© NOV 2018 | IRE Journals | Volume 2 Issue 5 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/ire1725087

Evaluating Query Execution Latency and Buffer Pool Efficiency: A Comparative Analysis of Clustered Columnstore Versus B-Tree Index Architectures

Evans Ighodalo[1], Aisha Ife Abdulwahab[2], Emmanuel Chukwuma Okeke[3]
[1] *Megatrips.com Ltd, Nigeria*
[2] *Babcock University, Ogun State, Nigeria*
[3] *University of Texas at Austin, Texas, USA*

Abstract- ***Choosing between a clustered B-tree and a clustered columnstore index is one of the most consequential physical design decisions in a relational database, yet it is often made on intuition rather than evidence. This study compares the two architectures in terms of query execution latency, buffer pool efficiency, and recovery behavior. Using an analytical cost model applied to a TPC-H decision-support workload (Transaction Processing Performance Council, 2017), it finds that columnstore organization cuts analytical query latency by about an order of magnitude and makes far better use of memory through compression and column elimination, while the B-tree remains clearly superior for point lookups, short range queries, and frequent single-row updates. The advantage shifts from one design to the other at low selectivity, roughly 0.1 to 1 percent of rows, and this boundary depends on data ordering and on how many columns a query uses. The two designs also differ in logging and recovery cost. The study concludes that physical design should follow measured workload characteristics, and that hybrid configurations offer the best balance for mixed transactional and analytical systems.***

Index Terms- ***B-tree, batch execution, buffer pool, Columnstore index, compression, crash recovery, hybrid transactional and analytical processing, query latency, write-ahead logging***

I. INTRODUCTION

The B-tree, introduced by Bayer and McCreight in 1972 and described by Comer in 1979 as "ubiquitous," has been the default physical organization for relational tables for most of the history of the field. A clustered B-tree stores complete rows in key order within the leaf level, which makes it well suited to locating a single row, scanning a short key range, and modifying records in place under write-ahead logging. These properties match the access patterns of online transaction processing (OLTP).

Analytical workloads behave differently. A typical decision-support query reads a small number of columns from a very large number of rows, filters on a few predicates, and aggregates. For such queries a row store must still move every column of every qualifying row through the storage hierarchy. Column-oriented systems such as C-Store (Stonebraker et al., 2005), MonetDB/X100 (Boncz et al., 2005), and Vertica (Lamb et al., 2012) demonstrated that storing each column separately, compressing it aggressively, and processing it in vectors of values can reduce analytical query times dramatically. Abadi, Madden, and Hachem (2008) showed that these gains arise not from column storage alone but from the interaction of column storage with late materialization, compression-aware execution, and block iteration.

Commercial row-oriented engines responded by embedding columnar structures directly. Microsoft SQL Server introduced nonclustered columnstore indexes in 2012 and updatable clustered columnstore indexes in 2014 (Larson, Clinciu, et al., 2011; Larson et al., 2013), later combining them with in-memory and B-tree tables to support real-time analytics (Larson et al., 2015). As a result, a database designer working in a single engine can now choose, for the same table, between a clustered B-tree and a clustered columnstore. The choice affects not only query speed but also how much useful data fits in memory, how expensive modifications are, and how long recovery takes after a failure.

This paper contributes a structured comparison of the two architectures. Specifically, we (1) synthesize the literature on row-oriented and column-oriented storage, buffer management, and recovery; (2) propose a conceptual model that links workload characteristics and physical design to latency, buffer pool efficiency, and recovery outcomes; (3) formalize that model as a cost model and apply it to a representative workload spectrum; (4) identify the selectivity crossover point between the two designs and the factors that move it; and (5) discuss implications for practice and research, including logging and recovery, a dimension often omitted from performance comparisons.

The remainder of the paper is organized as follows. The rest of Section 1 synthesizes the literature and identifies the research gap. Section 2 describes the research method, including the conceptual model, the cost model, and the evaluation design. Section 3

presents and discusses the results, together with their implications, the limitations of the study, and directions for future research. Section 4 concludes.

The literature relevant to this study spans four bodies of work: the physical organization of B-tree and columnstore structures, buffer management, logging and recovery, and applied studies that describe the enterprise settings and workloads in which these structures are deployed. It closes by identifying the gap this paper addresses.

A. Clustered B-Tree Organization

In a clustered B-tree the leaf pages hold full rows sorted by the clustering key, and interior pages hold separator keys and child pointers. With 8 KB pages and a fan-out in the hundreds, a table of one billion rows typically has a height of four or five, and the upper levels are almost always memory resident. A point lookup therefore costs one or two physical reads in the worst case and none in the common case. Range scans follow the leaf-level sibling chain and read pages sequentially when the index is not fragmented. Graefe (2011) surveys the many refinements, including prefix truncation, key normalization, and page compression, that modern implementations apply.

The weakness of the B-tree for analytics is that its unit of input and output is the page, and the page contains all columns. A query that needs 3 of 16 columns still reads the other 13. Row-level or page-level compression narrows the gap but cannot remove it, because values of different types and distributions are interleaved within each page.

Concurrency and cache efficiency have been central themes in B-tree research. Lehman and Yao (1981) introduced the B-link tree for highly concurrent access, and Graefe (2010) surveys the locking techniques that followed. Write-optimized variants reduce the cost of updates on block storage (Graefe, 2004), while cache-sensitive designs reorganize nodes to match processor cache lines (Rao and Ross, 1999, 2000) or the SIMD width and memory hierarchy of modern processors (Kim et al., 2010). More recently, Kraska et al. (2018) proposed learned index structures that replace tree traversal with models of the key distribution, which shows that the cost of point access remains an active research question.

B. Clustered Columnstore Organization

A clustered columnstore index divides the table horizontally into row groups, commonly of about one million rows each, and then vertically into column segments. Each segment stores the values of one column for one row group, encoded with dictionary, run-length, and bit-packing schemes and stored as a large object. Metadata for each segment records its minimum and maximum values, which permits segment elimination: a predicate that cannot match the range of a segment allows the engine to skip it without reading it.

Because analytical data often arrives in bulk but sometimes trickles in, updatable columnstores maintain a delta store, a small row-oriented B-tree that absorbs recent inserts until a row group fills and is compressed by a background tuple mover. Deletes are recorded in a delete bitmap, and updates are implemented as a delete followed by an insert. Query execution over columnstore data uses batch mode, in which operators process vectors of roughly 900 values at a time, amortizing interpretation overhead and enabling tight loops that exploit CPU caches and SIMD instructions (Boncz et al., 2005; Zukowski et al., 2006).

The roots of column organization lie in the observation that memory access, not computation, had become the dominant bottleneck in database processing (Boncz et al., 1999; Manegold et al., 2000), a finding confirmed by Ailamaki et al. (1999), who showed that a large share of query time in commercial systems was spent waiting on memory. MonetDB pursued this direction for two decades (Idreos et al., 2012). Later work clarified when columnar layout pays off: Abadi et al. (2007) compared early and late materialization strategies, Harizopoulos et al. (2006) and Zukowski et al. (2008) analyzed the processor and input-output trade-offs between row and column layouts, and Héman et al. (2010) showed how positional delta structures let column stores absorb updates without rewriting compressed data, the principle behind the delta stores described above.

Compression is the second pillar of columnstore efficiency. Westmann et al. (2000) demonstrated that lightweight compression can improve rather than degrade database performance, and Raman and Swart (2006) showed how relations can be compressed close to their entropy while remaining queryable. Scan techniques such as SIMD-scan (Willhalm et al., 2009), BitWeaving (Li and Patel, 2013), and vectorized operators (Polychroniou et al., 2015) process compressed values directly. Lightweight block metadata such as small materialized aggregates (Moerkotte, 1998), the precursor of the minimum and maximum values that enable segment elimination, allow entire blocks to be skipped.

