in: Proc. Int. Conf. on Computing and Information, Toronto, May 1989, pp. 351-357

SUPPORTING OBJECT-ORIENTED PROCESSING BY REDUNDANT STORAGE STRUCTURES

Andrea Sikeler

University Kaiserslautern, Department of Computer Science,
P.O. Box 3049, D-6750 Kaiserslautern, West Germany

The development of a new generation of database management systems (DBMS) capable of supporting advanced applications has emerged as an important direction in DBMS research. Adequate modeling and efficient processing of complex objects have proven to be one of the most important issues in this context. Thus, object-oriented data models allowing for the manipulation of complex objects are proposed as an appropriate interface of such a DBMS. However, this is not sufficient. The DBMS itself has to support complex object processing by a variety of storage structures, use of tuning mechanisms, and performance enhancements transparent at the data model interface. This paper presents some considerations as to the support of complex object processing by redundant storage structures. The underlying concept is to utilize storage redundancy for retrieval and to conceal storage redundancy in case of manipulation.

1. INTRODUCTION

For some time, database system research has been concentrating on the development of a new generation of database management systems (DBMS) capable of supporting so-called non-standard application areas (e.g. CAD/CAM). These advanced applications differ from conventional business applications in a number of critical aspects including data modeling and processing as well as storage structures and access methods. As a consequence, some important design guidelines have to be considered:

- The key idea is the DBMS kernel architecture [1, 2]. The overall non-standard DBMS (NDBS) consists of a neutral DBMS kernel incorporating all application-independent data management functions and of different application layers providing application-specific support (Fig. 1).
- The data model supported by the DBMS kernel has to be object-oriented in that it allows for the retrieval and manipulation of complex objects [3]. These complex objects have to be constructed dynamically from basic objects and relationships among these. All kinds of relationship types (1:1, 1:n, n:m) should be represented in a direct way allowing for symmetric traversal and use of complex objects.
- The DBMS kernel as the implementation of the data model should support object processing by a variety of storage structures, use of tuning mechanisms, and performance enhancements transparent at the data model interface. Therefore, a clean breakup of the DBMS kernel into different layers with appropriate tasks is mandatory.

This paper is concerned with the last aspect. It presents, in the framework of the PRIMA (prototype implementation of the molecule-atom data model) project [4] some of our considerations concerning the support of complex object processing in an efficient way by maintaining redundant storage structures.

2. THE PRIMA ARCHITECTURE

The DBMS kernel PRIMA makes available at its interface the molecule-atom data model (MAD model) and its query language MQL (MAD query language). In the MAD model the objects corresponding to the above mentioned complex objects are called *molecules*. Each molecule consists of more primitive molecules or atoms and belongs to a certain molecule type defined in terms of its component types and the relationships among these types. Each *atom* is composed of an arbitrary number of attributes and belongs to a corresponding atom type. The attributes' data types can be chosen from a richer selection than in conventional data models. With respect to molecule management the most important of these types are the IDENTIFIER type and the REFERENCE type. The *IDENTIFIER type* serves as a surrogate which allows for the identification of each atom. Based on this IDENTIFIER type it is easy to define the *REFERENCE type* as a set of typed references to other atoms. However, each reference requires a corresponding "back-reference" in order to support symmetric processing.

