PERFORMANCE COMPARISON OF INDEX STRUCTURES FOR MULTI-KEY RETRIEVAL*

Hans-Peter Kriegel
Lehrstuhl für Informatik I
Universität Würzburg
D-8700 Würzburg
West Germany

Abstract:
In this paper, we report on a performance comparison of four software implemented index structures for multi-key retrieval: the inverted file, the grid file and two variants of multidimensional B-trees. It turns out that the recently suggested structures multidimensional B-tree and grid file outperform the traditional inverted file.

1. Introduction

The problem of retrieving all the records satisfying a query involving a multiplicity of attributes, known as multi-key retrieval or associative retrieval, is a major concern in physical database organization. Physical database organization deals with the assignment of physical data records into pages of the secondary storage and the methods of retrieving the associated pages and records in response to a given query. The most commonly used method to support retrieval is an index or directory which relates combinations of attribute values to the physical records which have these value combinations. Various software methods for the construction of such indexes for different types of queries are at the disposal of the database designer, for a survey see [Kri 82 a and b] and [NHS 82].

In this paper we will report on a performance comparison of four index structures: the inverted file, the grid file and two variants of multidimensional B-trees.

For completeness sake, let us review some terminology. In the following, we consider a collection of N records which we call a file or a database. Each record, more exactly k-dimensional record, consists of an ordered k-tupel $x = (x_1,\ldots,x_k)$ of values and some associated information. x_i is called value of attribute A_i, $1 \leq i \leq k$.

Queries are categorized as follows:

(1) an exact match query specifies a value for each attribute;
(2) a partial match query specifies values for $s < k$ attributes with the remaining $k-s$ attributes unspecified;
(3) a range query specifies a range, i.e. an interval $[l_i,u_i]$, for each A_i, $1 \leq i \leq k$;
(4) a partial range query specifies a range for each of $s < k$ attributes.

* This work was done while the author was still with the Universität Dortmund, West Germany.

2. Structures selected for implementation

The author headed a project group of 12 graduate students with the working title "Performance comparison of dynamic index structures for databases". The goal of this project group was to get acquainted with the index structures available in the literature, to implement the most promising structures and to compare the implemented structures using test databases. For a survey of available index structures we refer to [Kri 82 a and b] and [NHS 82]. From the available structures the following four structures were selected for implementation: the inverted file, the multidimensional B-tree (MDBT), the k-dimensional B-tree (kB-tree) and the grid file. In the following, we will shortly present these structures.

The most obvious candidate for implementation is the inverted file since it is the only structure implemented in commercially available database systems. The inverted file is a simple generalization of single-key structures by using as many single-key structures as there are attributes. In order to answer a given query with s attributes specified, it is necessary to access s single-key structures and, further, to perform costly intersections in order to obtain the set of pointers to the records which represent answers to the query. For a more detailed description and an analysis of its average behavior we refer to [Car 75]. In order to motivate practitioners to use a structure different from inverted files we tried to optimize the implementation of the inverted file with respect to its retrieval behavior. Since it is by far too lenghty to describe the details of the implementation in this paper, let us just mention that with all the optimization the simple inverted file ends up with more lines of code than the sophisticated kB-tree. For the sake of comparability, B-trees were chosen as a basic structure for each attribute. The only further improvement in our implementation of the inverted file is to replace these B-trees by prefix B^+-trees. The obvious shortcoming of the inverted file is that with each query costly intersections of lists of answer candidates have to be performed. This is not necessary in any of the following structures.

