
<td>1004</td>
<td>10</td>
<td>100</td>
<td>5/01</td>
<td>10</td>
<td>1.25</td>
</tr>
<tr>
<td>1004</td>
<td>12</td>
<td>200</td>
<td>5/10</td>
<td>100</td>
<td>1.00</td>
</tr>
<tr>
<td>1005</td>
<td>15</td>
<td>300</td>
<td>4/30</td>
<td>5</td>
<td>1.00</td>
</tr>
<tr>
<td>1007</td>
<td>25</td>
<td>400</td>
<td>4/30</td>
<td>5</td>
<td>7.00</td>
</tr>
<tr>
<td>1012</td>
<td>15</td>
<td>800</td>
<td>5/01</td>
<td>50</td>
<td>1.25</td>
</tr>
<tr>
<td>1013</td>
<td>25</td>
<td>900</td>
<td>4/30</td>
<td>5</td>
<td>7.50</td>
</tr>
<tr>
<td>1014</td>
<td>25</td>
<td>900</td>
<td>5/10</td>
<td>5</td>
<td>7.50</td>
</tr>
<tr>
<td>1015</td>
<td>27</td>
<td>1000</td>
<td>5/01</td>
<td>15</td>
<td>1.75</td>
</tr>
<tr>
<td>1015</td>
<td>27</td>
<td>1000</td>
<td>5/10</td>
<td>7</td>
<td>1.75</td>
</tr>
<tr>
<td>1019</td>
<td>10</td>
<td>1000</td>
<td>5/01</td>
<td>10</td>
<td>1.50</td>
</tr>
<tr>
<td>1019</td>
<td>25</td>
<td>1300</td>
<td>4/30</td>
<td>2</td>
<td>7.50</td>
</tr>
</tbody>
</table>

### PRODUCTS RELATION

<table>
<thead>
<tr>
<th>PROD#</th>
<th>DIV#</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1.50</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>0.75</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>0.75</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>5.00</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
<td>1.50</td>
</tr>
</tbody>
</table>

### CUSTOMER RELATION

<table>
<thead>
<tr>
<th>CUST#</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Jones Co.</td>
<td>Kalamazoo, Michigan</td>
</tr>
<tr>
<td>1003</td>
<td>Brown Co.</td>
<td>Detroit, Michigan</td>
</tr>
<tr>
<td>1004</td>
<td>Smith Co.</td>
<td>Detroit, Michigan</td>
</tr>
<tr>
<td>1005</td>
<td>Acme Co.</td>
<td>Tampa, Florida</td>
</tr>
<tr>
<td>1007</td>
<td>Jones Co.</td>
<td>Miami, Florida</td>
</tr>
<tr>
<td>1012</td>
<td>Eastern Co.</td>
<td>London, England</td>
</tr>
<tr>
<td>1013</td>
<td>Western Co.</td>
<td>London, England</td>
</tr>
<tr>
<td>1014</td>
<td>Southern Co.</td>
<td>London, England</td>
</tr>
<tr>
<td>1015</td>
<td>Yurk Inc.</td>
<td>Toronto, Canada</td>
</tr>
<tr>
<td>1019</td>
<td>Northern Co.</td>
<td>Montreal, Canada</td>
</tr>
</tbody>
</table>
Site Definitions

The following are the key values in the organization relation that identify the sites in the network.

<table>
<thead>
<tr>
<th>SITE</th>
<th>DIV#</th>
<th>REG#</th>
<th>AREA#</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA Offices</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>S3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S4</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>REGIONAL Offices</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>S6</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DIVISIONAL Offices</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S7</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S8</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Transactions and Data Requirements

Also input into the model are the relational predicates that identify the data that are required for each of the application transactions along with the home site for the transaction.

<table>
<thead>
<tr>
<th>APPLICATION/TRANSACTION SITE</th>
<th>DATA REQUIREMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 S1</td>
<td>SELECT ORG WHERE DIV#, REG#, AREA# = 1, 1, 1 TO TEMP SEMIJOIN SALES WITH TEMP WHERE REP#=REP# TO RELA1</td>
</tr>
<tr>
<td>A2 S1</td>
<td>SELECT PRODUCTS TO RELA2</td>
</tr>
</tbody>
</table>
DATA REQUIREMENTS

A3 S1 SELECT ORG WHERE DIV#, REG#, AREA# = 1, 1, 1 TO TEMP
SEMIJOIN SALES WITH TEMP WHERE REP#=REP#
TO TEMP1
SEMIJOIN CUSTOMER WITH TEMP1 WHERE CUST#=CUST# TO RELA3

A4 S2 SELECT ORG WHERE DIV#, REG#, AREA# = 1, 1, 2 TO TEMP
SEMIJOIN SALES WITH TEMP WHERE REP#=REP#
TO RELA4

A5 S2 SELECT PRODUCTS TO RELA5

A6 S2 SELECT ORG WHERE DIV#, REG#, AREA# = 1, 1, 2 TO TEMP
SEMIJOIN SALES WITH TEMP WHERE REP#=REP#
TO TEMP1
SEMIJOIN CUSTOMER WITH TEMP1 WHERE CUST#=CUST# TO RELA6

A7 S3 SELECT ORG WHERE DIV#, REG#, AREA# = 2, 1, 1 TO TEMP
SEMIJOIN SALES WITH TEMP WHERE REP#=REP#
TO RELA7

A8 S3 SELECT PRODUCTS TO RELA8

A9 S3 SELECT ORG WHERE DIV#, REG#, AREA# = 2, 1, 1 TO TEMP
SEMIJOIN SALES WITH TEMP WHERE REP#=REP#
TO TEMP1
SEMIJOIN CUSTOMER WITH TEMP1 WHERE CUST#=CUST# TO RELA9

A10 S4 SELECT ORG WHERE DIV#, REG#, AREA# = 2, 1, 2 TO TEMP
SEMIJOIN SALES WITH TEMP WHERE REP#=REP#
TO RELA10

A11 S4 SELECT PRODUCTS TO RELA11

A12 S4 SELECT ORG WHERE DIV#, REG#, AREA# = 2, 1, 2 TO TEMP
SEMIJOIN SALES WITH TEMP WHERE REP#=REP#
TO TEMP1
SEMIJOIN CUSTOMER WITH TEMP1 WHERE CUST#=CUST# TO RELA12
APPLICATION/TRANSACTION SITE

DATA REQUIREMENTS

A13  S5  SELECT ORG WHERE DIV#, REG# = 1, 1
      TO TEMP
      SEMIJOIN SALES WITH TEMP WHERE REP#=REP#
      TO RELA13

A14  S5  SELECT ORG WHERE DIV#, REG# = 1, 1
      TO TEMP
      SEMIJOIN SALES WITH TEMP WHERE REP#=REP#
      TO RELA14

A15  S6  SELECT ORG WHERE DIV#, REG# = 2, 1
      TO TEMP
      SEMIJOIN SALES WITH TEMP WHERE REP#=REP#
      TO RELA15

A16  S6  SELECT ORG WHERE DIV#, REG# = 2, 1
      TO TEMP
      SEMIJOIN SALES WITH TEMP WHERE REP#=REP#
      TO RELA16

A17  S7  SELECT PRODUCTS WHERE DIV# = 2
      TO TEMP
      SEMIJOIN SALES WITH TEMP WHERE PROD# = PROD#
      TO RELA17

A18  S7  SELECT PRODUCTS WHERE DIV# = 2
      TO RELA18

A19  S7  SELECT PRODUCTS WHERE DIV# = 2
      TO TEMP
      SEMIJOIN SALES WITH TEMP WHERE PROD# = PROD#
      TO RELA19

A20  S8  SELECT PRODUCTS WHERE DIV# = 1
      TO TEMP
      SEMIJOIN SALES WITH TEMP WHERE PROD# = PROD#
      TO RELA20

A21  S8  SELECT PRODUCTS WHERE DIV# = 1
      TO RELA21

A22  S8  SELECT PRODUCTS WHERE DIV# = 1
      TO TEMP
      SEMIJOIN SALES WITH TEMP WHERE PROD# = PROD#
      TO RELA22

A23  S8  SELECT ORG TO RELA23
Also used as input to the model are the following example processing and communication costs along with the frequency of update and retrieval for each transaction.

<table>
<thead>
<tr>
<th>SITE SPACE</th>
<th>COST OF SPACE</th>
<th>COST OF RETRIEVAL</th>
<th>COST OF UPDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.001</td>
<td>.10</td>
<td>.20</td>
</tr>
<tr>
<td>2</td>
<td>.001</td>
<td>.10</td>
<td>.20</td>
</tr>
<tr>
<td>3</td>
<td>.001</td>
<td>.11</td>
<td>.22</td>
</tr>
<tr>
<td>4</td>
<td>.002</td>
<td>.20</td>
<td>.40</td>
</tr>
<tr>
<td>5</td>
<td>.002</td>
<td>.20</td>
<td>.42</td>
</tr>
<tr>
<td>6</td>
<td>.002</td>
<td>.22</td>
<td>.44</td>
</tr>
<tr>
<td>7</td>
<td>.003</td>
<td>.30</td>
<td>.50</td>
</tr>
<tr>
<td>8</td>
<td>.003</td>
<td>.33</td>
<td>.53</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SITE A</th>
<th>SITE B</th>
<th>COST OF COMMUNICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>S2</td>
<td>4.0</td>
</tr>
<tr>
<td>S1</td>
<td>S3</td>
<td>14.0</td>
</tr>
<tr>
<td>S1</td>
<td>S4</td>
<td>14.0</td>
</tr>
<tr>
<td>S1</td>
<td>S5</td>
<td>2.0</td>
</tr>
<tr>
<td>S1</td>
<td>S6</td>
<td>12.0</td>
</tr>
<tr>
<td>S1</td>
<td>S7</td>
<td>9.0</td>
</tr>
<tr>
<td>S1</td>
<td>S8</td>
<td>5.0</td>
</tr>
<tr>
<td>S2</td>
<td>S3</td>
<td>14.0</td>
</tr>
<tr>
<td>S2</td>
<td>S4</td>
<td>14.0</td>
</tr>
<tr>
<td>S2</td>
<td>S5</td>
<td>2.0</td>
</tr>
<tr>
<td>S2</td>
<td>S6</td>
<td>12.0</td>
</tr>
<tr>
<td>S2</td>
<td>S7</td>
<td>9.0</td>
</tr>
<tr>
<td>S2</td>
<td>S8</td>
<td>5.0</td>
</tr>
<tr>
<td>S3</td>
<td>S4</td>
<td>4.0</td>
</tr>
<tr>
<td>S3</td>
<td>S5</td>
<td>12.0</td>
</tr>
<tr>
<td>S3</td>
<td>S6</td>
<td>2.0</td>
</tr>
<tr>
<td>S3</td>
<td>S7</td>
<td>5.0</td>
</tr>
<tr>
<td>S3</td>
<td>S8</td>
<td>9.0</td>
</tr>
<tr>
<td>S4</td>
<td>S5</td>
<td>12.0</td>
</tr>
<tr>
<td>S4</td>
<td>S6</td>
<td>2.0</td>
</tr>
<tr>
<td>S4</td>
<td>S7</td>
<td>5.0</td>
</tr>
<tr>
<td>S4</td>
<td>S8</td>
<td>9.0</td>
</tr>
<tr>
<td>S5</td>
<td>S6</td>
<td>10.0</td>
</tr>
<tr>
<td>S5</td>
<td>S7</td>
<td>7.0</td>
</tr>
<tr>
<td>S5</td>
<td>S8</td>
<td>3.0</td>
</tr>
<tr>
<td>S6</td>
<td>S7</td>
<td>3.0</td>
</tr>
<tr>
<td>S6</td>
<td>S8</td>
<td>7.0</td>
</tr>
<tr>
<td>S7</td>
<td>S8</td>
<td>4.0</td>
</tr>
<tr>
<td>TRANSACTION SITE</td>
<td>RETRIEVALS</td>
<td>UPDATES</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>A1</td>
<td>S1</td>
<td>500</td>
</tr>
<tr>
<td>A2</td>
<td>S1</td>
<td>300</td>
</tr>
<tr>
<td>A3</td>
<td>S1</td>
<td>10</td>
</tr>
<tr>
<td>A4</td>
<td>S2</td>
<td>400</td>
</tr>
<tr>
<td>A5</td>
<td>S2</td>
<td>250</td>
</tr>
<tr>
<td>A6</td>
<td>S2</td>
<td>10</td>
</tr>
<tr>
<td>A7</td>
<td>S3</td>
<td>350</td>
</tr>
<tr>
<td>A8</td>
<td>S3</td>
<td>225</td>
</tr>
<tr>
<td>A9</td>
<td>S3</td>
<td>10</td>
</tr>
<tr>
<td>A10</td>
<td>S4</td>
<td>800</td>
</tr>
<tr>
<td>A11</td>
<td>S4</td>
<td>675</td>
</tr>
<tr>
<td>A12</td>
<td>S4</td>
<td>10</td>
</tr>
<tr>
<td>A13</td>
<td>S5</td>
<td>900</td>
</tr>
<tr>
<td>A14</td>
<td>S5</td>
<td>20</td>
</tr>
<tr>
<td>A15</td>
<td>S6</td>
<td>1150</td>
</tr>
<tr>
<td>A16</td>
<td>S6</td>
<td>20</td>
</tr>
<tr>
<td>A17</td>
<td>S7</td>
<td>900</td>
</tr>
<tr>
<td>A18</td>
<td>S7</td>
<td>60</td>
</tr>
<tr>
<td>A19</td>
<td>S7</td>
<td>20</td>
</tr>
<tr>
<td>A20</td>
<td>S8</td>
<td>1150</td>
</tr>
<tr>
<td>A21</td>
<td>S8</td>
<td>50</td>
</tr>
<tr>
<td>A22</td>
<td>S8</td>
<td>20</td>
</tr>
<tr>
<td>A23</td>
<td>S8</td>
<td>30</td>
</tr>
</tbody>
</table>

**Fragments Produced**

The following segments and subsequent fragments of relations were determined during Phase I of the model.
<table>
<thead>
<tr>
<th>SITE</th>
<th>RELATION</th>
<th>FREQR</th>
<th>FREQU</th>
<th>TUPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>1150.00</td>
<td>900.00</td>
<td>11 12 13 14</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>20.00</td>
<td>0.00</td>
<td>11 12 13 14</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>900.00</td>
<td>550.00</td>
<td>11 12</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>20.00</td>
<td>0.00</td>
<td>11 12</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1150.00</td>
<td>900.00</td>
<td>13 14</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>20.00</td>
<td>0.00</td>
<td>13 14</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>300.00</td>
<td>50.00</td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>250.00</td>
<td>25.00</td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>225.00</td>
<td>25.00</td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>675.00</td>
<td>65.00</td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>60.00</td>
<td>20.00</td>
<td>2 3 6</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>50.00</td>
<td>15.00</td>
<td>1 4 5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>10.00</td>
<td>0.00</td>
<td>1 2 3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>10.00</td>
<td>0.00</td>
<td>4 5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>10.00</td>
<td>0.00</td>
<td>6 7 8</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>10.00</td>
<td>0.00</td>
<td>9 10</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>30.00</td>
<td>10.00</td>
<td>1 2 3 4 5 6 7 8</td>
</tr>
</tbody>
</table>

FRAGMENT DATA

13 FRAGMENTS

<table>
<thead>
<tr>
<th>SEGMENT(S) &amp; SITE(S)</th>
<th>RELATION</th>
<th>SIZE</th>
<th>TUPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 5 6 11 12</td>
<td>1</td>
<td>108</td>
<td>1 3 4</td>
</tr>
<tr>
<td>1 5 5 8 8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 5 6 11 12</td>
<td>1</td>
<td>72</td>
<td>6 7</td>
</tr>
<tr>
<td>2 5 5 8 8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SEGMENT(S) & SITE(S) | RELATION | SIZE | TUPLES
---|---|---|---
3 7 8 11 12 | 1 | 108 | 8 9 10
3 6 6 8 8
4 7 8 11 12 | 1 | 72 | 13 14
4 6 6 8 8
1 5 6 9 10 | 1 | 72 | 2 5
1 5 5 7 7
4 7 8 9 10 | 1 | 72 | 11 12
4 6 6 7 7
13 14 15 16 18 | 2 | 72 | 1 4 5
1 2 3 4 8
13 14 15 16 17 | 2 | 72 | 2 3 6
1 2 3 4 7
19 | 3 | 72 | 1 2 3
1
20 | 3 | 48 | 4 5
2
21 | 3 | 72 | 6 7 8
3
22 | 3 | 48 | 9 10
4
23 | 4 | 96 | 1 2 3 4 5 6 7 8
8
FRAGMENT

SALES RELATION

<table>
<thead>
<tr>
<th>CUST#</th>
<th>PROD#</th>
<th>REP#</th>
<th>DATE</th>
<th>QTY</th>
<th>UNIT_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>10</td>
<td>100</td>
<td>4/30</td>
<td>5</td>
<td>1.50</td>
</tr>
<tr>
<td>1003</td>
<td>10</td>
<td>100</td>
<td>5/11</td>
<td>5</td>
<td>1.50</td>
</tr>
<tr>
<td>1004</td>
<td>10</td>
<td>100</td>
<td>5/01</td>
<td>10</td>
<td>1.25</td>
</tr>
<tr>
<td>F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>11</td>
<td>200</td>
<td>4/30</td>
<td>100</td>
<td>2.00</td>
</tr>
<tr>
<td>1004</td>
<td>12</td>
<td>200</td>
<td>5/10</td>
<td>100</td>
<td>1.00</td>
</tr>
<tr>
<td>F3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>15</td>
<td>300</td>
<td>4/30</td>
<td>5</td>
<td>1.00</td>
</tr>
<tr>
<td>1007</td>
<td>25</td>
<td>400</td>
<td>4/30</td>
<td>5</td>
<td>7.00</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
### SALES RELATION

<table>
<thead>
<tr>
<th>CUST#</th>
<th>PROD#</th>
<th>REP#</th>
<th>DATE</th>
<th>QTY</th>
<th>UNIT_PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1012</td>
<td>15</td>
<td>800</td>
<td>5/01</td>
<td>50</td>
<td>1.25</td>
</tr>
<tr>
<td>1013</td>
<td>25</td>
<td>900</td>
<td>4/30</td>
<td>5</td>
<td>7.50</td>
</tr>
<tr>
<td>1014</td>
<td>25</td>
<td>900</td>
<td>5/10</td>
<td>5</td>
<td>7.50</td>
</tr>
<tr>
<td>1015</td>
<td>27</td>
<td>1000</td>
<td>5/01</td>
<td>15</td>
<td>1.75</td>
</tr>
<tr>
<td>1015</td>
<td>27</td>
<td>1000</td>
<td>5/10</td>
<td>7</td>
<td>1.75</td>
</tr>
<tr>
<td>1019</td>
<td>10</td>
<td>1000</td>
<td>5/01</td>
<td>10</td>
<td>1.50</td>
</tr>
<tr>
<td>1019</td>
<td>25</td>
<td>1300</td>
<td>4/30</td>
<td>2</td>
<td>7.50</td>
</tr>
</tbody>
</table>

### PRODUCTS RELATION

<table>
<thead>
<tr>
<th>PROD#</th>
<th>DIV#</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>0.75</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>5.00</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1.50</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>0.75</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
<td>1.50</td>
</tr>
</tbody>
</table>

### CUSTOMER RELATION

<table>
<thead>
<tr>
<th>CUST#</th>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Jones Co.</td>
<td>Kalamazoo, Michigan</td>
</tr>
<tr>
<td>1003</td>
<td>Brown Co.</td>
<td>Detroit, Michigan</td>
</tr>
<tr>
<td>1004</td>
<td>Smith Co.</td>
<td>Detroit, Michigan</td>
</tr>
<tr>
<td>1005</td>
<td>Acme Co.</td>
<td>Tampa, Florida</td>
</tr>
<tr>
<td>1007</td>
<td>Jones Co.</td>
<td>Miami, Florida</td>
</tr>
<tr>
<td>1012</td>
<td>Eastern Co.</td>
<td>London, England</td>
</tr>
<tr>
<td>1013</td>
<td>Western Co.</td>
<td>London, England</td>
</tr>
<tr>
<td>1014</td>
<td>Southern Co.</td>
<td>London, England</td>
</tr>
<tr>
<td>1015</td>
<td>Yurk Inc.</td>
<td>Toronto, Canada</td>
</tr>
<tr>
<td>1019</td>
<td>Northern Co.</td>
<td>Montreal, Canada</td>
</tr>
</tbody>
</table>
Derived Clusters

The following clusters and associated sites were derived by the model during Phase I. The clustering was performed with processing threshold values of 15% and a communication cost threshold of 10. See Figure 4 for a graphical representation of the clustered architecture of the system.

<table>
<thead>
<tr>
<th>CLUSTER</th>
<th>SITE(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1, 2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4, 6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>7, 8</td>
</tr>
</tbody>
</table>

CLUSTER DATA

5 CLUSTERS

<table>
<thead>
<tr>
<th>COST OF RETRIEVAL</th>
<th>COST OF UPDATE</th>
<th>COST OF SPACE</th>
<th>CLUSTER SITES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.100</td>
<td>0.200</td>
<td>0.001</td>
<td>1 2</td>
</tr>
<tr>
<td>0.110</td>
<td>0.220</td>
<td>0.001</td>
<td>3</td>
</tr>
<tr>
<td>0.210</td>
<td>0.420</td>
<td>0.002</td>
<td>4 6</td>
</tr>
<tr>
<td>0.200</td>
<td>0.420</td>
<td>0.002</td>
<td>5</td>
</tr>
<tr>
<td>0.315</td>
<td>0.515</td>
<td>0.003</td>
<td>7 8</td>
</tr>
</tbody>
</table>
Figure 4. Clustered Architecture
At the end of Phase I, the model produces the following initial allocation of the fragments to the clusters. Also shown are the calculated values for potential benefit.

```
INITIAL ALLOCATION DATA
```

### 29 ALLOCATIONS OF FRAGMENTS

<table>
<thead>
<tr>
<th>FRAGMENT</th>
<th>CLUSTER</th>
<th>PERM</th>
<th>PBI</th>
<th>FREQR</th>
<th>FREQU</th>
<th>C_SITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>T</td>
<td>-950.00</td>
<td>500.00</td>
<td>300.00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>T</td>
<td>-280.00</td>
<td>920.00</td>
<td>550.00</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>P</td>
<td>320.00</td>
<td>1170.00</td>
<td>900.00</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>T</td>
<td>-1050.00</td>
<td>400.00</td>
<td>250.00</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>T</td>
<td>-230.00</td>
<td>920.00</td>
<td>550.00</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>P</td>
<td>370.00</td>
<td>1170.00</td>
<td>900.00</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>T</td>
<td>-1450.00</td>
<td>350.00</td>
<td>225.00</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>P</td>
<td>45.00</td>
<td>1170.00</td>
<td>900.00</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>T</td>
<td>45.00</td>
<td>1170.00</td>
<td>900.00</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>P</td>
<td>1070.00</td>
<td>1970.00</td>
<td>1575.00</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>T</td>
<td>-405.00</td>
<td>1170.00</td>
<td>900.00</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>T</td>
<td>-600.00</td>
<td>500.00</td>
<td>300.00</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>P</td>
<td>70.00</td>
<td>920.00</td>
<td>550.00</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>T</td>
<td>70.00</td>
<td>920.00</td>
<td>550.00</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>P</td>
<td>1420.00</td>
<td>1970.00</td>
<td>1575.00</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>T</td>
<td>-655.00</td>
<td>920.00</td>
<td>550.00</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>T</td>
<td>445.00</td>
<td>550.00</td>
<td>75.00</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>T</td>
<td>70.00</td>
<td>225.00</td>
<td>25.00</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>P</td>
<td>560.00</td>
<td>675.00</td>
<td>65.00</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>T</td>
<td>-115.00</td>
<td>50.00</td>
<td>15.00</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>T</td>
<td>440.00</td>
<td>550.00</td>
<td>75.00</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>T</td>
<td>65.00</td>
<td>225.00</td>
<td>25.00</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>P</td>
<td>555.00</td>
<td>675.00</td>
<td>65.00</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>T</td>
<td>-105.00</td>
<td>60.00</td>
<td>20.00</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>P</td>
<td>10.00</td>
<td>10.00</td>
<td>0.00</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>P</td>
<td>10.00</td>
<td>10.00</td>
<td>0.00</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>P</td>
<td>10.00</td>
<td>10.00</td>
<td>0.00</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>P</td>
<td>10.00</td>
<td>10.00</td>
<td>0.00</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>P</td>
<td>30.00</td>
<td>30.00</td>
<td>10.00</td>
<td>8</td>
</tr>
</tbody>
</table>
Once Phase II has completed, the allocation of data fragments to clusters is as shown below. Also included are the calculated values that determine the relative benefit. The relative benefits are only calculated for clusters to which the fragment was temporarily allocated in Phase I.

### FINAL CLUSTER ALLOCATION DATA

<table>
<thead>
<tr>
<th>FRAGMENT</th>
<th>CLUSTER</th>
<th>R_CLUSTER</th>
<th>CA</th>
<th>RCN</th>
<th>RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>4219.216</td>
<td>3049.800</td>
<td>-1169.416</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
<td>6000.108</td>
<td>2657.500</td>
<td>-3342.608</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>10384.108</td>
<td>773.500</td>
<td>-9610.608</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
<td>12668.391</td>
<td>8435.700</td>
<td>-4232.691</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1430.072</td>
<td>1100.000</td>
<td>-330.072</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>5</td>
<td>4861.016</td>
<td>6624.000</td>
<td>1762.984</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td>1509.422</td>
<td>947.250</td>
<td>-562.172</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>3</td>
<td>564.391</td>
<td>312.600</td>
<td>-251.791</td>
</tr>
</tbody>
</table>

The following table shows the allocation of fragments to clusters along with the indicator of being permanently allocated to the cluster or the allocation having been cancelled at the cluster.

### 29 ALLOCATIONS OF FRAGMENTS TO CLUSTERS

<table>
<thead>
<tr>
<th>FRAGMENT</th>
<th>CLUSTER</th>
<th>PERM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>P</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>P</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>P</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The last step in the algorithm is to determine the internal allocation of fragments to sites within the cluster. Clusters that have been allocated fragments of data will be processed and the individual site allocations of fragments determined. The following are the results of this final step.

***************
FINAL SITE ALLOCATION DATA
******************

<table>
<thead>
<tr>
<th>FRAGMENT</th>
<th>SITE</th>
<th>ALLOC</th>
<th>CA</th>
<th>CN</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>FALSE</td>
<td>4063.82</td>
<td>0.00</td>
<td>-4063.82</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>TRUE</td>
<td>826.07</td>
<td>4962.25</td>
<td>4136.18</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>FALSE</td>
<td>4063.72</td>
<td>0.00</td>
<td>-4063.72</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>TRUE</td>
<td>825.97</td>
<td>4962.25</td>
<td>4136.28</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>FALSE</td>
<td>2178.22</td>
<td>0.00</td>
<td>-2178.22</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>TRUE</td>
<td>619.72</td>
<td>2541.50</td>
<td>1921.78</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>FALSE</td>
<td>2629.64</td>
<td>1768.00</td>
<td>-861.64</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>TRUE</td>
<td>2253.14</td>
<td>2541.50</td>
<td>288.36</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>TRUE</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>TRUE</td>
<td>566.97</td>
<td>3883.50</td>
<td>3316.53</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>FALSE</td>
<td>2483.47</td>
<td>0.00</td>
<td>-2483.47</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>FALSE</td>
<td>2629.64</td>
<td>1768.00</td>
<td>-861.64</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>TRUE</td>
<td>2253.14</td>
<td>2541.50</td>
<td>288.36</td>
</tr>
<tr>
<td>FRAGMENT</td>
<td>SITE</td>
<td>ALLOC</td>
<td>CA</td>
<td>CN</td>
<td>B</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>TRUE</td>
<td>145.07</td>
<td>1230.00</td>
<td>1084.93</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>TRUE</td>
<td>240.07</td>
<td>1025.00</td>
<td>784.93</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>TRUE</td>
<td>169.19</td>
<td>1491.75</td>
<td>1322.56</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>FALSE</td>
<td>157.44</td>
<td>0.00</td>
<td>-157.44</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>TRUE</td>
<td>145.07</td>
<td>1230.00</td>
<td>1084.93</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>TRUE</td>
<td>240.07</td>
<td>1025.00</td>
<td>784.93</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>TRUE</td>
<td>169.19</td>
<td>1491.75</td>
<td>1322.56</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>FALSE</td>
<td>157.44</td>
<td>0.00</td>
<td>-157.44</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>TRUE</td>
<td>1.07</td>
<td>41.00</td>
<td>39.93</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>FALSE</td>
<td>0.07</td>
<td>0.00</td>
<td>-0.07</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>FALSE</td>
<td>0.05</td>
<td>0.00</td>
<td>-0.05</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>TRUE</td>
<td>1.05</td>
<td>41.00</td>
<td>39.95</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>TRUE</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>TRUE</td>
<td>2.20</td>
<td>22.10</td>
<td>19.90</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>FALSE</td>
<td>0.10</td>
<td>0.00</td>
<td>-0.10</td>
</tr>
<tr>
<td>13</td>
<td>7</td>
<td>FALSE</td>
<td>45.44</td>
<td>0.00</td>
<td>-45.44</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>TRUE</td>
<td>14.89</td>
<td>129.45</td>
<td>114.56</td>
</tr>
</tbody>
</table>

***************
FINAL SOLUTION
***************

<table>
<thead>
<tr>
<th>FRAGMENT</th>
<th>SITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
</tr>
</tbody>
</table>
CONCLUSION

A generalized model for data allocation in distributed relational databases has been presented. The model is applicable to a network of computers where there are different communications, update and retrieval costs between the various sites.

The required input is manageable, not overly complex yet it produces a reasonably accurate cost computation. The model produces a near optimal data distribution. The model is efficient, the distribution is achieved in polynomial time. The model provides considerable improvement over previous work.
APPENDIX

Program Listing
PROGRAM MAINMENU;

(* A Generalized Data Distribution Model for Distributed *)
(* Relational Databases *)
(* Patrick D. Yurk, M.S. *)
(* Western Michigan University *)
(* August 1989 *)

(* Main Menu Program *)
(* This program is the driver program from which the user initiates *)
(* the execution of the remaining programs used in this implementation *)
(* of the distribution model. *)
(* The user is presented with a menu of processes that can be executed. *)
(* The processes are listed in the order of execution. The processes *)
(* must be executed in order at least one time for a given set of data. *)
(* Thereafter, the processes may be executed in the order desired by *)
(* user for that set of data. A new set of data requires that the *)
(* be again executed in the order listed on the menu. *)

(* *)

TYPE
STR12 = STRING[12];

VAR

(* COMMON STORAGE AREA USED FOR VARIABLES THAT ARE PASSED BETWEEN MODULES *)
(* PDR, (* PROGRAM DRIVE *)
INDR, (* INPUT FILE DRIVE *)
OUTDR : STRING[1]; (* OUTPUT FILE DRIVE *)
(* END OF COMMON STORAGE AREA *)

OPTION : CHAR;
EXIT : BOOLEAN;
EX : FILE;

(* RESET THE SCREEN AN HALT *)

PROCEDURE HALIT;
BEGIN

TEXTBACKGROUND(BLACK);
CURSCR;
HALT
END;

(* Display window with message loading option, and load the program *)
PROCEDURE LOADING(PROGRAM_NAME: STR12);
BEGIN
WINDOW(5,16,29,20);
TEXTBACKGROUND(RED);
TEXTCOLOR(YELLOW);
CURSCR;
GOTOXY(1,3);
WRITE('OK, loading option....');
ASSIGN(EX, PDR+':'+PROGRAM_NAME);
CHAIN(EX);
END;

PROCEDURE GETDRIVES;
BEGIN
(* Get the disk drives the user wishes to operate with *)
(* )
REPEATE
GOTOXY(25,4);
READLN(PDR);
IF PDR = 'Q' OR PDR = 'q' THEN
HALIT;
IF PDR = '' THEN
PDR := 'A';
UNTIL PDR = 'B' OR PDR = 'A' OR PDR = 'b' OR PDR = 'a'
OR PDR = 'C' OR PDR = 'c'
ELSE
BEGIN
GOTOXY(25,4);
WRITE(PDR)
END;

(* Get program drive from the user *)
(* )
REP;
GOTOXY(25,4);
READLN(PDR);
IF PDR = 'Q' OR PDR = 'q' THEN
HALIT;
IF PDR = '' THEN
PDR := 'A';
UNTIL PDR = 'B' OR PDR = 'A' OR PDR = 'b' OR PDR = 'a'
OR PDR = 'C' OR PDR = 'c'
ELSE
BEGIN
GOTOXY(25,4);
WRITE(PDR)
END;
BEGIN (* GET INPUT DRIVE FROM THE USER *)
IF (NOT ((INDR = 'B') OR (INDR = 'A') OR (INDR = 'b') OR (INDR = 'a')
          OR (INDR = 'C') OR (INDR = 'c'))) THEN
  REPEAT
    GOTOXY(48,4);
    READIN(INDR);
    IF (INDR = 'Q') OR (INDR = 'q') THEN
      HALT;
    IF INDR = '' THEN
      INDR := 'A';
    UNTIL (INDR = 'B') OR (INDR = 'A') OR (INDR = 'b') OR (INDR = 'a')
          OR (INDR = 'C') OR (INDR = 'c')
ELSE
  BEGIN
    GOTOXY(48,4);
    WRITE(INDR);
  END;
END (* GET OUTPUT DRIVE FROM THE USER *)
IF (NOT ((OUTDR = 'B') OR (OUTDR = 'A') OR (OUTDR = 'b') OR (OUTDR = 'a')
          OR (OUTDR = 'C') OR (OUTDR = 'c'))) THEN
  REPEAT
    GOTOXY(74,4);
    READIN(OUTDR);
    IF (OUTDR = 'Q') OR (OUTDR = 'q') THEN
      HALT;
    IF OUTDR = '' THEN
      OUTDR := 'B';
    UNTIL (OUTDR = 'B') OR (OUTDR = 'A') OR (OUTDR = 'b') OR (OUTDR = 'a')
          OR (OUTDR = 'C') OR (OUTDR = 'c')
ELSE
  BEGIN
    GOTOXY(74,4);
    WRITE(OUTDR);
  END
END; (* GETDRIVES *)

BEGIN (* MAIN PROGRAM STARTS HERE *)

TEXT BACKGROUND(2);
CLSCR;
GOTOXY(15,1);
WRITE('A Generalized Data Distribution Model for Distributed');
GOTOXY(30,2);
WRITE('Relational Databases');
GOTOXY(10,4);
WRITE('Program Drive:');
GOTOXY(35,4);
WRITE('Input Drive:');
GOTOXY(60,4);
WRITE('Output Drive:');
G0TOXY(25,7);
WRITE('1. Input Sites & Costs');
G0TOXY(25,8);
WRITE('2. Input Global Relations');
G0TOXY(25,9);
WRITE('3. Define Application Transactions');
G0TOXY(25,10);
WRITE('4. Segment the Global Relations');
G0TOXY(25,11);
WRITE('5. Fragment the Segments of the Global Relations');
G0TOXY(25,12);
WRITE('6. Determine Clusters');
G0TOXY(25,13);
WRITE('7. Phase I Network Allocations');
G0TOXY(25,14);
WRITE('8. Phase II Network Allocations');
G0TOXY(25,15);
WRITE('9. Display/Print Results of the Model');
G0TOXY(25,16);
WRITE('Q. Exit the Model');
GETDRIVES;
G0TOXY(15,22);
WRITE('Please enter desired option: ');
EXIT := FALSE;

WHILE (NOT EXIT)
BEGIN
  G0TOXY(45,22);
  CLEAR;
  READIN(OPTION);
  G0TOXY(10,24);
  CLEAR;
  IF(OPTION IN ['1','2','3','4','5','6','7','8','9', 'Q'])
  THEN
    CASE OPTION OF
      '1' : LOADING('SITECOST.CHN');
      '2' : LOADING('RELATION.CHN');
      '3' : LOADING('TRANSACT.CHN');
      '4' : LOADING('SEGMENTS.CHN');
      '5' : LOADING('FRAGMENT.CHN');
      '6' : LOADING('CLUSTERS.CHN');
      '7' : LOADING('PHASE_I.CHN');
      '8' : LOADING('PHASE_II.CHN');
      '9' : LOADING('RESULTS.CHN');
    'Q', 'q' : EXIT := TRUE
  END (* CASE *)
ELSE
  BEGIN
    G0TOXY(10,24);
    WRITE('*** Please enter an option number of only 1 through 9 or Q ***'
    EXIT := FALSE
END (* IF OPTION IN *)
END; (* WHILE NOT EXIT *)
HALIT;

END. (* MAIN PROGRAM *)

PROGRAM SITECOST;
(*******************************************************************************)
A Generalized Data Distribution Model for Distributed Relational Databases

Patrick D. Yurk, M.S.
Western Michigan University
June 1989

Program to allow the user to input the architecture of the network and the global database.

The user is presented with a data input screen with the following layout: A data display area from rows 8 through 17. Input lines from rows 19 through 21. An error message line at row 23. Lines to display function key operations are at rows 24 and 25.

If the data input by the user is correct, it is then displayed in the data display area of the screen and the next item of information is input by the user.

This program allows input of the sites, operation costs at each site, the communication costs between sites, definitions of the relations and data in the global database.

