JOURNAL OF
Emerging Paradigms
in Computing Systems
JEPCS

Journal of Emerging Paradigms in Computing Systems (JEPCS) | Volume 02, Issue No. 01, 2026

ARTICLE TYPE: Research Article

Article ID: JEPCS-2026-0201-0058

From Flat Files to Structured Data Stores: A Reproducible Benchmark for Enterprise Data Modernization

Shi Chen[1,*]

[1] Independent Researcher, China

*Corresponding author: s.chen2712@gmail.com

Contributing authors: s.chen2712@gmail.com

Received: October 20, 2025 **Accepted:** December 13, 2025

Abstract- Many enterprise modernization projects begin with a simple change: records that were kept as exported flat files are moved into stores that can be queried repeatedly. We measured the local performance consequences of that change for CSV processed with pandas, a compressed HDF5 table, and indexed SQLite. The same digital transaction schema was created at 100,000, 500,000, and 1,000,000 rows, after which six equivalent analytical queries were executed, including a 30-day range filter. One warm-up run preceded three timed runs for each query, and median latency was used for comparison. At one million rows, HDF5 occupied 44.26 MiB rather than 92.09 MiB for CSV and was materialized in 1.81 s instead of 6.69 s. On selective queries it ran 5.18-9.07 times faster than the CSV baseline. SQLite took 209.86 MiB as five indexes were included. Building those indexes brought the 30-day filter down from 2.43 s to 0.10 s and the filtered departmental query from 1.91 s to 0.051 s. The top-product aggregation provides a counter-example: SQLite took 1.77 s, CSV 1.66 s . These measures disentangle two options for modernization that are often discussed together. Binary storage also had a smaller footprint in this benchmark and supported selective reads. Indexed relational storage took more work upfront, but changed the cost of repeated filters and point lookup.

Keywords- Enterprise digital transformation; data modernization; enterprise analytics; HDF5; SQLite

1. Introduction

It is possible for an enterprise to scan a form and retain the scanned work in exported files. If those records start to generate recurring reports or analytics, then how those records are stored is part of the problem of transformation. Vial [1] and Verhoef et al. [2] view digital transformation not as the acquisition of a new tool, but as technological change that is linked to changes in how organizations and value is created. The three reviews cited above (Nadkarni, 2021; Hanelt et al., 2021; Zhu et al., 2021) all come to a similar conclusion from different literatures: technology needs to fit into the existing processes, structures and capabilities.

This larger thesis is directly linked with the data layer. Legacy systems are recognized as a challenge for complex transformations, and coherent enterprise data architecture is tied to a better machine-learning capability, according to Cao and Iansiti [3]. Data, digital platforms, and organizational processes are also included in the same management problem by Han et al. [4]. In day to day implementation, it's a less abstract question. When operational records are reused again and again, the storage choice becomes practical rather than abstract. An enterprise then has to decide whether exported flat files are still sufficient or whether the records should move into a structured store.

CSV remains common for good reasons: the format is portable, easy to inspect, and can be produced by almost any business application. The same simplicity becomes costly when only a small part of a large file is needed, because an

unindexed file is normally parsed before the requested rows can be isolated. A binary table may reduce storage and parsing work, whereas a relational store can add indexes for frequently used keys. Neither choice is free. Compression may save disk space without helping every aggregation, and an index may make a lookup fast even though extra time and storage were spent when the database was built.

Database research gives several pieces of this comparison. Column-oriented systems were developed so that analytical queries would not have to read attributes they did not use [5, 6]. Pavlo et al. [7] showed that loading, execution, and tuning costs can change the ranking of large-scale analytical systems. Lakehouse work later separated durable data storage from the services that query and manage it [8]. But many enterprise teams start much closer to the desktop: they obtain departmental extracts, transaction exports or local databases. At that scale the immediate choice is not which distributed warehouse to buy but what is gained when repeated analysis moves away from a flat file.

We consider that smaller migration step with one fixed workload. The same transaction data are stored as CSV processed with pandas , as compressed HDF5 table , and as SQLite with B-tree indexes . The second configuration utilizes the binary table representation and selection mechanisms afforded by HDF5 [9]. The third uses the in-process relational engine from SQLite . Its storage is row oriented though the engine can perform analytical SQL [10]. For the two file-based cases, pandas offers the common tabular analysis layer [11].

The benchmark is driven by four questions:

- RQ1: investigates how the storage footprint and the materialization time scale as the transaction table grows from 100,000 to 1,000,000 rows.
- RQ2: compares end-to-end latencies for broad scans, aggregations, range filters and selective lookups.
- RQ3: investigates if one representation is still preferred over these query types.
- RQ4: uses the measured differences to discuss migration trade-offs for enterprise data modernization; it does not treat the experiment as a universal ranking of the three tools.

