

Workload-Aware Evaluation of Thread Parallelism and Memory Configuration for Analytical Query Processing in DuckDB

Yalmaz Najmaldeen Taher

Department of Computer Science, College of Computer Science and Information
Technology, University of Kirkuk, Kirkuk, Iraq.

Abstract

Performance of analytical queries can be highly dependent on resource allocation and workload type, making it potentially inefficient to allocate them uniformly. This study examines how the TPC-H workload is processed by DuckDB with varying levels of thread parallelism, dataset size and memory limits. A total of 1,980 executions were captured for all 22 TPC-H queries evaluated on one, two and four threads and at three scale factors (SF=1, SF=5, SF=10). To investigate the memory sensitivity of five structurally different queries at SF=10 under three memory constraints, namely 1 GB, 2 GB and 4 GB, a complementary counterbalanced experiment was conducted yielding a further 135 measurements. We observed that the four-thread execution outperforms the single-thread execution for all 22 queries at all three scales with mean speedups of $\sim 1.81\times$, $\sim 1.95\times$, and $\sim 1.94\times$ at SF=1, SF=5, and SF=10, respectively. Responses to the query, however, were not consistent, and the gains from parallelism (from 2 to 4 threads) were not consistent in all cases. The memory experiment also showed that the performance of Q9 decreased by around 34.0% from 1 GB to 4 GB while the performance of other selected queries was either insignificant, or even improved a little, or even had a plateau. These results suggest that the benefits of extra threads and configured memory can vary by workload, suggesting that query specific behavior should be taken into account when configuring resources for performing analytical queries.

Keywords: DuckDB; Analytical Query Processing; Thread-Level Parallelism; Memory Configuration; TPC-H; Workload-Aware Performance.

1. Introduction

Today, analytical database systems must be able to handle complex queries on large amounts of data while efficiently utilising available computational resources. Modern multi-core processors offer significant potential to improve the performance of analytical query execution by using parallel processing. The cost of the computational resources and the performance of the query is not necessarily linear, however. The key to successful parallel execution is the way the work is decomposed into threads and the way the execution engine makes use of the hardware (Leis et al., 2014). More recent work has likewise highlighted thread-level parallelism as a significant factor in lowering the latency of analytical queries, and

Published: 21 October 2026

DOI: <https://doi.org/10.70558/IJMRS.2026.v2.i4.301237>

Copyright © 2026 The Author(s). This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

demonstrated that it interacts with data-level processing and memory-access patterns is an important performance factor (Schmidt et al., 2025).

DuckDB is a suitable tool for studying these matters, since it has been created particularly as an embedded database system for analytical SQL processing (Raasveldt & Mühleisen, 2019). The analytical capabilities have then been analyzed in various execution environments. DuckDB-Wasm, for instance, adds browser support to DuckDB and has been tested with TPC-H at various scale factors (Kohn et al., 2022). Recently, the MotherDuck architecture has expanded the DuckDB ecosystem in the direction of hybrid client–cloud query processing and highlighted the possibility of single-node execution and allocating CPU cores and memory resources (Atwal et al., 2024). As these advances take place, the importance of efficient utilization of local computational resources becomes more relevant to understanding the analytical performance of DuckDB.

Analytical workloads can however have significantly different performance in different queries. Analytical queries can have various combinations of scans, joins, grouping, ordering, aggregation and nested operations, and they can interact with the actual execution mechanisms in various ways. For example, research on parallel groupjoin execution has demonstrated that contention in join and aggregation workloads can impact efficient parallel processing (Fent et al., 2023). Recent studies have also shown that various operators, like filters and hash joins, can modify the characteristics of data chunks and thus impact the efficiency of vectorized execution and CPU parallelism (Qiao & Zhang, 2025). These results suggest that aggregate workload metrics are not sufficient to characterize the response of a single analytical query to extra processing power.

Another aspect of resource sensitivity is memory availability. Joins, sorting and aggregation are examples of analytical operations that can be a limiting factor for working memory; the memory requirement can differ depending on the workload (Quader et al., 2026). Larger-than-memory processing has been studied in DuckDB recently, and it has been demonstrated that insufficient memory can lead to intermediate data being spilled to storage and that allocations for operators running concurrently can significantly impact performance (Kuiper et al., 2025). Therefore, if there are changes in memory availability or configured memory limits, it should not be assumed that these changes will have the same performance impact on all analytical queries.

These observations have motivated the study of query processing that is more resource-aware and workload-aware. In this case, RAW-HF not only considers the availability of resources in a system, but also the resource usage of individual query tasks to support resource allocation for individual queries based on the characteristics of query workloads (Patel & Bhise, 2024). Meanwhile, ongoing benchmarking studies have shown that there is significant performance variation between analytical engines and workloads in a well-controlled single-node setting. For instance, a recent study on contemporary SQL analytical engines, following the TPC-H benchmark, at the SF=1 and SF=10 levels, on a common multi-threaded single-node platform, revealed significant differences in the execution behavior of the compared systems (Szoke, 2026).

The existing researches, therefore, offer valuable knowledge on analytical execution architectures, parallel query processing, memory management, query optimization, and resource-aware processing. But these works tend to focus on specific execution modes, optimizations, system architectures, or cross-system performance. This provides space for a more detailed empirical study of the behavior of the analytical queries as they are subjected to reasonable variations in thread parallelism and configured memory constraints, as well as in the behavior of the queries themselves, depending on the size of the dataset and the workload characteristics that can be observed. This can be useful to find out if adding more resources give consistent benefits or if they have different benefits in different analytical queries.

This work proposes a controlled experimental analysis of analytical query processing in DuckDB with a TPC-H based workload to address this problem. The 22 TPC-H queries are executed with 1, 2, and 4 threads on three different scales of the dataset (SF=1, SF=5, and SF=10). A counterbalanced memory-sensitivity test also assesses representative queries with DuckDB memory limits of 1 GB, 2 GB, and 4 GB. This measured performance behavior is then compared to observable SQL structural attributes to analyze performance differences across queries while not assuming that the workload is equally responsive to variations in the evaluated thread and memory-limit settings.

Hence, this study aims at answering the following research questions:

RQ1: How does thread-level parallelism affect the execution performance of analytical queries in DuckDB across different dataset scales?

RQ2: To what extent does the performance benefit of thread-level parallelism vary across analytical queries with different workload characteristics?

RQ3: How sensitive are a structurally diverse subset of analytical queries to different memory limits under a counterbalanced experimental setting?

RQ4: What practical resource-configuration insights can be derived from the observed differences in performance behavior among analytical queries?

This study has four major contributions. First, it systematically evaluates all 22 TPC-H analytical queries using all three dataset scales with DuckDB using thread-level parallelism. Second, it offers a workload-aware analysis which is sensitive to the characteristics of these queries, and takes into account parallel speed-up differences between them, while focusing on the varying responses to these queries on heterogeneous systems without the assumption of uniform scalability. Third, it offers a balanced assessment of memory sensitivity for typical analytical queries across various DuckDB memory constraints, minimizing systematic query-order impacts. Lastly, the study arrives at empirically supported conclusions about the potential performance gains of adding more threads or memory for the evaluated queries, offering practical evidence for setting resources for a workload.