CONST
    PAGE_SIZE = 10;
    MAX_SITES = 8;
    MAX_REL = 4;

TYPE
    OP_CST = ARRAY [1..MAX_SITES, 1..3] OF REAL;
    CM_CST = ARRAY [1..MAX_SITES, 1..MAX_SITES] OF REAL;
    RL_SIZ = ARRAY [1..MAX_REL, 1..2] OF INTEGER;
    DD = STRING[15];
    STR40 = STRING[40];

VAR

REPRODUCED WITH PERMISSION OF THE COPYRIGHT OWNER. FURTHER REPRODUCTION PROHIBITED WITHOUT PERMISSION.
OP_COSTS : OP_CST;  (* ARRAY OF SITE OPERATIONAL COSTS *)
(* ONE ROW IN THE ARRAY FOR EACH SITE *)
(* COL 1 IS COST OF SPACE AT THE SITE *)
(* COL 2 IS COST OF RETRIEVAL AT THE SITE *)
(* COL 3 IS COST OF UPDATE AT THE SITE *)

COM_COSTS : CM_CST;  (* ARRAY OF COMMUNICATION COSTS BETWEEN *)
(* SITES. ONE ROW AND ONE COLUMN FOR EACH *)
(* SITE. ROW 1 COLUMN 2 IS THE COST OF *)
(* COMMUNICATION BETWEEN SITES 1 AND 2. *)
(* ROW 2 COLUMN 3 IS THE COST OF COMMUNICATION *)
(* BETWEEN SITES 2 AND 3. ETC... *)
(* COM_COSTS[I,J] = 0 WHERE I=J *)
(* COM_COSTS[I,J] = COM_COSTS[J,I] *)

REL_SIZE : RL_SIZ;  (* ARRAY OF RELATION SIZE DEFINITIONS. *)
(* ONE ROW IN THE ARRAY FOR EACH RELATION. *)
(* COL 1 IS THE NUMBER OF ROWS IN THE RELATION. *)
(* COL 2 IS THE SIZE OF EACH ROW IN THE RELATION. *)

DR  : STRING[1];
TEMPFILE,
RE_NAME,
FILENAME,
FILENAME1 : STRING[15];
FILENAMEA : ARRAY[1..10] OF STRING[15];
REALSEA,
ITEMNO,
TEMP,
YESNO,
NEWOILD : STRING[10];
FF,
INFILE : TEXT;
EX,
SS : FILE;
CH,
FUNCT : CHAR;
ITEM,
INDEX,
NUM_SITES,
PAGENUM,
RESULT : INTEGER;
NUM_STS,
CO_CO : REAL;

ERROR_FLAG,
OK : BOOLEAN;
STRING10 : ARRAY[1..10] OF STRING[10];

(* ERROR MESSAGE ***************************************************************
PROCEDURE ERROR(INDEX : INTEGER);
BEGIN (* ERROR MESSAGE *)
GOTOXY(10, 23);
TEXTCOLOR(WHITE);
CASE INDEX OF
  1 : WRITE('OUT OF RANGE !!');
  2 : WRITE('NO PREVIOUS DATA !!');
  3 : WRITE('END OF DATA !!');
  4 : WRITE('THE NUMBER OF SITE IS NOT MATCHED !!');
  5 : WRITE('THE NUMBER OF RELATION IS NOT MATCHED !!');
  6 : WRITE('THE NUMBER OF ATTRIBUTE IS NOT MATCHED THE NUMBER OF VALUE !!');
  7 : WRITE('TEMPFILE, IT DOES NOT EXIST, TRY AGAIN OR TYPE "QUIT" TO EXIT !!');
  8 : WRITE('INVALID INPUT !!');
 11 : WRITE('IT HAS TO BE REAL NUMBER !!');
 12 : WRITE('IT HAS TO BE INTEGER !!');
END;
TEXTCOLOR(YELLOW);
GOTOXY(1, 19)
END; (* ERROR MESSAGE *)

(* QUIT TO SYSTEM ***************************************************************
PROCEDURE HALT;
BEGIN (* QUIT *)
TEXTPAUSE(BLACK);
CLRSCR;
HALT
END; (* QUIT *)

(* CLEAR THE MESSAGE AND DATA INPUT AREA ***************************************************************
PROCEDURE CLEAR_MESSAGE;

VAR
  ROW : INTEGER;

BEGIN (* CLEAR MESSAGE *)
  FOR ROW := 20 TO 23 DO BEGIN
    GOTOXY(1, ROW);
    CLRCHR
  END;
END;
GOTOXY(1, 20)
END; (* CLEAR MESSAGE *)

(* SET THE PAGE FROM PAGE ONE **************************************************)
PROCEDURE INIT_PAGE(VAR RECENT_INDEX, PAGE_NUM : INTEGER);
BEGIN (* INITIALIZE PAGE *)
   RECENT_INDEX := PAGE_SIZE;
   PAGE_NUM := 1;
   GOTOXY(77, 1);
   WRITE(PAGE_NUM)
END; (* INITIALIZE PAGE *)

(* INCREMENT BY ONE PAGE *******************************************************)
PROCEDURE INCR_PAGE;
BEGIN (* INCREASE PAGE *)
   PAGE_NUM := PAGE_NUM + 1;
   GOTOXY(77, 1);
   WRITE(PAGE_NUM);
   INDEX := INDEX + PAGE_SIZE
END; (* INCREASE PAGE *)

(* DECREMENT BY ONE PAGE *******************************************************)
PROCEDURE DECR_PAGE;
BEGIN (* DECREASE PAGE *)
   PAGE_NUM := PAGE_NUM - 1;
   IF PAGE_NUM < 1 THEN PAGE_NUM := 1;
   GOTOXY(77, 1);
   WRITE(PAGE_NUM);
   INDEX := INDEX - PAGE_SIZE;
   IF INDEX < PAGE_SIZE THEN INDEX := PAGE_SIZE
END; (* DECREASE PAGE *)

(* MOVE THE CURSOR TO THE ORIGINAL POSITION WHENEVER INPUT DATA IS WRONG *****)
PROCEDURE AGAIN(Y : INTEGER);
BEGIN (* AGAIN *)
   GOTOXY(1, Y);
   CIREOL
END; (* AGAIN *)

(* HELP COMMAND TO ASSIST USERS TO UNDERSTAND HOW TO EXECUTE THE PROJECT *****)
PROCEDURE HELP_COM(FHELP : DD);

VAR
   HELP : ARRAY[1..200] OF STRING[80];
   HFUNCT : CHAR;
   LAST,
   RECENT_HELP,
HELP_NUM : INTEGER;

(* CLEAR THE HELP SCREEN *******************************************************)
PROCEDURE CLEAR_HELP;

VAR
  ROW : INTEGER;
BEGIN (* CLEAR HELP *)
  FOR ROW := 1 TO 17 DO
    BEGIN
      GOTOXY(1, ROW);
      CLEAR
    END
  END; (* CLEAR HELP *)

(* LOAD INFORMATION FROM THE HELP FILE ******************************************)
PROCEDURE READ_HELP;

VAR
  I : INTEGER;
  HELPFILE : TEXT;
BEGIN (* READ HELP *)
  I := 0;
  ASSIGN(HELPFILE, HELP);  
  RESET(HELPFILE);
  WHILE NOT EOF(HELPFILE) DO
    BEGIN
      I := I + 1;
      READIN(HELPFILE, HELP[I])
    END;
  CLOSE(HELPFILE);
  LAST := I
END; (* READ HELP *)

(* INCREMENT ONE PAGE FROM THE HELP FILE ******************************************)
PROCEDURE INCR_HELP;

BEGIN (* INCREASE PAGE OF HELP *)
  CLEAR_HELP;
  HELP_NUM := HELP_NUM + 1;
  GOTOXY(71, 1);
  TEXTCOLOR(RED);
  WRITE('PAGE ');
  TEXTCOLOR(YELLOW);
  WRITE(HELP_NUM);
  RECENT_HELP := RECENT_HELP + 16
END; (* INCREASE PAGE OF HELP *)

(* DECREMENT ONE PAGE FROM THE HELP FILE ******************************************)
PROCEDURE DECR_HELP;

BEGIN (* DECREASE PAGE OF HELP *)
CLEAR HELP;
HELP_NUM := HELP_NUM - 1;
GOTOXY(71, 1);
TEXTCOLOR(RED);
WRITE('PAGE ');
TEXTCOLOR(YELLOW);
WRITE(HELP_NUM);
RECENT_HELP := RECENT_HELP - 16
END; (* DECREASE PAGE OF HELP *)

(* SHOW HELP FILE ON THE SCREEN ***********************************************************)
PROCEDURE SHOW.Screen_HELP(INDEX: INTEGER);

VAR
  I : INTEGER;
BEGIN (* SHOW HELP SCREEN *)
  GOTOXY(1, 2);
  FOR I := RECENT_HELP - 15 TO RECENT_HELP DO
    IF I <= INDEX THEN
      WRITELN(HELP[I])
  END; (* SHOW HELP SCREEN *)

(* GO TO THE NEXT PAGE OF THE HELP FILE ***********************************************************)
PROCEDURE NEXT_PAGE_HELP;

BEGIN (* NEXT PAGE HELP *)
  IF RECENT_HELP < LAST THEN
    BEGIN
      INCR_HELP;
      SHOW_SCREEN_HELP(LAST)
    END
  ELSE
    ERROR(3)
  END; (* NEXT PAGE HELP *)

(* GO TO THE PREVIOUS PAGE OF THE HELP FILE ***********************************************************)
PROCEDURE PREV_PAGE_HELP;

BEGIN (* PREVIOUS PAGE HELP *)
  IF RECENT_HELP > 16 THEN
    BEGIN
      DECR_HELP;
      SHOW_SCREEN_HELP(LAST)
    END
  ELSE
    ERROR(2)
  END; (* PREVIOUS PAGE HELP *)

BEGIN (* HELP COMMAND *)
  READ HELP;
  CLEAR_HELP;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
RECENT_HELPC:=16;
HELP_NUM:=1;
GOTOXY(71,1);
TEXTCOLOR(RED);
WRITE('PAGE ');
TEXTCOLOR(YELLOW);
WRITE(HELP_NUM);
GOTOXY(1,24);
CIRCEL;
GOTOXY(1,25);
CIRCEL;
TEXTCOLOR(RED);
WRITE('E => END HELP  N => NEXT PAGE  P => PREVIOUS PAGE ');
WRITE('Q => QUIT TO SYSTEM ');
TEXTCOLOR(YELLOW);
SHOW_SCREEN_HELP(LAST);
HFUNCT:=' '; 
GOTOXY(1,19);
WRITE('FUNCTION : ');
WHILE (HFUNCT <> 'E') AND (HFUNCT <> 'e') DO
    BEGIN
        GOTOXY(15,19);
        CIRCEL;
        READLN(HFUNCT);
        CLEAR_MESSAGE;
        IF (HFUNCT <> 'E') AND (HFUNCT <> 'e') THEN
            CASE HFUNCT OF
                'N','n': NEXT_PAGE_HELP;
                'P','p': PREVIOUS_PAGE_HELP;
                'Q','q': HALT;
                ELSE ERROR(8)
            END
        END
    END
END; (* HELP COMMAND *)

(* SHOW THE COMMUNICATION COSTS SCREEN *)
PROCEDURE SHOW_SCREEN_COMM_COST(INDEX: INTEGER);

VAR
I,
LINE,
SITEA,
SITEB: INTEGER;

BEGIN (* SHOW SCREEN COMMUNICATION COSTS *)

FOR I := (INDEX - (PAGE_SIZE - 1)) TO INDEX DO
    BEGIN
        SITEA := ROUND(0.499+(I/NUM_SITES));
        SITEB := I MOD NUM_SITES;
        IF SITEB = 0 THEN SITEB := NUM_SITES;
        LINE := 7 + (I MOD PAGE_SIZE);

        ...
IF LINE = 7 THEN LINE := 7 + PAGE_SIZE;
GOTOXY(7, LINE);
WRITE(SITEA:2, ',', SITEB:2, ',', COM_COSTS[SITEA,SITEB]:6:3)
END

(* SHOW SCREEN COMMUNICATION COSTS *)

(* GO TO THE NEXT PAGE OF COMMUNICATION COSTS DATA *)
PROCEDURE NEXT_PAGE_COMCOAST;

BEGIN (* NEXT_PAGE_COMCOAST *)
    IF INDEX < (NUM_SITES*NUM_SITES) THEN
        BEGIN
            INCR_PAGE;
            SHOW_SCREEN_COMCOAST(INDEX)
        END
    ELSE
        BEGIN
            INDEX := NUM_SITES*NUM_SITES;
            ERROR(3)
        END
    END; (* NEXT_PAGE_COMCOAST *)

(* GO TO THE PREVIOUS PAGE OF COMMUNICATION COSTS DATA *)
PROCEDURE PREVIOUS_PAGE_COMCOAST;

BEGIN (* PREVIOUS_PAGE_COMCOAST *)
    IF INDEX > PAGE_SIZE THEN
        BEGIN
            DECR_PAGE;
            SHOW_SCREEN_COMCOAST(INDEX)
        END
    ELSE
        ERROR(2)
    END; (* PREVIOUS_PAGE_COMCOAST *)

(* CHECK THE FILE IF IT EXISTS OR NOT, NEW OR OLD FILE **********************)
PROCEDURE CHECK_FILE(VAR FILENAME : DD);

BEGIN (* CHECK FILE *)
    FILENAME := '1';
    REPEAT
        WRITE('ENTER FILE NAME : ');
        CLEAR;
        READIN(FILENAME);
        IF (FILENAME = 'Q') OR (FILENAME = 'q') THEN
            HALT
        ELSE
            BEGIN
                CLEAR MESSAGE;
                NEWOLD := '1';
                WRITELN;
                WHILE ((NEWOLD <> 'NEW') AND (NEWOLD <> 'OLD'))
AND (NEWOLD <> 'new') AND (NEWOLD <> 'old') DO BEGIN
    AGAIN(WHEREY-1);
    WRITE('NEW or OLD? :');
    READIN(NEWOLD);
    IF (NEWOLD = 'Q') OR (NEWOLD = 'q') THEN HALT
END;
ASSIGN(INFILE, INDR + ': ' + FILENAME + '.IN');
($1-RESET(INFILE) ($1+);
OK := (IRESULT = 0);
CLOSE(INFILE);
IF (((NEWOLD = 'OLD') OR (NEWOLD = 'old')) AND (NOT OK)) THEN BEGIN
    TEMPFILE := FILENAME;
    ERROR(7)
END ELSE IF (((NEWOLD = 'NEW') OR (NEWOLD = 'new')) AND (OK)) THEN BEGIN
    WRITELN;
    YESNO := '';
    WRITELN(FILENAME, ' ALREADY EXISTS. ');
    WRITELN;
    WHILE ((YESNO <> 'YES') AND (YESNO <> 'NO')
        AND (YESNO <> 'yes') AND (YESNO <> 'no')) DO BEGIN
        AGAIN(WHEREY-1);
        WRITE('DO YOU WANT TO INPUT ');
        WRITE('ANOTHER FILE NAME? (YES/NO) : ');
        READIN(YESNO);
        IF (YESNO = 'Q') OR (YESNO = 'q') THEN HALT
END;
    IF (YESNO = 'YES') OR (YESNO = 'yes') THEN BEGIN
        AGAIN(WHEREY-5);
        OK := FALSE
    END ELSE
    OK := TRUE
END
CLEAR_MESSAGE
END; (* CHECK FILE *)

(* LOAD THE SITE DATA FROM THE DISK ***************************************)
PROCEDURE COPY_FROM_SITE;

VAR
  I,
BEGIN (* COPY FROM SITE *)
   ASSIGN(INFILE, INDR+=''+FILENAME+'.IN');
   RESET(INFILE);
   READIN(INFILE, NUM_SITES);
   READIN(INFILE);
   FOR I := 1 TO NUM_SITES DO
      FOR J := 1 TO 3 DO
         READIN(INFILE, OP_COSTS[I,J]);
   READIN(INFILE);
   FOR I := 1 TO NUM_SITES DO
      FOR J := 1 TO NUM_SITES DO
         READIN(INFILE, COM_COSTS[I,J]);
   CLOSE(INFILE)
END; (* COPY FROM SITE *)

(* STORE THE SITE DATA TO THE DISK ***********************************************************)
PROCEDURE COPY_TO_SITE;

VAR
   I,
   J : INTEGER;

BEGIN (* COPY TO SITE *)
   ASSIGN(INFILE, INDR+=''+FILENAME+'.IN');
   REWRITE(INFILE);
   WRITEIN(INFILE, NUM_SITES:2);
   WRITEIN(INFILE);
   FOR I := 1 TO NUM_SITES DO
      FOR J := 1 TO 3 DO
         WRITEIN(INFILE, OP_COSTS[I,J]:6:3);
   WRITEIN(INFILE);
   FOR I := 1 TO NUM_SITES DO
      FOR J := 1 TO NUM_SITES DO
         WRITEIN(INFILE, COM_COSTS[I,J]:6:3);
   CLOSE(INFILE)
END; (* COPY TO SITE *)

(* DISPLAY THE VALUE ON THE DATA AREA OF THE SITE *)
PROCEDURE COST_MARK(STRING40 : STR40; Y : INTEGER);

BEGIN (* COST_MARK *)
   GOTOXY(1,Y);
   WRITE(STRING40);
   CLREOL
END; (* COST_MARK *)

(* SHOW THE SCREEN FORMAT FOR INPUT OF COMMUNICATION COSTS *)
PROCEDURE SCREEN_COMCOST;

VAR
COUNT: INTEGER;

BEGIN (* SCREEN_COMMCOST *)
  CLRSCR;
  TEXTCOLOR(RED);
  WRITELN('":30, 'COMMUNICATION COSTS', '":13, 'PAGE');
  WRITELN('":30, 'COMMUNICATION');
  WRITELN('NUMBER OF SITES: ');
  WRITELN('SITEA SITEB COST');
  WRITELN('---------------------');
  TEXTCOLOR(YELLOW)
END; (* SCREEN_COMMCOST *)

(* SHOW THE SCREEN FORMAT FOR INPUT OF SITE INFO *)
PROCEDURE SCREEN_COST;
BEGIN (* SCREEN_COST *)
  CLRSCR;
  TEXTCOLOR(RED);
  WRITELN('":22, 'SYSTEM SITES & COSTS');
  WRITELN('":22, '****************************');
  GOTOXY(1, 4);
  WRITELN('NUMBER OF SITES: ');
  GOTOXY(1, 6);
  WRITELN('SITE COST OF SPACE COST OF RETRIEVAL COST OF UPDATE');
  WRITELN('--------------------- --------------------- ---------------------');
  (* SHOW THE FUNCTION KEYS ON ROW 24 *)
  GOTOXY(1, 24);
  WRITELN('Q => QUIT TO SYSTEM');
  TEXTCOLOR(YELLOW);
  GOTOXY(1, 19)
END; (* SCREEN_COST *)

(* DISPLAY THE DATA VALES FOR THE SITE COSTS *)
PROCEDURE COST_VALUE;

VAR
  I: INTEGER;

BEGIN (* COST_VALUE *)
  GOTOXY(18, 4);
  WRITE(NUM_SITES:2);
  FOR I := 1 TO NUM_SITES DO
    BEGIN
      GOTOXY(2, I+7);
      WRITE(I:2);
      GOTOXY(12, I+7);
      WRITE(OP_COSTS[I, 1]:6:3);
      GOTOXY(30, I+7);
WRITE(OP\_COSTS[I,2]:6:3);
GOTOXY(50,I+7);
WRITE(OP\_COSTS[I,3]:6:3);
END

BEGIN (* COST\_VALUE *)

(* SHOW FUNCTION KEYS OF TYPE, NEW AND OLD, ON THE ROW 24 ***********)
PROCEDURE\_FUNCT\_NEW;
BEGIN (* COST FUNCTION *)
GOTOXY(1, 24);
TEXT\_COLOR\(RED\);
WRITE('H => HELP
Q => QUIT TO SYSTEM');
TEXT\_COLOR\(YELLOW\);
GOTOXY(1, 19)
END; (* COST FUNCTION *)

(* SHOW FUNCTION KEYS FOR SITE ON LAST TWO ROWS ***********)
PROCEDURE\_FUNCT\_COST;
BEGIN (* COST FUNCTION *)
GOTOXY(1, 24);
TEXT\_COLOR\(RED\);
WRITE('R => REPLACE');
Q \Rightarrow\ \text{QUIT TO SYSTEM}');
TEXT\_COLOR\(YELLOW\);
GOTOXY(1, 19)
END; (* COST FUNCTION *)

(* SHOW FUNCTION KEYS FOR COMMUNICATION COSTS ON LAST TWO ROWS*)
PROCEDURE\_FUNCT\_COM\_COST;
BEGIN (* COMMUNICATION FUNCTION *)
GOTOXY(1, 24);
TEXT\_COLOR\(RED\);
WRITE('E \Rightarrow END MODIFY
H \Rightarrow HELP
R \Rightarrow REPLACE');
Q \Rightarrow\ \text{QUIT TO SYSTEM}');
TEXT\_COLOR\(YELLOW\);
GOTOXY(1, 19)
END; (* COMMUNICATION FUNCTION *)

(* INPUT COST DATA OF SITE ***********)
PROCEDURE\_ENTER\_COST(S1 : STR40; VAR CO : REAL; INDEX1, INDEX2 : INTEGER);
BEGIN (* ENTER COST *)
REPEAT
COST\_MARK(S1, 19);
READ\(REAL\_STR\);\nVAL\(REAL\_STR, CO, RESULT\);
IF (REAL\_STR = 'Q' OR (REAL\_STR = 'q')) THEN
HALT
ELSE
IF (REALSTR = 'H') OR (REALSTR = 'h') THEN
BEGIN
    HELP_COM(FDR+'HELP1.HLP');
    SCREEN_COST;
    COST_VALUE;
    FUNCT_NEW
END
ELSE
IF RESULT <> 0 THEN
    ERROR(8)
UNTIL RESULT = 0;
CLEAR_MESSAGE;
GOTOXY(INDEX1,INDEX2);
WRITE(C0)
END; (* ENTER COST *)

(* INPUT NEW COST AND SIZE DATA OF THE SITE *******************************************)
PROCEDURE NEW_COST;
VAR
  I : INTEGER;
BEGIN (* NEW COST *)
  ENTER_COST('NUMBER OF SITES: ', NUM_STS, 18, 4);
  NUM_SITES := TRUNC(NUM_STS);
  FOR I := 1 TO NUM_SITES DO
  BEGIN
    ENTER_COST('SITE: '+CHR(48+I) + ' COST OF SPACE : ', OP_COSTS[I,1], 12, I+7
    ENTER_COST('SITE: '+CHR(48+I) + ' COST OF RETRIEVAL : ', OP_COSTS[I,2], 30,
    ENTER_COST('SITE: '+CHR(48+I) + ' COST OF UPDATE : ', OP_COSTS[I,3], 50, I+
  END
END; (* NEW COST *)

(* INPUT COST OF COMMUNICATIONS BETWEEN SITES A AND B *)
PROCEDURE NEW_COMCOST;
VAR
  SITEA,
  SITEB,
  RESULT : INTEGER;
BEGIN (* NEW COMCOST *)
  FOR SITEA := 1 TO NUM_SITES DO
  BEGIN
    FOR SITEB := 1 TO NUM_SITES DO
    BEGIN
      IF SITEA = SITEB THEN
        COM_COSTS[SITEA,SITEB] := 0
      ELSE

IF COM_COSTS[SITEA, SITEB] = -1 THEN
BEGIN
  REPEAT
    GOTOXY(1, 19);
    WRITE('SITE ', SITEA, ' TO SITE ', SITEB, ' COMMUNICATION COST: '); 
    READLN(TEMP);
    VAL(TEMP, COM_COSTS[SITEA, SITEB], RESULT);
    IF (TEMP = 'Q') OR (TEMP = 'q') THEN
      HALT
    ELSE
      IF (TEMP = 'H') OR (TEMP = 'h') THEN
        BEGIN
          HELP_COM(PDR+'HELP1.HLP');
          SCREEN_COMCOST;
          GOTOXY(76, 1);
          WRITE(PAGE_NUM);
          SHOW_SCREEN_COMCOST(INDEX - 1);
          FUNCT_NEW
        END
      ELSE
        IF (RESULT <> 0) THEN
          ERROR(8);
        UNTIL (RESULT = 0);
    COM_COSTS[SITEB, SITEA] := COM_COSTS[SITEA, SITEB];
    SHOW_SCREEN_COMCOST(INDEX)
  END (* IF = -1 *)
END (* FOR SITEB *)
END (* FOR SITEA *)
END; (* NEW COMCOST *)

(* REPLACE THE COST OF SITE *************************************************)
PROCEDURE REPLACE_COST;
VAR
  TEMPSSTR : STRING[10];
  INTEMP  : REAL;
  SITENO   : INTEGER;
BEGIN (* REPLACE COST *)
  WRITE('SITE NO. : ');
  SITENO := ' '; 
  READIN(SITENO);
  VAL(SITENO, ITEM, RESULT);
  IF (ITEM > 0) AND (ITEM <= NUM_SITES) AND (SITENO <> '') AND (RESULT = 0) THEN
    BEGIN
      WRITE('SITE ',ITEM,' SPACE: ');
      TEMPSSTR := ' '; 
      READ(TEMPSSTR);
      VAL(TEMPSSTR, INTEMP, RESULT);
      IF (TEMPSSTR < ' ') AND (RESULT = 0) THEN
        OP_COSTS[ITEM,1] := INTEMP
ELSE
    ERROR(11);
WRITE(' SITE ',ITEM, ' COST OF RETRIEVAL: ');
TEMPSTR := '1';
READ(TEMPSTR);
VAL(TEMPSTR, INTEMP, RESULT);
IF (TEMPSTR <> '1') AND (RESULT = 0) THEN
    OP_COSTS[ITEM,2] := INTEMP
ELSE
    ERROR(11);
WRITE(' SITE ',ITEM, ' COST OF UPDATE: ');
TEMPSTR := '1';
READ(TEMPSTR);
VAL(TEMPSTR, INTEMP, RESULT);
IF (TEMPSTR <> '1') AND (RESULT = 0) THEN
    OP_COSTS[ITEM,3] := INTEMP
ELSE
    ERROR(11)
END
ELSE
    ERROR(12)
END; (* REPLACE COST *)

(* MODIFY COST AND SIZE DATA OF SITE ***********************************************************)
PROCEDURE OLD_COST;
BEGIN (* OLD COST *)
    FUNCT := '1';
    WHILE (FUNCT <> 'E') AND (FUNCT <> 'e') DO
        BEGIN
            COST_VALUE;
            FUNCT_COST;
            GOTOXY(1, 19);
            WRITE(' FUNCTION : ');
            CLEOREL;
            READIN(FUNCT);
            CLEAR_MESSAGE;
            IF (FUNCT <> 'E') AND (FUNCT <> 'e') THEN
                CASE FUNCT OF
                    'H', 'h' : BEGIN
                        HELP_COMM(FDR+:HELP1.HLP');
                        SCREEN_COST;
                        COST_VALUE;
                        FUNCT_COST
                    END;
                    'R', 'r' : REPLACE_COST;
                    'Q', 'q' : HALT;
                    ELSE ERROR(8)
                    END (* CASE *)
                END (* WHILE *)
            END; (* OLD_COST *)
(* REPLACE COMMUNICATION COSTS DATA *)
PROCEDURE REPLACE_COMCOST;

VAR
  SITEA,
  SITEB,
  RESULT : INTEGER;

BEGIN (* REPLACE_COMCOST *)
  REPEAT
    GOTOXY(1, 19);
    CLEAR;
    WRITE('SITEA: ');  READIN(TEMP);
    VAL(TEMP, SITEA, RESULT);
    IF (TEMP = 'Q') OR (TEMP = 'q') THEN HALT
    ELSE IF (TEMP = 'H') OR (TEMP = 'h') THEN BEGIN
      HELP_COM(PDR+'HELP1.HLP');
      SCREEN_COMCOST;
      GOTOXY(76, 1);
      WRITE(PAGE_NUM);
      SHOW_SCREEN_COMCOST(INDEX - 1);
      FUNCT_NEW
    END ELSE IF (RESULT <> 0) THEN ERROR(8);
    UNTIL (RESULT = 0);
  REPEAT
    GOTOXY(15, 19);
    WRITE('SITEB: ');  READIN(TEMP);
    VAL(TEMP, SITEB, RESULT);
    IF (TEMP = 'Q') OR (TEMP = 'q') THEN HALT
    ELSE IF (TEMP = 'H') OR (TEMP = 'h') THEN BEGIN
      HELP_COM(PDR+'HELP1.HLP');
      SCREEN_COMCOST;
      GOTOXY(76, 1);
      WRITE(PAGE_NUM);
      SHOW_SCREEN_COMCOST(INDEX);
      FUNCT_NEW
    END ELSE IF (RESULT <> 0) THEN ERROR(8);
    UNTIL (RESULT = 0);
REPEAT

GOTOXY(1, 19);
WRITE('SITE ', SITEA,' TO SITE ', SITEB,' COMMUNICATION COST: '); 
READIN(TMP);
VAL(TMP, COMM_COST[SITEA, SITEB], RESULT);
IF (TMP = 'Q' OR (TMP = 'q')) THEN
HAILIT
ELSE
IF (TMP = 'H' OR (TMP = 'h')) THEN
BEGIN
HELP_COM(PDR+'HELP1.HLP');
SCREEN_COMCOST;
GOTOXY(76, 1);
WRITE(PAGE_NUM);
SHOW_SCREEN_COMCOST(INDEX);
FUNCT_NEW
END
ELSE
IF (RESULT <> 0) THEN
ERROR(8);
UNTIL (RESULT = 0);
COMM_COST[SITEB, SITEA] := COMM_COST[SITEA, SITEB];
SHOW_SCREEN_COMCOST(INDEX);
END; (* REPLACE_COMCOST *)

(* MODIFY OLD COMMUNICATION COST DATA *)
PROCEDURE OLD_COMCOST;

BEGIN (* OLD COMCOST *)
SHOW_SCREEN_COMCOST(INDEX);
FUNCT_COMCOST;
FUNCT := ' ';
WHILE (FUNCT <> 'E') AND (FUNCT <> 'e') DO
BEGIN
GOTOXY(1, 19);
WRITE('FUNCTION : '); 
CIREOL;
READIN(FUNCT);
CLEAR_MESSAGE;
IF (FUNCT <> 'E') AND (FUNCT <> 'e') THEN
CASE FUNCT OF
'H', 'h' : BEGIN
HELP_COM(PDR+'HELP1.HLP');
SCREEN_COMCOST;
GOTOXY(76, 1);
WRITE(PAGE_NUM);
SHOW_SCREEN_COMCOST(INDEX);
FUNCT_COMCOST
END;

'N', 'n' : NEXT_PAGE_COMCOST;
'P', 'p' : PREV_PAGE_COMCOST;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
'R', 'r': REPLACE_COMCOST;
'Q', 'q': HALIT;
ELSE ERROR(8)
END

END; (* OLD COMCOST *)

(* SELECT DRIVER A OR B, THEN EXECUTE THE APPROPRIATE ROUTINE OF SITE
   ACCORDING TO THE NEW OR OLD SITE FILE *******************************************)

BEGIN (* MAIN *)
  WINDOW(1,1,80,25);
  TEXTBACKGROUND(BLACK);
  CLRSCR;
  SCREEN_COMCOST;

  CHECK_FILE(FILENAME);
  CLEAR_MESSAGE;

  IF (NEWOLD = 'NEW') OR (NEWOLD = 'new') THEN
    BEGIN
      FUNCT_NEW;
      NEW_COST;
      OLD_COST;
      SCREEN_COMCOST;
      FUNCT_NEW;
      INIT_PAGE(INDEX, PAGE_NUM);
      NEW_COMCOST;
      OLD_COMCOST;
    END

  ELSE
  IF (NEWOLD = 'OLD') OR (NEWOLD = 'old') THEN
    BEGIN
      COPY_FROM_SITE;
      OLD_COST;
      SCREEN_COMCOST;
      INIT_PAGE(INDEX, PAGE_NUM);
      OLD_COMCOST;
    END

  ELSE
    HALIT;
    COPY_TO_SITE;
    ASSIGN(EX, PERF'MAINMENU.COM');
    EXECUTE(EX)
  END

PROGRAM RELATION;

(********************************************************************************)
(*
(* A Generalized Data Distribution Model for Distributed
(* Relational Databases
(*
(*
Program to allow the user to input the architecture of the network and the global database.

The user is presented with a data input screen with the following layout. A data display area from rows 8 through 17. Input lines from rows 19 through 21. An error message line at row 23. Lines to display function key operations are at rows 24 and 25.

If the data input by the user is correct, it is then displayed in the data display area of the screen and the next item of information is input by the user.

This program allows input of the sites, operation costs at each site, the communication costs between sites, definitions of the relations and data in the global database.

***************************************************************************

CONST
PAGE_SIZE = 10;
MAX_SITES = 8;
MAX_RELIS = 4;

TYPE
TEMP_TY = PACKED ARRAY[1..12] OF CHAR;
DD = STRING[15];
EE = STRING[10];

VAR
***************************************************************************

(* COMMON STORAGE AREA USED FOR VARIABLES THAT ARE PASSED BETWEEN MODULES *)
(* *)
(* PDR, Indr, Outdr : STRING[1]; (* PROGRAM DRIVE * )
(* INPUT FILE DRIVE * )
(* OUTPUT FILE DRIVE * )
(* *)
(* END OF COMMON STORAGE AREA *)
***************************************************************************

DATA2, INFILE1 : TEXT;
DR : STRING[1];
TEMPFILE,
RE_NAME,
FILENAME,
FILENAME1 : STRING[15];
FILENAMEA : ARRAY[1..10] OF STRING[15];
SINDEX,
NUM_REL : INTEGER;
REALSTR,
ITEMNO,
YESNO,
MODI,
NEWOLD : STRING[5];
INFILE,
FF : TEXT;
SS : FILE;
FUNCT : CHAR;
ROW,
COL,
ITEM,
INDEX,
INDEX1,
INDEX2,
INDEX3,
INDEX4,
INDEX5,
INDEX6,
INDEX7,
INDEX8,
INDEX9,
INDEX10,
INDEX11,
INDEX12,
INDEX13,
INDEX14,
INDEX15,
INDEX16,
INDEX17,
INDEX18,
INDEX19,
INDEX20,
INDEX21,
INDEX22,
INDEX23,
INDEX24,
INDEX25,
INDEX26,
INDEX27,
INDEX28,
INDEX29,
INDEX30,
INDEX31,
INDEX32,
INDEX33,
INDEX34,
INDEX35,
INDEX36,
INDEX37,
INDEX38,
INDEX39,
INDEX40,
INDEX41,
INDEX42,
INDEX43,
INDEX44,
INDEX45,
INDEX46,
INDEX47,
INDEX48,
INDEX49,
INDEX50,
INDEX51,
INDEX52,
INDEX53,
INDEX54,
INDEX55,
INDEX56,
INDEX57,
INDEX58,
INDEX59,
INDEX60,
INDEX61,
INDEX62,
INDEX63,
INDEX64,
INDEX65,
INDEX66,
INDEX67,
INDEX68,
INDEX69,
INDEX70,
INDEX71,
INDEX72,
INDEX73,
INDEX74,
INDEX75,
INDEX76,
INDEX77,
INDEX78,
INDEX79,
INDEX80,
INDEX81,
INDEX82,
INDEX83,
INDEX84,
INDEX85,
INDEX86,
INDEX87,
INDEX88,
INDEX89,
INDEX90,
INDEX91,
INDEX92,
INDEX93,
INDEX94,
INDEX95,
INDEX96,
INDEX97,
INDEX98,
INDEX99,
INDEX100,
INDEX101,
INDEX102,
INDEX103,
INDEX104,
INDEX105,
INDEX106,
INDEX107,
INDEX108,
INDEX109,
INDEX110,
INDEX111,
INDEX112,
INDEX113,
INDEX114,
INDEX115,
INDEX116,
INDEX117,
INDEX118,
INDEX119,
INDEX120,
INDEX121,
INDEX122,
INDEX123,
INDEX124,
INDEX125,
INDEX126,
INDEX127,
INDEX128,
INDEX129,
INDEX130,
INDEX131,
INDEX132,
INDEX133,
INDEX134,
INDEX135,
INDEX136,
INDEX137,
INDEX138,
INDEX139,
INDEX140,
INDEX141,
INDEX142,
INDEX143,
INDEX144,
INDEX145,
INDEX146,
INDEX147,
INDEX148,
INDEX149,
INDEX150,
INDEX151,
INDEX152,
INDEX153,
INDEX154,
INDEX155,
INDEX156,
INDEX157,
INDEX158,
INDEX159,
INDEX160,
INDEX161,
INDEX162,
INDEX163,
INDEX164,
INDEX165,
INDEX166,
INDEX167,
INDEX168,
INDEX169,
INDEX170,
INDEX171,
INDEX172,
INDEX173,
INDEX174,
INDEX175,
INDEX176,
INDEX177,
INDEX178,
INDEX179,
INDEX180,
INDEX181,
INDEX182,
INDEX183,
INDEX184,
INDEX185,
INDEX186,
INDEX187,
INDEX188,
INDEX189,
INDEX190,
INDEX191,
INDEX192,
INDEX193,
INDEX194,
INDEX195,
INDEX196,
INDEX197,
INDEX198,
INDEX199,
INDEX200,
INDEX201,
INDEX202,
INDEX203,
INDEX204,
INDEX205,
INDEX206,
INDEX207,
INDEX208,
INDEX209,
INDEX210,
INDEX211,
INDEX212,
INDEX213,
INDEX214,
INDEX215,
INDEX216,
INDEX217,
INDEX218,
INDEX219,
INDEX220,
INDEX221,
INDEX222,
INDEX223,
INDEX224,
INDEX225,
INDEX226,
INDEX227,
INDEX228,
INDEX229,
INDEX230,
INDEX231,
INDEX232,
INDEX233,
INDEX234,
INDEX235,
INDEX236,
INDEX237,
INDEX238,
INDEX239,
INDEX240,
INDEX241,
INDEX242,
INDEX243,
INDEX244,
INDEX245,
INDEX246,
INDEX247,
INDEX248,
INDEX249,
INDEX250,
INDEX251,
INDEX252,
INDEX253,
INDEX254,
INDEX255,
INDEX256,
INDEX257,
INDEX258,
INDEX259,
INDEX260,
INDEX261,
INDEX262,
INDEX263,
INDEX264,
INDEX265,
INDEX266,
INDEX267,
INDEX268,
INDEX269,
INDEX270,
INDEX271,
INDEX272,
INDEX273,
INDEX274,
INDEX275,
INDEX276,
INDEX277,
INDEX278,
INDEX279,
INDEX280,
INDEX281,
INDEX282,
INDEX283,
INDEX284,
INDEX285,
INDEX286,
INDEX287,
INDEX288,
INDEX289,
INDEX290,
INDEX291,
INDEX292,
INDEX293,
INDEX294,
INDEX295,
INDEX296,
INDEX297,
INDEX298,
INDEX299,
INDEX300,
INDEX301,
INDEX302,
INDEX303,
INDEX304,
INDEX305,
INDEX306,
INDEX307,
INDEX308,
INDEX309,
INDEX310,
INDEX311,
INDEX312,
INDEX313,
INDEX314,
INDEX315,
INDEX316,
INDEX317,
INDEX318,
INDEX319,
INDEX320,
INDEX321,
INDEX322,
INDEX323,
INDEX324,
INDEX325,
INDEX326,
INDEX327,
INDEX328,
INDEX329,
INDEX330,
INDEX331,
INDEX332,
INDEX333,
INDEX334,
INDEX335,
INDEX336,
INDEX337,
INDEX338,
INDEX339,
INDEX340,
INDEX341,
INDEX342,
INDEX343,
INDEX344,
INDEX345,
INDEX346,
INDEX347,
INDEX348,
INDEX349,
INDEX350,
INDEX351,
INDEX352,
INDEX353,
INDEX354,
INDEX355,
INDEX356,
INDEX357,
INDEX358,
INDEX359,
INDEX360,
INDEX361,
INDEX362,
INDEX363,
INDEX364,
INDEX365,
INDEX366,
INDEX367,
INDEX368,
INDEX369,
INDEX370,
INDEX371,
INDEX372,
INDEX373,
INDEX374,
INDEX375,
INDEX376,
INDEX377,
INDEX378,
INDEX379,
INDEX380,
INDEX381,
INDEX382,
5: WRITE('THE NUMBER OF RELATION IS NOT MATCHED !!');
6: WRITE('THE NUMBER OF ATTRIBUTE IS NOT MATCHED THE NUMBER OF VALUE !!');
7: WRITE('TEMPFILE,' DOES NOT EXIST, TRY AGAIN OR TYPE 'QUIT' TO EXIT !!')
8: WRITE('INVALID INPUT !!')
11: WRITE('IT HAS TO BE REAL NUMBER !!')
12: WRITE('IT HAS TO BE INTEGER !!')
END;

BEGIN (* QUIT *)
TEXTBACKGROUND(BLACK);
CLRSCR;
HALT
END; (* QUIT *)

(* CLEAR THE MESSAGE AND DATA INPUT AREA *********************************)
PROCEDURE CLEAR_MESSAGE;
VAR
ROW : INTEGER;
BEGIN (* CLEAR MESSAGE *)
FOR ROW := 20 TO 23 DO
BEGIN
GOTOXY(1, ROW);
CLREOL
END;
GOTOXY(1, 20)
END; (* CLEAR MESSAGE *)

(* CLEAR THE DATA AREA ***********************************************************)
PROCEDURE CLEAR_DATA(INDEX : INTEGER);
VAR
ROW : INTEGER;
BEGIN (* CLEAR DATA *)
FOR ROW := 8 TO 17 DO
BEGIN
GOTOXY(INDEX, ROW);
CLREOL
END
END; (* CLEAR DATA *)

