Column-Oriented Databases: A Survey

Author One Author Two Author Three Author Four

April 2024

Abstract

This survey explores the landscape of column-oriented databases, focusing on pivotal database aspects that shape their design, performance, and utility. Beginning with an introduction to the topic area, the survey outlines the significance of column-oriented databases in the era of big data. The related work section provides a structured overview of foundational research prototypes and modern implementations, highlighting key advancements in storage, compression, query processing, indexing, concurrency control, and real-time analytics. Emphasising the importance of database aspects, the survey offers a comparative analysis of key approaches, evaluating the pros and cons across various dimensions such as scalability, efficiency, reliability, and adaptability. Conclusions and future directions are drawn from the discussion, outlining potential avenues for research and development in column-oriented databases. Through this survey, readers gain insight into the critical database aspects that underpin the evolution and advancement of column-oriented databases, setting the stage for informed decision-making and future exploration of database technologies.

1 Introduction

1.1 Motivation

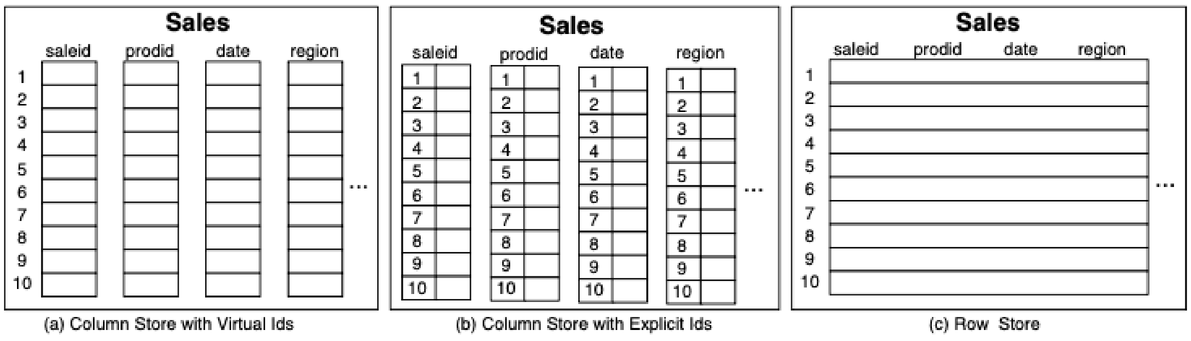
Sales
saleid
prodid
date
region
(a) Column Store with Virtual Ids
(b) Column Store with Explicit Ids
(c) Row Store

Figure 1: Physical layout of column-oriented vs row-oriented databases. [3]

The technological advancement has led to exponential growth of data generated from various sources (i.e., social media, IoT, online transactions, etc). Traditional commercial database management system stores data in a row-oriented manner, that is, all columns of a given row in a table are stored contiguously. Such a storage approach is efficient for Online Transaction Processing(OLTP) tasks for operations such as insert, update, and delete. However, when it comes to Online Analytical Processing (OLAP) tasks, a row-oriented database can be much less effective due to analytical queries usually involving retrieval of the number of columns over a large portion of data, using a row-oriented approach creates significant overhead in retrieving and processing the required columns. The increasing need from both academic and industrial communities to understand(analyze) these large volumes of data has led to the emergence of Column-Oriented databases[24]. Unlike a row-oriented approach, a column-oriented database stores all values of a particular column contiguously. This approach would be particularly advantageous for OLAP tasks due to a couple of reasons. First, a column-oriented design would achieve much better compression since data values within a column are typically much more repetitive than across columns. Second, when querying a large portion of a few columns of a table, only the columns assessed in the query need to be scanned. Whereas in the row-store, all columns need to be retrieved first. Together, this can result in significant improvement for read-intensive workloads on large volumes of data. In this survey, we will explore the design and implementation of column-oriented databases. Primarily focus on the techniques that enable the efficiency for handling read-intensive workloads. This article aims to provide a better understanding of how column-oriented databases have evolved and what future searches are to further enhance the capabilities of column-oriented databases in the big data era.