2. Related Work

2.1 Analytical Query Processing and Parallelism

The capability of database engines to make use of the latest multi-core processors is becoming more and more vital for efficient analytical query processing. One promising parallelism is

morsel-driven parallelism, where query processing is broken up into pieces of work that can be flexibly scheduled to cores. This method aimed at balancing the load and to adapt parallel computations to the actual hardware, taking into account memory locality and NUMA properties (Leis et al., 2014). This research also demonstrated that the number of processing units available is not the only component of effective analytical parallelism; the coordination of the execution work and data access across the processing units is also important.

Some analytical database engines have also implemented techniques to make the database run more efficiently on the CPU. Vectorized execution works with sets of tuples instead of single tuples and minimises the overhead of interpreting tuples, allowing for better optimisation of modern processors. The vectorized execution in conjunction with query compilation has been shown to offer complementary performance benefits for analytical processing (Lang et al., 2016). Another strategy for improving execution efficiency is to query compilation, but compilation is an overhead that can become significant for short running queries or structurally complex queries. Previous experiments with TPC-H have indicated that lowering such compilation latency can have a significant impact on the overall query execution performance (Funke et al., 2022). The studies have shown that the analytical performance is dependent on the interactions between workload properties, execution mechanisms, and related processing overheads.

Parallel performance may also depend on the relational operations in a query. Joins and aggregations are some of the key components of analytical workloads and parallel processing can create issues that may not be the same for all types of queries. The results of groupjoins and nested aggregates, combined with the results of other types of parallel operations, suggest that parallel execution may be limited by contention in many-core systems and that specialized planning and execution techniques may help improve the performance of these operations (Fent et al., 2023). This means that the more parallel processing capacity is added, the less uniform the performance will be for queries which have varying combinations of joins and aggregation operations.

The following questions have been directly addressed in recent studies of modern analytical engines, like DuckDB. Filters and hash joins have been proven to decrease the number of valid data entries in data chunks, which in turn leaves smaller chunks that might not fully take advantage of the performance benefits of vectorized processing. Dynamic compression techniques for these chunks were then suggested and benchmarked in DuckDB on various analytical workloads, such as TPC-H (Qiao & Zhang, 2025). This is another indication that query operators and intermediate execution attributes can affect the utilization of computational resources.

A more recent study on analytical query processing using a hardware-parallelism point of view distinguishes between three levels of parallelism: thread-level, data-level and instruction-level. In particular, thread-level parallelism can break up an operator, execution pipeline, or input into multiple threads, and data-level parallelism can take advantage of SIMD processing within each thread. The combination of these mechanisms and their differing memory-access patterns can have a significant impact on the efficiency of analytical execution (Schmidt et al., 2025).

Thus, it is not guaranteed that uniformly higher numbers of threads can be assumed to have uniformly greater benefits for different types of queries with varying processing characteristics.

All of these studies show the significance of parallel execution, vectorized processing, query compilation, and the behaviour of operators in the performance of analytical databases (Fent et al., 2023; Funke et al., 2022; Lang et al., 2016; Leis et al., 2014; Qiao & Zhang, 2025; Schmidt et al., 2025). But, their focus is mostly on suggesting or assessing specific implementation methods and optimizations. The present study is complementary in that it is an empirical study of changes to the fixed analytical engine as it processes a complete 22-query TPC-H workload and several dataset scales. This allows analysis to not only determine if there is a performance gain from more parallelism, but determine if the individual analytical queries can consistently gain from higher thread-level parallelism.

2.2 Memory and Resource-Aware Query Processing

Availability of memory is a critical factor in analytical query processing, as operators working on the database might need to use a lot of memory to store intermediate data structures. Memory requirements vary for operations like joins, sorting and aggregation based on the nature and makeup of the workload. The study of workload memory prediction has revealed that memory requirement may differ in the various query templates and workloads, which has encouraged the development of methods that predict working memory requirements instead of a static allocation policy (Quader et al., 2026). This indicates that workload characteristics should be taken into account when configuring memory, not just as a system parameter.

This is particularly significant if intermediate data structures can't be kept in memory at all. This issue has been studied in recent research on hash-join processing in modern analytical database systems where insufficient memory can cause intermediate data to be spilled to secondary storage, with an impact on query performance. To enhance execution performance when memory is limited, adaptive methods for larger-than-memory hash joins have been proposed, such as runtime compression and dynamic memory management (Kuiper et al., 2025). The results show that the performance of analytical operations on memory and memory availability can be tightly coupled.

Resource allocation also concerns other memory than memory alone. Elastic analytical systems have studied mechanisms that can be used to select a way to execute and the level of parallelism based on workload and resource characteristics. For instance, the analytical workloads, such as TPC-H (Bian et al., 2023), have been studied in the context of cost-based decisions for analyzing algorithm and parallelism for the execution of analytical workloads. This is part of a larger trend to make use of the available computational resources for different kinds of workloads, not assuming that a single configuration of resources is suitable for all analytical queries.

The same arguments apply when analytical processing is executed in limited or specific processing contexts. Previous studies about DuckDB in confidential analytical processing settings have found performance to impact by factors like memory available, caching, NUMA behavior, and swapping (Battiston et al., 2024). These environments are not the typical local analytical processing environments, but the results reinforce that the memory and hardware-

resource conditions can also impact query execution behavior, and therefore should be taken into account when evaluating performance metrics.

More explicitly workload-aware approaches have tried to correlate resource allocation to the nature of the individual database tasks. For instance, RAW-HF keeps track of the availability of system resources and the resource consumption of query tasks, which it utilizes to allocate resources based on the workload characteristics for each query (Patel & Bhise, 2024). This is consistent with the general rule that resource needs vary by query and that decisions about resources might be influenced by workload characteristics.

All these studies show that working memory, memory management, parallel execution resources and workload-aware resource allocation are critical elements in analytical query processing (Battiston et al., 2024; Bian et al., 2023; Kuiper et al., 2025; Patel & Bhise, 2024; Quader et al., 2026). But, these studies are primarily on memory-prediction techniques, dedicated memory-management facilities, elastic execution architectures or resource-allocation frameworks. In this paper, we take a complementary experimental approach to examining the performance of representative analytical queries as the memory limit is intentionally modified in the same DuckDB environment. To minimize systematic execution-order effects, a counterbalanced experimental design is used, and only the effect of memory sensitivity is analyzed, which examines whether memory sensitivity is consistent across structurally different analytical queries.

2.3 DuckDB and TPC-H-Based Analytical Evaluation

DuckDB has garnered more and more interest as a system suitable for efficient OLAP processing in an embedded setting. It is designed in such a way that it can be used to process analytical SQL workloads without depending on a separate database server, and also uses techniques designed to be efficient on modern hardware (Raasveldt & Mühleisen, 2019). This design has rendered DuckDB useful for analytical applications, as well as experimental studies to examine the behavior of query execution and optimization.