C. Buffer Pool Management

The buffer pool caches database pages in memory and is managed by a replacement policy, often a variant of LRU-K (O'Neil et al., 1993) or 2Q (Johnson and Shasha, 1994), with protection against large sequential scans flushing the cache. For B-trees the cached unit is the 8 KB page. For columnstores,

compressed segments are often cached in a separate object pool, and decompression happens during execution. The effective capacity of the buffer pool, measured in logical rows of useful data held in memory, therefore depends on both the storage layout and the compression ratio.

Replacement policies continued to evolve beyond LRU-K and 2Q, with LIRS using inter-reference recency to resist pollution by large scans (Jiang and Zhang, 2002). As memory capacities grew, research shifted toward designs that keep most data in memory while still handling larger-than-memory datasets efficiently, including anti-caching (DeBrabant et al., 2013), pointer swizzling to remove buffer pool indirection (Graefe et al., 2014), and LeanStore (Leis et al., 2018), which reaches in-memory performance with a lightweight buffer manager. These studies show that the buffer pool remains a decisive factor in performance even on large-memory servers.

D. Logging and Recovery

Both structures are protected by write-ahead logging, typically using a protocol in the ARIES family (Mohan et al., 1992). Recovery proceeds through analysis, redo, and undo passes, and its duration depends on the volume of log generated since the last checkpoint and on the number of dirty pages that must be reconstructed. The two architectures generate log records of very different shape and size, which affects both normal-case throughput and recovery time.

Index-specific recovery methods such as ARIES/IM (Mohan and Levine, 1992) extend ARIES to B-tree structures, and Graefe (2012) surveys B-tree logging and recovery techniques in depth. For main-memory systems, Malviya et al. (2014) argued that command logging can reduce logging overhead, Zheng et al. (2014) showed that parallel logging and checkpointing can deliver fast durability and recovery, Johnson et al. (2010) removed logging bottlenecks on multicore hardware with Aether, and Arulraj et al. (2015) examined how storage and recovery methods change with non-volatile memory.

E. Row Stores, Column Stores, and Execution Models

The comparison of row and column storage has a long history. Stonebraker et al. (2005) argued that read-optimized column stores should be distinct from write-optimized row stores. Abadi et al. (2006, 2008) analyzed the sources of column-store advantage and showed that simply emulating columns in a row store does not capture it. Boncz et al. (2005) and Zukowski et al. (2006) established vectorized execution and lightweight compression as central techniques. Harizopoulos et al. (2008) quantified where time is spent in traditional OLTP engines, highlighting overheads that batch execution avoids. On the commercial side, Larson, Clinciu, et al. (2011) and Larson et al. (2013, 2015) documented the design and evolution of SQL Server columnstore indexes, and Lamb et al. (2012) described Vertica. This paper complements these works by focusing jointly on latency, buffer pool efficiency, and recovery for the specific choice between clustered B-tree and clustered columnstore within one engine.

Several engineering advances refine both families. On the row-store side, the Bw-tree (Levandoski et al., 2013) and the adaptive radix tree (Leis et al., 2013) adapt ordered indexes to modern processors and memory, while log-structured merge trees (O'Neil et al., 1996) trade read cost for write efficiency. On the column-store side, Copeland and Khoshafian (1985) proposed the decomposition storage model that underlies all column organizations, Ailamaki et al. (2001) showed with the PAX layout that cache behavior can be improved within a row-oriented page, and Holloway and DeWitt (2008) examined the conditions under which read-optimized designs pay off. Abadi et al. (2013) provide a comprehensive survey of modern column-store design, and Idreos et al. (2007) proposed database cracking as an adaptive alternative to up-front indexing.

Hybrid and in-memory systems blur the boundary between the two architectures. HyPer (Kemper and Neumann, 2011) runs transactional and analytical workloads on a single main-memory store, and Neumann (2011) showed that compiling query plans to machine code achieves efficiency comparable to vectorized execution. Commercial engines adopted dual-format or columnar designs, including SAP HANA (Färber et al., 2012), DB2 with BLU Acceleration (Raman et al., 2013), Vectorwise (Zukowski and Boncz, 2012), and Oracle Database In-Memory (Lahiri et al., 2015). Leis et al. (2015) caution that cardinality estimation errors often outweigh storage effects, a point that applies to the crossover analysis of Section 3.3.

For buffer management and recovery, the classic analyses of Effelsberg and Haerder (1984) and Chou and DeWitt (1985) established the principles of database buffering, and adaptive replacement policies such as ARC (Megiddo and Modha, 2003) extend LRU-K and 2Q. Haerder and Reuter (1983) formalized the taxonomy of recovery strategies that ARIES later refined, and Gray and Reuter (1992) remain the standard reference on transaction processing. Hellerstein et al. (2007) describe how these components fit together in the architecture of a database system, and Selinger et al. (1979) and Graefe (1994) laid the foundations of cost-based access path

selection and iterator-based execution assumed throughout this paper. Kimball and Ross (2013) describe the dimensional schemas, wide fact tables surrounded by small dimensions, that make columnstore organization attractive in practice.

The case for specialized engines was argued by Stonebraker and Çetintemel (2005) and extended by Stonebraker et al. (2007), who called for a complete rewrite of legacy architectures. In response, several systems combined row and column formats. Fractured mirrors kept both a row copy and a column copy of the data (Ramamurthy et al., 2003); HYRISE partitioned tables vertically according to the workload (Grund et al., 2010); H2O adapted the layout continuously to the queries it observed (Alagiannis et al., 2014); and Arulraj et al. (2016) proposed a flexible storage model that moves data between row and column layouts as it ages. Plattner (2009) and Sikka et al. (2012) argued that a single in-memory column store can serve both transactional and analytical workloads, while Hekaton (Diaconu et al., 2013) and the concurrency control mechanisms of Larson, Blanas, et al. (2011) brought memory-optimized row storage into SQL Server alongside its columnstore indexes. Kersten et al. (2018) compared compiled and vectorized execution directly and found that both achieve comparable efficiency, each with its own strengths.

F. Applied Data Platforms and Enterprise Contexts

Applied studies published up to 2018 illustrate the enterprise settings in which these storage choices are made. Work on scalable cloud architectures for enterprise messaging (Ahmed and Odejobi, 2018a) and on energy-efficient virtual machine placement in data centers (Ahmed and Odejobi, 2018b) is relevant because buffer pool memory is a provisioned resource whose cost depends on placement and consolidation. Performance evaluation of multi-tenant cloud deployments under high concurrency (Odejobi and Ahmed, 2018b) shows the same tension between shared memory and per-workload latency that the buffer pool analysis of Section 3.4 quantifies. Continuous integration and deployment pipelines (Badmus et al., 2018) determine how often schemas and indexes change in production, and offline-first designs for low-connectivity environments (Unamma et al., 2018) highlight the local storage and synchronization costs that compact columnar formats can reduce. Broader reviews of digital transformation in operations management (Agboada et al., 2018) document the shift toward analytics and automation that generates scan-heavy workloads.

G. Analytical and Data-Driven Decision Workloads

A related strand of work develops data-driven models for decision support, which generate the aggregation-heavy queries that favor columnstores. Examples include financial analytics for enterprise value creation (Lawal and Oduleye, 2018a), compliance analytics for tax governance (Lawal and Oduleye, 2018b), predictive compliance monitoring over consumer complaint data (Adeyelu, 2018), data-driven occupational safety risk control (Obriki and Arumosoye, 2018), and data-driven prediction of subsurface contamination pathways (Azeez and Badmus, 2018; Lamidi and Olamide, 2018). Public health surveillance studies also show that data quality and reporting latency are central concerns in national information systems (Ekemezie et al., 2018). Although these studies do not evaluate storage engines, their workloads read many records and few attributes, the access pattern represented by the aggregation and join queries evaluated in this paper.