1.2 Survey Structure

This survey aims to provide a comprehensive overview of column-oriented databases, exploring their design, implementation, and practical applications. In the related works section, we will first explore the foundational prototypes: MonetDB, C-Store, and VectorWise, which have laid the groundwork for modern column-oriented database systems. These prototypes demonstrate the efficiency and potential of columnar stores in handling read-intensive workloads on large volumes of data. Many of the techniques introduced by these systems have been widely adopted and further innovated upon, shaping the advancement of column store databases. Then, We will discuss three representative commercial products that utlise the column-store concepts for efficient analytical query handling: SAP HANA, ClickHouse, and Google BigQuery. Each of these systems demonstrates unique adaptations of the columnar storage principles, catering to different operational needs and performance criteria. Following this, we will delve into a detailed discussion of the main techniques used in columnar stores to optimize performance, highlighting innovations in data compression, indexing, and query processing that enhance speed and efficiency. After summarizing the key papers that have significantly contributed to column-oriented architecture in the related work section, we will conduct a comparative analysis to better understand the strengths and limitations of these key approaches. At the end of the survey, by summarising the result found in the comparison section, we will discuss the areas future research should be focused on to further enhance the adaptability and efficiency of column-oriented databases.

2 Related Work Details

2.1 Foundational Research Prototypes

In the development of column-oriented databases, three research prototypes: MonetDB, VectorWise and C-Store had laid the groundwork for numerous commercial column-store implementations that are well-known in the industry today.

2.1.1 MonetDB

MonetDB is a column-oriented database management system designed to support applications requiring big data analysis efficiently. It innovates in database architecture by adopting a vertical fragmentation model. This will store each attribute in separate columns to improve query performance and data compression rate. MonetDB is able to access the necessary columns for a query by storing data in columns which leads to faster query processing and reduced I/O operations [14]. This design principle is crucial for MonetDB's success in handling large datasets and complex analytical queries. Furthermore, this highlights columnar storage better catering to analytical workloads. More specifically, in read-intensive queries by storing data of the same column together. The storage method is also beneficial for executing aggregation operations and data compressions. Based on columnar storage principles, the system is able to exploit hierarchical memory systems and optimise memory access patterns. This demonstrates the importance of columnar storage in shaping the performance and efficiency of MonetDB.

2.1.2 VectorWise

VectorWise is a column-store system that introduced a vectorized execution model. It tends to reach a balance between the full materialisation of intermediate results and the functional overhead of tuple-at-a-time iterators. In VectorWise, data processing is done one block or vector of a column at a time. This is opposed to one column-at-a-time or one tuple-at-a-time which leads to enhancing reading and scanning efficiency. Despite its fast scan speed, VectorWise permits only one index in each table in its DDL. It implies that the physical order of the rows is determined by the index keys rather than by the insertion sequence [6]. The system also utilises explicit I/O in an advanced manner. Dynamically optimising I/O operations for concurrent queries through features like the Active Buffer Manager (ABM) and Cooperative Scans. Additionally, VectorWise implements Positional Delta Trees to handle updates and incorporates high-speed compression algorithms for improved performance [3]. The vectorized execution model in VectorWise combines pipelining with array-loops code patterns, which enables efficient query processing and minimising memory hierarchy reads and writes.

2.1.3 C-Store

C-Store is an early column-store research prototype with key features such as the primary representation of data on disk being a set of column files. Each column file contains data compressed using a column-specific compression method and sorted based on an attribute in the table. This collection of files is known as the ”read-optimised store” (ROS). Additionally, newly loaded data is stored in a write-optimised store (”WOS”) where data is uncompressed and not vertically partitioned. This enables efficient data loading and cost amortisation of compression and sorting. C-Store employs a compressed execution approach where tuple blocks are kept in compressed form for as long as possible to facilitate pipelined execution. Furthermore, the system utilises deleted bitmaps and temporary tables to handle load and update operations [26]. The advantages of C-Store include efficient query execution, compressed storage, and optimization features for analytic queries. However, compared to row stores, column stores may be slower in loading and updating data due to the separate writing of each column and data compression.

2.2 Commercial Adoption

Following the ground work set by MonetDB, VectorWise and C-Store, there has been many adoption and refinement of column-oriented database techniques across industry. We will discuss three widely deployed column-oriented database systems: HANA, Vertica and BigQuery that leverage the advantage of column-storage such as improved compression, efficient I/O and enhanced query performance, designed to handle analytical query on large volumes of data.