The usefulness of DuckDB has then been investigated for various execution contexts. DuckDB-Wasm added analytical support to DuckDB and tested its performance on TPC-H workloads of varying scale factors, showing that DuckDB could support analytical queries in a resource-limited client environment (Kohn et al., 2022). Later, MotherDuck expanded the DuckDB ecosystem to hybrid client–cloud analytical processing, where the distribution of the query execution can be determined based on workload and system considerations (Atwal et al., 2024). These studies are indicative of the expanding number of use cases in which DuckDB can be deployed, and also of the continued importance of the analytical workloads it serves and how they behave in response to the computational resources available to them.

DuckDB is also the base for some recent research on query-processing and optimization techniques. There have been a few efforts to replace the traditional join-order-dependent execution, and the idea has been incorporated into DuckDB, which have been tested with known analytical workloads such as the TPC-H (Zhao et al., 2025). Other work has examined theoretically based methods for the evaluation of acyclic queries, and showed that they are applicable to several database systems, including DuckDB (Wang et al., 2025). The results of

these studies indicate that the performance of analytical queries may be significantly affected by execution and optimization decisions, which further justifies query level performance (and not just workload level) analysis.

TPC-H is still popular for benchmarking analytical database systems due to its workload of queries that have varying relational processing and structure. The TPC-H has been used in recent comparative studies of modern analytical and lakehouse engines, with multiple scale factors, but under the same one-node environment (including DuckDB in the tested systems; Szoke, 2026). These evaluations also give valuable information about the differences in system level performance and show that controlled TPC-H experiments are valuable for the study of analytical processing. The focus of cross-system benchmarks is more on the differences between database engines than it is on the differences in the resource sensitivity of individual queries in a given engine.

All of this work has explored various complementary aspects of DuckDB, such as its embedded analytical architecture (Raasveldt & Mühleisen, 2019), browser-based execution (Kohn et al., 2022), hybrid client–cloud processing (Atwal et al., 2024), query optimization (Wang et al., 2025; Zhao et al., 2025), and comparative analytical benchmarking (Szoke, 2026). The above studies, along with the vast amount of research on parallel query execution and memory-aware processing described in the previous sections, offer a solid basis for the study of resource-sensitive analytical performance.

However, relatively little focus has been placed on a controlled query-level evaluation that takes into account thread-level parallelism, dataset size, workload types and memory-limit sensitivity all within a single DuckDB environment. In the present study, this view is taken by examining all of the 22 TPC-H queries for one-, two-, and four-thread configurations, with three different dataset sizes, and augmenting this study with a counterbalanced memory experiment on structurally different representative queries. The study does not suggest a new execution engine or optimization algorithm, but rather it experimentally characterizes the heterogeneous responses to the evaluated thread and memory configurations, and provides practical evidence for workload-aware resource configuration.

3. Experimental Methodology

The methodology used for this study was a controlled experiment to study the impact of thread parallelism, dataset scale, and memory configuration on the performance of analytical queries in DuckDB. The evaluation was performed using the TPC-H analytical workload and both overall trends and the behavior of each query were taken into account. The experimental design was divided into two parts: First, the performance of the thread-level parallelism was analyzed for all 22 TPC-H queries on three data sizes. Second, a focused counterbalanced experiment was designed to study the memory sensitivity of representative queries, while minimizing the effects of query order. The measurements obtained were then analyzed with the characteristics of the structural queries to provide a workload-aware analysis of the observed performance behavior.

3.1 Research Design

The main experiment involved studying the impact of thread-level parallelism and the scale of the data set. Three scale factors (SF=1, SF=5, SF=10) were tested, and each of the 22 queries (Q1–Q22) were run with 1, 2, and 4 threads. One warm-up execution followed by 10 recorded executions for each combination of scale factor, query and thread configuration. A single design generated 1,980 measurements of the execution time (3 scale factors, 22 queries, 3 thread configurations, 10 repetitions).

To explore memory sensitivity in the largest dataset examined, SF=10, and with a fixed number of threads, four was used. Five different queries (Q1, Q6, Q9, Q18, and Q22) were tested across different DuckDB memory limits (1GB, 2GB, and 4GB). After some preliminary measurements, the final memory experiment was a counterbalanced design where each memory configuration was presented in a different position in the execution. Three counterbalanced cycles were conducted; each cycle had three trials per condition for a total of 135 trials were recorded. Each condition had a warm up execution before the recorded runs.

Table 1 summarizes the overall experimental design, which consists of the datasets scales used, the query sets, the type of resources used, the repetitions, and the number of executions recorded.

Table 1. Summary of the Experimental Design

Experimental Factor	Thread-Parallelism Experiment	Memory-Sensitivity Experiment
Dataset Scale	SF=1, SF=5, SF=10	SF=10
Evaluated Queries	Q1–Q22	Q1, Q6, Q9, Q18, Q22
Thread Configurations	1, 2, 4 threads	4 threads
DuckDB Memory Limits	Default configuration	1 GB, 2 GB, 4 GB
Execution Design	Each query evaluated under each thread configuration	Three counterbalanced memory-order cycles
Memory-Order Sequences	—	(1, 2, 4), (2, 4, 1), (4, 1, 2) GB
Warm-up	1 execution per condition	1 execution per condition
Recorded Repetitions	10 per condition	3 per condition in each cycle
Number of Conditions	198	45*
Total Recorded Executions	1,980	135
Primary Measure	Execution time (seconds)	Execution time (seconds)

The 45 memory-test conditions correspond to 5 queries $\times$ 3 memory limits $\times$ 3 counterbalancing cycles, with three recorded executions per condition.

3.2 Experimental Environment

Experiments were performed on a Windows 10 system that had an Intel Core i7-5600U processor running at 2.60 GHz. The processor has 2 physical cores and 4 logical processors, leading to the evaluation of 1, 2 and 4-thread execution configurations. The system was fitted with around 12 GB of physical memory, and a 512 GB 2.5 inch SATA SSD (AGI AI178, model AGI512G17AI178). The Python version was Python 3.13.9 (64-bit) and DuckDB 1.5.6. To ensure a consistent execution environment, the TPC-H databases and experimental result files were kept locally for the entire evaluation. The hardware and software setup employed in all the experiments is summarized in Table 2.

Table 2. Experimental Environment

Component	Experimental Configuration
Operating System	Windows 10 (Build 10.0.19045.6456)
Processor	Intel Core i7-5600U @ 2.60 GHz
Physical Cores	2
Logical Processors	4
Physical Memory	Approximately 12 GB
Local Storage	512 GB 2.5-inch SATA SSD (AGI AI178, AGI512G17AI178)
Python Version	Python 3.13.9 (64-bit)
DuckDB Version	DuckDB 1.5.6
Analytical Workload	TPC-H
Evaluated Scale Factors	SF=1, SF=5, and SF=10

3.3 TPC-H Dataset and Workload

TPC-H is a decision-support benchmark that includes 22 queries that span various structural and computational characteristics, and was used to represent analytically oriented database workloads in the experimental workload (Transaction Processing Performance Council, 2022). The benchmark queries include various types of data access and processing such as filtering, aggregation, grouping, ordering, joins, and nested query structures. The 22 queries (Q1–Q22) were all used for all the evaluations to avoid limiting the evaluations to a small subset of the workload.

