Workload Insights From The Snowflake Data Cloud: What Do Production Analytic Queries Really Look Like?

Jan Vincent Szlang
Sebastian Bress
Sebastian Cattes
Snowflake, Inc.
Berlin, Germany

Jonathan Dees
Florian Funke
Max Heimel
Snowflake, Inc.
Berlin, Germany

Michel Oleynik
Ismail Oukid
Tobias Maltenberger*
Snowflake, Inc.
Berlin, Germany

firstname.lastname@snowflake.com

ABSTRACT

Capturing the characteristics of real-world analytical workloads is challenging yet critical for advancing industry practices and academic research. Historically, obtaining accurate query and data characteristics has been difficult, largely because detailed workload information has often been confined to on-premises database systems. With the rise of cloud-native databases like Snowflake, it has become possible to analyze production query workloads at scale and in greater detail. Leveraging this capability, this study presents a comprehensive analysis of analytics workloads across diverse customers and industries. In particular, we investigate the query characteristics of 667 million queries issued by the most popular BI tools against Snowflake over a two-week period. Based on this dataset, this paper makes two primary contributions: first, we conduct a detailed examination of query properties, with particular attention to filters, joins, aggregations, and other previously underexplored aspects. Second, we uncover unique and practically relevant query patterns that are typically absent from standard database benchmarks.

PVLDB Reference Format:
Jan Vincent Szlang, Sebastian Bress, Sebastian Cattes, Jonathan Dees, Florian Funke, Max Heimel, Michel Oleynik, Ismail Oukid, and Tobias Maltenberger. Workload Insights From The Snowflake Data Cloud: What Do Production Analytic Queries Really Look Like?. PVLDB, 18(12): 5126 - 5138, 2025.
doi:10.14778/3750601.3750632

1 INTRODUCTION

Understanding and addressing real-world problems is a cornerstone of database research. Yet, gaining visibility into relevant workloads remains a significant challenge. Most real-world query patterns are locked away in proprietary applications or isolated within private infrastructure, limiting visibility and hindering a deeper understanding of how database systems are actually used. To address this lack of visibility, the database community relies on synthetic benchmarks such as TPC-H [24], TPC-DS [23], or Clickbench [6], which aim to replicate the complexity and characteristics of real-world data processing tasks. While these benchmarks offer a valuable framework for evaluating database systems, they fall short of capturing the full diversity and dynamics of actual workloads, leaving many critical patterns unexplored [11, 26, 29].

The emergence of cloud-based data platforms such as Amazon Redshift [3, 13], Google Big Query [20], and Snowflake [9] is surfacing previously inaccessible insights into real-world workloads. Unlike traditional on-premise systems, these platforms operate as a service, enabling providers to collect detailed telemetry on how customers query and interact with their data. This offers unprecedented visibility into workload patterns, helping providers identify key challenges and prioritize engineering efforts that directly address real-world customer needs. Furthermore, some cloud providers have begun sharing workload traces and other insights with the research community, fostering collaboration and validating emerging ideas [26, 30].

In this paper, we share insights gained from over a decade of experience with analytical workloads at Snowflake. Specifically, we analyze a representative production workload consisting of 667 million analytical queries executed by our customers. Using this dataset, we provide an overview of key workload metrics and conduct an in-depth analysis of SQL query patterns, examining their characteristics and frequency in real-world scenarios. Additionally, we identify query patterns that are significant for industrial applications but are not adequately captured in standard benchmarks, highlighting opportunities to refine evaluation frameworks. These findings offer valuable perspectives for understanding real-world workloads and open up new directions for future research.

Our contributions are two-fold:

(1) We analyze a representative analytical workload, focusing on SQL patterns and other properties that have not been extensively studied in the context of cloud data platforms.
(2) By comparing our findings with the TPC-DS benchmark, we highlight opportunities to broaden its scope, offering recommendations to better capture the diversity and complexity of modern analytical workloads.

The primary focus of our study is to highlight how analytical queries are structured and shaped in practice. This emphasis on detailed query properties complements recent workload analyses – such as the analysis of the Redset dataset published by van Renen et al. [26] – which have primarily investigated data characteristics (e.g., table properties, data skew, and value distributions) and broader workload trends. By explicitly analyzing real-world SQL query texts

*Work of Tobias Maltenberger was done when employed at Snowflake Inc., he is now working at Google Inc.

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750632

and execution plans, we provide novel insights into analytical query patterns that complement the results of prior studies. By explicitly analysing real-world SQL query texts and plans – revealing, for instance, that 31% of all statements are metadata queries and that 78% of LIMIT queries still return more than one million rows – we complement the data-centric observations of Redset.