2.2.1 SAP HANA

SAP HANA[9] is an in-memory column-oriented database that is optimised for both OLTP and OLAP workloads. For analytical query processing, HANA’s column-store uses efficient compression schemes in combination with cache-aware and parallel algorithms to ensure performance on read-intensive tasks. Compression not only allows more data on a single load, but also improves the query processing such as exploit run-length encoding to compute aggregate. The scanning process is further accelerated by using SIMD algorithms to operate directly on compressed data.

Meanwhile, HANA also explores the benefit of using the column-store approach to handle OLTP tasks especially for ERP systems. HANA proposed OLTP workload could also benefit from the better compression scheme available in column-store. Many columns in the ERP system are not used which only contain default values of no value at all, and many columns have small domains which makes compression efficient. Consequently, column-store reduces the need for indices which improve the overall query throughput. However, column-store also creates significant performance overhead from managing memory for many columns and resource-intensive updates. HANA addresses these challenges by scheduling and parallel processing. Thus HANA showcases the benefits and possibility for a column-oriented approach for handling both OLAP and OLTP workload, providing a more flexible approach that could better address different business needs.

2.2.2 Vertica

Vertica[18] is a column-oriented analytical database that commercializes the C-Store research prototypes for handling read-intensive analytical workload on petabytes of data. Vertica utilized projection techniques introduced by C-Store which allows more efficient processing of frequent queries compared to materialized view by reducing the maintenance cost as well as the implementation complexity. Meanwhile, Vertica avoided the implementation of join-indices proposed by C-Store due to the costs of using such indices far outweigh the benefits in practice. The use of super projection with compression has shown better performance as explicitly storing row ids consume significant disk space for large tables. Also, Vertica developed their own compression schemes that can automatically applied based on the characteristics of the data column. Similar to C-Store, Vertica has a Read Optimised Stroe(ROS) and a Write Optimized Store(WOS) to ensure query efficiency without impacting the performance by write-intensive tasks.

Furthermore, in order to fully utilize the advantage of column-oriented design, instead of using the minimal optimizer introduced by C-Store, Vertica developed their own patented query optimizer(V2Opt) [27] tailored for Vertica's unique columnar storage architecture and distributed execution engine. This optimizer chooses join ordering with a worklist-based approach that takes distribution information into account and terminates when memory exhaust[23]. The Vertica implementation proves the importance of a specifically designed query optimizer to realize the full power of novel database systems.

2.2.3 BigQuery

BigQuery[10] is another widely employed column-oriented database developed by Google which utilized the Dremel[22] query engine. As data used in web and scientific computing is often non-relational and in nested format, normalizing and recombining such data is usually restrictive at web scale. Thus Google developed the Dremel query engine that is capable of operating on nested data in place. For instance, Dremel can execute many queries over another storage layer that would ordinarily require a sequence of MapReduce jobs, but at a fraction of the execution time[21]. Unlike MapReduce[8] that often involve lengthy job sequences and substantial data shuffling across distributed systems, Dremel uses a nested column-oriented storage designed to significantly enhance the query execution speed to complement MapReduce.

The nested column-store introduced by Dremel has several key core features to allow store all values of a given field consecutively to improve retrieval efficiency. First, Dremel introduces the concept of repetition and definition levels to preserve the hierarchical structure of nested data in a columnar format. The encoding scheme stores repetition and definition levels alongside compressed field values in column-store to preserve the structure and avoid the need for explicit NULL values. Second, to efficiently split Records into columns with the encoded format, Dremel creates a tree of field writers to match the schema hierarchy. Also, Dremel introduced Finite State Machine(FSM)techniques to ensure efficient data assembly and utilized multi-level execution trees to optimize the query execution on nested data. In sum, BigQuery's query engine Dremel showcased an innovative column-store design to address the needs for analyzing large volumes of nested non-relational data.