Using DuckDB, the TPC-H extension was used to locally generate the experimental databases. Specifically, three databases that are scale factors of 1, 5, and 10 (SF=1, SF=5, and SF=10) were generated using the dbgen functionality. DuckDB's TPC-H extension includes the data-

generation functionality as well as the SQL query definitions; the 22 queries are available via the `tpch_queries()` function (DuckDB, n.d.-b). The generated `LINEITEM` table contained 6,001,215 rows at SF=1, 29,999,795 rows at SF=5, and 59,986,052 rows at SF=10. To see if the performance impact of thread-level parallelism is consistent as the analytical data set grew larger, these three scales were used.

The number of rows in the evaluated dataset and number of rows in the `LINEITEM` table are summarized in Table 3.

Table 3. TPC-H Dataset Scales Used in the Experiments

Scale Factor	LINEITEM Rows	Role in the Experiments
SF=1	6,001,215	Thread-parallelism experiment
SF=5	29,999,795	Thread-parallelism experiment
SF=10	59,986,052	Thread-parallelism and memory-sensitivity experiments

The SQL queries that were run in the experiments were directly retrieved from DuckDB's `tpch_queries()` function. This made sure that the same query definitions were applied uniformly to the various scale factors and resource configurations evaluated. The workload was then described in terms of structural query properties such as the number of tables referred to, the existence of grouping, ordering, aggregation and subquery operations. The above properties were employed in the workload-aware analysis to give a structural context to the analysis of the difference in performance responses between queries.

3.4 Thread-Parallelism Experiment

The purpose of the thread-parallelism experiment was to assess the impact of various levels of parallelism on the performance of analytical queries as the size of the data set grows. The three different thread configurations were studied: 1, 2, and 4 threads. The configurations were chosen to reflect the processing capabilities of the experimental platform, which will have two physical cores and four logical processors. The number of threads used for parallel execution of queries can be set explicitly using the `threads` configuration parameter in DuckDB.

All 22 TPC-H queries were tested for each thread configuration at SF=1, SF=5, and SF=10. There was one warm up and ten recorded executions for each experimental condition. The warm up time was not included in the reported times and the 10 following execution times were kept for statistical analysis. The thread experiment thus included 198 different experimental conditions (3 scale factors $\times$ 22 queries $\times$ 3 thread configurations) and 1,980 query executions were recorded.

The main performance measure used were execution time in seconds. The average, median, standard deviation, minimum and maximum run times were determined for each experimental condition, based on ten runs. Furthermore, to characterize the amount of performance gain achieved with increased thread-level parallelism, speedup over single-thread execution was employed.

3.5 Memory-Sensitivity Experiment

To investigate if there are performance differences when the DuckDB memory limit is changed for analytical queries, a complementary experiment was performed. In this experiment, the largest scale of data (SF=10) was considered, and the number of threads was set to four. Three memory-limit settings were tested: 1 GB, 2 GB, and 4 GB. The values have been set with DuckDB's `memory_limit`. The documentation for DuckDB says that this parameter affects the memory limit of the system; however, the amount of memory that is consumed by the process may be higher than it is set in the parameter, as some memory is not managed by the buffer manager (DuckDB, n.d.-a).

The final memory-sensitivity analysis included five queries (Q1, Q6, Q9, Q18, and Q22). These queries were chosen to form a representative subset with varying structures in the TPC-H workload, from relatively simple single-table queries to queries that join multiple tables, group, order, and have nested query structures. The goal was not to draw general conclusions about the memory characteristics of all the TPC-H queries from this subset, but to determine if the selected queries with diverse structures behave differently when the memory limit is changed.

Initial experiments suggested that the order of execution may affect the time that was observed. Thus, the final experiment was not repeated three times using the same three memory configurations, but rather employed a counterbalanced design. Three execution sequences were used:

Cycle 1: 1 GB → 2 GB → 4 GB

Cycle 2: 2 GB → 4 GB → 1 GB

Cycle 3: 4 GB → 1 GB → 2 GB

This arrangement provided for each memory configuration to be executed in the first, second and third positions over the three cycles, which eliminated systematic bias due to a fixed order of memory configuration execution. Three executions were recorded for each query and memory configuration, after a warm-up execution, within each cycle. The whole experiment was therefore conducted with a total of 135 measurements recorded:

$5 \text{ queries} \times 3 \text{ memory settings} \times 3 \text{ cycles} \times 3 \text{ recorded runs} = 135 \text{ measurements}$

The execution time and relative differences for the three different memory configurations per query were analyzed. Due to the fact that the experiment was run by changing DuckDB's memory limit, instead of observing physical RAM usage, the results are interpreted as "memory-limit sensitivity" and not as "memory-limit usage."

3.6 Workload Characterization

For a workload-aware interpretation of the experimental results, the 22 TPC-H queries were also classified based on the SQL structure that can be observed. The query statements were directly extracted from DuckDB via the `tpch_queries()` function, and analyzed with the same set of structural indicators. This function gives the full list of 22 query definitions that are supported by DuckDB (DuckDB, n.d.-b).

The number of different TPC-H tables used in each query and the use of GROUP BY, ORDER BY, aggregation functions, nested subqueries and EXISTS predicates were considered in the characterization. The selected indicators were chosen because they give a repeatable description of differences in the structure of the query without assuming an internal execution strategy chosen by the database system. Grouping, joining, and sorting may be significant components of processing in an analytical workload, but may not be present in the actual runtime behavior, depending on the generated execution plan and the data processed. In addition to grouping, joining, and sorting, DuckDB documentation lists them as some of the important operators used by analytical queries (DuckDB, n.d.-c).

Table 4 summarizes the characteristics of the five queries used in the focused memory-sensitivity experiment.

Table 4. Structural Characteristics of the Representative TPC-H Queries

Query	Referenced Tables	GROUP BY	ORDER BY	Nested Subqueries	EXISTS
Q1	1	Yes	Yes	0	No
Q6	1	No	No	0	No
Q9	6	Yes	Yes	1	No
Q18	3	Yes	Yes	1	No
Q22	2	Yes	Yes	3	Yes

The structural analysis showed significant variation in the 22 queries. For instance, Q1 refers to only one table and contains aggregation, grouping and ordering without nested subqueries, while Q6 is a relatively simple query on one table, with no aggregation, grouping and ordering or nested subqueries. Q9 on the other hand refers to 6 tables and has grouping, ordering and a nested subquery. Both Q18 and Q22 contain multiple tables referenced, grouping and ordering, as well as a nested subquery. Both Q18 and Q22 include multiple tables referenced, grouping and ordering, and a nested subquery. These differences were then taken into account in conjunction with the values of the parallel speedups measured, to give a structural context to the differences in performance behavior among queries.

Importantly, the structural indicators were not used as direct measures of computational complexity nor were they used as a replacement for physical query-plan analysis. For instance, the more tables or subqueries referenced, the less likely it is that parallel execution is useful. The characterization was used mainly to describe and explain the diversity of workload characteristics, not to create a deterministic relationship between each of the SQL features and their execution performance.