The multidimensional B-tree (MDBT), suggested in [Sc Ou 82], was selected for implementation because of its simple design and its expected good performance. The MDBT is simply a hierarchy of B-trees. Its basic organization is depicted in figure 1. Each level of the tree corresponds to a different attribute. The values of the i-th attribute are organized as B-trees of order m_i at level i, $1 \leq i \leq k$, where m_i may vary according to the length of the values of the i-th attribute. Now each attribute value x_i has besides its LOSON and HISON pointer known from B-trees an additional EQSON pointer referring to a B-tree at level i+1 storing all different values of attribute A_{i+1} with common value x_i for attribute A_i. With the structure described so far, exact match queries can be performed efficiently. Both, partial match and range queries require sequential processing on each level. Since the level i trees are simple B-trees not supporting sequential processing this is achieved by linking the roots of level i+1 together using NEXT pointers and providing an entry point LEVEL (i+1), $0 \leq i \leq k-1$, to the beginning of each such linked list. In addition to the NEXT pointer, each level root receives two more pointers LEFT and RIGHT for supporting partial match queries. For each level tree T the LEFT pointer in the root of T points to the filial set of T's smallest key, the RIGHT pointer in its root points to the filial set of the largest key of T. For a description of the algorithms for partial match and range queries using NEXT, LEFT, RIGHT and LEVEL pointers we refer to [Sc Ou 82].

The basic assumption for the MDBT organization is that the values of an attribute are uniformly distributed within its corresponding range and are independent of the other attribute values. This assumption is made to make sure that all level trees on the same level have the same height which guarantees a maximal height of log N+k. In his PhD thesis, Christodoulakis [Chr 81] provided evidence that for most real life databases the assumptions of uniformity and

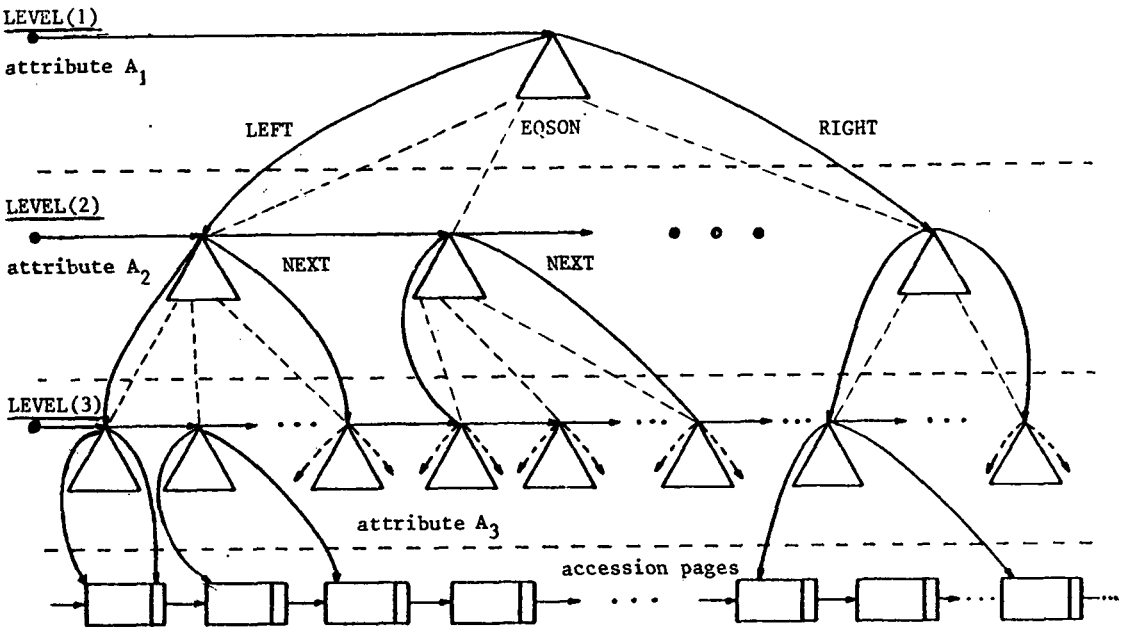

Figure 1: MDBT for 3 attributes

independence of attribute values are not satisfied. It is rather obvious that the maximal height of an MDBT storing N k-dimensional records which do not fulfill the basic assumption may be as much as O(k • logN).

This was motivation for us to suggest the k-dimensional B-tree (kB-tree), see [Gü Kr 80], which guarantees a worst case height of logN + k. If the distribution of the records in the file space is such that the level trees are of different height on the same level, the kB-tree positions the level trees in such a way that the total height of the tree never exceeds log N+k. Thus the kB-tree gracefully adapts to the distribution of the records in the file space. It follows from this basic idea that the level trees for all levels with the exception of the lowest are biased B-trees, i.e. B-trees where the leaves may have different distances from the root. This biasing is realized by a property of the keys. For details of the kB-tree see [Gü Kr 80] and for a comparison of the structure of MDBT and kB-tree see [Kri 82 b]. From the more complex structure of the kB-tree we expect worse update behavior, better retrieval behavior and more lines of code than for the MDBT. Both kB-trees and MDBT's use the same type of pointers to support partial match, range and partial range queries.

