The Design and Implementation of Modern Column-Oriented Database Systems

Daniel Abadi
Yale University
dna@cs.yale.edu

Peter Boncz
CWI
P.Boncz@cwi.nl

Stavros Harizopoulos
Amiato, Inc.
stavros@amiato.com

Stratos Idreos
Harvard University
stratos@seas.harvard.edu

Samuel Madden
MIT CSAIL
madden@csail.mit.edu

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
United States
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

D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos and S. Madden. *The Design and Implementation of Modern Column-Oriented Database Systems*. Foundations and Trends® in Databases, vol. 5, no. 3, pp. 197–280, 2012.

This Foundations and Trends® issue was typeset in LaTeX using a class file designed by Neal Parikh. Printed on acid-free paper.

ISBN: 978-1-60198-755-6

Foundations and Trends® in Databases
Volume 5, Issue 3, 2012

Editorial Scope

Topics

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, 2012, Volume 5, 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. 5, No. 3 (2012) 197–280
© 2013 D. Abadi, P. Boncz, S. Harizopoulos,
S. Idreos and S. Madden
DOI: 10.1561/1900000024

The Design and Implementation of Modern Column-Oriented Database Systems

Daniel Abadi
Yale University
dna@cs.yale.edu

Peter Boncz
CWI
P.Boncz@cwi.nl

Stavros Harizopoulos
Amiato, Inc.
stavros@amiato.com

Stratos Idreos
Harvard University
stratos@seas.harvard.edu

Samuel Madden
MIT CSAIL
madden@csail.mit.edu

Contents

Abstract

In this article, we survey recent research on *column-oriented* database systems, or *column-stores*, where each attribute of a table is stored in a separate file or region on storage. Such databases have seen a resurgence in recent years with a rise in interest in analytic queries that perform scans and aggregates over large portions of a few columns of a table. The main advantage of a column-store is that it can access just the columns needed to answer such queries. We specifically focus on three influential research prototypes, MonetDB [46], VectorWise [18], and C-Store [88]. These systems have formed the basis for several well-known commercial column-store implementations. We describe their similarities and differences and discuss their specific architectural features for compression, late materialization, join processing, vectorization and adaptive indexing (database cracking).

D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos and S. Madden. *The Design and Implementation of Modern Column-Oriented Database Systems*. Foundations and Trends® in Databases, vol. 5, no. 3, pp. 197–280, 2012.

DOI: 10.1561/1900000024.

1

Introduction

Database system performance is directly related to the efficiency of the system at storing data on primary storage (e.g., disk) and moving it into CPU registers for processing. For this reason, there is a long history in the database community of research exploring physical storage alternatives, including sophisticated indexing, materialized views, and vertical and horizontal partitioning.

Column-stores. In recent years, there has been renewed interest in so-called *column-oriented systems*, sometimes also called *column-stores*. Early influential efforts include the academic systems MonetDB [46], VectorWise [18][1] and C-Store [88] as well as the commercial system SybaseIQ [66]. VectorWise and C-Store evolved into the commercial systems Ingres VectorWise [99] and Vertica [60], respectively, while by late 2013 all major vendors have followed this trend and shipped column-store implementations in their database system offerings, highlighting the significance of this new technology, e.g., IBM [11], Microsoft [63], SAP [26] and Oracle.

Column-store systems completely vertically partition a database into a collection of individual columns that are stored separately. By

[1]Initially named MonetDB/X100.

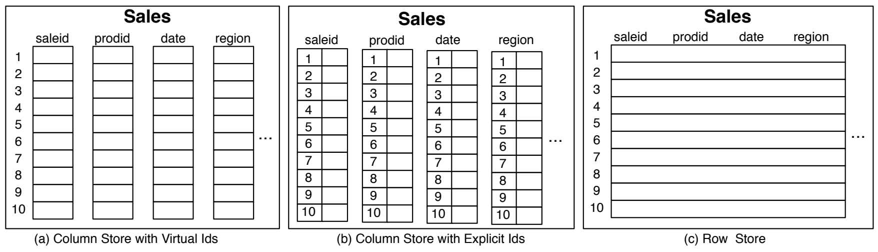
Sales
saleid
prodid
date
region
1
2
3
4
5
6
7
8
9
10
...
(a) Column Store with Virtual Ids
(b) Column Store with Explicit Ids
(c) Row Store

Figure 1.1: Physical layout of column-oriented vs row-oriented databases.

storing each column separately on disk, these column-based systems enable queries to read just the attributes they need, rather than having to read entire rows from disk and discard unneeded attributes once they are in memory. A similar benefit is true while transferring data from main memory to CPU registers, improving the overall utilization of the available I/O and memory bandwidth. Overall, taking the column-oriented approach to the extreme allows for numerous innovations in terms of database architectures. In this paper, we discuss modern column-stores, their architecture and evolution as well the benefits they can bring in data analytics.