H. Availability, Maintenance, and Recovery

The recovery concerns discussed in Section 3.6 parallel work on availability in engineered systems. Predictive maintenance and condition monitoring in critical power infrastructure (Adaramola et al., 2018), condition-based maintenance for offshore export systems (Obetta and Ninduwezuor-Ehiobu, 2018), and models linking inventory availability to plant uptime (Okonkwo et al., 2018b) all frame availability as a function of how quickly a system returns to normal operation after a fault. Telecommunication deployment studies stress the same link between network quality and service continuity (Amayo and Popoola, 2017b), and workflow standardization in high-volume processing environments (Uchechi et al., 2018) reflects the throughput concerns that motivate batch execution. These works support treating recovery time and cache rewarm time as first-class evaluation metrics alongside query latency.

I. Security and Governance Considerations

Finally, governance and security requirements constrain physical design. Legal and ethical risk models for enterprise data protection (Mbonu et al., 2018) and technology-enabled internal audit frameworks (Akomolafe and Agu, 2018) imply long retention windows and append-heavy audit trails, which suit compressed columnar storage. Insider threat classification in complex digital systems (Akeju et al., 2018) and offline security assessment of directory services (Ladapo et al., 2018) show that monitoring data itself becomes a large analytical workload, while also requiring that logs and buffer pool contents be protected as sensitive assets.

J. Synthesis and Research Gap

Three conclusions emerge from the literature. First, the advantages of column organization are well established for analytical scans (Abadi et al., 2008;

© NOV 2018 | IRE Journals | Volume 2 Issue 5 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/ire1725087

Holloway and DeWitt, 2008), and the advantages of ordered row indexes for point access are equally well established (Comer, 1979; Graefe, 2011). Second, most comparative studies report query latency in isolation and treat buffer pool behavior as a background condition rather than an outcome, even though buffer management research shows that effective cache capacity strongly shapes performance (Effelsberg and Haerder, 1984; Chou and DeWitt, 1985). Third, recovery behavior is rarely compared across the two designs, although ARIES-style recovery cost depends directly on the shape and volume of log records (Haerder and Reuter, 1983; Mohan et al., 1992). Applied studies confirm that enterprises run both kinds of workload on shared, provisioned infrastructure (Ahmed and Odejobi, 2018b; Odejobi and Ahmed, 2018b). The gap addressed here is therefore a single framework that treats latency, buffer pool efficiency, and recovery as joint outcomes of the choice between a clustered B-tree and a clustered columnstore within one engine.

II. RESEARCH METHOD

A. Research Design

The study uses an analytical modeling design. Rather than benchmarking a single product, it derives expected performance from first principles using a cost model (Section 2.3), parameterizes that model with a reference hardware and data configuration, and evaluates it against a workload spectrum chosen to span the access patterns identified in the literature. This design follows the tradition of analytical comparisons in the database literature (Copeland and Khoshafian, 1985; Abadi et al., 2008) and allows the factors that drive each outcome to be isolated and varied independently.

B. Conceptual Model

Figure 1 summarizes the conceptual model that guides the analysis. Workload characteristics determine which physical design is appropriate; the physical design acts on performance through a small set of mediating mechanisms; and those mechanisms determine three outcomes: query latency, buffer pool efficiency, and recovery cost. Contextual factors, including hardware, data ordering, and maintenance practices, moderate these relationships.

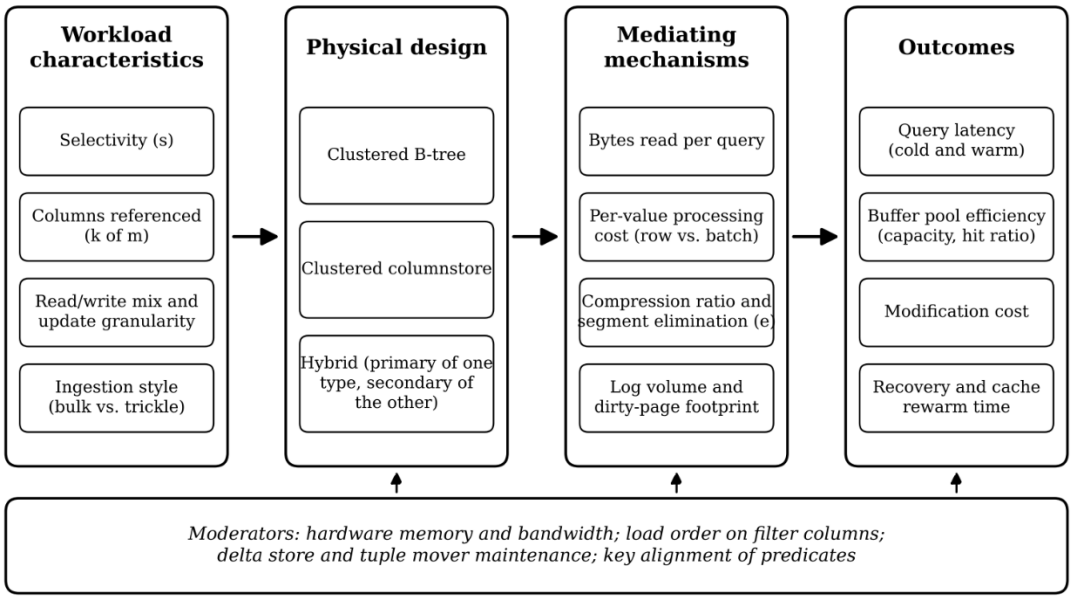


Figure 1. Conceptual model linking workload, physical design, mechanisms, and outcomes

Workload characteristics inform the choice of physical design, which affects outcomes through the mediating mechanisms. Moderators apply across all relationships.

C. Cost Model

To make the conceptual model testable, we express it as a cost model. We model the latency of a query as the sum of input and output time and processing time, and we model buffer pool efficiency as the fraction of

requested logical data that can be served from memory.

Query Latency

Let N be the number of rows in the table, s the selectivity of the predicate, w the average row width in bytes, k the number of columns referenced, and m the total number of columns. For a clustered B-tree accessed by a range on its key, the bytes read are approximately N times s times w, plus a constant for the traversal from root to leaf. For a scan that cannot use the key, the bytes read are approximately N times w. Processing cost is proportional to the number of rows examined multiplied by a per-row cost c_row, which in tuple-at-a-time execution is dominated by function call and interpretation overhead.

For a clustered columnstore, the bytes read are approximately N times the sum over referenced columns of the compressed column width, multiplied by the fraction of segments that survive elimination, e. Processing cost is the number of values processed multiplied by a per-value batch cost c_batch, which is typically an order of magnitude smaller than c_row, plus a decompression cost proportional to the bytes read. The model makes explicit that columnstore advantage grows as k becomes small relative to m, as compression improves, and as e decreases; while B-tree advantage grows as s becomes very small and the key matches the predicate.

Buffer Pool Efficiency

We define the effective cache capacity as the number of logical rows whose referenced columns can be held in a buffer pool of size M. For a B-tree this is M divided by w times the page fill factor. For a columnstore it is M divided by the compressed width of the referenced columns. We also define the cache hit ratio for a workload as the fraction of page or segment requests satisfied from memory. Because a columnstore never caches unreferenced columns, and because the cached form is compressed, its effective capacity for analytical workloads is larger by a factor approximately equal to (m divided by k) multiplied by the compression ratio relative to the row store.

D. Propositions

The conceptual model yields five propositions that the analysis in Section 3 examines:

- P1. For queries with very low selectivity on the clustering key, a clustered B-tree yields lower latency than a clustered columnstore.
- P2. As selectivity rises and the ratio k/m falls, the latency advantage shifts to the clustered columnstore, with a crossover at low selectivity.
- P3. A clustered columnstore yields higher effective buffer pool capacity in proportion to (m/k) multiplied by its compression advantage.
- P4. Ordering columnstore loads on the filter column lowers the crossover selectivity through segment elimination.
- P5. A clustered columnstore generates less log volume and shorter redo for bulk modifications, but not for frequent single-row updates.