The first implementation of MDBT's and kB-trees demonstrated one basic shortcoming of multidimensional B-trees: index storage utilization may be very low and thus the number of index pages very high, primarily caused by underfilled level roots and underfilled chains of nodes. By giving up the one-to-one correspondence of a conceptual node of the structure to a physical page and accomodating several MDBT or kB-tree nodes in one page, we achieve an average storage utilization of at least 70 % in modified MDBT's and modified kB-trees,see [KAHH 84]. As a result, retrieval is considerably improved. Since storage utilization is of no concern any more, the association of attributes to the levels in the multidimensional B-trees can be purely based on the probability with which an attribute is specified in a query. Obviously the attribute with the highest probability of being specified in a query is associated to the highest level in the tree and the association is continued in decreasing order of probabilities. This improves the performance of partial match and partial range queries. For the performance comparison the

modified versions of MDBT's and kB-trees are used.

All known index structures appear to fall into one of two broad categories: those that organize the specific set of data and those that organize the embedding space from which the actual data is drawn. Inverted files using B-trees for each attribute, MDBT's and kB-trees are all search tree based structures and therefore belong to the first category. The most interesting representative of the second category for multi-key retrieval is the grid file [NHS 82] which partitions the data space according to an orthogonal grid. The grid on a k-dimensional data space is defined by k 1-dimensional arrays, called the scales. Each element of a scale represents a (k-1)-dimensional hyperplane that partitions the space into two. There is a one-to-one correspondence between the grid defined by the scales and a k-dimensional dynamic array, called the grid directory. An element of this array is a pointer to a disk block, called a data page or data bucket, which contains all data points that lie in the corresponding grid cell. To avoid low bucket occupancy, several grid cells may share a bucket. Such a set of grid cells is called a bucket region. Bucket regions are only allowed to have the shape of a k-dimensional rectangular box. These bucket regions are pairwise disjoint, together they span the data space. Fig. 2 shows the organization scheme of the grid file.

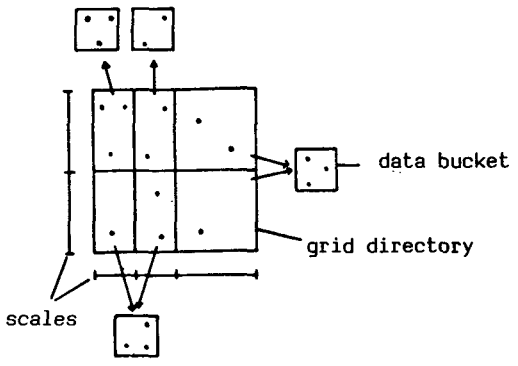

Figure 2: grid file for 2 attributes

The grid directory is likely to be large and must therefore be kept on disk, but the linear scales are small and can be kept in central memory. Therefore the grid file realizes the two-disk-access principle for exact match queries: by searching the scales, the k attribute values of a record are converted into interval indices without any disk accesses; these indices provide direct access to the correct element of the grid directory on disk, where the bucket address is located. In a second access the correct data bucket (i.e. the bucket that contains the record to be searched for, if it exists) is read from disk.

Obviously, only exact match queries which are rare in practical applications can be answered in two disk accesses. Partial match, range and partial range queries require far more disk accesses. One particular problem with these queries is the following: since many grid cells may point to the same data bucket, measures have to be taken to prevent redundant accesses to the same data bucket. For a description of the complex queries (partial match, range and partial range queries) as well as insertions and deletions we refer to [NHS 82]. For the complex queries we expect good behavior of the grid file if the assumptions of uniform distribution and independence for all attributes are fullfilled.