Data Layout and Access Patterns. Figure 1.1 illustrates the basic differences in the physical layout of column-stores compared to traditional row-oriented databases (also referred to as *row-stores*): it depicts three alternative ways to store a `sales` table which contains several attributes. In the two column-oriented approaches (Figure 1.1(a) and Figure 1.1(b)), each column is stored independently as a separate data object. Since data is typically read from storage and written in storage in blocks, a column-oriented approach means that each block which holds data for the sales table holds data for one of the columns only. In this case, a query that computes, for example, the number of sales of a particular product in July would only need to access the `prodid` and `date` columns, and only the data blocks corresponding to these columns would need to be read from storage (we will explain the differences between Figure 1.1(a) and Figure 1.1(b) in a moment). On the hand, in the row-oriented approach (Figure 1.1(c)), there is just a single data object containing all of the data, i.e., each block in storage, which holds data for the sales table, contains data from all columns of the table. In this way, there is no way to read just the particular attributes needed for a particular query without also transferring the surrounding attributes. Therefore, for this query, the row-oriented approach will be forced to read in significantly more data, as both the needed attributes and the surrounding attributes stored in the same blocks need to be read. Since data transfer costs from storage (or through a storage hierarchy) are often the major performance bottlenecks in database systems, while at the same time database schemas are becoming more and more complex with fat tables with hundreds of attributes being common, a column-store is likely to be much more efficient at executing queries, as the one in our example, that touch only a subset of a table's attributes.

Tradeoffs. There are several interesting tradeoffs depending on the access patters in the workload that dictate whether a column-oriented or a row-oriented physical layout is a better fit. If data is stored on magnetic disk, then if a query needs to access only a single record (i.e., all or some of the attributes of a single row of a table), a column-store will have to seek several times (to all columns/files of the table referenced in the query) to read just this single record. However, if a query needs to access many records, then large swaths of entire columns can be read, amortizing the seeks to the different columns. In a conventional row-store, in contrast, if a query needs to access a single record, only one seek is needed as the whole record is stored contiguously, and the overhead of reading all the attributes of the record (rather than just the relevant attributes requested by the current query) will be negligible relative to the seek time. However, as more and more records are accessed, the transfer time begins to dominate the seek time, and a column-oriented approach begins to perform better than a row-oriented approach. For this reason, column-stores are typically used in analytic applications, with queries that scan a large fraction of individual tables and compute aggregates or other statistics over them.

Column-store Architectures. Although recent column-store systems employ concepts that are at a high level similar to those in early research proposals for vertical partitioning [12, 22, 55, 65], they include many architectural features beyond those in early work on vertical partitioning, and are designed to maximize the performance on analytic workloads on modern architectures. The goal of this article is to survey these recent research results, architectural trends, and optimizations. Specific ideas we focus on include:

- **Virtual IDs [46].** The simplest way to represent a column in a column-store involves associating a tuple identifier (e.g., a numeric primary key) with every column. Explicitly representing this key bloats the size of data on disk, and reduces I/O efficiency. Instead, modern column-stores avoid storing this ID column by using the position (offset) of the tuple in the column as a virtual identifier (see Figure 1.1(a) vs Figure 1.1(b)). In some column-stores, each attribute is stored as a fixed-width dense array and each record is stored in the same (array) position across all columns of a table. In addition, relying on fixed-width columns greatly simplifies locating a record based on its offset; for example accessing the i-th value in column A simply requires to access the value at the location $startOf(A) + i * width(A)$. No further bookkeeping or indirections are needed. However, as we will discuss later on and in detail in Section 4, a major advantage of column-stores relative to row-stores is improved compression ratio, and many compression algorithms compress data in a non-fixed-length way, such that data cannot simply be stored in an array. Some column-stores are willing to give up a little on compression ratio in order to get fixed-width values, while other column-stores exploit non-fixed width compression algorithms.

- **Block-oriented and vectorized processing [18, 2].** By passing cache-line sized blocks of tuples between operators, and operating on multiple values at a time, rather than using a conventional tuple-at-a-time iterator, column-stores can achieve substantially better cache utilization and CPU efficiency. The use of vectorized CPU instructions for selections, expressions, and other types of arithmetic on these blocks of values can further improve throughput.