Unlike previous work (e.g., Snowset by Vuppalapati et al. [30] and Redset by van Renen et al. [26]), we explicitly decided against releasing the dataset analyzed in this study. This decision was driven by our emphasis on SQL query texts and query plans rather than abstracted workload traces. Proper anonymization of raw SQL queries and execution plans is particularly challenging, as constants, table names, function properties, or even query shapes might unintentionally reveal customer-specific or personally identifiable information (PII). To mitigate this risk, we prioritize customer privacy and data security, and consequently have chosen not to share detailed query data publicly. Additionally, we currently have no plans to propose new benchmarks based on our results.

The paper is structured as follows: Section 2 provides an overview of the analyzed production workload, highlighting trends and characteristics in real-world queries. Section 3 examines query patterns across SQL components, offering insights into operator usage and complexity. Section 4 discusses key findings on TPC-DS and query patterns. Section 5 reviews related work, and Section 6 concludes.

2 WORKLOAD OVERVIEW

In this section, we provide an overview of the analyzed production workload, including details on its collection process, key properties, and query complexity.

2.1 Methodology

The dataset analyzed in this study comprises query traces, telemetry data, and query plan information collected from Snowflake's internal warehouse, Snowhouse, over a seven-day period (October 21 to 27, 2024). It includes 667 million production queries from a single cloud region, with runtimes ranging from 0.2 seconds to 18 hours (median: 0.7 seconds; 99th percentile: 120 seconds). To ensure our analysis reflects representative analytical workloads, we focused exclusively on queries generated by the most widely used BI tools among Snowflake customers, accounting for more than 60% of all BI queries executed during this week[1]. These queries typically support traditional BI reports, dashboards, and exploratory analyses conducted by data analysts. We deliberately excluded other use cases, such as ETL or data engineering, to maintain consistency and reduce noise. Note that we did not deduplicate the dataset, as the findings remained nearly identical regardless of deduplication.

Our analysis provides a comprehensive view of general characteristics and functional coverage in analytical workloads. It draws on occurrence-based metrics, such as the distribution of query types observed by Snowflake. Note that we explicitly exclude memory and CPU consumption metrics because they reflect system-specific implementation choices rather than the nature of the workload itself. Detailed execution times are also omitted to avoid unintentionally revealing financial details, since Snowflake's billing depends on virtual warehouse runtime. Unless stated otherwise, query attributes such as the number and type of joins are extracted directly from optimized query plans. When deeper insights, such as actual cardinalities, are required, we rely on runtime telemetry collected by the execution engine.

[1]We chose not to publish a more detailed workload specification to avoid disclosing sensitive business information and to respect customer confidentiality.

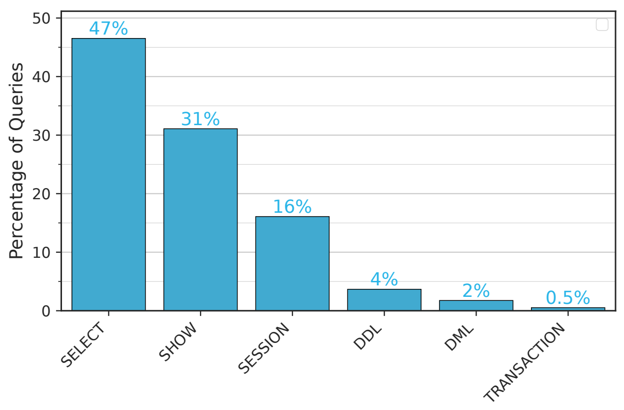

Figure 1: Statement Types

2.2 General Workload Patterns

We will now present an analysis of the workload's characteristics, focusing on statement types, SQL features, and operator usage.

2.2.1 Statement Types. Figure 1 shows the distribution of statement types in the workload. *SELECT* queries dominate (47%), followed by *SHOW* commands, which retrieve metadata about database objects such as tables, views, roles, and warehouses (31%). Together, these account for 78% of all queries, emphasizing the read-heavy nature of analytical workloads. *SESSION* statements, which set parameters like warehouse and schema, make up 16%, while *DDL* and *DML* contribute 4% and 2%, respectively. TRANSACTION statements (*BEGIN*, *COMMIT*, *ROLLBACK*) are rare at 0.5%. Three key observations emerge:

1. Metadata Query Prevalence: The high proportion of SHOW commands (31%) underscores the critical role of catalog performance for BI tools, which frequently access metadata for reporting and visualization tasks. This highlights an area of significant importance in analytical workloads.
2. DDL Queries Outnumber DML Queries: DDL queries (4%) occur twice as often as DML queries (2%). For BI-centric workloads, this ratio is not unexpected and likely reflects the frequent creation and use of views in BI scenarios, where data is often restructured for analysis.
3. Read-Heavy Nature of BI Workloads: With only 4% of queries modifying data, the workload is predominantly read-focused, yielding a read/write ratio of 25:1. This trend reflects the BI-centric nature of our workload, though other studies [26] have observed higher DML proportions in analytical workloads, likely due to different use cases.