2.3 Compression

In column-oriented databases, one of the major advantages is improved data compression. Data is stored in individual columns which improve the locality of the processing and thus improves the degree of data compression(i.e., values within each column are much more repetitive than cross columns)[4]. Since different compression algorithms process data in different manners, the compression efficiency largely depends on the data characteristics of the specific column being compressed. Thus, various compression techniques that are tailored for different types of data in column stores have been developed. In [2] the author surveyed the common compression approaches in column-oriented databases, the results from the experiments highlight the importance of choosing the right compression techniques according to the data types of a particular data column. For commercial implementations, Run-length encoding, Dictionary-encoding, delta-encoding are commonly adopted in databases like Vertica and SAP HANA to tailor for different data types. Meanwhile, the compression method used in the column-oriented database enhances CPU performance by allowing direct operation on compressed data through SIMD algorithms[5]. Hence, the efficiency of compression techniques has significant implications for the performance of column-oriented databases. Recent research has focused on advanced compression techniques such as PIDS[15] introduced in 2022, this adaptive approach address the challenges of efficient compression and retrieval simultaneously by identify common patterns of data attributes and decompose into smaller, more manageable sub-attributes. This evolution highlights the potential for specialized adaptive compression techniques to efficiently manage large-scale read-intensive analytical workload.

2.4 Query Processing and Optimization

Given the growing trend of data volumes, Column-oriented database has a number of advantages over conventional SQL databases in the domain of query processing and optimization. These advantages are mostly highlighted in vectorised and parallel processing and late materialization.

2.4.1 Vectorized and Parallel Processing

As data is stored in columns rather than rows, vectorized processing is particularly suitable for column-oriented database as it process data in batches rather than rows. By processing data in batches, it improves local CPU cache utilisation and could better leverage the CPU capability, enhancing query processing time [1]. Also, with the recent rises of in-memory databases, vectorized processing has become a key feature to enable efficient process of read-intensive workload. For instance, many column store implementation such as Vertica, HANA has integrated the vectorized processing capability to ensure performance on large volumes of data.

Moving on, parallel processing refers to executing data operations across multiple processors at the same time. SIMD (Single Instruction, Multiple Data) is a common parallel processing method that is used on column-oriented databases. This type of parallel processing simultaneously processes operations on vectors of columns of data in a single CPU cycle, significantly reducing latency [1]. However, traditional SIMD operations are often limited by the contiguous data storage format. Given the emergence of distributed systems and distributed data structure, traditional SIMD lacks efficiency where data is not physically co-located. Recent research[11] has showed that partition-based SIMD could resolves this issue by dividing data from different positions into smaller partitions during the parallel processing process.

2.4.2 Late Materialization

Late materalisation is a type of query processing method that delays the construction of complete tuples or rows, until it is necessary in the late stage of query processing. In other words, this approach is trying to use positions as long as possible, which minimises the transfer of data. This technique is particularly efficient, if the system only wants to access a subset of data from compressed data. Nonetheless, there are also uses of early materialisation, which constructs tuples as soon as the data are accessed. A classic study highlights that late materialisation shows much better performance than early materialisation in column-based databases, by reducing the I/O operations and more efficient use of memory resources[5].

However, late materialisation could slow down database performance when most of the data is used in an instruction, as it needs to re-read every base-column to construct the data tuples. Recent studies have proposed an adaptive hybrid approach[17] where the system combine the advantage of early and late materialization strategies. This approach adjust the timing of data materialization dynamically based on the query workload and system performance matrix, effectively address the draw backs of late materialization to handle diverse and dynamic query workloads.

2.5 Indexing

Despite column-stores quick access time, exploring efficient indexing strategies could further enhance the performance. As data is stored by column rather than rows, column-store introduced number of indexing techniques tailored for such architecture. In this section, we explore several common indexing strategies utilized in column store system.

2.5.1 Column-Store Indexing Techniques

ZoneMap[25]: is one of the indexing techniques that is frequently used in column-oriented databases. This indexing technique divides and portion the data into several zones based on their value. The metadata structure allows it to enhance query efficiency and optimizes data storage by allowing the system to quickly identify and skip zones that do not meet the query criteria. ZoneMap indexing is particularly useful for range queries by avoiding unnecessary data scans. However, ZoneMap's efficiency suffers when dealing with high cardinality columns as having high number of unique value reduce the chance of a zone can be skipped.