Rather than propose another enterprise architecture, we isolate one decision that appears inside many such architectures. A deterministic generator produces the transaction data, the records are written in three equivalent representations, and six queries with the same semantics are run against each one. We report storage, write time, and query latency from those executions. One result is especially useful for the migration question: the indexed relational case is very strong on selective access, yet it is not the fastest case for every aggregation. Mixed workloads therefore matter more than a single best-case query.

Section 2 gives the digital-transformation and data-management context. Section 3 defines the workload and the three representations, Section 4 records the execution protocol, and Section 5 examines the measured results. The limitations are discussed in Section 6 before the paper is concluded in Section 7.

2. Background and Related Work

2.1. Enterprise Digital Transformation and Data Architecture

Vial [1] conceptualizes digital transformation as a process that changes value creation and organizational structure, and is constrained by barriers within the firm. Verhoef et al. [2] differentiate between digitization, digitalization and digital transformation and relate these stages to resources, structure and performance. The reviews of Hanelt et al. [12] and Zhu et al. [13] broaden that view: enterprise architecture and continuous organizational adaptation are cross-cutting themes in the DT literature.

Cao and Iansiti [3] are closer to the problem studied here as they directly relate digital transformation with data architecture and legacy systems. Their analysis suggests that architecture coherence is one of the capabilities that enable analytics and machine learning to create value. Similarly, Alghamdi [14] considers enterprise architecture as a bridge between transformation goals and technological options. Yet, storage is not enough, research on data governance argues for control, usability and management of the data used for decisions over time [15].

Those studies explain why the data layer is important, but leave open a practical question at the level of local analytics. One department may get a CSV export . A workflow may store an intermediate binary table . An application may store the same records in an indexed relational database . The logical information can be the same , but the cost of writing and querying it may vary . We measure this change with both the schema and the six query semantics fixed.

2.2. Storage Layout and Analytical Workloads

A query is run and the physical layout is shown. Some analytical queries scan many rows, but few columns. Others filter first and aggregate a much smaller result. In column-store research, layout and execution strategy were shown to affect the cost of these operations [5, 6]. Pavlo et al. [7] added an important systems point: loading and tuning must be considered together with execution time. On a single machine the mechanisms are simpler but the distinction is still there. The text must be parsed, binary tables can project or select data, and an index can be used by a relational engine before most rows are touched.

HDF5 offers a flexible binary data model with various storage and access methods [9]. This benchmark uses the table format, via pandas/PyTables, that allows column projection and predicate-based selection. SQLite takes a different path. It is a B-tree based relational DBMS, but historically it has been used for transactional work, and it also supports analytical SQL [10]. By adding indexes, our SQLite configuration changes repeated access from file-style scanning to key- and range-oriented lookup where the query permits it.

We do not model a cloud warehouse or a lakehouse in this experiment. Those systems also have to deal with concurrency, distributed execution, governance, and integration with analytical or machine-learning services [8]. Here, all three configurations fit on one machine, so storage representation and local query behavior can be examined without those additional effects. The scope is intentionally the earlier modernization step, where a local extract begins to be stored in a form designed for repeated use.

2.3. Position of the Present Study

Most database benchmarks begin with the engines and ask which system runs a workload faster. We begin with an enterprise transaction extract instead and ask what changes when the same records are kept in progressively more structured forms. Table 1 places that question beside the prior work used to motivate the benchmark.

Table 1: *Prior work used to frame the storage and enterprise-data comparison.*

Study	Primary focus	Relation to this paper
Abadi et al. [6]	Column versus row stores	Physical-layout evidence used to motivate the analytical comparison.
Pavlo et al. [7]	Large-scale analysis systems	Loading and execution costs considered together in system comparison.
Folk et al. [9]	HDF5 data model and implementation	Storage and access mechanisms used by the HDF5 case.
Gaffney et al. [10]	SQLite architecture and performance	Architectural basis for the embedded relational case.
Cao and Iansiti [3]	DT, data architecture, legacy systems	Enterprise-level link between data architecture, legacy systems, and DT capability.
This study	Local enterprise data modernization	One schema and six query semantics held fixed across CSV, HDF5, and indexed SQLite at three table sizes.

3. Benchmark Design

3.1. Experimental Workflow

We generate one transaction table for each target size and reuse those records across the three storage configurations. After the table has been written as CSV, HDF5, and SQLite, the same six logical queries are executed against each representation. Figure 1 summarizes the sequence. File size and materialization time are recorded when the representations are created; query latency is measured later, after one warm-up execution.