Table 1: Prevalence of Query Features

Order By	Join	Limit	Union All
54%	46%	25%	6%
Group By	**Window Functions**	**CTEs**	
55%	11%	25%	

Table 2: Distribution of Operator Types in Queries

Operator Type	Share (%)	Operator Type	Share (%)
Projection	44%	Union All	2%
Filter	16%	Top K	1%
Aggregate	14%	Sort	1%
Table Scan	10%	Window	0.6%
Join	10%	Limit	0.03%

2.2.2 *SQL Features.* Table 1 shows the frequency of core SQL constructs in the workload. *ORDER BY* and *GROUP BY* appear in over half of the queries, highlighting the importance of sorting and aggregation tasks. *Joins* occur in 46% of queries, reflecting the emphasis on integrating data from multiple sources. Common Table Expressions (CTEs), used in 25% of queries, play a key role in structuring complex queries. Similarly, 25% of queries include a *LIMIT* clause, commonly used by BI tools to restrict rows in visualizations. Lastly, 11% of queries employ window functions for advanced analytics such as cumulative sums and moving averages.

2.2.3 *Physical Operators.* We examine the distribution of physical operators in the workload to understand its underlying properties. Table 2 summarizes this data. Core relational operators, such as *Projection*, *Filter*, *Aggregate*, *Table Scan*, and *Join*, dominate, accounting for 94% of operations and highlighting their central role in analytical queries. Transformational operators, including *Union All*, *Sort*, and *Window*, contribute 3.6%, adding complexity by reshaping or reordering data. Finally, *Top K* (1%) and *Limit* (0.03%) are rare but capture important aspects of real-world workloads. Note that Tables 1 and 2 highlight different perspectives: the first shows how often a feature appears in a query (e.g., 25% of all queries utilize *LIMIT*), while the second shows how often a given operator appears overall (e.g., 0.03% of all operator types are *LIMITS*). The same reasoning applies to discrepancies of *JOIN* and window function features compared to their operator counts.

Table 3 presents the CPU time breakdown for operators, based on internal per-operator CPU profiling data collected by Snowflake's execution engine. Comparing this with the occurrence-based distribution from Table 2, we find that although *Table Scans* constitute only 10% of operators, they account for 48.2% of CPU time, including evaluating pushed-down and join filters. Conversely, *Projection* operators represent 44% of occurrences but use just 10.1% of CPU time. Similar trends occur for *Aggregate*, *Join*, and *Sort* operators. Unsurprisingly, scanning and filtering data consume most CPU time, followed by *Joins*, *Projections*, and *Aggregates*. Operator types consuming less than 1% of total CPU time are grouped under the *Other* category.

Table 3: Time Breakdown of Physical Operators

Operator Type	Share (%)	Operator Type	Share (%)
Table Scan & Filter	48.2%	Sort	4.1%
Join	16.4%	Result	1.8%
Aggregate	14.8%	DML Operators	1.2%
Projection	10.1%	Other	3.4%

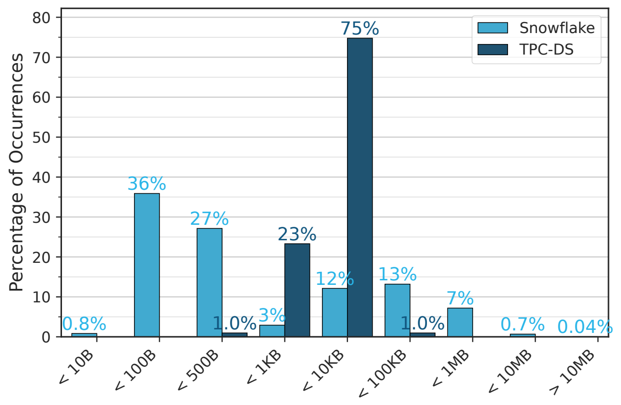

Figure 2: SQL Statement Length in Bytes

2.3 Understanding Query Complexity

SQL statement complexity plays a critical role in query optimization and execution. While database benchmarks often include challenging queries, these are typically well-understood, with established strategies for processing. In contrast, real-world queries can be significantly more complex, making it infeasible to compute optimal execution plans or manually evaluate their quality. To quantify query complexity, we analyze two metrics: (1) SQL statement length and (2) the number of physical operators and expressions in the execution plan.