- **Late materialization [3, 50].** Late materialization or late tuple reconstruction refers to delaying the joining of columns into wider tuples. In fact, for some queries, column-stores can completely avoid joining columns together into tuples. In this way, late materialization means that column-stores not only store data one column-at-a-time, they also process data in a columnar format. For example, a select operator scans a single column at a time with a tight for-loop, resulting in cache and CPU friendly patterns (as opposed to first constructing tuples containing all attributes that will be needed by the current query and feeding them to a traditional row-store select operator which needs to access only one of these attributes). In this way, late materialization dramatically improves memory bandwidth efficiency.

- **Column-specific compression [100, 2].** By compressing each column using a compression method that is most effective for it, substantial reductions in the total size of data on disk can be achieved. By storing data from the same attribute (column) together, column-stores can obtain good compression ratios using simple compression schemes.

- **Direct operation on compressed data [3].** Many modern column-stores delay decompressing data until it is absolutely necessary, ideally until results need to be presented to the user. Working over compressed data heavily improves utilization of memory bandwidth which is one of the major bottlenecks. Late materialization allows columns to be kept in a compressed representation in memory, whereas creating wider tuples generally requires decompressing them first.

- **Efficient join implementations [67, 2].** Because columns are stored separately, join strategies similar to classic semi-joins [13] are possible. For specific types of joins, these can be much more efficient than traditional hash or merge joins used in OLAP settings.

- **Redundant representation of individual columns in different sort orders [88].** Columns that are sorted according to a particular attribute can be filtered much more quickly on that attribute. By storing several copies of each column sorted by attributes heavily used in an application's query workload, substantial performance gains can be achieved. C-Store calls groups of columns sorted on a particular attribute *projections*. Virtual IDs are on a per-projection basis. Additionally, low-cardinality data that is stored in sorted order can be aggressively compressed.

- **Database cracking and adaptive indexing [44].** Database cracking avoids sorting columns up-front. Instead, a column-store with cracking can adaptively and incrementally sort (index) columns as a side-effect of query processing. No workload knowledge or idle time to invest in indexing is required. Each query partially reorganizes the columns it touches to allow future queries to access data faster. Fixed-width columns allow for efficient physical reorganization, while vector processing means that we can reorganize whole blocks of columns efficiently in one go, making adaptive indexing a realistic architecture feature in modern column-stores.

- **Efficient loading architectures [41, 88].** Finally, one concern with column-stores is that they may be slower to load and update than row-stores, because each column must be written separately, and because data is kept compressed. Since load performance can be a significant concern in data warehouse systems, optimized loaders are important. For example, in the C-Store system, data is first written into an uncompressed, write-optimized buffer (the "WOS"), and then flushed periodically in large, compressed batches. This avoids doing one disk seek per-attribute, per-row and having to insert new data into a compressed column; instead writing and compressing many records at a time.

Are These Column-store Specific Features? Some of the features and concepts described above can be applied with some variations to row-store systems as well. In fact, most of these design features have been inspired by earlier research in row-store systems and over the years several notable efforts both in academia and industry tried to achieve similar effects for individual features with add-on designs in traditional row-stores, i.e., designs that would not disturb the fundamental row-store architecture significantly.

For example, the EVI feature in IBM DB2 already in 1997 allowed part of the data to be stored in a column-major format [14], providing some of the I/O benefits modern column-stores provide. Similarly, past research on fractured mirrors [78] proposed that systems store two copies of the data, one in row-store format and one in column-store format or even research on hybrid formats, such as PAX [5], proposed that each relational tuple is stored in a single page as in a normal row-store but now each page is internally organized in columns; this does not help with disk I/O but allows for less data to be transferred from main-memory to the CPU. In addition, research on index only plans with techniques such as indexing anding, e.g., [69, 25], can provide some of the benefits that late materialization provides, i.e., it allowed processors to work on only the relevant part of the data for some of the relational operators, better utilizing the memory hierarchy. In fact, modern index advisor tools, e.g., [21], always try to propose a set of "covering" indexes, i.e., a set of indexes where ideally every query can be fully answered by one or more indexes avoiding access to the base (row-oriented) data. Early systems such Model 204 [72] relied heavily on bitmap indexes [71] to minimize I/O and processing costs. In addition, ideas similar to vectorization first appeared several years ago [74, 85] in the context of row-stores. Futhrermore, compression has been applied to row-stores, e.g., [30, 82] and several design principles such as decompressing data as late as possible [30] as well as compressing both data and indexes [31, 47] have been studied.