Since the design of the grid file leaves some important decisions to the implementor, we have to discuss how these open issues were solved in our implementation. Our implementation of the grid file neither uses the idea of a resident grid directory [NHS 82] nor the idea of postponing changes to the grid directory to some time later utilizing time stamps [JSBS 83]. For storing the grid directory, we use the conventional array organization scheme of allocating the array elements in consecutive locations. The scales are organized as attribute trees storing the splitting history of each attribute. These attribute trees allow efficient computation of the region a given bucket and prevent a modification of the grid partition as long as possible. This is very important since in our implementation the introduction of a new partitioning line may necessitate

the update of the whole grid directory. Obviously, our implementation is designed to speed up retrieval at the cost of slowing down updates.

In the recent literature, far more structures are suggested for the multi-key retrieval problem. However, the remaining structures are either not dynamic or are not suitable for implementation on external storage devices. The first three structures, inverted files, MDBT's and kB-trees use the paradigm of organizing the actual data whereas the grid file organizes the embedding space. Which of the two paradigms is better? The author performed a worst case comparison of the four structures and it turned out that, with the exception of exact match queries, the kB-tree was superior to the other structures in retrieval and update. What concerns the average case, Flajolet and Puech [Fl Pu 83] recently presented an average case analysis of the partial match query time for k-d trees versus k-d tries and thus for data organization versus space organization. They could show that for partial match queries k-d tries asymptotically outperform k-d trees, however under the assumption of uniform distribution and independence for each attribute. This result yielded additional motivation to perform a comparison of implemented structures where we can vary distributions and dependencies much easier than in an analytic model.

Before presenting the performance comparison, we want to point out the difference in the clustering effect of multidimensional B-trees (MDBT and kB-tree) and the grid file. In the multidimensional B-trees, the records that have the same value for the attributes in the highest levels of the tree are stored physically close together in the data pages. In the grid file, however, those records are clustered in the data pages which are close together in the data space. We will see in our comparisons how these different clustering effects speed up different types of queries.

3. Experimental setup

Since there were four index structures selected for implementation, the group of twelve students was divided into four subgroups of three students each. Specification, implementation and test for correctness for each structure were shifted cyclically among the subgroups, i.e. the subgroup who specified the inverted file, implemented the MDBT and tested again the inverted file for correctness. For completeness sake, let us mention that the index structures were implemented in SIMULA 67 on the Siemens 7748 mainframe of the computer science department, Universität Dortmund, West Germany. The whole index resided on disk storage using the PAM file concept. In order to compare the performance of the four selected index structures we generated seven databases each of which consisted of 20 000 to 100 000 records.To each of these databases the following files were created:

(i) an insertion file inserting 75 % of all records
(ii) an update file deleting 10 % of the records and inserting the remaining 25 % of the records
(iii) query files for all types of queries:
The query files are tailored to the specific database in order to show the behavior of the structure in dependence of record and query characteristics more clearly.

For six databases all four index structures where generated, for the largest database consisting of 100 000 records the inverted file was omitted because of its poor performance in the other databases. Thus we performed 27 comparison runs each of which was carried out in the following sequence:
Using the insertion file the major part of the index was built up and completed with the update file. Then all query files were run on this index. For each query file the following values were measured:

(i) the average CPU-time per operation
(ii) the average number of page accesses per operation
(iii) the average storage utilization for each index structure
(iv) the average number of answers for each query type.

At the end of each comparison run, the following space parameters were measured:

(i) the number of index pages (not including data pages)
(ii) the number of data pages
(iii) the number of records

Now, it is not possible to characterize the result of these comparisons in a short statement. In this paper, we will present the trend of the results considering the following database in detail.
The database contains 60 000 records with 3 attributes. In order to create a distribution different from uniform, the database consists of 3 partial databases where the values of each attribute follow a Poisson distribution and the attributes are correlated.
For the database five query files were generated. In the following, we will present the characteristics of three of these query files:

Query file 1:

(1) exact match queries: 200 successful and 20 unsuccessful queries
(2) 20 partial match queries with the first two attributes specified, i.e. the attributes which are associated to the highest two levels in the tree

(3) 20 range queries where

for attribute 1 the covered range is 20 %
for attribute 2 the covered range is 50 %
for attribute 3 the covered range is 80 %

Here covered range denotes the ratio of the size of the range specified in the query to the size of the range of all values of the attribute.