2.3.1 *Statement Length.* Our analysis of statement lengths includes view definitions, recursively accounting for nested views. Figure 2 shows the distribution of statement lengths in bytes using quasi-logarithmic binning, that is, base 10 buckets with an additional split at 500 B to isolate the large share of sub 500 B statements.

While 67% of queries are under 1KB and 79% are under 10KB, 21% exceed 10KB, with some queries reaching 10MB or more. Although only 0.74% of queries surpass 10MB, database systems must be able to handle even such extreme cases efficiently.

Compared to TPC-DS, which primarily features queries ranging from 500B to 10KB, real-world workloads exhibit a broader range of statement lengths. TPC-DS queries are generally longer on average but lack compact statements (<100B) and extremely large ones (>100KB). This contrast highlights the diversity of real-world queries and the need for systems that can efficiently process both ends of the complexity spectrum.

2.3.2 *Operator and Expression Count.* The number of physical operators and expressions in a query plan approximates for execution complexity. In this study, we assume uniform complexity for each operator, recognizing that some operators may be more expensive than others. Similar proxies have been employed in prior work such

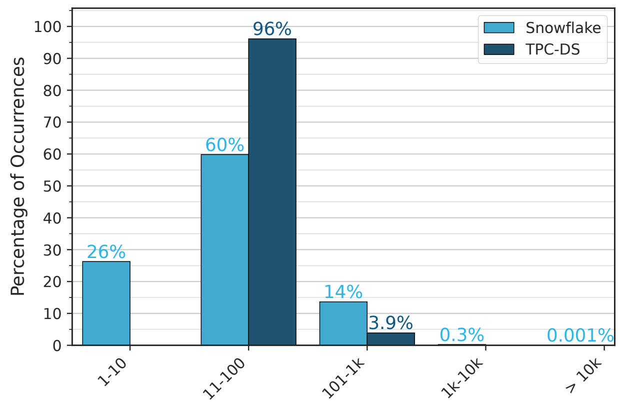

Figure 3: Distribution of Number of Operators per Query

Table 4: Distribution of Projection and Filter Expressions

Projection Expressions			
Category	**1–10**	**11–100**	**101–1K**
Snowflake	40%	46%	14%
TPC-DS	54%	45%	1%
Filter Expressions			
Category	**1–10**	**11–100**	**101–1K**
Snowflake	24%	65%	11%
TPC-DS	54%	45%	1%

as Redset [26], Gupta et al.[14], and Jain et al. [15, 28]. Figure 3 shows that over 13% of queries involve 101 to 1,000 operators, reflecting the complexity of real-world workloads. Similarly, Table 4 illustrates expression counts in projections and filter predicates, with most queries containing a maximum of 100 expressions. A notable proportion (14%) includes 100 to 1,000 expressions, with some queries even exceeding 1,000.

In comparison, TPC-DS queries typically involve fewer operators and expressions, with most containing between 11 and 100 operators and no more than 100 expressions. Unlike the uniform distribution in TPC-DS, real-world queries exhibit a long tail, with a higher proportion of queries featuring extensive operator and expression counts. This distribution underscores the unique challenges posed by real-world workloads and highlights the broader spectrum of query complexity.

3 QUERY PATTERNS

This section examines query patterns observed in our production workload, structured by SQL clause type. We analyze the properties of the *SELECT*, *FROM*, *WHERE*, *GROUP BY*, *ORDER BY*, *LIMIT*, and *UNION ALL* clauses, highlighting their use in real-world queries. Unless specified otherwise, results are presented by occurrence. To provide context for these patterns, we compare them with TPC-DS benchmark queries, illustrating the unique characteristics and challenges of real-world workloads.

Table 5: Distribution of Top Scalar Expressions for Snowflake (SF) and TPC-DS (DS)

Expression	**SF (%)**	**DS (%)**	**Expression**	**SF (%)**	**DS (%)**
ifthenelse	13.9	9.0	variantToText	2.7	0.0
equal	9.0	5.8	variantToNumber	2.3	0.0
extract	8.9	0.0	minus	1.6	3.8
numberCast	6.6	48.4	multiply	1.4	8.2
ifnull	5.5	3.1	greaterThan	0.9	1.4
and	3.0	1.5	divide	0.5	3.1

3.1 SELECT

Our analysis of the *SELECT* clause covers expression complexity, the most commonly used functions, and the distribution of data types for both scalar and aggregate expressions. While analytical window functions are also part of the *SELECT* clause, we omit a detailed discussion of their properties due to space constraints.