What the column-stores described in this monograph contribute (other than proposing new data storage and access techniques) is an architecture designed from scratch for the types of analytical applications described above; by starting with a blank sheet, they were free to push all these ideas to extremes without worrying about being compatible with legacy designs. In the past, some variations of these ideas

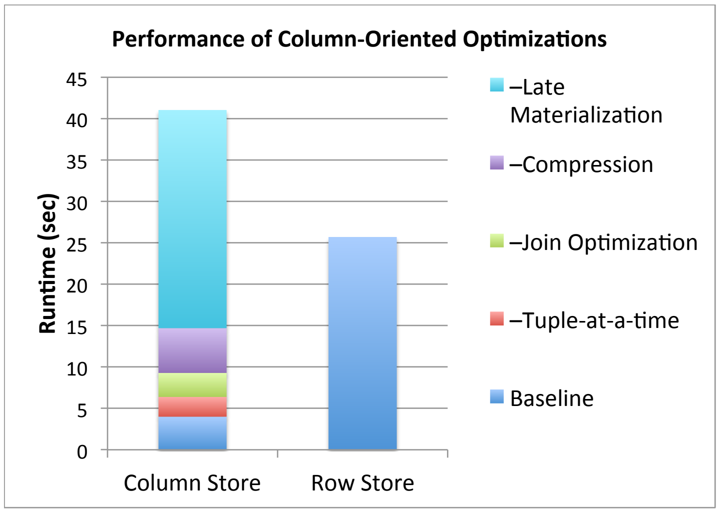

Figure 1.2: Performance of C-Store versus a commercial database system on the SSBM benchmark, with different column-oriented optimizations enabled.

have been tried out in isolation, mainly in research prototypes over traditional row-store designs. In contrast, starting from data storage and going up the stack to include the query execution engine and query optimizer, these column-stores were designed substantially differently from traditional row-stores, and were therefore able to maximize the benefits of all these ideas while innovating on all fronts of database design. We will revisit our discussion in defining modern column-stores vs. traditional row-stores in Section 4.9.

Performance Example. To illustrate the benefit that column-orientation and these optimizations have, we briefly summarize a result from a recent paper [1]. In this paper, we compared the performance of the academic C-Store prototype to a commercial row-oriented ("row-store") system. We studied the effect of various column-oriented optimizations on overall query performance on SSBM [73] (a simplified version of the TPC-H data warehousing benchmark). The average runtime of all queries in the benchmark on a scale 10 database (60 million tuples) is shown in Figure 1.2. The bar on the left shows the performance of C-Store as various optimizations are removed; the "baseline" system with all optimizations takes about 4 seconds to answer all queries, while the completely unoptimized system takes about 40 seconds. The bar on the right shows the performance of the commercial row-store system. From these results it is apparent that the optimized column-store is about a factor of 5 faster than the commercial row-store, but that the unoptimized system is somewhat slower than the commercial system. One reason that the unoptimized column-store does not do particularly well is that the SSBM benchmark uses relatively narrow tables. Thus, the baseline I/O reduction from column-orientation is reduced. In most real-world data-warehouses, the ratio of columns-read to table-width would be much smaller, so these advantages would be more pronounced.

Though comparing absolute performance numbers between a full-fledged commercial system and an academic prototype is tricky, these numbers show that unoptimized column-stores with queries that select a large fraction of columns provide comparable performance to row-oriented systems, but that the optimizations proposed in modern systems can provide order-of-magnitude reductions in query times.

Monograph Structure. In the rest of this monograph, we show how the architecture innovations listed above contribute to these kinds of dramatic performance gains. In particular, we discuss the architecture of C-Store, MonetDB and VectorWise, describe how they are similar and different, and summarize the key innovations that make them perform well.

In the next chapter, we trace the evolution of vertically partitioned and column-oriented systems in the database literature, and discuss technology trends that have caused column-oriented architectures to become more favorable for analytic workloads. Then, in Chapters 3 and 4, we describe the high level architecture and detailed internals primarily of C-Store, MonetDB and VectorWise but also those of subsequent commercial implementations. Finally, in Chapter 5, we discuss future trends and conclude.

References

[1] Daniel J. Abadi, Samuel Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 967–980, 2008.