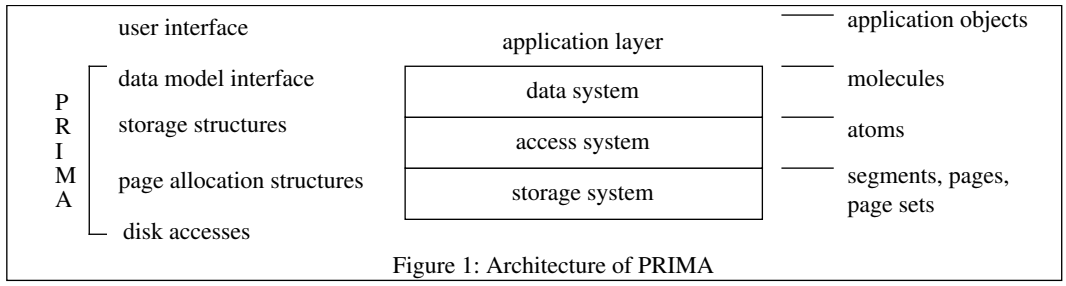

Figure 1: Architecture of PRIMA

In order to map this MAD model to external storage devices PRIMA is divided into three layers representing different levels of abstraction (Fig. 1):

- The data system dynamically builds the objects, i.e. molecules, available at the data model interface. For this purpose, the molecule-set-oriented MQL statements are transformed into lower layer programs containing series of access system calls.
- The access system provides a one-atom-at-a-time interface similar to that of the Research Storage System (RSS) of SYSTEM R [5].
- The storage system implements a set of "infinite" linear address spaces by managing the database buffer and organizing the external storage devices, thus being responsible for the data exchange between main storage and disk storage (i.e. it represents a lower level of abstraction than RSS [5]). For this purpose, the database is divided into various segments consisting of a set of logically ordered pages. All pages of a segment are of equal size (1/2, 1, 2, 4, or 8 kbytes) which is kept fixed during the lifetime of a segment. The five page sizes, however, are not sufficient when considering the mapping process performed by the access system. Therefore, *page sequences* are introduced as predefined page sets supported by physical clustering. A page sequence is a set of logically consecutive pages of a segment which contain one single object spanning these pages [6]. Additionally, the storage system provides means to handle not only such predefined page sets but also arbitrary page sets.

Throughout the rest of the paper we concentrate on the design of the access system.

3. THE BASIC VERSION OF THE ACCESS SYSTEM

Most of the storage structures supported by the access system are intended to increase performance rather than functionalism. Therefore, a basic version of the access system (BAS), comprising means to support data definition, manipulation, and "simple" retrieval, is sufficient in order to guarantee the functional behaviour of MQL. However, before we go into particulars concerning these operations, we first describe the mapping of atoms onto the different "containers" offered by the storage system.

3.1 THE MAPPING OF ATOM TYPES AND ATOMS

In the BAS, each atom type is assigned to a single segment which includes all atoms of the corresponding type. This decision essentially simplifies some of the operations, particularly the deletion of all atoms of an atom type, although the addressing concept described in section 5.1 would also permit to store atoms of different types in the same segment.

Atoms are mapped onto so-called physical records. A physical record is a byte string of variable length, which represents, in case of the BAS, exactly one atom. Physical records are, in turn, mapped onto pages and page sequences according to their current length. If a record goes beyond the page size of the segment assigned to the corresponding atom type, it is mapped onto a page sequence. Otherwise, it is mapped onto a page. A page may contain multiple records without intermediate space. Thus, modifying a record may cause movement of other records within the page. However, only the record to be modified may move to another page to a page sequence. Since page sequences have also to be searched sequentially for a physical record, only one physical record is stored in a page sequence in order to minimize search overhead.

In order to locate a physical record, a physical address indicating the appropriate segment and page or page sequence is assigned to each physical record. However, this physical address cannot be used as logical address, since with respect to the additional storage structures multiple physical records may be assigned to a single atom. Therefore, we use a special address structure maintaining all physical addresses assigned to a logical address (section 5.1). A logical address always corresponds to the IDENTIFIER attribute of an atom. The appropriate value is assigned by the access system when the atom is inserted and it is released when the atom is deleted. Modification of a logical address is not allowed.