(4) 20 partial range queries where

the first two attributes are specified, and

for attribute 1 the covered range is 20 %
for attribute 2 the covered range is 50 %

This query file obviously represents a good case for the tree structures.

Query file 2:

(1) 20 partial match queries with the last two attributes specified

(2) 20 range queries where

for attribute 1 the covered range is 80 %
for attribute 2 the covered range is 50 %
for attribute 3 the covered range is 10 %

(3) 20 partial range queries where

the last two attributes are specified, and

for attribute 1 the covered range is 50 %
for attribute 2 the covered range is 20 %

This query file represents a worst case for the tree structures. It occurs as a result of a faulty design, i.e. a wrong association of attributes to the levels in the trees.

Query file 3:

(1) 20 partial match queries where the probability that an attribute is specified is

60 % for attribute 1
30 % for attribute 2
10 % for attribute 3

(2) 20 range queries where

for attribute 1 the covered range is 20 %
for attribute 2 the covered range is 45 %
for attribute 3 the covered range is 30 %

(3) 20 partial range queries where the probability that an attribute is specified is

80 % for attribute 1
70 % for attribute 2
50 % for attribute 3 and

for attribute 1 the covered range is 20 %
for attribute 2 the covered range is 70 %
for attribute 3 the covered range is 20 %

This query file as well as the remaining two query files represent an average case.

4. Results of the experiments.

In the following tables, for operations the average CPU-time per operation is given in m sec and the average number of page accesses per operation is reported. We use the following abbreviations:

EMQ for exact match query
PMQ for partial match query
RAQ for range query
PRQ for partial range query

4.1 Update

	MDBT		kB-tree		grid file		inv. file	
	CPU-time	page accesses	CPU-time	page accesses	CPU-time	page accesses	CPU-time	page accesses
insertion	336.90	5.87	321.34	5.70	51.56	2.22	718.90	60.35
deletion	93.30	6.13	94.14	6.07	62.36	5.46	413.15	21.54

4.2 Number of pages

	MDBT	kB-tree	grid file	inv. file
index pages	208	208	18	130
data pages	552	552	580	721

4.3 Average storage utilization

	MDBT	kB-tree	grid file	inv. file
index pages	76.9 %	78.9 %	100 %	70.3 %
data pages	69.9 %	69.6 %	64.0 %	69.6 %

4.4 Retrieval

a) Query file 1

	MDBT		kB-tree		grid file		inv. file	
	CPU-time	page accesses	CPU-time	page accesses	CPU-time	page accesses	CPU-time	page accesses
EMQ	53.6	3.61	51.4	3.43	31.3	1.65	330.3	13.41
PMQ	1369.8	19.80	1346.0	19.65	4431.2	24.45	6341.9	193.13
RAQ	5214.8	68.30	5149.2	68.30	10350.8	56.20	51410.8	201.34
PRQ	11277.3	311.50	11706.8	310.85	65288.5	368.75	61780.5	435.65

	EMQ	PMQ	RAQ	PRQ
average number of answers	0.9	789.6	1574.1	3951.6

b) Query file 2

	MDBT		kB-tree		grid file		inv. file	
	CPU-time	page accesses	CPU-time	page accesses	CPU-time	page accesses	CPU-time	page accesses
PMQ	540.8	10.60	521.4	10.29	363.4	10.00	250.3	13.59
RAQ	9233.0	118.80	9124.2	118.30	5341.1	36.65	11291.1	214.85
PRQ	9959.1	137.35	9806.6	135.35	8363.2	57.45	12338.1	245.65

	EMQ	PMQ	RAQ	PRQ
average number of answers	-	2.5	458.10	855.2

c) Query file 3

	MDBT		kB-tree		grid file		inv. file	
	CPU-time	page accesses	CPU-time	page accesses	CPU-time	page accesses	CPU-time	page accesses
PMQ	14235	240.20	13893	239.30	58063	182.45	22673	334.10
RAQ	4180	55.55	4102	55.20	4303	26.55	14210	220.45
PRQ	10123	155.55	9902	154.45	29327	104.95	30115	352.85

