Modern B-Tree Techniques

Modern B-Tree Techniques

Goetz Graefe

Hewlett-Packard Laboratories
USA
goetz.graefe@hp.com

now
the essence of knowledge

Boston – Delft

Foundations and Trends® in Databases

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
USA
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is G. Graefe, Modern B-Tree Techniques, Foundation and Trends® in Databases, vol 3, no 4, pp 203–402, 2010

ISBN: 978-1-60198-482-1

Foundations and Trends® in Databases

Volume 3 Issue 4, 2010

Editorial Scope

Foundations and Trends® in Databases covers a breadth of topics relating to the management of large volumes of data. The journal targets the full scope of issues in data management, from theoretical foundations, to languages and modeling, to algorithms, system architecture, and applications. The list of topics below illustrates some of the intended coverage, though it is by no means exhaustive:

- Data Models and Query Languages
- Query Processing and Optimization
- Storage, Access Methods, and Indexing
- Transaction Management, Concurrency Control and Recovery
- Deductive Databases
- Parallel and Distributed Database Systems
- Database Design and Tuning
- Metadata Management
- Object Management
- Trigger Processing and Active Databases
- Data Mining and OLAP
- Approximate and Interactive Query Processing
- Data Warehousing
- Adaptive Query Processing
- Data Stream Management
- Search and Query Integration
- XML and Semi-Structured Data
- Web Services and Middleware
- Data Integration and Exchange
- Private and Secure Data Management
- Peer-to-Peer, Sensornet and Mobile Data Management
- Scientific and Spatial Data Management
- Data Brokering and Publish/Subscribe
- Data Cleaning and Information Extraction
- Probabilistic Data Management

Information for Librarians

Foundations and Trends® in Databases, 2010, Volume 3, 4 issues. ISSN paper version 1931-7883. ISSN online version 1931-7891. Also available as a combined paper and online subscription.

Foundations and Trends® in
Databases
Vol. 3, No. 4 (2010) 203–402
© 2011 G. Graefe
DOI: 10.1561/1900000028

now
the essence of knowledge

Modern B-Tree Techniques

Goetz Graefe

Hewlett-Packard Laboratories, USA, goetz.graefe@hp.com

Abstract

Invented about 40 years ago and called ubiquitous less than 10 years later, B-tree indexes have been used in a wide variety of computing systems from handheld devices to mainframes and server farms. Over the years, many techniques have been added to the basic design in order to improve efficiency or to add functionality. Examples include separation of updates to structure or contents, utility operations such as non-logged yet transactional index creation, and robust query processing such as graceful degradation during index-to-index navigation.

This survey reviews the basics of B-trees and of B-tree indexes in databases, transactional techniques and query processing techniques related to B-trees, B-tree utilities essential for database operations, and many optimizations and improvements. It is intended both as a survey and as a reference, enabling researchers to compare index innovations with advanced B-tree techniques and enabling professionals to select features, functions, and tradeoffs most appropriate for their data management challenges.

Contents

1

Introduction

Less than 10 years after Bayer and McCreight [7] introduced B-trees, and now more than a quarter century ago, Comer called B-tree indexes ubiquitous [27]. Gray and Reuter asserted that "B-trees are by far the most important access path structure in database and file systems" [59]. B-trees in various forms and variants are used in databases, information retrieval, and file systems. It could be said that the world's information is at our fingertips because of B-trees.

1.1 Perspectives on B-trees

Figure 1.1 shows a very simple B-tree with a root node and four leaf nodes. Individual records and keys within the nodes are not shown. The leaf nodes contain records with keys in disjoint key ranges. The root node contains pointers to the leaf nodes and separator keys that divide the key ranges in the leaves. If the number of leaf nodes exceeds the number of pointers and separator keys that fit in the root node, an intermediate layer of "branch" nodes is introduced. The separator keys in the root node divide key ranges covered by the branch nodes (also known as internal, intermediate, or interior nodes), and separator

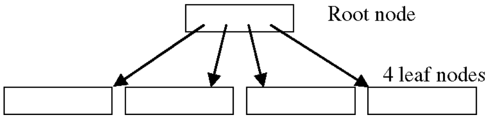

Fig. 1.1 A simple B-tree with root node and four leaf nodes.

keys in the branch nodes divide key ranges in the leaves. For very large data collections, B-trees with multiple layers of branch nodes are used. One or two branch levels are common in B-trees used as database indexes.