3.2 THE OPERATIONS

The *creation of a new atom type* requires little effort within the access system. A new segment as well as a corresponding address structure are initialized. User-defined keys, which are an additional feature of the MAD model, however, raise some problems. The overhead to check for duplicates may become enormous without appropriate access path structures, such as B-trees, allowing for fast value-dependent access. Therefore, an access path structure should be initialized for each user-defined key. This, however, is only possible when the current configuration of the access system includes appropriate components implementing the desired structures (see chapter 4). In order to *delete an atom type*, the appropriate segment and address structure as well as all dependent storage structures are deleted.

Manipulation operations work on single atoms identified by their logical address. When *inserting* an atom values may be assigned to all or only selected attributes. The access system has to check whether or not values are assigned to all attributes belonging to a user-defined key and whether or not these values are unique. Hence, either the above mentioned access path structures are utilized or all atoms of the atom type are searched sequentially. Additionally, for each REFERENCE attribute it has to be checked whether or not the referenced atoms exist. For this purpose, the address structure of the referenced atom types is used. Afterwards, the logical address is determined and the corresponding physical record is constructed and stored either in a page or a page sequence. Accordingly, it is allowed to *modify* single attribute values within an atom, either as a whole or selected parts, depending on the corresponding attribute type. Again the access system has to check for user defined keys and REFERENCE attributes before the corresponding physical record is modified. If the size of a physical record changes due to the modification, other records within the same page have to be rearranged. Moreover, the record itself has to be stored in another page or even in another page sequence if it does not longer fit in its original page. *Deleting* an atom is quite simple. The corresponding physical record is erased and the other records within the same page are rearranged. Performing any manipulation operation, the access system is responsible for the automatic maintenance of the *referential integrity* defined by the REFERENCE attributes (system-enforced integrity). Therefore, each manipulation operation on a REFERENCE attribute includes implicit manipulation operations on other atoms to adjust the appropriate back REFERENCE attributes.

Retrieval operations of the BAS include direct access to a single atom identified by its logical address as well as sequential access (scan) to all atoms of a certain type, either in system-defined order (atom-type scan) or sorted following a certain sort criterion (sort scan). In accordance to the manipulation operations, it is possible to read an atom as a whole or to project single attribute values (again as a whole or selected parts). For this purpose, a so-called projection list has to be specified which determines the (parts of the) attributes to be retrieved as well as the ordering of the attributes within the result atom. In case of the scan operations additional characteristics may be specified in order to further describe the result set subsequently delivered by the scan. Both the atom-type scan and the sort scan allow for the definition of a simple search argument (SSA) which restricts the result set to atoms satisfying the specified condition. However, specifying a sort criterion and an additional start/stop condition is restricted to a sort scan, i.e. the sort criterion defines the sequence in which the atoms of the result set are delivered and the start/stop condition limits the result set to a certain range within the sort sequence defined by the sort criterion. When *directly accessing* an atom, the address structure of the appropriate atom type is utilized to determine the physical address assigned to the corresponding logical address. This physical address is passed to the storage system which delivers a page or a page sequence containing the appropriate physical record. This physical record has to be interpreted according to the specified projection list. An *atom-type scan* is performed by scanning the address structure of the corresponding atom type in order to obtain all logical addresses. For each logical address the appropriate physical record is determined and it is checked whether it satisfies the specified SSA. Is so, the result atom is built up according to the projection list. A *sort scan*, however, requires much more effort within the access system. All atoms of the appropriate atom type have to be sorted according to the specified sort criterion. Simultaneously, the SSA as well as the start/stop condition have to be evaluated. In order to successively deliver the result set it has to be stored in a temporary segment. More information about sorting is given in the following chapter.

4. EXTENSIONS TO THE ACCESS SYSTEM

Although the BAS is sufficient in order to guarantee the functionalism of MQL, it is not sufficient with respect to performance aspects. Therefore, we are introducing additional storage structures. The underlying concept is to make storage redundancy available outside the access system by offering appropriate retrieval operations, whereas in case of manipulation operations storage redundancy is concealed by the access system.