3.1.1 Scalar Expressions. Table 5 highlights the most frequently used functions in scalar expressions. Our workload demonstrates a strong prevalence of conditional logic (*ifthenelse*: 13.9%, *ifnull*: 5.5%) and temporal functions (*extract*: 8.9%), alongside various casting and text manipulation functions, such as *numberCast* and *variantToNumber*. These functions reflect the application-driven nature of modern workloads, where logic and transformation are frequently pushed into the database. Additionally, the complexity of expressions is evident in the depth of expression trees, as shown in Figure 4. While 88% of expressions in our workload have a depth of 10 or less, approximately 12% exhibit depths between 11 and 100, and a small fraction exceeds 100. Such deeply nested expressions highlight the challenges posed by hierarchical conditions and complex transformations, which place increased demands on query planners and execution engines.

Compared to our production workload, TPC-DS queries tend to involve simpler expressions, dominated by numerical casts (*numberCast*: 48.4%) and arithmetic functions, including *minus*, *multiply*, and *divide*. TPC-DS queries rarely feature deeply nested expressions, with most expressions having a depth of 10 or less. This contrast underscores the need for benchmarks like TPC-DS to evolve, incorporating characteristics such as functional diversity and structural complexity observed in real-world workloads, to better reflect the demands of modern analytical systems.

Table 6: Aggregation Functions Used (Top 5)

	anyvalue	**sum**	**count**	**max**	**min**
Snowflake (%)	58	15	12	11	5
TPC-DS (%)	13	64	21	0.4	1

3.1.2 Aggregate Expressions. Table 6 shows that *anyvalue*[2] is the most common aggregation function in our workload, used in 58% of queries, followed by *sum* (15%), *count* (12%), *max* (11%), and *min* (5%). Figure 5 highlights the diversity of column types processed by these functions. While numeric data remains the primary target

[2]anyvalue returns a single non-deterministic value from a group of rows. BI tools use it to add descriptive columns to aggregate queries without bloating the GROUP BY.

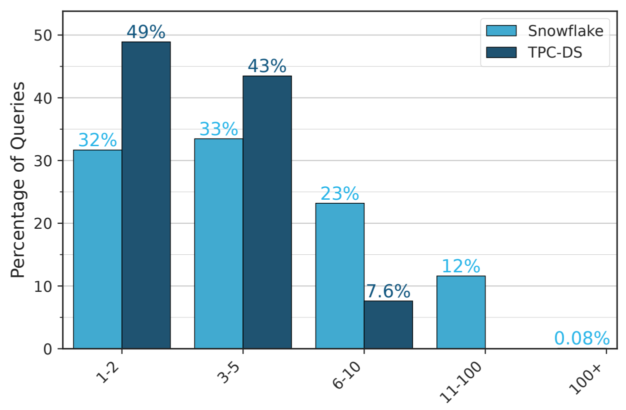

Figure 4: Scalar Expression Tree Height

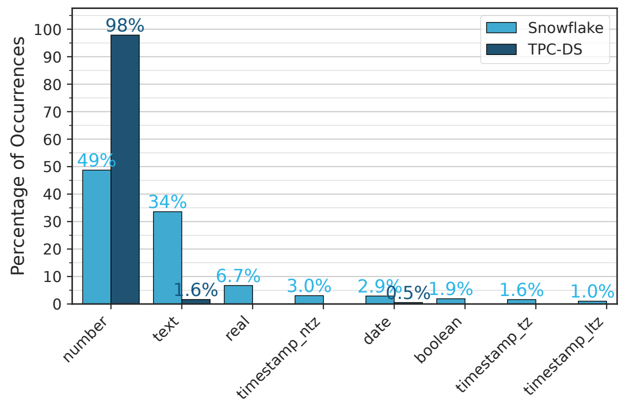

Figure 5: Logical Types used by Aggregation Functions

(49%), text columns account for 34% of aggregations, with smaller proportions for *real* (7%) and *timestamp* (3%, *ntz*). These results demonstrate that real-world workloads rely heavily on processing a wide range of data types, with significant emphasis on non-numeric inputs such as *text*.

In comparison, TPC-DS queries show a starkly different pattern. Numeric types dominate (98%), with minimal use of *text* (2%). Similarly, *sum* (64%) and *count* (21%) are the most frequent aggregation functions, while *anyvalue* accounts for just 13%. These differences highlight the limitations of TPC-DS in representing the functional and data type diversity of real-world workloads, where non-numeric data and less conventional functions like *anyvalue* play a critical role.

3.2 FROM

For our analysis of the *FROM* clause, we focus on the properties of joins in our production workload, discussing their types, the number of joins per query, key types, and the observed input cardinalities and join selectivities.