	EMQ	PMQ	RAQ	PRQ
average number of answers	-	7487.3	550.9	3731.2

d) Average over all query files

	MDBT		kB-tree		grid file		inv. file	
	CPU-time	page accesses	CPU-time	page accesses	CPU-time	page acceses	CPU-time	page accesses
EMQ	54	3.61	52	3.43	31	1.65	330	13.41
PMQ	4815	78.33	4701	77.92	16408	59.12	6135	120.78
RAQ	6209	80.90	6125	80.60	6665	39.80	19423	337.18
PRQ	8916	135.30	8746	134.17	23351	89.38	25176	287.35

5. Interpretation of the results of the experiments

What concerns the space requirement the grid file is the clear winner, the inverted file the clear looser. Considering the number of index pages, the whole grid directory is accomodated on 18 pages whereas the multidimensional B-trees need 10 times more space for the index. This results from the fact that the grid directory only stores pointers to data pages whereas the multidimensional B-trees store on top of that attribute values and further pointers. However, an almost explosion of the grid directory when inserting the last of the 60 000 records indicates that the distribution is not yet far enough from uniform to bring the partitioning process of the grid file into trouble. Storage utilization of all four structures is reasonable. From the small size of the grid directory, we expect a positive influence on the update and retrieval behavior of the grid file.

Considering insertions and deletions, the grid file is the best structure with respect to CPU-time and number of page accesses per operation, the inverted file the worst with an extreme difference of 2.22 vs. 60.35 page accesses for an insertion. The update behavior of the grid file is much better than what we expected from the primitive partitioning scheme, however the inverted file with its redundancy fulfills our negative expectations. The superiority of the grid file with respect to both, CPU-time and number of page accesses, continues for exact match queries, not surprisingly, since this type of query is perfectly supported by the grid file with its two-disk-access principle. Up to now, the kB-tree is on rank two and the MDBT on rank three.

At first glance, no clear statements are possible for the complex queries, since, depending on the query file, the kB-tree or the grid file is superior. Consider for example query file 1, where for partial (match and range) queries the first two attributes are specified. These types of queries take advantage of the clustering effect of multidimensional B-trees which, for the kB-trees, results in lower number of page accesses and roughly 25 % of the CPU-time of the grid file. Part of the high CPU-time of the grid file is due to the fact that the grid file does not support partial (match and range) queries, but treats these in the same way as range queries. If we consider query file 2 where the last two attributes are specified, the picture turns around and the grid file is the winner. However, this case can only occur as a result of a faulty design of the multidimensional B-trees, i.e. a wrong association of attributes to the levels in the tree. This faulty design cannot be excluded, but should be very unlikely. Let us therefore consider the average over all five query files. Considering CPU-time, the kB-tree is the winner, considering page accesses the grid file is the winner. For an overall rating let us make a conservative assumption: suppose that the average time to perform a physical disk access is 15 m sec (which is rather slow) and let us compute the total (response) time as CPU-time plus number of disk accesses times 15 m sec. Adding CPU-time and disk access time is conservative in view of advanced multiuser systems and intelligent disk controllers. For advanced configurations disk access time is almost negligible in a multiuser environment with a CPU utilization of at least 50 %. For the kB-tree and the grid file, the following table lists total time in m sec according to the above conservative assumptions for the average over all query files:

total time	kB-tree	grid file
PMQ	5870	17295
RAQ	7334	7262
PRQ	10759	24692

For partial match and partial range queries the kB-tree needs less than 50 % of the total time of the grid file, for range queries the kB-tree needs approximately 1 % more total time. Thus, in the considered database for complex queries the kB-tree is the best choice, for the remaining operations and the size of the index the grid file is the best choice. For databases with record distributions different from nonuniform and dependent we refer to the conclusions.

6. Conclusions

Let us summarize the results of our comparison runs. For databases where the attributes are nonuniformly distributed and correlated the grid file performs best in space requirement, insertions, deletions and exact match queries, whereas the kB-tree performs best for partial match queries, range queries and partial range queries. For a database with uniformly distributed, independent attributes, the grid file is overall winner. However, as we know from [Chr 81], these databases are unlikely in real life. The more the distribution of records in the data space differs from uniform and independent, the more kB-trees take over in the complex queries which are frequent in practical applications.