4.1 ACCESS PATH STRUCTURES

Access path structures provide appropriate means for fast value-dependent access to the atoms of a single type. However, depending on different characteristics of the access pattern one access path structure may be more efficient than another. Therefore, multiple access path structures should be supported by the access system with regard to single attribute and multiple attribute retrieval as well as spatial and temporal access. In PRIMA we decided to utilize

- B-tree [7] for single attribute access,
- grid file [8] for multiple attribute access,
- R-tree [9] for spatial access and
- no special structure for temporal access.

In both tree structures references to the appropriate atoms are maintained. These references correspond to the logical addresses of the atoms. As a consequence, an additional access to the address structure is required in order to obtain the corresponding physical addresses. However, the access path structures are isolated from the redundancy introduced by the different extensions of the access system. Moreover, moving a physical record does not affect any access path structure. On the other hand, the two-disk-access principle is an important characteristic of the grid file structure. Therefore, a grid file directly maintains physical records, although maintenance of logical addresses should be possible. As a result, redundancy is introduced, since multiple grid files may be defined for a single atom type.

We have decided to provide a uniform access path scan operation for all access path structures supported by the access system. Thus, any access path structure may be added or removed without affecting the data system (except the optimizer). As a consequence, the access path scan itself becomes more complex. In case of multiple attribute retrieval, for example, start/stop conditions and directions (ascending, descending) may be specified individually for every attribute, thus extending the scan operation.

4.2 SORT ORDERS

A sort scan is expected to be a rather frequent operation, since sorting may considerably speed up internal processing in the data system (e.g. merging-scan join). However, sorting an entire atom type, repeatedly for each sort scan, is expensive and time consuming. Therefore, a sort scan may be supported by a so-called sort order consisting of a sorted list of physical records, one for each atom of the corresponding atom type. Initially, these physical records are stored in subsequent pages according to the sort criterion. Manipulation operations, however, would require a reorganization of the whole structure. Therefore, the corresponding pages are chained. As a consequence, physical records may be added or removed at arbitrary points as pages are split or merged. Sorting itself is performed in a straightforward manner. Starting with an empty sort order an atom-type scan is initialized and each atom delivered by the scan is successively inserted into the sort order at the appropriate point. However, since this procedure is rather inefficient, different sort techniques (e.g. [10]) have to be investigated with respect to their applicability in the access system. Moreover, the extent in which existing access path structures or sort orders may be utilized has to be considered.

4.3 ATOM-CLUSTER TYPES AND ATOM CLUSTERS

The concept of *atom clusters* [11] has been introduced in order to accelerate construction of frequently used molecules, by allocating all atoms of a corresponding molecule in physical contiguity. Atom clusters, which are restricted to molecules of a non-recursive, hierarchical structure, correspond to a possibly heterogeneous atom set described by a so-called characteristic atom. This characteristic atom simply contains references to all atoms belonging to the atom cluster (Fig. 2a). Each atom cluster is mapped onto one physical record containing the characteristic atom as well as all referenced atoms (Fig. 2b). The physical record is, in turn, mapped onto a page or a page sequence depending on its current length (Fig. 2c). For this purpose, the physical record is divided into multiple subrecords, each containing all atoms of a single atom type. All subrecords are subsequently mapped onto pages. If a subrecord exceeds the free space available within a page, a new page is allocated. If a subrecord requires multiple pages, these pages are exclusively used by this subrecord. However, in order to locate an atom within an atom cluster, i.e. within a page sequence, an additional address structure (section 5.1) is required.

Performing a manipulation operation on an atom, the access system is responsible for maintaining the appropriate atom clusters, i.e. the access system has to automatically include atoms into an atom cluster, move them from one cluster into another, and delete them from a cluster depending on the corresponding manipulation operation [11].