3.2.1 Join Types. Table 7 shows the distribution of join types in our workload. While inner joins are the most common (59%), outer joins account for a substantial 37%, followed by semi (3%) and anti (1%) joins. In contrast, TPC-DS queries are dominated by inner joins

Table 7: Distribution of Join Types

	INNER	OUTER	SEMI	ANTI
Snowflake (%)	59	37	3	1
TPC-DS (%)	82	3	13	1

Table 8: Distribution of Number of Joins per Query for Snowflake (SF) and TPC-DS (DS) Workload

Range	SF (%)	DS (%)	Range	SF (%)	DS (%)
1-2	34.0	9.0	31-50	1.8	0.0
3-5	27.0	45.0	51-100	1.0	0.0
6-10	19.0	32.0	101-500	0.5	0.0
11-20	12.0	13.0	501-1000	0.02	0.0
21-30	5.0	1.0	1000+	0.0003	0.0

(82%) and underrepresent outer joins (3%). This discrepancy highlights the greater reliance of real-world workloads on outer joins to maintain completeness in reports and analyses by preserving information about entities without matching counterparts. Efficient implementations of all join types, particularly outer and semi joins, are essential to handle the diverse and complex scenarios found in real-world queries.

3.2.2 Number of Joins per Query. Table 8 shows the distribution of joins among queries with at least one join. Most queries are simple, with 34% containing one or two joins, 27% having three to five joins, and 19% featuring six to ten joins. Notably, 20% of queries include ten or more joins, with a small fraction (0.52%) exceeding 100 joins and some even surpassing 1,000. In contrast, TPC-DS queries rarely exceed ten joins, underrepresenting the occasional complexity of real-world workloads, where a notable fraction of queries involve significantly larger join graphs.

These results highlight the significant demands placed on query optimizers in production workloads. Static optimization techniques, while sufficient for simpler queries, struggle to handle large-scale joins effectively. Complex queries with extensive join graphs require dynamic optimization strategies, such as adaptive join ordering, runtime partitioning, and parallel join execution, to avoid performance bottlenecks. Advanced runtime methods, like Holistic Join Broadcast Decisions [8], emphasize the importance of adaptivity in managing skewed intermediate results and optimizing memory usage for large join workloads.

3.2.3 Join Key Types. Our analysis of key types focuses on equality-based join predicates, either single conditions (e.g., $T_1.a = T_2.a$) or conjunctions (e.g., $T_1.a = T_2.a \land T_1.b = T_2.b \land T_1.c = T_2.c$). These predicates dominate analytical workloads and provide a clear framework for evaluating the distribution of join keys.

Figure 6 shows that nearly half of all join keys in our workload use text-based types, while 39% rely on fixed integer types, with the remainder spanning date and timestamp columns. For physical data types (Figure 7), 47% are LOBs (internal text format), and integer keys are distributed across sizes, including 32-bit (20%), 16-bit, and 8-bit. In contrast, TPC-DS predominantly uses 32-bit integer keys,

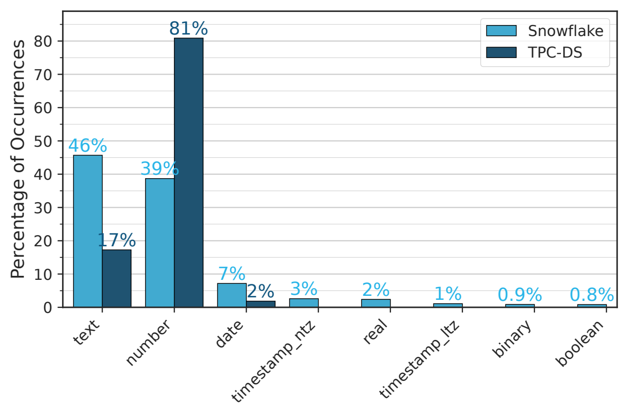

Figure 6: Join Keys Logical Types

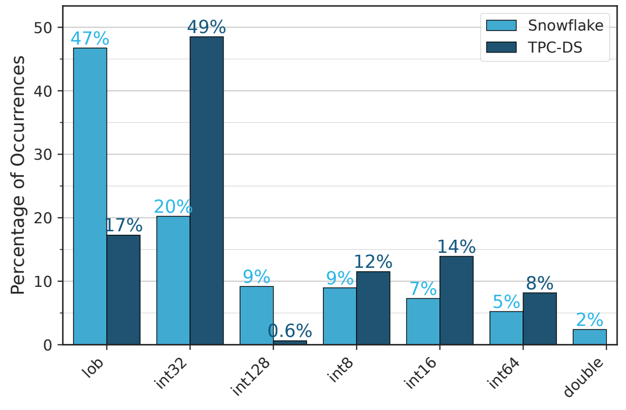

Figure 7: Join Keys Physical Types

which make up 49% of all join keys in its queries, compared to just 17% that join on text keys.