3.7 Performance Metrics and Statistical Analysis

The main performance criterion used in the study was execution time. For the thread-parallism experiment, the ten execution times were recorded for each scale factor, query and thread

configuration after the warm-up execution. Descriptive statistics such as mean, median, standard deviation, minimum and maximum values were determined for each experimental condition. The mean and median were left in place for complementary summaries of central tendency and to help identify potential timing variability.

Speedup over single-thread performance was also used as a measure of parallel performance. $\text{speedup} = \text{execution time of a single thread} / \text{execution time of (n) threads}$, as in equation 1.

$$S_n = \frac{T_1}{T_n} \quad (1)$$

- S_n : The speedup achieved when using (n) threads.
- T_1 : The query execution time using one thread.
- T_n : The query execution time using (n) threads.

where T_1 is the execution time when using one thread, and T_n is the execution time when using (n) threads. Any value over 1 would then mean that the execution is faster than the single thread baseline.

The analysis was done at aggregate level and query specific level. Overall trends across the workload and dataset sizes were identified using aggregate comparisons, while query-level comparisons were used to explore heterogeneous responses that may not have been apparent from workload-wide averages. The measured speedups were then evaluated in conjunction with the structural workload characteristics discussed in Section 3.6. In the counterbalanced memory experiment, execution times were compared for each representative query individually for the three memory-limit settings: 1 GB, 2 GB, and 4 GB. Since the measurements of each experimental cycle were not considered to be perfectly independent, the differences in the measurements were described and the magnitude of the practical effects was emphasized, rather than each timing repetition being treated as an independent statistical sample. A Friedman test was applied separately at each scale factor for the 22 queries as matched experimental units to assess if the differences among the three thread configurations at each scale factor were statistically significant. If a difference in overall means was found, pairwise Wilcoxon signed-rank tests were performed with Holm correction for multiple comparisons. An effect-size measure, Kendall's W, was also reported as an effect. Timing measurements were repeated for each query–configuration condition and were not considered independent experimental units and therefore were not summarized first within each query–configuration condition.

4. Results and Discussion

The experimental results are presented and the impact of the thread-level parallelism, the size of the datasets, the nature of the workloads, and the configuration of the memory on the analytical query performance is discussed. The analysis focuses on the overall behavior as well as on the query-level and scale-dependent behavior. The memory-sensitivity results are then analyzed based on the counterbalanced experiment, and finally, analyzed from a workload-aware resource-configuration perspective.

4.1 Overall Effect of Thread Parallelism

As the number of execution threads increased, analytical query execution time improved in general in the three examined data scale sizes. The mean runtime at SF=1 across Q1-Q22 for one thread was 0.183 s, for two threads it was 0.126 s and for four threads it was 0.099 s, as shown in Table 5. This trend was also seen at SF=5, with the mean runtimes of about 0.963 s, 0.646 s, and 0.493 s. For SF=10, the average runtimes dropped from 2.150 s for one thread to 1.399 s for two threads and 1.134 s for four threads.

Table 5. Mean Execution Time Across Thread Configurations and Dataset Scales

Scale Factor	1 Thread (s)	2 Threads (s)	4 Threads (s)
SF=1	0.183	0.126	0.099
SF=5	0.963	0.646	0.493
SF=10	2.150	1.399	1.134

A Friedman test was used to test for any significant difference in execution time between the three thread configurations for the 22 queries at each dataset size. Table 6 summarizes the results.

Table 6. Statistical Analysis of Thread-Configuration Effects

Scale Factor	Friedman $\chi^2(2)$	p-value	Kendall's W	Effect
SF=1	42.091	<0.001	0.957	Strong
SF=5	42.091	<0.001	0.957	Strong
SF=10	40.364	<0.001	0.917	Strong

The Friedman test showed that there were statistically significant differences between the three thread configurations at all evaluated scales of the datasets ($p < 0.001$). The Kendall's W values corresponding to the above were between 0.917 and 0.957, which showed a high effect of the thread configuration on the queries evaluated. Pairwise Wilcoxon signed rank tests and Holm corrections also showed significant differences between these combinations at each scale factor (adjusted $p < 0.001$).

The query level results also indicated that 4-thread execution outperformed the single-thread execution in all 22 queries at all three scale factors. However, the improvement was varying from query to query. The mean speedup from one to four threads was approximately $1.81\times$ at SF=1, $1.95\times$ at SF=5, and $1.94\times$ at SF=10. The median speedups were roughly $1.88\times$, $2.00\times$ and $2.02\times$, respectively.

The results show the consistent performance gain from one to four threads of parallelism in the evaluated environment. The speedups observed, however, were less than the linear speedups, suggesting that the extra parallelism did not lead to the expected proportional reduction in execution time. This is a natural thing to do as the amount of useful parallelism varies based

on the query and its operators, and because parallel execution adds coordination and resource sharing overhead. Parallelism is also mentioned in the documentation, but it is noted that the performance benefits from parallelism vary depending on the workload and that in certain scenarios more threads can result in either a decrease in performance or even a performance degradation (DuckDB, n.d.-c).

4.2 Dataset Scale and Parallel Speedup

As the number of data in the dataset increased, the relative benefit of four-thread execution was fairly consistent. The mean speedup for 4 threads over 1 thread is shown in Figure 1, the result is from approximately $1.81\times$ at SF=1 to $1.95\times$ at SF=5, and is nearly the same at SF=10 with a mean speedup of $1.94\times$. These median speedups were $1.88\times$, $2.00\times$, and $2.02\times$, respectively. The above results show that the relative advantage of thread-level parallelism was better at SF=5 and SF=10 than at SF=1 and that the average speedup was almost constant between SF=5 and SF=10.