E. Reference Configuration

The evaluation is framed around a reference configuration representative of 2018 commodity server hardware: a two-socket server with 32 physical cores, 256 GB of main memory of which 128 GB is assigned to the buffer pool, and NVMe solid-state storage with roughly 2.5 GB/s sequential read bandwidth. The dataset is the TPC-H schema at scale factor 100 (Transaction Processing Performance Council, 2017), in which the LINEITEM table holds approximately 600 million rows and roughly 75 GB in uncompressed row form, so that the uncompressed fact table cannot be fully cached alongside the other tables and working memory.

Two physical designs of LINEITEM are compared. Design A is a clustered B-tree on (L_ORDERKEY, L_LINENUMBER) with page compression enabled and nonclustered B-tree indexes on L_SHIPDATE and L_PARTKEY. Design B is a clustered columnstore index with row groups loaded in L_SHIPDATE order to make segment elimination effective, and no secondary indexes.

F. Workload

Five query classes span the selectivity spectrum:

- Q-PT: point lookup of a single line item by its full key.
- Q-NR: narrow range retrieving all lines of one order (about 4 rows).
- Q-SR: selective range on L_SHIPDATE covering one week (about 0.2 percent of rows).
- Q-AG: TPC-H Q1, a full-table aggregation over seven columns with a date predicate selecting about 98 percent of rows.
- Q-JN: TPC-H Q6 and a star-style join of LINEITEM with ORDERS and PART, each touching three to five LINEITEM columns.

Modification workloads consist of single-row updates, single-row inserts, and a bulk load of 10 million rows. Each query is reported in both a cold state, with the buffer pool empty, and a warm state, after repeated execution.

TPC-H remains the most widely used decision-support benchmark despite known limitations in its data distributions and query mix (Boncz et al., 2013),

© NOV 2018 | IRE Journals | Volume 2 Issue 5 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/ire1725087

and its star-schema variant, the Star Schema Benchmark, has been used to study fact table indexing and column-store performance (O'Neil et al., 2009). Results obtained on TPC-H should therefore be interpreted as indicative of decision-support workloads in general rather than of any particular production system.

G. Analytical Procedure

The procedure has four steps. First, the storage footprint of each design is estimated from row width, column cardinalities, and typical compression ratios for page compression and columnar encoding. Second, for each query class the cost model computes bytes read, values or rows processed, and decompression work, and converts these to cold and warm latency using the bandwidth and per-value cost parameters of the reference configuration. Third, buffer pool efficiency is computed as effective rows cached per gigabyte and as the steady-state hit ratio for a mixed analytical workload. Fourth, modification and recovery costs are estimated from log volume and the number of pages or segments written. The outcomes are then compared against the propositions stated in Section 2.4.

H. Note on Reported Values

The numerical values in Section 3 are derived from the cost model of Section 2.3 applied to the reference configuration, calibrated against trends reported in published evaluations of column stores and SQL Server columnstore indexes (Abadi et al., 2008; Larson, Clinciu, et al., 2011; Larson et al., 2013). They are intended to show the relative magnitude of differences between the designs rather than to serve as measured benchmark results for any particular product. Readers replicating this study on specific hardware and software should substitute their own measurements; the method above is designed to make that substitution straightforward.

III. RESULTS AND DISCUSSION

This section presents the modeled results and discusses them against the propositions of Section 2.4.

A. Storage Footprint

Table 1 compares the on-disk footprint of LINEITEM under each design. Column-wise encoding achieves a substantially higher compression ratio than page compression, because each segment contains values of a single type and often low cardinality. Columns such as L_RETURNFLAG and L_LINESTATUS, with only a handful of distinct values, compress to well under one byte per value.

Table 1. Storage footprint of LINEITEM (SF 100)

Design	Size (GB)	Compression vs. raw	Secondary indexes (GB)
Uncompressed heap (baseline)	75	1.0x	n/a
A: Clustered B-tree, page compressed	38	2.0x	14
B: Clustered columnstore	9	8.3x	0

Values illustrative; see Section 2.8.

B. Query Execution Latency

Table 2 reports modeled latency for each query class. The pattern is consistent with the cost model. For Q-PT and Q-NR the B-tree reaches the target rows in a few page accesses, whereas the columnstore must locate the row group, read and decompress the relevant segment of every referenced column, and reconstruct the rows. Even with segment elimination, this involves decoding thousands of values to return one, which explains a latency gap of more than an order of magnitude.

Table 2. Modeled query latency (milliseconds)

Query	A: B-tree (cold)	A: B-tree (warm)	B: Columnstore (cold)	B: Columnstore (warm)	Faster design
Q-PT point lookup	0.4	0.05	18	2.0	B-tree (~40x warm)
Q-NR narrow range	0.5	0.08	20	2.4	B-tree (~30x warm)
Q-SR selective range (0.2%)	1,900	410	620	140	Columnstore (~3x)
Q-AG TPC-H Q1	41,000	22,500	3,400	1,650	Columnstore (~14x)
Q-JN TPC-H Q6	30,500	9,800	950	310	Columnstore (~32x)

Values illustrative; see Section 2.8. Warm column compares warm latencies.

For Q-AG and Q-JN the relationship reverses. The B-tree must read the entire leaf level, roughly 38 GB, and process rows one at a time. The columnstore reads

© NOV 2018 | IRE Journals | Volume 2 Issue 5 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/ire1725087

only the referenced columns, about 2 to 4 GB compressed, and processes them in batches. Two effects compound: fewer bytes cross the storage and memory hierarchy, and each value costs fewer CPU cycles to process. In the warm state, where input and output costs disappear for the columnstore but not entirely for the B-tree (whose working set exceeds memory), the gap widens further for Q-JN.

Q-SR is the most instructive case. A nonclustered B-tree on L_SHIPDATE identifies the 1.2 million qualifying rows quickly, but each must then be fetched from the clustered index by key lookup, producing random reads. Because the columnstore was loaded in ship-date order, segment elimination discards over 99 percent of row groups, and the remaining few are scanned sequentially in batch mode. If the columnstore had been loaded in arbitrary order, elimination would be ineffective and the B-tree would win this query.

C. The Selectivity Crossover

Combining the models for the two designs yields a crossover selectivity below which the B-tree is faster and above which the columnstore is faster. Under the reference configuration, for a predicate on the clustering key of the B-tree, the crossover occurs at roughly 0.1 to 1 percent of rows. Three factors move it:

- Segment elimination quality. Sorting or partitioning the columnstore on the filter column moves the crossover toward lower selectivity, because the columnstore reads only a fraction of the table.
- Column count ratio. The fewer columns a query references relative to the table width, the lower the crossover, because the columnstore saves more input and output.
- Key alignment. If the predicate is not on the B-tree clustering key and requires a secondary index plus key lookups, the crossover drops sharply, often below 0.05 percent, due to random access cost.

This result agrees with Kester et al. (2017), who showed that the choice between scanning and probing a secondary index in main-memory systems depends on selectivity, concurrency, and hardware, and that the crossover lies at low selectivity. It is also an instance of the trade-off among read cost, update cost, and memory footprint formalized in the RUM conjecture (Athanassoulis et al., 2016), which holds that no access method can be optimal on all three at once.

D. Buffer Pool Efficiency

Table 3 reports effective cache capacity and hit ratios for a mixed analytical workload consisting of repeated executions of Q-AG, Q-JN, and Q-SR. With 128 GB of buffer pool, the compressed columnstore can hold all referenced LINEITEM columns, plus dimension tables, entirely in memory. The B-tree cannot: its leaf level alone, together with secondary indexes, approaches half of the buffer pool, and concurrent scans compete with the other tables.