Projection[26] is another commonly used indexing approaches in column-store. Introduced by C-Store as an alternative approach to indexing where tables are replicated with each replica potentially organized differently by key attributes. When query is executed, only the required projections will be loaded which significant reduce the I/O operations and improves query efficiency. However, as these projections are pre-defined, when the query require a pattern that not in the projections may involve the issue of over indexing or under indexing which may affect the efficiency of the query process.

2.5.2 Adaptive Indexing

Adaptive indexing. is an technique introduced in column store to dynamically adjust indexes based on actual query patterns and workload changes over time. As traditional indexing approach require upfront decision on which indexing approach we are going to use. The dynamic and unpredictable nature of big data make using pre-defined indexing less efficient. Adaptive indexing address this issue by continually modifying indexes in response to actual usage patterns, thus optimizing data retrieval without the need for pre-defined indexing strategies.

Database cracking [13] is one of the simple approaches of adaptive indexing. It initially indexes the data in the simple forms such as hash table or B plus tree. Based on the basic indexing form the, the query will firstly look up the basic form. If the query cannot indicate the aiming data based on the initial data indexing form, the query efficiency will greatly slow. Thus, the indexing form needs to be updated and cracked. As the example mentioned in Felix Martin Schuhknecht's work [3]: if the query has a query range of low-high where the data are found separately in two partitions, then the basic indexing form starts cracking (rearrange) that allocated the query range in simply one slot. In this process of cracking the data can be queried in an efficient and low-cost way. As shown in the approach of database cracking, adaptive indexing is able to adjust the form and structure according to the incoming query which can maintain a high query efficiency. Nevertheless, adaptive indexing is fully operated in an automagical way, with no manual adjustment.Due to its characteristic of dynamic adjusting of adaptive indexing the process needs a large number of computational resources. In doing so, the cost of the column-oriented dataset will be higher.

3 Comparison of key approaches

3.1 C-Store vs. VectorWise vs. MonetDB

Prototype	**Indexing**	**Query Optimization**	**Update Handling**	**I/O Optimization**	**Load/Update Speed**
C-store	Column-store Architecture	Read-Optimized Store (ROS)	Write-Optimized Store (WOS)	Compressed Execution	Slow
VectorWise	Vectorized Execution Model	Dynamically Buffer Manager	Positional Delta Trees	High-speed Compression	Medium
MonetDB	Column-store Architecture	Column-at-a-Time Algebra	Database Cracking	Memory-Mapped Files	Fast

Table 1: Comparison of Database Prototypes

C-store, VectorWise and MonetDB are column-based database systems designed to handle analytical workloads efficiently. C-store standout for its ROS and WOS architecture along with its compressed execution approach and efficient update handling. MonetDB emphasises late materialisation, column-at-a-time algebra, database cracking for indexing, and runtime query optimization. VectorWise distinguishes itself from C-Store and MonetDB through its vectorized execution model, advanced I/O optimization strategies, high-speed compression algorithms, and efficient update management using Positional Delta Trees. These features enable VectorWise to excel in loading and updating speed by processing data in blocks, dynamically optimising I/O operations, implementing effective compression techniques, and efficiently managing updates, setting it apart from the approaches taken by C-Store and MonetDB in optimising query processing and performance.

3.2 Comparison of Compression Techniques

Compression Techniques	**Compression Ratio**	**Suitable Data Types**	**Direct Operation**	**CPU Usage**
Dictionary Encoding	High	Strings, Categorical data	Yes	Medium
Run-Length Encoding	High	Repetitive Numeric Data	Yes	Low
Bit-Packing	Medium	Integer, Fixed-Range Numeric	No	High
Delta Encoding	High	Sequential Numeric Data	Limited	Medium
Frame of Reference	Medium	Ordered Numeric Data	Yes	Low

Table 2: Comparison of common compression techniques

As illustrated in Table2, selecting the right compression technique in column-oriented databases is crucial for optimising both storage and performance, particularly for read-intensive tasks. Techniques like Dictionary Encoding and Run-Length Encoding are excellent for categorical and repetitive numeric data respectively, offering high compression and supporting direct operations, thus enhancing query speed. In contrast, Bit-Packing, though effective for fixed-range numeric data, lacks direct operation support and consumes more CPU, making it less suitable for read-heavy environments. Delta Encoding and Frame of Reference also provide targeted benefits for sequential and ordered numeric data, balancing efficiency and resource use. Therefore it is crucial for column stores to align compression strategies with specific data types and operational needs to realise the full potential of such database design.