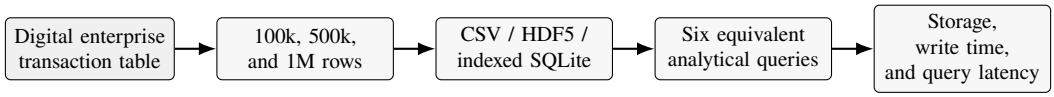


Figure 1: *Benchmark workflow. The same generated records and logical queries are used for CSV, HDF5, and SQLite.*

3.2. Enterprise Transaction Workload

To keep scale, cardinality, and category distributions under our control, the workload is generated rather than taken from a company dataset. NumPy is run with seed 20260301, and three independent tables containing 100,000, 500,000, and 1,000,000 rows are produced. Transactions are dated from January 2024 through December 2025. The fields and their roles are listed in Table 2.

Revenue is derived from transaction status rather than sampled as an unrelated value. Completed transactions receive the full discounted revenue, pending transactions receive one half, and returned or cancelled transactions receive zero. The rule is deliberately simple and is not presented as the accounting policy of a real company. It gives the benchmark a measure that depends on several generated fields and that can be checked across all three storage implementations.

3.3. Data Representations

The CSV + pandas case is our flat-file baseline. For every timed query, pandas parses the CSV anew and the timestamp column is converted to a datetime value before the requested analytical operation is performed.

For the HDF5 table, pandas/PyTables writes table format with Blosc compression level 5. Timestamp, customer, product, region, department, channel, and status are made queryable. Broad queries read only the columns they require, while range and equality conditions are pushed into HDF5 selection before the remaining aggregation is completed in pandas.

Table 2: *Schema of the generated enterprise transaction table.*

Field	Type / cardinality	Generation role
transaction_id	integer	Unique transaction identifier
timestamp	datetime	Uniformly sampled across a two-year interval
customer_id	integer	Customer identifier; cardinality grows with table size
product_id	integer, 5,000 values	Product identifier used in top-product aggregation
region	categorical, 8 values	Geographic reporting dimension
department	categorical, 12 values	Product/business department
channel	categorical, 4 values	Web, mobile, store, or partner channel
quantity	integer, 1-10	Units in a transaction
unit_price	floating point	Log-normal price distribution
discount	6 discrete values	Discount from 0 to 0.30
status	categorical, 4 values	Completed, returned, cancelled, or pending
net_revenue	floating point	Revenue derived from quantity, price, discount, and status
cost	floating point	Cost derived from net revenue and a sampled cost ratio

The SQLite case stores the same fields in one relational table. Once loading is complete, five B-tree indexes are created: timestamp, customer identifier, region, product identifier, and a composite key over region, channel, status, and timestamp. We include both insertion and index construction in SQLite materialization time. The later lookup savings are therefore reported together with the preparation cost that made them possible.

Table 3: *Six logical queries executed against each representation.*

ID	Query	Operation
Q1	Full revenue scan	Sum net revenue over the complete table
Q2	Recent 30-day revenue	Filter December 2-31, 2025 and sum net revenue
Q3	Regional aggregation	Group by eight regions and sum net revenue
Q4	Top products	Group by product, sum revenue, rank, and return the top 10
Q5	Customer lookup	Sum revenue for customer ID 1234
Q6	Filtered department aggregation	Filter region, channel, status, and one-year time range, then group revenue by department

3.4. Analytical Queries

The query set is deliberately mixed. Q1 and Q3 require broad access to the table. Q2 and Q6 filter prior to aggregation, while Q4 groups on a high-cardinality product field and ranks the result. Q5 needs one customer identifier. Each query operates in a precise way, as detailed in Table 3. No representation has a different logical role.

We checked that the three implementations returned same quantities at 100,000 rows before measuring the full benchmark. The CSV result was compared to the HDF5 and SQLite outputs. The relative differences for the reported floating point aggregations were less than 7×10^{-8}. The rest of the variance is because of representation and aggregation order, not because of a difference in the meaning of the query.

4. Experimental Environment

All measurements were made on one Linux virtual machine. It revealed five Intel Xeon Platinum 8573C vCPUs and 5.81 GiB RAM. Software versions were: Python 3.13.5, pandas 2.2.3, NumPy 2.3.5, PyTables 3.10.2, and SQLite 3.46.1. The reproducibility material includes source code, generated measurements and the inputs to the figure.

The transaction table is stored in memory for a given scale and then written separately to CSV, HDF5 and SQLite. The materialization time is measured with a monotonic wall-clock timer, the storage measurement is given by the resulting file size. For SQLite, the timing continues until all 5 indexes have been created and committed. Index creation is included in the cost reported.