[2] Daniel J. Abadi, Samuel R. Madden, and Miguel Ferreira. Integrating compression and execution in column-oriented database systems. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 671–682, 2006.

[3] Daniel J. Abadi, Daniel S. Myers, David J. DeWitt, and Samuel R. Madden. Materialization strategies in a column-oriented DBMS. In *Proceedings of the International Conference on Data Endineering (ICDE)*, pages 466–475, 2007.

[4] R. Abdel Kader, P. Boncz, S. Manegold, and M. van Keulen. ROX: run-time optimization of xqueries. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 615–626. ACM, 2009.

[5] Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, and Marios Skounakis. Weaving relations for cache performance. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 169–180, 2001.

[6] Martina-Cezara Albutiu, Alfons Kemper, and Thomas Neumann. Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 5(10):1064–1075, 2012.

[7] Sihem Amer-Yahia and Theodore Johnson. Optimizing queries on compressed bitmaps. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 329–338, 2000.

[8] G. Antoshenkov. Byte-aligned data compression. U.S. Patent Number 5,363,098, 1994.

[9] Cagri Balkesen, Gustavo Alonso, Jens Teubner, and M. Tamer Özsu. Multi-Core, Main-Memory Joins: Sort vs. Hash Revisited. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 7(1):85–96, 2013.

[10] Cagri Balkesen, Jens Teubner, Gustavo Alonso, and M. Tamer Özsu. Main-memory hash joins on multi-core CPUs: Tuning to the underlying hardware. In *Proceedings of the International Conference on Data Endineering (ICDE)*, pages 362–373, 2013.

[11] Ronald Barber, Peter Bendel, Marco Czech, Oliver Draese, Frederick Ho, Namik Hrle, Stratos Idreos, Min-Soo Kim, Oliver Koeth, Jae-Gil Lee, Tianchao Tim Li, Guy M. Lohman, Konstantinos Morfonios, René Müller, Keshava Murthy, Ippokratis Pandis, Lin Qiao, Vijayshankar Raman, Richard Sidle, Knut Stolze, and Sandor Szabo. Business Analytics in (a) Blink. *IEEE Data Eng. Bull.*, 35(1):9–14, 2012.

[12] Don S. Batory. On searching transposed files. *ACM Transactions on Database Systems*, 4(4):531–544, 1979.

[13] Philip A. Bernstein and Dah-Ming W. Chiu. Using semi-joins to solve relational queries. *J. ACM*, 28(1):25–40, 1981.

[14] R. Bestgen and T. McKinley. Taming the business intelligence monster. *IBM Systems Magazine*, 2007.

[15] Carsten Binnig, Stefan Hildenbrand, and Franz Färber. Dictionary-based order-preserving string compression for main memory column stores. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 283–296, 2009.

[16] Peter Boncz. Monet: A next-generation DBMS kernel for query-intensive applications. *University of Amsterdam, PhD Thesis*, 2002.

[17] Peter Boncz, Stefan Manegold, and Martin Kersten. Database architecture optimized for the new bottleneck: Memory access. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 54–65, 1999.

[18] Peter Boncz, Marcin Zukowski, and Niels Nes. MonetDB/X100: Hyper-pipelining query execution. In *Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR)*, 2005.

[19] Peter A. Boncz and Martin L. Kersten. MIL primitives for querying a fragmented world. *VLDB Journal*, 8(2):101–119, 1999.

[20] Nicolas Bruno. Teaching an old elephant new tricks. In *Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR)*, 2009.

[21] Surajit Chaudhuri and Vivek R. Narasayya. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 146–155, 1997.

[22] George P. Copeland and Setrag N. Khoshafian. A decomposition storage model. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 268–279, 1985.

[23] D. J. Dewitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H. I. Hsiao, and R. Rasmussen. The gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, 1990.

[24] David DeWitt. From 1 to 1000 mips, November 2009. PASS Summit 2009 Keynote.

[25] Amr El-Helw, Kenneth A. Ross, Bishwaranjan Bhattacharjee, Christian A. Lang, and George A. Mihaila. Column-oriented query processing for row stores. In *Proceedings of the International Workshop On Data Warehousing and OLAP*, pages 67–74, 2011.

[26] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[27] Avrilia Floratou, Jignesh M. Patel, Eugene J. Shekita, and Sandeep Tata. Column-Oriented Storage Techniques for MapReduce. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 4(7):419–429, 2011.

[28] Clark D. French. “One Size Fits All” Database Architectures Do Not Work for DDS. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 449–450, 1995.