(* SET THE PAGE FROM PAGE ONE ***********************************************************)
PROCEDURE INIT_PAGE(VAR RECENT_INDEX, PAGE_NUM : INTEGER);
BEGIN (* INITIALIZE PAGE *)
    RECENT_INDEX := PAGE_SIZE;
    PAGE_NUM := 1;
    GOTOXY(76, 1);
    WRITE(PAGE_NUM)
END; (* INITIALIZE PAGE *)

(* INCREMENT BY ONE PAGE *****************************************************)
PROCEDURE INCR_PAGE;
BEGIN (* INCREASE PAGE *)
    CLEAR_DATA(6);
    PAGE_NUM := PAGE_NUM + 1;
    GOTOXY(76, 1);
    WRITE(PAGE_NUM);
    RECENT_INDEX := RECENT_INDEX + PAGE_SIZE
END; (* INCREASE PAGE *)

(* DECREMENT BY ONE PAGE *****************************************************)
PROCEDURE DECR_PAGE;
BEGIN (* DECREASE PAGE *)
    CLEAR_DATA(6);
    PAGE_NUM := PAGE_NUM - 1;
    GOTOXY(76, 1);
    WRITE(PAGE_NUM);
    RECENT_INDEX := RECENT_INDEX - PAGE_SIZE
END; (* DECREASE PAGE *)

(* GET VALUES FROM THE INPUT STRING *****************************************************)
PROCEDURE SUBST(VAR TEMPAR : TEMPTY);

VAR
    CAL : INTEGER;
BEGIN (* SUBSTRING *)
    CAL := 0;
    TEMPAR := ' ';
    WHILE TEMP[COUNT+1] = ' ' DO
        COUNT := COUNT + 1;
    END DO
    WHILE ((TEMP[COUNT+1] <> ' ') AND (COUNT+1 <= LENGTH(TEMP))) DO
        BEGIN
            COUNT := COUNT + 1;
            CAL := CAL + 1;
            TEMPAR[CAL] := TEMP[COUNT]
        END
    END;
END; (* SUBSTRING *)

(* GET ATTRIBUTE NAMES FROM THE INPUT STRING FOR RELATION OF GLOBAL DATA BASE*)
PROCEDURE SUBST1(VAR TEMPAR : TEMPTY);

VAR
    CAL : INTEGER;
BEGIN (* SUBST1 *)

    CAL := 0;
    TEMPAR := ' '; 
    IF TEMP[COUNT+1] = ' ' THEN
        COUNT := COUNT + 1;
    WHILE TEMP[COUNT+1] = ' ' DO
        COUNT := COUNT + 1;
    WHILE ((TEMP[COUNT+1] <> ')') AND (COUNT+1 <= LENGTH(TEMP))) DO 
        BEGIN
            COUNT := COUNT + 1;
            CAL := CAL + 1;
            TEMPAR[CAL] := TEMP[COUNT]
        END
    END; (* SUBST1 *)

(* MOVE THE CURSOR TO THE ORIGINAL POSITION WHENEVER INPUT DATA IS WRONG *****)
PROCEDURE AGAIN(Y : INTEGER);
BEGIN (* AGAIN *)
    GOTOXY(1, Y);
    CLEAREOL
END; (* AGAIN *)

(* HELP COMMAND TO ASSIST USERS TO UNDERSTAND HOW TO EXECUTE THE PROJECT *****)
PROCEDURE HELP_CM(FHELP : DD);

VAR
    HELP : ARRAY[1..200] OF STRING[80];
    HFUNCT : CHAR;
    LAST,
    RECENT_Help,
    HELP_NUM : INTEGER;

(* CLEAR THE HELP SCREEN *********************************************)
PROCEDURE CLEAR_HELP;

VAR
    ROW : INTEGER;
BEGIN (* CLEAR HELP *)
    FOR ROW := 1 TO 17 DO 
        BEGIN
            GOTOXY(1, ROW);
            CLEAREOL
        END
    END; (* CLEAR HELP *)

(* LOAD INFORMATION FROM THE HELP FILE ***********************************)
PROCEDURE READ_HELP;

VAR
    I : INTEGER;
    HELPFILE : TEXT;
BEGIN (* READ HELP *)
  I := 0;
  ASSIGN(HELPFILE, PDR+'FHELP');
  RESET(HELPFILE);
  WHILE NOT EOF(HELPFILE) DO
    BEGIN
      I := I + 1;
      READIN(HELPFILE, HELP[I])
    END;
  CLOSE(HELPFILE);
  LAST := I
END; (* READ HELP *)

(* INCREMENT ONE PAGE FROM THE HELP FILE *******************************************)
PROCEDURE INCR_HELP;

BEGIN (* INCREASE PAGE OF HELP *)
  CLEAR_HELP;
  HELP_NUM := HELP_NUM + 1;
  GOTOXY(71, 1);
  TEXTCOLOR(RED);
  WRITE('PAGE ');
  TEXTCOLOR(YELLOW);
  WRITE(HELP_NUM);
  RECENT_HELP := RECENT_HELP + 16
END; (* INCREASE PAGE OF HELP *)

(* DECREMENT ONE PAGE FROM THE HELP FILE *******************************************)
PROCEDURE DECR_HELP;

BEGIN (* DECREASE PAGE OF HELP *)
  CLEAR_HELP;
  HELP_NUM := HELP_NUM - 1;
  GOTOXY(71, 1);
  TEXTCOLOR(RED);
  WRITE('PAGE ');
  TEXTCOLOR(YELLOW);
  WRITE(HELP_NUM);
  RECENT_HELP := RECENT_HELP - 16
END; (* DECREASE PAGE OF HELP *)

(* SHOW HELP FILE ON THE SCREEN *******************************************)
PROCEDURE SHOW_SCREEN_HELP(INDEX : INTEGER);

VAR
  I : INTEGER;

BEGIN (* SHOW HELP SCREEN *)
  GOTOXY(1, 2);
  FOR I := RECENT_HELP - 15 TO RECENT_HELP DO
    IF I <= INDEX THEN
      WRITELN(HELP[I])

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
PROCEDURE NEXT PAGE _HELP;
BEGIN (* NEXT PAGE HELP *)
   IF RECENT _HELP < LAST THEN
      BEGIN
         INCR _HELP;
         SHOW _SCREEN _HELP (LAST)
      END
   ELSE
      ERROR (3)
   END; (* NEXT PAGE HELP *)

PROCEDURE PREV_PAGE _HELP;
BEGIN (* PREVIOUS PAGE HELP *)
   IF RECENT _HELP > 16 THEN
      BEGIN
         DECR _HELP;
         SHOW _SCREEN _HELP (LAST)
      END
   ELSE
      ERROR (2)
   END; (* PREVIOUS PAGE HELP *).

BEGIN (* HELP COMMAND *)
   READ _HELP;
   CLEAR _HELP;
   RECENT _HELP := 16;
   HELP _NUM := 1;
   GOTOXY (71, 1);
   TEXTCOLOR (RED);
   WRITE ('PAGE ');
   TEXTCOLOR (YELLOW);
   WRITE (HELP _NUM);
   GOTOXY (1, 24);
   CLEAR;
   GOTOXY (1, 25);
   CLEAR;
   TEXTCOLOR (RED);
   WRITE ('E => END HELP   N => NEXT PAGE   P => PREVIOUS PAGE   ');
   WRITE ('Q => QUIT TO SYSTEM') ;
   TEXTCOLOR (YELLOW);
   SHOW _SCREEN _HELP (LAST);
   HFUNCT := ' ';
   GOTOXY (1, 19);
   WRITE ('FUNCTION : ');
   WHILE (HFUNCT <> 'E') AND (HFUNCT <> 'e') DO
      BEGIN
      END;
GOTOXY(15, 19);
CLEAR;
READIN(HFUNCT);
CLEAR MESSAGE;
IF (HFUNCT <> 'E') AND (HFUNCT <> 'e') THEN
  CASE HFUNCT OF
  'N', 'n' : NEXT_PAGE_HELP;
  'P', 'p' : PREV_PAGE_HELP;
  'Q', 'q' : HALT;
  ELSE     ERROR(8)
  END
END;
(* HELP COMMAND *)

(* SHOW ONE PAGE OF RELATION DATA TO THE SCREEN ******************************************************)
PROCEDURE SHOW_SCREEN_REL(INDEX : INTEGER);

VAR
  I : INTEGER;
BEGIN (* SHOW SCREEN RELATION *)
  GOTOXY(1, 7);
  FOR I := RECENT_INDEX - 9 TO RECENT_INDEX DO
    IF I <= INDEX THEN
      BEGIN
        GOTOXY(9, WHEREY+1);
        WRITE(I:2, '\', NUM_REC[I]:3);
        WRITE(' ', NUM_BYTE[I]:3)
      END
    END; (* SHOW SCREEN RELATION *)

(* GO TO THE NEXT PAGE DATA OF RELATION PARAMETER ******************************************************)
PROCEDURE NEXT_PAGE_REL;

BEGIN (* NEXT PAGE RELATION *)
  IF RECENT_INDEX < NUM_REL THEN
    BEGIN
      INCR_PAGE;
      SHOW_SCREEN_REL(NUM_REL)
    END
  ELSE     ERROR(3)
END; (* NEXT PAGE RELATION *)

(* GO TO THE PREVIOUS PAGE DATA OF RELATION PARAMETER ******************************************************)
PROCEDURE PREV_PAGE_REL;

BEGIN (* PREVIOUS PAGE RELATION *)
  IF RECENT_INDEX > PAGE_SIZE THEN
    BEGIN
      DECR_PAGE;
    END
SHOW_SCREEN_REL(NUM_REL)

ELSE ERROR(2)
END; (* PREVIOUS PAGE RELATION *)

(* CHECK THE FILE IF IT EXISTS OR NOT, NEW OR OLD FILE ***********************)
PROCEDURE CHECK_FILE(VAR FILENAME : DD);
BEGIN (* CHECK FILE *)
FILENAME := '';
REPEAT
WRITE('ENTER FILE NAME : ');
CLRCOL;
READLN(FILENAME);
IF (FILENAME = 'Q') OR (FILENAME = 'q') THEN
HALT
ELSE BEGIN
CLEAR_MESSAGE;
NEWOLD := '';
WRITEIN;
WHILE ((NEWOLD <> 'NEW') AND (NEWOLD <> 'OLD')
  AND (NEWOLD <> 'new') AND (NEWOLD <> 'old')) DO BEGIN
AGAIN(WHEREY-1);
WRITE('(NEW or OLD)? : ');
READLIN(NEWOLD);
IF (NEWOLD = 'Q') OR (NEWOLD = 'q') THEN
HALT
END;
ASSIGN(INFILE, INDR + ': ' + FILENAME+'.IN');
{$1-} RESET(INFILE) {$I+};
OK := (IRESULT = 0);
CLOSE(INFILE);
IF (((NEWOLD = 'OLD')OR(NEWOLD = 'old')) AND (NOT OK)) THEN BEGIN
TEMPFILE := FILENAME;
ERROR(7)
END ELSE IF ((NEWOLD = 'NEW') OR (NEWOLD = 'new')) AND (OK) THEN BEGIN
WRITEIN;
YESNO := '';
WRITEIN(FILENAME, ' ALREADY EXISTS. ');
WRITEIN;
WHILE ((YESNO <> 'YES') AND (YESNO <> 'NO')
  AND (YESNO <> 'yes') AND (YESNO <> 'no')) DO BEGIN
AGAIN(WHEREY-1);
WRITE('DO YOU WANT TO INPUT '?);
WRITE('ANOTHER FILE NAME? (YES/NO) : ');
READLN(YESNO);
IF (YESNO = 'Q') OR (YESNO = 'q') THEN
  HALT;
END;
IF (YESNO = 'YES') OR (YESNO = 'yes') THEN
  BEGIN
    AGAIN(WHHERY-5);
    OK := FALSE
  END
ELSE
  OK := TRUE
END
UNTIL OK;
CLEAR MESSAGE
END; (* CHECK FILE *)

(* SHOW THE SCREEN FORMAT OF RELATION PARAMETER ****************************)
PROCEDURE SCREEN_REL_PAR;
VAR
  COUNT : INTEGER;
BEGIN (* SCREEN_REL_PAR *)
  CLRSCR;
  TEXTCOLOR(RED);
  WRITELN('':30, 'RELATION PARAMETER', '':20, 'PAGE');
  WRITELN('':30, '        ');
  WRITELN;
  WRITELN('11 NUMBER OF RELATION : ');
  WRITELN('RELATION NUMBER OF RECORDS PER RECORD');
  FOR COUNT := 1 TO 8 DO
    WRITE('_________');
  FOR COUNT := 1 TO 10 DO
    WRITELN(COUNT:2);
  FOR COUNT := 1 TO 8 DO
    WRITE('_________');
  TEXTCOLOR(YELLOW)
END; (* SCREEN_REL_PAR *)

(* SHOW FUNCTION KEYS FOR FILE *****************)
PROCEDURE FUNCT_FILE;
BEGIN (* FILE FUNCTION *)
  GOTOXY(1, 24);
  TEXTCOLOR(RED);
  WRITE('Q => QUIT TO SYSTEM');
  TEXTCOLOR(YELLOW);
  GOTOXY(1, 19)
END; (* FILE FUNCTION *)

(* SHOW FUNCTION KEYS OF TYPE, NEW AND OLD, ON THE ROW 24 *******************)
PROCEDURE FUNCT_NEW;
BEGIN
  GOTOXY(1, 24);
  TEXTCOLOR(RED);
  WRITE('H => HELP Q => QUIT TO SYSTEM');
  TEXTCOLOR(YELLOW);
  GOTOXY(1, 19)
END;

(* SHOW FUNCTION KEYS FOR RELATION PARAMETER ******************) PROCEDURE FUNCT_RELIA;
BEGIN (* RELATION FUNCTION *)
  GOTOXY(1, 24);
  TEXTCOLOR(RED);
  WRITE('E => END MODIFY H => HELP R => REPLACE');
  WRITE('N => NEXT PAGE P => PREVIOUS PAGE Q => QUIT TO SYSTEM');
  TEXTCOLOR(YELLOW);
  GOTOXY(1, 19)
END; (* RELATION FUNCTION *)

(* SHOW FUNCTION KEYS FOR GLOBAL DATA BASE ON LAST TWO ROWS ******************) PROCEDURE FUNCT_GLOBAL;
BEGIN (* GLOBAL FUNCTION *)
  GOTOXY(1, 24);
  TEXTCOLOR(RED);
  WRITE('E => END MODIFY H => HELP R => REPLACE');
  WRITE('Q => QUIT TO SYSTEM');
  WRITE('D => DELETE I => INSERT N => NEXT PAGE');
  TEXTCOLOR(YELLOW);
  GOTOXY(1, 19)
END; (* GLOBAL FUNCTION *)

PEND;

(* FILE FUNCTION *)

PROCEDURE SUBST2(VAR ABC : EE);
BEGIN
  ABC := '';
  WHILE TEMP[COUNT+1] = ' ' DO
    COUNT := COUNT + 1;
  WHILE (TEMP[COUNT+1] <> ' ') AND (COUNT+1 <= LENGTH(TEMP)) DO
    BEGIN
      COUNT := COUNT + 1;
    END;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
ABC := ABC + TEMP[COUNT]
END
END;

(* INPUT NEW NUMBER OF RELATION, NUMBER OF RECORD FOR EACH RELATION AND
NUMBER OF BYTE FOR EACH RECORD OF RELATION PARAMETER ***********************)

PROCEDURE NEW_REL_PAR;
BEGIN (* NEW RELATION PARAMETER *)
  REPEAT
    GOTOXY(1, 19);
    WRITE('NUMBER OF RELATIONS : '); CLREOL;
    READIN(ITEMNO);
    VAL(ITEMNO, NUM_REL, RESULT);
    IF (ITEMNO = 'Q') OR (ITEMNO = 'q') THEN HALIT
    ELSE IF (ITEMNO = 'H') OR (ITEMNO = 'h') THEN
      BEGIN
        HELP_OCM('HELP1.HLP'); SCREEN_REL_PAR;
        GOTOXY(76, 1);
        WRITE(PAGELN, NUMJREL);
        FUNCT_NEW
      END
    ELSE IF RESULT <> 0 THEN ERROR(8)
    UNTIL RESULT = 0;
    CLEAR_MESSAGE;
    GOTOXY(28, 4);
    WRITE(NUM_REL);
    FOR INDEX := 1 TO NUM_REL DO
      BEGIN
        REPEAT
          INDEX1 := 0;
          GOTOXY(1, 19);
          WRITE('RELATION ', INDEX, ' : '); CLREOL;
          READIN(TEMP);
          IF (TEMP = 'Q') OR (TEMP = 'q') THEN HALIT
          ELSE IF (TEMP = 'H') OR (TEMP = 'h') THEN
            BEGIN
              HELP_OCM('HELP1.HLP'); SCREEN_REL_PAR;
              GOTOXY(76, 1);
            END
          ELSE
            BEGIN
              HELP_OCM('HELP1.HLP'); SCREEN_REL_PAR;
              GOTOXY(76, 1);
            END
        END
      END
END.
WRITE(PAGE_NUM);
GOTOXY(28, 4);
WRITE(NUM_REL);
SHOW_SCREEN_REL(INDEX - 1);
FUNCT_NEW
END
ELSE
BEGIN
COUNT := 0;
WHILE (COUNT <= LENGTH(TEMP)) DO
BEGIN
INDEX1 := INDEX1 + 1;
SUBST2(STRING10[INDEX1]);
END;
IF (INDEX1 = 2) THEN
BEGIN
VAL(STRING10[1], NUM_REC[INDEX1], RESULT);
VAL(STRING10[2], NUM_BYTE[INDEX1], RESULT1);
IF (RESULT = 0) OR (RESULT1 = 0) THEN
ERROR(8)
END
ELSE
ERROR(6)
END
UNTIL (INDEX1 = 2) AND (RESULT = 0) AND (RESULT1 = 0);
CLEAR_MESSAGE;
IF ((INDEX MOD 10 = 1) AND (INDEX > 1)) THEN
INCR_PAGE;
GOTOXY(9, (INDEX-1) MOD 10 + 8);
WRITE(INDEX:2, ', NUM_REC[INDEX]:3);
WRITEIN( ', NUM_BYTE[INDEX]:3)
END
END; (* NEW RELATION PARAMETER *)

(* REPLACE THE DATA OF RELATION PARAMETER ***********************************)
PROCEDURE REPLACE_REL_PAR;
VAR
INDEX : INTEGER;
TEMPINT : INTEGER;
BEGIN (* REPLACE RELATION PARAMETER *)
WRITE('ITEM NO. : '); ITEMNO := '';
READLN(ITEMNO);
VAL(ITEMNO, ITEM, RESULT);
IF (ITEMNO = '') AND (ITEM = 11) AND (RESULT = 0) THEN
BEGIN
WRITE('NUMBER OF RELATION : ');
ITEMNO := '';
READLN(ITEMNO);
VAL(ITEMNO, TEMPINT, RESULT);
REproduced with perm ission of the copyright ow ner. Further reproduction prohibited without permission.
IF (RESULT <> 0) OR (ITEMNO = '') THEN
  ERROR(8)
ELSE
  BEGIN
    NUM_REL := TEMPINT;
    GOTOXY(28,4);
    CLREOL;
    WRITE(NUM_REL)
  END
END

ELSE
  IF (((PAGE_NUM-1) * PAGE_SIZE + ITEM <= NUM_REL) AND (ITEM > 0)
      AND (ITEM < 11) AND (ITEMNO <> '') AND (RESULT = 0)) THEN
    BEGIN
      INDEX := ((PAGE_NUM-1) * PAGE_SIZE + ITEM);
      WRITE('RELATION ', INDEX:2);
      WRITE(' (# OF RECORDS): OLD = ',NUM_REC[INDEX]:2,' NEW = ');
      CLREOL;
      TEMP := '';
      READLN(TEMP);
      VAL(TEMP, TEMPINT, RESULT);
      IF RESULT <> 0 THEN
        ERROR(8)
      ELSE
        BEGIN
          IF TEMP = '' THEN
            BEGIN
              GOTOXY(47,21);
              WRITE(NUM_REC[INDEX]:2)
            END
          ELSE
            BEGIN
              NUM_REC[INDEX] := TEMPINT;
              GOTOXY(9, (INDEX-1) MOD 10 + 8);
              WRITE(INDEX:2,' :18, NUM_REC[INDEX]:3)
            END;
            GOTOXY(13, 22);
            WRITE(' (# OF BYTES) : OLD = ',NUM_BYTE[INDEX]:2,' NEW
            TEMP := '';
            READLN(TEMP);
            VAL(TEMP, TEMPINT, RESULT);
            IF RESULT <> 0 THEN
              ERROR(8)
            ELSE
              BEGIN
                IF TEMP = '' THEN
                  BEGIN
                    GOTOXY(47,22);
                    WRITE(NUM_BYTE[INDEX]:2)
                  END
                ELSE
                  BEGIN

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
NUM_BYTE[INDEX] := TEMPINT;
GOTOXY(9, (INDEX-1) MOD 10 + 8);
CLREOL;
WRITE(INDEX:2, ':18, NUM_REC[INDEX]:
WRITE('':23, NUM_BYTE[INDEX]:3)
END
ELSE
ERROR(1)
END; (* REPLACE RELATION PARAMETER *)

(* LOAD THE DATA OF RELATION PARAMETER FROM THE DISK ************************)
PROCEDURE DISPLAY_ERCM_REL;
VAR
C1 : INTEGER;
BEGIN (* DISPLAY FROM RELATION PARAMETER *)
GOTOXY(28, 4);
WRITE(NUM_REL);
FOR C1 := 1 TO NUM_REL DO
  IF C1 <= 10 THEN
    BEGIN
      GOTOXY(9, C1+7);
      WRITE(C1:2, ':18, NUM_REC[C1]:3);
      WRITE('':23, NUM_BYTE[C1]:3)
    END
  END;
END; (* DISPLAY FROM RELATION PARAMETER *)

(* LOAD THE RELATION PARAMETER DATA FROM THE DISK **********************)
PROCEDURE COPY_FROM_REL;
VAR
INDEX : INTEGER;
BEGIN (* COPY FROM REL *)
  ASSIGN(INFILE, INDR:FILENAME', IN');
  RESET(INFILE);
  READ(INFILE, NUM_REL);
  FOR INDEX := 1 TO NUM_REL DO
    READ(INFILE, NUM_REC[INDEX], NUM_BYTE[INDEX]);
  CLOSE(INFILE)
END; (* COPY FROM REL *)

(* STORE THE RELATION PARAMETERS TO THE DISK *******************************
PROCEDURE COPY_TO_REL;
VAR
INDEX : INTEGER;

BEGIN (* COPY TO RELATION PARAMETER FILE *)
ASSIGN(INFILE, INDRH:'FILENAME.IN');
REWRI TE(INFILE);
WRITE(INFILE, NUM_REL, ' ');
FOR INDEX := 1 TO NUM_REL DO
  WRITE(INFILE, NUM_REC[INDEX], ' ', NUM_BYTE[INDEX], ' ');
CLOSE(INFILE)
END; (* COPY TO RELATION PARAMETER FILE *)

(* MODIFY OLD DATA OF RELATION PARAMETER ******************************)
PROCEDURE OLD_REL_PAR;

BEGIN (* OLD RELATION PARAMETER *)
  FOREACH RELIA;
  FUNCT := ' '
  WHILE (FUNCT <> 'E') AND (FUNCT <> 'e') DO
    BEGIN
      GOTOXY(1, 19);
      WRITE('FUNCTION : ');
      CLEAR;
      READIN(FUNCT);
      CLEAR_MESSAGE;
      IF (FUNCT <> 'E') AND (FUNCT <> 'e') THEN
        CASE FUNCT OF
          'H', 'h' : HELP_OCM('HELP1.HLP');
          'N', 'n' : NEXT_PAGE_REL;
          'P', 'p' : PREV_PAGE_REL;
          'R', 'r' : REPLACE_REL_PAR;
          'Q', 'q' : HALT;
          ELSE ERROR(8)
        END;
      END;
    END;
  END;
END; (* OLD RELATION PARAMETER *)

(* SHOW SCREEN FORMAT OF THE GLOBAL DATA BASE ******************************)
PROCEDURE SCREEN_GLOBAL;

VAR COUNT : INTEGER;
BEGIN (* SCREEN_GLOBAL *)
  clrscr;
  textcolor(red);
  writeln(' *:30, 'GLOBAL RELATION', INDEX, ':20, 'PAGE ');
  writeln(' *:30, '-------');
  writeln;
  writeln('11 RELATION NAME : ');
  writeln;
  writeln('12');
  for count := 1 to 8 do
    write('_________');
  for count := 1 to 10 do
    writeln(count; 2 );
  for count := 1 to 8 do
    write('_________');
  textcolor(yellow)
END; (* SCREEN_GLOBAL *)

(* LOAD DATA OF THE GLOBAL DATA BASE FROM THE DISK **************************)
PROCEDURE COPY_FROM_GLOBAL(VAR FILENAME : DD);

BEGIN (* COPY FROM GLOBAL *)
  assign(infile, indr'; '+FILENAME'; '.IN');
  reset(infile);
  readin(infile, re_name);
  gotoxy(29, 4);
  write(re_name);
  index1 := 0;
  gotoxy(6, 6);
  while not eof(infile) do
    begin
      index1 := index1 + 1;
      read(infile, field_name[index1]);
      write(field_name[index1])
    end;
  readin(infile);
  index3 := 0;
  while not eof(infile) do
    begin
      index3 := index3 + 1;
      gotoxy(6, index3+7);
      col := 0;
      while not eof(infile) do
        begin
          col := col + 1;
          read(infile, emp[index3, col]);
          if index3 <= 10 then
            write(emp[index3, col])
        end;
      readin(infile);
    end;
  close(infile)
END; (* COPY FROM GLOBAL *)

(* STORE DATA OF THE GLOBAL DATA BASE TO THE DISK ***********************)
PROCEDURE COPY_TO_GLOBAL(FILENAME : DD);

BEGIN (* COPY TO GLOBAL *)
ASSIGN(INFILE, INDR+'FILENAME'+IN);
REWRITE(INFILE);
WRITELN(INFILE, NAME);
FOR ROW := 1 TO INDEX1 DO
  WRITE(INFILE, FIELD_NAME[ROW]);
WRITELN(INFILE);
FOR ROW := 1 TO INDEX3 DO
  BEGIN
    FOR COL := 1 TO INDEX1 DO
      WRITE(INFILE, EMP[ROW, COL]);
    WRITELN(INFILE)
  END;
CLOSE(INFILE)
END; (* COPY TO GLOBAL *)

(* DISPLAY DATA OF THE GLOBAL DATA BASE ON THE SCREEN ***********************)
PROCEDURE SHOW_SCREEN(INDEX : INTEGER);

VAR
I,
J : INTEGER;
BEGIN (* SHOW SCREEN *)
GOIOXY(1, 7);
FOR I := RECENT_INDEX - 9 TO RECENT_INDEX DO
  IF I <= INDEX THEN
    BEGIN
      GOIOXY(6, WHEREY+1);
      FOR J := 1 TO INDEX1 DO
        WRITE(EMP[I, J])
      END
    END
END; (* SHOW SCREEN *)

(* GET VALUE FROM THE INPUT STRING, AND STORE IT TO THE APPROPRIATE FIELD ***)
PROCEDURE GET_SUBST(INDEX : INTEGER);

VAR
FLAG : BOOLEAN;
I,
FIELD_COUNT : INTEGER;
TEMPA : ARRAY[1..10] OF TEMPTY;
BEGIN (* GET SUBSTRING *)
WRITE('VALUE :
READIN(TEMP);
COUNT := 0;
FIELD_COUNT := 0;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
WHILE ((COUNT+1) <= LENGTH(TEMP)) DO
BEGIN
  FIELD_COUNT := FIELD_COUNT + 1;
  SUBST(TEMPA[FIELD_COUNT])
END;
IF FIELD_COUNT <> INDEX THEN
BEGIN
  ERROR_FLAG := TRUE;
  ERROR(6)
END ELSE
BEGIN
  IF ((FUNCTION = 'I') OR (FUNCTION = 'i')) AND (RECENT_INDEX < INDEX3) THEN
BEGIN
    CLEAR_DATA(6);
    PAGE_NUM := INDEX DIV PAGE_SIZE;
    IF INDEX MOD PAGE_SIZE <> 0 THEN
      PAGE_NUM := PAGE_NUM + 1;
    GOTOXY(76, 1);
    WRITE(PAGE_NUM);
    RECENT_INDEX := PAGE_NUM * 10;
    SHOW_SCREEN(INDEX-1)
END;
    GOTOXY(6, (INDEX-1) MOD 10 + 8);
FOR COUNT := 1 TO FIELD_COUNT DO
BEGIN
  EMPCINDEX, COUNT] := TEMPA[COUNT];
  WRITE(TE MPA[COUNT])
END
END; (* GET SUBSTRING *)

(* INPUT THE RELATION NAME FOR EACH RELATION OF THE GLOBAL DATA BASE *******)
PROCEDURE INPUT_REL_NAME(WINDEX : INTEGER);
BEGIN (* INPUT RELATION NAME *)
REPEAT
GOTOXY(1, 19);
CLREOL;
WRITE('RELATION NAME : ');
READIN(RE_NAME);
IF (RE_NAME = 'Q') OR (RE_NAME = 'q') THEN
  HALT
ELSE IF (RE_NAME = 'H') OR (RE_NAME = 'h') THEN
BEGIN
  HELP_ACC('HELP2.HLP');
  SCREEN_GLOBAL;
  GOTOXY(76, 1);
  WRITE(PAGE_NUM);
  IF (NEWOLD = 'OLD') OR (NEWOLD = 'old') THEN
    BEGIN

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
SHOW SCREEN(WINDEX);
GOTOXY(29, 4);
WRITE(RE_NAME);
GOTOXY(6, 6);
FOR INDEX2 := 1 TO INDEX1 DO
  WRITE(FIELD_NAME[INDEX2]);
FUNCT_GLOBAL
END
ELSE
  FUNCT_NEW
END
UNTIL (RE_NAME <> 'H') AND (RE_NAME <> 'h');
GOTOXY(29, 4);
WRITE(RE_NAME)
END; (* INPUT RELATION NAME *)

(* INPUT ATTRIBUTE NAMES OF THE RELATION ************************************)
PROCEDURE INPUT_AT_NAME(WINDEX : INTEGER);
BEGIN (* INPUT ATTRIBUTE NAME *)
  REPEAT
    GOTOXY(1, 19);
    CLEAR_EOL;
    WRITELN('ATTRIBUTE NAMES (SEPERATE WITH COMMAS)');
    CLEAR_MESSAGE;
    GOTOXY(1, 20);
    READLN(TMP);
    IF (TMP = 'Q') OR (TMP = 'q') THEN
      HALT
    ELSE
      IF (TMP = 'H') OR (TMP = 'h') THEN
        BEGIN
          HELP_COM('HELP2.HLP');
          SCREEN_GLOBAL;
          GOTOXY(76, 1);
          WRITE(PAGE_NUM);
          GOTOXY(29, 4);
          WRITE(RE_NAME);
          IF (NEWOLD = 'OLD') OR (NEWOLD = 'old') THEN
            BEGIN
              SHOW SCREEN(WINDEX);
              GOTOXY(6, 6);
              FOR INDEX2 := 1 TO INDEX1 DO
                WRITE(FIELD_NAME[INDEX2]);
            FUNCT_GLOBAL
            END
          ELSE
            FUNCT_NEW
            END
        END
      END
UNTIL (TMP <> 'H') AND (TMP <> 'h');
COUNT := 0;
INDEX1 := 0;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
WHILE (COUNT+1 <= LENGTH(TEMP)) DO
    BEGIN
        INDEX1 := INDEX1 + 1;
        SUBST1(FIELD_NAME[INDEX1])
    END;
END;
GOTOXY(6, 6);
FOR INDEX2 := 1 TO INDEX1 DO
    GOTOXY(1, 20);
    WRITE(FIELD_NAME[INDEX2]);
    CIRED;
    GOTOXY(1, 19);
    CIRED
END; /* INPUT ATTRIBUTE NAME */

(* GO TO THE NEXT PAGE DATA OF THE RELATION *****************************)
BEGIN (* NEXT PAGE *)
    IF RECENT_INDEX < INDEX3 THEN
        BEGIN
            INCR_PAGE;
            SHOW_SCREEN(INDEX3)
        END
    ELSE
        ERROR(3)
    END; /* NEXT PAGE */
END; /* NEXT PAGE */

(* GO TO THE PREVIOUS PAGE DATA OF THE RELATION *****************************)
BEGIN (* PREVIOUS PAGE *)
    IF RECENT_INDEX > PAGE_SIZE THEN
        BEGIN
            DECR_PAGE;
            SHOW_SCREEN(INDEX3)
        END
    ELSE
        ERROR(2)
    END; /* PREVIOUS PAGE */
END; /* PREVIOUS PAGE */

(* INSERT THE NEW TUPLE DATA OF THE RELATION *****************************)
BEGIN (* INSERT *)
    ERROR_FLAG := FALSE;
    INDEX3 := INDEX3 + 1;
    GET_SUBST(INDEX3);
    IF ERROR_FLAG THEN
        INDEX3 := INDEX3 - 1
    END; /* INSERT */

(* SHOW THE OLD VALUE OF THE RELATION THAT YOU NEED TO REPLACE*)
ON THE DATA INPUT AREA AND READ THE NEW VALUE ****************************

PROCEDURE TMODIFY(I : INTEGER);

VAR
  TEMP : TEMPTY;
BEGIN
  GOTOXY(1, 21);
  WRITE(FIELD_NAME[I], ': OLD = ', EMP[(PAGE_NUM-1)*10+ITEM,I], ' NEW = '); CLEAR;
  TEMP := ' ';
  READ(TMP);
  IF TEMP <> ' ' THEN
    BEGIN
      EMP[(PAGE_NUM-1)*PAGE_SIZE+ITEM, I] := TEMP;
      GOTOXY((I-1) * 12 + 6, ITEM + 7);
      WRITE(EMP[(PAGE_NUM-1)*PAGE_SIZE+ITEM, I])
    END;
END;

(* REPLACE DATA OF THE RELATION FOR THE GLOBAL DATABASE ****************************)

PROCEDURE REPLACE;

VAR
  I : INTEGER;
BEGIN (* REPLACE *)
  WRITE('ITEM NO. : '); ITEM NO;
  READLN(ITEM NO);
  VAL(ITEM NO, ITEM, RESULT);
  IF (ITEM NO <> '') AND (ITEM > 0) AND (ITEM < 13) AND (RESULT = 0) THEN
    CASE ITEM OF
      11 : INPUT_RE_NAME(INDEX3);
      12 : INPUT_AT_NAME(INDEX3);
      ELSE IF (((PAGE_NUM-1) * PAGE_SIZE + ITEM) <= INDEX3) THEN
        FOR I := 1 TO INDEX1 DO
          TMODIFY(I)
      ELSE ERROR(1)
      END;
    ELSE ERROR(1)
    END;
END (* REPLACE *)

(* DELETE THE TUPLE OF DATA FOR THE RELATION THAT YOU DON'T WANT IT **********)

PROCEDURE DELETE;

VAR
  FIELD_COUNT : INTEGER;
BEGIN (* DELETE *)
INDEX3 := INDEX3 - 1;
IF ((PAGE_NUM-1) * PAGE_SIZE + ITEM) = INDEX3 + 1 THEN
    BEGIN
        GOTOXY(6, ITEM+7);
        CIREOL
    END
ELSE
    BEGIN
        GOTOXY(6, ITEM + 6);
        FOR COUNT := ((PAGE_NUM-1) * PAGE_SIZE + ITEM) TO INDEX3 DO
            BEGIN
                GOTOXY(6, WHEREY + 1);
                FOR FIELD_COUNT := 1 TO INDEX1 DO
                    BEGIN
                        EMP[COUNT, FIELD_COUNT] := EMP[COUNT+1, FIELD_COUNT];
                        IF COUNT <= (PAGE_NUM * PAGE_SIZE) THEN
                            WRITE(EMP[COUNT, FIELD_COUNT])
                    END;
            END;
        IF INDEX3+1 <= (PAGE_NUM * PAGE_SIZE) THEN
            BEGIN
                GOTOXY(6, INDEX3 MOD PAGE_SIZE + 8);
                CIREOL
            END
    END
END; (* DELETE *)

(* INPUT THE NEW RELATION NAME; ATTRIBUTE NAMES AND DATA OF THE RELATION ****)
PROCEDURE NEW_GLOBAL;
BEGIN (* NEW GLOBAL *)
    INDEX1 := 0;
    INPUT_RE_NAME(0);
    INPUT_AT_NAME(0);
    INDEX3 := NUM_REC[INDEX];
    FOR TINDEX := 1 TO NUM_REC[INDEX] DO
        REPEAT
            INDEX2 := 0;
            GOTOXY(1, 19);
            WRITE('VALUE : ');
            CIREOL;
            READIN(TEMP);
            CLEAR_MESSAGE;
            IF (TEMP = 'Q') OR (TEMP = 'q') THEN
                HALIT
            ELSE
                IF (TEMP = 'H') OR (TEMP = 'h') THEN
                    BEGIN
                        HELP_COM('HELP2.HLP');
                        SCREEN_GLOBAL;
                        GOTOXY(76, 1);
                        WRITE(PAGE_NUM);
SHOW SCREEN(TINDEX - 1);
GOTOXY(29, 4);
WRITE(RE_NAME);
GOTOXY(6, 6);
FOR INDEX2 := 1 TO INDEX1 DO
    WRITE(FIELD_NAME[INDEX2]);
END
ELSE
    BEGIN
        COUNT := 0;
        WHILE ((COUNT+1) <= LENGTH(TEMP)) DO
            BEGIN
                INDEX2 := INDEX2 + 1;
                SUBST(EMP[TINDEX ,INDEX2])
            END;
            IF INDEX2 <> INDEX1 THEN
                ERROR(6)
            ELSE
                BEGIN
                    IF (TINDEX MOD 10 = 1) AND (TINDEX > 1) THEN
                        INCR_PAGE;
                    GOTOXY(6, (TINDEX-1) MOD 10 + 8);
                    FOR COUNT := 1 TO INDEX1 DO
                        WRITE(EMP[TINDEX, COUNT])
                    END
                END;
        END;
    END
UNTIL (TEMP = 'H') AND (TEMP <> 'h') AND (INDEX2 = INDEX1);
END; (* NEW GLOBAL *)

(* MODIFY OLD DATA OF RELATIONS FOR THE GLOBAL DATA BASE **************)
PROCEDURE OLDLOBAL;
BEGIN (* OLD GLOBAL *)
    FUNCTLOBAL;
    FUNCT := 'I';
    WHILE (FUNCT <> 'E') AND (FUNCT <> 'e') DO
        BEGIN
            GOTOXY(1, 19);
            WRITE('FUNCTION : ');
            CLEAR;
            READIN(FUNCT);
            CLEAR MESSAGE;
            IF (FUNCT <> 'E') AND (FUNCT <> 'e') THEN
                CASE FUNCT OF
                    'D', 'd' : BEGIN
                        WRITE('ITEM NO. : ');
                        ITEMNO := '';
                        READIN(ITEMNO);
                        VAL(ITEMNO, ITEM, RESULT);
                        IF (ITEMNO<>'' AND ITEM>0 AND ITEM<11) AND (RESULT=0)
                            AND ((PAGE_NUM-1)*PAGE_SIZE+ITEM <= INDEX3) THEN
DELETE
ELSE
ERROR(1)
END;
'H','h' : BEGIN
HELPCOMM('HELP2.HELP');
SCREENGLOBAL;
GOTOXY(76,1);
WRITE(PAGENUM);
SHOWSCREEN(INDEX);
GOTOXY(29,4);
WRITE(RE_NAME);
GOTOXY(6,6);
FOR INDEX2 := 1 TO INDEX1 DO
  WRITE(FIELD_NAME[INDEX2]);
  FUNCTGLOBAL
END;
'I','i' : INSERT;
'N','n' : NEXT_PAGE;
'P','p' : PREV_PAGE;
'R','r' : REPLACE;
'O','q' : HALT;
ELSE
  ERROR(8)
END
END; (* OLD_GLOBAL *)

(* CHECK IF THE RELATION IS NEW OR OLD, THEN EXECUTE THE APPROPRIATE ROUTINE*)
PROCEDURE GLOBAL_RELATION;
BEGIN (* GLOBAL_RELATION *)
  FOR INDEX := 1 TO NUM_REL DO
    BEGIN
      SCREENGLOBAL;
      FUNCT_FILE;
      CHECK_FILE(FILENAMEA[INDEX]);
      INIT_PAGE(RECENT_INDEX, PAGENUM);
      IF (NEWOLD = 'NEW') OR (NEWOLD = 'new') THEN
        BEGIN
          FUNCTNEW;
          NEWGLOBAL
        END
      ELSE
        IF (NEWOLD = 'OLD') OR (NEWOLD = 'old') THEN
          COPY_FROM_GLOBAL(FILENAMEA[INDEX])
        ELSE
          HALT;
          OLDGLOBAL;
          NUMREC[INDEX] := INDEX3;
          COPY_TO_GLOBAL(FILENAMEA[INDEX])
        END
    END;
END; (* GLOBAL_RELATION *)
(* PUT ALL FILES OF RELATION INTO ONE FILE *****************************************)

PROCEDURE COMBINE;

VAR STR1 : STRING[80];

BEGIN
  WHILE NOT EOF(INFILE1) DO
    BEGIN
      READLN(INFILE1, STR1);
      WRITELN(DATA2, STR1)
    END
  END;

BEGIN (* MAIN *)
  WINDOW(1,1,80,25);
  TEXTBACKGROUND(BLACK);
  CLRSCR;
  SCREEN REL_PAR;
  CHECK_FILE(FIENAME);
  CLEAR_MESSAGE;
  IF (NEWOLD = 'NEW') OR (NEWOLD = 'new') THEN BEGIN
    FUNCT NEW;
    INIT PAGE(RECENT_INDEX, PAGE_NUM);
    NEW_REL_PAR;
    OLD_REL_PAR
  END
  ELSE IF (NEWOLD = 'OLD') OR (NEWOLD = 'old') THEN BEGIN
    COPY FROM REL;
    DISPLAY FROM REL;
    INIT PAGE(RECENT_INDEX, PAGE_NUM);
    OLD_REL_PAR
  END
  ELSE HALT;
  COPY TO REL;
  GLOBAL RELATION;
  ASSIGN(DATA2, PDR+':TRAN.DAT');
  REWRITE(DATA2);
  ASSIGN(INFILE1, INDR+':'+FILENAME+'.IN');
  RESET(INFILE1);
  COMBINE;
  CLOSE(DATA2);
  CLOSE(INFILE1);
  ASSIGN(DATA2, PDR+':REL.DAT');
  REWRITE(DATA2);
  FOR SINDEX := 1 TO NUM_REL DO
    BEGIN
      ASSIGN(INFILE1, INDR+':'+FILENAMEA[SINDEX]+'.IN');
RESET(INFILE1);
COMBINE;
WRITEIN(DATA2);
CLOSE(INFILE1)
END;
CLOSE(DATA2);
ASSIGN(SS, PDR+':MAINMENU.COM');
EXECUTE(SS)
END.

