FAST ACCESS SEQUENTIAL STRUCTURES

G. Martella F.A. Schreiber
Istituto di Elettronica - Politecnico di Milano
P. Leonardo da Vinci, 32 - 20133 Milano - Italy

1. INTRODUCTION

The minimization of the product (time x storage) is a very common goal to be achieved in many problems in the world of data management. Unfortunately very often the requirements for minimizing one of the two factors are in contrast with those for minimizing the other one, so that a compromise must be looked for according to the peculiar problem or application in concern. This is the case of query-oriented information systems in which large amounts of data have to be searched for retrieving the records answering some particular questions, expressed in terms of the value taken by some attributes of those records, called keys. If the amount of data to be examined is very large and/or the storage medium is a slow one, the time to give the answer can be very long if an appropriate storage organization is not chosen. One of the most widely used techniques to solve this problem is the *file inversion* with respect to one or more of the relevant attributes, creating secondary indices which put in relation the key value with the storage locations [1].
This concept can be extended to obtain Query-inverted File Organizations in which the addresses of all the storage blocks containing the records answering a particular request are put in relation with the request itself (fig. 1). All the records are stored in contiguous storage locations in order to reduce the access time to answer the query [2].

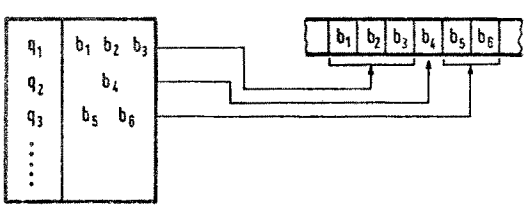


Fig. 1

This organization, however, requires a lot of storage since records answering more than one query are to be stored in a redundant way.
In [1,3] the *consecutive retrieval property* (CR) has been defined for

a query set as the peculiarity of keeping the storage consecutiveness of all records answering a query, while avoiding the duplication of any of them. Such an organization clearly minimizes both storage space and retrieval time, but it is not applicable to whatever set of queries on whatever set of records. So it becomes of interest the consideration of consecutive retrieval *with redundancy* (CRWR) while trying to keep the redundancy as low as possible.
In [3] it has been investigated the possibility of organizing a file on a CR basis only by examining the set of queries and their structure, extending then this organization to dynamic files.
In this paper we shall present a solution to the CRWR problem in dynamic files with fast access properties to be used in real-time systems and we shall give some comparison data with other data organizations.

2. - THE QUERY GRAPH

In this section we introduce some definitions about the notion of covering of two queris and a graph representation of the query set; a formal treatment of this subject can be found in [3], from which paper we report some conclusions.
Given a set $\{Q\} \equiv (q_1, \ldots, q_2)$ of *queries*, a set $\{A\} \equiv (a_1, \ldots, a_N)$ of *attributes*, a set $\{V\} \equiv (v_{a_1 1}, \ldots, v_{a_1 k}, \ldots, v_{a_N 1}, \ldots, v_{a_N p})$ of *values* which can be given to each attribute, a query q_1 is identified as a string of one value from $\{V\}$ for each one of the attributes of the set $\{A\}$ which uniquely specifies a set of records in a file (characteristic values). A value $v_{a_j i}$ of an attribute a_j may make another attribute a_k meaningless; an *indifference condition* X is entered as value of an attribute a_i whenever the value of a_i is not essential in answering a particular query q_1 (notice that no more than n-1 indifference conditions can be specified in a query).
The set $\{Q\}$ of all allowable queries can then be built knowing the set of values of each attribute and their characteristics. We should give each attribute all its possible values, the indifference condition included; owing to the aforesaid uncompatibility among the values of some different attributes, the corresponding "theoric" queries must be deleted, thereby reducing the dimensions of the set $\{Q\}$.
Let us consider now a query q_1 with only m attributes having some specified value, being the other n-m indifference conditions. Be $\{R\}$ the set of records constituting the whole file; be $\rho(q_1) \in \{R\}$ the set of records answering q_1; the set $\rho(q_1)$ then is constituted by all the records identified by the assigned values for the key attributes

and by any possible configuration of values for the remaining attributes.

It is possible now to build a *Covering Table* C as a matrix the colunms of which correspond each to an attribute a_j and the rows to a query q_i. Entries c_{ij} represent the value taken by the attribute a_j in the query q_i.