Table 3. Buffer pool efficiency under a mixed analytical workload

Metric	A: Clustered B-tree	B: Clustered columnstore
Bytes cached per logical row (referenced columns)	~63 B (full row)	~4 to 7 B
Effective rows cached per GB	~16 million	~150 to 250 million
Steady-state hit ratio, analytical mix	71%	99%+
Pages or segments evicted per query (Q-AG)	~2.1 million pages	near 0
Sensitivity to concurrent OLTP scans	High	Low (separate object cache)

Values illustrative; see Section 2.8.

The effective capacity advantage, roughly ten-fold, matches the prediction of the model: a column ratio m/k of about 2 to 5 for these queries multiplied by a compression advantage of about 4 relative to page compression. An important secondary effect is cache pollution. A full B-tree scan pushes millions of pages through the buffer pool and, unless the replacement policy recognizes the scan, can evict hot pages belonging to transactional workloads. Columnstore segments are read in large sequential units and, in engines that cache them separately, do not displace B-tree pages.

For the point-lookup workload the picture inverts. The upper levels of a B-tree occupy only a few megabytes, and a skewed lookup workload achieves hit ratios near 100 percent with a tiny footprint. A columnstore serving the same lookups must keep entire segments resident to answer queries about individual rows, wasting memory on values that are never returned.

E. Modification Performance

Table 4 summarizes modification costs. Single-row inserts into the columnstore land in the delta store, which is itself a B-tree, so insert cost is comparable to the clustered B-tree. Single-row updates are more expensive because they require locating the row in a compressed segment, marking it deleted in the bitmap, and inserting a new version. Bulk loads strongly favor the columnstore: batches above roughly 100,000 rows

© NOV 2018 | IRE Journals | Volume 2 Issue 5 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/ire1725087

bypass the delta store and are compressed directly into row groups, producing far less log and fewer page writes.

Table 4. Modeled modification cost

Operation	A: Clustered B-tree	B: Clustered columnstore
Single-row insert (latency)	~0.1 ms	~0.12 ms (delta store)
Single-row update (latency)	~0.15 ms	~0.9 ms (delete + insert)
Bulk load, 10 million rows	~95 s	~28 s
Log generated, bulk load	~4.6 GB	~0.7 GB

Values illustrative; see Section 2.8.

F. Logging and Recovery

The two architectures differ in how they interact with write-ahead logging, with consequences for both normal operation and crash recovery.

For a clustered B-tree, each row modification produces a log record describing a change to a specific page, and recovery under ARIES replays these records against dirty pages identified during the analysis pass. Recovery time is proportional to the log volume since the last checkpoint and to the number of random page reads needed to bring dirty pages back into memory. Heavy update workloads on large B-trees, especially those with many secondary indexes, therefore lengthen redo time, because each logical change is logged once per index.

For a clustered columnstore, compressed row groups are written as large objects, and bulk-loaded row groups can be logged efficiently, often as allocations of whole segments rather than individual row changes. This reduces log volume for bulk operations, as Table 4 indicates, and correspondingly shortens redo after a failure that interrupts or follows a large load. However, the delta store and delete bitmap are maintained as ordinary row-oriented structures and are logged conventionally. A columnstore subjected to frequent small updates accumulates delta-store rows and bitmap entries whose redo cost resembles that of a B-tree, and whose presence also degrades query performance until the tuple mover compresses them.

Two operational points follow. First, the tuple mover and index reorganization are themselves logged operations; scheduling them close to a checkpoint, rather than just before a failure-prone maintenance window, limits redo exposure. Second, after recovery a columnstore begins with a cold object cache, but because its working set is small relative to the equivalent B-tree, it reaches steady-state hit ratios faster. In the reference configuration, the analytical working set of the columnstore can be reloaded from NVMe storage in a few seconds, whereas warming the B-tree working set requires reading tens of gigabytes.

G. Assessment of Propositions

The findings support all five propositions. P1 is supported by the point and narrow range results in Table 2, where the B-tree is faster by a factor of roughly 30 to 40 in the warm state. P2 is supported by the reversal for the aggregation and join queries and by the crossover analysis in Section 3.3. P3 is supported by Table 3, where effective capacity rises about ten-fold, consistent with the predicted product of column ratio and compression advantage. P4 is supported by the selective range query, which favors the columnstore only because load order enables segment elimination. P5 is supported by Table 4 and Section 3.6, which show compact logging for bulk loads but row-store-like costs for single-row updates routed through the delta store and delete bitmap. Taken together, the results indicate that neither architecture dominates; the appropriate choice depends on where a workload sits in the space defined by selectivity, column ratio, and update granularity.

H. Implications

Practical Implications

For practitioners, the results support a workload-driven, rather than ideological, choice of physical design. We summarize the guidance as follows:

- Choose a clustered B-tree when the dominant access pattern is point lookup or short range retrieval on a known key, when rows are updated individually and frequently, or when queries usually return most columns of few rows.
- Choose a clustered columnstore when queries scan large fractions of the table, reference a small subset of columns, and aggregate; when data arrives mostly in bulk; and when memory is insufficient to cache the row-oriented working set.
- Order columnstore loads on the most common filter column, usually a date, to make segment elimination effective; without this, selective queries lose much of their advantage.
- Consider hybrid designs. A clustered columnstore with a nonclustered B-tree on a lookup key supports both scans and point access, at the cost of extra maintenance. Conversely, a clustered B-tree with a nonclustered columnstore supports operational analytics on transactional tables, as described by Larson et al. (2015).
- Monitor delta store size and deleted-row fraction in columnstores; when they grow, both query

latency and recovery cost increase, and reorganization should be scheduled.

Automated physical design tools can support these choices. Index and materialized view advisors already weigh scan and seek costs when recommending structures (Chaudhuri and Narasayya, 1997; Agrawal et al., 2000), and bitmap and variant indexes (O'Neil and Quass, 1997) offer intermediate options for data warehouses, which are typically organized around the star schemas described by Chaudhuri and Dayal (1997). Extending such advisors to recommend between clustered B-tree and clustered columnstore organization, using the variables of the conceptual model, is a natural next step.

Theoretical and Research Implications

For research, the study suggests that comparisons of storage architectures should report buffer pool efficiency and recovery cost alongside latency, because the three outcomes respond differently to the same design choice. The conceptual model also reframes the choice between a clustered B-tree and a clustered columnstore as a position in a continuous design space rather than a binary decision, which aligns with hybrid systems such as HyPer (Kemper and Neumann, 2011) and dual-format engines (Lahiri et al., 2015). Finally, the propositions offer a testable structure that empirical studies can confirm or refine across engines and hardware generations.

I. Limitations

This study has several limitations. The analysis assumes a single-node engine and a TPC-H-like schema with a wide fact table; very narrow tables reduce the column elimination benefit. Latency values are model-derived, and real systems exhibit effects the model omits, such as lock contention, memory grant waits, parallelism skew, and plan selection errors. Finally, the reference hardware reflects 2018 commodity servers; larger memories narrow the cold-cache gap but do not change the fundamental CPU efficiency advantage of batch execution.

In addition, the workload spectrum covers five query classes and three modification patterns; real workloads mix these in proportions that change over time, and concurrency between transactional and analytical work can create interference effects not captured by the single-query model. Compression ratios were taken from typical values for the TPC-H data distribution and may differ substantially for skewed or high-cardinality production data. Finally, the study does not evaluate security or governance overheads, such as encryption at rest or auditing, which can affect both latency and log volume.

J. Future Research

Several directions follow from this work:

- Empirical validation of the propositions through measured benchmarks across several engines, using the method of Section 2 so that results can be compared directly with the modeled values.
- Extension of the cost model to concurrent mixed workloads, capturing interference between transactional updates and analytical scans in a shared buffer pool.
- Analysis of in-memory and distributed settings, where network transfer and memory bandwidth replace storage input and output as the dominant cost.
- Study of adaptive and self-tuning physical design, building on database cracking (Idreos et al., 2007) and adaptive replacement policies (Megiddo and Modha, 2003), so that the engine can move a table across the crossover as its workload changes.
- Measurement of recovery time and cache rewarm time as explicit performance metrics, including the effect of tuple mover scheduling and checkpoint frequency on columnstore recovery.