PROGRAM TRANSACT;

************!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!**********
(*  
(*  A Generalized Data Distribution Model for Distributed  
(*  Relational Databases  
(*  
(*  Patrick D. Yurk, M.S.  
(*  Western Michigan University  
(*  AUGUST 1989 
(*  
(*  Program to input transaction and global relation information.  
(*  
(*  THIS PART OF PROGRAM IS A HUMAN INTERFACE PROVIDES FOR AN INTERACTIVE  
(*  MODE WHICH PROVIDES FOR AN EASY AND CONVENIENT WAY TO ENTER NEW DATA OR  
(*  

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(* IMPLEMENT THE EXISTING DATA. AS THE SYSTEM IS ACTIVATED, INPUT PARAMETERS*)
(* ARE REQUESTED. TRANSACTION PARAMETERS INCLUDE A LIST OF TRANSACTIONS. FOR*)
(* EACH TRANSACTION THE FOLLOWING IS INCLUDED: THE HOME SITE OF THE TRANSACTION, THE DATA REQUIRED *)
(* FOR THE TRANSACTION (GIVEN IN TERMS OF RELATIONAL EXPRESSIONS ON THE *)
(* GLOBAL RELATIONS) AND WORKLOAD CHARACTERISTICS. *)

**TYPE**  
STRTYPE = STRING[80];  
STRTYPE = STRING[120];  
STRARYTYPE = ARRAY [1..5] OF STRTYPE;  
FILETYPE = ARRAY [1..50] OF STRTYPE;  
FILEARYTYPE = ARRAY [1..50] OF STRTYPE;  
SITEARY = ARRAY [1..50] OF INTEGER;  
TRPTYPE = ARRAY [1..300] OF STRTYPE;

**VAR**

(* COMMON STORAGE AREA USED FOR VARIABLES THAT ARE PASSED BETWEEN MODULES *)

PDR, (* PROGRAM DRIVE *)  
INDR, (* INPUT FILE DRIVE *)  
OUTDR : STRING[1]; (* OUTPUT FILE DRIVE *)  

(* END OF COMMON STORAGE AREA *)

SITE : SITEARY;

DONE : BOOLEAN;
T.C : INTEGER;
DRY : TEXT;
JJ : FILE;
FILE1, FILE2, DRIV : ST1RTYPE;

**PROCEDURE** ERROR(COD : CHAR; RW : INTEGER);

BEGIN  
GOTOXY(6,RW);  
TEXTCOLOR(15+BLink);  
CASE COD OF  
'1' : WRITE('!! FILE EXIST or ILLEGAL FILE NAME !! (QUIT or ENTER again) ');  
'2' : WRITE('!! NOT FOUND or ILLEGAL FILE NAME !! (QUIT or ENTER again) ');  
'3' : WRITE('!! INVALID DATA ITEM !!');  
'4' : WRITE('!! INVALID NUMBER OF DATA !!');  
'5' : WRITE('!! CAN NOT BE INSERTED !!');  
'6' : WRITE('!! END OF DATA FILE !!');  
'7' : WRITE('!! NO PREVIOUS PAGE !!');  
END;
IF COD IN ["3'..'7"] THEN
BEGIN
  DELAY(1500);
  GOTOXY(1,23);
  CIREOL
END;
TEXTCOLOR(14)
END;

(*-----------------------------------------------------------------------*)
(* PROCEDURE STCHECK *)
(* THIS PROCEDURE IS TO CHECK INPUT OF THE INTERACTIVE MODE. *)
(*-----------------------------------------------------------------------*)

PROCEDURE STCHECK(VAR STR1 : STRTYPE; STR2, STR3, STR4, STR5 : STRTYPE;
                    LTH : INTEGER);
BEGIN
  WHILE NOT ((STR1 = STR2) OR (STR1 = STR3) OR (STR1 = STR4) OR
              (STR1 = STR5)) DO
    BEGIN
      GOTOXY(LTH,WHEREY-1);
      CIREOL;
      READLN(STR1);
      IF (COPY(STR1,1,1) = 'Q') OR (COPY(STR1,1,1) = 'q') THEN
        BEGIN
          TEXTBACKGROUND(BLACK);
          CLRSCR;
          HALT
        END
    END;
END;

(*-----------------------------------------------------------------------*)
(* PROCEDURE STCHECKL *)
(* THIS PROCEDURE IS TO CHECK THE INPUT OF THE FUNCTION KEY. *)
(*-----------------------------------------------------------------------*)

PROCEDURE STCHECKL(VAR F_N : STRTYPE);
BEGIN
  WHILE NOT ((F_N='I') OR (F_N='D') OR (F_N='R') OR (F_N='P') OR (F_N='N')
            OR (F_N='H') OR (F_N='e') OR (F_N='q') OR (F_N='i') OR (F_N='d')
            OR (F_N='r') OR (F_N='p') OR (F_N='n') OR (F_N='h')
            OR (F_N='e') OR (F_N='q')) DO
    BEGIN
      GOTOXY(24,22);
      CIREOL;
      READLN(F_N)
    END;
END;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
PROCEDURE INTCHECK(INTEGER T_N: var TR_N; INTEGER; MESTR: STRTYPE; VAR D: BOOLEAN); 

VAR TR_NS : STRTYPE;
CC, I : INTEGER;
DONE : BOOLEAN;
TRCH : CHAR;

BEGIN
  GOTOXY(1,IC);
  WRITE(MESTR);
  DONE := FALSE;
  WHILE NOT DONE DO
    BEGIN
      READIN(TR_NS);
      IF (TR_NS = 'E') OR (TR_NS = 'e') THEN
        BEGIN
          D := TRUE;
          DONE := TRUE
        END
      ELSE BEGIN
        DONE := TRUE;
        FOR I:=1 TO LENGTH(TR_NS) DO
          BEGIN
            TRCH := COPY(TR_NS,1,1);
            IF NOT (TRCH IN ['0'..'9']) THEN
              DONE := FALSE
          END;
      IF DONE THEN
        BEGIN
          VAL(TR_NS,TR_N,CC);
          IF (TR_N > T_N) OR (TR_N < 1) THEN
            DONE := FALSE
        END;
      IF NOT DONE THEN
        BEGIN
          GOTOXY(1+LENGTH(MESTR),IC);
          CIREOL
        END
    END
  END
END;

PROCEDURE CHARCHECK(*
  (* THIS PROCEDURE IS TO CHECK IF THE INPUT CHARACTER IS CORRECT OR NOT. *)
  (***************************************************************************************)
  (*)
  (* THIS PROCEDURE IS TO CHECK IF THE INPUT CHARACTER IS CORRECT OR NOT. *)
  (***************************************************************************************)

PROCEDURE INTCHECK(INTEGER T_N: var TR_N; INTEGER; MESTR: STRTYPE; VAR D: BOOLEAN); 

VAR TR_NS : STRTYPE;
CC, I : INTEGER;
DONE : BOOLEAN;
TRCH : CHAR;

BEGIN
  GOTOXY(1,IC);
  WRITE(MESTR);
  DONE := FALSE;
  WHILE NOT DONE DO
    BEGIN
      READIN(TR_NS);
      IF (TR_NS = 'E') OR (TR_NS = 'e') THEN
        BEGIN
          D := TRUE;
          DONE := TRUE
        END
      ELSE BEGIN
        DONE := TRUE;
        FOR I:=1 TO LENGTH(TR_NS) DO
          BEGIN
            TRCH := COPY(TR_NS,1,1);
            IF NOT (TRCH IN ['0'..'9']) THEN
              DONE := FALSE
          END;
      IF DONE THEN
        BEGIN
          VAL(TR_NS,TR_N,CC);
          IF (TR_N > T_N) OR (TR_N < 1) THEN
            DONE := FALSE
        END;
      IF NOT DONE THEN
        BEGIN
          GOTOXY(1+LENGTH(MESTR),IC);
          CIREOL
        END
    END
  END
END;
PROCEDURE CHARECHECK(VAR TDATA : STRTYPE; RW : INTEGER);

VAR I : INTEGER;
DONE : BOOLEAN;
TCH : CHAR;

BEGIN
DONE := FALSE;
WHILE NOT DONE DO
BEGIN
DONE := TRUE;
FOR I:=1 TO LENGTH(TDATA) DO
BEGIN
  TCH := COPY(TDATA, I, 1);
  IF NOT (TCH IN ['A'..'Z', 'a'..'z', '0'..'9']) THEN
    DONE := FALSE
END;
IF NOT DONE THEN
BEGIN
  GOTOXY(1, RW);
  CLREOL;
  READIN(TDATA)
END
END

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(* PROCEDURE CLRTABLE *)
(* TO CLEAR THE DISPLAYED TABLE AND PREPARE TO ANOTHER PAGE. *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

PROCEDURE CLRTABLE(IND1, IND2 : INTEGER);

VAR I : INTEGER;

BEGIN
  FOR I:=1 TO IND2 DO
  BEGIN
    GOTOXY(1, I+IND1);
    CLREOL
  END
END;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
(* PROCEDURE CHELP *)
(* TO CHECK IF THE USER NEED HELP OR NOT. *)
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

PROCEDURE CHELP(VAR Y_N : STRTYPE);
BEGIN
WRITE(' DO YOU NEED HELP? (YES or NO) ');
READLN(Y_N);
IF (COPY(Y_N,1,1) = 'Q') OR (COPY(Y_N,1,1) = 'q') THEN
BEGIN
TEXBACKGROUND(BLACK);
CLRSR;
HALT
END;
STRCHECK(Y_N,'YES','NO','yes','no',37);
WRITELN;
CLRSR
END;

(******************************************************************************)
(* PROCEDURE SCR1 *)
(* TO BUILD THE SCREEN FORMAT FOR WORKLOAD/RELIABILITY REQUIREMENTS. *)
(******************************************************************************)

PROCEDURE SCR1;

BEGIN
TEXTEXTCOLOR(BLUE);
WRITE(' :5,'HOM'E', ' :5,'TRANSACTION FREQUENCY OF FREQUENCY OF ');
WRITE(' :8,'SITE', ' :7,'NUMBER', ' :7,'RETRIEVAL', ' :7,'UPDATE', ' :7);
WRITE('__________________________');
WRITE('');
GOTOXY(1,WHEREY+14);
WRITE('__________________________');
WRITE('');
TEXTCOLOR(YELLOW)
END;

(******************************************************************************)
(* PROCEDURE SCR2 *)
(* TO BUILD THE SCREEN FORMAT FOR THE TRANSACTION INPUT. *)
(******************************************************************************)

PROCEDURE SCR2;

VAR I : INTEGER;

BEGIN
TEXTEXTCOLOR(BLUE);
GOTOXY(1,2);
WRITE('__________________________________________');
WRITE('');
WRITE('__________________________');
WRITE('');
WRITE(' TRANSACTION '); TEXTCOLOR(15);
WRITE('');
WRITE(''); TEXTCOLOR(BLUE);

WRITE(' SITE ');
TEXTOOIOR(15);
WRITEIN('l');
TEXTOOIOR(BLUE);
WRITE('____________________________________________________________');
WRITEEN('___________________');
TEXTOOIOR(YELLOW)
END;

(*FUNCTION BLS*)
(* TO FLEXIBLE THE BLANK BETWEEN THE DATA OF THE WORKLOAD/RELIABILITY TABLE *)

FUNCTION BLS(NO : INTEGER) : STR1TYPE;
VAR BL : STR1TYPE;
I : INTEGER;
BEGIN
BL := ' ';
FOR I:=l TO NO DO
  BL := BL + ' ';
BLS := BL
END;

(*PROCEDURE WFILE*)
(* THIS PROCEDURE IS TO WRITE THE INPUT TO THE FILE AFTER FORMATED. *)

PROCEDURE WFILE(VAR FILEARY:FILEARYTYPE;Rw:INTEGER;S1,S2,S3,S4:STR1TYPE);
BEGIN
FILEARY[Rw] := BLS(8-LENGTH(S1))+S1+BLS(13-LENGTH(S2))+S2+BLS(14-LENGTH(S3))
  +S3+BLS(15-LENGTH(S4))+S4
END;

(*PROCEDURE HELP*)
(* THIS PROCEDURE IS TO BUILD THE HELP SCREEN AND THEN READ THE HELP3.HLP AND HELP4.HLP FILES TO THE SCREEN ACCORDING TO USER NEED.*)

PROCEDURE HELP(FILE_NAME : STR1TYPE);
TYPE HSTYPE = ARRAY [1..200] OF STR1TYPE;
VAR FNN : STRTYPE;
HDONE : BOOLEAN;
HSTR : HSTRTYPE;
I, TII, HPAGE : INTEGER;
THF : TEXT;

(*********************** PROCEDURE W_HSC ***********************)

PROCEDURE W_HSC(HP : INTEGER);

VAR J : INTEGER;
BEGIN (* W_HSC *)
  FOR J := 1 TO 23 DO
    BEGIN
      GOTOXY(1, J);
      CLRSEL
    END;
    TEXTCOLOR(WHITE);
    GOTOXY(1, 1);
    FOR J := 23 * HP - 22 TO 23 * HP DO
      WRITELN(HSTR[J]);
    TEXTCOLOR(YELLOW)
  END; (* W_HSC *)
BEGIN (* HELP *)
  ASSIGN(THF, PDR+";'+FILE_NAME);
  RESET(THF);
  CLRSCR;
  FOR I := 1 TO 200 DO
    HSTR[I] := ";
  I := 0;
  WHILE NOT EOF(THF) DO
    BEGIN
      I := I + 1;
      READIN(THF, HSTR[I])
    END;
  CLOSE(THF);
  HPAGE := 1;
  TII := (I DIV 23) + 1;
  W_HSC(1);
  WRITELN(' ..... SELECT FUNCTION : ');
  WRITE(' N : NEXT PAGE  P : PREVIOUS PAGE  E : END HELP  Q : QUIT TO SYSTEM')
  GOTOXY(30, 24);
  HDONE := FALSE;
  WHILE NOT HDONE DO
    BEGIN
      READIN(F_NN);
      WHILE NOT ((F_NN = 'N') OR (F_NN = 'P') OR (F_NN = 'Q') OR (F_NN = 'E') OR
                  (F_NN = 'n') OR (F_NN = 'p') OR (F_NN = 'q') OR (F_NN = 'e')) DO
        BEGIN
          * Further code...
        END
      HDONE := TRUE;
    END
END.
GOTOXY(30,24);
CLRBL;
READIN(F_NN)
END;

CASE F_NN OF 'n' : F_NN := 'N';
  'p' : F_NN := 'P';
  'q' : F_NN := 'Q';
  'e' : F_NN := 'E'
END;

CASE F_NN OF 'N' : IF HPAGE = TII THEN
  BEGIN
    GOTOXY(30,24);
    TEXTCOLOR(WHITE+BLINK);
    WRITE('END OF HELP');
    DELAY(2500);
    TEXTCOLOR(YELLOW)
    END
  ELSE BEGIN
    HPAGE := HPAGE + 1;
    W_HSC(HPAGE)
    END;
  'P' : IF HPAGE = 1 THEN
  BEGIN
    GOTOXY(30,24);
    TEXTCOLOR(WHITE+BLINK);
    WRITE('NO PREVIOUS HELP PAGE');
    DELAY(2500);
    TEXTCOLOR(YELLOW)
    END
  ELSE BEGIN
    HPAGE := HPAGE - 1;
    W_HSC(HPAGE)
    END;
  'E' : HDONE := TRUE;
  'Q' : BEGIN
    TEXTBACKGROUND(BLACK);
    CLRSCR;
    HALT;
    END
END;
GOTOXY(30,24);
CLRBL
END;

(* HELP *)

(*******************************************************************************)
(*  PROCEDURE W_SCR  *)
(*  TO WRITE THE FILE INTO THE SCREEN TABLE.  *)
(*******************************************************************************)

PROCEDURE W_SCR(FILEARY : FILEARYTYPE;WI : INTEGER);
VAR I : INTEGER;

BEGIN
    TEXTCOLOR(WHITE);
    GOTOXY(1,6);
    FOR I:=WI-11 TO WI DO
        WRITELN(FILEARY[I]);
    TEXTCOLOR(YELLOW)
END;

(*---------------------------------------------*

(* PROCEDURE CHECKERROR *)
(* TO CHECK THE INPUT DAT FOR THE WORKLOAD/RELIABILITY REQUIREMENTS. *)
(*---------------------------------------------*)

PROCEDURE CHECKERROR(VAR DATASTR : STRTYPE;VAR STRARY :STRARYTYPE;TT:INTEGER);

TYPE TSTRTYPE = ARRAY [1..20] OF STRTYPE;

VAR CDONE, DONE1, DONE2 : BOOLEAN;
    TSTR : TSTRTYPE;
    TSR : STRARYTYPE;
    I, BL_CT, TCT : INTEGER;
    TCH : CHAR;

BEGIN
    CDONE := FALSE;
    WHILE NOT CDONE DO
        BEGIN
            BL CT := 0;
            DONE1 := FALSE;
            FOR I:=1 TO LENGTH(DATASTR) DO
                BEGIN
                    TCH := COPY(DATASTR,I,1);
                    IF NOT (TCH IN ['0'..'9', ' ', ' ']) THEN
                        BEGIN
                            ERROR('3',23);
                            DONE1 := TRUE
                        END
                    END;
                    IF NOT DONE1 THEN
                        BEGIN
                            TSR := '';
                            TCT := 0;
                            DONE2 := TRUE;
                            FOR I:=0 TO LENGTH(DATASTR)-1 DO
                                BEGIN
                                    IF COPY(DATASTR,I+1,1) <> '' THEN
                                        BEGIN
                                            DONE2 := FALSE;
                                            TSR := TSR + COPY(DATASTR,I+1,1)
                                        END
                                    END;
                                END;
                        END;
        END;
    END;
END.
ELSE IF NOT DONE2 THEN
BEGIN
  DONE2 := TRUE;
  TCT := TCT + 1;
  TSTR[TCT] := "";
  TSTR[TCT] := TSR;
  TSR := "";
END;

IF TSR <> "" THEN
BEGIN
  TCT := TCT + 1;
  TSTR[TCT] := TSR;
END;

IF TCT <> 4 THEN
ERROR('4',23)
ELSE BEGIN
  CDONE := TRUE;
  FOR I:=1 TO 5 DO
  BEGIN
    STRARY[I] := "";
    STRARY[I] := TSTR[I]
  END;
END;

IF NOT CDONE THEN
BEGIN
  GOTOXY(1+LENGTH(DATASTR),TT);
  CLEOL;
  READLN(DATASTR);
  IF DATASTR = '0' THEN
    CDONE := TRUE;
  GOTOXY(6,23);
  CLEOL;
  GOTOXY(1+LENGTH(DATASTR),TT)
END;
END;

(* PROCEDURE P_TS *)
PROCEDURE P_TS(TRN : INTEGER);
VAR I : INTEGER;
BEGIN
  TEXTCOLOR(15);
  GOTOXY(18,4);
WRITE(TRN:2);
I := 0;
WHILE TRN > 0 DO
BEGIN
 I := I + 1;
  TRN := TRN - SITE[I]
END;
GOTOXY(28,4);
WRITEIN(I:2);
TEXTCOLOR(14)
END;

(* PROCEDURE FILL_TRAN *)
(* TO FILL THE TRANSACTION IF THE TRANSACTION FILE IN NEW. *)
(****************************************************************************)

PROCEDURE FILL_TRAN(FILE_NAME : STRTYPE; T_C : INTEGER);
VAR I, CT, ID, ICT, SITENO : INTEGER;
 TAEL : ARRAY [1..5] OF STRTYPE;
 OPERARY : STRTYPE;
 INN : TEXT;

(********************** CKHQ ************************************************)
PROCEDURE CKHQ;

BEGIN (* CKHQ *)
WHILE (COPY(OPERARY,1,1) = 'Q') OR (COPY(OPERARY,1,1) = 'q') OR
 (COPY(OPERARY,1,1) = 'H') OR (COPY(OPERARY,1,1) = 'h') DO
END
ELSE IF (COPY(OPERARY,1,1) = 'H') OR (COPY(OPERARY,1,1) = 'h') THEN
BEGIN
HELP('HELP4.HLP');
SCR2;
P_TS(CT);
FOR I:=1 TO ICT DO
BEGIN
GOTOXY(1,5+2*I);
WRITEIN(TAEL[I])
END;
GOTOXY(1,21);
WRITEIN('DATA NEEDED ?');
GOTOXY(6,24);
TEXTCOLOR(BLUE);
WRITEIN('!! YOU MUST ENTER ',T_C:2,' TRANSACTIONS !!');
TEXTCOLOR(YELLOW);
GOTOXY(6,25);
WRITE('H : HELP  Q : QUIT TO SYSTEM');
GOTOXY(1,22);
READLN(OPERARY)
END
END; (* CKHQ * )

BEGIN (* FILL Tran *)
ASSIGN(INN,'FILE_NAME');
REWRITE(INN);
CT := 0;
CLRSCR;
SCR2;
GOTOXY(1,21);
WRITELN('DATA NEEDED ?');
GOTOXY(6,24);
TEXTCOLOR(BLUE);
WRITELN('!! YOU MUST ENTER ',T_C:2,' TRANSACTIONS !!');
TEXTCOLOR(YELLOW);
GOTOXY(6,25);
WRITE('H : HELP  Q : QUIT TO SYSTEM');
WHILE CT <> T_C DO
BEGIN
  FOR I:=1 TO 5 DO
    TAPL[I] := ';
    ICT := 0;
    CLRTABLE(6,10);
    CT := CT + 1;
    P_TS(CT);
    GOTOXY(1,22);
    READLN(OPERARY);
    CKHQ;
    CHARCHECK(OPERARY,22);
    ICT := ICT + 1;
    TAPL[ICT] := OPERARY;
    ID := 0;
    WHILE COPY(OPERARY,LENGTH(OPERARY),1) <> '.' DO
      BEGIN
        ID := ID + 1;
        WRITELN(INN,OPERARY);
        GOTOXY(1,5+ID*2);
        WRITELN(OPERARY);
        GOTOXY(1,23);
        CIROLE;
        GOTOXY(1,22);
        CIROLE;
        READLN(OPERARY);
        CKHQ;
        CHARCHECK(OPERARY,22);
        ICT := ICT + 1;
        TAPL[ICT] := OPERARY
      END
    END
  END
END;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
END;
IF COPY(OPERARY,LENGTH(OPERARY),1) = '.' THEN
BEGIN
  WRITELN(INN,COPY(OPERARY,1,LENGTH(OPERARY)-1));
  GOTOXY(1,7+2*ID);
  WRITELN(COPY(OPERARY,1,LENGTH(OPERARY)-1));
  GOTOXY(1,22);
  CRKEY;
  GOTOXY(1,23);
  CRKEY;
  DELAY(2500);
  CRTABLE(6,10);
END;
CLOSE(INN)
END; (* FILLTRAN *)

(******************************************************************************)
(* PROCEDURE FILL_TABLE *)
(* TO FILL THE WORKLOAD/RELIABILITY REQUIREMENTS IF THE FILE IS NEW. *)
(******************************************************************************)

PROCEDURE FILL_TABLE(FILE_NAME : STRTYPE; VAR T_C : INTEGER);

TYPE TEMPARITYPE = ARRAY [1..50,1..5] OF STRING[10];
VAR DONE : BOOLEAN;
    DE : CHAR;
    TEMPARY : TEMPARITYPE;
    DATAST : STRTYPE;
    TDATA : FILETYPE;
    STRARY : STRARYTYPE;
    DATA1 : TEXT;

(****************************************************************************** PROCEDURE PTEMP ******************************************************************************)

PROCEDURE PTEMP;

BEGIN (* PTEMP *)
  TEMPARY[HT,1] := STRARY[3];
  TEMPARY[HT,2] := STRARY[4];
END; (* PTEMP *)

(****************************************************************************** CKHQ ******************************************************************************)

PROCEDURE CKHQ;

BEGIN (* CKHQ *)
  WHILE (COPY(DATASTR,1,1) = 'Q') OR (COPY(DATASTR,1,1) = 'q') OR
       (COPY(DATASTR,1,1) = 'H') OR (COPY(DATASTR,1,1) = 'h') DO
IF (COPY(DATASTR,1,1) = 'Q') OR (COPY(DATASTR,1,1) = 'q') THEN
BEGIN
  TEXTBACKGROUND(BLACK);
  CLRSCR;
  HALT
END
ELSE IF (COPY(DATASTR,1,1) = 'H') OR (COPY(DATASTR,1,1) = 'h') THEN
BEGIN
  HELP('HELP3.HLP');
  SCR1;
  FOR I:=1 TO S_C DO
    BEGIN
      GOTOXY(1,5+I);
      WRITEIN(TDATA[I])
    END;
  GOTOXY(6,22);
  WRITE('ENTER DATA : ');
  GOTOXY(6,24);
  TEXTCOLOR(BLUE);
  WRITEIN('FILL THE TABLE, PRINT AN "0" AT THE END OF THIS TABLE ');
  TEXTCOLOR(YELLOW);
  GOTOXY(6,25);
  WRITE('H : HELP  Q : QUIT TO SYSTEM');
  GOTOXY(20,22);
  READLN(DATASTR)
END
END; (* CKH Q *)

BEGIN (* FILL_TABLE *)
  ASSIGN(DATA1,FDR+':'+FILE_NAME);
  REWRITE(DATA1);
  CLRSCR;
  SCR1;
  GOTOXY(6,22);
  WRITE('ENTER DATA : ');
  GOTOXY(6,24);
  TEXTCOLOR(BLUE);
  WRITEIN('FILL THE TABLE, PRINT AN "0" AT THE END OF THIS TABLE ');
  TEXTCOLOR(YELLOW);
  GOTOXY(6,25);
  WRITE('H : HELP  Q : QUIT TO SYSTEM');
  GOTOXY(20,22);
  T_C := 0;
  H_T := 0;
  H_SC := 1;
  L_C := 1;
  DONE := FALSE;
  READLN(DATASTR);
  CKHQ;
  VAL(DATASTR,VV,CC);
  DE := COPY(DATASTR,1,1);
IF (V = 0) AND (D IN ['0'..'9']) THEN
BEGIN
  STRY[1] := '0';
  DONE := TRUE
END
ELSE
  CHECKERROR(DAIA,STRAY,22);
WHILE NOT ((STRAY[1] = '0') AND (DONE)) DO
BEGIN
  IF STRAY[1] <> '0' THEN
    BEGIN
      T_C := T_C + 1;
      IF T_C = (12 * L_C) + 1 THEN
        BEGIN
          L_C := L_C + 1;
          S_C := 1;
          FOR I:=1 TO 50 DO
            TDATA[I] := dRTABIE(5,12)
        END
    END
  ELSE
    BEGIN
      IF L_C = 1 THEN
        S_C := T_C MOD (12 * L_C)
      ELSE
        S_C := T_C MOD (12 * (L_C-1));
      IF S_C = 0 THEN
        S_C := 12 ;
      END;
      GOTOXY(1, S_C+5);
      WMEL(DATASA,STRAY[1],STRAY[2],STRAY[3],STRAY[4]);
      WRITELN(DATASA[S_C])
    END;
  END;
  VAL(STRAY[1],V,CC);
  IF V = H_SC THEN
    BEGIN
      H_T := H_T + 1;
      PTEMP
    END
  ELSE BEGIN
    H_SC := H_SC + 1;
    WRITELN(DATASA,H_T);
    FOR I:=1 TO H_T DO
      BEGIN
        FOR J:=1 TO 2 DO
          WRITELN(DATASA,COPY(TEMPARY[I,J],1,LENGTH(TEMPARY[I,J]));
          (* IF TEMPARY[I,3] = '0' THEN
            WRITELN(DATASA,TEMPARY[I,3])
          ELSE *)
            WRITELN(DATASA,COPY(TEMPARY[I,3],2,LENGTH(TEMPARY[I,3])-1));
        END;
        H_T := 1;
        PTEMP
      END
    END
  END;
END
END;
IF STRARY[1] <> 'O' THEN
BEGIN
GOTOXY(20,22);
CLEPOL;
READIN(DAIASTR);
CHKQ;
VAL(DAIASTR,VV,CC);
IF VV = 0 THEN
STRARY[1] := 'O'
ELSE BEGIN
CHECKERROR(DAIASTR,STRARY,22);
VAL(DAIASTR,VV,CC);
IF VV = 0 THEN
BEGIN
STRARY[1] := 'O';
DONE := TRUE
END
END
ELSE
DONE := TRUE
END;
CLOSE(DAIA1)
END; (* FILE_TABLE *)

FUNCTION FILE_CHECK(MESSAGE : STR1TYPE;IH : INTEGR
VAR F_M : INTEGR;
VAR FILE_NAME : STR1TYPE) : BOOLEAN;

VAR Q_E, N_0 : STR1TYPE;
DONE : BOOLEAN;

FUNCTION EXIST(FILENAME : STR1TYPE) : BOOLEAN;

VAR FIL : TEXT;
BEGIN (* EXIST *)
ASSIGN(FIL,FDR+'+'FILENAME);
($I-)
RESET(FIL);
($I+)
EXIST := (IRESULT = 0);
CLOSE(FIL)
END; (* EXIST *)

(* TO SEE IF THE FILE IS EXISTED OR NOT, AND THE FILE IS NEW OR OLD. *)

FUNCTION EXIST(FILENAME : STR1TYPE) : BOOLEAN;

VAR FIL : TEXT;
BEGIN (* EXIST *)
ASSIGN(FIL,FDR+'+'FILENAME);
($I-)
RESET(FIL);
($I+)
EXIST := (IRESULT = 0);
CLOSE(FIL)
END; (* EXIST *)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
BEGIN (* FILE_CHECK *)
  DONE := FALSE;
  GOTOXY(1,24);
  WRITE(' Q : QUIT TO SYSTEM');
  GOTOXY(1,3);
  WRITEIN;
  WRITEIN(COPY(MESSAGE,1,LIIH));
  WRITEIN;
  WRITEIN;
  WRITEIN;
  WRITEIN(NEW or OLD ? ');
  WRITEIN;
WHILE NOT DONE DO
BEGIN
  GOTOXY(1,10);
  CLRREOL;
  GOTOXY(1,12);
  CLRREOL;
  GOTOXY(19,8);
  CLRREOL;
  GOTOXY(6,6);
  CLRREOL;
  READIN(FILE_NAME);
  IF (COPY(FILE_NAME,1,1) = 'Q') OR (COPY(FILE_NAME,1,1) = 'q') THEN
  BEGIN
    TEXTBACKGROUND(BLACK);
    CLRSCR;
    HALT
  END;
  FILE_NAME := FILE_NAME + '.IN';
  GOTOXY(19,8);
  READIN(NO);
  IF (COPY(NO,1,1) = 'Q') OR (COPY(NO,1,1) = 'q') THEN
  BEGIN
    TEXTBACKGROUND(BLACK);
    CLRSCR;
    HALT
  END;
  STRCHECK(NO,'NEW','OLD','new','old',19);
  IF (NO = 'NEW') OR (NO = 'new') THEN
  BEGIN
    IF EXIST(FILE_NAME) THEN
    BEGIN
      ERROR('1',10);
      READIN(Q_E);
      STRCHECK(Q_E,'QUIT','ENTER','quit','enter',66);
      IF (Q_E = 'QUIT') OR (Q_E = 'quit') THEN
      BEGIN
        DONE := TRUE;
        FILE_CHECK := FALSE
        END
      END
    END
  END
END
ELSE BEGIN
DONE := TRUE;
FILE_CHECK := TRUE;
F_M := 1
END
ELSE BEGIN
IFS NOT EXIST (FILE_NAME) THEN
BEGIN
ERROR('2',10);
READIN(Q_E);
STRCHECK(Q_E,'QUIT','ENTER','quit','enter',65);
IF (Q_E = 'QUIT') OR (Q_E = 'QUIT') THEN
BEGIN
DONE := TRUE;
FILE_CHECK := FALSE
END
END
ELSE BEGIN
DONE := TRUE;
FILE_CHECK := TRUE;
F_M := 2
END
END;
WRITEIN
END; (* FILE_CHECK *)

(* PROCEDURE MID_R *)
PROCEDURE MID_R(MESTR:STR;IPTYPE;VAR TRP:TRF;VAR N,P,T,N:INTEGER;
CH:STRTYPE;TT:INTEGR);

TYPE TSTRTYPE = ARRAY [1..20] OF STRTYPE;

VAR DATASTR : STRTYPE;
TRST : TSTRTYPE;
DONE2, D, D1 : BOOLEAN;
TSR, SSTR :STRTYPE;
TR_N, I, J, II, TCT : INTEGER;

PROCEDURE SUB;
VAR ID : INTEGER;
BEGIN (* SUB *)
ID := 0;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
WHILE COPY(DATASTR, LENGTH(DATASTR), 1) <> '.' DO
BEGIN
  ID := ID + 1;
  TRP[(TR_N-1) * 10 + ID] := DATASTR;
  GOTOXY(1, 21);
  CLEORL;
  GOTOXY(1, 22);
  CLEORL;
  GOTOXY(1, 6 + 2 * ID);
  CLEORL;
  GOTOXY(1, 5 + 2 * ID);
  CLEORL;
  WRITEIN(DATASTR);
  GOTOXY(1, 21);
  READIN(DATASTR);
  CHARCHECK(DATASTR, 21);
END;
TRP[(TR_N-1) * 10 + ID+1] := COPY(DATASTR, 1, LENGTH(DATASTR) - 1);
GOTOXY(1, 21);
CLEORL;
GOTOXY(1, 22);
CLEORL;
GOTOXY(1, 8 + 2 * ID);
CLEORL;
GOTOXY(1, 7 + 2 * ID);
CLEORL;
WRITEIN(COPY(DATASTR, 1, LENGTH(DATASTR) - 1));
END; (* SUB *)
BEGIN (* M_ID_R *)
IF (CH = 'I') AND (T_N = TT) THEN
  ERROR('5', 23)
ELSE BEGIN
  D := FALSE;
  IF CH <> 'I' THEN
    BEGIN
      INTCHECK(19, T_N, TR_N, MESTR, D);
      IF NOT D THEN
        BEGIN
          N_P := TR_N;
          P_TS(N_P)
        END
      END;
    IF NOT D THEN
      BEGIN
        IF CH <> 'D' THEN
          BEGIN
            IF CH = 'I' THEN
              BEGIN
                N_P := T_N + 1;
                P_TS(N_P);
                CILTAB(6, 10)
              END
            END
          END
        END
      END
    END
END
END;
IF CH = 'R' THEN
BEGIN
CLRTABLE(6,10);
TEXTCOLOR(WHITE);
FOR I:=(TR_N*10-9) TO TR_N*10-5 DO
BEGIN
GOTOXY(1,5+2*(I-(TR_N-1)*10));
WRITEIN(TRP[I])
END;
TEXTCOLOR(YELLOW)
END;
GOTOXY(1,22);
CIRCLE;
GOTOXY(1,20);
CIRCLE;
IF CH = 'I' THEN
BEGIN
WRITEIN('DATA NEEDED (or E for do nothing) : ');
GOTOXY(1,21);
READIN(DATASTR);
IF (DATASTR = 'E') OR (DATASTR = 'e') THEN
BEGIN
D := TRUE;
N_P := 1;
P_TS(N_P);
FOR I:=1 TO 5 DO
BEGIN
GOTOXY(1,5+2*I);
WRITEIN(TRP[I])
END
ELSE
CHARCHECK(DATASTR,21)
END
ELSE IF CH = 'R' THEN
BEGIN
FOR I:=TR_N*10-9 TO TR_N*10-5 DO
IF TRP[I] <> '' THEN
BEGIN
TSR := '1';
TCT := 0;
DONE2 := TRUE;
FOR J:=1 TO LENGTH(TRP[I]) DO
IF COPY(TRP[I],J,1) <> ' ' THEN
BEGIN
DONE2 := FALSE;
TSR := TSR + COPY(TRP[I],J,1)
END
ELSE IF NOT DONE2 THEN
BEGIN
DONE2 := TRUE;
END
TCT := TCT + 1;
TSR[TC] := "";
TSTR[TC] := TSR;
TSR := ""
END;
IF TSR <> "" THEN
BEGIN
TCT := TCT + 1;
TSTR[TC] := TSR
END;
GOTOXY(1,20);
WRITELN('>> MODIFY LINE ' ,I-((TR_N-1)*10)+1,' OF THIS TRANS
<<');
D1 := TRUE;
II := 0;
WHILE (II < TCT) AND D1 DO
BEGIN
II := II + 1;
SSTR := "";
GOTOXY(1,21);
CLEAR;
WRITELN('OLD = ',TSTR[II],', NEW(type "" to next 1
');
READLN(SSTR);
IF SSTR = '.' THEN
BEGIN
D1 := FALSE;
GOTOXY(1,5+2*(I-(TR_N-1)*10));
CLEAR;
TRP[I] := "";
FOR J:=1 TO TCT-1 DO
BEGIN
WRITE(TSTR[J]+' ');
TRP[I] := TRP[I]+TSTR[J]+' '
END;
TRP[I] := TRP[I] + TSTR[TCT];
WRITE(TSTR[TCT])
END;
IF SSTR = "" THEN
BEGIN
GOTOXY(LENGTH(TSTR[II])+40,21);
WRITE(TSTR[II]);
DELAY(800)
END
ELSE
TSR[II] := SSTR
END;
IF D1 THEN
BEGIN
GOTOXY(1,5+2*(I-(TR_N-1)*10));
CLEAR;
TRP[I] := "";
FOR II:=1 TO TCT-1 DO
    BEGIN
        WRITE(TSTR[II]+" ");
        TRP[I] := TRP[I] + TSTR[II] + " ";
    END;
    TRP[I] := TRP[I] + TSTR[TCT];
    WRITE(TSTR[TCT]);
END
END
END;
END
END;