Def.: A query q_i is said to *cover* a query q_k (simbolically $q_i > q_k$) if and only if, for each attribute a_j $1 \leq j \leq n$,

either:

$$c_{ij} = c_{kj}$$

or

$$c_{ij} = x$$

Obviously, whenever two distinct queries q_i, q_k are expressed with the same number of indifference conditions, no covering possibility exists between them.

It is possible to demonstrate that the covering operation is transitive [3]. If a subset $\{q_l\}$ of $\{Q\}$ exists such that $q_{l_1} > q_{l_2} > \ldots > q_{l_s}$ it is called a *covery chain*.

A *covering graph* can be built from a *covering table* under the following rules :

1. - to each query $q_i \in \{Q\}$ a node corresponds in the graph;
2. - whenever $q_i > q_k$ and there is no q_j such that $q_i > q_j > q_k$, there is an oriented edge from q_i to q_k.

In [3] it has been proved that whenever the covering graph, we call hereafter the "query-graph", is a treee, a CR organization is possible; moreover, whenever {Q} can be subdivied in subsets such that the "root--queries" of the various subsets are all disjoint and all the covering graphs for the subsets are trees, a CR organization is still possible.

Let now C be a general covering table. The associated *covering graph* will have some source nodes corresponding to queries not covered by any other query (at least one of such nodes always exists) and some sink nodes corresponding to the lowest level queries. However it is possible to consider only *covering graphs* with a single source node without limiting the generality of the related considerations. It is in fact possible to add to the query set a "dummy" query covering all the others, this query being described by the intersection set of the characteristic values of attributes for all the source nodes. Would this set be empty, the dummy query could be represented as: "read the whole file".

We consider now in a general covering table two queries q_i and q_j such that:

$$q_i \not\succ q_j \quad \text{and} \quad q_j \not\succ q_i$$

$$\rho(q_i) \cap \rho(q_j) \neq \phi$$

The two following conditions must then be met

a) if attribute a_k has non indifferent values both for q_i and q_j it must be

$$c_{ij} = c_{jk} \ (\neq x)$$

b) for at least two attributes a_l, a_m it must be

$$c_{il} = x \neq c_{jl} \quad \text{and} \quad c_{jm} = x \neq c_{im}$$

These conditions are expressed in the covering graph by the existence of a node, corresponding to the query consisting of all the specified values of q_i and q_j, with two incident edges; the covering graph then is no more a tree.
Even if there are some instances of non-tree query-graphs still allowing a non-redundant CR organization, these graphs correspond to the existence of two one-dimension CR organization with *conjunctive ends* (see theorem 4 in [1] and Fig. 2), this property is not generally true for any general query-graph.

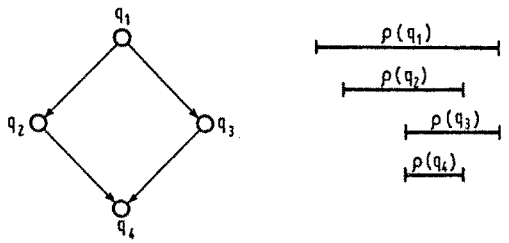


Fig. 2

The problem then is to look for a CRWR organization which assures a fast access-time by multiplexing the records answering some subsets of queries.

3. - REDUNDANT CR ORGANIZATIONS

We have already noticed that the query graph is always a cycle-free oriented graph (acyclic digraph) in which all paths leave from a single "source" node to reach one or more "sink" nodes. For acyclic digraphs the possibility exists of ordering the nodes in several different *levels* on the base of their distance from the source node; for query-graphs, for which the condition 2 of section 2 holds, this fact corresponds to assigning to each level all the queries having the same number of indifference conditions in a decreasing order. Therefore the source will represent the query with the maximum number of indifference conditions, while the sinks will represent queries with no indifference condition.
In the following we propose two different approaches to the definition of a redundant CR organization by transforming general query-graphs in query-trees.
The first approach can be called the "natural splitting" since the query-tree is obtained by splitting and multiplexing all the nodes having more than one incoming edge as in the following algorithm.

A_1 - The root of the query-tree is made to coincide with the source node of the query-graph;

A_2 - At the next level, nodes having more than one incoming edge are multiplexed, together with all their outgoing edges to the lower level, as many times as the number of the input edges;

A_3 - On the graph, modified as in step A_2, step A_2 is repeated until the sink nodes are reached.