Complementing this "data structures perspective" on B-trees is the following "algorithms perspective." Binary search in a sorted array permits efficient search with robust performance characteristics. For example, a search among 10^9 or 2^{30} items can be accomplished with only 30 comparisons. If the array of data items is larger than memory, however, some form of paging is required, typically relying on virtual memory or on a buffer pool. It is fairly inefficient with respect to I/O, however, because for all but the last few comparisons, entire pages containing tens or hundreds of keys are fetched but only a single key is inspected. Thus, a cache might be introduced that contains the keys most frequently used during binary searches in the large array. These are the median key in the sorted array, the median of each resulting half array, the median of each resulting quarter array, etc., until the cache reaches the size of a page. In effect, the root of a B-tree is this cache, with some flexibility added in order to enable array sizes that are not powers of two as well as efficient insertions and deletions. If the keys in the root page cannot divide the original large array into sub-arrays smaller than a single page, keys of each sub-array are cached, forming branch levels between the root page and page-sized sub-arrays.

B-tree indexes perform very well for a wide variety of operations that are required in information retrieval and database management, even if some other index structure is faster for some individual index operations. Perhaps the "B" in their name "B-trees" should stand for their balanced performance across queries, updates, and utilities. Queries include exact-match queries ("=" and "in" predicates), range queries ("<" and "between" predicates), and full scans, with sorted output if required. Updates include insertion, deletion, modifications of existing data associated with a specific key value, and "bulk" variants of those operations, for example bulk loading new information and purging out-of-date records. Utilities include creation and removal of entire indexes, defragmentation, and consistency checks. For all of those operations, including incremental and online variants of the utilities, B-trees also enable efficient concurrency control and recovery.

1.2 Purpose and Scope

Many students, researchers, and professionals know the basic facts about B-tree indexes. Basic knowledge includes their organization in nodes including one root and many leaves, the uniform distance between root and leaves, their logarithmic height and logarithmic search effort, and their efficiency during insertions and deletions. This survey briefly reviews the basics of B-tree indexes but assumes that the reader is interested in more detailed and more complete information about modern B-tree techniques.

Commonly held knowledge often falls short when it comes to deeper topics such as concurrency control and recovery or to practical topics such as incremental bulk loading and structural consistency checking. The same is true about the many ways in which B-trees assist in query processing, e.g., in relational databases. The goal here is to make such knowledge readily available as a survey and as a reference for the advanced student or professional.

The present survey goes beyond the "classic" B-tree references [7, 8, 27, 59] in multiple ways. First, more recent techniques are covered, both research ideas and proven implementation techniques. Whereas the first twenty years of B-tree improvements are covered in those references, the last twenty years are not. Second, in addition to core data structure and algorithms, the present survey also discusses their usage, for example in query processing and in efficient update plans. Finally, auxiliary algorithms are covered, for example defragmentation and consistency checks.

During the time since their invention, the basic design of B-trees has been improved upon in many ways. These improvements pertain to additional levels in the memory hierarchy such as CPU caches, to multi-dimensional data and multi-dimensional queries, to concurrency control techniques such as multi-level locking and key range locking, to utilities such as online index creation, and to many more aspects of B-trees. Another goal here is to gather many of these improvements and techniques in a single document.

The focus and primary context of this survey are B-tree indexes in database management systems, primarily in relational databases. This is reflected in many specific explanations, examples, and arguments. Nonetheless, many of the techniques are readily applicable or at least transferable to other possible application domains of B-trees, in particular to information retrieval [83], file systems [71], and "No SQL" databases and key-value stores recently popularized for web services and cloud computing [21, 29].

A survey of techniques cannot provide a comprehensive performance evaluation or immediate implementation guidance. The reader still must choose what techniques are required or appropriate for specific environments and requirements. Issues to consider include the expected data size and workload, the anticipated hardware and its memory hierarchy, expected reliability requirements, degree of parallelism and needs for concurrency control, the supported data model and query patterns, etc.