IF NOT D THEN
BEGIN
    CASE CH OF
        'I' : BEGIN
            T_N := T_N + 1;
            TR_N := T_N;
            SUB
        END;
        'D' : BEGIN
            FOR I:=1 TO 10 DO
                TRP[(TR_N-1)*10+I] := ";
            CURTABLE(6,10)
        END
    END
END END;

FOR I:=1 TO 4 DO
BEGIN
    GOTOXY(1,18+I);
    CIREOL
END;
GOTOXY(1,22);
WRITE(' SELECT FUNCTION : ')
END
END; (* M1_D_R *)

("""
(* PROCEDURE I_D_R *)
(* TO CREATE THE MODIFIED MODE FOR THE WORKLOAD/RELIABILITY REQUIREMENTS. *)
("""

PROCEDURE I_D_R(MESTR:STRTYPE;VAR FILEARY:FILEARRAY;VAR N_PC, T_N:INTEGER; CH:STRTYPE);

VAR DATASTR, ST : STRTYPE;
FILEARY : FILEARYTYPE;
D : BOOLEAN;
TR_N, VV, CC : INTEGER;

("""
PROCEDURE CHSCR """

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
PROCEDURE CHSCR;
BEGIN (* CHSCR *)
  IF (TR N MOD 12)=0 THEN
    GOTOXY(1,17)
  ELSE
    GOTOXY(1,(TR N MOD 12)+5);
  CLR EOL
END; (* CHSCR *)

PROCEDURE RP(RPSTR : STRTYPE; RPN : INTEGER);
VAR RP : STRTYPE;
BEGIN
  GOTOXY(6,21); CLR EOL;
  RP := "";
  WRITE(RPSTR,'OLD = ',STRARY[RPN],' NEW = ');
  READIN(RP);
  IF RP = "" THEN
    BEGIN
      GOTOXY(LENGTH(RPSTR)+LENGTH(STRARY[RPN])+19,21);
      WRITE(STRARY[RPN]);
      DELAY(1000)
    END
  ELSE
    STRARY[RPN] := RP
END;

BEGIN (* I_D_R *)
  D := FALSE;
  IF CH <> 'I' THEN
    INTCHECK(20,T_N,TR_N,MESTR,D);
  IF NOT D THEN
    BEGIN
      IF CH <> 'D' THEN
        BEGIN
          GOTOXY(6,21);
          CLR EOL;
          IF CH = 'I' THEN
            BEGIN
              WRITE('DATA NEEDED(or E for do nothing) : ');
              READIN(DATASTR);
              IF (DATASTR='E') OR (DATASTR='e') THEN
                D := TRUE
              ELSE
                CHECKERROR(DATASTR,STRARY,21)
            END
          ELSE IF CH = 'R' THEN
            BEGIN
              CHECKERROR(FILEARY[TR N],STRARY,21);
              RP('FREQUENCY OF RETRIEVAL : ',3);
              RP('FREQUENCY OF UPDATE : ',4);
            END
        END
      IF CH = 'I' THEN
        BEGIN
          WRITE('DATA NEEDED(or E for do nothing) : ');
          READIN(DATASTR);
          IF (DATASTR='E') OR (DATASTR='e') THEN
            D := TRUE
          ELSE
            CHECKERROR(DATASTR,STRARY,21)
        END
      ELSE IF CH = 'R' THEN
        BEGIN
          CHECKERROR(FILEARY[TR N],STRARY,21);
          RP('FREQUENCY OF RETRIEVAL : ',3);
          RP('FREQUENCY OF UPDATE : ',4);
        END
    END
  END
END;
RP('ALLOW FAILURE RATE : ',5)
END

IF NOT D THEN
BEGIN
CASE CH OF
'I' : BEGIN
   VAL(STRARY[1],VV,CC);
   SITE[VV] := SITE[VV] + 1;
   T N := T N + 1;
   TR N := TR N;
   WFILE(FILEARY,T N,STRARY[1],STRARY[2],STRARY[3],STRARY[4])
END;

'D' : BEGIN
   CHSCR;
   STR(TR N:2,ST);
   CHECKERROR(FILEARY[TR N],STRARY,21);
   WFILE(FILEARY,TR N,STRARY[1],ST,'O','O')
END;

'R' : BEGIN
   FILEARY[TR N] := ' ';
   WFILE(FILEARY,TR N,STRARY[1],STRARY[2],STRARY[3],STRARY[4])
END
END;

IF (TR N>(N_PC-12)) AND (TR N<=N_PC) THEN
BEGIN
   CHSCR;
   WRITEN(FILEARY[TR N])
END
ELSE BEGIN
   IF (TR N MOD 12) = 0 THEN
      N_PC := TR N
   ELSE
      N_PC := TR N + (12 - (TR N MOD 12));
   CLRTABLE(5,12);
   W_SCR(FILEARY,N_PC)
END
END
END;
GOTOXY(1,20);
CLRBL;
GOTOXY(1,21);
CLRBL
END; (* I_D_R *)
PROCEDURE SELFUN (NCT, NR, SC : INTEGER; VAR T_N : INTEGER; N_P : INTEGER; CHT : STR1TYPE; VAR FILEARY : FILEARYTYPE; VAR TRP : TRIPTYPE; MFILE : STR1TYPE; TT : INTEGER);

VAR F_N : STR1TYPE;
I, J, K, L : INTEGER;
STARY : STRARYTYPE;
DONE1 : BOOLEAN;
TFILE : TEXT;

BEGIN
GOTOXY(1,22);
WRITE(' SELECT FUNCTION : ');
GOTOXY(1,24);
WRITEIN(' I : INSERT D : DELETE R : REPLACE P : PRVIOUS PAGE');
WRITE(' N : NEXT PAGE H : HELP E : NOT MODIFY or END MODIFY Q : QUIT TO SYST
DONE1 := FALSE;
WHILE NOT DONE1 DO
BEGIN
GOTOXY(24,22);
CLREOL;
READIN(F_N);
STRCHECK(F_N);
END;
GOTOXY(1,23);
CLREOL;
CASE F_N OF
'I' : CASE CHT OF 'W' : I_D_R('',FILEARY,N_P,T_N,F_N);
'D' : MT_D_R('',TRP,N_P,T_N,F_N,TT)
END;
'D' : CASE CHT OF 'W' : I_D_R('WHICH TRANSACTION(or E for do nothing) : FILEARY,N_P,T_N,F_N);
'D' : MT_D_R('WHICH TRANSACTION(or E for do nothing) : TRP,N_P,T_N,F_N,TT)
END;
'R' : CASE CHT OF 'W' : I_D_R('WHICH TRANSACTION(or E for do nothing) : FILEARY,N_P,T_N,F_N);
'D' : MT_D_R('WHICH TRANSACTION(or E for do nothing) : TRP,N_P,T_N,F_N,TT)
END;
'N' : BEGIN
IF N_P >= T_N THEN
ERROR('6',23)
ELSE BEGIN
CLRTABLE(NCT,NR);
N_P := N_P + SC;

END;}
CASE CHT OF 'W' : W_SCR(FH:EARY, N_P);
'D' : BEGIN
  P_T S(N_P);
  TEXTCOLOR(WHITE);
  FOR I:=(N_P*10-9) TO N_P*10-5 DO
  BEGIN
    GOTOXY(1,5+2*(I-(N_P-1)*10));
    WRITEIN(TRP[I])
  END;
  TEXTCOLOR(YELLOW)
END
END;

'P' : BEGIN
  IF N_P = SCII THEN
  ERROR('7',23)
  ELSE BEGIN
    CLEARTABLE(NCT,NR);
    N_P := N_P - SCII;
    CAS E CHT OF 'W' : W_SCR(FH:EARY, N_P);
  'D' : BEGIN
   P_T S(N_P);
   TEXTCOLOR(WHITE);
   FOR I:=(N_P*10-9) TO N_P*10-5 DO
   BEGIN
    GOTOXY(1,5+2*(I-(N_P-1)*10));
    WRITEIN(TRP[I])
  END;
  TEXTCOLOR(YELLOW)
END
END;

'H' : BEGIN
CASE CHT OF 'W' : BEGIN
HELP('HELP3.HLP') ;
SCRL;
W_SCR(FH:EARY, N_P)
END;
'D' : BEGIN
HELP ('HELP4.HLP') ;
SCRL;
P_T S(N_P);
TEXTCOLOR(WHITE);
FOR I:=(N_P*10-9) TO N_P*10-5 DO
BEGIN
GOTOXY(1,5+2*(I-(N_P-1)*10));
WRITEIN(TRP[I])
END;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
TEXTCOLOR(YELLOW)
END
GO TO XY(1,22);
WRITE(' SELECT FUNCTION : ');
GO TO XY(1,24);
WRITEIN(' I : INSERT D : DELETE R : REPLACE P : PREVIOUS PAGE');
WRITE(' N : NEXT PAGE H : HELP E : NOT MODIFY OR END MODIFY Q : QUIT TO SY END;

'E' : BEGIN
DONE1 := TRUE;
ASSIGN(TFILE,PDR+';'+FILER);
REWRITE(TFILE);
CASE CHOT OF 'W' : BEGIN
I := 1;
K := 0;
WHILE SITE[I] <> 0 DO
BEGIN
WRITEIN(TFILE,SITE[I] ;1);
FOR J := 1 TO SITE[I] DO
BEGIN
K := K + 1;
CHECKERROR(FILEARY[K],STRARY ;21);
FOR L := 1 TO 2 DO
WRITEIN(TFILE,STRARY[L+2]);
END;
I := I + 1
END
END;

'D' : FOR I := 1 TO 300 DO
IF TRP[I] <> '' THEN
WRITEIN(TFILE,TRP[I])
END;
CLOSE(TFILE)
END;

'Q' : BEGIN
TEXTBACKGROUND(BLACK);
CLRSRCR;
HALT;
END
END END

(* ************************************************************** *)
(* PROCEDURE M W R                                           *)
(* THIS IS THE ORIGINAL PROCEDURE USE TO CALL THE SELFUN FOR WORKLOAD AND *)
(* RELIABILITY REQUIREMENT.                                   *)
(* ************************************************************** *)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
PROCEDURE M_W_R(FILE_NAME : STR1TYPE; VAR T_N : INTEGER);

VAR FILEARY : FILEARYTYPE;
    TRP : TRPTYPE;
    I, S_N, VV, CC, N_PC, TT : INTEGER;
    STR1, STR2, STR3, STR4, S_TN, ST1, ST2 : STR1TYPE;
    MFILE : TEXT;

BEGIN (* M_W_R *)
    ASSIGN(MFILE,FDR+' '+FILE_NAME);
    RESET(MFILE);
    T_N := 0;
    S_N := 0;
    FOR I:=1 TO 50 DO
        BEGIN
            FILEARY[I] := "";
            SITE[I] := 0
        END;
    WHILE NOT EOF(MFILE) DO
        BEGIN
            READLN(MFILE,S_TN);
            S_N := S_N + 1;
            VAL(S_TN,VV,CC);
            SITE[S_N] := VV;
            FOR I:=1 TO VV DO
                BEGIN
                    T_N := T_N + 1;
                    READLN(MFILE,STR1);
                    READLN(MFILE,STR2);
                    STR(S_N:1,ST1);
                    STR(T_N:2,ST2);
                    WFHE(FILEARY,T_N,ST1,ST2,ST1,ST2)
                END;
        END;
    CLOSE(MFILE);
    CLRSCR;
    SCR1;
    N_PC := 12;
    W_SCR(FILEARY,12);
    TT := 0;
    SELFUN(5,12,12,T_N,N_PC,'W',FILEARY,TRP,MFILE_NAME,TT)
END; (* M_W_R *)

(* *************************************************************************************************************)
(* PROCEDURE M_DR *)
(* THIS PROCEDURE IS TO CALL THE PROCEDURE SELFUN FOR TRANSACTION. *)
(* *************************************************************************************************************)

PROCEDURE M_DR(FILE_NAME : STR1TYPE;T_N : INTEGER);

VAR TRP : TRPTYPE;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
FILEARY : FILEARYTYPE;
OPERARY : FILETYPE;
I, OPN, PCT, TT : INTEGER;
MFILE : TEXT;

(*********************** PROCEDURE MKPAGE ***********************)

PROCEDURE MKPAGE(VAR OP: FILETYPE; VAR TRP: TRPTYPE);

VAR I, IND1, IND2, HMR, TP, CT : INTEGER;
ST : STR1TYPE;
DONE : BOOLEAN;
BEGIN (* MKPAGE *)
TP := 0;
HMR := 0;
IND1 := 0;
IND2 := 1;
WHILE IND1 < OPN DO
BEGIN
  DONE := TRUE;
  IND1 := IND1 + 1;
  ST := '';
  FOR I : = 1 TO LENGTH(OP[IND1]) DO
  BEGIN
    ST := ST + COPY(OP[IND1], 1,1);
    IF (COPY(OP[IND1], 1,1) = ' ') OR (I = LENGTH(OP[IND1])) THEN
    BEGIN
      IF (COPY(ST, 1, 4) = 'TEMP') AND (I = LENGTH(OP[IND1])) AND DONE THEN
      BEGIN
        DONE := FALSE;
        HMR := HMR + 1;
        ST := ''
      END;
    ELSE
    BEGIN
      ST := ''
    END;
  END;
  IF DONE THEN
  BEGIN
    CT := 0;
    FOR I : = IND2 TO IND2 + HMR DO
    BEGIN
      CT := CT + 1;
      TRP[TP+CT] := OP[I]
    END;
    TP := TP + 10;
    HMR := 0;
    IND2 := IND1 + 1
  END
END
TT := TP DIV 10

REPRODUCED WITH PERMISSION OF THE COPYRIGHT OWNER. FURTHER REPRODUCTION PROHIBITED WITHOUT PERMISSION.
BEGIN (* M DR *)
    ASSIGN (MFILE, PDR + ' : ' + FILE_NAME);
    RESET (MFILE);
    FOR I := 1 TO 50 DO
        OPERARY[I] := ' ;
    FOR I := 1 TO 300 DO
        TRP[I] := ' ;
    OPEN := 0;
    WHILE NOT EOF (MFILE) DO
        BEGIN
            OPEN := OPEN + 1;
            READLN (MFILE, OPERARY[OPEN])
        END;
    CLOSE (MFILE);
    MKPAGE (OPERARY, TRP);
    CLRSCR;
    SCR2;
    PCT := 1;
    TEXTCOLOR (WHITE);
    FOR I := 1 TO 5 DO
        BEGIN
            GOTOXY (1, 5 + 2*I);
            WRITEIN (TRP[I])
        END;
    TEXTCOLOR (YELLOW);
    SELFUN (6/10, 1, TT, 'D', FILEARY, TRP, FILE_NAME, T_N)
END;  (* M DR *)

****************************************************************************
(* PROCEDURE WORKLOAD_RELIABILITY *)
****************************************************************************

PROCEDURE WORKLOAD_RELIABILITY (VAR DONE : BOOLEAN; VAR FILE_NAME : STR1TYPE;
    VAR T_C : INTEGER);

VAR Y_N : STR1TYPE;
    F_M : INTEGER;

BEGIN
    IF FILE_CHECK ("ENTER FILE NAME FOR WORKLOAD/RELIABILITY REQUIREMENTS",
      58, F_M, FILE_NAME) THEN
        BEGIN
            DONE := TRUE;
            CHelp (Y_N);
            IF F_M = 1 THEN
                BEGIN
                    IF (Y_N = 'YES') OR (Y_N = 'yes') THEN
                        HELP ('HELP3.HELP');
                        FILL_TABLE (FILE_NAME, T_C);
                        M_W_R (FILE_NAME, T_C)
END
ELSE BEGIN
  IF F_M = 2 THEN
    BEGIN
      IF (Y_N = 'YES') OR (Y_N = 'yes') THEN
        HELP('HELP3.HLP');
        M_WR(FILE_NAME,T_C)
      END
    END
  ELSE
    DONE := FALSE
END;

(*****************************************************************************)
(* PROCEDURE DATA_REQUIRE                                  *)
(*****************************************************************************)
PROCEDURE DATA_REQUIRE(VAR DONE : BOOLEAN;VAR FILE_NAME:STR1TYPE;T_C: INTEGER);
VAR  Y_N : STR1TYPE;
  F_M : INTEGER;
BEGIN
  IF FILE_CHECK(' ENTER FILE NAME FOR THE DATA REQUIREMENTS',46,
     F_M,FILE_NAME) THEN
    BEGIN
      DONE := TRUE;
      HELP(Y_N);
      IF F_M = 1 THEN
        BEGIN
          IF (Y_N = 'YES') OR (Y_N = 'yes') THEN
            HELP('HELP4.HLP');
            FILL_TRNS(FILE_NAME,T_C);
            M_DR(FILE_NAME,T_C)
          END
        ELSE IF F_M = 2 THEN
          BEGIN
            IF (Y_N = 'YES') OR (Y_N = 'yes') THEN
              HELP('HELP4.HLP');
              M_DR(FILE_NAME,T_C)
            END
          END
        ELSE
          DONE := FALSE
        END
    END;

(*****************************************************************************)
(* PROCEDURE COPY                                *)
(*****************************************************************************)
(* COPY THE DATA FILE TO THE RESERVED DATA FILE TO CONTINUE CHECK.              *)
(*****************************************************************************)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
PROCEDURE COPY(FILE1, SOURCE : STR1TYPE);

VAR ST1 : STR1TYPE;
  W_O, CFHE : TEXT;

BEGIN
  ASSIGN(W_O,FDR*': '+SOURCE);
  REWRITE(W_O);
  ASSIGN(CFILE,FDR*': '+FILE1);
  RESET(CFILE);
  WHILE NOT EOF(CFILE) DO
    BEGIN
      READIN(CFILE,ST1);
      WRITELN(W_O,ST1)
    END;
  CLOSE(W_O);
  CLOSE(CFILE)
END;

BEGIN (* MAIN PROGRAM TRANSACT *)
  WINDO(1,1,80,25);
  TEXTBACKGROUND(2);
  CLRSCR;
  WRITELN;
  WRITELN('TRANSACTION PARAMETERS :');
  DONE := FALSE;
  WORKLOAD_RELIABILITY(DONE,FILE2,T_C);
  IF DONE THEN
    BEGIN
      COPY(FILE2,'WORK.DAT');
      CLRSCR;
      WRITELN;
      WRITELN;
      DATA_REQUIRE(DONE,FILE1,T_C);
      IF DONE THEN
        BEGIN
          COPY(FILE1,'OPER.DAT');
          ASSIGN(JJ,FDR*':MAINMENU.COM');
          EXECUTE(JJ)
        END
      ELSE BEGIN
        TEXTBACKGROUND(BLACK);
        CLRSCR
      END
    END
  ELSE BEGIN
    TEXTBACKGROUND(BLACK);
    CLRSCR
  END
END. (* MAIN PROGRAM TRANSACT *)
PROGRAM SEGMENTS;

(* ********************* ********************************************** *)
(* *
(*  A Generalized Data Distribution Model for Distributed *
(*  Relational Databases *
(* *
(* *
(* Patric D. Yurk, M.S. *
(* Western Michigan University *
(* August 1989 *
(* *
(* *
(* Program to segment the global database *
(* *
(* THIS PROGRAM TAKES THE DATABASE AND NUMBERS THE RELATIONS AND ADDS A KEY *)
(* FIELD CALLED NUMKEY TO ALL THE RELATIONS. IT THEN USES THE RELATIONAL *)
(* OPERATORS SELECT AND SEMIJOIN AS INPUT TO BREAK THE DATABASE DOWN INTO *)
(* SEGMENTS. THE RELATION NUMBERS AND THE VALUES IN THE NUMKEY FIELDS ARE *)
(* USED TO DEFINE SETS OF INTEGERS THAT CAN BE USED BY THE ALLOCATE *)
(* FRAGMENTS PROGRAM TO DEFINE DISJOINT FRAGMENTS AND ALLOCATE THEM TO THE *)
(* SITES OF THE DISTRIBUTED DATABASE. IT ALSO READS IN STATISTICAL DATA ON *)
(* THE USE OF THE DATA BASE WHICH IS WRITTEN OUT FOR USE BY THE ALLOCATE *)
(* FRAGMENTS PROGRAM TO DETERMINE WHERE TO ALLOCATE THE FRAGMENTS. *)
(* ********************* ********************************************** *)

CONST FIELD_LENGTH=12;
MAX_RELATIONS=10;
MAX_RECORDS=20;
MAX_SEG=30;
MAX_FTKTDS=15;

VAR
(* COMMON STORAGE AREA USED FOR VARIABLES THAT ARE PASSED BETWEEN MODULES *)
(*
   PER, (* PROGRAM DRIVE *)
   INDR, (* INPUT FILE DRIVE *)
   OUTDR : STRING[1]; (* OUTPUT FILE DRIVE *)
(*
(* END OF COMMON STORAGE AREA *)

ERROR: (* AN INCORRECT WORD WAS ENCOUNTERED IN THE INPUT FILE WHEN A KEYWORD WAS EXPECTED *)

BOOLEAN;
J : INTEGER;
SS : FILE;
COST_FILE: STRING[80];

PROCEDURE WORK_FILES;

PROCEDURE OUTHEAD;

BEGIN (* OUTFILE HEADING *)
CLRSCR;
WRITELN;
WRITELN('" :25, 'OUTU T_FILE');
WRITELN('" :25, '__________');
WRITELN;
WRITELN(' FILE DESCRIPTION
WRITELN('A modified copy of site and relation
WRITELN(' parameters as used by Allocate');
WRITELN;
WRITELN('Data of segments and fragments and
WRITELN(' related space and workload');
WRITELN;
WRITELN('The tuples of each fragment
WRITELN;
WRITELN('The statistical input data in a
WRITELN;
D3.OUT  --> ');
SEG1.OUT  --> ');
FRAG.OUT  --> ');
IN.OUT  --> ');}
WRITEIN(' tabulated forms');
WRITEIN;
WRITEIN('The allocation, space and benefit
        OUT.CUT  --> ');
WRITEIN(' of fragment');
WRITEIN;
WRITEIN('The tuples in each segment
        SEG.OUT  --> ');
END;

PROCEDURE WORK SCREEN;
("""
(* THE DEFAULT AND THE INPUT FILE NAME.
""
"""
VAR
 I, TC, CC : INTEGER;
RQ, NY, FN, TD, TCC : STRING[80];
TF : TEXT;
BEGIN
CLRSCR;
GOTOXY(6,12);
WRITE('DO YOU LIKE TO RUN SEGMENT OR QUIT (Run or Quit)? ');
READIN(RQ);
WHILE NOT ((RQ = 'R') OR (RQ = 'r') OR (RQ = 'Q') OR
            (RQ = 'q')) DO
  BEGIN
    GOTOXY(56,12);
    CLRRED;
    READIN(RQ)
  END;
IF (RQ = 'Q') OR (RQ = 'q') THEN
  BEGIN
    TEXTBACKGROUND(BLACK);
    CLRSCR;
    HALT
  END;
CLRSCR;
GOTOXY(1,12);
WRITE('ENTER COST FILE NAME: ');
READIN(COST_FILE);
CLRSCR;
GOTOXY(1,12);
WRITE('WOULD YOU LIKE TO ISSUE OUTPUT FILE NAME (Yes or No)? ');
READIN(NY);
WHILE NOT((NY='y') OR (NY='Y') OR (NY='n') OR (NY='N')) DO
  BEGIN
    GOTOXY(57,12);
    CLRRED;
    READIN(NY)
  END;
ASSIGN(TF,FDR+:DRIVE.DAT');
RESET(TF);
READIN(TF,TD); READIN(TF,TCC);
CLOSE(TF);
ASSIGN(TF,'DRIVE.DAT');
REWIRTE(TF);
WRITELN(TF,TD);
VAL(TCC,TC,CC);
TC:=TC+1;
WRITELN(TF,TC);
STR(TC,TCC);
IF (NY='Y') OR (NY='y') THEN
BEGIN
  OUT HEAD;
  FOR I:=1 TO 6 DO
  BEGIN
    FN := ' '; 
    GOTOXY(63,5+I*3);
    READIN(FN);
    IF FN=' ' THEN
    BEGIN
      CASE I OF 1 : FN := 'D3'+TCC;
        2 : FN := 'SEG1'+TCC;
        3 : FN := 'FRAG'+TCC;
        4 : FN := 'IN'+TCC;
        5 : FN := 'OUT'+TCC;
        6 : FN := 'SEG'+TCC
    END;
    GOTOXY(63,5+I*3);
    WRITE(FN)
    END;
  WRITELN(TF,FN+.OUT)
END;
CLOSE(TF)
END;
IF (NY='N') OR (NY='n') THEN
BEGIN
  WRITELN(TF,'DS'+TCOJ-.OUT');
  WRITELN(TF,'SEC'+TOC-.OUT');
  WRITELN(TF,'FRAG'+TCOJ-.OUT');
  WRITELN(TF,'IN'+TCOJ-.OUT');
  WRITELN(TF,'OUT'+TCOJ-.OUT');
  WRITELN(TF,'SEG'+TCOJ-.OUT');
  CLOSE(TF)
END;
CLRSCR;
TEXTCOLOR(WHITE);
GOTOXY(1,12);
WRITE('  WAIT A MINUTE, PLEASE. ........')
END;

PROCEDURE COMBINE;
(**********************************************************************)
(* TO COMBINE TWO FILE TO THE ONE (DATA.DAT). *)
VAR STR1 : STRING[80];

BEGIN
  WHILE NOT EOF(INFILE1) DO
    BEGIN
      READLIN(INFILE1,STR1);
      WRITELIN(DATA2,STR1)
      END
  END;

BEGIN (* PROCEDURE WORK_FILES *)
  WORK SCREEN;
  ASSIGN(DATA2,FDRH:DATA.DAT');
  REWRITE(DATA2);
  ASSIGN(INFILE1,PDRH:TRAN.DAT1);
  RESET(INFILE1);
  COMBINE;
  CLOSE(INFILE1);
  ASSIGN(INFILE1,PDRH:WORK.DAT');
  RESET(INFILE1);
  COMBINE;
  CLOSE(INFILE1);
  CLOSE(DATA2);
END; (* PROCEDURE WORK_FILES *)

PROCEDURE NUMBER_RELATIONS;

(* THIS PROCEDURE ADDS AN EXTRA FIELD TO EACH RELATION IN THE DATABASE. *)
(* THIS FIELD CONSISTS OF AN INTEGER GIVING THE POSITION OF THE RECORD IN *)
(* THE RELATION. IT ALSO ASSIGN A N INTEGER TO EACH RELATION. *)

LABEL EXIT;

VAR RELATION_NAME,
   FIELD_NAME:
      STRING[FIELD_LENGTH];
I,K,                            (* COUNTERS *)
NR:                             (* NUMBER OF RELATIONS IN THE DATABASE *)
   INTEGER;

FIELD:                          (* A FIELD VALUE IN THE CURRENT RECORD *)
      STRING[FIELD_LENGTH];

FIL:                            (* OUTPUT FILE NAME, THIS CHANGES FOR EACH RELATION *)
STRING[20];

RELATION_FILE, (* INPUT FILE CONTAINING THE DATABASE *)
NEW_FILE: (* THE OUTPUT FILE, THIS CHANGES FOR EACH RELATION *)

TEXT;

(**********************************************************************)
(* THIS PROCEDURE PRINTS THE NUMBER K IN THE FIELD NUMKEY AND PADS IT WITH *)
(* SPACES *)
(* *)
(* VARIABLE GLOBAL TO NUMBER_RELATIONS *)
(* *)
(* NEW_FILE THE OUTPUT FILE WHICH GETS THE DATABASE WITH THE ADDITIONAL *)
(* NUMKEY FIELD IN EACH RELATION *)
(**********************************************************************)

PROCEDURE PRINT(K: INTEGER);

VAR I:INTEGER;
  K_STRING: (* THE NUMBER OF THE KTH RECORD CONVERTED TO A STRING *)
    STRING[FIELD_LENGTH];

BEGIN
  STR(K,K_STRING);
  WRITE(NEW_FILE,K_STRING);
  FOR I:=1 TO FIELD_LENGTH-LENGTH(K_STRING) DO
    WRITE(NEW_FILE,' ');
END;

(**********************************************************************)

BEGIN (* NUMBER_RELATIONS *)
  ASSIGN(RELATION_FILE, FDR+'REL.DAT');
  RESET(RELATION_FILE);
  I:=1;
  WHILE NOT EOF(RELATION_FILE) DO
    BEGIN
      READLN(RELATION_FILE,RELATION_NAME);
      FIL:= RELATION_NAME+'.'+FIL;
      ASSIGN(NEW_FILE,FIL+':'+FIL);
      REWRITE(NEW_FILE);
      WRITELN(NEW_FILE,1);
      WRITELN(NEW_FILE,RELATION_NAME);
      WRITE(NEW_FILE, 'NUMKEY ');
      WHILE NOT EOF(RELATION_FILE) DO
        BEGIN
          READ(RELATION_FILE,FIELD_NAME);
          WRITE(NEW_FILE,FIELD_NAME);
        END;
      READLN(RELATION_FILE);
      WRITELN(NEW_FILE);
    END;
  END;

REproduced with permission of the copyright owner. Further reproduction prohibited without permission.
K:=1;  (* INITIALIZE RECORD COUNTER *)
WHILE TRUE DO
BEGIN
READ(RELATION_FILE,FIELD);
IF FIELD=' ' THEN GOTO EXIT;  (* BLANK LINE SIGNALS THE END *)
(* OF THE RELATION, EXIT LOOP *)
PRINT(K);
WRITE(NEW_FILE,FIELD);
WHILE NOT EOF(RELATION_FILE) DO
BEGIN
READ(RELATION_FILE,FIELD);
WRITE(NEW_FILE,FIELD);
END;
READIN(RELATION_FILE);
WRITEIN(NEW_FILE);
K:=K+1;  (* INCREMENT RECORD COUNTER *)
END;
EXIT:
I:=I+1;
(* INCREMENT RELATION COUNTER *)
READIN(RELATION_FILE);
CLOSE(NEW_FILE);
END;
NR:=I-1;
CLOSE(RELATION_FILE);
END;  (* NUMBER RELATIONS *)

(* This procedure takes the relational operators, select and semi-join as *)
(* Input. It then operates on the relations created in the procedure *)
(* number relations and outputs the segments of the database defined by the *)
(* relational operators that give the applications of the database at each *)
(* site *)
(* *)
(* GLOBAL VARIABLE *)
(* Error an incorrect word was encountered in the input file *)
(* when a keyword was expected *)
(* *)
(* INTERNAL PROCEDURES *)
(* Select *)
(* Semi-join *)
(* *)

PROCEDURE OPERATORS;
LABEL STOP;

VAR CH:  (* A SINGLE CHARACTER IN THE RELATIONAL OPERATOR INPUT FILE *)
CHAR;

RELATION_NUM:  (* THE INTEGER ASSIGNED TO EACH RELATION BY THE
**NUMBER_RELATIONS_PROCEDURE**

* INTEGER:  
  - TEMP_F, 
  - FILET, 
  - WORD:

* STRING[20];

* DONE, 
  - TEMPORARY:  
    - TEMP_FILE:

* BOOLEAN;

* INN, 
  - OUT, 
  - TEMP_FILE:

* TEXT;

******************************************************************************

(* THIS PROCEDURE SELECTS THE RECORDS MEETING A BOOLEAN CONDITION FROM A *)
(* RELATION IN THE DATABASE. THE RECORDS CAN EITHER BE WRITTEN TO THE *)
(* OUTPUT FILE OR TO A TEMPORARY FILE *)
(* VARIABLES GLOBAL TO THE OPERATORS PROCEDURE *)
(* )
(* )
(* RELATION_NUM THE INTEGER ASSIGNED TO EACH RELATION BY THE *)
(* NUMBER_RELATIONS_PROCEDURE *)
(* DONE THE SELECT STATEMENT IN THE INPUT FILE HAS NO *)
(* CONDITIONS SO IT IS FINISHED *)
(* TEMPORARY THE CURRENT SELECT STATEMENT IS TO A TEMPORARY FILE *)
(* INN INPUT FILE *)
(* OUT OUTPUT FILE *)
(* TEMP_FILE TEMPORARY FILE *)
(* TEMP NAME OF TEMPORARY FILE *)
(* ERROR AN INCORRECT WORD WAS ENCOUNTERED IN THE INPUT FILE *)
(* WHEN A KEYWORD WAS EXPECTED *)
(******************************************************************************)

PROCEDURE SELECT;

LABEL EXIT;

VAR RELATION_NAME, 
  - WORD, 
  - FIELD, 
  - OPERATOR, 
  - VALUE, 
  - FIL:  
    - (* NAME OF THE RELATION THE CURRENT SELECT IS OPERATING ON *)
    - (* A KEYWORD *)
    - (* NAME OF THE FIELD THE CURRENT SELECT IS CHECKING *)
    - (* THE BOOLEAN OPERATOR USED IN THE CURRENT SELECT *)
    - (* THE VALUE TO MEET IN THE BOOLEAN CONDITION OF THE CURRENT SELECT *)
    - (* NAME OF THE FILE CONTAINING THE RELATION FOR THE
CURRENT SELECT

STRING[20];

CH: (* SINGLE CHARACTER IN THE INPUT FILE *)
    CHAR;

N, (* COUNTER FOR NUMBER OF FIELDS IN THE CURRENT RELATION *)

I,J, (* COUNTERS AND LOOP CONTROL VARIABLES *)

F: (* INTEGER REPRESENTING THE POSITION OF THE FIELD IN THE BOOLEAN CONDITION FOR THE CURRENT SELECT *)

INTEGER;

FIELD_NAME,

FIELD_VALUE:
    ARRAY[1..MAX_RECORDS] OF STRING[FIELD_LENGTH];

OK, (* BOOLEAN CONDITION HAS BEEN MET *)

DONE: (* THERE IS NO BOOLEAN CONDITION IN THE CURRENT SELECT SO THE STATEMENT IS FINISHED *)

BOOLEAN;

RELATION_FILE: (* INPUT FILE FOR THE RELATION IN THE CURRENT SELECT *)

TEXT;

BEGIN (* SELECT *)

RELATION_NAME:='';

READ(INN,CH);

DONE:=FALSE;

REPEAT

    RELATION_NAME:=RELATION_NAME+CH;
    IF EOLN(INN) THEN DONE:=TRUE;
    IF NOT DONE THEN READ(INN,CH);

UNTIL (CH=' ') OR DONE;

FIL:=RELATION_NAME+'.DAT';

ASSIGN(RELATION_FILE,FDR+:FIL);

RESET(RELATION_FILE);

IF DONE THEN

BEGIN

WHILE NOT EOF(RELATION_FILE) DO

    BEGIN

        READ(RELATION_FILE,CH);

        WRITE(OUT,CH);

    END;

END

ELSE

BEGIN (* ELSE IF NOT DONE *)

    WORD:='';

    READ(INN,CH);

    REPEAT

        WORD:=WORD+CH;

        READ(INN,CH);

    END;

END
UNTIL CH=' ';
IF WORD<> 'WHERE' THEN
BEGIN
  WRITELN(WORD, '? :WHERE EXPECTED');
  ERROR:=TRUE;
END;
FIELD:=' ';
READ(INN, CH);
REPEAT
  FIELD:=FIELD+CH;
  READ(INN, CH);
UNTIL CH=' ';
FOR I:=LENGTH(FIELD)+1 TO FIELD_LENGTH DO
  FIELD:=FIELD+' ';
OPERATOR:=' ';
READ(INN, CH);
REPEAT
  OPERATOR:=OPERATOR+CH;
  READ(INN, CH);
UNTIL CH=' ';
IF LENGTH(OPERATOR)=1 THEN OPERATOR:=OPERATOR+' ';
READ(INN, CH);
VALUE:=CH;
WHILE (NOT EOLN(INN)) AND (NOT TEMPORARY) DO
  BEGIN
    READ(INN, CH);
    IF CH=' ' THEN TEMPORARY:=TRUE;
    VALUE:=VALUE+CH;
  END;
FOR I:=LENGTH(VALUE)+1 TO FIELD_LENGTH DO
  VALUE:=VALUE+' ';
IF TEMPORARY THEN
BEGIN
  WORD:=' ';
  READ(INN, CH);
  REPEAT
    WORD:=WORD+CH;
    READ(INN, CH);
  UNTIL CH=' ';
  IF WORD<> 'TO' THEN
    BEGIN
      WRITELN(WORD, '? :TO EXPECTED');
      ERROR:=TRUE;
    END;
  READ(INN, CH);
  TEMP:=CH;
  WHILE NOT EOLN(INN) DO
    BEGIN
      READ(INN, CH);
      TEMP:=TEMP+CH;
    END;
  FILET:= TEMP+.DAT';
ASSIGN (TEMP_FILE, PDR$+:`+FILET);  
REWRITE (TEMP_FILE);  
END;
READLN (RELATION_FILE, RELATION_NUM);  
IF TEMPORARY THEN WRITELN (TEMP_FILE, RELATION_NUM) 
ELSE WRITELN (OUT, RELATION_NUM) ;  
READLN (RELATION_FILE) ;  
IF TEMPORARY THEN WRITELN (TEMP_FILE, RELATION_NAME) 
ELSE WRITELN (OUT, RELATION_NAME) ;  
N:=O;  
WHILE NOT EOF (RELATION_FILE) DO 
BEGIN 
N:=N+1;  
READ (RELATION_FILE, FIELD_NAME[N]);  
FOR I:=LENGTH (FIELD_NAME[N]) +1 TO FIELD_LENGTH DO 
FIELD_NAME[N] :=FIELD_NAME[N]+`+`;  
IF TEMPORARY THEN WRITE (TEMP_FILE, FIELD_NAME[N]) 
ELSE WRITE (OUT, FIELD_NAME[N]);  
END; 
READLN (RELATION_FILE) ;  
IF TEMPORARY THEN WRITELN (TEMP_FILE) 
ELSE WRITELN (OUT) ;  
FOR I:=1 TO N DO 
IF (FIELD_NAME[I]) = FIELD THEN 
BEGIN 
F:=I;  
GOTO EXIT;  
END; 
EXIT: 
WHILE NOT EOF (RELATION_FILE) DO 
BEGIN 
I:=0;  
WHILE I<F DO 
BEGIN 
I:=I+1;  
READ (RELATION_FILE, FIELD_VALUE[I]);  
FOR J:= LENGTH (FIELD_VALUE[I]) +1 TO FIELD_LENGTH DO 
FIELD_VALUE[I] :=FIELD_VALUE[I]+`+`;  
END; 
OK:=FALSE; 
IF OPERATOR='=' THEN IF FIELD_VALUE[I] = VALUE THEN OK:=TRUE; 
IF OPERATOR='<>' THEN IF FIELD_VALUE[I] <> VALUE THEN OK:=TRUE; 
IF OPERATOR='<' THEN IF FIELD_VALUE[I] < VALUE THEN OK:=TRUE; 
IF OPERATOR='>' THEN IF FIELD_VALUE[I] > VALUE THEN OK:=TRUE; 
IF OPERATOR='=' THEN IF FIELD_VALUE[I] >= VALUE THEN OK:=TRUE; 
IF OK THEN 
BEGIN 
WHILE I<N DO 
BEGIN 
I:=I+1;  
READ (RELATION_FILE, FIELD_VALUE[I]);