Fig. 3 shows a graphical representation of the algorithm while in Fig. 4 a complete example is carried out.
For such an approach it is interesting to evaluate how many copies of each node are produced, since this value gives a first measure of the amount of redundancy which has been introduced.
We can notice that the nodes belonging to levels 1 (the root) and 2 are never multiplexed. At level 3, each node is multiplexed as many times as the number of its "fathers". At subsequent levels, each node is multiplexed as many times as the number of its fathers having a single "grandfather" plus the number of grandfathers of the fathers having more than one grandfather and so on.
To formalize this "tongue twister" calculation let us call:

$P_s(t_i)$ - the number of fathers of node t_i having a single grandfather

$P_m(t_i)$ - the number of fathers of node t_i having multiple grand-

fathers

l - the level the node t_i belongs to

$p^k(t_i)$ - the father(s) on the k-th generation starting from (t_i)

$\#(t_i)$ - the number of copies of node (t_i)

then

$$\#(t_i) = \sum_{o^k}^{l-1} P_s\,(P_m^k\,(t_i))$$

with the obvious conventions that: $P_m^\circ(t_i) = t_i$,

$P_m^k(t_i) = 0$ if no father with multiple grandfathers exists,

$P_s(t_i) = 0$ if no father with single grandfather exists (i.e. the root),

$P_s(0) = 0$.

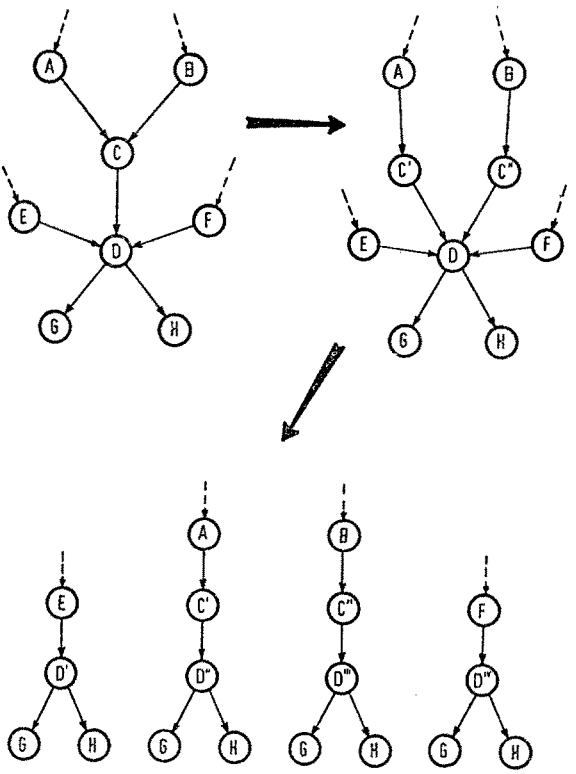


Fig. 3

A more exact definition for the redundancy of a data organization in query-oriented systems has been proposed by Ghosh [1]. Let us indicate by $|\rho(q_i)|$ the number of different records answering a query q_i, by m the number of distinct records in the file, and by m_1 the number of records in a particular file organization. The redundancy is defined

to be

$$R = \frac{m_1 - m}{m}$$

where $m = \left| \bigcup_{i=1}^{N} \rho(q_i) \right|$

In our case the value of m_1 can be recursively evaluated on the query-tree in the following way:

$$m_1 = |\rho(t_1)| \ , \ |\rho(t_i)| = |\rho(t_i) \cap (U\rho(\phi_{t_i})| + \sum |\rho(\phi_{t_i})| \quad i=1, \ldots, N$$

where ϕ_{t_i} denotes the sons of node t_i and N is the number of nodes in the query-tree.

In Fig. 4 a possible organization of the records on the storage medium is shown. To retrieve the records an index must be built giving the set of queries answered by each node of the tree, the nodes representing the initial address of a subfile, and the number of blocks occupied on the storage. In answering a particular query, a search must be made in the index for the node t_i at which the query is at the highest level, so that only the needed records are retrieved with a single access. But we can save a very large amount of time mainly in replying to a set of queries all belonging to the same subtree. In fact, choosing the node in the tree answering to the highest level query, we can retrieve the records also answering all the other queries with a single access.

The index for this kind of organization can become a rather large table depending on the nature of the query set so that the research in it can become lenghty and cumbersome. So let us examine a second approach, based on the notion of spanning tree, for transforming the general query-graph in a set of trees $\{T\} \equiv (t_1, \ldots, t_q)$.