3.3 Query Optimization

Techniques	**Performance**	**Cost & Complexity**	**Scalability**	**Reliability**
Vectorized	High for uniform data	**Low;** Increases with non-uniform data	Moderate; Limited by data variability	**Moderate;** Prone to mixed type errors
Parallel	Accelerates high-volume operations	**High;** Resource-intensive	**High;** Needs distributed architecture	**High;** Requires robust control
Late Materialization	Delays execution to reduce loading	**Moderate;** Complex to implement	**High;** Scales with data needs	**High;** May delay response

Table 3: Comparison of Query Processing Techniques

Vectorized Processing is highly effective for handling uniform data, offering high performance and low complexity. However, vectorized processing relies on SIMD capabilities which require uniformity in data type and size to perform operations efficiently, the reliance of SIMD making it less efficient when handling heterogeneous datasets. Similarly, while parallel processing enhance query processing speed by distributing operations to multiple processors, a sophisticated concurrency control mechanism such as Multi-Version Concurrency Control (MVCC[7]) is required to effectively manage data integrity and avoid conflicts during parallel transactions. Meanwhile, late materialization delays query execution to avoid unnecessary data loading, but the complexity to implement as well as the delay may was impact the query response time, an adaptive hybrid approach discussed in [17] would better suite the dynamic needs.

3.4 Comparison of Indexing Techniques

Technique	**Query Efficiency**	**Storage Optimization**	**Data Retrieval Speed**	**Adaptability**
Zonemap	Moderate	High	Moderate	Moderate
Adaptive Indexing	High	Low	High	High
Projection Indexing	High	Moderate	High	Low

Table 4: Comparison of Indexing Techniques Across Various Database Aspects

Zonemap Indexing offers moderate efficiency across most aspects but requires manual updates and only benefit from range queries. Adaptive Indexing are well suited when the data are frequently changed and with different distributions in different stages, although it may increase storage demands due to dynamic index creation. Projection Indexing provides excellent query efficiency and retrieval speeds by focusing on specific columns but lacks flexibility, struggling to adapt to new query types or changes in data structure, leading to potential over-indexing. The limitations of these methods underscore the necessity for more sophisticated indexing techniques that can dynamically and efficiently address the evolving and complex needs of big data environments. Therefore, as illustrated from Table 4, to effectively manage and process the vast and varies data common in the big data era, it is crucial for database management system to choose an efficient indexing techniques that could balance adaptability, storage efficiency and query performance.

4 Discussion

4.1 Summary of key insights

In this survey, we have summarised the key insights of column-oriented database technologies, highlighting their advantages in handling read-intensive analytical queries. We first explore the foundational prototypes and how the techniques introduced involved into commercial implementations. Then, we delve deeper into column database's internal techniques such as advanced compression techniques, query optimization methods as well as column-store specific indexing strategies. Through a comparative analysis, we highlighted the strengths and weaknesses of systems like MonetDB, C-Store, and VectorWise, and discussed their optimization for analytical tasks such as column-at-a-time processing and late materialization. Additional, we also provided detailed comparison of the internal techniques in regard of their efficiency, scalability and adaptability and discuss how different techniques address the dynamic query needs. This comprehensive review underscores the significant impact of columnar storage principles on the field of data management, showcasing their pivotal role in enhancing the performance and scalability of database systems.

4.2 Border Implication of Column-Stores

The optimization techniques introduced by column-oriented database have border implication across different database system implementations. For instance, traditional SQL databases have integrated column-store techniques such as column storage, vectorized processing and compression to optimize read intensive workloads [20][19]. This allows SQL databases to leverage columnar efficiency while maintaining compatibility with existing applications. Similarly, graph databases[12] have adopted columnar principles to optimize the storage and retrieval of adjacency lists, improving the efficiency of traversing relationships and accessing node properties, particularly in scenarios with complex relationship patterns. The adoption of column-oriented database techniques in other emerging databases demonstrating their capacity to enhance data processing beyond their original context. Therefore, column-oriented database principles have shown significant potential for adaptation and optimization in various database systems.