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
END;
FOR I:=1 TO N DO
BEGIN
    IF TEMPORARY THEN WRITE(TMP_FILE, FIELD_VALUE[I])
    ELSE WRITE(OUT, FIELD_VALUE[I]);
END;
FOR I:=1 TO N DO
BEGIN
    IF TEMPORARY THEN WRITEIN(TMP_FILE)
    ELSE WRITEIN(OUT);
END;
READIN(RELATION_FILE);
END;
END; (* ELSE IF NOT DONE *)
CLOSE(RELATION_FILE);
END; (* SELECT *)

-----------------------------------------------------------------------
(* THIS PROCEDURE PERFORMS A SEMIJOIN OF A FIRST RELATION WITH SECOND *)
(* RELATION WHERE THE SPECIFIED FIELDS OF EACH RELATION MATCH THE OUTPUT *)
(* MAY BE TO A TEMPORARY FILE *)
(* *)
(* VARIABLES GLOBAL TO THE OPERATORS PROCEDURE *)
(* *)
(* RELATION_NUM THE INTEGER ASSIGNED TO EACH RELATION BY THE *)
(* NUMBER RELATIONS PROCEDURE *)
(* INN INPUT FILE *)
(* OUT OUTPUT FILE *)
(* ERROR AN INCORRECT WORD WAS ENCOUNTERED IN THE INPUT FILE *)
(* WHEN A KEYWORD WAS EXPECTED *)
(* TEMPFILE TEMPORARY FILE *)
(* TEMPN NAME OF TEMPORARY FILE *)
(* TEMPARY THE CURRENT SELECT STATEMENT IS TO A TEMPORARY FILE *)
(* *)
(* *)
(* CLEAR CLEARS ALL THE FIELD VALUES IN BOTH THE CURRENT RELATIONS *)
(* SORT Sorts THE RELATION IN THE CURRENT SEMIJOIN ON THE FIELDS TO BE MATCHED *)
-----------------------------------------------------------------------

PROCEDURE SEMIJOIN;
LABEL EXIT,EXIT1;
TYPE ARY=ARRAY[1..MAX_RECORDS,1..MAX_FIELDS] OF STRING[FIELD_LENGTH];

VAR RELATION_ONE, (* THE FIRST RELATION IN THE CURRENT SEMIJOIN *)
    RELATION_TWO, (* THE SECOND RELATION IN THE CURRENT SEMIJOIN *)
    FILE1, (* NAME OF THE FILE CONTAINING THE FIRST RELATION *)
    FILE2, (* NAME OF THE FILE CONTAINING THE SECOND RELATION *)
    WORD, (* A KEYWORD *)
    FIELD1, (* THE MATCH FIELD IN THE FIRST RELATION *)

FIELD2, (* THE MATCH FIELD IN THE SECOND RELATION *)
OPERATOR: (* THE OPERATOR IN THE BOOLEAN CONDITION, MUST BE "=" *)
STRING[20];

I, J, K, (* LOOP CONTROL VARIABLES AND COUNTERS *)
S, (* COUNTER FOR THE NUMBER OF FIELDS IN THE FIRST RELATION *)
P1, (* POSITION OF THE MATCH FIELD IN THE FIRST RELATION *)
P2, (* POSITION OF THE MATCH FIELD IN THE SECOND RELATION *)
N, (* NUMBER OF RECORDS IN THE FIRST RELATION *)
M, (* NUMBER OF RECORDS IN THE FIRST RELATION *)

INTEGER;

FIELD NAME:
ARRAY[1..MAX_FIELDS] OF STRING[FIELD_LENGTH];

FIELD ONE, (* THE FIELD VALUES IN THE FIRST RELATION *)
FIELD TWO: (* THE FIELD VALUES IN THE FIRST RELATION *)
ARY;

RELATION_FILE1, (* INPUT FILE CONTAINING THE FIRST RELATION *)
RELATION_FILE2: (* INPUT FILE CONTAINING THE FIRST RELATION *)

TEXT;

**************************************************************************
PROCEDURE CLEAR;
;
VAR I, J: INTEGER;
BEGIN
FOR I:=1 TO MAX_RECORDS DO
  FOR J:=1 TO MAX_FIELDS DO
    BEGIN
      FIELD ONE[I, J]:=' ';  
      FIELD TWO[I, J]:=' '; 
    END;
END;

**************************************************************************
(* THIS PROCEDURE PERFORMS A RECURSIVE QUICKSORT ON THE RECORDS OF *)
(* REALTION A IN ASCENDING ORDER OF FIELD P. F IS THE NUMBER THE FIRST *)
(* RECORD IN A, L IS THE NUMBER OF THE LAST *)
(*)
(*)
(* PARTITION *)
(**************************************************************************)

PROCEDURE SORT(VAR A:ARY; P, F, L: INTEGER);

VAR J: (* POSITION OF PARTITIONING ELEMENT *)

INTEGER;
PROCEDURE PARTITION(VAR B, E: INTEGER);
VAR I: (* COUNTER *) INTEGER;
   T, (* TEMPORARY FIELD VALUE USED IN SWAPPING *)
   V1: (* FIELD VALUE OF FIRST RECORD IN A *)
STRING[FIELD_LENGTH];
   V: (* FIRST RECORD IN A *)
ARRAY[1..10] OF STRING[FIELD_LENGTH];
BEGIN
  FOR K:=1 TO S DO
    V[K]:=A[B,K];
    V1:=A[B,P];
    I:=B;
    REPEAT
      I:=I+1;
      UNTIL A[I,P]>=V1;
    REPEAT
      E:=E-1;
      UNTIL A[E,P]<=V1;
      IF I<=E THEN
        FOR K:=1 TO S DO
          BEGIN
          END;
        UNTIL I>=E;
        FOR K:=1 TO S DO
          BEGIN
            A[B,K]:=A[E,K];
            A[E,K]:=V[K];
          END;
    END;
END (* SORT *)
IF F<L THEN
BEGIN
  J:=L+1;
  PARTITION(F, J);
  SORT(A, P, F, J-1);
  SORT(A, P, J+1, L);
END; (* SORT *)
BEGIN (* SEMIJOIN *)
RELATION_ONE:='';
READ(INN, CH);
REPEAT
  RELATION_ONE:=RELATION_ONE+CH;
  READ(INN, CH);
UNTIL CH=' '
WORD=''
READ(INN, CH);
REPEAT
  WORD:=WORD+CH;
  READ(INN, CH);
UNTIL CH=' '
IF WORD<> 'WITH' THEN
  BEGIN
    WRITELN(WORD,'? : WITH EXPECTED');
    ERROR:=TRUE;
  END;
RELATION_TWO:='';
READ(INN, CH);
REPEAT
  RELATION_TWO:=RELATION_TWO+CH;
  READ(INN, CH);
UNTIL CH=' '
FILE1:= RELATION_ONE+'.DAT';
FILE2:= RELATION_TWO+'.DAT';
WORD=''
READ(INN, CH);
REPEAT
  WORD:=WORD+CH;
  READ(INN, CH);
UNTIL CH=' '
IF WORD<> 'WHERE' THEN
  BEGIN
    WRITELN(WORD,'? : WHERE EXPECTED');
    ERROR:=TRUE;
  END;
FIELD1=''
READ(INN, CH);
REPEAT
  FIELD1:=FIELD1+CH;
  READ(INN, CH);
UNTIL CH=' '
FOR I:=LENGTH(FIELD1)+1 TO FIELD_LENGTH DO
  FIELD1:=FIELD1+' ';
OPERATOR=''
READ(INN, CH);
REPEAT
  OPERATOR:=OPERATOR+CH;
  READ(INN, CH);
UNTIL CH=' '
IF OPERATOR<> '=' THEN
BEGIN
  WRITELN(OPERATOR,'? : = EXPECTED');
  ERROR:=TRUE;
END;
READ(INN,CH);
FIELD2:=CH;
WHILE (NOT EOLN(INN)) AND (NOT TEMPORARY) DO
  BEGIN
    READ(INN,CH);
    IF CH='"' THEN TEMPORARY:=TRUE;
    FIELD2:=FIELD2+CH;
  END;
FOR I:=LENGTH(FIELD2)+1 TO FIELD_LENGTH DO
  FIELD2:=FIELD2+' '; 
IF TEMPORARY THEN
  BEGIN
    WORD:='';
    READ(INN,CH);
    REPEAT
      WORD:=WORD+CH;
      READ(INN,CH);
    UNTIL CH=';r';
    IF WORD<> 'TO' THEN
      BEGIN
        WRITELN(WORD,'? : TO EXPECTED');
        ERROR:=TRUE;
      END;
  READ(INN,CH);
  TEMP:=CH;
  WHILE NOT EOLN(INN) DO
    BEGIN
      READ(INN,CH);
      TEMP:=TEMP+CH;
    END;
  FILET:=TEMP+'.DAT';
  ASSIGN(TEMP_FILE,FDR::'+FILE');
  REWRITE(TEMP_FILE);
END;
CLEAR;
ASSIGN(RELATION_FILE1,FDR::'+FILE1);
RESET(RELATION_FILE1);
ASSIGN(RELATION_FILE2,FDR::'+FILE2);
RESET(RELATION_FILE2);
READIN(RELATION_FILE1,RELAATION_NUM);
IF TEMPORARY THEN WRITELN(TEMP_FILE,RELAATION_NUM)
ELSE WRITELN(OUT,RELAATION_NUM);
READIN(RELATION_FILE1);
IF TEMPORARY THEN WRITELN(TEMP_FILE,RELAATION_ONE)
ELSE WRITELN(OUT,RELAATION_ONE); 
S:=S+0;
WHILE NOT EOLN(RELATION_FILE1) DO
  BEGIN
S:=S+1;
READ(RELATION_FILE1, FIELD_NAME[S]);
FOR I:=LENGTH(FIELD_NAME[S])+1 TO FIELD_LENGTH DO
   FIELD_NAME[S]:=FIELD_NAME[S]+' ';
   IF TEMPORARY THEN WRITE(TEMP_FILE, FIELD_NAME[S])
ELSE WRITE(OUT, FIELD_NAME[S]);
END;
READIN(RELATION_FILE1);
IF TEMPORARY THEN WRITEIN(TEMP_FILE)
ELSE WRITEIN(OUT);
FOR I:=1 TO S DO
   IF FIELD_NAME[I]=FIELD1 THEN
      BEGIN
         P1:=I;
         GOTO EXIT;
         END; EXIT:
         I:=0;
   WHILE NOT EOF(RELATION_FILE1) DO
      BEGIN
         I:=I+1;
         J:=0;
         WHILE NOT EOF(RELATION_FILE1) DO
            BEGIN
               J:=J+1;
               READ(RELATION_FILE1, FIELD_ONE[I,J]);
               FOR K:=LENGTH(FIELD_ONE[I,J])+1 TO FIELD_LENGTH DO
                  FIELD_ONE[I,J]:=FIELD_ONE[I,J]+' ';
               END;
               READIN(RELATION_FILE1);
            END;
         N:=I;
      END;
   N:=I;
   READIN(RELATION_FILE2);
   READIN(RELATION_FILE2);
   I:=0;
   WHILE NOT EOF(RELATION_FILE2) DO
      BEGIN
         I:=I+1;
         READ(RELATION_FILE2, FIELD_NAME[I]);
         FOR K:=LENGTH(FIELD_NAME[I])+1 TO FIELD_LENGTH DO
            FIELD_NAME[I]:=FIELD_NAME[I]+' ';
         IF FIELD_NAME[I]=FIELD2 THEN
            BEGIN
               P2:=I;
               GOTO EXIT1;
            END;
         END;
     EXIT1:
     READIN(RELATION_FILE2);
     I:=0;
   WHILE NOT EOF(RELATION_FILE2) DO
      BEGIN

I := I + 1;
J := 0;
WHILE NOT EOF(RELATION_FILE2) DO
  BEGIN
    J := J + 1;
    READ(RELATION_FILE2,FIELD_TWO[I,J]);
    FOR K := LENGTH(FIELD_TWO[I,J]) + 1 TO FIELD_LENGTH DO
      FIELD_TWO[I,J] := FIELD_TWO[I,J] + ' ';
  END;
  READIN(RELATION_FILE2);
END;
M := I;
SORT(FIELD_ONE,P1,1,N);
SORT(FIELD_TWO,P2,1,M);
I := 1; J := 1;
WHILE (I <= N) AND (J <= M) DO
  BEGIN
    IF FIELD_ONE[I,P1] < FIELD_TWO[J,P2] THEN I := I + 1;
    IF FIELD_ONE[I,P1] > FIELD_TWO[J,P2] THEN J := J + 1;
    IF FIELD_ONE[I,P1] = FIELD_TWO[J,P2] THEN
      BEGIN
        FOR K := 1 TO S DO
          IF TEMPORARY THEN WRITE(TEMP_FILE,FIELD_ONE[I,K])
            ELSE WRITE(OUT,FIELD_ONE[I,K]);
        IF TEMPORARY THEN WRITE(TEMP_FILE)
          ELSE WRITE(OUT);
        I := I + 1;
      END;
  END;
CLOSE(RELATION_FILE1);
CLOSE(RELATION_FILE2);
IF TEMPORARY THEN CLOSE(TEMP_FILE);
END; (* SEMIJOIN *)

(*************************************************************************)

BEGIN (* OPERATORS *)
  ASSIGN(INN, FOR ' OPER.DAT');
  RESET(INN);
  ASSIGN(OUT, OUTDIR ' SEG.OUT');
  REWRITE(OUT);
  WHILE NOT EOF(INN) DO
    BEGIN
      ERROR := FALSE;
      WORD := '';
      READ(INN, CH);
      TEMPORARY := FALSE;
      DONE := FALSE;
      REPEAT
        WORD := WORD + CH;
        IF EOF(INN) THEN DONE := TRUE;
        IF NOT DONE THEN READ(INN, CH);
      END;
    END;
END;
UNTIL (CH=' ') OR DONE;
IF WORD='SELECT' THEN SELECT
ELSE IF WORD='SEMIJOIN' THEN SEMIJOIN
ELSE
BEGIN
  WRITELN(WORD,'? :SELECT OR SEMIJOIN EXPECTED');
  GO TO STOP;
END;
IF ERROR THEN GO TO STOP;
READIN(INN);
IF NOT TEMPORARY THEN WRITELN(OUT);
END;
CLOSE(INN);
CLOSE(OUT);
STOP:
END; (* OPERATORS *)

(* THIS PROCEDURE TAKES THE SEGMENTS OF THE GLOBAL DATABASE GIVEN BY *)
(* PROCEDURE OPERATORS AND FOR EACH RELATION DEFINES THE SEGMENTS OF THAT *)
(* RELATION AS A SET OF INTGERS REPRESENTING THE POSITION OF THE RECORDS *)
(* IN THE RELATION. THIS PROCEDURE ALSO READS IN THE NUMBER OF RELATIONS IN *)
(* THE DATABASE, THE NUMBER OF RECORDS IN EACH RELATION, THE NUMBER OF SITES*)
(* IN THE DISTRIBUTED DATABASE, THE AVERAGE COST OF A UNIT OF SPACE, THE *)
(* AVERAGE COST OF A UNIT RETRIEVAL, THE AVERAGE COST OF AN UPDATE, AND THE *)
(* AVERAGE COST OF COMMUNICATION BETWEEN TWO SITES. ALL OF THIS INFORMATION *)
(* IS OUTPUT ALONG WITH THE SEGMENTS FOR USE BY THE ALLOCATE FRAGMENTS *)
(* PROGRAM AND IT IS ALSO OUTPUT TO A FILE LABELING THE INPUT *)
(*)
(* INTERNAL PROCEDURES *)

PROCEDURE NUMBER_SEG;

CONST MAX_FRAG=40;
MAX_SITE=10;

TYPE FRAGSET=SET OF 1..MAX_FRAG;

VAR I,J, (* COUNTERS AND LOOP CONTROL VARIABLES *)
NR, (* NUMBER OF RELATIONS IN THE GLOBAL DATABASE *)
NSITE, (* NUMBER OF SITES IN THE DISTRIBUTED DATABASE *)
N_APP, (* NUMBER OF APPLICATIONS OF THE CURRENT RELATION AT *)
(* THE CURRENT SITE *)
RELATION_NUM, (* THE NUMBER OF THE CURRENT RELATION *)
NUMKEY, (* FIELD GIVING THE POSITION OF A RECORD IN THE *)
(* RELATION *)
C: (* ERROR CODE REQUIRED BY THE VAL FUNCTION *)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
INTEGER;

NUM: (* THE NUMBER OF THE SEGMENT OF A GIVEN RELATION *)
ARRAY[1..MAX_FRAG] OF INTEGER;

REC_SIZE, (* THE NUMBER OF BYTES IN A RECORD *)
REC_UNIT, (* DEFINE NEW VARIABLE THE NUMBER OF UNITS IN A RECORD *)
R: (* THE NUMBER OF RECORDS IN A RELATION *)
ARRAY[1..MAX_RELATIONS] OF INTEGER;

NUM_STRING: (* NUMKEY CONVERSION TO A STRING *)
STRING[FIELD_LENGTH];

SEG: (* ARRAY OF SETS OF INTEGRERS REPRESENTING THE SEGMENTS OF THE DATABASE *)
ARRAY[1..MAX_RELATIONS,1..MAX_FRAG] OF FRAGSET;

SITE OF, (* THE SITE OF A GIVEN SEGMENT *)
FREQ_R, (* FREQUENCY OF RETRIEVAL OF A GIVEN SEGMENT *)
FREQ_U, (* FREQUENCY OF UPDATE OF A GIVEN SEGMENT *)
AV_RET_SIZE: (* AVERAGE RETRIEVAL SIZE IN NUMBER OF RECORDS *)
ARRAY[1..MAX_RELATIONS,1..MAX_FRAG] OF INTEGER;

SEGMENT_FILE, (* FILE CONTAINING THE SEGMENTS FROM PROCEDURE OPERATORS *)
DATA, (* INPUT FILE *)
IN_PUT, (* OUTPUT FILE CONTAINING THE INFORMATION IN DATA WITH EXPLANATION *)
GST,
OUT: (* OUTPUT FILE *)
TEXT;

(************************************************************************************)
(* THIS PROCEDURE CLEAR THE SEGMENTS *)
(************************************************************************************)

PROCEDURE CLEAR;

VAR I,J:INTEGER;
BEGIN
FOR I:=1 TO NR DO
  FOR J:=1 TO MAX_SITE DO
    BEGIN
      SEG[I,J]:=[];
    END;
END;

(************************************************************************************)
(* THIS PROCEDURE PRINTS THE SET X TO THE OUTPUT FILE *)
(************************************************************************************)
PROCEDURE PRINT(X:FRAGSET);
VAR I:INTEGER;
BEGIN
  FOR I:=1 TO MAX_FRAG DO
    IF I IN X THEN
      WRITE(OUT, ' ', I);
      WRITELN(OUT);
END;

(* THIS PROCEDURE OUTPUTS THE INFORMATION TO BE USED BY THE FRAGMENT *)
(* ALLOCATION PROGRAM *)

PROCEDURE OUT_PUT;
VAR I,J,J:INTEGER;
BEGIN
  ASSIGN(OUT,OUTDR'+':D3.OUT');
  REWRITE(OUT);
  WRITELN(OUT,NSITE);
  WRITELN(OUT,NR);
  FOR I:=1 TO NR DO
    BEGIN
      WRITELN(OUT,R[I]);
      WRITELN(OUT,REC_SIZE[I]);
    END;
  FOR I:=1 TO NR DO
    BEGIN
      FOR J:=1 TO NUM[I]-1 DO
        BEGIN
          WRITELN(OUT,SITE_OF[I,J]);
          WRITELN(OUT,FREQ_R[I,J]);
          WRITELN(OUT,FREQ_U[I,J]);
          PRINT(SEG[I,J]);
        END;
      WRITELN(OUT,0);
    END;
  CLOSE(OUT);
END;

BEGIN (* NUMBER_SEG *)
  ASSIGN(SEGMENT_FILE,OUTDR'+':SEG.OUT');
  RESET(SEGMENT_FILE);
  ASSIGN(DATA,PDR'+':DATA.DAT');
  RESET(DATA);
  ASSIGN(IN_PUT,OUTDR'+':IN.OUT');
REWRITE(IN_PUT);
WRITEIN(IN_PUT,'********************');
WRITEIN(IN_PUT,'(* FILE NAME: IN.OUT *)');
WRITEIN(IN_PUT,'********************');
WRITEIN(IN_PUT); WRITEIN(IN_PUT);
READ(DATA,NR);
WRITEIN(IN_PUT,'NUMBER OF RELATIONS : ',NR);
WRITEIN(IN_PUT);
WRITEIN(IN_PUT,'NUMBER OF RECORDS IN, RECORD_SIZE, RECORD_UNIT OF EACH RELATION')
WRITEIN(IN_PUT);
WRITEIN(IN_PUT,'RELATION # ':13,'RECORDS':9,'RECORD_SIZE':13);
WRITEIN(IN.Put, '-------------------------------------------------------------');
FOR I:=1 TO NR DO
BEGIN
 WRITE(IN_RJT,I:7);
 READ(DATA,R[I]);
 WRITE(IN_PUT,R[I]:12);
 READ(DATA,REC_SIZE[I]);
 WRITE(IN_PUT,REC_SIZE[I]:12);
 NUM[I]:=1;
 WRITEIN(IN_PUT);
END;
ASSIGN(CST,INLRf':'+COST_FHE+' .IN');
RESET(CST);
READIN(CST,NSITE);
CLOSE(CST);
CLEAR;
WRITEIN(IN.Put,' ':15,'FREQUENCY OF','  FREQUENCY  OF ');
WRITEIN(IN.Put,' ':15,'RETRIEVAL ','  UPDATE  ');
WRITEIN(IN.Put, ' ':15, '--------------------------------------------');
WRITEIN(IN_PUT);
FOR I:=1 TO NSITE DO
BEGIN
 READIN(DATA,N_APP);
 WRITEIN(IN_PUT,' ':15,'NUMBER OF APPLICATIONS AT SITE ',N_APP);
 WRITEIN(IN_PUT);
FOR J:=1 TO N_APP DO
 BEGIN
 READIN(SEGMENT_FILE,RELATION_NUM);
 READIN(SEGMENT_FILE);
 READIN(SEGMENT_FILE);
 READIN(SEGMENT_FILE,NUMKEY);
 SITE_OF[RELATION_NUM,NUM[RELATION_NUM]]:=I;
 READIN(DATA,FREQ_R[RELATION_NUM,NUM[RELATION_NUM]]);
 WRITE(IN_PUT,' ':15,FREQ_R[RELATION_NUM,NUM[RELATION_NUM]]:6);
 READIN(DATA,FREQ_U[RELATION_NUM,NUM[RELATION_NUM]]);
 WRITE(IN_PUT,FREQ_U[RELATION_NUM,NUM[RELATION_NUM]]:13);
 WRITEIN(IN_PUT);
 WHILE NUMKEY<>0 DO
 BEGIN
 SEG[RELATION_NUM,NUM[RELATION_NUM]]:=
 SEG[RELATION_NUM,NUM[RELATION_NUM]]+[NUMKEY];

code...
READIN(SEGMENT_FILE, NUM_STRING);
VAL(NUM_STRING, NUMKEY, C);
IF NUM_STRING="" THEN NUMKEY:=0;
END;
NUM[RELATION_NUM]:=NUM[RELATION_NUM]+1;
END;
END;
OUT_PUT;
CLOSE(IN_PUT);
END; (* NUMBER_SEG *)

BEGIN (* MAIN *)
WINDOW(1,1,80,25);
WORK_FILES;
NUMBER_RELATIONS;
OPERATORS;
IF NOT ERROR THEN
BEGIN
NUMBER_SEG;
ASSIGN(SS, FDR+'\:MAINMENU.COM');
EXECUTE(SS)
END
END. (* MAIN *)
PROGRAM FRAGMENT;

(CONST MAX_SITE=10;
 MAX_REL=4;
 MAX_FRAG=30;
 MAX_APPLICATION=30;
 MAX_REC_SIZE=50;
 MAX_TUPLE=20;
MAX_FIELD_LENGTH = 12;

TYPE
FRAGTYPE = RECORD
  SEGM: ARRAY [1..MAX_APPLICATION] OF INTEGER;
  SITES: ARRAY [1..MAX_APPLICATION] OF INTEGER;
  RELATION: INTEGER;
  SIZE: INTEGER;
  TUPLES: SET OF 1..MAX_TUPLE;
END;

SEGTYPE = RECORD
  SITES: INTEGER;
  RELATION: INTEGER;
  FREQR: REAL;
  FREQU: REAL;
  TUPLES: SET OF 1..MAX_TUPLE;
END;

VAR

(* COMMON STORAGE AREA USED FOR VARIABLES THAT ARE PASSED BETWEEN MODULES *)
(* PROGRAM DRIVE *)
(* INPUT FILE DRIVE *)
(* OUTPUT FILE DRIVE *)
(* END OF COMMON STORAGE AREA *)

FRAGMENTS: ARRAY [1..MAX_FRAG] OF FRAGTYPE;
SEGMENTS: ARRAY [1..MAX_APPLICATION] OF SEGTYPE;
REC_SIZE: ARRAY [1..MAX_APPLICATION] OF INTEGER;
FRAG_FILENAME : STRING[15];
FRAGMENT_FILE (* OUTPUT FILE OF THE FRAGMENTS *)
  : TEXT;
EX : FILE;

NFRAGS,
S,I,J,
NSEGS,
FI : INTEGER;
NIF : BOOLEAN;

(* THIS PROCEDURE CLEARS THE FRAGMENTS AND SEGMENTS TABLES *)
PROCEDURE TCLEAR;

VAR I,J : INTEGER;

BEGIN
  FOR I:=1 TO MAX_FRAG DO
    FOR J:=1 TO MAX_APPLICATION DO
      BEGIN
        FRAGMENTS[I].SEGM[J]:=0;
        FRAGMENTS[I].SITES[J]:=0;
      END;
      FRAGMENTS[I].TUPLES:=[];
      FRAGMENTS[I].RELATION:=0;
      FRAGMENTS[I].SIZE:=0;
      END;

  FOR I:=1 TO MAX_APPLICATION DO
    BEGIN
      SEGMENTS[I].TUPLES:=[];
      SEGMENTS[I].SITES:=0;
      SEGMENTS[I].RELATION:=0;
    END;

END; (* TCLEAR *)

PROCEDURE OUT_FRAG;

VAR
  C,I,J : INTEGER;
  INFILE: TEXT;

BEGIN
  ASSIGN(INFILE, OUTDIR+'.'+FRAG_FILENAME+'.'+OUT');
  WRITELN(INFILE,'**************');
  WRITELN(INFILE,'FRAGMENT DATA');
  WRITELN(INFILE,'**************');
  WRITELN(INFILE);
  WRITELN(INFILE,NFRAGS,' FRAGMENTS');
  WRITELN(INFILE);
  WRITELN(INFILE,'SEGMENT(S)');
  WRITELN(INFILE,' & SITE(S)');
  WRITELN(INFILE,'----------------------- RELATION SIZE TUPLES');
  FOR I := 1 TO NFRAGS DO
    BEGIN

  (*...*)

  (*********hilf1*********)

PROCEDURE TCLEAR;

VAR I,J : INTEGER;

BEGIN
  FOR I:=1 TO MAX_FRAG DO
    FOR J:=1 TO MAX_APPLICATION DO
      BEGIN
        FRAGMENTS[I].SEGM[J]:=0;
        FRAGMENTS[I].SITES[J]:=0;
      END;
      FRAGMENTS[I].TUPLES:=[];
      FRAGMENTS[I].RELATION:=0;
      FRAGMENTS[I].SIZE:=0;
      END;

  FOR I:=1 TO MAX_APPLICATION DO
    BEGIN
      SEGMENTS[I].TUPLES:=[];
      SEGMENTS[I].SITES:=0;
      SEGMENTS[I].RELATION:=0;
    END;

END; (* TCLEAR *)

PROCEDURE OUT_FRAG;

VAR
  C,I,J : INTEGER;
  INFILE: TEXT;

BEGIN
  ASSIGN(INFILE, OUTDIR+'.'+FRAG_FILENAME+'.'+OUT');
  WRITELN(INFILE,'**************');
  WRITELN(INFILE,'FRAGMENT DATA');
  WRITELN(INFILE,'**************');
  WRITELN(INFILE);
  WRITELN(INFILE,NFRAGS,' FRAGMENTS');
  WRITELN(INFILE);
  WRITELN(INFILE,'SEGMENT(S)');
  WRITELN(INFILE,' & SITE(S)');
  WRITELN(INFILE,'----------------------- RELATION SIZE TUPLES');
  FOR I := 1 TO NFRAGS DO
    BEGIN

  (*...*)
FOR J := 1 TO 7 DO
  WRITE(INFILE, FRAGMENTS[I].SEGMENT[J]:3);
  WRITE(INFILE, FRAGMENTS[I].RELATION:8);
  WRITE(INFILE, FRAGMENTS[I].SIZE:10);
  WRITE(INFILE, ' ');
FOR J := 1 TO MAX_TUPLE DO
  IF (J IN FRAGMENTS[I].TUPLES) THEN WRITE(INFILE, J:3);
  WRITELN(INFILE);
FOR J := 1 TO 7 DO
  WRITE(INFILE, FRAGMENTS[I].SITES[J]:3);
  WRITE(INFILE);
  WRITE(INFILE);
END;
WRITELN(INFILE, '***************************');
WRITELN(INFILE, 'GMEMENT DATA');
WRITELN(INFILE, '***************************');
WRITELN(INFILE);
WRITELN(INFILE, NSEGS, ' SEGMENTS');
WRITELN(INFILE);
WRITELN(INFILE, 'SITE RELATION FREQR FREQ TUPLES');
WRITELN(INFILE, '---------------------');
FOR I := 1 TO NSEGS DO
  BEGIN
    WRITE(INFILE, SEGMENES[I].SITES:3);
    WRITE(INFILE, SEGMENES[I].RELATION:9);
    WRITE(INFILE, SEGMENES[I].FREQ:13:2);
    WRITE(INFILE, SEGMENES[I].FREQ:11:2);
    WRITE(INFILE, ' ');
    FOR J := 1 TO MAX_TUPLE DO
      IF (J IN SEGMENES[I].TUPLES) THEN WRITE(INFILE, J:3);
    WRITELN(INFILE);
    WRITELN(INFILE);
  END;
CLOSE(INFILE);
END; (* OUT_FRAG *)

(* INPUT FILE NAMES *)
PROCEDURE FRAG_SCREEN;
BEGIN (* FRAG_SCREEN *)
  CLRSCR;
  TEXTCOLOR(RED);
  WRITELN(' ', 34, 'PAIRWISE DISJOINT FRAGMENTS');
  WRITELN(' ', 34, '---------------------');
  WRITELN;
  WRITELN('FRAGMENT FILE NAME:');
  (* SHOW THE FUNCTION KEYS ON ROW 24 *)
  GOTOXY(1, 24);
  WRITE('Q => QUIT TO SYSTEM');
  TEXTCOLOR(YELLOW);
END; (* FRAG_SCREEN *)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(* GET SEGMENT AND RELATION DATA FROM INPUT FILES *)

PROCEDURE GET_SEGMENTS;

VAR

FRQR, FRQU: REAL;
I, DMY, S, ST, N, NR: INTEGER;
INN: TEXT;

BEGIN
ASSIGN(INN,'D3.OUT');
RESET(INN);
(* GET RELATION DATA *)
READIN(INN, DMY);
READIN(INN, NR);
FOR I:=1 TO NR DO
BEGIN
READIN(INN, DMY);
READIN(INN, REC_SIZE[I]);
END;

S:=1;
FOR I:=1 TO NR DO
BEGIN
READIN(INN, ST);
READIN(INN, FRQR);
READIN(INN, FRQU);
WHILE ST<>0 DO (* ZERO IS USED IN THE INN FILE TO SEPARATE THE INPUT * )
FOR EACH RELATION
BEGIN
SEGMENTS[S].SITES:=ST;
SEGMENTS[S].RELATION:=I;
SEGMENTS[S].FREQR:=FRQR;
SEGMENTS[S].FREQU:=FRQU;
WHILE NOT EOIN(INN) DO BEGIN
READ(INN, N);
SEGMENTS[S].TUPLES:=SEGMENTS[S].TUPLES+[N];
END;
READIN(INN, ST);
S:=S+1;
IF ST<>0 THEN
BEGIN
READIN(INN,FRQR);
READIN(INN,FRQU);
END;
END; (* WHILE ST *)
END; (* FOR I *)
NSEGS:=S-1;
CLOSE(INN);
END; (* GET_SEGMENTS *)

(* ADD A SEGMENT TO THE FRAGMENT TABLE AS A NEW FRAGMENT *)
PROCEDURE ADD_SEGMENT(S:INTEGER);
VAR
  I: INTEGER;
BEGIN
  I:=1;
  WHILE (FRAGMENTS[I].RELATION <> 0) I:=I+1;
  NFRAGS:=I;
  FRAGMENTS[I].SEGM :=S;
  FRAGMENTS[I].SITES :=SEGM.SITES;
  FRAGMENTS[I].RELATION :=SEGM.RELATION;
  FRAGMENTS[I].TUPLES :=SEGM.TUPLES;
END; (* ADD_SEGMENT *)

(* ADD SEGMENTS FROM A SEGMENT TO THE FRAGMENT SPECIFIED *)
PROCEDURE ADD_SEGM(S,FI:INTEGER);
VAR
  J: INTEGER;
BEGIN
  J:=1;
  WHILE (FRAGMENTS[FI].SEGM[J] <> 0) DO J:=J+1;
  FRAGMENTS[FI].SEGM[J] :=S;
  FRAGMENTS[FI].SITES[J] :=SEGM.SITES;
END; (* ADD_SEGM *)

(* ADD THE INTERSECTION OF SEGMENT AND FRAGMENT SPECIFIED TO THE FRAGMENT TABLE AS FRAGMENT *)
PROCEDURE ADD_INTERSECTION(S,FI:INTEGER);
VAR
  I,J: INTEGER;
BEGIN
  I:=1;
  ...
WHILE (FRAGMENTS[I].RELATION <> 0) DO I:=I+1;
NFRAGS:=I;
J:=1;
WHILE (FRAGMENTS[FI].SEG[J] <> 0) DO
BEGIN
  FRAGMENTS[I].SEG[J]:=FRAGMENTS[FI].SEG[J];
  FRAGMENTS[I].SITE[J]:=FRAGMENTS[FI].SITE[J];
  J:=J+1;
END;
FRAGMENTS[I].SEG:=S;
FRAGMENTS[I].SITE:=SEGMENTS[S].SITE;
FRAGMENTS[I].RELATION:=SEGMENTS[S].RELATION;
FRAGMENTS[I].TUPLES:=FRAGMENTS[FI].TUPLES*SEGMENTS[S].TUPLES;
END; (* ADD_INTERSECTION *)

(* REMOVE THE INTERSECTION OF TUPLES IN SEGMENT AND FRAGMENT FROM THE FRAGMENT *)
PROCEDURE REMOVE_FROM_FRAG(S,FI: INTEGER);
BEGIN
  FRAGMENTS[FI].TUPLES:=FRAGMENTS[FI].TUPLES-(FRAGMENTS[FI].TUPLES*SEGMENTS[S].TUPLES);
END; (* REMOVE_FRAG *)

(* REMOVE THE INTERSECTION OF TUPLES IN SEGMENT AND FRAGMENT FROM THE SEGMENT *)
PROCEDURE REMOVE_FROM_SEG(S,FI: INTEGER);
BEGIN
  SEGMENTS[S].TUPLES:=SEGMENTS[S].TUPLES-(SEGMENTS[S].TUPLES*FRAGMENTS[FI].TUPLES);
END; (* REMOVE_SEG *)

(* CALCULATE SIZE OF FRAGMENTS *)
PROCEDURE SIZE_FRAG;
VAR
  I,J,NT: INTEGER;
BEGIN
  I:=1;
  WHILE (FRAGMENTS[I].RELATION <> 0) DO
  BEGIN
    NT:=0;
    FOR J:=1 TO MAX_TUPLE DO
      IF (J IN FRAGMENTS[I].TUPLES) THEN NT:=NT+1;
    FRAGMENTS[I].SIZE:=NT*REC_SIZE[FRAGMENTS[I].RELATION];
    I:=I+1;
  END;
END; (* SIZE_FRAG *)

PROCEDURE HALT;
BEGIN
    WINDOW(1,1,80,25);
    TEXTBACKGROUND(BLACK);
    CLRSCR;
    HALT;
END;