The query is executed once for pre-timing and 3 more times for measurement. In the tables and figures we use the median of the three runs. The operating-system cache is kept warm, so the protocol is a repeated local analysis, not a cold-start disk test. Even so, CSV and HDF5 are read from scratch for each timed query . SQLite opens a new connection for every execution, but its pages can also be available from the operating-system cache.

Concurrency is missing from the experiment. No network transfer, cloud object store, distributed query engine, or concurrent user load is added. If we ignore those factors, we can more directly associate the measured differences with local storage and query behavior, but the timings should not be interpreted as performance of the cloud service.

Table 4: *Measured storage footprint and materialization time.*

Rows	Format	Size (MiB)	Write (s)	Size/CSV
100,000	CSV	9.02	0.597	1.00
100,000	HDF5	4.48	0.262	0.50
100,000	SQLite	20.71	0.774	2.30
500,000	CSV	45.94	3.174	1.00
500,000	HDF5	21.91	0.916	0.48
500,000	SQLite	104.55	4.163	2.28
1,000,000	CSV	92.09	6.694	1.00
1,000,000	HDF5	44.26	1.813	0.48
1,000,000	SQLite	209.86	9.176	2.28

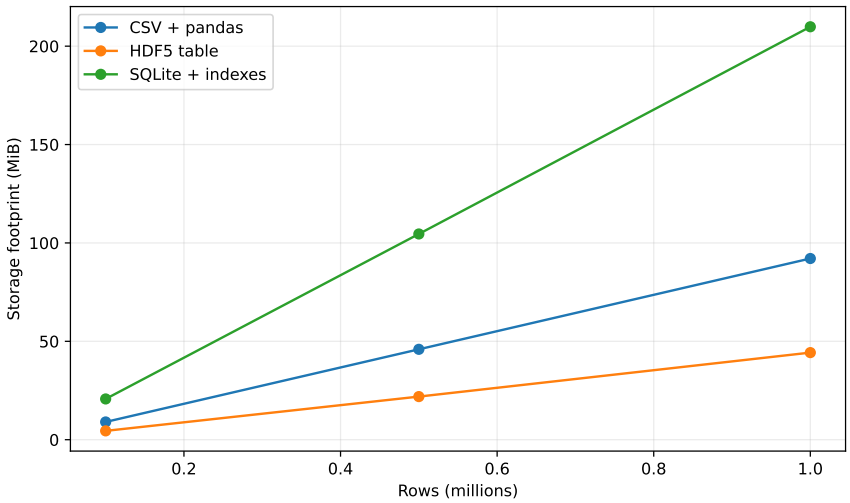


Figure 2: *Storage footprint from 100,000 to 1,000,000 rows; SQLite includes five indexes.*

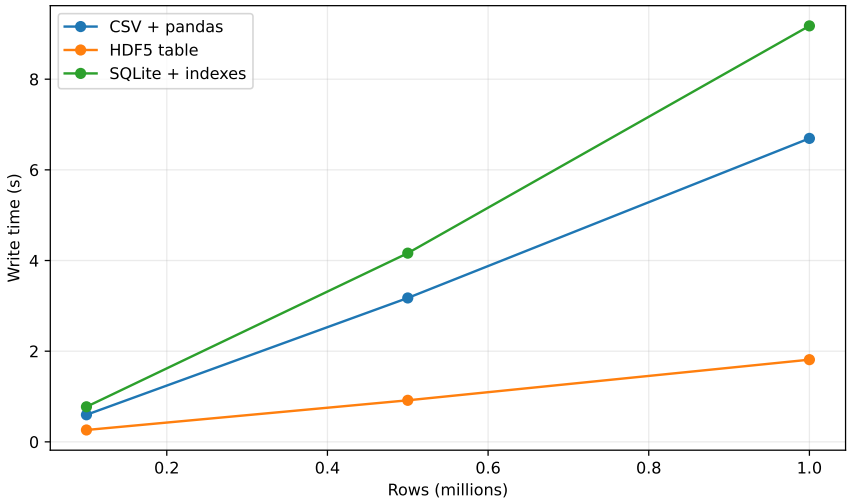


Figure 3: *Materialization time; SQLite timing includes construction of the five indexes.*

5. Results and Discussion

5.1. Storage Footprint and Materialization Cost

As the table increases, the files separate clearly (Table 4). For one million rows, CSV uses 92.09 MiB while HDF5 uses 44.26 MiB, so in this run the HDF5 representation is 51.9% smaller. Indexed SQLite is 209.86 MiB That larger value is the base relational table and the five B-tree indexes that are then used by the selective queries.