1.3 New Hardware

Flash memory, flash devices, and other solid state storage technology are about to change the memory hierarchy in computer systems in general and in data management in particular. For example, most current software assumes two levels in the memory hierarchy, namely RAM and disk, whereas any further levels such as CPU caches and disk caches are hidden by hardware and its embedded control software. Flash memory might also remain hidden, perhaps as large and fast virtual memory or as fast disk storage. The more likely design for databases, however, seems to be explicit modeling of a memory hierarchy with three or even more levels. Not only algorithms such as external merge sort but also storage structures such as B-tree indexes will need a re-design and perhaps a re-implementation.

Among other effects, flash devices with their very fast access latency are about to change database query processing. They likely will shift the break-even point toward query execution plans based on index-to-index navigation, away from large scans and large set operations such as sort and hash join. With more index-to-index navigation, tuning the set of indexes including automatic incremental index creation, growth, optimization, etc. will come more into focus in future database engines.

As much as solid state storage will change tradeoffs and optimizations for data structures and access algorithms, many-core processors will change tradeoffs and optimizations for concurrency control and recovery. High degrees of concurrency can be enabled only by appropriate definitions of consistent states and of transaction boundaries, and recovery techniques for individual transactions and for the system state must support them. These consistent intermediate states must be defined for each kind of index and data structure, and B-trees will likely be first index structure for which such techniques are implemented in production-ready database systems, file systems, and key-value stores.

In spite of future changes for databases and indexes on flash devices and other solid state storage technology, the present survey often mentions tradeoffs or design choices appropriate for traditional disk drives, because much of the presently known and implemented techniques have been invented and designed in this context. The goal is to provide comprehensive background knowledge about B-trees for those researching and implementing techniques appropriate for the new types of storage.

1.4 Overview

The next section (Section 2) sets out the basics as they may be found in a college level text book. The following sections cover implementation techniques for mature database management products. Their topics are implementation techniques for data structures and algorithms

(Section 3), transactional techniques (Section 4), query processing using B-trees (Section 5), utility operations specific to B-tree indexes (Section 6), and B-trees with advanced key structures (Section 7). These sections might be more suitable for an advanced course on data management implementation techniques and for a professional developer desiring in-depth knowledge about B-tree indexes.

References

[1] V. N. Anh and A. Moffat, "Index compression using 64-bit words," *Software: Practice and Experience*, vol. 40, no. 2, pp. 131–147, 2010.

[2] G. Antoshenkov, D. B. Lomet, and J. Murray, "Order-preserving compression," *International Conference on Data Engineering*, pp. 655–663, 1996.

[3] R. Avnur and J. M. Hellerstein, "Eddies: Continuously adaptive query processing," *Special Interest Group on Management of Data*, pp. 261–272, 2000.

[4] S. Bächle and T. Härder, "The real performance drivers behind XML lock protocols," *DEXA*, pp. 38–52, 2009.

[5] L. N. Bairavasundaram, M. Rungta, N. Agrawal, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. M. Swift, "Analyzing the effects of disk-pointer corruption," *Dependable Systems and Networks*, pp. 502–511, 2008.

[6] R. Bayer, "The universal B-Tree for multidimensional indexing: General concepts," *World Wide Computing and its Applications*, pp. 198–209, 1997.

[7] R. Bayer and E. M. McCreight, "Organization and maintenance of large ordered indexes," *SIGFIDET Workshop*, pp. 107–141, 1970.

[8] R. Bayer and E. M. McCreight, "Organization and maintenance of large ordered indexes," *Acta Informatica*, vol. 1, pp. 173–189, 1972.

[9] R. Bayer and M. Schkolnick, "Concurrency of operations on B-trees," *Acta Informatica*, vol. 9, pp. 1–21, 1977.

[10] R. Bayer and K. Unterauer, "Prefix B-trees," *ACM Transactions on Database Systems*, vol. 2, no. 1, pp. 11–26, 1977.

[11] M. A. Bender, E. D. Demaine, and M. Farach-Colton, "Cache-oblivious B-trees," *SIAM Journal of Computing (SIAMCOMP)*, vol. 35, no. 2, pp. 341–358, 2005.