BEGIN (* MAIN *)
    WINDOW(1,1,80,25);
    TEXTBACKGROUND(BLACK);
    CLRSCR;
    FRAG SCREEN;
    GOTOXY(22,4);
    FRAG_FILENAME := ' ';
    READIN(FRAG_FILENAME);
    IF (FRAG_FILENAME = 'Q') OR (FRAG_FILENAME = 'q') THEN
        HALT;
    TCLRAR;
    GET_SEGMENTS;
    WINDOW(20,15,55,19);
    TEXTBACKGROUND(YELLOW);
    TEXTCOLOR(RED);
    CLRSCR;
    GOTOXY(1,3);
    WRTEM('DETERMINING DISJOINT FRAGMENTS ...');
    NFRAGS:=0;
    S:=1;
    WHILE ( S <= NSEGS ) DO
        BEGIN
            FI:= 0;
            J:=1;
            WHILE ( (FI = 0) AND (J <= NFRAGS) ) DO
                BEGIN
                    IF ( SEGMENTS[S].RELATION = FRAGMENTS[J].RELATION ) THEN FI:= J;
                    J:=J+1;
                    END; (* WHILE *)
                (* IF SEGMENT'S RELATION NOT IN THE FRAGMENTS TABLE THEN ADD IT *)
                IF ( FI = 0 ) THEN
                    ADD_SEGMENT(S)
                ELSE
                    BEGIN
                        NIF:=TRUE;
                        WHILE ( (FI <= NFRAGS) AND NIF ) DO
                            BEGIN
                                J:=1;
                                WHILE ( (J <= MAX_TUPLE ) AND NIF ) DO
                                    BEGIN
                                    END;
                                END;
                        END;
        END;
    END;
IF ( J IN SEGMEN[S].TUPLES) THEN
    IF ( J IN FRAGMEN[T].TUPLES ) THEN NIF:=FALSE;
    J:=J+1
END; (* WHILE J *)
FI:=FI+1
END; (* WHILE FI *)
FI:=FI-1;
(* IF ALL SEGMENT'S TUPLES NOT IN THE FRAGMENTS TABLE THEN ADD IT *)
IF ( NIF ) THEN ADD_SEGMENT(S)
ELSE
    BEGIN
(* IF ALL SEGMENTS TUPLES IN ONE FRAGMENT IN THE FRAGMENTS TABLE*)
    IF ( SEGMEN[S].TUPLES <= FRAGMEN[FI].TUPLES )
    THEN
        BEGIN
            IF ( FRAGMEN[FI].TUPLES <= SEGMEN[S].TUPLES )
                THEN ADD SEGMENT(S,FI)
            ELSE
                BEGIN
                    ADD INTERSECTION(S,FI);
                    REMOVE FROM_FRAG(S,FI)
                END (* IF FRAGMENTS *)
        END
    ELSE
        BEGIN
            IF ( FRAGMEN[FI].TUPLES <= SEGMEN[S].TUPLES )
                THEN
                    BEGIN
                        ADD SEGMENT(S,FI);
                        REMOVE FROM_SEG(S,FI);
                        IF ( SEGMEN[S].TUPLES <> [] ) THEN S:=S-1;
                    END
            ELSE
                BEGIN
                    ADD INTERSECTION(S,FI);
                    REMOVE FROM_FRAG(S,FI);
                    REMOVE FROM_SEG(S,NFRAGS);
                    IF ( SEGMEN[S].TUPLES <> [] ) THEN S:=S-1;
                END; (* IF ALL OF A FRAGMENT'S TUPLES *)
        END; (* IF ALL SEGMENTS TUPLES IN *)
    END; (* IF ALL SEGMEN'S TUPLES IN *)
END; (* IF SEGMENTS RELATION NOT IN *)

S:=S+1;
END; (* WHILE S *)
SIZE_FRAG;
OUT_FRAG;
WINDOW(1,1,80,25);
PROGRAM CLUSTERS;

(* A Generalized Data Distribution Model for Distributed Relational Databases *)
(* Patrick D. Yurk, M.S. *)
(* Western Michigan University *)
(* August 1989 *)
(* *)
(* Program to determine the sites that should be grouped together into clusters. *)
(* The program constructs clusters of sites based upon the costs of space, retrieval, updates and communications. The costs for each site in a cluster must be within a threshold ratio that is entered by the user. *)
(* The communications costs for each site is averaged and the average communications costs for each site is used to determine clusters. *)
(* *)
(* *)
(*CONST*)

MAX_SITES = 8;

(*TYPE*)

OP_CST = ARRAY [1..MAX_SITES, 1..3] OF REAL;
CM_CST = ARRAY [1..MAX_SITES, 1..MAX_SITES] OF REAL;
CLUST_TYPE = ARRAY [1..MAX_SITES, 1..5] OF REAL;
CLUS_CST_TYPE = RECORD
  COSTS : ARRAY [1..3] OF REAL;
  SITES : SET OF 1..MAX_SITES;
END;
VAR

(* COMMON STORAGE AREA USED FOR VARIABLES THAT ARE PASSED BETWEEN MODULES *)
(*
  PDR, (* PROGRAM DRIVE *)
  INDR, (* INPUT FILE DRIVE *)
  OUTDR : STRING[1]; (* OUTPUT FILE DRIVE *)
(*
(* END OF COMMON STORAGE AREA *)
(*

OP_CST : OP_CST; (* ARRAY OF SITE OPERATIONAL COSTS *)
(* ONE ROW IN THE ARRAY FOR EACH SITE *)
(* COL 1 IS COST OF SPACE AT THE SITE *)
(* COL 2 IS COST OF RETRIEVAL AT THE SITE *)
(* COL 3 IS COST OF UPDATE AT THE SITE *)

CM_CST : CM_CST; (* ARRAY OF COMMUNICATION COSTS BETWEEN SITES. ONE ROW AND ONE COLUMN FOR EACH SITE *)
(* SITE. ROW 1 COLUMN 2 IS THE COST OF COMMUNICATION BETWEEN SITES 1 AND 2. *)
(* ROW 2 COLUMN 3 IS THE COST OF COMMUNICATION BETWEEN SITES 2 AND 3. ETC. *)
(* COM_CST[I,J] = 0 WHERE I=J *)
(* COM_CST[I,J] = COM_CST[J,I] *)

CLST_TYPE: CLUST_TYPE; (* ARRAY OF CLUSTER INFORMATION *)
(* ONE ROW IN THE ARRAY FOR EACH SITE *)
(* COL 1 IS CLUSTER NUMBER *)
(* COL 2 IS THE SITE NUMBER *)
(* COL 3 IS COST OF SPACE AT THE SITE *)
(* COL 4 IS COST OF RETRIEVAL AT THE SITE *)
(* COL 5 IS COST OF UPDATE AT THE SITE *)

CLUST_DATA: ARRAY [1..MAX_SITES] OF CLUST_TYPE;
(* ARRAY OF CLUSTER COSTSAND SITES, ONE FOR EACH CLUSTER *)
(* COSTS[1] IS AVERAGE COST OF RETREIVAL *)
(* COSTS[2] IS AVERAGE COST OF UPDATE *)
(* COSTS[3] IS AVERAGE COST OF SPACE *)
(* SITES IS THE SET OF SITES IN THE CLUSTER *)

CLUST_FILENAME,
COST_FILENAME : STRING[15];
IN_STR : STRING[10];
INFILE : TEXT;
EX : FILE;
SPACE THRSH,
UPD_RET THRSH,
LAST CLUST,
MAX CLUST,
COM THRSH : REAL;

I,
SITE,
CLUST,
NUM CLUSTS,
NUM SITES,
RESULT : INTEGER;

OK,
FOUND : BOOLEAN;

(* QUIT TO SYSTEM ******************************************************)
PROCEDURE HALT;
BEGIN (* QUIT *)
   TEXTBACKGROUND(BLACK);
   CLRSR;
   HALT
END; (* QUIT *)

(* LOAD THE SITE DATA FROM THE DISK *************************************************)
PROCEDURE COPY_FROM_SITE;

VAR
   I,
   J : INTEGER;

BEGIN (* COPY FROM SITE *)
   READIN(INFILE, NUM_SITES);
   READIN(INFILE);
   FOR I := 1 TO NUM_SITES DO
      FOR J := 1 TO 3 DO
         READIN(INFILE, OP_COSTS[I,J]);
      READIN(INFILE);
   FOR I := 1 TO NUM_SITES DO
      FOR J := 1 TO NUM_SITES DO
         READIN(INFILE, COM_COSTS[I,J]);
   CLOSE(INFILE)
END; (* COPY FROM SITE *)

(* STORE THE CLUSTER DATA TO THE DISK ******************************************)
PROCEDURE COPY_TO_CLUSTER;

VAR
   C,I : INTEGER;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
BEGIN (* COPY TO CLUSTER *)
  ASSIGN(INFILE, OUTDR+1: 'CLUST_FILENAME.OUT');
  REWRITE(INFILE);
  WRITEIN(INFILE, '***************');
  WRITEIN(INFILE, 'CLUSTER DATA');
  WRITEIN(INFILE, '***************');
  WRITEIN(INFILE);
  WRITEIN(INFILE, 'NUM_CLUSTERS CLUSTERS');
  WRITEIN(INFILE);
  WRITEIN(INFILE, 'COST OF COST OF COST OF CLUSTER');
  WRITEIN(INFILE, 'RETRIEVAL UPDATE SPACE SITES');
  WRITEIN(INFILE, '------------------------------------------');
  FOR C := 1 TO NUM_CLUSTERS DO
    BEGIN
      WRITE(INFILE, CLUSTER_DATA[C], COSTS[1]:7:3,
      CLUSTER_DATA[C], COSTS[2]:10:3,
      CLUSTER_DATA[C], COSTS[3]:11:3);
      FOR I:=1 TO MAX_SITES DO
        IF (I IN CLUSTER_DATA[C].SITES) THEN
          WRITE(INFILE, I:4);
      END;
    END;
  END; (* COPY TO CLUSTER *)

(* INPUT FILE NAMES AND CLUSTER THRESHOLD RATIOS *)
PROCEDURE SCREEN_CLUSTER;
BEGIN (* SCREEN_CLUSTER *)
  CLRSCE;
  TEXTCOLOR(RED);
  WRITEIN('34, 'CLUSTER THRESHOLD RATIOS');
  WRITEIN('34, '-------------------------------');
  WRITEIN('CLUSTER FILE NAME:');
  WRITEIN('COST FILE NAME:');
  WRITEIN('SINGLE SITE CLUSTERS? (Y/N):');
  WRITEIN('COST OF SPACE THRESHOLD RATIO:');
  WRITEIN('COST OF UPDATE/RETRIEVAL THRESHOLD RATIO:');
  WRITEIN('COST OF COMMUNICATION THRESHOLD:');
  (* SHOW THE FUNCTION KEYS ON ROW 24 *)
  GOTOXY(1, 24);
  WRITE('Q => QUIT TO SYSTEM');
  TEXTCOLOR(YELLOW);
END; (* SCREEN_CLUSTER *)

(* CALCULATE CLUSTER DATA *)
PROCEDURE CALC_CLUSTER_DATA;
VAR
   C, I, N, V : INTEGER;
   TOTCR, TOTCU, TOTCSP, CS : REAL;
BEGIN (* CALC_CLUSTER_DATA *)
   NUM_CLUSTS := 0;
   FOR C := 1 TO NUM_SITES DO
      BEGIN
         TOTCR := 0;
         TOTCU := 0;
         TOTCSP := 0;
         N := 0;
         CS := 0;
         CLUSTER_DATA[C].SITES := [];
         FOR I := 1 TO NUM_SITES DO
            BEGIN
               IF ( C = CLUST_SITE[I,1] ) THEN
                  BEGIN
                     TOTCR := TOTCR + CLUST_SITE[I,4];
                     TOTCU := TOTCU + CLUST_SITE[I,5];
                     TOTCSP := TOTCSP + CLUST_SITE[I,3];
                     V := TRUNC(CLUST_SITE[I,2]);
                     CLUSTER_DATA[C].SITES := CLUSTER_DATA[C].SITES + [V];
                     N := N + 1;
                  END;
            END; (* FOR I *)
         IF ( N = 0 ) THEN
            BEGIN
               CLUSTER_DATA[C].COSTS[1] := 0;
               CLUSTER_DATA[C].COSTS[2] := 0;
               CLUSTER_DATA[C].COSTS[3] := 0;
            END ELSE
            BEGIN
               CLUSTER_DATA[C].COSTS[1] := TOTCR / N;
               NUM_CLUSTS := NUM_CLUSTS + 1;
            END;
      END; (* FOR C *)
END; (* CALC_CLUSTER_COSTS *)

(* ZERO OUT THE CLUST_SITE ARRAY *)
PROCEDURE ZERO_CLUSTERS;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
VAR
    I, J : INTEGER;
BEGIN (* ZERO_CLUSTERS *)
    FOR I := 1 TO NUM_SITES DO
        FOR J := 1 TO 5 DO
            CLUST_SITE[I, J] := 0;
        END; (* ZERO_CLUSTERS *)
    END;

(* ADD A SITE TO THE CLUSTER *)
PROCEDURE ADD_SITE;
VAR
    I : INTEGER;
    FOUND : BOOLEAN;
BEGIN (* ADD_SITE *)
    I := 1;
    FOUND := FALSE;
    (* LOOK FOR AVAILABLE CLUSTER SLOT *)
    WHILE (I <= NUM_SITES) AND (NOT FOUND) DO
        BEGIN
            IF (CLUST_SITE[I, 1] = 0) THEN (* FOUND AN EMPTY SLOT *)
                BEGIN
                    CLUST_SITE[I, 1] := LAST_CLUST; (* ADD SITE TO THE CLUSTER *)
                    CLUST_SITE[I, 2] := SITE;
                    CLUST_SITE[I, 3] := OP_COSTS[SITE, 1];
                    CLUST_SITE[I, 4] := OP_COSTS[SITE, 2];
                    CLUST_SITE[I, 5] := OP_COSTS[SITE, 3];
                    FOUND := TRUE;
                END;
                I := I + 1;
                END (* WHILE *)
        END; (* ADD_SITE *)

(* CHECK COMMUNICATIONS COST OF POTENTIAL NEW SITE FOR THE CLUSTER *)
FUNCTION CHECK_CC(SITE: INTEGER; CLUST: REAL): BOOLEAN;
VAR
    CHECK : BOOLEAN;
    I, J : INTEGER;
BEGIN (* CHECK_CC *)
    CHECK := TRUE;
    I := 1;
    WHILE (I <= NUM_SITES) AND (CLUST_SITE[I, 1] <> 0) AND (CHECK = TRUE) DO
        BEGIN
            IF (CLUST_SITE[I, 1] = CLUST) THEN
                BEGIN
                    END; (* CHECK_CC *)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\( J := \text{TRUNC(CLUST\_SITE}[I,2]); \)
\( \text{IF} \ (\text{COM\_COSTS}[\text{SITE}, J] > \text{COM\_THRESH}) \ \text{THEN} \)
\( \quad \text{CHECK} := \text{FALSE}; \)
\( \quad \text{END;} \quad (* \text{IF} \ \text{CLUST\_SITE} *) \)
\( I := I + 1; \)
\( \text{END;} \quad (* \text{WHILE} *) \)

\( \text{CHECK} := \text{CHECK}; \)
\( \text{END;} \quad (* \text{CHECK\_CC} *) \)

(* USER WISHES TO USE SINGLE SITE CLUSTERS *)
PROCEDURE SINGLE\_SITE\_CLUSTERS;

VAR
\( I : \text{INTEGER}; \)

BEGIN (* SINGLE\_SITE\_CLUSTERS *)

\( \text{ZERO} \ \text{CLUSTERS}; \)
\( \text{FOR} \ I := 1 \ \text{TO} \ \text{NUM\_SITES} \ \text{DO} \)
\( \text{BEGIN} \quad (* \text{ADD} \ \text{SITE} \ \text{TO} \ \text{THE} \ \text{CLUSTER} *) \)
\( \quad \text{CLUST\_SITE}[I, 1] := I; \)
\( \quad \text{CLUST\_SITE}[I, 2] := I; \)
\( \quad \text{CLUST\_SITE}[I, 3] := \text{OP\_COSTS}[I, 1]; \)
\( \quad \text{CLUST\_SITE}[I, 4] := \text{OP\_COSTS}[I, 2]; \)
\( \quad \text{CLUST\_SITE}[I, 5] := \text{OP\_COSTS}[I, 3]; \)
\( \text{END;} \quad (* \ \text{SINGLE\_SITE\_CLUSTERS} *) \)

BEGIN (* MAIN *)

WINDOW(1, 1, 80, 25);
TEXTBACKGROUND(BLACK);
CLRSRC;
SCREEN\_CLUSTER;
GO\_TO\_XY(20, 4);
\( \text{CLUST\_FILENAME} := \text{'}Q\text{'}; \)
READ\_IN(CLUST\_FILENAME);
\( \text{IF} \ (\text{CLUST\_FILENAME} = \text{'}Q\text{'}) \ \text{OR} \ (\text{CLUST\_FILENAME} = \text{'}q\text{'}) \ \text{THEN} \)
HALT;

\( \text{OK} := \text{FALSE}; \)
\( \text{WHILE} \ (\text{NOT} \ \text{OK}) \ \text{DO} \)
\( \text{BEGIN} \quad \text{GO}\_TO\_XY(20, 5);
\( \quad \text{COST\_FILENAME} := \text{'}Q\text{'}; \)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
READLN(COST_FILENAME);
IF (COST_FILENAME = 'Q') OR (COST_FILENAME = 'q') THEN HALT;
ASSIGN(INFILE, INDR + '!' + COST_FILENAME+'.'IN');
{$I-} RESET(INFILE) {$I+};
OK := (IORESULT = 0);
IF (NOT OK) THEN
BEGIN
  GOTOXY(10,23);
  TEXTCOLOR(WHITE);
  WRITE(INDR4+':'+COST_FILENAME+'.'IN, DOES NOT EXIST, TRY AGAIN OR TYPE QUIT
END
GOTOXY(10,23); CLEAR;
COPY_FROM_SITE;

GOTOXY(30,7);
IN_STR:='';
READIN(IN_STR);
IF (IN_STR = 'Q') OR (IN_STR = 'q') THEN HALT;
IF (NOT (IN_STR = 'Y') OR (IN_STR = 'y')) THEN
BEGIN
  RESULT:=99;
  WHILE (RESULT <> 0) DO
  BEGIN
    GOTOXY(43,9);
    IN_STR := '';
    READIN(IN_STR);
    IF (IN_STR = 'Q') OR (IN_STR = 'q') THEN HALIT;
    VAL(IN_STR,SPACE_THRSH,RESULT);
    IF ((RESULT <> 0) OR (SPACE_THRSH >= 1 ))THEN
    BEGIN
      GOTOXY(10,23);
      TEXTCOLOR(WHITE);
      WRITE('INVALID RATIO. MUST BE A REAL NUMBER < 0. TRY AGAIN OR TYPE QUIT
    END;
  END;
  RESULT:=99;
  WHILE (RESULT <> 0) DO
  BEGIN
    GOTOXY(43,11);
    IN_STR := '';
    READIN(IN_STR);
    IF (IN_STR = 'Q') OR (IN_STR = 'q') THEN HALIT;
    VAL(IN_STR,UPD_RET_THRSH,RESULT);
    IF ((RESULT <> 0) OR (UPD_RET_THRSH >= 1 ))THEN
BEGIN
    GOTOXY(10,23);
    TEXTCOLOR(WHITE);
    WRITE('INVALID RATIO. MUST BE A REAL NUMBER < 0. TRY AGAIN OR TYPE QUIT')
END;
GOTOXY(10,23); CLRBL;

RESULT:=99;
WHILE (RESULT <> 0) DO
BEGIN
    GOTOXY(43,13);
    IN_STR := ' '; READLN(IN_STR);
    IF (IN_STR = 'Q') OR (IN_STR = 'q') THEN HALT;
    VAL(IN_STR,OM_THRSH,RESULT);
    IF (RESULT <> 0) THEN BEGIN
        GOTOXY(10,23);
        TEXTCOLOR(WHITE);
        WRITE('INVALID ENTRY. MUST BE A REAL NUMBER. TRY AGAIN OR TYPE QUIT')
    END END;
GOTOXY(10,23); CLRBL;

WINDOW(20,15,50,19);
TEXTBACKGROUND(GREEN);
TEXTCOLOR(WHITE);
CLRSCR;
GOTOXY(1,3);
WRITE('CLUSTERS BEING DETERMINED...');

ZERO_CLUSTERS;
FOR SITE:=1 TO NUM_SITES DO (* DETERMINE THE CLUSTER IN WHICH TO PUT EACH S *)
BEGIN
    MAX_CLUST:=0;
    LAST_CLUST:=0;
    (* SCAN THROUGH EXISTING CLUSTERS *)
    CLUST:=1;
    FOUND:=FALSE;
    WHILE (CLUST <= NUM_SITES) AND (NOT FOUND) DO
    BEGIN
        IF (CLUSTSITE(CLUST,1) = 0) THEN (* IF ALL CLUSTERS SCANNED *) BEGIN
            CLUSTSITE(CLUST,1):=MAX_ClUST+1; (* ADD A NEW CLUSTER *)
            CLUSTSITE(CLUST,2):=SITE;
            CLUSTSITE(CLUST,3):=OP_COSTS[SITE,1];
            CLUSTSITE(CLUST,4):=OP_COSTS[SITE,2];
            CLUSTSITE(CLUST,5):=OP_COSTS[SITE,3];
        END;
FOUND:=TRUE;
END
ELSE (* CHECK THRESHOLDS ON THIS CLUSTER *)
BEGIN
IF LAST_CLUST < CLUST_SITE[CLUST,1] (* NEXT CLUSTER TO CHECK? *)
THEN
BEGIN
(* SPACE COSTS WITHIN THRESHOLD LIMITS? *)
LAST_CLUST:=CLUST_SITE[CLUST,1];
IF (MAX_CLUST<LAST_CLUST) THEN MAX_CLUST:=LAST_CLUST;
IF ( (OP_costs[SITE,1]<=CLUST_SITE[CLUST,3] +
SPACE_THRESH*CLUST_SITE[CLUST,3]) AND
(UPD_RET_THRESH*CLUST_SITE[CLUST,3]) )
THEN
(* RETRIEVAL COSTS WITHIN THRESHOLD LIMITS? *)
IF ( (OP_costs[SITE,2]<=CLUST_SITE[CLUST,4] +
UPD_RET_THRESH*CLUST_SITE[CLUST,4]) ) AND
(UPD_RET_THRESH*CLUST_SITE[CLUST,4])
THEN
(* UPDATE COSTS WITHIN THRESHOLD LIMITS? *)
IF ( (OP_costs[SITE,3]<=CLUST_SITE[CLUST,5] +
UPD_RET_THRESH*CLUST_SITE[CLUST,5]) ) AND
(UPD_RET_THRESH*CLUST_SITE[CLUST,5])
THEN
(* COMMUNICATION COSTS WITHIN THRESHOLD LIMITS? *)
IF (CHECK_CC(SITE,LAST_CLUST))
BEGIN
(* ADD SITE TO THIS CLUSTER *)
ADD_SITE;
FOUND:=TRUE;
END
END; (* ELSE *)
CLUST:=CLUST+1;
END (* while CLUSTER *)
END; (* FOR SITE *)
END (* IF NOT IN STR *)
ELSE
SINGLE_SITE_CLUSTERS;
CALC_CLUSTER_DATA;
COPY_TO_CLUSTER;
WINDOW(1,1,80,25);
ASSIGN(EX,PDR+':MAINMENU.COM');
EXECUTE(EX)
Program to perform the first phase of the allocation algorithm for clusters.

The program initially allocates fragments to all clusters that use the fragment. The potential benefit of allocation of the fragment to the cluster is calculated. The cluster with the potential benefit of allocation for the fragment is then marked as having the fragment permanently allocated.

CONST MAX_SITES=10;
MAX_REL=4;
MAX_FRAG=30;
MAX_APPLICATION=30;
MAX_REC_SIZE=50;
MAX_TUPLE=20;
MAX_FIELD_LENGTH=12;
MAX_ALLOC=300; (* MAX_SITES*MAX_FRAG *)

TYPE
FRAGTYPE = RECORD
  SEGM: ARRAY [1..MAX_APPLICATION] OF INTEGER;
  SITES: ARRAY [1..MAX_APPLICATION] OF INTEGER;
  RELATION: INTEGER;
  SIZE: INTEGER;
  TUPLES: SET OF 1..MAX_TUPLE;
END;

SEGTYPE = RECORD
  SITES: INTEGER;
END;
RELATION: INTEGER;
FREQR: REAL;
FREQU: REAL;
TUPLES: SET OF 1..MAX_TUPLE;
END;

ALLOC_TYPE = RECORD
  FRAGMENT: INTEGER;
  CLUSTER: INTEGER;
  PERM: CHAR;
  PERC: REAL;
  FREQR: REAL;
  FREQU: REAL;
  CS: INTEGER;
END;

CLSCST_TYPE = RECORD
  COSTS: ARRAY [1..3] OF REAL;
  SITES: SET OF 1..MAX_SITES;
END;

OP_CST = ARRAY [1..MAX_SITES, 1..3] OF REAL;
CM_CST = ARRAY [1..MAX_SITES, 1..MAX_SITES] OF REAL;
CLUST_TYPE = ARRAY [1..MAX_SITES, 1..6] OF REAL;

VAR

(* COMMON STORAGE AREA USED FOR VARIABLES THAT ARE PASSED BETWEEN MODULES *)
(* *)
  PDR,  (* PROGRAM DRIVE *)
  INDR,  (* INPUT FILE DRIVE *)
  OUTDR : STRING[1];  (* OUTPUT FILE DRIVE *)
(* *)
(* END OF COMMON STORAGE AREA *)
(* **********************************************)

FRAGMENTS: ARRAY [1..MAX_FRAG] OF FRAGTYPE;
SEGMENTS: ARRAY [1..MAX_APPLICATION] OF SEGTYPE;
ALLOCATION: ARRAY [1..MAX_ALLOC] OF ALLOCATE;

FRAG_FILENAME : STRING[15];
FRAGMENT_FILE  (* FILE OF THE FRAGMENTS *) : TEXT;
EX : FILE;
NFRAGS,
S,I,J,C,
K, CF,
NSEGS,
NUM_ALLOC,
NUM_CLUST,
FI : INTEGER;
NIF : BOOLEAN;

OP_COSTS : OP_CST; (* ARRAY OF SITE OPERATIONAL COSTS *)
(* ONE ROW IN THE ARRAY FOR EACH SITE *)
(* COL 1 IS COST OF SPACE AT THE SITE *)
(* COL 2 IS COST OF RETRIEVAL AT THE SITE *)
(* COL 3 IS COST OF UPDATE AT THE SITE *)

COM_COSTS : CM_CST; (* ARRAY OF COMMUNICATION COSTS BETWEEN *)
(* SITES. ONE ROW AND ONE COLUMN FOR EACH SITE *)
(* SITES. ROW 1 COLUMN 2 IS THE COST OF *)
(* COMMUNICATION BETWEEN SITES 1 AND 2. *)
(* ROW 2 COLUMN 3 IS THE COST OF COMMUNICATION *)
(* BETWEEN SITES 2 AND 3. ETC... *)
(* COM_COSTS[I,J] = 0 WHERE I=J *)
(* COM_COSTS[I,J] = COM_COSTS[J,I] *)

CLUST_SITE: CLUST_TYPE; (* ARRAY OF CLUSTER INFORMATION *)
(* ONE ROW IN THE ARRAY FOR EACH SITE *)
(* COL 1 IS CLUSTER NUMBER *)
(* COL 2 IS THE SITE NUMBER *)
(* COL 3 IS COST OF SPACE AT THE SITE *)
(* COL 4 IS COST OF RETRIEVAL AT THE SITE *)
(* COL 5 IS COST OF UPDATE AT THE SITE *)
(* COL 6 IS AVERAGE COST OF COMMUNICATION *)
(* FOR THE SITE *)

CLUSTER_DATA: ARRAY [1..MAX_SITES] OF CLCST_TYPE;
(* ARRAY OF CLUSTER COSTS AND SITES, ONE *)
(* FOR EACH CLUSTER *)
(* COSTS[1] IS AVERAGE COST OF RETRIEVAL *)
(* COSTS[2] IS AVERAGE COST OF UPDATE *)
(* COSTS[3] IS AVERAGE COST OF SPACE *)
(* SITES IS THE SET OF SITES IN THE CLUSTER *)

CLUST_FILENAME,
COST_FILENAME,
ALLOC_FILENAME : STRING[15];

INSTR : STRING[10];
INFILE : TEXT;
TOT_FREQ : ARRAY[1..MAX_FRAG] OF REAL;
MXFRQ : ARRAY[1..MAX_ALLOC] OF REAL;
MXFBI : ARRAY[1..2] OF REAL;

SITE,
CLUST,
NUM_SITES,
RESULT : INTEGER;

OK,
FOUND : BOOLEAN;

(* QUIT TO SYSTEM ******************************************************)
PROCEDURE HALT;
BEGIN (* QUIT *)
  TEXTBACKGROUND(BLACK);
  CLRSCR;
  HALT
END; (* QUIT *)

(* THIS PROCEDURE CLEARS THE FRAGMENTS AND SEGMENTS TABLES *)
(* *****************************************************************)
PROCEDURE TCLEAR;

VAR I, J : INTEGER;

BEGIN
  FOR I := 1 TO MAX_FRAG DO
    BEGIN
      FOR J := 1 TO MAX_APPLICATION DO
        BEGIN
          FRAGMENTS[I].SEGM[J] := 0;
          FRAGMENTS[I].SITES[J] := 0;
          END;
          FRAGMENTS[I].TUPLES := [];
          FRAGMENTS[I].RELATION := 0;
          FRAGMENTS[I].SIZE := 0;
          END;
          FOR I := 1 TO MAX_APPLICATION DO
            BEGIN
              SEGMENTS[I].TUPLES := [];
              SEGMENTS[I].SITES := 0;
              SEGMENTS[I].RELATION := 0;
              END;
PROCEDURE OOPy_ERCMITE;
VAR
  I, J: INTEGER;
  INF1: TEXT;
BEGIN (* COpy ERCM SITE *)
  ASSIGN(INF1, INDRf' : 1 +COST_FILENAME+' .IN');
  RESET(INF1);
  READIN(INF1); 
  FOR I := 1 TO NUM_SITES DO
    FOR J := 1 TO 3 DO
      READLN(INF1, OP COSTS[I,J]);
    READIN(INF1);
  FOR I := 1 TO NUM_SITES DO
    FOR J := 1 TO NUM_SITES DO
      READIN(INF1, COM_COSTS[I,J]);
  CLOSE(INF1)
END; (* COpy ERCM SITE *)

PROCEDURE OOPy_ERCM_CLUSTER;
VAR
  I, C, J: INTEGER;
  R : REAL;
  INF1: TEXT;
BEGIN (* COpy ERCM CLUSTER *)
  ASSIGN(INF1, OUTDRf1: ' +CLUSTFILENAME+' .OUT');
  RESET(INF1);
  READIN(INF1);
  READIN(INF1);
  READLN(INF1);
  READ(INF1, NUMjCLUST);
  FOR C := 1 TO NUMjCLUST DO
    BEGIN
      CLUSTER_DATA[C].COSTS[1]:0;
      CLUSTER_DATA[C].COSTS[2]:0;
      CLUSTER_DATA[C].COSTS[3]:0;
    END;
END; (* COpy ERCM CLUSTER *)

(* LOAD THE SITE DATA FROM THE DISK ************************************)
PROCEDURE COPY_FROM_SITE;
VAR
  I, J: INTEGER;
  INF1: TEXT;
BEGIN (* COPY FROM SITE *)
  ASSIGN(INF1, INDR+ ' : +COST_FILENAME+' .IN');
  RESET(INF1);
  READIN(INF1);
  READIN(INF1);
  READLN(INF1);
  FOR I := 1 TO NUM_SITES DO
    FOR J := 1 TO 3 DO
      READLN(INF1, OP COSTS[I,J]);
    READIN(INF1);
  FOR I := 1 TO NUM_SITES DO
    FOR J := 1 TO NUM_SITES DO
      READIN(INF1, COM_COSTS[I,J]);
  CLOSE(INF1)
END; (* COPY FROM SITE *)

(* LOAD THE CLUSTER DATA FROM THE DISK ************************************)
PROCEDURE COPY_FROM_CLUSTER;
VAR
  I, C, J: INTEGER;
  R : REAL;
  INF1: TEXT;
BEGIN (* COPY FROM CLUSTER *)
  ASSIGN(INF1, OUTDR+ ' : +CLUST_FILENAME+' .OUT');
  RESET(INF1);
  READIN(INF1);
  READIN(INF1);
  READLN(INF1);
  READ(INF1, NUMjCLUST);
  FOR C := 1 TO NUMjCLUST DO
    BEGIN
      CLUSTER_DATA[C].COSTS[1]:0;
      CLUSTER_DATA[C].COSTS[2]:0;
      CLUSTER_DATA[C].COSTS[3]:0;
    END;
END; (* COPY FROM CLUSTER *)
CLUSTER_DATA[C].SITES:=[ ];
END;
READ(INFL2);
READ(INFL2);
READ(INFL2);
READ(INFL2);
READ(INFL2);
FOR C := 1 TO NUM_CIUST DO
BEGIN
READ(INFL2,CLUSTER_DATA[C].COSTS[1],
CLUSTER_DATA[C].COSTS[2],
CLUSTER_DATA[C].COSTS[3] );
WHILE (NOT EOF(INFL2)) DO
BEGIN
READ(INFL2,I);
CLUSTER_DATA[C].SITES:=CLUSTER_DATA[C].SITES+[I];
END;
READ(INFL2);
END;
CLOSE(INFL2);
END; (* COPY FROM CLUSTER *)

(*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*)

PROCEDURE COPY_FROM_FRAGMENT;

VAR
C,I,J,K : INTEGER;
INFL3: TEXT;
BEGIN
TCLEAR;
ASSIGN(INFL3, OUTDR+: ' + FRAG_FILENAME+: .OUT');
RESET(INFL3);
READ(INFL3);
READ(INFL3);
READ(INFL3);
READ(INFL3);
READ(INFL3);
READ(INFL3, NFRAGS);
READ(INFL3);
READ(INFL3);
READ(INFL3);
READ(INFL3);
READ(INFL3);
READ(INFL3);
FOR I := 1 TO NFRAGS DO
BEGIN
FOR J:=1 TO 7 DO
READ(INFL3,FRAGMENTS[I].SEGM[J]);
READ(INFL3,FRAGMENTS[I].RELATION);
READ(INFL3, FRAGMENTS[I].SIZE);
J:=1;
WHILE (NOT EOIN(INFL3) ) DO
BEGIN
  READ(INFL3,K);
  FRAGMENTS[I].TUPLES:=FRAGMENTS[I].TUPLES+[K];
  J:=J+1;
END;
READLN(INFL3);
FOR J:=1 TO 7 DO
  READ(INFL3,FRAGMENTS[I].SITES[J]);
READ(INFL3);
END;
READ(INFL3);
READ(INFL3);
READ(INFL3);
READ(INFL3);
READ(INFL3);
READ(INFL3);
READ(INFL3); FOR I:=1 TO NSEGS DO
BEGIN
  READ(INFL3,SEGMENTS[I].SITES);
  READ(INFL3,SEGMENTS[I].RELATION);
  READ(INFL3,SEGMENTS[I].FREQ);
  READ(INFL3,SEGMENTS[I].FREQU);
  J:=1;
  WHILE (NOT EOIN(INFL3) ) DO
  BEGIN
    READ(INFL3,K);
    SEGMENTS[I].TUPLES:=SEGMENTS[I].TUPLES+[K];
    J:=J+1
  END;
  READLN(INFL3);
END; (* COPY_FROM_FRAGMENT *)

(* COPY ALLOCATION DATA TO THE DISK *****************************************)
PROCEDURE COPY_TO_ALLOC;

VAR
I, J : INTEGER;
OUTFILE: TEXT;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
BEGIN (* COPY TO ALLOC *)
  ASSIGN(OUTFILE, OUTDRH: 'ALLOC.FILENAME: ' .OUT');
  REWRITE(OUTFILE);
  WRITELN(OUTFILE, 'INITIAL ALLOCATION DATA');
  WRITELN(OUTFILE, 'ALLOCATIONS OF FRAGMENTS');
  WRITELN(OUTFILE);
  WRITELN(OUTFILE, 'FRAGMENT CLUSTER PBM FREQR FREQU C_SITE');
FOR I := 1 TO NUM_ALLOC DO
  BEGIN
    WRITE(OUTFILE, ALLOCATION[I].FRAGMENT:5);
    WRITE(OUTFILE, ALLOCATION[I].CLUSTER:10);
    WRITE(OUTFILE, ALLOCATION[I].PERM:8);
    WRITE(OUTFILE, ALLOCATION[I].PBI:12:2);
    WRITE(OUTFILE, ALLOCATION[I].FREQR:8:2);
    WRITE(OUTFILE, ALLOCATION[I].FREQU:8:2);
    WRITE(OUTFILE, ALLOCATION[I].CS:8);
    WRITELN(OUTFILE);
  END;
CLOSE(OUTFILE);
END; (* COPY TO ALLOC *)

(* INPUT FILE NAMES *)
PROCEDURE SCREEN_PHASE_I;
BEGIN (* SCREEN_PHASE_I *)
  CLRSCR;
  TEXTCOLOR(RED);
  WRITELN(' :36, 'ALLOCATION - PHASE I.');
  WRITELN(' :36, '---------------');
  WRITELN('ALLOCATION FILE NAME:');
  WRITELN('CLUSTER FILE NAME:');
  WRITELN('FRAGMENT FILE NAME:');
  WRITELN('COST FILE NAME:');
  (* SHOW THE FUNCTION KEYS ON ROW 24 *)
  GOTOXY(1, 24);
  WRITE('Q => QUIT TO SYSTEM');
  TEXTCOLOR(YELLOW);
END; (* SCREEN_PHASE_I *)

BEGIN (* MAIN *)
  WINDOW(1,1,80,25);
OK := FALSE;
WHILE (NOT OK) DO
BEGIN
GOTOXY(23,6);
CLUST_FILENAME := ' ';
READLN(CLUST_FILENAME);
IF (CLUST_FILENAME = 'Q') OR (CLUST_FILENAME = 'q') THEN
HALT;
ASSIGN(INFILE, OUTDR + ':' + CLUST_FILENAME + '.OUT');
($1-) RESET(INFILE) ('$1+');
OK := (IRESULT = 0);
IF (NOT OK) THEN
BEGIN
GOTOXY(10,23);
TEXTCOLOR(WHITE);
WRITE(OUTDR + ':+CLUST_FILENAME+.OUT, DOES NOT EXIST, TRY AGAIN OR TYPE Q
END
END;
CLOSE(INFILE);

OK := FALSE;
WHILE (NOT OK) DO
BEGIN
GOTOXY(23,8);
FRAG_FILENAME := ' ';
READLN(FRAG_FILENAME);
IF (FRAG_FILENAME = 'Q') OR (FRAG_FILENAME = 'q') THEN
HALT;
ASSIGN(INFILE, OUTDR + ':' + FRAG_FILENAME + '.OUT');
($1-) RESET(INFILE) ('$1+');
OK := (IRESULT = 0);
IF (NOT OK) THEN
BEGIN
GOTOXY(10,23);
TEXTCOLOR(WHITE);
WRITE(INDR + ':+FRAG_FILENAME+.IN, DOES NOT EXIST, TRY AGAIN OR TYPE Q
END
END;
CLOSE(INFILE);

OK := FALSE;
WHILE (NOT OK) DO
BEGIN
GOTOXY(23,10);
COST_FILENAME := '';
READIN(COST_FILENAME);
IF (COST_FILENAME = 'Q') OR (COST_FILENAME = 'q') THEN
HALT;
ASSIGN(INFILE, INDR + ':' + COST_FILENAME + '.IN');
($1$) RESET(INFILE) ($1$);
OK := (IRESULT = 0);
IF (NOT OK) THEN
BEGIN
  GOTOXY(10,23);
  TEXTCOLOR(WHITE);
  WRITE(INDR+':' + COST_FILENAME + '.IN, DOES NOT EXIST, TRY AGAIN OR TYPE QUIT
END
CLOSE(INFILE);

GOTOXY(10,23); CLEAR;
WINDOW(13,9,53,13);
TEXTBACKGROUND(WHITE);
TEXTCOLOR(GREEN);
CLRSRC;
GOTOXY(1,3);
WRITE('READ SITES, CLUSTERS, AND FRAGMENTS ...');
COPY_FROM_SITE;
COPY_FROM_CLUSTER;
COPY_FROM_FRAGMENT;

WINDOW(15,11,55,15);
TEXTBACKGROUND(GREEN);
TEXTCOLOR(WHITE);
CLRSRC;
GOTOXY(1,3);
WRITE('ALLOCATING FRAGMENTS TO ALL CLUSTERS ...');
DELAY(1000);
NUM_ALLOC := 0;
FOR I := 1 TO NFRAGS DO
BEGIN
  J := 1;
  WHILE (FRAGMENTS[I].SITES[J] <> 0) DO
  BEGIN
    C := 0;
    K := 1;
    WHILE (C = 0) DO
    BEGIN
      IF (FRAGMENTS[I].SITES[J] IN CLUSTER_DATA[K].SITES) THEN C := K;
      K := K + 1;
    END;
    FOUND := FALSE;
    K := 1;
    WHILE (K <= NUM_ALLOC) AND (NOT FOUND) DO
    BEGIN

IF ( (I = ALLOCATION[K].FRAGMENT) AND (C = ALLOCATION[K].CLUSTER) ) THEN
  BEGIN
    FOUND:=TRUE;
    ALLOCATION[K].FREQR:=ALLOCATION[K].FREQR
    + SEGMENTS[FRAGMENTS[I].SEGMENT[J]].FREQR;
    ALLOCATION[K].FREQU:=ALLOCATION[K].FREQU
    + SEGMENTS[FRAGMENTS[I].SEGMENT[J]].FREQU;
    K:=K-1;
  END;
K:=K+1;
END; (* WHILE *)
IF ( NOT FOUND ) THEN
  BEGIN
    NUM_ALLOC:=NUM_ALLOC+1;
    K:=NUM_ALLOC;
    ALLOCATION[K].FRAGMENT:=I;
    ALLOCATION[K].CLUSTER:=C;
    ALLOCATION[K].PERM='T';
    ALLOCATION[K].PBI:=0;
    ALLOCATION[K].FREQR:=SEGMENTS[FRAGMENTS[I].SEGMENT[J]].FREQR;
    ALLOCATION[K].FREQU:=SEGMENTS[FRAGMENTS[I].SEGMENT[J]].FREQU;
  END;
TOT_FREQU[I]:=TOT_FREQU[I] + SEGMENTS[FRAGMENTS[I].SEGMENT[J]].FREQU;
IF ( MXFREQ[((I-1)*NUM_CLUSTER)+C] < SEGMENTS[FRAGMENTS[I].SEGMENT[J]].FREQR ) THEN
  BEGIN
    MXFREQ[((I-1)*NUM_CLUSTER)+C]:=SEGMENTS[FRAGMENTS[I].SEGMENT[J]].FREQR;
    ALLOCATION[K].CS:=FRAGMENTS[I].SITES[J];
  END;
J:=J+1;
END; (* WHILE FRAGMENT *)
END; (* FOR I *)

WINDOW(17,13,57,17);
TEXTBACKGROUND(WHITE);
TEXTCOLOR(GREEN);
CLRSCR;
GOTOXY(1,3);
WRITE('CALCULATING POTENTIAL BENEFIT (PBI) ...');
DELAY(1000);
CF:=1;
MXPBI[1]:=0;
MXPBI[2]:=-32766;
FOR I:= 1 TO NUM_ALLOC DO
  BEGIN
    ALLOCATION[I].PBI:= ALLOCATION[I].FREQR
    - (TOT_FREQU[ALLOCATION[I].FRAGMENT] - ALLOCATION[I].FREQ

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
IF (CF = ALLOCATION[I].FRAGMENT )
    THEN
        BEGIN
            IF (MXPBI[2] < ALLOCATION[I].PBI )
                THEN
                    BEGIN
                        MXPBI[1]:=I;
                        MXPBI[2]:=ALLOCATION[I].PBI;
                    END;
                ELSE
                    BEGIN
                        ALLOCATION[TRUNC(MXPBI[1])].PERM:='P';
                        MXPBI[1]:=I;
                        MXPBI[2]:=ALLOCATION[I].PBI;
                        CF:=ALLOCATION[I].FRAGMENT;
                    END;
            END; (* FOR *)
        END;
    END;

ALLOCATION[TRUNC(MXPBI[1])].PERM:='P';

WINDOW(19,15,59,19);
TEXTBACKGROUND(GREEN);
TEXTCOLOR(WHITE);
CLRSCR;
GOTOXY(1,3);
WRITE('OUTPUT RESULTS TO ALLOCATION FILE ...');
COPY_TO_ALLOC;

WINDOW(1,1,80,25);
ASSIGN(EX,PDR+'\MAINMENU.COM');
EXECUTE(EX)

END.
Program to perform the second phase of the allocation algorithm for clusters.