4.3 Future Directions

Building upon the comparative insights and related work discussed, future research directions for column-oriented databases could focus on refining compression techniques and optimising query processing. Inspired by systems like SAP HANA and Google BigQuery, advancements could include developing hybrid and adaptive compression algorithms that dynamically adjust to data usage patterns[15][16]. Additionally, further enhancing query optimization strategies, such as the late materialisation techniques employed in C-Store and the vectorized query processing seen in VectorWise, could benefit from more advanced query planning algorithms[17] that dynamically predict query patterns and optimise execution in real time. Moreover, with the increasing needs for databases that can handle both transactional and analytical workloads, similar to HANA's implementation of column-store, future research could also focus on enhancing the flexibility of column-oriented databases. Lastly, research could also explore machine learning integration to automate and optimise database management tasks[28], enabling more adaptive and intelligent systems that can efficiently handle evolving data loads and query complexities. These initiatives could substantially elevate the operational efficiency and analytical capabilities of column-oriented databases in handling large-scale, diverse datasets.

4.4 Conclusion

Conclude with reflections on the evolving role of column-oriented databases in the broader landscape of database management systems. In general the column oriented database has played a significant role in the area of database management systems for its ability to improve the query performance,high data compression and flexibility. In the aspect of foundational research prototype, commercial implementation, compression and indexing strategy, clearly demonstrates the benefits and possible approach of the column oriented database. With the comparison of different approaches which declares the advantages and disadvantages for different approaches in the column oriented database.

References

[1] Daniel Abadi. “Query execution in column-oriented database systems”. In: (Nov. 2008).

[2] Daniel Abadi, Samuel Madden, and Miguel Ferreira. “Integrating compression and execution in column-oriented database systems”. In: *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’06. Chicago, IL, USA: Association for Computing Machinery, 2006, pp. 671–682. ISBN: 1595934340. DOI: 10.1145/1142473.1142548. URL: https://doi.org/10.1145/1142473.1142548.

[3] Daniel Abadi et al. *The Design and Implementation of Modern Column-Oriented Database Systems*. 2013. DOI: 10.1561/1900000024.

[4] Daniel J. Abadi, Peter A. Boncz, and Stavros Harizopoulos. “Column-oriented database systems”. In: *Proc. VLDB Endow.* 2.2 (Aug. 2009), pp. 1664–1665. ISSN: 2150-8097. DOI: 10.14778/1687553.1687625. URL: https://doi.org/10.14778/1687553.1687625.

[5] Daniel J. Abadi et al. “Materialization Strategies in a Column-Oriented DBMS”. In: *2007 IEEE 23rd International Conference on Data Engineering*. 2007, pp. 466–475. DOI: 10.1109/ICDE.2007.367892.

[6] P.A. Boncz and M. Zukowski. “Vectorwise: Beyond Column Stores”. English. In: *IEEE Data Engineering Bulletin* 35.1 (2012), pp. 21–27. ISSN: 1053-1238.

[7] Jan Böttcher et al. “Scalable garbage collection for in-memory MVCC systems”. In: *Proc. VLDB Endow.* 13.2 (Oct. 2019), pp. 128–141. ISSN: 2150-8097. DOI: 10.14778/3364324.3364328. URL: https://doi.org/10.14778/3364324.3364328.

[8] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: simplified data processing on large clusters”. In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492. URL: https://doi.org/10.1145/1327452.1327492.

[9] Franz Färber et al. “The SAP HANA database - An architecture overview”. In: *IEEE Data Eng. Bull.* 35 (Mar. 2012), pp. 28–33.

[10] S Fernandes and J Bernardino. “What is BigQuery?” In: *Proceedings of the 19th International Database Engineering & Applications Symposium*. ACM. 2015.

[11] Viacheslav Galaktionov, Evgeniy Klyuchikov, and George A. Chernishev. “Position Caching in a Column-Store with Late Materialization: An Initial Study”. In: *International Workshop on Data Warehousing and OLAP*. 2020. URL: https://api.semanticscholar.org/CorpusID:212728236.

[12] Pranjal Gupta, Amine Mhedhbi, and Semih Salihoglu. “Columnar storage and list-based processing for graph database management systems”. In: *Proc. VLDB Endow.* 14.11 (July 2021), pp. 2491–2504. ISSN: 2150-8097. DOI: 10.14778/3476249.3476297. URL: https://doi.org/10.14778/3476249.3476297.