[12] M. A. Bender and H. Hu, "An adaptive packed-memory array," *ACM Transactions on Database Systems*, vol. 32, no. 4, 2007.

[13] H. Berenson, P. A. Bernstein, J. Gray, J. Melton, E. J. O'Neil, and P. E. O'Neil, "A critique of ANSI SQL isolation levels," *Special Interest Group on Management of Data*, pp. 1–10, 1995.

[14] P. A. Bernstein and D.-M. W. Chiu, "Using semi-joins to solve relational queries," *Journal of the ACM*, vol. 28, no. 1, pp. 25–40, 1981.

[15] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[16] D. Bitton and D. J. DeWitt, "Duplicate record elimination in large data files," *ACM Transactions on Database Systems*, vol. 8, no. 2, pp. 255–265, 1983.

[17] P. A. Boncz, M. L. Kersten, and S. Manegold, "Breaking the memory wall in MonetDB," *Communications of the ACM*, vol. 51, no. 12, pp. 77–85, 2008.

[18] M. J. Carey, D. J. DeWitt, J. E. Richardson, and E. J. Shekita, "Object and file management in the EXODUS extensible database system," *International Journal on Very Large Data Bases*, pp. 91–100, 1986.

[19] M. J. Carey, D. J. DeWitt, J. E. Richardson, and E. J. Shekita, "Storage management for objects in EXODUS," in *Object-Oriented Concepts, Databases, and Applications*, (W. Kim and F. H. Lochovsky, eds.), ACM Press and Addison-Wesley, 1989.

[20] M. J. Carey, E. J. Shekita, G. Lapis, B. G. Lindsay, and J. McPherson, "An incremental join attachment for Starburst," *International Journal on Very Large Data Bases*, pp. 662–673, 1990.

[21] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM on Theoretical Computer Science*, vol. 26, no. 2, 2008.

[22] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, "NiagaraCQ: A scalable continuous query system for internet databases," *Special Interest Group on Management of Data*, pp. 379–390, 2000.

[23] L. Chen, R. Choubey, and E. A. Rundensteiner, "Merging R-trees: Efficient strategies for local bulk insertion," *GeoInformatica*, vol. 6, no. 1, pp. 7–34, 2002.

[24] S. Chen, P. B. Gibbons, T. C. Mowry, and G. Valentin, "Fractal prefetching B$^+$-Trees: Optimizing both cache and disk performance," *Special Interest Group on Management of Data*, pp. 157–168, 2002.

[25] J. Cheng, D. Haderle, R. Hedges, B. R. Iyer, T. Messinger, C. Mohan, and Y. Wang, "An efficient hybrid join algorithm: A DB2 prototype," *International Conference on Data Engineering*, pp. 171–180, 1991.

[26] H.-T. Chou and D. J. DeWitt, "An evaluation of buffer management strategies for relational database systems," *International Journal on Very Large Data Bases*, pp. 127–141, 1985.

[27] D. Comer, "The ubiquitous B-tree," *ACM Computing Surveys*, vol. 11, no. 2, pp. 121–137, 1979.

[28] W. M. Conner, "Offset value coding," *IBM Technical Disclosure Bulletin*, vol. 20, no. 7, pp. 2832–2837, 1977.

[29] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: Amazon’s highly available key-value store,” *Symposium on Operating Systems Principles*, pp. 205–220, 2007.

[30] D. J. DeWitt, J. F. Naughton, and J. Burger, “Nested loops revisited,” *Parallel and distributed Information Systems*, pp. 230–242, 1993.

[31] K. P. Eswaran, J. Gray, R. A. Lorie, and I. L. Traiger, “The notions of consistency and predicate locks in a database system,” *Communications of ACM*, vol. 19, no. 11, pp. 624–633, 1976.

[32] A. Fekete, D. Liarokapis, E. J. O’Neil, P. E. O’Neil, and D. Shasha, “Making snapshot isolation serializable,” *ACM Transactions on Database Systems*, vol. 30, no. 2, pp. 492–528, 2005.

[33] P. M. Fernandez, “Red Brick warehouse: A read-mostly RDBMS for open SMP platforms,” *Special Interest Group on Management of Data*, p. 492, 1994.