[29] Clark D. French. Teaching an OLTP Database Kernel Advanced Data Warehousing Techniques. In *Proceedings of the International Conference on Data Endineering (ICDE)*, pages 194–198, 1997.

[30] G.Graefe and L.Shapiro. Data compression and database performance. In ACM/IEEE-CS Symp. On Applied Computing, pages 22 -27, April 1991.

[31] Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. Compressing relations and indexes. In *Proceedings of the International Conference on Data Endineering (ICDE)*, pages 370–379, 1998.

[32] Goetz Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[33] Goetz Graefe. Efficient columnar storage in b-trees. *SIGMOD Rec.*, 36(1):3–6, 2007.

[34] Goetz Graefe. Modern B-Tree Techniques. *Foundations and Trends in Databases*, 3(4):203–402, 2011.

[35] Goetz Graefe, Felix Halim, Stratos Idreos, Harumi Kuno, and Stefan Manegold. Concurrency Control for Adaptive Indexing. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 5(7):656–667, 2012.

[36] Goetz Graefe, Stratos Idreos, Harumi Kuno, and Stefan Manegold. Benchmarking Adaptive Indexing. In *Proceedings of the TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC)*, pages 169–184, 2010.

[37] Felix Halim, Stratos Idreos, Panagiotis Karras, and Roland H. C. Yap. Stochastic Database Cracking: Towards Robust Adaptive Indexing in Main-Memory Column-Stores. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 5(6):502–513, 2012.

[38] Alan Halverson, Jennifer L. Beckmann, Jeffrey F. Naughton, and David J. Dewitt. A Comparison of C-Store and Row-Store in a Common Framework. Technical Report TR1570, University of Wisconsin-Madison, 2006.

[39] Richard A. Hankins and Jignesh M. Patel. Data morphing: an adaptive, cache-conscious storage technique. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 417–428, 2003.

[40] Stavros Harizopoulos, Velen Liang, Daniel J. Abadi, and Samuel R. Madden. Performance tradeoffs in read-optimized databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 487–498, 2006.

[41] S. Héman, M. Zukowski, N.J. Nes, L. Sidirourgos, and P. Boncz. Positional update handling in column stores. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 543–554, 2010.

[42] William Hodak. Exadata hybrid columnar compression. Oracle Whitepaper, 2009. http://www.oracle.com/technetwork/database/exadata/index.html.

[43] Allison L. Holloway, Vijayshankar Raman, Garret Swart, and David J. DeWitt. How to barter bits for chronons: compression and bandwidth trade offs for database scans. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 389–400, 2007.

[44] Stratos Idreos. Database Cracking: Towards Auto-tuning Database Kernels. *CWI, PhD Thesis*, 2010.

[45] Stratos Idreos, Ioannis Alagiannis, Ryan Johnson, and Anastasia Ailamaki. Here are my Data Files. Here are my Queries. Where are my Results? In *Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR)*, pages 57–68, 2011.

[46] Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, Sjoerd Mullender, and Martin L Kersten. MonetDB: Two Decades of Research in Column-oriented Database Architectures. *IEEE Data Eng. Bull.*, 35(1):40–45, 2012.

[47] Stratos Idreos, Raghav Kaushik, Vivek R. Narasayya, and Ravishankar Ramamurthy. Estimating the compression fraction of an index using sampling. In *Proceedings of the International Conference on Data Endineering (ICDE)*, pages 441–444, 2010.

[48] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Database cracking. In *Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR)*, pages 68–78, 2007.

[49] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Updating a cracked database. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 413–424, 2007.

[50] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Self-organizing tuple reconstruction in column stores. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 297–308, 2009.

[51] Stratos Idreos, Stefan Manegold, Harumi Kuno, and Goetz Graefe. Merging What's Cracked, Cracking What's Merged: Adaptive Indexing in Main-Memory Column-Stores. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 4(9):585–597, 2011.

[52] Ryan Johnson, Vijayshankar Raman, Richard Sidle, and Garret Swart. Row-wise parallel predicate evaluation. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 1(1):622–634, 2008.

[53] Theodore Johnson. Performance measurements of compressed bitmap indices. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 278–289, 1999.

[54] Ilkka Karasalo and Per Svensson. The design of cantor: a new system for data analysis. In *Proceedings of the 3rd international workshop on Statistical and scientific database management*, pages 224–244, 1986.

[55] Illka Karasalo and Per Svensson. An overview of cantor: a new system for data analysis. In *Proceedings of the 2nd international Workshop on Statistical Database Management (SSDBM)*, pages 315–324, 1983.