Thus the final decision will depend on the distribution of records in the data space and distribution of query types. To make this difficult decision slightly easier, let us mention the following properties of both structures: the grid file is the simpler structure, however only the kB-tree supports sequential processing and guarantees a nearly optimal behavior when access frequencies for the records are known, see [Kri 82 a and b].

With recent advances in Very Large Scale Integration (VLSI) technology, there is a trend to replace software solutions by VLSI hardware solutions. In [KMO 84], the first efficient VLSI solution to the multi-key retrieval problem is presented. When such a VLSI solution is actually built, we are ready for the most interesting performance comparison: software versus VLSI hardware. Possibly the result of this comparison will make the decision grid file or kB-tree superfluous.

Acknowledgement

First of all, I would like to thank Horst Heckhoff without whose help and system knowledge this performance comparison could not have been carried out. I am very grateful to the 12 students of the project group for spending many evenings and weekends with design, implementation and comparison runs: Martin Angst, Thomas Anhaus, Martin Ester, Ernst Hüppe, Nikolaus Hütter, Petra Kempkes, Udo Loers, Rita Mannss, Werner Schilling, Heinz-Günter Schlüter, Jutta Schulze-Bergkamen, Rainer Wurzel.

References:

[Car 75] Cárdenas, A.F., Analysis and performance of inverted data base structures, Communications of the ACM 18, 5(May 1975), 253 - 263.

[Chr 81] Christodoulakis, S., Estimating Selectivities in Data Bases, PhD Thesis, Department of Computer Science, University of Toronto, also available as Technical Report CSRG-136 (December 1981).

[Fl Pu 83] Flajolet, P. and Puech, C., Tree structures for partial match retrieval, Proc. 24th Annual Symposium on Foundations of Computer Science, 282 - 288 (1983).

[Gü Kr 80] Güting, H. and Kriegel, H.P., Multidimensional B-trees: An efficient dynamic file structure for exact match queries, Proc. 10th GI Annual Conference, Informatik-Fachberichte 33, 375 - 388, Springer, Berlin-Heidelberg-New York (1980).

[Hi Ni 83] Hinrichs, K. and Nievergelt, J., The grid file: a data structure designed to support proximity queries on spatial objects, in Proc. of the 9th Conference on Graphtheoretic Concepts in Computer Science(WG 83), Hanser Publishing Company (1983).

[JSBS 83] Joshi, S.M., Sanyal, S., Banerjee,S. and Srikumar, S., Grid files for a relational database management system, Tata Institute of Fundamental Research, Bombay, India.

[Kri 82 a] Kriegel, H.P., Dynamic tree-based index structures for associative retrieval in database systems, Habilitation Thesis, Abteilung Informatik, Universität Dortmund, also available as Technical Report No. 147, Abteilung Informatik, Universität Dortmund(1982).

[Kri 82 b] Kriegel, H.P.,
Variants of multidimensional B-trees as dynamic index structures for associative retrieval in database systems, in Proc. of the 8th Conference on Graphtheoretic Concepts in Computer Science (WG 82), 109 - 128, Hanser Publishing Company(1982).

[KAHH 84] Kriegel, H.P., Anhaus, T., Hüppe,E. and Hütter, N.,
Modified multidimensional B-trees: design, implementation and performance comparison, in preparation.

[KMO 84] Kriegel, H.P., Mannss, R. and Overmars, M.,
The inverted file tree machine: efficient multi-key retrieval for VLSI, submitted for publication.

[NHS 82] Nievergelt, J., Hinterberger, H. and Sevcik, K.C.,
The grid file: an adaptable, symmetric multi-key file structure, Report No.46, Institut für Informatik, ETH Zürich (revised July 1982), appeared in ACM Transactions on Database Systems Vol. 9, No. 1, 38 - 71 (1984).

[Sc Ou 82] Scheuermann, P. and Ouksel, M.,
Multidimensional B-trees for associative searching in database systems, Information Systems Vol. 7, No. 2, 123 - 137 (1982).