IV. CONCLUSION

Clustered B-trees and clustered columnstores are optimized for opposite ends of the access spectrum. A cost model based on bytes read and per-value processing cost explains the observed differences: the columnstore reduces analytical latency by roughly an order of magnitude and increases effective buffer pool capacity by a similar factor, while the B-tree retains a decisive advantage for point access and individual updates. The crossover occurs at low selectivity, approximately 0.1 to 1 percent, and can be shifted by data ordering, column ratios, and key alignment. Recovery behavior also differs, with columnstores benefiting from compact logging of bulk loads but inheriting row-store costs in their delta structures. Database designers should select physical organization based on measured workload characteristics and should consider hybrid configurations where workloads mix transactional and analytical access.

REFERENCES

[1] Abadi, D. J., Madden, S. R., and Ferreira, M. C. (2006). Integrating compression and execution in column-oriented database systems. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 671 to 682.

[2] Abadi, D. J., Madden, S. R., and Hachem, N. (2008). Column-stores vs. row-stores: How different are they really? In Proceedings of the ACM SIGMOD International Conference on Management of Data, 967 to 980.

[3] Abadi, D. J., Myers, D. S., DeWitt, D. J., and Madden, S. R. (2007). Materialization strategies in a column-oriented DBMS. In Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE), 466 to 475.

[4] Abadi, D., Boncz, P., Harizopoulos, S., Idreos, S., and Madden, S. (2013). The design and implementation of modern column-oriented database systems. Foundations and Trends in Databases, 5(3), 197 to 280.

[5] Adaramola, T. S., Fadero, S., & Gideon, E. N. (2018). Predictive maintenance and condition monitoring in critical power and energy infrastructure. Iconic Research and Engineering Journals, 2(5), 454-477. https://doi.org/10.64388/IREV2I5-1722489

[6] Adeyelu, O. O. (2018). A predictive compliance monitoring framework for detecting systemic safety risks through aviation consumer complaint data. Iconic Research and Engineering Journals, 1(8), 250-276. https://doi.org/10.64388/IREV1I8-1718478

[7] Agboada, P., Ezeako, R., Okoli, I. E., & Uzondu, C. S. (2018). Digital transformation in operations management: A review of advances in automation, analytics, and process redesign. International Journal of Engineering Technology Research & Management, 2(11), 109-134.

[8] Agrawal, S., Chaudhuri, S., and Narasayya, V. R. (2000). Automated selection of materialized views and indexes in SQL databases. In Proceedings of the 26th International Conference on Very Large Data Bases (VLDB), 496 to 505.

[9] Ahmed, K. S., & Odejobi, O. D. (2018a). Conceptual framework for scalable and secure cloud architectures for enterprise messaging. IRE Journals, 2(1), 1-15.

[10] Ahmed, K. S., & Odejobi, O. D. (2018b). Resource allocation model for energy-efficient virtual machine placement in data centers. IRE Journals, 2(3), 1-10.

[11] Ailamaki, A., DeWitt, D. J., Hill, M. D., and Skounakis, M. (2001). Weaving relations for cache performance. In Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), 169 to 180.

[12] Ailamaki, A., DeWitt, D. J., Hill, M. D., and Wood, D. A. (1999). DBMSs on a modern processor: Where does time go? In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB), 266 to 277.

[13] Akeju, B., Edivri, J., Ogbole, J. I., Okoruwa, P. O., Fadayomi, O., & Abolaji, T. O. (2018). Conceptual model for insider threat classification and risk modeling in complex digital systems. Iconic Research and Engineering Journals, 1(9), 476-492. https://doi.org/10.64388/IREV1I9-1713778

[14] Akomolafe, O., & Agu, M. U. (2018). A conceptual model for enhancing internal audit quality through technology-enabled risk assessment frameworks. Iconic Research and Engineering Journals, 1(9), 458-475.

[15] Alagiannis, I., Idreos, S., and Ailamaki, A. (2014). H2O: A hands-free adaptive store. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1103 to 1114.

[16] Amayo, E. B., & Popoola, T. T. (2017b). Scientific study of telecommunication deployment in achieving quality network. International Journal of Trend in Research and Development, 4(5), 311-314.

[17] Arulraj, J., Pavlo, A., and Dulloor, S. R. (2015). Let's talk about storage and recovery methods for non-volatile memory database systems. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 707 to 722.

[18] Arulraj, J., Pavlo, A., and Menon, P. (2016). Bridging the archipelago between row-stores and column-stores for hybrid workloads. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 583 to 598.

[19] Athanassoulis, M., Kester, M. S., Maas, L. M., Stoica, R., Idreos, S., Ailamaki, A., and Callaghan, M. (2016). Designing access methods: The RUM conjecture. In Proceedings of the 19th International Conference on Extending Database Technology (EDBT), 461 to 466.

[20] Azeez, L. O., & Badmus, O. B. (2018). Data-driven framework for predicting subsurface contamination pathways in complex remediation projects. Iconic Research and Engineering Journals, 2(5), 312-335.

[21] Badmus, O., Dosunmu, A. A., & Ozowara, D. E. (2018). A systematic review of CI/CD pipeline strategies in Salesforce DevOps: Tools, practices, and deployment outcomes. Iconic Research and Engineering Journals, 2(6).

[22] Bayer, R., and McCreight, E. (1972). Organization and maintenance of large ordered indices. Acta Informatica, 1(3), 173 to 189.

[23] Boncz, P. A., Manegold, S., and Kersten, M. L. (1999). Database architecture optimized for the new bottleneck: Memory access. In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB), 54 to 65.

[24] Boncz, P., Neumann, T., and Erling, O. (2013). TPC-H analyzed: Hidden messages and lessons learned from an influential benchmark. In Performance Characterization and Benchmarking (TPCTC 2013), Lecture Notes in Computer Science, 8391, 61 to 76. Springer.

[25] Boncz, P., Zukowski, M., and Nes, N. (2005). MonetDB/X100: Hyper-pipelining query execution. In Proceedings of the Conference on Innovative Data Systems Research (CIDR).

[26] Chaudhuri, S., and Dayal, U. (1997). An overview of data warehousing and OLAP technology. ACM SIGMOD Record, 26(1), 65 to 74.

[27] Chaudhuri, S., and Narasayya, V. R. (1997). An efficient, cost-driven index selection tool for Microsoft SQL Server. In Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB), 146 to 155.

[28] Chou, H.-T., and DeWitt, D. J. (1985). An evaluation of buffer management strategies for relational database systems. In Proceedings of the 11th International Conference on Very Large Data Bases (VLDB), 127 to 141.

[29] Comer, D. (1979). The ubiquitous B-tree. ACM Computing Surveys, 11(2), 121 to 137.

[30] Copeland, G. P., and Khoshafian, S. N. (1985). A decomposition storage model. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 268 to 279.

[31] DeBrabant, J., Pavlo, A., Tu, S., Stonebraker, M., and Zdonik, S. (2013). Anti-caching: A new approach to database management system architecture. Proceedings of the VLDB Endowment, 6(14), 1942 to 1953.

[32] Diaconu, C., Freedman, C., Ismert, E., Larson, P.-A., Mittal, P., Stonecipher, R., Verma, N., and Zwilling, M. (2013). Hekaton: SQL Server's memory-optimized OLTP engine. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1243 to 1254.

[33] Effelsberg, W., and Haerder, T. (1984). Principles of database buffer management. ACM Transactions on Database Systems, 9(4), 560 to 595.

[34] Ekemezie, U. C., Ohunyon, G., & Sydney, E. (2018). Evaluating IDSR implementation in Benue State, Nigeria: DHIS2 data quality and surveillance capacity. Iconic Research and Engineering Journals, 2(6).

[35] Färber, F., Cha, S. K., Primsch, J., Bornhövd, C., Sigg, S., and Lehner, W. (2012). SAP HANA database: Data management for modern business applications. ACM SIGMOD Record, 40(4), 45 to 51.