This discrepancy highlights the importance of optimizing join performance under diverse conditions, particularly for text keys, which dominate real-world workloads. Compression techniques, such as dictionary encoding to map text keys into compact integer formats, can significantly improve efficiency. Additionally, join algorithms must be evaluated across a broader range of key types to ensure robustness. Expanding benchmarks like TPC-DS to include a broader variety of key types, particularly text keys, would enhance their ability to model the complexities of real-world analytical workloads and provide a more comprehensive foundation for evaluating join algorithms.

3.2.4 Join Input Cardinalities. Accurately measuring join input cardinalities is challenging due to runtime optimizations such as pruning, bloom filters, and dynamic re-partitioning, which can significantly alter observed input sizes. These optimizations, while improving execution performance, complicate efforts to capture logical input sizes or account for adjustments applied before the join. As a result, the measurements reported here reflect observed execution behavior but may not fully represent the logical input cardinalities.

Table 9 presents the distribution of join input cardinalities for both the build and probe sides, covering a range from fewer than 10

Table 9: Join Input Cardinalities for Snowflake (SF) and TPC-DS (DS) after Evaluation of Pushdown Filters

Size Range	Build Input		Probe Input	
	SF (%)	DS (%)	SF (%)	DS (%)
<10	27	6	13	3
<100	18	20	11	2
<500	12	13	9	1
<1K	5	4	4	1
<10K	16	12	18	3
<100K	12	10	17	4
<1M	7	15	17	12
<10M	2	7	8	15
>10M	0.7	13	3	58

rows to over 10 million rows. On the build side, 78% of inputs have fewer than 10,000 rows, reflecting a skew toward smaller inputs. The probe side, however, exhibits a more balanced distribution, with 45% of inputs exceeding 10,000 rows. These differences highlight the need for join algorithms that handle asymmetry effectively, where one side of a join may be orders of magnitude larger than the other.

Compared to TPC-DS, our production workload demonstrates a broader range of input cardinalities, particularly on the probe side. TPC-DS queries tend to focus on large inputs exceeding 10 million rows but underrepresent smaller and mid-sized joins, partially due to evaluations using SF10000. Real-world workloads, in contrast, demand join algorithms capable of handling diverse input sizes, where small build sides are often paired with much larger probe sides. Ensuring consistent performance across this spectrum is critical for robust, production-grade query processing.

3.2.5 Join Selectivities. As discussed in the previous section, measuring input cardinalities is prone to underestimating cardinalities due to runtime optimizations, making direct cardinality measurements unreliable representations of logical behavior. To overcome this limitation, we instead roughly categorize joins based on the relationship between their output size and the largest input size:

- **Preserving:** Output size approximately matches the input size (within 5% slack).
- **Exploding:** Output size is larger than the input size.
- **Filtering:** Output size is smaller than the input size.
- **Cartesian:** Output size is the product of the input (excluding joins with 0 or 1 rows).

This categorization offers a practical and robust framework for understanding join behavior, providing insights into the runtime characteristics of joins without relying on potentially misleading direct selectivity measurements.

Table 10: Join Categorization for Snowflake (SF) and TPC-DS (DS)

	Preserving	Exploding	Filtering	Cartesian
SF (%)	70	16	13	0.9
DS (%)	72	7	21	0.2

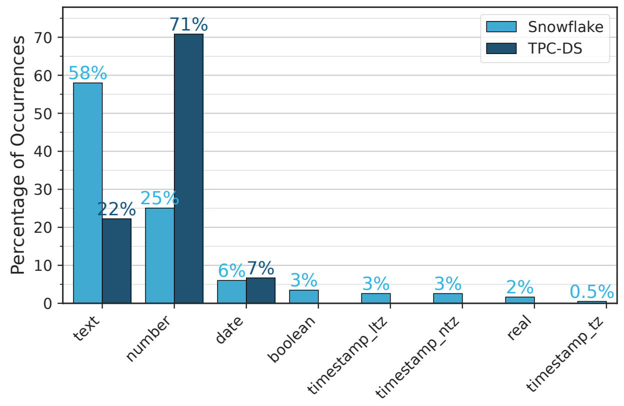

Figure 8: Filter Column Logical Type

Table 10 shows that preserving joins dominate (70%), followed by exploding (16%), filtering (13%), and Cartesian joins (0.9%). This distribution suggests that real-world workloads frequently involve joins that preserve input cardinalities. The significant share of exploding joins highlights the need for efficient handling of large intermediate results, requiring robust materialization strategies and dynamic resource management, as static optimization decisions may fail to fully account for runtime conditions.