[56] Alfons Kemper and Thomas Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *Proceedings of the International Conference on Data Endineering (ICDE)*, pages 195–206, 2011.

[57] Alfons Kemper, Thomas Neumann, Florian Funke, Viktor Leis, and Henrik Mühe. HyPer: Adapting Columnar Main-Memory Data Management for Transactional AND Query Processing. *IEEE Data Eng. Bull.*, 35(1):46–51, 2012.

[58] Setrag Khoshafian, George Copeland, Thomas Jagodis, Haran Boral, and Patrick Valduriez. A query processing strategy for the decomposed storage model. In *Proceedings of the International Conference on Data Endineering (ICDE)*, pages 636–643, 1987.

[59] Setrag Khoshafian and Patrick Valduriez. Parallel execution strategies for declustered databases. In *Proceedings of the International Workshop on Database Machines*, pages 458–471, 1987.

[60] Andrew Lamb, Matt Fuller, Ramakrishna Varadarajan, Nga Tran, Ben Vandier, Lyric Doshi, and Chuck Bear. The vertica analytic database: C-store 7 years later. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 5(12):1790–1801, 2012.

[61] P.Å. Larson, C. Clinciu, E.N. Hanson, A. Oks, S.L. Price, S. Rangarajan, A. Surna, and Q. Zhou. Sql server column store indexes. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 1177–1184, 2011.

[62] Per-Åke Larson, Cipri Clinciu, Campbell Fraser, Eric N. Hanson, Mostafa Mokhtar, Michal Nowakiewicz, Vassilis Papadimos, Susan L. Price, Srikumar Rangarajan, Remus Rusanu, and Mayukh Saubhasik. Enhancements to sql server column stores. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 1159–1168, 2013.

[63] Per-Åke Larson, Eric N. Hanson, and Susan L. Price. Columnar Storage in SQL Server 2012. *IEEE Data Eng. Bull.*, 35(1):15–20, 2012.

[64] Zhe Li and Kenneth A. Ross. Fast joins using join indices. *VLDB Journal*, 8:1–24, April 1999.

[65] R.A. Lorie and A.J. Symonds. A relational access method for interactive applications. In *Courant Computer Science Symposia, Vol. 6: Data Base Systems*. Prentice Hall, 1971.

[66] Roger MacNicol and Blaine French. Sybase IQ multiplex - designed for analytics. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1227–1230, 2004.

[67] S. Manegold, P. Boncz, N. Nes, and M. Kersten. Cache-conscious radix-decluster projections. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 684–695, 2004.

[68] A. Moffat and J. Zobel. Compression and fast indexing for multi-gigabyte text databases. *Australian Computer Journal*, 26(1):1–9, 1994.

[69] C. Mohan, Donald J. Haderle, Yun Wang, and Josephine M. Cheng. Single Table Access Using Multiple Indexes: Optimization, Execution, and Concurrency Control Techniques. pages 29–43, 1990.

[70] Thomas Neumann. Efficiently Compiling Efficient Query Plans for Modern Hardware. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 4(9):539–550, 2011.

[71] Patrick O'Neil and Dallan Quass. Improved query performance with variant indexes. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 38–49, 1997.

[72] Patrick E. O'Neil. Model 204 architecture and performance. In *Proceeding of the International Workshop on High Performance Transaction Systems*, pages 40–59, 1987.

[73] Patrick E. O'Neil, Elizabeth J. O'Neil, and Xuedong Chen. The Star Schema Benchmark (SSB). `http://www.cs.umb.edu/~poneil/StarSchemaB.PDF`.

[74] S. Padmanabhan, T. Malkemus, R. Agarwal, and A. Jhingran. Block oriented processing of relational database operations in modern computer architectures. In *Proceedings of the International Conference on Data Endineering (ICDE)*, pages 567–574, 2001.

[75] Sriram Padmanabhan, Bishwaranjan Bhattacharjee, Timothy Malkemus, Leslie Cranston, and Matthew Huras. Multi-Dimensional Clustering: A New Data Layout Scheme in DB2. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 637–641, 2003.

[76] M. Poess and D. Potapov. Data compression in oracle. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 937–947, 2003.

[77] Bogdan Raducanu, Peter A. Boncz, and Marcin Zukowski. Micro adaptivity in vectorwise. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 1231–1242, 2013.

[78] Ravishankar Ramamurthy, David Dewitt, and Qi Su. A case for fractured mirrors. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 89 – 101, 2002.