B_1 - Take a spanning tree for the query-graph. This tree will have a root $t_1 \equiv q_1$ and will leave a set of co-trees for the remaining part of the query-graph.

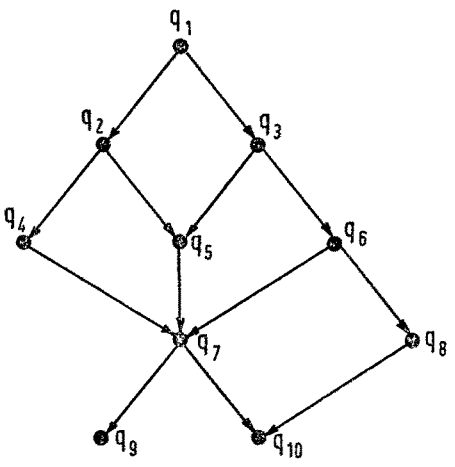
The choose of the spanning tree must be made in such a way as to minimize the number of disjoint rooted co-trees.

This condition corresponds to minimize the repetition of the already considered paths as imposed by step B_3. To build the other trees, the following steps are needed:

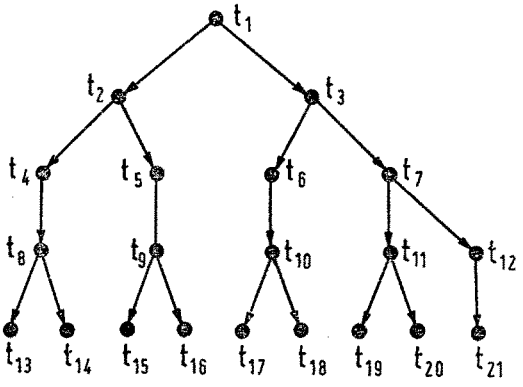
B_2 - leaving from the root of the next cotree follow the path on the cotree until the last node of the cotree itself is reached.

B_3 - if the node reached in step B_2 is a leaf, repeat step B_2. If it is not a leaf, move further on the spanning tree either until the leaves are reached or until the root of a new cotree is reached, then repeat step B_2.

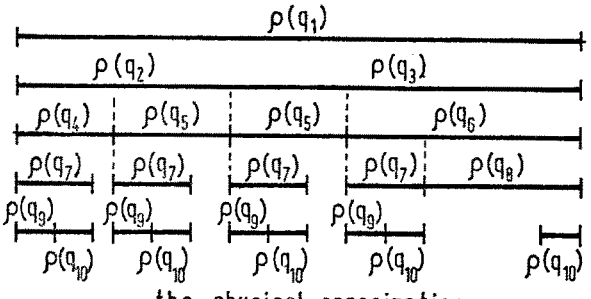
Query	Attribute values
q_1	α_1 × × × × ×
q_2	α_1 × β_1 × × ×
q_3	α_1 γ_1 × × × ×
q_4	α_1 × β_1 δ_3 × ×
q_5	α_1 γ_1 β_1 × × ×
q_6	α_1 γ_1 × × ϑ_1 ×
q_7	α_1 γ_1 β_1 δ_3 ϑ_1 ×
q_8	α_1 γ_1 β_1 × ϑ_1 ε_1
q_9	α_1 γ_1 β_1 δ_3 ϑ_1 ε_2
q_{10}	α_1 γ_1 β_1 δ_3 ϑ_1 ε_1



the query-graph



the derived query-tree



the physical organization

Fig. 4

An example of such a procedure is shown in fig. 5 for the same query-graph of fig. 4.

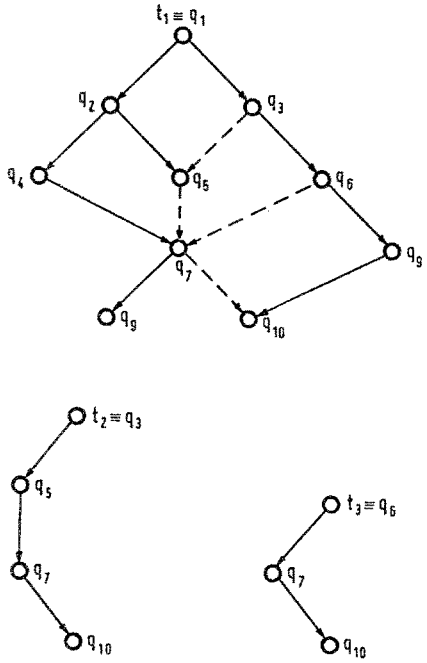


Fig. 5

For each one of the defined trees a CR organization is built on the storage medium. The redundancy is clearly increased because we must consider now

$$m_1 = \sum_{1}^{n}{}_i \; m_{1i} \; ,$$

where m_{1i} is the lenght of the subfile corresponding to the i-th tree. However we have a remarkable gain in the search lengh on the index table as we shall see later on.