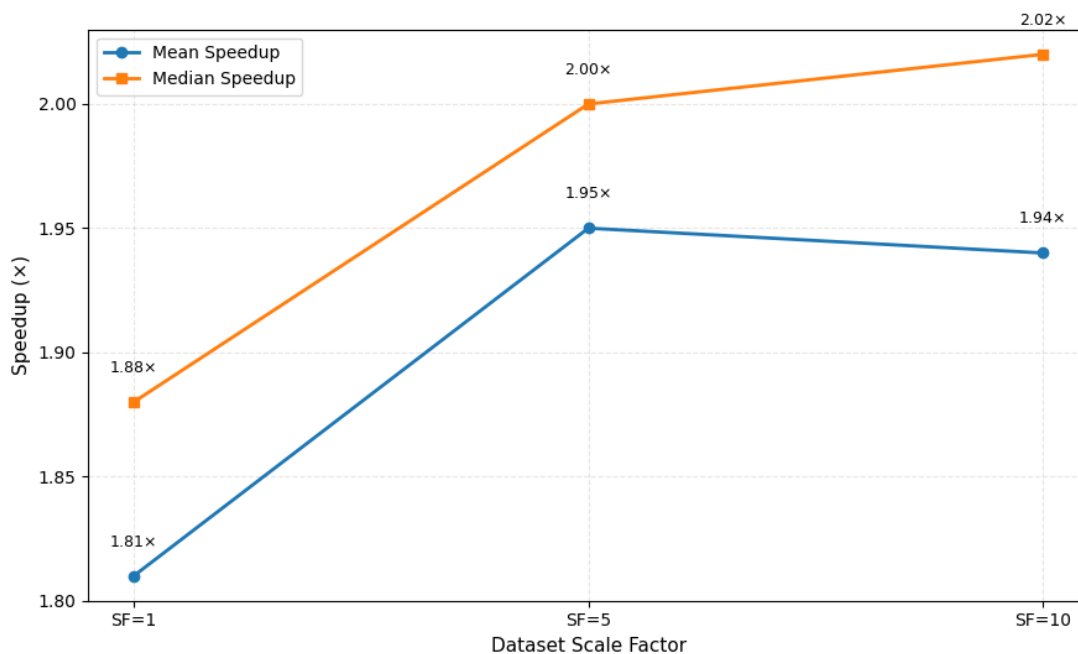


Figure 1. Mean and Median Speedup from One to Four Threads Across Dataset Scales

Moreover, the distribution of query level speedups suggests that the effect of more threads does not depend on the size of the dataset. The 1-to-4-thread speedup ranged from $1.05\times$ to $2.14\times$ at SF=1, from $1.19\times$ to $2.30\times$ at SF=5, and from $1.28\times$ to $2.26\times$ at SF=10. The minimum speedup was greater at the larger scale factors tested, but there was still significant scatter in the results for each individual query. This variation inspired the question level analysis discussed in the next section.

4.3 Query-Level and Workload-Aware Analysis

The benefit of four-thread execution was found to be significantly different at SF=10. Q1 achieved a speedup of approximately $2.26\times$, while Q3, Q5, Q6, Q12, and Q14 achieved speedups of approximately $2.18\times$, $2.11\times$, $2.06\times$, $2.17\times$, and $2.17\times$, respectively. However, Q16

saw a significantly smaller speedup of around 1.28 $\times$, and Q22 saw around 1.63 $\times$. While all queries were helped by 4 threads compared to one, the amount of help was highly variable according to query.

Figure 2 displays the speedup achieved by running the four threads versus the one thread for all 22 queries at SF=10.

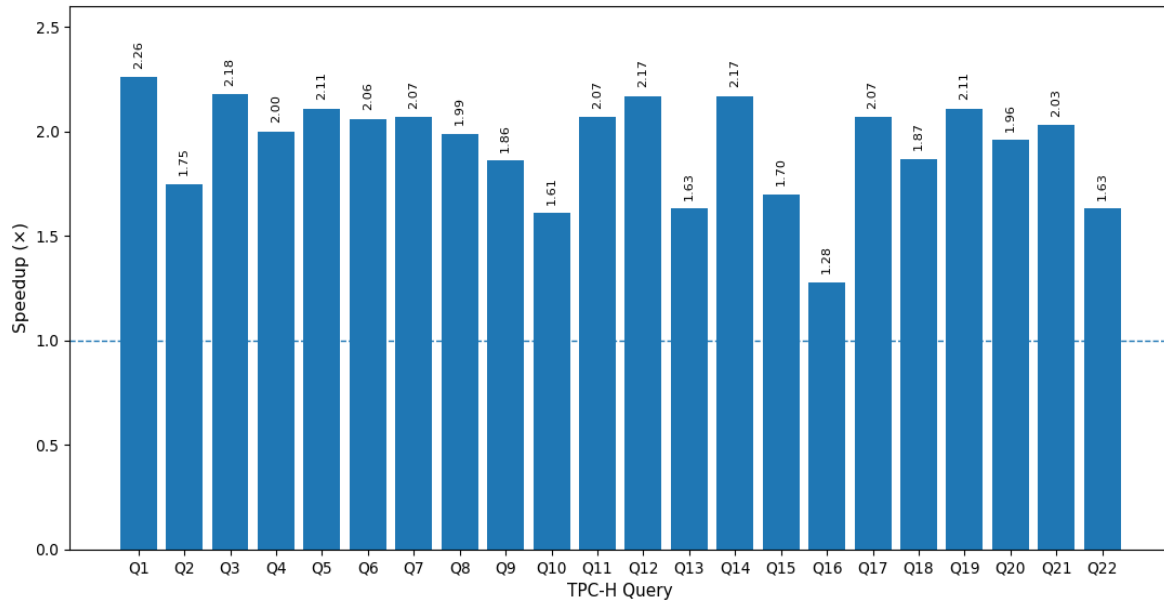


Figure 2. Query-Level Speedup from One to Four Threads at SF=10

The intermediate thread configuration also showed that the four thread configuration did not necessarily result in a further reduction of execution time. In the following, two typical cases with SF=10 are discussed.

Table 7. Execution Times of Q13 and Q16 Across Thread Configurations at SF=10

Query – SF=10	1 Thread	2 Threads	4 Threads
Q13	5.327 s	3.127 s	3.270 s
Q16	0.707 s	0.538 s	0.552 s

With SF=10, Q13 went from ~5.327s (one thread) to 3.127s (two threads) and up to 3.270s (four threads). Likewise the performance of the Q16 dropped from 0.707 s to 0.538 s with increase of threads from 1 to 2 and 0.552 s for 4 threads. The cases show that for some queries, increasing the number of threads beyond four returns less benefit than the single-thread baseline, but there is still a benefit to four threads.

4.4 Memory-Sensitivity Analysis

The counterbalanced memory experiment showed that the impact of raising the DuckDB memory limit was significantly different for the different queries used. Table 8 shows the average runtimes for the five representative queries when SF=10 and four threads are used.

Table 8. Mean Execution Times Under Different Memory Limits at SF=10

Query	1 GB (s)	2 GB (s)	4 GB (s)	General Response
Q1	1.197	1.195	1.167	Limited sensitivity
Q6	0.343	0.343	0.347	Nearly unchanged
Q9	4.232	3.170	2.791	Strong improvement
Q18	3.100	3.013	2.963	Moderate/limited improvement
Q22	0.775	0.721	0.730	Improvement followed by plateau

The differences in memory sensitivity among the five representative queries are further illustrated in Figure 3.