[79] V. Raman, G. Attaluri, R. Barber, N. Chainani, D. Kalmuk, V. KulandaiSamy, J. Leenstra, S. Lightstone, S. Liu, G. M. Lohman, T. Malkemus, R. Mueller, I. Pandis, B. Schiefer, D. Sharpe, R. Sidle, A. Storm, and L. Zhang. DB2 with BLU Acceleration: So much more than just a column store. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 6(11), 2013.

[80] Vijayshankar Raman, Lin Qiao, Wei Han, Inderpal Narang, Ying-Lin Chen, Kou-Horng Yang, and Fen-Ling Ling. Lazy, adaptive rid-list intersection, and its application to index anding. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 773–784, 2007.

[81] Vijayshankar Raman, Garret Swart, Lin Qiao, Frederick Reiss, Vijay Dialani, Donald Kossmann, Inderpal Narang, and Richard Sidle. Constant-Time Query Processing. In *Proceedings of the International Conference on Data Endineering (ICDE)*, pages 60–69, 2008.

[82] Mark A. Roth and Scott J. Van Horn. Database compression. *SIGMOD Rec.*, 22(3):31–39, 1993.

[83] Felix Martin Schuhknecht, Alekh Jindal, and Jens Dittrich. The Uncracked Pieces in Database Cracking. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 7(2), 2013.

[84] Minglong Shao, Jiri Schindler, Steven W. Schlosser, Anastassia Ailamaki, and Gregory R. Ganger. Clotho: Decoupling memory page layout from storage organization. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 696–707, 2004.

[85] Ambuj Shatdal, Chander Kant, and Jeffrey F. Naughton. Cache Conscious Algorithms for Relational Query Processing. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 510–521, 1994.

[86] Lefteris Sidirourgos and Martin L. Kersten. Column imprints: a secondary index structure. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 893–904, 2013.

[87] Michael Stonebraker. The case for partial indexes. *SIGMOD Record*, 18(4):4–11, 1989.

[88] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel R. Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alexander Rasin, Nga Tran, and Stan B. Zdonik. C-Store: A Column-Oriented DBMS. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 553–564, 2005.

[89] Dimitris Tsirogiannis, Stavros Harizopoulos, Mehul A. Shah, Janet L. Wiener, and Goetz Graefe. Query processing techniques for solid state drives. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 59–72, 2009.

[90] Stephen Weyl, James Fries, Gio Wiederhold, and Frank Germano. A modular self-describing clinical databank system. *Computers and Biomedical Research*, 8(3):279 – 293, 1975.

[91] K. Wu, E. Otoo, and A. Shoshani. Compressed bitmap indices for efficient query processing. Technical Report LBNL-47807, 2001.

[92] K. Wu, E. Otoo, and A. Shoshani. Compressing bitmap indexes for faster search operations. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 99–108, 2002.

[93] K. Wu, E. Otoo, A. Shoshani, and H. Nordberg. Notes on design and implementation of compressed bit vectors. Technical Report LBNL/PUB-3161, 2001.

[94] Jingren Zhou and Kenneth A. Ross. A Multi-Resolution Block Storage Model for Database Design. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS)*, pages 22–33, 2003.

[95] Jingren Zhou and Kenneth A. Ross. Buffering Database Operations for Enhanced Instruction Cache Performance. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 191–202, 2004.

[96] M. Zukowski. Balancing vectorized query execution with bandwidth-optimized storage. *University of Amsterdam, PhD Thesis*, 2009.

[97] M. Zukowski, S. Héman, and P. Boncz. Architecture-conscious hashing. In *Proceedings of the International Workshop on Data Management on New Hardware (DAMON)*, 2006.

[98] M. Zukowski, S. Héman, N. Nes, and P. Boncz. Cooperative scans: dynamic bandwidth sharing in a DBMS. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 723–734, 2007.

[99] Marcin Zukowski and Peter A. Boncz. Vectorwise: Beyond column stores. *IEEE Data Eng. Bull.*, 35(1):21–27, 2012.

[100] Marcin Zukowski, Sandor Heman, Niels Nes, and Peter Boncz. Super-Scalar RAM-CPU Cache Compression. In *Proceedings of the International Conference on Data Endineering (ICDE)*, 2006.

[101] Marcin Zukowski, Niels Nes, and Peter Boncz. DSM vs. NSM: CPU performance tradeoffs in block-oriented query processing. In *Proceedings of the International Workshop on Data Management on New Hardware (DAMON)*, pages 47–54, 2008.