[34] C. Freedman blog of October 07, 2008, retrieved August 16, 2011, at http://blogs.msdn.com/craigfr/archive/2008/10/07/random-prefetching.aspx.

[35] P. Gassner, G. M. Lohman, K. B. Schiefer, and Y. Wang, “Query optimization in the IBM DB2 family,” *IEEE Data Engineerring on Bulletin*, vol. 16, no. 4, pp. 4–18, 1993.

[36] G. H. Gonnet, L. D. Rogers, and J. A. George, “An algorithmic and complexity analysis of interpolation search,” *Acta Informatica*, vol. 13, pp. 39–52, 1980.

[37] N. K. Govindaraju, J. Gray, R. Kumar, and D. Manocha, “GPUTeraSort: High performance graphics co-processor sorting for large database management,” *Special Interest Group on Management of Data*, pp. 325–336, 2006.

[38] G. Graefe, “Options in physical database design,” *Special Interest Group on Management of Data Record*, vol. 22, no. 3, pp. 76–83, 1993.

[39] G. Graefe, “Query evaluation techniques for large databases,” *ACM Computing Surveys*, vol. 25, no. 2, pp. 73–170, 1993.

[40] G. Graefe, “Iterators, schedulers, and distributed-memory parallelism,” *Software: Practice and Experience*, vol. 26, no. 4, pp. 427–452, 1996.

[41] G. Graefe, “Per-Åke Larson: B-tree indexes and CPU caches,” *International Conference on Data Engineering*, pp. 349–358, 2001.

[42] G. Graefe, “Executing nested queries,” *Database Systems for Business, Technology and Web*, pp. 58–77, 2003.

[43] G. Graefe, “Sorting and indexing with partitioned B-Trees,” *Classless Inter Domain Routing*, 2003.

[44] G. Graefe, “Write-optimized B-trees,” *International Journal on Very Large Data Bases*, pp. 672–683, 2004.

[45] G. Graefe, “B-tree indexes, interpolation search, and skew,” *DaMoN*, p. 5, 2006.

[46] G. Graefe, “Implementing sorting in database systems,” *ACM Computing Surveys*, vol. 38, no. 3, 2006.

[47] G. Graefe, “Efficient columnar storage in B-trees,” *Special Interest Group on Management of Data Record*, vol. 36, no. 1, pp. 3–6, 2007.

[48] G. Graefe, “Hierarchical locking in B-tree indexes,” *Database Systems for Business, Technology and Web*, pp. 18–42, 2007.

[49] G. Graefe, "Master-detail clustering using merged indexes," *Informatik Forschung und Entwicklung*, vol. 21, no. 3–4, pp. 127–145, 2007.

[50] G. Graefe, "The five-minute rule 20 years later and how flash memory changes the rules," *Communications of the ACM*, vol. 52, no. 7, pp. 48–59, 2009.

[51] G. Graefe, "A survey of B-tree locking techniques," *ACM Transactions on Database Systems*, vol. 35, no. 3, 2010.

[52] G. Graefe, R. Bunker, and S. Cooper, "Hash joins and hash teams in Microsoft SQL server," *International Journal on Very Large Data Bases*, pp. 86–97, 1998.

[53] G. Graefe and H. A. Kuno, "Self-selecting, self-tuning, incrementally optimized indexes," *Extending Database Technology*, pp. 371–381, 2010.

[54] G. Graefe and R. Stonecipher, "Efficient verification of B-tree integrity," *Database Systems for Business, Technology and Web*, pp. 27–46, 2009.

[55] G. Graefe and M. J. Zwilling, "Transaction support for indexed views," *Special Interest Group on Management of Data*, pp. 323–334, 2004. (Extended version: Hewlett-Packard Laboratories technical report HPL-2011-16.).

[56] J. Gray, "Notes on data base operating systems," in *Operating System — An Advanced Course. Lecture Notes in Computer Science #60*, (R. Bayer, R. M. Graham, and G. Seegmüller, eds.), Berlin Heidelberg New York: Springer-Verlag, 1978.

[57] J. Gray and G. Graefe, "The five-minute rule ten years later, and other computer storage rules of thumb," *Special Interest Group on Management of Data Record*, vol. 26, no. 4, pp. 63–68, 1997.