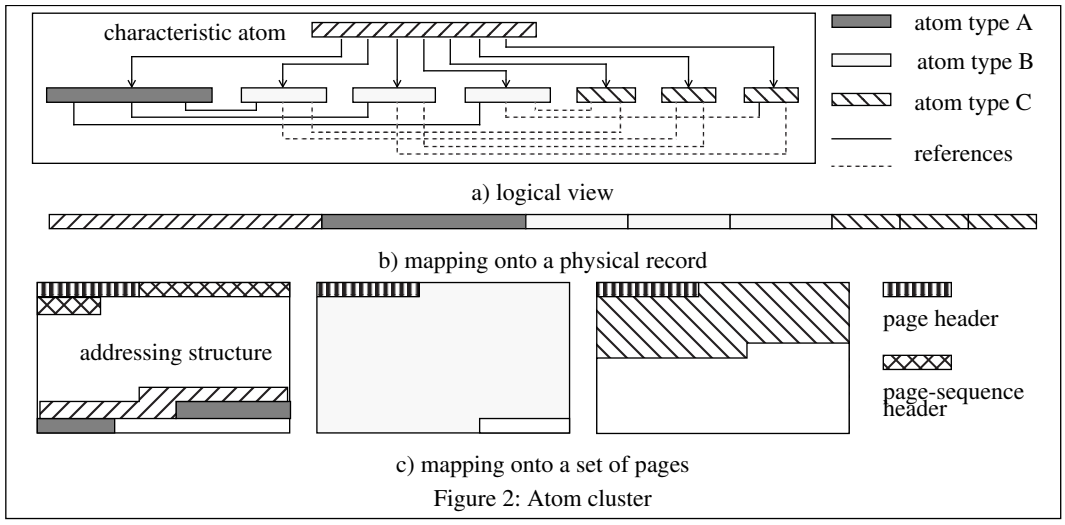

Figure 2: Atom cluster

Whereas the scans, described up until now, only support access to a homogeneous atom set belonging to one atom type, the two scans defined for atom-cluster types allow for fast access to a heterogeneous atom set across several atom types. The atom-cluster type scan delivers all characteristic atoms of an atom-cluster type in a system-defined order. Similar to all other scans, the result set of the atom-cluster type scan may be restricted by a complex search argument (CSA), which must be decidable in one pass through a single atom cluster (single scan property [6]). Subsequently, direct access to all atoms belonging to an atom cluster is possible since each characteristic atom contains the corresponding logical addresses. The atom cluster scan, however, offers another possibility for accessing the atoms of an atom cluster. It reads all atoms of a certain atom type within one single atom cluster in a system-defined order, again with the possible restriction by a simple search argument.

4.4 PARTITIONS

A further extension to the access system are partitions. A partition allows for a vertical partitioning of an atom type. Thus, frequently used attributes of an atom type may be clustered and stored independently from other attributes clustered in a similar way. As a consequence, multiple physical records, one for each attribute cluster, are assigned to a single atom. Each of these physical records consists of the appropriate attribute values as well as the IDENTIFIER attribute in order to locate the physical record within a page. A partition is automatically utilized by the access system. Therefore, no explicit retrieval operation referring to a partition is required.

5. MAINTAINING REDUNDANCY

In order to conceal the storage redundancy originating from the above extensions we have introduced the concept of a logical record (i.e. atom) provided at the access system interface and physical records stored in the "containers" offered by the storage system, i.e. each physical record represents an atom in either storage structure. As a consequence, an arbitrary number of physical records may be associated with each atom. The relation between an atom and all its associated physical records is maintained by a sophisticated address structure described in the following section. However, that is not sufficient. Especially in case of manipulation operations, new concepts have to be investigated in order to speed up a single manipulation operation.

5.1 THE ADDRESSING CONCEPT

Each *atom* is uniquely identified by its logical address which is assigned when the atom is inserted. A logical address consists of an atom-type identifier (unique within the schema) and of an atom identifier (unique within an atom type). Thus, a logical address is unique system-wide and independent of every storage structure. The concept of logical addresses is also sufficient for the addressing of *atom clusters*, since each atom cluster is described by a characteristic atom and the logical address of this characteristic atom may be utilized in order to access the atom cluster.