4. - BUILDING THE FILE STRUCTURE

Given a covering graph built from a covering table as we saw in section 2 and the set $\{T\} \equiv \{t_1, \ldots, t_j, \ldots, t_2\}$ of the trees obtained by algorithm B in section 3, for every tree t_i we can consider a characteristic string S_i constituted by the union of all the characteristic values of the L_i queries belonging to t_i, i.e.:

$$S_i = \bigcup_{j=1, L_i} \{v_{a_{1_j}}, \ldots, v_{a_{k_j}}\}$$

As seen in section 3 for every t_i it is possible to have a CR organization constituting a subfile stored at address I_i.
An index can be built in which the associations S_i, L_i, I_i are listed for all the t_i (Fig. 6).

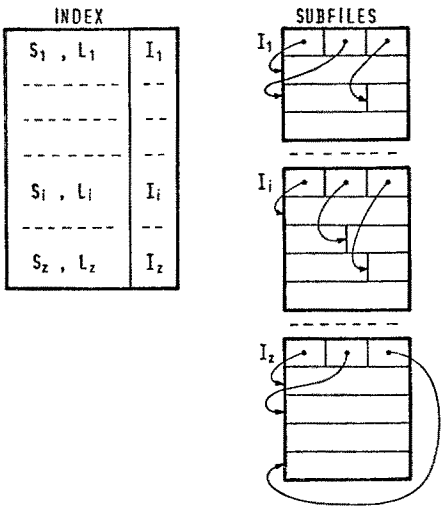


Fig. 6

The first block of the subfile, at address I_i, contains only the pointers to the beginning of the $\rho(q_k)$ (k=1, ..., L_i), while data are stored beginning from the second block.

5. - FILE MANIPULATION

Let us now examine how some operations can be performed on the proposed structure.

5.1. - SEARCHING

Let $q_k \in \{Q\}$ be a query with characteristic string

$$S_{q_k} \equiv \{v_{a_1 K}, \ldots, v_{a_N k}\}$$

If there is an S_i such that

$$S_{q_k} \cap S_i \neq \Phi \qquad\qquad i=1, \ldots, z$$

the query can be answered.

Moreover, if

$$S_{q_k} \cap S_i > S_{q_k} \cap S_j \neq \Phi \qquad j=1,\ldots, z \quad i \neq j$$

the set $\rho(q_k)$ belongs to the subfile stored at address I_i, associated in the index to the string S_i.
If

$$S_{q_k} \cap S_i \equiv S_{q_k} \cap S_j \neq \Phi \qquad j=1,\ldots, z \quad i \neq j$$

the set $\rho(q_k)$ belongs to more than one subfile. In this case the access will be made in the subfile for which L_i is the smallest.
It must be noticed that the access is directly made to the address I_i then the retrieval is sequential in the corresponding subfile.
This method can be immediately extended to a set of queries $\{Q^*\}$. In this case a characteristic string must be considered built as the union of the characteristic strings of each query of the set $\{Q^*\}$.

5.2. - UPDATING

An updating operation can consist in:

a - insertion of new records
b - deletion of existing records
c - modification of some values in existing records.

These operations can be performed in the proposed organization in the following way:

a - *Insertion*

Let ρ^* be the record to be inserted, described by the characteristic string

$$S_{\rho^*} = \{v_{a_1\rho^*}, \ldots, v_{a_N\rho^*}\}.$$

The following cases can be considered:

a.1. - $S_{\rho^*} \cap S_i = \Phi \qquad i=1, \ldots, z$

In this case the complete reorganization of the file structure seen in section 4 is necessary. Actually such a case is seldom encountered in very large files.

a.2. - $$\left.\begin{array}{l} S_{\rho^*} \cap S_i \neq \Phi \\ S_{\rho^*} \cap S_j = \Phi \end{array}\right\} \qquad 1 \leq i \leq k \leq j \leq z$$

In this case ρ^* must be inserted in all the k subfiles for which the intersection is not empty. It must be noticed that anyway the index is not affected by the operation.