[58] J. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger, "Granularity of locks and degrees of consistency in a shared data base," in *IFIP Working Conference on Modelling in Data Base Management Systems*, pp. 365–394, 1976.

[59] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993.

[60] J. Gryz, K. B. Schiefer, J. Zheng, and C. Zuzarte, "Discovery and application of check constraints in DB2," *International Conference on Data Engineering*, pp. 551–556, 2001.

[61] H. S. Gunawi, C. Rubio-González, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and B. Liblit, "EIO: Error handling is occasionally correct," *FAST*, pp. 207–222, 2008.

[62] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *Special Interest Group on Management of Data*, pp. 47–57, 1984.

[63] L. M. Haas, M. J. Carey, M. Livny, and A. Shukla, "Seeking the truth about ad hoc join costs," *VLDB Journal*, vol. 6, no. 3, pp. 241–256, 1997.

[64] R. A. Hankins and J. M. Patel, "Effect of node size on the performance of cache-conscious B^+-trees," *SIGMETRICS*, pp. 283–294, 2003.

[65] T. Härder, "Implementierung von Zugriffspfaden durch Bitlisten," *GI Jahrestagung*, pp. 379–393, 1975.

[66] T. Härder, "Implementing a generalized access path structure for a relational database system," *ACM Transactions on Database Systems*, vol. 3, no. 3, pp. 285–298, 1978.

[67] T. Härder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Computing Surveys*, vol. 15, no. 4, pp. 287–317, 1983.

[68] G. Held and M. Stonebraker, “B-trees re-examined,” *Communications of the ACM*, vol. 21, no. 2, pp. 139–143, 1978.

[69] A. L. Holloway, V. Raman, G. Swart, and D. J. DeWitt, “How to barter bits for chronons: Compression and bandwidth trade offs for database scans,” *Special Interest Group on Management of Data*, pp. 389–400, 2007.

[70] W. W. Hsu and A. J. Smith, “The performance impact of I/O optimizations and disk improvements,” *IBM Journal of Research and Development*, vol. 48, no. 2, pp. 255–289, 2004.

[71] http://en.wikipedia.org/wiki/Btrfs, retrieved December 6, 2009.

[72] B. R. Iyer, “Hardware assisted sorting in IBM’s DB2 DBMS,” *COMAD*, 2005. (Hyderabad).

[73] I. Jaluta, S. Sippu, and E. Soisalon-Soininen, “Concurrency control and recovery for balanced B-link trees,” *International Journal on Very Large Data Bases Journal*, vol. 14, no. 2, pp. 257–277, 2005.

[74] C. Jermaine, A. Datta, and E. Omiecinski, “A novel index supporting high volume data warehouse insertion,” *International Journal on Very Large Data Bases*, pp. 235–246, 1999.

[75] T. Johnson and D. Shasha, “Utilization of B-trees with inserts, deletes and modifies,” *Principles of Database Systems*, pp. 235–246, 1989.

[76] J. R. Jordan, J. Banerjee, and R. B. Batman, “Precision locks,” *Special Interest Group on Management of Data*, pp. 143–147, 1981.

[77] R. Kimball, *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley, 1996.

[78] R. Kooi, “The optimization of queries in relational databases,” Ph.D. thesis, Case Western Reserve University, 1980.

[79] H. F. Korth, “Locking primitives in a database system,” *Journal of ACM*, vol. 30, no. 1, pp. 55–79, 1983.

[80] K. Küspert, “Fehlererkennung und Fehlerbehandlung in Speicherungsstrukturen von Datenbanksystemen,” in *Informatik Fachberichte*, vol. 99, Springer, 1985.

[81] P. L. Lehman and S. B. Yao, “Efficient locking for concurrent operations on B-trees,” *ACM Transactions on Database Systems*, vol. 6, no. 4, pp. 650–670, 1981.

[82] H. Leslie, R. Jain, D. Birdsall, and H. Yaghmai, “Efficient search of multidimensional B-trees,” *International Journal on Very Large Data Bases*, pp. 710–719, 1995.