We compare the storage and write cost in Figure 2 and Figure 3 respectively. The smallest representation and fastest to materialize at all tested sizes is HDF5. It is 3.69 times faster to write than the CSV baseline at one million rows. SQLite at the same scale takes 9.18 s, 1.37x the CSV write time. The extra time and disk space are paid for before any analytical query is launched, as the indexes are built during loading.

These storage figures are not meant to be taken as a general competition between HDF5 and relational databases. The HDF5 file is compressed. The SQLite file stores extra index structures so that later predicates can avoid broad scans. Those are different design decisions in a migration project. Whether the preparation cost for SQLite is worth it, depends on the subsequent usage of the indexed access paths.

5.2. Query Latency at One Million Rows

At one million rows, no representation is fastest for all six queries (Table 5 and Figure 4). SQLite has the lowest latency for Q1, Q2, Q5, and Q6 and is also lower on Q3, but Q4 is slightly faster with CSV and HDF5. HDF5 stays near CSV on the broad scan and group-by cases; once a predicate removes rows before the pandas step, its advantage becomes much larger.

Table 5: *Median latency of the six queries at one million rows.*

Query	CSV (s)	HDF5 (s)	SQLite (s)
Q1 Full revenue scan	1.825	1.723	0.126
Q2 Recent 30-day revenue	2.432	0.469	0.101
Q3 Regional aggregation	1.834	1.816	0.960
Q4 Top products	**1.662**	1.668	1.770
Q5 Customer lookup	1.628	0.179	**0.00013**
Q6 Filtered department aggregation	1.907	0.375	0.051

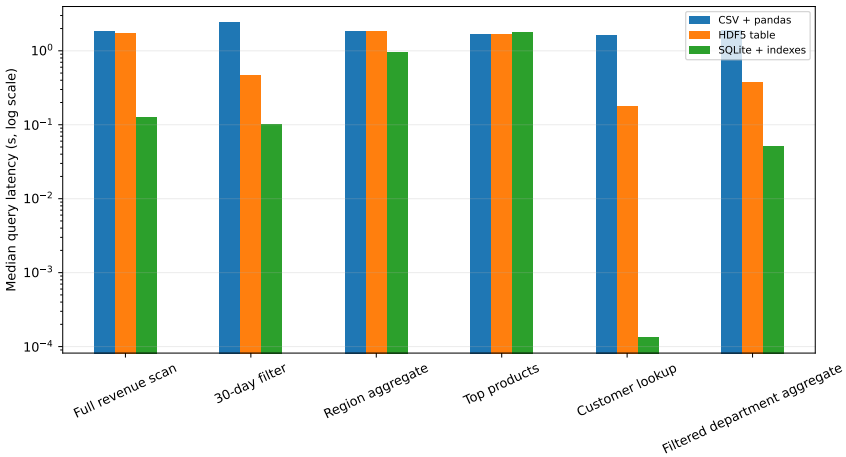


Figure 4: *Query latency at one million rows on a logarithmic vertical axis.*

Q5 produces the widest separation. The CSV path parses the full file before customer 1234 is selected, and its median latency is 1.628 s. HDF5 returns the same logical value in 0.179 s. Under the warm-cache protocol, SQLite uses the customer index and completes the lookup in approximately 0.00013 s. Relative to CSV, that is roughly a 12,200 times speedup for this particular query. The size of the difference is workload specific, but the reason is visible in the access path: the indexed lookup does not scan and parse the entire file.

The range-oriented queries show the same pattern without the extreme Q5 gap. For the 30-day Q2 query at one million rows, latency changes from 2.432 s with CSV to 0.469 s with HDF5 and 0.101 s with SQLite. Q6 changes from 1.907 s to 0.375 s and 0.051 s. Expressed against the CSV baseline, HDF5 is 5.18 and 5.09 times faster on Q2 and Q6, while SQLite is 24.15 and 37.14 times faster.

Q4 is the useful counterexample. Revenue has to be grouped over 5,000 product identifiers and the groups are then ranked. CSV takes 1.662 s, HDF5 1.668 s, and SQLite 1.770 s. The indexes created for selective predicates do not remove the large aggregation in this query, so their storage cost does not buy a latency reduction here. In other words, the same indexed configuration that performs very well on Q5 is slightly slower than the flat-file baseline on Q4.

5.3. How the Advantage Changes with Query Type

Speedup over CSV is plotted in Figure 5 for the one-million-row workload. Across the six queries, HDF5 has a median speedup of 3.07 and a geometric mean of 2.52. Its largest gain is 9.07 times on the customer lookup. The full revenue scan and regional aggregation remain close to CSV because most of the table still has to be read and processed.