The program calculates the cost of not allocating the fragment to the cluster and the benefit of allocating the fragment to the cluster. Allocation is cancelled for those fragments with benefit < 0 for the cluster. Otherwise, the fragment is permanently allocated to the cluster. Lastly, allocation of fragments to sites within each cluster is determined.

CONST MAX_SITES=10;
MAX_REL=4;
MAX_FRAG=30;
MAX_APPLICATION=30;
MAX_REC_SIZE=50;
MAX_TUPLE=20;
MAX_FIELD.LENGTH=12;
MAX_ALLOC=300; (*MAX_SITES*MAX_FRAG*)

TYPE
   FRAGTYPE = RECORD
      SEGMENT: ARRAY [1..MAX_APPLICATION] OF INTEGER;
      SITES: ARRAY [1..MAX_APPLICATION] OF INTEGER;
      RELATION: INTEGER;
      SIZE: INTEGER;
      TUPLES: SET OF 1..MAX_TUPLE;
   END;

   SECTYPE = RECORD
      SITES: INTEGER;
      RELATION: INTEGER;
      FREQR: REAL;
      FREQU: REAL;
      TUPLES: SET OF 1..MAX_TUPLE;
   END;

   ALLOC.TYPE = RECORD

FRAGMENT: INTEGER;
CLUSTER: INTEGER;
PDM: CHAR;
FBI: REAL;
FREQR: REAL;
FREQU: REAL;
CS: INTEGER;
END;

BENTYPE = RECORD
FRAGMENT: INTEGER;
CLUSTER: INTEGER;
CA: REAL;
R_CLUSTER: INTEGER;
RCN: REAL;
RB: REAL;
END;

ALLOC_SITE_TYPE = RECORD
FRAGMENT: INTEGER;
SITE: INTEGER;
ALLOC: BOOLEAN;
CA: REAL;
CN: REAL;
B: REAL;
END;

CLSCST_TYPE = RECORD
COSTS: ARRAY [1..3] OF REAL;
SITES: SET OF 1..MAX_SITES;
END;

SITE_SET_TYPE = SET OF 1..MAX_SITES;

OP_CST = ARRAY [1..MAX_SITES, 1..3] OF REAL;
CM_CST = ARRAY [1..MAX_SITES, 1..MAX_SITES] OF REAL;
CLUST_TYPE = ARRAY [1..MAX_SITES, 1..6] OF REAL;

VAR
(* COMMON STORAGE AREA USED FOR VARIABLES THAT ARE PASSED BETWEEN MODULES *)
(* * *)
(* *)
(* PDR, *)
(* INDR, *)
(* OUTDR : STRING[1]; *)
(* *)
(* END OF COMMON STORAGE AREA *)
(* * *)

FRAGMENTS: ARRAY[1..MAX_FRAG] OF FRAGTYPE;
SEGMENTS: ARRAY[1..MAX_APPLICATION] OF SECTYPE;
ALLOCATION: ARRAY[1..MAX_ALLOC] OF ALLOCTYPE;

BENEFITS: ARRAY[1..MAX_ALLOC] OF BENETYPE;
ALLOC_SITE: ARRAY[1..MAX_ALLOC] OF ALLOC_SITE_TYPE;

FRAG_FILENAME : STRING[15];

FRAGMENT_FILE (* FILE OF THE FRAGMENTS *) : TEXT;

EX : FILE;

NFRAGS,
S,I,J,C,
K,CF,M,
FP,NP,P,
T,NT,
NSEGS,
NBEN,
TINF,
NSITE,
NUM_ALLOC,
NUM_CLUST,
NUM_ALLOC_SITE,
FI : INTEGER;

NIF : BOOLEAN;


OP_COSTS : OP_CST; (* ARRAY OF SITE OPERATIONAL COSTS *)
(* ONE ROW IN THE ARRAY FOR EACH SITE *)
(* COL 1 IS COST OF SPACE AT THE SITE *)
(* COL 2 IS COST OF RETRIEVAL AT THE SITE *)
(* COL 3 IS COST OF UPDATE AT THE SITE *)

COM_COSTS : CM_CST; (* ARRAY OF COMMUNICATION COSTS BETWEEN *)
(* ONE ROW AND ONE COLUMN FOR EACH SITE *)
(* SITE. ROW 1 COLUMN 2 IS THE COST OF *)
(* COMMUNICATION BETWEEN SITES 1 AND 2. *)
(* ROW 2 COLUMN 3 IS THE COST OF COMMUNICATION *)
(* BETWEEN SITES 2 AND 3. ETC... *)
(* COM_COSTS[I,J] = 0 WHERE I=J *)
(* COM_COSTS[I,J] = COM_COSTS[J,I] *)

CLUST_SITE: CLUST_TYPE; (* ARRAY OF CLUSTER INFORMATION *)
(* ONE ROW IN THE ARRAY FOR EACH SITE *)
(* COL 1 IS CLUSTER NUMBER *)
(* COL 2 IS THE SITE NUMBER *)
(* COL 3 IS COST OF SPACE AT THE SITE *)
(* COL 4 IS COST OF RETRIEVAL AT THE SITE *)
(* COL 5 IS COST OF UPDATE AT THE SITE *)
(* COL 6 IS AVERAGE COST OF COMMUNICATION *)
(* FOR THE SITE *)
CLUSTER_DATA: ARRAY [1..MAX_SITES] OF CS_COST_TYPE;
(* ARRAY OF CLUSTER COSTS AND SITES, ONE *)
(* FOR EACH CLUSTER *)
(* COSTS[1] IS AVERAGE COST OF RETRIEVAL *)
(* COSTS[2] IS AVERAGE COST OF UPDATE *)
(* COSTS[3] IS AVERAGE COST OF SPACE *)
(* SITES IS THE SET OF SITES IN THE CLUSTER *)

CLUST_FILENAME,
COST_FILENAME,
ALLOC_FILENAME,
FNL_ALLOC_FILENAME : STRING[15];

IN_STR : STRING[10];

INFILE : TEXT;

TOT_FREQU : ARRAY[1..MAX_FRAG] OF REAL;
MXFREQ : ARRAY[1..MAX_ALLOC] OF REAL;
MXPBI : ARRAY[1..2] OF REAL;
FREQ_RR : ARRAY[1..MAX_FRAG,1..MAX_SITES] OF REAL;
FREQ_RU : ARRAY[1..MAX_FRAG,1..MAX_SITES] OF REAL;
CA : ARRAY[1..MAX_FRAG,1..MAX_SITES] OF REAL;
CN : ARRAY[1..MAX_FRAG,1..MAX_SITES] OF REAL;
B : ARRAY[1..MAX_FRAG,1..MAX_SITES] OF REAL;
ALLOCATE : ARRAY[1..MAX_FRAG,1..MAX_SITES] OF BOOLEAN;
FRAG_SIZE : ARRAY[1..MAX_FRAG] OF REAL;
SITESOF : ARRAY[1..MAX_FRAG] OF SITE_SET_TYPE;
SITSET : SITE_SET_TYPE;
SITES : ARRAY[1..MAX_SITES] OF INTEGER;
FRAG : ARRAY[1..MAX_FRAG] OF INTEGER;

SITE,
CLUST,
NUM_SITES,
RESULT : INTEGER;

OK,
FOUND : BOOLEAN;

C_US,
C_U,
A_VC,
C_COM,
CRU : REAL;

(* QUIT TO SYSTEM ***************************************************)
PROCEDURE HALT;
BEGIN (* QUIT *)
  TEXTBACKGROUND(BLACK);
  CLRSCR;
  HALT
END; (* QUIT *)

(* THIS PROCEDURE CLEARS THE FRAGMENTS AND SEGMENTS TABLES *)

PROCEDURE TCLEAR;

VAR I, J: INTEGER;

BEGIN
  FOR I := 1 TO MAX_FRAG DO
    BEGIN
      FOR J := 1 TO MAX_APPLICATION DO
        BEGIN
          FRAGMENTS[I].SEGM[J] := 0;
          FRAGMENTS[I].SITES[J] := 0;
        END;
      FRAGMENTS[I].TUPLES := [];
      FRAGMENTS[I].RELATION := 0;
      FRAGMENTS[I].SIZE := 0;
    END;
  END;

  FOR I := 1 TO MAX_APPLICATION DO
    BEGIN
      SEGMENTS[I].TUPLES := [];
      SEGMENTS[I].SITES := 0;
      SEGMENTS[I].RELATION := 0;
    END;

END; (* TCLEAR *)

(* LOAD THE SITE DATA FROM THE DISK *)

PROCEDURE COPY_FROM_SITE;

VAR
  I,
  J: INTEGER;
  INF1: TEXT;

BEGIN (* COPY FROM SITE *)
  ASSIGN(INF1, INDRH+'+OST_FILENAME+IN');
  RESET(INF1);
READIN(INF1, NUM_SITES);
READIN(INF1);
FOR I := 1 TO NUM_SITES DO
  FOR J := 1 TO 3 DO
    READIN(INF1, OOP_COSTS[I, J]);
    READIN(INF1);
  END;
END; (* COPY FROM SITE *)

(* LOAD THE CLUSTER DATA FROM THE DISK **********************************************)

PROCEDURE COPY_FROM_CLUSTER;

VAR
  I, C, J : INTEGER;
  R : REAL;
  INFL2: TEXT;

BEGIN (* COPY FROM CLUSTER *)
  ASSIGN(INFL2, OUDIR+'+'+CLUSTER_FILENAME+'.OUT');
  RESET(INFL2);
  READIN(INFL2);
  READIN(INFL2);
  READIN(INFL2);
  READIN(INFL2);
  READ(INFL2, NUM_CLUSTER);
  FOR C := 1 TO NUM_CLUSTER DO
    BEGIN
      CLUSTER_DATA[C].COSTS[1] := 0;
      CLUSTER_DATA[C].COSTS[2] := 0;
      CLUSTER_DATA[C].COSTS[3] := 0;
      CLUSTER_DATA[C].SITES :=[];
    END;
  END;
END;
END;
READIN(INFL2);
END;
CLOSE(INFL2);
END; (* COPY FROM CLUSTER *)

(* LOAD ALLOCATION DATA FROM THE DISK *******************************************)
PROCEDURE COPY_FROM_ALLOC;

VAR
  I,
  J : INTEGER;
  CH : CHAR;
  OUTFILE : TEXT;

BEGIN (* COPY TO ALLOC *)
  ASSIGN(OUTFILE, OUTER+':'+ALLOC_FILENAME+'.OUT');
  RESET(OUTFILE);
  READIN(OUTFILE);
  READIN(OUTFILE);
  READIN(OUTFILE);
  READIN(OUTFILE);
  READOUTFILE,NUM_ALLOC);
  READIN(OUTFILE);
  READIN(OUTFILE);
  READIN(OUTFILE);
  READIN(OUTFILE);
  FOR I := 1 TO NUM_ALLOC DO
    BEGIN
      READ(OUTFILE, ALLOCATION[I].FRAGMENT);
      READ(OUTFILE, ALLOCATION[I].CLUSTER);
      CH := ' ';
      WHILE (CH = ' ') DO READ(OUTFILE, CH);
      ALLOCATION[I].PERM := CH;
      READ(OUTFILE, ALLOCATION[I].PSI);
      READ(OUTFILE, ALLOCATION[I].FREQ);
      READ(OUTFILE, ALLOCATION[I].FREQU);
      READ(OUTFILE, ALLOCATION[I].CS);
      READIN(OUTFILE);
    END;
  CLOSE(OUTFILE);
END; (* COPY FROM ALLOC *)

(********************************************************************)
PROCEDURE COPY_FROM_FRAGMENT;

VAR
C, I, J, K: INTEGER;
INFL3: TEXT;
BEGIN
  CLEAR;
  ASSIGN(INFL3, OUTDIR+':'+'FRAG_FILENAME'+.OUT');
  RESET(INFL3);
  READIN(INFL3);
  READIN(INFL3);
  READIN(INFL3);
  READIN(INFL3);
  READ(INFL3,NFRAGS);
  READIN(INFL3);
  READIN(INFL3);
  READIN(INFL3);
  READIN(INFL3);
  READIN(INFL3);
  FOR I := 1 TO NFRAGS DO
    BEGIN
      FOR J := 1 TO 7 DO
        BEGIN
          READ(INFL3,FRAGMENTS[I].SEG[J]);
          READ(INFL3,FRAGMENTS[I].RELATION);
          READ(INFL3,FRAGMENTS[I].SIZE);
          J:=1;
          WHILE (NOT EOF(INFL3) ) DO
            BEGIN
              READ(INFL3,K);
              FRAGMENTS[I].TUPLES:=FRAGMENTS[I].TUPLES+[K];
              J:=J+1;
            END;
          READIN(INFL3);
          FOR J := 1 TO 7 DO
            READ(INFL3,FRAGMENTS[I].SITES[J]);
          READIN(INFL3);
        END;
      READIN(INFL3);
      READIN(INFL3);
      READIN(INFL3);
      READIN(INFL3);
      READ(INFL3,NSEGGS);
      READIN(INFL3);
      READIN(INFL3);
      READIN(INFL3);
      READIN(INFL3);
      FOR I := 1 TO NSEGGS DO
        BEGIN
          READ(INFL3,SEGMENTS[I].SITES);
          READ(INFL3,SEGMENTS[I].RELATION);
          READ(INFL3,SEGMENTS[I].FREQR);
          READ(INFL3,SEGMENTS[I].FREQU);
J := 1;
WHILE (NOT EOF(INFL3)) DO
BEGIN
READ(INFL3,K);
SEGMENTS[I].TUPLES := SEGMENTS[I].TUPLES + [K];
J := J + 1
END;
READLN(INFL3);
READLN(INFL3);
END;
(* COPY FROM_FRAGMENT *)

(* COPY ALLOCATION DATA TO THE DISK ******************************************************)
PROCEDURE COPY_TO_ALLOC;
VAR
I,
J : INTEGER;
OUTFILE: TEXT;
BEGIN (* COPY TO_ALLOC *)
ASSIGN (OUTFILE, OUTDRH:1+INL_ALLOC_FILENAME:1.CUT');
REWRITE (OUTFILE);
WRITELN(OUTFILE, 'FINAL CLUSTER ALLOCATION DATA');
WRITELN(OUTFILE, 'FRAGMENT CLUSTER R_CLUST RB');
FOR I := 1 TO NUM_ALLOC DO
BEGIN
WRITELN(OUTFILE, BENEFITS[I].FRAGMENT: 5,
BENEFITS[I].CLUSTER: 10,
BENEFITS[I].R_CLUSTER: 11,
BENEFITS[I].CA: 13:3,
BENEFITS[I].RCN: 10:3,
BENEFITS[I].RB: 10:3);
END;
WRITELN (OUTFILE);
WRITELN (OUTFILE);
WRITELN (OUTFILE, NUM_ALLOC, 'ALLOCATIONS OF FRAGMENTS TO CLUSTERS');
WRITELN(OUTFILE);
WRITELN(OUTFILE, 'FRAGMENT CLUSTER PERM');
WRITELN(OUTFILE, 'FRAGMENT CLUSTER PERM');
FOR I := 1 TO NUM_ALLOC DO
BEGIN
WRITE(OUTFILE, ALLOCATION[I].FRAGMENT: 5);
END;
CLOSE(INFL3);
END; (* COPY_FROM_FRAGMENT *)
WRITE (OUTFILE, ALLOCATION[I].CLUSTER:10);
WRITE (OUTFILE, ALLOCATION[I].PERM:8);
WRITE (OUTFILE);
END;
WRITE (OUTFILE);
WRITE (OUTFILE, '**************************************************************');
WRITE (OUTFILE, ' FINAL SITE ALLOCATION DATA.');
WRITE (OUTFILE, '**************************************************************');
WRITE (OUTFILE);
WRITE (OUTFILE, '--------------- SITE ALLOC CA CN B' );
WRITE (OUTFILE);
WRITE (OUTFILE);
FOR J:=1 TO NFRAGS DO
BEGIN
FOR I:=1 TO NUM_ALLOC_SITE DO
BEGIN
IF (ALLOC_SITE[I].FRAGMENT = J ) THEN
WRITE (OUTFILE, ALLOC_SITE[I].FRAGMENT:5,
ALLOC_SITE[I].SITE:9,
ALLOC_SITE[I].ALLOC:8,
ALLOC_SITE[I].CA:10:2,
ALLOC_SITE[I].CN:10:2,
ALLOC_SITE[I].B:10:2);
END; (* FOR I *)
END; (* FOR J *)
WRITE (OUTFILE);
WRITE (OUTFILE, '**************************************************************');
WRITE (OUTFILE, ' FINAL SOLUTION');
WRITE (OUTFILE, '**************************************************************');
WRITE (OUTFILE);
WRITE (OUTFILE, '--------------- SITE');
WRITE (OUTFILE, '---------- ----- ');
WRITE (OUTFILE);
WRITE (OUTFILE);
FOR J:=1 TO NFRAGS DO
BEGIN
FOR I:=1 TO NUM_ALLOC_SITE DO
BEGIN
IF( (ALLOC_SITE[I].FRAGMENT = J ) AND
(ALLOC_SITE[I].ALLOC) )
THEN
WRITE (OUTFILE, ALLOC_SITE[I].FRAGMENT:5,
ALLOC_SITE[I].SITE:9);
END; (* FOR I *)
END; (* FOR J *)
CLOSE (OUTFILE);
END; (* COPY_TO_ALLOC *)
(* INPUT FILE NAMES *)
PROCEDURE SCREEN_PHASE_II;
BEGIN (* SCREEN_PHASE_II * )
PROCEDURE ALLOCATE_TO_SITES;

*** ALLOCATE FRAGMENTS TO SITES WHERE THERE IS A POSITIVE BENEFIT ***

(* GLOBALS REQUIRED: *)

(* TNF TOTAL NUMBER OF FRAGMENTS TO ALLOCATE *)
(* NSITE NUMBER OF SITES FOR WHICH TO ALLOCATE FRAGMENTS *)
(* CUS COST OF A SPACE UNIT (1 BYTE) *)
(* CUR AVERAGE COST OF A UNIT RETRIEVAL (1 RECORD) *)
(* AVCU AVERAGE COST OF AN UPDATE *)
(* CCOM AVERAGE COST OF COMMUNICATION BETWEEN 2 SITES *)
(* FREQ_RR[M,I] NUMBER OF RETRIEVALS OF FRAGMENT M AT SITE I *)
(* FREQ_RU[M,I] NUMBER OF UPDATES OF FRAGMENT M AT SITE I *)
(* SITESSOF[M] SET OF SITES USING FRAGMENT M *)
(* FRAG_SIZE[M] SIZE OF FRAGMENT M (BYTES) *)

(* GLOBALS RETURNED: *)

(* CA[M,I] COST OF ALLOCATION OF FRAGMENT M TO SITE I *)
(* CN[M,I] COST OF NOT ALLOCATING FRAGMENT M TO SITE I *)
(* B[M,I] BENEFIT OF ALLOCATING FRAGMENT M TO SITE I *)
(* ALLOCATE[M,I] ALLOCATION OF FRAGMENT M TO SITE I, TRUE OR FALSE *)

(*)
AVNRU (* AVERAGE NUMBER OF UPDATES ISSUED/DAY TO FRAGMENT M BY SITES *)
(* REMOTE TO SITE I *)
:REAL;

CS, (* COST OF SPACE USED BY FRAGMENT M AT SITE I *)
CR, (* COST OF LOCAL RETRIEVALS OF FRAGMENT M AT SITE I *)
CLU, (* COST OF LOCAL UPDATES TO FRAGMENT M AT SITE I *)
RMCF, (* COMMUNICATION FREQUENCY OF RETRIEVALS FROM REMOTE SITE *)
RU,
CMU,
RR,
CMR
:REAL;

FUNCTION REMOTE(I,J:INTEGER) :REAL;

VAR
K: INTEGER;
X: REAL;
BEGIN
X:=0;
FOR K:=1 TO NFRAGS DO
  IF(K IN SITESS_OF[I]) AND (K<>J) THEN
    X:=X+FREQ_RU[I,K];
    REMOTE:=X;
END;

BEGIN (* ALLOCATE TO SITES *)
FOR M:=1 TO TNF DO
  FOR I:=1 TO NSITE DO
  BEGIN
    CS := C_US * FRAG_SIZE[M];
    AVNR := FREQ_RR[M,I];
    CR := C_US * AVNR;
    AVNU := FREQ_RU[M,I];
    CLU:= AVCU * AVNU;
    AVNRU := REMOTE(M,I);
    RU:= AVNRU * AVCU;
    CMU:= AVNRU * C_COM;
    CA[M,I]:= CS + CR + CLU + RU + CMU;
    RR:= CR;
    CMR:= C_COM * AVNR;
    CN[M,I]:= RR + CMR;
    B[M,I]:=CN[M,I]-CA[M,I];
  END;

(*)
(* ALLOCATE FRAGMENTS TO SITES WHERE THERE IS A POSITIVE BENEFIT *)
(* ALLOCATE_FRAGMENTS_TO_SITES WHERE THERE IS A_POSITIVE_BENEFIT *)

FOR M:=1 TO NFRAG DO
    FOR I:=1 TO NSITE DO
        IF B[M,I] > 0
            THEN ALLOCATE[M,I]:=TRUE
            ELSE ALLOCATE[M,I]:=FALSE;

END; (* ALLOCATE_TO_SITES *)

PROCEDURE GET_FREQ(M,I,J,SITE: INTEGER);
VAR
    K: INTEGER;
BEGIN (* GET_FREQ *)
    FREQ_RR[M,I]:=0;
    FREQ_RU[M,I]:=0;
    K:=1;
    WHILE ((0 <> FRAGMENTS[ALLOCATIONS[J].FRAGMENT].SITES[K])
        AND (0 = FREQ_RR[M,I])) DO
        BEGIN
            IF (SITE = FRAGMENTS[ALLOCATIONS[J].FRAGMENT].SITES[K])
                THEN
                    BEGIN
                    END; (* IF THEN *)
                K:=K+1;
        END; (* WHILE *)
    END; (* GET_FREQ *)

FUNCTION NEXT_TEMP(VAR NT:INTEGER; T,P: INTEGER): BOOLEAN;
VAR
    I: INTEGER;
    MINCC: REAL;
BEGIN
    NEXT_TEMP:=TRUE;
    NT:=0;
    MINCC:=32767;
    FOR I:= 1 TO NUM_ALLOC DO (* FIND THE TEMPORARY CLUSTER WITH MINIMUM *)
        BEGIN (* COMMUNICATION COST RELATIVE TO CLUSTER FP *)
            IF ( (ALLOCATIONS[I].FRAGMENT = M) AND (ALLOCATIONS[I].PERM = 'T')
                AND (COM_COSTS[ALLOCATIONS[P].CS,ALLOCATIONS[I].CS] < MINCC )
                AND (I<>T) )
                THEN
                    BEGIN
                        (* CODE *)
                    END; (* IF THEN *)
        END; (* FOR *)
    END; (* NEXT_TEMP *)

END;
\begin{verbatim}
NT:=I;
MINCC:= COM_COSTS[ALLOCATION[P].CS,ALLOCATION[I].CS];
END; (* IF *)
END; (* FOR *)
IF( NT = 0 ) THEN NEXT_TEMP:=FALSE;
END;

FUNCTION MEMBER : BOOLEAN;

VAR
  K: INTEGER;
  MEM: BOOLEAN;

BEGIN
  MEM:=FALSE;
  K:=1;
  WHILE ( (K <= MAX_SITES) AND (NOT MEM) ) DO
  BEGIN
    IF (SITES[I] = FRAGMENTS[ALLOCATION[J].FRAGMENT].SITES[K])
    THEN MEM:=TRUE;
    K:=K+1;
  END;
  MEMBER:=MEM;
END;

FUNCTION NEXT_PERM(VAR NP: INTEGER; P: INTEGER) :BOOLEAN;

VAR
  I: INTEGER;

BEGIN
  NEXT_PERM:=TRUE;
  NP:=0; (* FIND NEXT PERMANENT CLUSTER FOR THIS FRAGMENT *)
  I:=P+1;
  WHILE ( (NP = 0) AND (I <= NUM_ALLOC) ) DO
  BEGIN
    IF ( (ALLOCATION[I].FRAGMENT = M) AND (ALLOCATION[I].PERM = 'P') )
    THEN NP:=I;
    I:=I+1;
  END;
  IF (NP = 0) THEN NEXT_PERM:=FALSE;
END;

BEGIN (* MAIN *)
  WINDOW(1,1,80,25);
  TEXTBACKGROUND(BLACK);
\end{verbatim}
CLRSCR;
SCREEN_PHASE_II;

OK:=FALSE;
WHILE (NOT OK) DO
BEGIN
  GOTOXY(24,4);
  ALLOC_FILENAME := '';
  READIN(ALLOC_FILENAME);
  IF (ALLOC_FILENAME = 'Q') OR (ALLOC_FILENAME = 'q') THEN
    HALT;
  ASSIGN(INFILE, OUTDR + ':1' + ALLOC_FILENAME+'.'OUT');
  ($I-) RESET(INFILE) ($I+);
  OK := (IORESULT = 0);
  IF (NOT OK) THEN
    BEGIN
      GOTOXY(10,23);
      TEXTCOLOR(WHITE);
      WRITE(OUTDR+':1'+ALLOC_FILENAME+'.'OUT, DOES NOT EXIST, TRY AGAIN OR TYPE Q
    END;
  END;
OK:=FALSE;
WHILE (NOT OK) DO
BEGIN
  GOTOXY(24,6);
  CLUST_FILENAME := '';
  READIN(CLUST_FILENAME);
  IF (CLUST_FILENAME = 'Q') OR (CLUST_FILENAME = 'q') THEN
    HALT;
  ASSIGN(INFILE, OUTDR + ':1' + CLUST_FILENAME+'.'OUT');
  ($I-) RESET(INFILE) ($I+);
  OK := (IORESULT = 0);
  IF (NOT OK) THEN
    BEGIN
      GOTOXY(10,23);
      TEXTCOLOR(WHITE);
      WRITE(OUTDR+':1'+CLUST_FILENAME+'.'OUT, DOES NOT EXIST, TRY AGAIN OR TYPE Q
    END;
  END;
OK:=FALSE;
WHILE (NOT OK) DO
BEGIN
  GOTOXY(24,8);
  COST_FILENAME :='';
  READIN(COST_FILENAME);
  IF (COST_FILENAME = 'Q') OR (COST_FILENAME = 'q') THEN
    HALT;
  ASSIGN(INFILE, INDR + ':1' + COST_FILENAME+'.'IN');
  ($I-) RESET(INFILE) ($I+);
  OK := (IORESULT = 0);
IF (NOT OK) THEN
BEGIN
  GOTOXY(10,23);
  TEXTCOLOR(WHITE);
  WRITE(INDR+':'+COST_FILENAME+'.IN, DOES NOT EXIST, TRY AGAIN OR TYPE QUIT
END
END;

OK:=FALSE;
WHILE (NOT OK) DO
BEGIN
  GOTOXY(24,10);
  FRAG_FILENAME := '';
  READIN(FRAG_FILENAME);
  IF (FRAG_FILENAME = 'Q') OR (FRAG_FILENAME = 'q') THEN
    HALT;
  ASSIGN(INFILE, OUTDR+'.' +FRAG_FILENAME='.OUT');
  {$I-} RESET(INFILE) {$I+};
  OK := (IORESULT = 0);
  IF (NOT OK) THEN
BEGIN
  GOTOXY(10,23);
  TEXTCOLOR(WHITE);
  WRITE(OUTDR+':'+FRAG_FILENAME='.OUT, DOES NOT EXIST, TRY AGAIN OR TYPE QUIT
END
END;

GOTOXY(24,12);
FNL_ALLOC_FILENAME := '';
READIN(FNL_ALLOC_FILENAME);

GOTOXY(10,23); CLRSCR;
WINDOW(15,11,55,15);
TEXTBACKGROUND(WHITE);
TEXTCOLOR(GREEN);
CLRSCR;
GOTOXY(1,3);
WRITE('READ SITE, CLUSTER, INITIAL ALLOC DATA...');
COPY_FROM_SITE;
COPY_FROM_FRAGMENT;
COPY_FROM_CLUSTER;
COPY_FROM_ALLOC;

WINDOW(17,13,57,17);
TEXTBACKGROUND(GREEN);
TEXTCOLOR(WHITE);
CLRSCR;
GOTOXY(1,3);
WRITE('CALCULATING RELATIVE BENEFITS ...');

NBEN:=0;
FOR M:= 1 TO NFRAGS DO
BEGIN  
FP:=0;  
I:=1;  
WHILE (FP = 0) DO  (* FIND FIRST PERMANENT CLUSTER FOR THIS FRAGMENT *)  
BEGIN  
IF ( (ALLOCATION[I].FRAGMENT = M) AND (ALLOCATION[I].PERM = 'P') )  
THEN FP:=I;  
I:=I+1;  
END;  
P:=FP-1;  
T:=0;  
WHILE ( NEXT_TEMP(NP,T,FP) ) DO  (* IF TEMPORARY FRAGMENT FOUND *)  
BEGIN  
T:=NP;  
WHILE ( NEXT_PERM(NP,P) ) DO  
BEGIN  
P:=NP;  
BNEN:=BNEN+1;  
BENEFITS[BNEN].FRAGMENT:=M;  
BENEFITS[BNEN].CLUSTER:=ALLOCATION[T].CLUSTER;  
BENEFITS[BNEN].R_CLUSTER:=ALLOCATION[P].CLUSTER;  
(* CALCULATE COST OF REMOTE UPDATE FOR FRAGMENT M AT CLUSTER T *)  
CRU:=0;  
FOR I:= 1 TO NUM_ALLOC DO  (* FIND OTHER CLUSTERS USING FRAGMENT M *)  
BEGIN  
IF ( (ALLOCATION[I].FRAGMENT = M) AND (I <> T) )  
THEN  
BEGIN  
*ALLOCATION[I].FREQ)  
+(COM_COSTS[ALLOCATION[T].CS,ALLOCATION[I].CS]  
*ALLOCATION[I].FREQ);  
END;  (* IF *)  
END;  (* FOR *)  
(* CALCULATE COST OF ALLOCATION FOR FRAGMENT M AT CLUSTER T *)  
*ALLOCATION[T].FREQ)  
+(CLUSTER_DATA[ALLOCATION[T].CLUSTER].COSTS[2]  
*ALLOCATION[T].FREQ)  
+(CLUSTER_DATA[ALLOCATION[T].CLUSTER].COSTS[3]  
*FRAGMENTS[M].SIZE)  
+CRU;  
(* CALCULATE RELATIVE COST OF NOT ALLOCATING FRAGMENT M AT CLUSTER T *)  
*ALLOCATION[T].FREQ)  
+(COM_COSTS[ALLOCATION[P].CS,ALLOCATION[T].CS]  
*ALLOCATION[T].FREQ);  
(* CALCULATE RELATIVE BENEFIT FOR FRAGMENT M AT CLUSTER T *)  
BENEFITS[BNEN].RB:=BENEFITS[BNEN].RCN-BENEFITS[BNEN].CA;  
(* CANCEL ALLOCATION OF FRAGMENT M TO CLUSTER T IF RB < 0 *)  
IF (BENEFITS[BNEN].RB < 0 )
THEN
BEGIN
  ALLOCATION[T].PERM := 'C';
  P := NUM_ALLOC;
END;
END; (* WHILE NEXT_PERM *)

IF (ALLOCATION[T].PERM = 'T') THEN ALLOCATION[T].PERM := 'P';
P := P - 1;
END; (* WHILE NEXT_TEMP *)

END; (* FOR M *)

WINDOW(19, 15, 59, 19);
TEXTBACKGROUND(WHITE);
TEXTCOLOR(GREEN);
CIRSCR;
GOTOXY(1, 3);
WRITE('DETERMINING SITE ALLOCATION ...');

NUM_ALLOC_SITE := 0;
FOR C := 1 TO NUM_CLUST DO
BEGIN
  NSITE := 0;
  SITESET := [];
  FOR I := 1 TO MAX_SITES DO
  BEGIN
    IF (I IN CLUSTER_DATA[C].SITES) THEN
    BEGIN
      NSITE := NSITE + 1;
      SITES[NSITE] := I;
      SITESET := SITESET + [NSITE];
    END;
  END;
  IF (NSITE = 1) THEN
  BEGIN
    FOR J := 1 TO NUM_ALLOC DO
    BEGIN
      IF (C = ALLOCATION[J].CLUSTER) AND
      ('P' = ALLOCATION[J].PERM) THEN
      BEGIN
        NUM_ALLOC_SITE := NUM_ALLOC_SITE + 1;
        ALLOC_SITE[NUM_ALLOC_SITE].FRAGMENT := ALLOCATION[J].FRAGMENT;
        ALLOC_SITE[NUM_ALLOC_SITE].SITE := SITES[NSITE];
        ALLOC_SITE[NUM_ALLOC_SITE].ALLOC := TRUE;
        ALLOC_SITE[NUM_ALLOC_SITE].CA := 0;
        ALLOC_SITE[NUM_ALLOC_SITE].CN := 0;
        ALLOC_SITE[NUM_ALLOC_SITE].B := 0;
      END;
    END;
  END;
END;
BEGIN
  M:=0;
  FOR J:= 1 TO NUM_ALLOC DO
    BEGIN
      IF ( ( C = ALLOCATION[J].CLUSTER ) AND
          ( 'P' = ALLOCATION[J].PERM ) )
      THEN
        BEGIN
          M:=M+1;
          FRAG[M]:=ALLOCATION[J].FRAGMENT;
          SITES_S[M]:=[I];
          FRAG_SIZE[M]:=FRAGMENTS[ALLOCATION[J].FRAGMENT].SIZE;
          FOR I:= 1 TO NSITE DO
            BEGIN
              FREQ_JRR[M,I]:=0;
              FREQ_RU[M,I]:=0;
              IF ( MEMBER )
              THEN
                BEGIN
                  SITES_S[M]:=SITES_S[M]+[I];
                  GET_FREQ(M,I,J,SITES[I]);
                END;
            END;
        END;
      END;
    END;
  END;

TNF:=M;
C_UR:=CLUSTER_DATA[C].COSTS[1];
AVCU:=CLUSTER_DATA[C].COSTS[2];
C_US:=CLUSTER_DATA[C].COSTS[3];
C_CCM:=C_CCM_COSTS[SITES[1],SITES[2]];
ALLOCATE TO SITES;
FOR M:= 1 TO TNF DO
  BEGIN
    FOR I:= 1 TO NSITE DO
      BEGIN
        NUM_ALLOC_SITE:=NUM_ALLOC_SITE+1;
        ALLOC_SITE[NUM_ALLOC_SITE].FRAGMENT:=FRAG[M];
        ALLOC_SITE[NUM_ALLOC_SITE].SITE:=SITES[I];
        ALLOC_SITE[NUM_ALLOC_SITE].ALLOC:=ALLOCATE[M,I];
        ALLOC_SITE[NUM_ALLOC_SITE].CA:=CA[M,I];
        ALLOC_SITE[NUM_ALLOC_SITE].CN:=CN[M,I];
        ALLOC_SITE[NUM_ALLOC_SITE].B:=B[M,I];
      END;
  END;
END; (* FOR C *)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
WINDOW(21,17,61,21);
TEXTBACKGROUND(GREEN);
TEXTCOLOR(WHITE);
CLRSCR;
GOTOXY(1,3);
WRITE('OUTPUT FINAL ALLOCATION ...');
COPY_TO_ALLOC;

WINDOW(1,1,80,25);
ASSIGN(EX,PDR+':MAINMENU.COM');
EXECUTE(EX)

END.
BIBLIOGRAPHY