[36] Graefe, G. (1994). Volcano: An extensible and parallel query evaluation system. IEEE Transactions on Knowledge and Data Engineering, 6(1), 120 to 135.

[37] Graefe, G. (2004). Write-optimized B-trees. In Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), 672 to 683.

[38] Graefe, G. (2010). A survey of B-tree locking techniques. ACM Transactions on Database Systems, 35(3), Article 16.

[39] Graefe, G. (2011). Modern B-tree techniques. Foundations and Trends in Databases, 3(4), 203 to 402.

[40] Graefe, G. (2012). A survey of B-tree logging and recovery techniques. ACM Transactions on Database Systems, 37(1), Article 1.

[41] Graefe, G., Volos, H., Kimura, H., Kuno, H., Tucek, J., Lillibridge, M., and Veitch, A. (2014). In-memory performance for big data. Proceedings of the VLDB Endowment, 8(1), 37 to 48.

[42] Gray, J., and Reuter, A. (1992). Transaction processing: Concepts and techniques. Morgan Kaufmann.

[43] Grund, M., Krüger, J., Plattner, H., Zeier, A., Cudre-Mauroux, P., and Madden, S. (2010). HYRISE: A main memory hybrid storage engine. Proceedings of the VLDB Endowment, 4(2), 105 to 116.

[44] Haerder, T., and Reuter, A. (1983). Principles of transaction-oriented database recovery. ACM Computing Surveys, 15(4), 287 to 317.

[45] Harizopoulos, S., Abadi, D. J., Madden, S., and Stonebraker, M. (2008). OLTP through the looking glass, and what we found there. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 981 to 992.

[46] Harizopoulos, S., Liang, V., Abadi, D. J., and Madden, S. (2006). Performance tradeoffs in read-optimized databases. In Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB), 487 to 498.

[47] Hellerstein, J. M., Stonebraker, M., and Hamilton, J. (2007). Architecture of a database system. Foundations and Trends in Databases, 1(2), 141 to 259.

[48] Héman, S., Zukowski, M., Nes, N. J., Sidirourgos, L., and Boncz, P. (2010). Positional update handling in column stores. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 543 to 554.

[49] Holloway, A. L., and DeWitt, D. J. (2008). Read-optimized databases, in depth. Proceedings of the VLDB Endowment, 1(1), 502 to 513.

[50] Idreos, S., Groffen, F., Nes, N., Manegold, S., Mullender, K. S., and Kersten, M. L. (2012). MonetDB: Two decades of research in column-oriented database architectures. IEEE Data Engineering Bulletin, 35(1), 40 to 45.

[51] Idreos, S., Kersten, M. L., and Manegold, S. (2007). Database cracking. In Proceedings of the Conference on Innovative Data Systems Research (CIDR), 68 to 78.

[52] Jiang, S., and Zhang, X. (2002). LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. In Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 31 to 42.

[53] Johnson, R., Pandis, I., Stoica, R., Athanassoulis, M., and Ailamaki, A. (2010). Aether: A scalable approach to logging. Proceedings of the VLDB Endowment, 3(1), 681 to 692.

[54] Johnson, T., and Shasha, D. (1994). 2Q: A low overhead high performance buffer management replacement algorithm. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), 439 to 450.

[55] Kemper, A., and Neumann, T. (2011). HyPer: A hybrid OLTP and OLAP main memory database system based on virtual memory snapshots. In Proceedings of the 27th IEEE International Conference on Data Engineering (ICDE), 195 to 206.

[56] Kersten, T., Leis, V., Kemper, A., Neumann, T., Pavlo, A., and Boncz, P. (2018). Everything you always wanted to know about compiled and vectorized queries but were afraid to ask. Proceedings of the VLDB Endowment, 11(13), 2209 to 2222.

[57] Kester, M. S., Athanassoulis, M., and Idreos, S. (2017). Access path selection in main-memory optimized data systems: Should I scan or should I probe? In Proceedings of the ACM

SIGMOD International Conference on Management of Data, 715 to 730.

[58] Kim, C., Chhugani, J., Satish, N., Sedlar, E., Nguyen, A. D., Kaldewey, T., Lee, V. W., Brandt, S. A., and Dubey, P. (2010). FAST: Fast architecture sensitive tree search on modern CPUs and GPUs. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 339 to 350.

[59] Kimball, R., and Ross, M. (2013). The data warehouse toolkit: The definitive guide to dimensional modeling (3rd ed.). Wiley.

[60] Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. (2018). The case for learned index structures. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 489 to 504.

[61] Ladapo, O. O., Dosunmu, A. A., Jooda, D., & Abolaji, T. O. (2018). Lessons learned from offline assessment of security-critical systems: The case of Microsoft Active Directory. Iconic Research and Engineering Journals, 2(6), 277-299. https://doi.org/10.64388/IREV2I6-1717205

[62] Lahiri, T., Chavan, S., Colgan, M., Das, D., Ganesh, A., Gleeson, M., Hase, S., Holloway, A., Kamp, J., Lee, T.-H., Loaiza, J., Macnaughton, N., Marwah, V., Mukherjee, N., Mullick, A., Muthulingam, S., Raja, V., Roth, M., Soylemez, E., and Zait, M. (2015). Oracle Database In-Memory: A dual format in-memory database. In Proceedings of the 31st IEEE International Conference on Data Engineering (ICDE), 1253 to 1258.

[63] Lamb, A., Fuller, M., Varadarajan, R., Tran, N., Vandiver, B., Doshi, L., and Bear, C. (2012). The Vertica analytic database: C-Store 7 years later. Proceedings of the VLDB Endowment, 5(12), 1790 to 1801.

[64] Lamidi, O. B. A., & Olamide, A. (2018). Spatially explicit risk modeling framework for tracking subsurface contaminant migration in data-limited remediation sites. Iconic Research and Engineering Journals, 2(6), 178-198.

[65] Larson, P.-A., Birka, A., Hanson, E. N., Huang, W., Nowakiewicz, M., and Papadimos, V. (2015). Real-time analytical processing with SQL Server. Proceedings of the VLDB Endowment, 8(12), 1740 to 1751.

[66] Larson, P.-A., Blanas, S., Diaconu, C., Freedman, C., Patel, J. M., and Zwilling, M. (2011). High-performance concurrency control mechanisms for main-memory databases. Proceedings of the VLDB Endowment, 5(4), 298 to 309.

[67] Larson, P.-A., Clinciu, C., Fraser, C., Hanson, E. N., Mokhtar, M., Nowakiewicz, M., Papadimos, V., Price, S. L., Rangarajan, S., Rusanu, R., and Saubhasik, M. (2013). Enhancements to SQL Server column stores. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1159 to 1168.

[68] Larson, P.-A., Clinciu, C., Hanson, E. N., Oks, A., Price, S. L., Rangarajan, S., Surna, A., and Zhou, Q. (2011). SQL Server column store indexes. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1177 to 1184.

[69] Lawal, O. A., & Oduleye, T. E. (2018a). A conceptual model for financial analytics driven enterprise value creation in technology firms. Iconic Research and Engineering Journals, 2(2).

[70] Lawal, O. A., & Oduleye, T. E. (2018b). A review and conceptual framework for tax governance and cross-border compliance analytics. Iconic Research and Engineering Journals, 2(5).

[71] Lehman, P. L., and Yao, S. B. (1981). Efficient locking for concurrent operations on B-trees. ACM Transactions on Database Systems, 6(4), 650 to 670.

[72] Leis, V., Gubichev, A., Mirchev, A., Boncz, P., Kemper, A., and Neumann, T. (2015). How good are query optimizers, really? Proceedings of the VLDB Endowment, 9(3), 204 to 215.

[73] Leis, V., Haubenschild, M., Kemper, A., and Neumann, T. (2018). LeanStore: In-memory data management beyond main memory. In Proceedings of the 34th IEEE International Conference on Data Engineering (ICDE), 185 to 196.