b - *deletion*

A search operation must be triggered as in a.2, then the record is deleted from the k subfiles in which it was stored.

c - *modification*

Let $S_{\rho_j} = \{v_{a_1 j}, \ldots, v_{a_N j}\}$ be the characteristic string of the record ρ_j to be modified.
In all the instances for which

$$S_{\rho_j} \cap S_i \neq \phi \qquad i=1, \ldots, z$$

the modified value is substituted to the old one.
Let now $S^*_{\rho_j}$ be the modified characteristic string. A search in the index must be made until

$$S^*_{\rho_j} \cap S_j \neq \phi \qquad i \neq 1, \ldots, z$$

and an insertion operation of all the modified records must be performed while deleting the old instances.

6. - PERFORMANCE EVALUATION

The proposed data organization is suitable for applications having the following features:

- the structure of queries is unknown a-priori
- the response time is a critical parameter
- each query is answered by more than one record
- batches of queries have often to be answered at the same time
- mass storage occupation is not critical
- sequential processing of the stored data is to be made

A comparison can be made with other data structures on the base of some figures of merit such as access time, storage occupation, file manipulation and updating operation complexity, etc.

The proposed data structure belongs to the class of secondary index organizations [4], however, under certain conditions, it offers better efficiency than other structures of the same class.
Since, among the structures of this class, that having characteristics the most similar to the proposed one is the inverted list structure, we are going to compare their performance with respect to access time and manipulation complexity.
As to the access time for the inverted list structure we have:

$$T_{acc} = T_{ind} + M * T_D$$

where

T_{acc} = Access time

T_{ind} = time spent in sequentially retrieving, in the index file, all the values of the characteristic string, in extracting for each value the list of the addresses at which the records having the particular value are stored, then in intersecting all these lists.

T_D = Direct access time

$M*T_D$ = Access time to the M different addresses resulting from the intersection of the address lists.

For the proposed organization

$$T^*_{acc} = T^*_{ind} + 2\ T_D + (M^* - 1)\ T_s$$

where

T^*_{acc} = access time

T^*_{ind} = time spent in matching S_{q_k} against all of the S_i, as already seen in section 5.1., for extracting the address of the required subfile.

T_D = Direct Access time for the first block of the subfile

T_s = Sequential Access time

$(M^*-1)T_s$ = Sequential Access time for the part of the subfile concerning q_k.

We can notice that generally $M < B$, if B is the number of all the blocks in the subfile.
Under certain conditions $T^*_{acc} < T_{acc}$. In fact: $T^*_{ind} < T_{ind}$ always

$M^* < M$ if the blocking factor >1. This is a frequently verified condition.

$T_S \leq T_D$ always. The equal sign applies in the condition (statistically seldom occuring) in which next direct address in the following in the sequential organization.

The complexity of updating operations of the two structures is nearly equal. In fact in the inverted list structure every kind of updating operation requires an expensive updating of the index file. In the proposed CR organization the index file is never affected, but updating operations can require the expensive task of inserting records in a sequential file.

7. - CONCLUDING REMARKS

We have proposed a sequential data structure with secondary index with fast access properties which can be fruitfully used in applications requiring:

- a high dynamics of the data file
- a-priori unknown query structures
- multiple access keys
- sequential data processing
- low response time
- batches of queries to be simultaneously processed.

Under these requirements the performances of the proposed structure is better with respect to other structures of the same class. A comparison example has been given with inverted list structure.
It must be noticed that the method for the analysis and the implementation of the data bank structure can be completely automatized. This makes possible the use of the computer not only in the data management phase but also in the design and implementation of the data bank itself.

REFERENCES

1. - GOSH S.P. "Consecutive Storage of Relevant Records with Redundancy", Communications ACM - August 1975 pp. 464-471

2. - WAKSMAN, A., and GREEN, M.W.:
"On the consecutive retrieval property on file organization" - I E E E Trans. on Computers - C-23 1974, pp. 173-174

3. - MARTELLA G., and M.G. SAMI:
"On the problem of Query-Oriented File Organization"
XXII Rassegna Internationale Elettronica Nucleare Aerospaziale - Roma March 1975

4. - BRACCHI, G., MARTELLA, G.:
"Sistemi Generalizzati per la Gestione delle Informazioni: Le tecniche di organizzazione dei dati". Rivista di Informatica, Vol. II, n. 3, 1971, pp. 1-48.