[13] Stratos Idreos, Martin Kersten, and Stefan Manegold. “Database Cracking.” In: Jan. 2007, pp. 68–78.

[14] Stratos Idreos et al. “MonetDB: Two Decades of Research in Column-oriented Database Architectures”. In: *IEEE Data Eng. Bull.* 35 (Jan. 2012).

[15] Hao Jiang et al. “PIDS: attribute decomposition for improved compression and query performance in columnar storage”. In: *Proceedings of the VLDB Endowment* 13 (Feb. 2020), pp. 925–938. DOI: 10.14778/3380750.3380761.

[16] Yingting Jin et al. “Adaptive Compression Algorithm Selection Using LSTM Network in Column-oriented Database”. In: *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. 2019, pp. 652–656. DOI: 10.1109/ITNEC.2019.8729341.

[17] Evgeniy Klyuchikov et al. “Hybrid Materialization in a Disk-Based Column-Store”. In: CODS-COMAD ’24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 164–172. ISBN: 9798400716348. DOI: 10.1145/3632410.3632422. URL: https://doi.org/10.1145/3632410.3632422.

[18] Andrew Lamb et al. “The vertica analytic database: C-store 7 years later”. In: *Proc. VLDB Endow.* 5.12 (Aug. 2012), pp. 1790–1801. ISSN: 2150-8097. DOI: 10.14778/2367502.2367518. URL: https://doi.org/10.14778/2367502.2367518.

[19] Per-Ake Larson et al. “Enhancements to SQL server column stores”. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’13. New York, New York, USA: Association for Computing Machinery, 2013, pp. 1159–1168. ISBN: 9781450320375. DOI: 10.1145/2463676.2463708. URL: https://doi.org/10.1145/2463676.2463708.

[20] Per-Åke Larson et al. “SQL server column store indexes”. In: June 2011, pp. 1177–1184. DOI: 10.1145/1989323.1989448.

[21] KH Lee et al. “Parallel Data Processing with MapReduce: A Survey”. In: *ACM SIGMOD Record* 40.4 (2012), pp. 11–20.

[22] S Melnik et al. “Dremel: Interactive Analysis of Web-Scale Datasets”. In: *Communications of the ACM* 54.6 (2011), pp. 114–123.

[23] Maximilian Rieger. “Considering Distributed Processing in the Query Optimizer”. In: *Proceedings of the VLDB 2023 PhD Workshop, co-located with the 49th International Conference on Very Large Data Bases (VLDB 2023)*. Supervised by Prof. Dr. Thomas Neumann, Technische Universität München. Vancouver, Canada: CEUR Workshop Proceedings, Aug. 2023. URL: http://ceur-ws.org.

[24] Aisha Siddiqa, Ahmad Karim, and Abdullah Gani. “Big data storage technologies: a survey”. In: *Frontiers of Information Technology & Electronic Engineering* 18 (2017), pp. 1040–1070. DOI: https://doi.org/10.1631/FITEE.1500441.

[25] Lefteris Sidirourgos and Martin Kersten. “Column imprints: a secondary index structure”. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’13. New York, New York, USA: Association for Computing Machinery, 2013, pp. 893–904. ISBN: 9781450320375. DOI: 10.1145/2463676.2465306. URL: https://doi.org/10.1145/2463676.2465306.

[26] Mike Stonebraker et al. “C-store: a column-oriented DBMS”. In: *Making Databases Work: The Pragmatic Wisdom of Michael Stonebraker*. Association for Computing Machinery and Morgan & Claypool, 2018, pp. 491–518. ISBN: 9781947487192. URL: `https://doi.org/10.1145/3226595.3226638`.

[27] Nga Tran et al. “The Vertica Query Optimizer: The case for specialized query optimizers”. In: *2014 IEEE 30th International Conference on Data Engineering*. 2014, pp. 1108–1119. DOI: `10.1109/ICDE.2014.6816727`.

[28] Xiang Yu et al. “Reinforcement Learning with Tree-LSTM for Join Order Selection”. In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 2020, pp. 1297–1308. DOI: `10.1109/ICDE48307.2020.00116`.