[83] N. Lester, A. Moffat, and J. Zobel, “Efficient online index construction for text databases,” *ACM Transactions on Database Systems*, vol. 33, no. 3, 2008.

[84] Q. Li, M. Shao, V. Markl, K. S. Beyer, L. S. Colby, and G. M. Lohman, “Adaptively reordering joins during query execution,” *International Conference on Data Engineering*, pp. 26–35, 2007.

[85] D. B. Lomet, “Key range locking strategies for improved concurrency,” *International Journal on Very Large Data Bases*, pp. 655–664, 1993.

[86] D. B. Lomet, “B-tree page size when caching is considered,” *Special Interest Group on Management of Data Record*, vol. 27, no. 3, pp. 28–32, 1998.

[87] D. B. Lomet, "The evolution of effective B-tree page organization and techniques: A personal account," *Special Interest Group on Management of Data Record*, vol. 30, no. 3, pp. 64–69, 2001.

[88] D. B. Lomet, "Simple, robust and highly concurrent B-trees with node deletion," *International Conference on Data Engineering*, pp. 18–28, 2004.

[89] D. B. Lomet and M. R. Tuttle, "Redo recovery after system crashes," *International Journal on Very Large Data Bases*, pp. 457–468, 1995.

[90] P. McJones (ed.), "The 1995 SQL reunion: People, projects, and politics," Digital Systems Research Center, Technical Note 1997-018, Palo Alto, CA. Also http://www.mcjones.org/System_R.

[91] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," *FAST*, 2003.

[92] K. Mehlhorn, *Data Structures and Algorithms 1: Sorting and Searching*. Springer, 1984.

[93] C. Mohan, "ARIES/KVL: A key-value locking method for concurrency control of multiaction transactions operating on B-tree indexes," *International Journal on Very Large Data Bases*, pp. 392–405, 1990.

[94] C. Mohan, "Disk read-write optimizations and data integrity in transaction systems using write-ahead logging," *International Conference on Data Engineering*, pp. 324–331, 1995.

[95] C. Mohan, D. J. Haderle, B. G. Lindsay, H. Pirahesh, and P. M. Schwarz, "ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging," *ACM Transactions on Database Systems*, vol. 17, no. 1, pp. 94–162, 1992.

[96] C. Mohan, D. J. Haderle, Y. Wang, and J. M. Cheng, "Single table access using multiple indexes: Optimization, execution, and concurrency control techniques," *Extending Database Technology*, pp. 29–43, 1990.

[97] C. Mohan and F. E. Levine, "ARIES/IM: An efficient and high concurrency index management method using write-ahead logging," *Special Interest Group on Management of Data*, pp. 371–380, 1992.

[98] C. Mohan and I. Narang, "Algorithms for creating indexes for very large tables without quiescing updates," *Special Interest Group on Management of Data*, pp. 361–370, 1992.

[99] Y. Mond and Y. Raz, "Concurrency control in B^+-trees databases using preparatory operations," *International Journal on Very Large Data Bases*, pp. 331–334, 1985.

[100] P. Muth, P. E. O'Neil, A. Pick, and G. Weikum, "The LHAM log-structured history data access method," *International Journal on Very Large Data Bases Journal*, vol. 8, no. 3–4, pp. 199–221, 2000.

[101] C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D. B. Lomet, "AlphaSort: A cache-sensitive parallel external sort," *International Journal on Very Large Data Bases Journal*, vol. 4, no. 4, pp. 603–627, 1995.

[102] E. J. O'Neil, P. E. O'Neil, and K. Wu, "Bitmap index design choices and their performance implications," *IDEAS*, pp. 72–84, 2007.

[103] P. E. O'Neil, "Model 204 architecture and performance," *HPTS*, pp. 40–59, 1987.

[104] P. E. O'Neil, "The SB-tree: An index-sequential structure for high-performance sequential access," *Acta Informatica*, vol. 29, no. 3, pp. 241–265, 1992.

[105] P. E. O'Neil and G. Graefe, "Multi-table joins through bitmapped join indices," *Special Interest Group on Management of Data Record*, vol. 24, no. 3, pp. 8–11, 1995.

[106] J. A. Orenstein, "Spatial query processing in an object-oriented database system," *Special Interest Group on Management of Data*, pp. 326–336, 1986.