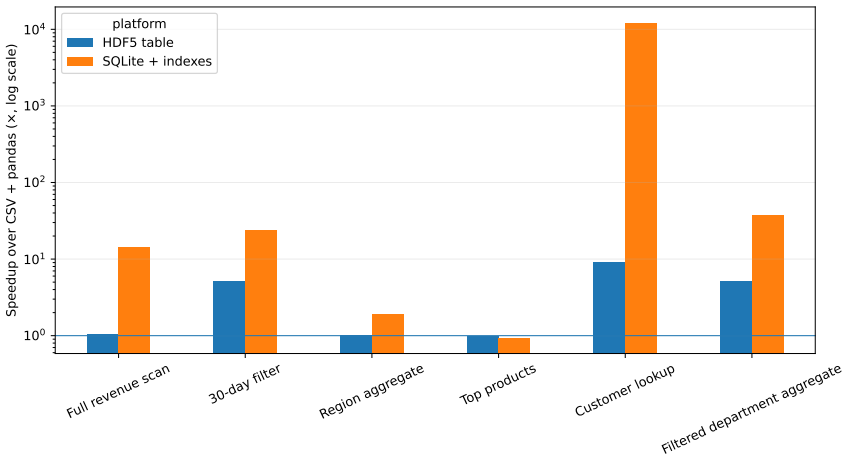


Figure 5: *Speedup relative to CSV + pandas at one million rows; values above one are faster than the baseline.*

SQLite has a median speedup of 19.30 across the same six queries and a geometric mean of 25.63. Those summaries are pulled upward by Q5, where the indexed lookup is several orders of magnitude faster, while Q4 is slightly slower than CSV. For an enterprise workload, the six individual query results are therefore more informative than either aggregate speedup. A system dominated by point lookups and constrained filters will not experience the same benefit as one dominated by full-table ranking and group-by work.

Figure 6 follows Q5 across the three table sizes. CSV latency increases because every lookup includes file parsing. HDF5 also grows with table size, although table selection removes part of the work before the pandas step. SQLite stays around the sub-millisecond range under the warm-cache conditions because the B-tree lookup touches only a small portion of the database. Here, modernization changes the access path itself, not merely the file extension.

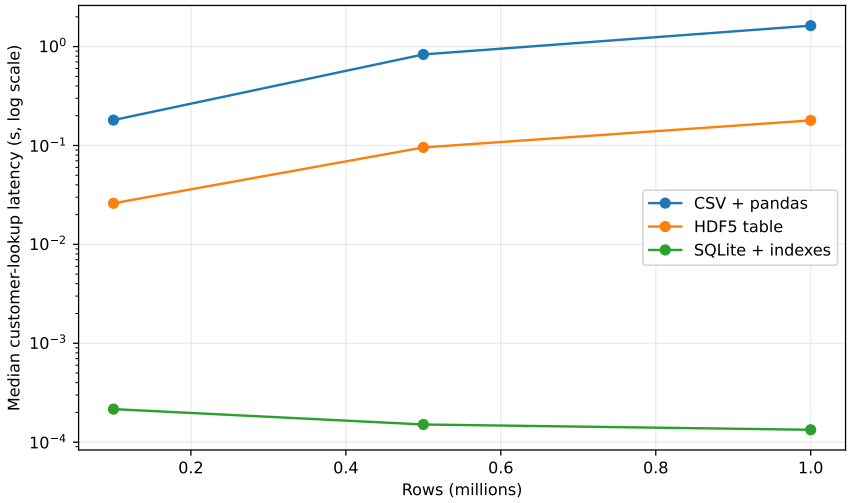


Figure 6: *Customer-lookup latency across table sizes (logarithmic vertical axis).*

5.4. Scaling of the Mixed Workload

To summarize the mixed workload without hiding the individual queries, Figure 7 plots the arithmetic mean of the six median latencies at each scale. It is not presented as a standardized benchmark score; all six queries simply receive equal weight. At 100,000 rows, the means are 0.181 s for CSV, 0.106 s for HDF5, and 0.034 s for SQLite. At one million rows, the corresponding values are 1.881 s, 1.038 s, and 0.501 s.