In order to locate a *physical record* within the "containers" offered by the storage system, the physical address assigned to each physical record consists, as already mentioned, of a segment number and of a page number or

page-sequence number indicating the page or page sequence in which the record is stored. However, an information part is required in order to distinguish between a page number and a page-sequence number. Additionally, a physical address contains an identification of the storage structure the physical record belongs to.

Mapping a logical address onto the appropriate physical addresses requires a flexible address structure, since a variable number of physical records may exist for each atom. Therefore, a so-called address list is assigned to each atom, consisting of the corresponding logical address, of a length field indicating the number of physical addresses assigned to the logical address, and of the physical addresses themselves. The access to such an address list is performed by means of an appropriate address translation method based on linear virtual hashing [12].

The mapping of a physical address onto the appropriate logical addresses is implicitly contained in each physical record. If a physical record corresponds to either an attribute cluster or a complete atom, the IDENTIFIER attribute, i.e. the logical address, resides at the beginning of the physical record. Otherwise, if a physical record corresponds to an atom cluster, the appropriate characteristic atom includes all logical addresses of the atoms belonging to the atom cluster (Fig. 2).

Moreover, each physical record assigned to an atom cluster requires an additional address structure which allows for the fast location of a single atom within the corresponding physical record. This address structure strongly depends on the current size of the physical record. If the record fits into a single page, no additional address structure is necessary, since within a page a sequential search is performed. If the record is spread over a page sequence, the address structure initially consists of a simple table indicating for each atom type the (first) page to which the appropriate subrecord is mapped (Fig. 2). In addition, each subrecord which requires multiple pages contains a further table indicating for each atom the page in which it is stored.

5.2 UPDATING STORAGE REDUNDANCY

Introducing storage redundancy serves to speed up retrieval. On the other hand, however, it slows down manipulation, since during a single manipulation operation on an atom multiple physical records and access paths may have to be altered in order to achieve consistent storage structures. Sequential manipulation of all physical records and access path structures results in a lack of efficiency which is not acceptable. Therefore, new concepts such as deferred update and concurrent update have to be investigated in more detail with respect to their applicability in PRIMA [13].

Deferred update means that during an manipulation operation on an atom initially only one of the appropriate physical records is altered. All other physical records as well as the corresponding access paths are marked as invalid. Finally, a "process" is initialized which alters the invalid structures in a deferred manner, whereas the manipulation operation itself is finished. Consequently, when performing a retrieval operation on such an invalid structure, each physical record has to be checked as to whether or not it is valid. This, however, requires some extra overhead, especially in case of access paths and sort orders, which slows down the retrieval operation.

The problem of maintaining invalid storage structures, however, is avoided by *concurrent update*. Concurrent update means that each manipulation operation on an atom invokes a number of processes which alter the appropriate physical records and access paths in parallel. The manipulation operation is finished when all processes are finished.

Nevertheless, additional investigations, in particular concerning the underlying hardware architecture [14], are still necessary in order to determine the best way to utilize this kind of parallelism (e.g. with respect to extensibility or performance).

6. SUMMARY

Advanced applications require a new generation of DBMS which has to support the adequate modeling and efficient processing of complex objects dynamically constructed from other simple or complex objects. One way to handle these complex objects in an appropriate way is a clean breakup of the DBMS into different layers with appropriate tasks. In PRIMA, for example, three layers are distinguishable:

- The data system transforms complex objects into simple objects and vice versa.
- The access system maps logical records, i.e. simple objects, onto physical records, and vice versa, thus concealing the physical representation of simple objects from the data system.
- The storage system manages the database buffer and organizes the external storage devices providing different kinds of "containers" for storing physical records.