[74] Leis, V., Kemper, A., and Neumann, T. (2013). The adaptive radix tree: ARTful indexing for main-memory databases. In Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE), 38 to 49.

[75] Levandoski, J. J., Lomet, D. B., and Sengupta, S. (2013). The Bw-Tree: A B-tree for new hardware platforms. In Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE), 302 to 313.

[76] Li, Y., and Patel, J. M. (2013). BitWeaving: Fast scans for main memory data processing. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 289 to 300.

[77] Malviya, N., Weisberg, A., Madden, S., and Stonebraker, M. (2014). Rethinking main memory OLTP recovery. In Proceedings of the 30th IEEE International Conference on Data Engineering (ICDE), 604 to 615.

[78] Manegold, S., Boncz, P. A., and Kersten, M. L. (2000). Optimizing database architecture for the new bottleneck: Memory access. The VLDB Journal, 9(3), 231 to 246.

[79] Mbonu, I. S., Aliliele, C., Iwuanyanwu, U., & Oluoha, O. M. (2018). A conceptual framework for legal and ethical risk modeling in enterprise data protection governance systems. Iconic Research and Engineering Journals, 2(2), 207-226. https://doi.org/10.64388/IREV2I2-1714911

[80] Megiddo, N., and Modha, D. S. (2003). ARC: A self-tuning, low overhead replacement cache. In Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST), 115 to 130.

[81] Moerkotte, G. (1998). Small materialized aggregates: A light weight index structure for data warehousing. In Proceedings of the 24th International Conference on Very Large Data Bases (VLDB), 476 to 487.

[82] Mohan, C., and Levine, F. (1992). ARIES/IM: An efficient and high concurrency index management method using write-ahead logging. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 371 to 380.

[83] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., and Schwarz, P. (1992). ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Transactions on Database Systems, 17(1), 94 to 162.

[84] Neumann, T. (2011). Efficiently compiling efficient query plans for modern hardware. Proceedings of the VLDB Endowment, 4(9), 539 to 550.

[85] O'Neil, E. J., O'Neil, P. E., and Weikum, G. (1993). The LRU-K page replacement algorithm for database disk buffering. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 297 to 306.

[86] O'Neil, P., and Quass, D. (1997). Improved query performance with variant indexes. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 38 to 49.

[87] O'Neil, P., Cheng, E., Gawlick, D., and O'Neil, E. (1996). The log-structured merge-tree (LSM-tree). Acta Informatica, 33(4), 351 to 385.

[88] O'Neil, P., O'Neil, E., Chen, X., and Revilak, S. (2009). The star schema benchmark and augmented fact table indexing. In Performance Evaluation and Benchmarking (TPCTC 2009), Lecture Notes in Computer Science, 5895, 237 to 252. Springer.

[89] Obetta, P. C., & Ninduwezuor-Ehiobu, N. (2018). A condition-based maintenance concept for CALM buoy bearings and swivels to improve offshore crude export availability. Iconic Research and Engineering Journals, 2(6), 385-410. https://doi.org/10.64388/IREV2I6-1722579

[90] Obriki, O. D., & Arumosoye, O. M. (2018). Conceptual modeling of data-driven occupational safety risk control in large-scale energy infrastructure projects. Iconic Research and Engineering Journals, 1(7), 169-189. https://doi.org/10.64388/IREV1I7-1714414

[91] Odejobi, O. D., & Ahmed, K. S. (2018b). Performance evaluation model for multi-tenant

Microsoft 365 deployments under high concurrency. IRE Journals, 1(11), 92-107.

[92] Okonkwo, C. S., Ogunwole, O., & Okeke, O. T. (2018b). Model for inventory availability and plant uptime improvement in energy facilities. Iconic Research and Engineering Journals, 2(4), 160-172. https://doi.org/10.64388/IREV2I4-1713120

[93] Plattner, H. (2009). A common database approach for OLTP and OLAP using an in-memory column database. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1 to 2.

[94] Polychroniou, O., Raghavan, A., and Ross, K. A. (2015). Rethinking SIMD vectorization for in-memory databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1493 to 1508.

[95] Ramamurthy, R., DeWitt, D. J., and Su, Q. (2003). A case for fractured mirrors. The VLDB Journal, 12(2), 89 to 101.

[96] Raman, V., and Swart, G. (2006). How to wring a table dry: Entropy compression of relations and querying of compressed relations. In Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB), 858 to 869.

[97] Raman, V., Attaluri, G., Barber, R., Chainani, N., Kalmuk, D., KulandaiSamy, V., Leenstra, J., Lightstone, S., Liu, S., Lohman, G. M., Malkemus, T., Mueller, R., Pandis, I., Schiefer, B., Sharpe, D., Sidle, R., Storm, A., and Zhang, L. (2013). DB2 with BLU Acceleration: So much more than just a column store. Proceedings of the VLDB Endowment, 6(11), 1080 to 1091.

[98] Rao, J., and Ross, K. A. (1999). Cache conscious indexing for decision-support in main memory. In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB), 78 to 89.

[99] Rao, J., and Ross, K. A. (2000). Making B+-trees cache conscious in main memory. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 475 to 486.

[100] Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., and Price, T. G. (1979). Access path selection in a relational database management system. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 23 to 34.

[101] Sikka, V., Färber, F., Lehner, W., Cha, S. K., Peh, T., and Bornhövd, C. (2012). Efficient transaction processing in SAP HANA database: The end of a column store myth. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 731 to 742.

[102] Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., and Zdonik, S. (2005). C-Store: A column-oriented DBMS. In Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), 553 to 564.

[103] Stonebraker, M., and Çetintemel, U. (2005). "One size fits all": An idea whose time has come and gone. In Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE), 2 to 11.

[104] Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., and Helland, P. (2007). The end of an architectural era (It's time for a complete rewrite). In Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB), 1150 to 1160.

[105] Transaction Processing Performance Council. (2017). TPC Benchmark H (Decision Support) Standard Specification, Revision 2.17.3.

[106] Uchechi, N., Ominyi, M., & Jones, B. C. (2018). Procedural standardization and workflow optimization in high-volume dispute resolution clinics: A systematic review. Iconic Research and Engineering Journals, 2(6), 420-452. https://doi.org/10.64388/IREV2I6-1722789

[107] Unamma, U. M.-L., Okwesa, I. J., Iwuanyanwu, U., & Ajiga, D. (2018). An offline-first design framework for digital learning in low-connectivity environments. Iconic Research and Engineering Journals,

2(3), 192-211. https://doi.org/10.64388/IREV2I3-1722627

[108] Westmann, T., Kossmann, D., Helmer, S., and Moerkotte, G. (2000). The implementation and performance of compressed databases. ACM SIGMOD Record, 29(3), 55 to 67.

[109] Willhalm, T., Popovici, N., Boshmaf, Y., Plattner, H., Zeier, A., and Schaffner, J. (2009). SIMD-scan: Ultra fast in-memory table scan using on-chip vector processing units. Proceedings of the VLDB Endowment, 2(1), 385 to 394.

[110] Zheng, W., Tu, S., Kohler, E., and Liskov, B. (2014). Fast databases with fast durability and recovery through multicore parallelism. In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 465 to 477.

[111] Zukowski, M., and Boncz, P. (2012). Vectorwise: Beyond column stores. IEEE Data Engineering Bulletin, 35(1), 21 to 27.

[112] Zukowski, M., Heman, S., Nes, N., and Boncz, P. (2006). Super-scalar RAM-CPU cache compression. In Proceedings of the 22nd International Conference on Data Engineering (ICDE).

[113] Zukowski, M., Nes, N., and Boncz, P. (2008). DSM vs. NSM: CPU performance tradeoffs in block-oriented query processing. In Proceedings of the 4th International Workshop on Data Management on New Hardware (DaMoN), 47 to 54.