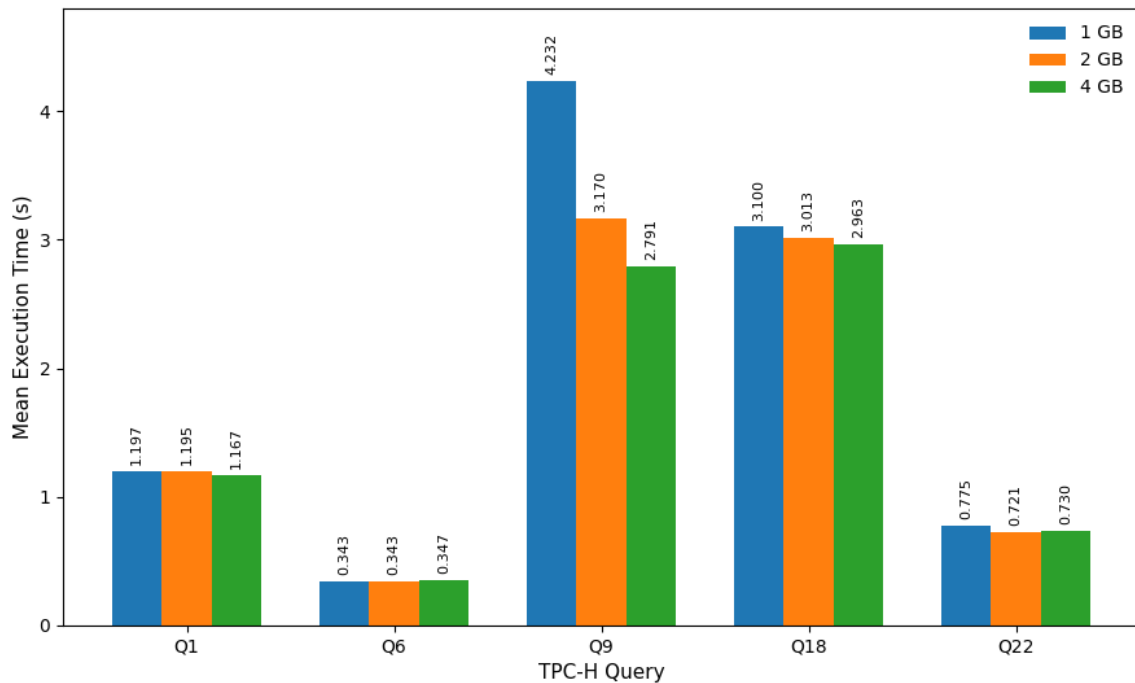


Figure 3. Mean Execution Time of Representative Queries Under Different Memory Limits at SF=10

The results do not indicate a uniform relationship between a larger memory limit and lower execution time. For Q1, increasing the memory limit from 1 GB to 4 GB reduced the mean execution time only slightly, from approximately 1.197 s to 1.167 s. Q6 was even less sensitive, The average execution times are still around 0.34 s for all three configurations. The results indicate that under the tested conditions, the queries did not benefit from the larger memory limit that was configured.

On the other hand, the selected queries showed the most memory-sensitive behaviour in the case of Q9. The execution time, averaged over the 100 runs, decreased from around 4.232 s at 1 GB to 3.170 s at 2 GB and 2.791 s at 4 GB for a reduction in execution time of around 34.0%

between 1 GB and 4 GB. Q18 experienced a less pronounced decrease in time, from around 3.100 s at 1 GB to 2.963 s at 4 GB, a decrease of about 4.4%. In case of Q22, the mean execution time dropped from 0.775 s at 1 GB to 0.721 s at 2 GB, and the same execution time of 0.730 s was obtained when the memory limit was increased to 4 GB. The results in this pattern indicate that there is little performance gain when the memory limit configured on Q22 was increased from 2 GB to 4 GB.

In general, the counterbalanced results show that sensitivity to memory limits ranged significantly across the queries selected. For Q9, a significant increase in the configured memory limit resulted in significantly better performance; for Q1 and Q18, the performance improved rather insignificantly, for Q6 it remained nearly constant, and for Q22 it seemed to be only a little higher. As such, the queries evaluated did not show a consistent performance response to more memory that was added to the configuration.

4.5 Workload-Aware Resource-Configuration Implications

The overall results of the evaluated queries indicate that, when combining the two results, the use of more computational resources does not lead to consistent performance gains in the experimental settings investigated in this research. Execution time was always faster with four-thread execution than with single-thread execution across all 22 queries on all dataset scales. The degree of improvement was also highly dependent on the query, however, and for some queries, the move from 2 to 4 threads did not yield any further improvement in execution time (as seen in the case of Q13 and Q16 at SF=10).

A similar, and in some cases more pronounced, variation was observed in the memory-sensitivity experiment. The execution time for Q9 was significantly reduced with higher memory limits but comparatively small improvements were observed with Q1 and Q18, and Q6 was almost unchanged. The performance of Q22 grew from 1 GB to 2 GB with 4 GB gaining a minimal improvement. The results show that there is no guarantee that the highest thread count/memory limit evaluated in an experiment will lead to any further performance improvement for any query in that experiment.

Combined, the results suggest an approach to resource configuration based on workload, using the potential performance gain of adding threads or memory and balancing it against the actions of individual queries or sets of queries. This information could be used in resource-constrained environments to identify workloads that could gain significantly from increased resources and workloads that could only gain a comparatively small amount from increased resources. This work does not claim to offer an automatic resource-selection mechanism or to uncover optimum configurations, but rather gives empirical evidence of the diversity of reactions to the resource configurations considered and suggests the need for further research on workload-aware resource-management strategies.

5. Threats to Validity and Limitations

The results of this study should be interpreted with a number of limitations in mind. First, all experiments were run on a single hardware platform running DuckDB 1.5.6. The parallel-scaling behavior observed in the evaluated processor might be different on systems with a

larger number of cores, other processor architectures or other storage and memory configuration. The results should therefore be seen as performance under the assessed experimental conditions and not as representative of the performance of DuckDB as a whole.

Second, it evaluated the TPC-H analytical workload with DuckDB TPC-H extension at three data sizes (SF=1, SF=5, SF=10). While all 22 queries are part of the thread-parallelism experiment, TPC-H is a specific set of analytical workloads, and does not cover the full spectrum of real-world analytical workloads. In addition, the workload characterization in this study relied on SQL structural properties that can be observed in the database, and not on detailed physical execution-plan or operator-level measurements.

Third, a warm-up run was used before the runs in the thread-parallelism experiment, but the different thread configurations were determined in a fixed order. This means that there are still residual execution-order and caching effects that can't be ruled out. The repeated measurements and warm-up process give a common standard of comparison inside the experimental environment, but not necessarily remove all of the potential system-level variability.

Fourth, the memory-sensitivity analysis was deliberately restricted to five typical queries for SF=10. A counterbalanced design was chosen because it minimizes systematic effects of execution order that were seen during preliminary testing but cannot be totally ruled out in a local experimental environment caused by operating system caching and other background system effects. Moreover, the configured `memory_limit` in DuckDB does not necessarily represent the total amount of memory used. The memory results are thus only related to the configured memory limit, and not to actual memory usage of the process.

Last, but not least, the execution time is the leading performance measure taken into consideration in this study. The experiment was not designed to make direct measurements of CPU utilization, actual memory consumption, disk I/O or energy consumption. Therefore, the outcomes highlight the diversity of performance outcomes with threads and memory settings, but not the physical nature of all the differences observed. Future work may involve expanding the evaluation to other hardware platforms, more data, other types of analytical workloads, and detailed operator-level resource measurements.