We have discussed new concepts and implementations for storage structure support on complex objects in order to speed up various types of retrieval operations. The design of such additional storage structures includes:

- different kinds of access path structures for fast value-dependent access
- sort orders for efficient support of sequential processing according to a given sort criterion
- atom clusters for fast processing of frequently used molecules
- partitions for separating frequently used attribute clusters from those less frequently used.

However, these storage structures introduce storage redundancy to be maintained by the access system. The underlying concept is to make storage redundancy available outside the access system by offering appropriate retrieval operations (i.e. scans), whereas storage redundancy has to be concealed by the access system in case of manipulation. However, sequential manipulation of all storage structures existing for a corresponding atom results in a lack of efficiency which is not acceptable. Therefore, exploiting parallelism seems to be a natural way to speed up a single manipulation operation. In order to investigate this aspect in more detail, we are integrating the proposed storage structures into an already running prototype of PRIMA in order to obtain more experiences in utilizing parallelism in this context.

ACKNOWLEDGMENT

I would like to thank T. Härder and H. Schöning for their helpful suggestions on a earlier version of this paper which helped to improve the presentation of the important issues. Thanks are also due to I. Littler for preparing the manuscript. Furthermore, I would like to thank the anonymous referees for their fruitful comments.

REFERENCES

[1] Härder, T., Reuter, A.: Architektur von Datenbanksystemen für Non-Standard-Anwendungen, in: Proc. GI Conf. on Database Systems in Office, Engineering, and Science Environments, Karlsruhe, Informatik-Fachberichte 94 (Springer, 1985) pp. 253-286.

[2] Paul, H.-B., Schek, H.-J., Scholl, H.M., Weikum, G., Deppisch, U.: Architecture and Implementation of the Darmstadt Database Kernel System, in: Proc. ACM SIGMOD, San Francisco (1987) pp. 196-207.

[3] Batory, D.S., Buchman, A.P.: Molecular Objects, Abstract Data Types and Data Models: A Framework, in: Proc. 10th VLDB, Singapore (1984) pp. 172-184.

[4] Härder, T. (ed.): The PRIMA Project - Design and Implementation of a Non-Standard Database System, SFB 124 Research Report 26/88, University Kaiserslautern (1988).

[5] Astrahan, M.M., et al.: A History and Evaluation of SYSTEM R, in: CACM 24:10 (1981) pp. 632-646.

[6] Deppisch, U., Paul, H.-B., Schek, H.-J.: A Storage System for Complex Objects, in: Proc. Int. Workshop on Object Oriented Database Systems, Asilomar (1986) pp. 183-195.

[7] Bayer, R., McCreight, E.: Organization and Maintenance of Large Ordered Indexes, in: Acta Informatica 1 (1972) pp. 173-189.

[8] Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The Grid File: An Adaptable, Symmetric Multikey File Structure, in: ACM TODS 9:1 (1984) pp. 38-71.

[9] Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching, in: Proc. ACM SIGMOD, Boston (1984) pp. 47-57.

[10] Härder, T.: A Scan-Driven Sort Facility for a Relational Database System, in: Proc. 3rd Int. VLDB, Kyoto (1977) pp. 236-243.

[11] Schöning, H., Sikeler, A.: Cluster Mechanisms Supporting the Dynamic Construction of Complex Objects, in: Proc. 3rd Int. Conf. on Foundations of Data Organization and Algorithms, Paris (1989).

[12] Litwin, W.: Linear Hashing - A New Algorithm for Files and Table Addressing, in: Proc. Int. Conf. on Databases, Aberdeen (1980) pp. 260-276.

[13] Härder, T., Schöning, H., Sikeler, A.: Parallelism in Processing Queries on Complex Objects, in: Proc. Int. Symp. on Databases in Parallel and Distributed Systems, Austin (1988), pp. 131-143.

[14] Härder, T., Schöning, H., Sikeler, A.: Evaluation of Hardware Architectures for Parallel Execution of Complex Database Operations, in: 3rd Annual Parallel Processing Symp., Fullerton CA (1989).