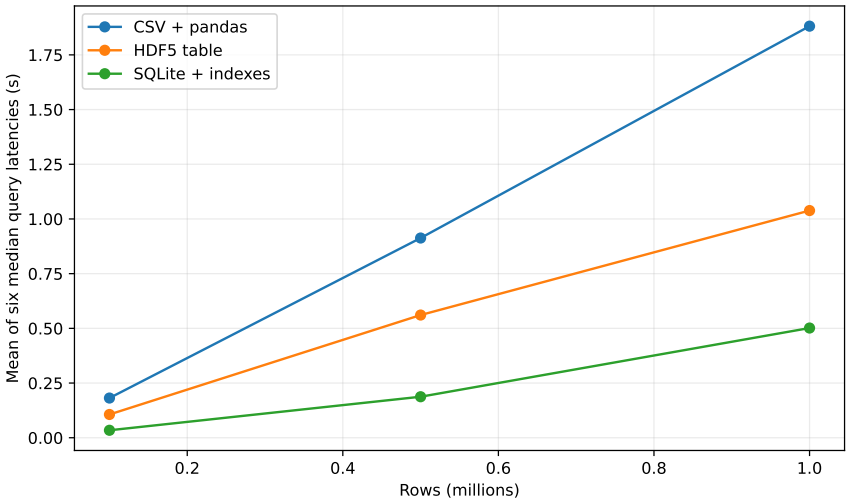


Figure 7: *Arithmetic mean of the six median query latencies at each table size.*

For the scan-heavy queries, the three curves grow roughly with table size. The indexed customer lookup behaves differently, but its absolute latency is so small that it contributes little to the arithmetic mean. This is why the per-query results remain important. An index can change the scaling of the predicate it serves, while an operation that still touches most rows continues to grow with the amount of data processed.

5.5. Implications for Enterprise Data Modernization

HDF5 represents the smaller modernization step in this benchmark. At 500,000 and 1,000,000 rows, its file size is about 48% of the CSV size. It also reduces latency on selective filters when rows are discarded by HDF5 selection before pandas performs the remaining calculation. No relational schema or index design is required to obtain those particular gains.

SQLite changes a different part of the cost profile . More disk space is consumed and materialization takes more time, but repeated access through customer, time-range, status, or other indexed predicates becomes much cheaper. The trade-off is easier to justify when the data will be queried many times. The added preparation may not be recovered for a temporary extract that is written, scanned and discarded once.

The Q4 result underscores the point that a technology label alone is not enough. CSV has the lowest Q4 latency in this experiment, HDF5 is nearly the same. Based on that query alone, there would be little reason to build the indexed SQLite case with the workload. Q2, Q5 and Q6 are a different story because their predicates can use structure that CSV doesn't have. Thus, a migration decision cannot be based on a single overall benchmark number, but requires the expected query mix.

This narrow systems result can however still be placed in the wider DT discussion. Cao and Iansiti [3] define data architecture as a capability with which an enterprise exploits analytics and other digital technologies. These measurements are only of a local storage decision in that capability. They do not measure organizational transformation but quantify the cost of three technical choices that may arise during it.

6. Limitations and Threats to Validity

Digital data make the benchmark controllable, but also limit what the numbers mean. We know the sizes and cardinalities exactly, but the generator does not capture the skew, missing values, schema changes or correlations of every enterprise dataset. The workload should be interpreted as an experiment reproducing a system rather than a model of an industry or company.

All timings are taken from one machine and software stack. The measured seconds may vary with a different CPU, storage device, memory size, filesystem, compression library or operating-system cache. We report the environment so

that a replication can compare the pattern of results, but the absolute timings should not be assumed to transfer unchanged to other hardware.

The timing protocol also promotes repeated local analysis. One warm-up run and three measured runs are performed, without clearing the operating-system cache between runs. Cold-start behavior can differ, particularly when a flat file or database no longer fits comfortably in memory. Three repetitions are sufficient for the compact comparison reported here, but not for a study that bases its main claim on small timing differences or confidence intervals.

Our three storage cases represent only one early step in modernization. HDF5 is a binary tabular structure, not a cloud data warehouse, and SQLite is an embedded row store, not a distributed analytical engine. The environment available did not include Parquet, DuckDB, PostgreSQL, lakehouse systems, cloud object storage and distributed warehouses. The results do not therefore rank modern enterprise platforms or cloud architectures, they compare flat file processing with two structured local alternatives.

The product name is just as important as the configuration of your choice. Five SQLite indexes support the predicates in this workload, and we also report their construction time and storage cost. Another query mix may need another index set and would change the balance of write cost and read cost. The HDF5 results also depend on compression, table layout, and which columns are searchable. We report these settings as part of the benchmark, not as universally optimal choices.

7. Conclusion

We compared three local representations of the same enterprise transaction table: CSV processed with pandas, compressed HDF5 table, and indexed SQLite. Each representation was tested with six queries ranging from broad aggregation to selective customer lookup, at 100k, 500k and 1M rows.