[107] Y. Perl, A. Itai, and H. Avni, "Interpolation search — a Log Log N search," *Communications of the ACM*, vol. 21, no. 7, pp. 550–553, 1978.

[108] V. Raman and G. Swart, "How to wring a table dry: Entropy compression of relations and querying of compressed relations," *International Journal on Very Large Data*, pp. 858–869, 2006.

[109] F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, and R. Bayer, "Integrating the UB-tree into a database system kernel," *International Journal on Very Large Data Bases*, pp. 263–272, 2000.

[110] J. Rao and K. A. Ross, "Making B^+-trees cache conscious in main memory," *Special Interest Group on Management of Data*, pp. 475–486, 2000.

[111] G. Ray, J. R. Haritsa, and S. Seshadri, "Database compression: A performance enhancement tool," *COMAD*, 1995.

[112] D. Rinfret, P. E. O'Neil, and E. J. O'Neil, "Bit-sliced index arithmetic," *Special Interest Group on Management of Data*, pp. 47–57, 2001.

[113] C. M. Saracco and C. J. Bontempo, *Getting a Lock on Integrity and Concurrency.* Database Programming & Design, 1997.

[114] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: A large-scale field study," *SIGMETRICS/Performance*, pp. 193–204, 2009.

[115] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," *Special Interest Group on Management of Data*, pp. 23–34, 1979.

[116] S. Sen and R. E. Tarjan, "Deletion without rebalancing in multiway search trees," *ISAAC*, pp. 832–841, 2009.

[117] D. G. Severance and G. M. Lohman, "Differential files: Their application to the maintenance of large databases," *ACM Transactions on Database Systems*, vol. 1, no. 3, pp. 256–267, 1976.

[118] R. C. Singleton, "Algorithm 347: An efficient algorithm for sorting with minimal storage," *Communications of the ACM*, vol. 12, no. 3, pp. 185–186, 1969.

[119] V. Srinivasan and M. J. Carey, "Performance of on-line index construction algorithms," *Extending Database Technology*, pp. 293–309, 1992.

[120] M. Stonebraker, "Operating system support for database management," *Communications of the ACM*, vol. 24, no. 7, pp. 412–418, 1981.

[121] M. Stonebraker, "Technical perspective — one size fits all: An idea whose time has come and gone," *Communications of the ACM*, vol. 51, no. 12, p. 76, 2008.

[122] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik, "C-store: A column-oriented DBMS," *International Journal on Very Large Data Bases*, pp. 553–564, 2005.

[123] P. Valduriez, "Join indices," *ACM Transactions on Database Systems*, vol. 12, no. 2, pp. 218–246, 1987.

[124] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 2nd ed., 1999.

[125] K. Wu, E. J. Otoo, and A. Shoshani, "On the performance of bitmap indices for high cardinality attributes," *International Journal on Very Large Data Bases*, pp. 24–35, 2004.

[126] K. Wu, E. J. Otoo, and A. Shoshani, "Optimizing bitmap indices with efficient compression," *ACM Transactions on Database Systems*, vol. 31, no. 1, pp. 1–38, 2006.

[127] A. Zandi, B. Iyer, and G. Langdon, "Sort order preserving data compression for extended alphabets," *Data Compression Conference*, pp. 330–339, 1993.

[128] J. Zhou and K. A. Ross, "Buffering accesses to memory-resident index structures," *International Journal on Very Large Data Bases*, pp. 405–416, 2003.

[129] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Computing Surveys*, vol. 38, no. 2, 2006.

[130] C. Zou and B. Salzberg, "Safely and efficiently updating references during on-line reorganization," *International Journal on Very Large Data Bases*, pp. 512–522, 1998.

[131] M. Zukowski, S. Héman, N. Nes, and P. A. Boncz, "Super-scalar RAM-CPU cache compression," *International Conference on Data Engineering*, p. 59, 2006.

[132] M. Zukowski, S. Héman, N. Nes, and P. A. Boncz, "Cooperative scans: Dynamic bandwidth sharing in a DBMS," *International Journal on Very Large Data Bases*, pp. 723–734, 2007.