6. Conclusion and Future Work

In this study, the authors examine how thread-level parallelism, database size, and memory allocation impact the processing of analytical queries in DuckDB with a TPC-H workload. All 22 TPC-H queries were evaluated for three different dataset sizes and three different thread configurations with the exception of a counterbalanced test for memory sensitivity on representative queries. The results revealed that the execution time for the queries and scales evaluated showed a consistent reduction in execution time as the number of threads increased from one to four threads. But some queries benefited from a greater improvement than others and some benefitted from more than 2 threads of parallelism but less than 4 threads. The statistical analysis also revealed significant differences between the various thread configurations for all three scales of the datasets.

The memory experiment showed that the memory sensitivity of the selected queries varied significantly. In some cases, the performance of the queries was not sensitive to the memory limit while for Q9, the performance increased significantly with the memory limit. The results of the study show that the use of thread parallelism and configured memory limits has not yielded uniform improvements in performance for the queries and experimental conditions examined. Rather, the observed behavior suggests that workload characteristics should be taken into account when choosing resource configuration when processing analytical queries. Future work will include additional systems with more physical and/or logical core counts, larger data sets and other analytical workloads. Additional experiments might include detailed CPU utilization, real memory used, disk I/O, and physical query operators to explore the root causes of the variations in resource sensitivity among queries. The differences in the queries observed also inspire further research in adaptive workload-aware mechanisms that might select thread and memory configurations based on the characteristics of the query and available system resources.

References

- Atwal, R. J., Boncz, P., Boyd, R., Courtney, A., Döhmen, T., Gerlinghoff, F., Huang, J., Hwang, J., Hyde, R., Felder, E., Lacouture, J., LeMaout, Y., Leskes, B., Liu, Y., Monahan, A., Perkins, D., Tereshko, T., Tigani, J., Ursa, N., Wang, S., & Welsch, Y. (2024). MotherDuck: DuckDB in the cloud and in the client. *Proceedings of the 14th Annual Conference on Innovative Data Systems Research (CIDR 2024)*. <https://www.cidrdb.org/cidr2024/papers/p46-atwal.pdf>
- Battiston, I., Feliuss, L., Ansmink, S., Kuiper, L., & Boncz, P. (2024). DuckDB-SGX2: The good, the bad and the ugly within confidential analytical query processing. In *Proceedings of the 20th International Workshop on Data Management on New Hardware (DaMoN '24)* (Article 14, pp. 1–5). Association for Computing Machinery. <https://doi.org/10.1145/3662010.3663447>
- Bian, H., Sha, T., & Ailamaki, A. (2023). Using cloud functions as accelerator for elastic data analytics. *Proceedings of the ACM on Management of Data*, 1(2), Article 161. <https://doi.org/10.1145/3589306>
- DuckDB. (n.d.-a). *Configuration*. Retrieved October 1, 2026, from <https://duckdb.org/docs/stable/configuration/overview>
- DuckDB. (n.d.-b). *TPC-H extension*. Retrieved October 1, 2026, from https://duckdb.org/docs/stable/core_extensions/tpch
- DuckDB. (n.d.-c). *Tuning workloads*. Retrieved October 1, 2026, from https://duckdb.org/docs/stable/guides/performance/how_to_tune_workloads
- Fent, P., Birler, A., & Neumann, T. (2023). Practical planning and execution of groupjoin and nested aggregates. *The VLDB Journal*, 32, 1165–1190. <https://doi.org/10.1007/s00778-022-00765-x>

- Funke, H., Mühlig, J., & Teubner, J. (2022). Low-latency query compilation. *The VLDB Journal*, 31, 1171–1184. <https://doi.org/10.1007/s00778-022-00741-5>
- Kohn, A., Moritz, D., Raasveldt, M., Mühleisen, H., & Neumann, T. (2022). DuckDB-Wasm: Fast analytical processing for the web. *Proceedings of the VLDB Endowment*, 15(12), 3574–3577. <https://doi.org/10.14778/3554821.3554847>
- Kuiper, L., Groß, P., Boncz, P., & Mühleisen, H. (2025). Saving private hash join. *Proceedings of the VLDB Endowment*, 18(8), 2748–2760. <https://doi.org/10.14778/3742728.3742762>
- Lang, H., Mühlbauer, T., Funke, F., Boncz, P., Neumann, T., & Kemper, A. (2016). Data blocks: Hybrid OLTP and OLAP on compressed storage using both vectorization and compilation. In *Proceedings of the 2016 International Conference on Management of Data* (pp. 311–326). Association for Computing Machinery. <https://doi.org/10.1145/2882903.2882925>
- Leis, V., Boncz, P., Kemper, A., & Neumann, T. (2014). Morsel-driven parallelism: A NUMA-aware query evaluation framework for the many-core age. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (pp. 743–754). Association for Computing Machinery. <https://doi.org/10.1145/2588555.2610507>
- Patel, M., & Bhise, M. (2024). RAW-HF framework to monitor and allocate resources in real time for database management systems. *Software Impacts*, 20, 100643. <https://doi.org/10.1016/j.simpa.2024.100643>
- Qiao, Y., & Zhang, H. (2025). Data chunk compaction in vectorized execution. *Proceedings of the ACM on Management of Data*, 3(1), Article 26. <https://doi.org/10.1145/3709676>
- Quader, S., Jaramillo, A., Mukhopadhyay, S., Abuoda, G., Zuzarte, C., Kalmuk, D., Litoiu, M., & Papagelis, M. (2026). LearnedWMP: Workload memory prediction using distribution of query templates. In *Proceedings of the 29th International Conference on Extending Database Technology (EDBT 2026)* (pp. 69–81). <https://doi.org/10.48786/EDBT.2026.06>
- Raasveldt, M., & Mühleisen, H. (2019). DuckDB: An embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data* (pp. 1981–1984). Association for Computing Machinery. <https://doi.org/10.1145/3299869.3320212>
- Schmidt, L., Pietrzyk, J., Hildebrandt, J., Krause, A., Habich, D., & Lehner, W. (2025). Rethinking MIMD-SIMD interplay for analytical query processing in in-memory database engines. *Proceedings of the 15th Annual Conference on Innovative Data Systems Research (CIDR 2025)*. <https://vldb.org/cidrdb/papers/2025/p7-schmidt.pdf>
- Szoke, M.-A. (2026). Modern alternatives to Hive: A systematic review and single-node benchmark of SQL-on-Hadoop and lakehouse engines. *Information Systems*, 136, 102635. <https://doi.org/10.1016/j.is.2025.102635>

- Transaction Processing Performance Council. (2022). *TPC Benchmark H (Decision Support): Standard specification* (Revision 3.0.1). https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf
- Wang, Q., Chen, B., Dai, B., Yi, K., Li, F., & Lin, L. (2025). Yannakakis+: Practical acyclic query evaluation with theoretical guarantees. *Proceedings of the ACM on Management of Data*, 3(3), Article 235. <https://doi.org/10.1145/3725423>
- Zhao, J., Su, K., Yang, Y., Yu, X., Koutris, P., & Zhang, H. (2025). Debunking the myth of join ordering: Toward robust SQL analytics. *Proceedings of the ACM on Management of Data*, 3(3), Article 146. <https://doi.org/10.1145/3725283>