The measurements do not favor one best representation. HDF5 reduced storage from 92.09 MiB to 44.26 MiB and was 3.69 times faster to materialize from the in-memory table with 1 million rows. On the selective operations, it reached 5.18-9.07 times of the baseline CSV. It took 209.86 MiB and 9.18 s for SQLite to load and index the same table with one million rows. This preparation was followed by a 30-day filter that took 0.101 s, the filtered departmental aggregation took 0.051 s and the customer lookup took about 0.00013 s. Q4 was the other way, with the top product aggregation being a little slower than CSV.

For enterprise digital transformation the practical lesson is limited to the workload we measured. The migration from flat files should be considered in light of what queries will actually be repeated after the data is moved. The footprint and parsing cost can be reduced with compact binary storage and indexed relational storage can repay extra preparation when selective access is common. A broader follow-up should include Parquet, DuckDB, PostgreSQL and cloud object storage, repeat the benchmark on public enterprise-scale data, and test cold-cache behavior, concurrency and end-to-end energy use.

Author Declarations

Author Contributions (CRediT): Shi Chen: Conceptualization, Methodology, Software, Investigation, Data Curation, Formal Analysis, Validation, Visualization, Writing - Original Draft, Writing - Review & Editing. The author has read and approved the submitted version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The benchmark uses a deterministic digital enterprise transaction generator. No confidential or proprietary enterprise data were used.

Acknowledgments: No acknowledgments are reported.

Conflicts of Interest: The author declares no conflict of interest.

Ethics Statement: Not applicable.

Declaration of Generative AI: During manuscript preparation, an AI-assisted tool was used for code prototyping, literature organization, LaTeX drafting, and language editing. The author reviewed the benchmark design, executed results, numerical comparisons, citations, and revised text and remains responsible for the accuracy, integrity, and content of the manuscript.

References

[1] G. Vial, "Understanding digital transformation: A review and a research agenda," *The Journal of Strategic Information Systems*, vol. 28, no. 2, pp. 118–144, 2019.

[2] P. C. Verhoef, T. Broekhuizen, Y. Bart, A. Bhattacharya, J. Q. Dong, N. Fabian, and M. Haenlein, "Digital transformation: A multidisciplinary reflection and research agenda," *Journal of Business Research*, vol. 122, pp. 889–901, 2021.

[3] R. Cao and M. Iansiti, "Digital transformation, data architecture, and legacy systems," *Journal of Digital Economy*, vol. 1, no. 1, pp. 1–19, 2022.

[4] X. Han, Y. Hu, L. Wang, and R. Zhou, "Enterprise digital management: Research review, current status and prospects," *Management System Engineering*, vol. 3, p. 8, 2024.
[5] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik, "C-store: A column-oriented dbms," in *Proceedings of the 31st International Conference on Very Large Data Bases*, 2005, pp. 553–564.
[6] D. J. Abadi, S. Madden, and N. Hachem, "Column-stores vs. row-stores: How different are they really?" in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 2008, pp. 967–980.
[7] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, 2009, pp. 165–178.
[8] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," in *11th Conference on Innovative Data Systems Research (CIDR)*, 2021.
[9] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the hdf5 technology suite and its applications," in *Proceedings of the 2011 EDBT/ICDT Workshop on Array Databases*, 2011, pp. 36–47.
[10] K. P. Gaffney, M. Prammer, L. C. Brasfield, D. R. Hipp, D. R. Kennedy, and J. M. Patel, "Sqlite: Past, present, and future," *Proceedings of the VLDB Endowment*, vol. 15, no. 12, pp. 3535–3547, 2022.
[11] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 56–61.
[12] A. Hanelt, R. Bohnsack, D. Marz, and C. Antunes Marante, "A systematic review of the literature on digital transformation: Insights and implications for strategy and organizational change," *Journal of Management Studies*, vol. 58, no. 5, pp. 1159–1197, 2021.
[13] X. Zhu, S. Ge, and N. Wang, "Digital transformation: A systematic literature review," *Computers & Industrial Engineering*, vol. 162, p. 107774, 2021.
[14] H. Alghamdi, "Assessing the impact of enterprise architecture on digital transformation success: A global perspective," *Sustainability*, vol. 16, no. 20, p. 8865, 2024.
[15] B. M. V. Bernardo, H. Sao Mamede, J. M. P. Barroso, and V. M. P. D. d. Santos, "Data governance & quality management: Innovation and breakthroughs across different fields," *Journal of Innovation & Knowledge*, vol. 9, no. 4, p. 100598, 2024.