Compared to our workload, TPC-DS includes more filtering joins and fewer exploding joins, reflecting an underrepresentation of scenarios where join outputs expand significantly. To effectively evaluate join performance, algorithms must be tested under diverse conditions that mirror the complexity of real-world workloads.

3.2.6 Discussion. Our analysis highlights two key insights about join patterns. First, limiting benchmarks to narrow key types, like 32-bit integers, covers only 20% of real-world cases. To better reflect modern analytical workloads, benchmarks should include diverse data types – textual keys, various integer sizes (8, 16, 64, 128 bits), and floating-point values. Second, 70% of joins preserve or increase input cardinalities, underscoring the significant impact of join materialization costs. This highlights the need for evaluations that consider not only filtering but also the preservation and expansion of result sizes, ensuring that algorithms and system optimizations scale efficiently.

3.3 WHERE

Our analysis of the *WHERE* clause examines the filter predicates in our workload, focusing on four key aspects: the accessed column types, the applied functions, as well as their observed complexity and selectivity.

3.3.1 Filtered Column Types. Figure 8 shows that text columns dominate filtering in our workload (58%), with numeric filters accounting for only 25%. In contrast, TPC-DS filters are primarily numeric (71%) and less text-focused (22%). Date columns appear infrequently in both cases (6–7%), while our workload includes a small proportion of timestamp, boolean, and real filters.

The prevalence of text filtering in production workloads suggests a reliance on flexible, domain-specific representations, such as storing date or boolean values as strings. While this approach improves adaptability, it introduces challenges for optimization, as existing indexing and filtering techniques often assume numeric or date-oriented predicates.

Table 11: Distribution of Top Filter Expressions for Snowflake (SF) and TPC-DS (DS)

Expr	SF (%)	DS (%)	Expr	SF (%)	DS (%)
equals	21.7	17.7	not	3.8	0.0
isnotnull	16.7	28.0	greatereq	3.2	10.1
and	9.3	15.5	contains	3.1	0.0
ifthenelse	6.3	0.1	isnull	2.6	0.4
notequal	4.1	1.2	or	2.5	3.7
in	3.9	10.0	lessthan	1.5	0.5

3.3.2 Common Predicate Functions. Table 11 highlights the distribution of top filter expressions in our workload. Equality checks (including IN lists) are the most common predicates (25.6%), followed by *IS NOT NULL* (16.7%) and logical operators (*AND*, *OR*, *NOT*) at 15%. Interestingly, text-oriented predicates like *CONTAINS* appear in 3.1% of queries, whereas TPC-DS predicates focus more on numeric and equality-based filters.

This diversity in predicate types, including text-specific filters and non-numeric predicates, illustrates the complexity of production environments. Queries like `WHERE product_name = 'X'` rely on categorical matching, requiring efficient handling of text-based predicates. Future optimizations could explore hardware-accelerated techniques (e.g., SIMD) or algorithms tailored for text filtering to address the unique challenges posed by real-world workloads.

3.3.3 Nesting Depth of Filters. Figure 9 shows that while 82% of filters in our workload are shallow ($\leq$ 5), 15% exhibit moderate complexity (6–10 levels), and a notable fraction extends beyond 10 levels. By comparison, TPC-DS filters are predominantly shallow, with 84% at depths of just 1–2 and none exceeding 10.

The presence of deeply nested filters in real-world workloads highlights opportunities for advanced optimization. Strategies like reordering selective predicates to minimize intermediate results or improve runtime performance become critical. Structurally complex filters emphasize the need for robust query planners capable of handling intricate filtering conditions, ensuring efficient performance in modern analytical workloads.

3.3.4 Selectivity Distribution. We derive filter selectivities by comparing the input cardinality of table scans to the output cardinality of the final filtering operator in a scan pipeline. Queries with incomplete statistical summaries or those terminated early due to limit conditions are excluded to ensure accuracy. Due to limitations in our query telemetry, our analysis of selectivities can sometimes include join-derived filters, such as bloom filters.

Table 12 summarizes the selectivity distribution of filters in our workload. Approximately 13% of filters eliminate all rows (selectivity $\approx$ 0), while 19% let all rows pass (selectivity $\approx$ 1). High-selectivity filters (0 < selectivity $\leq$ 0.2) dominate, accounting for 46%, with the remaining selectivities distributed across other ranges. In comparison, TPC-DS queries exhibit a narrower range, with 86% of filters removing at least 80% of rows.

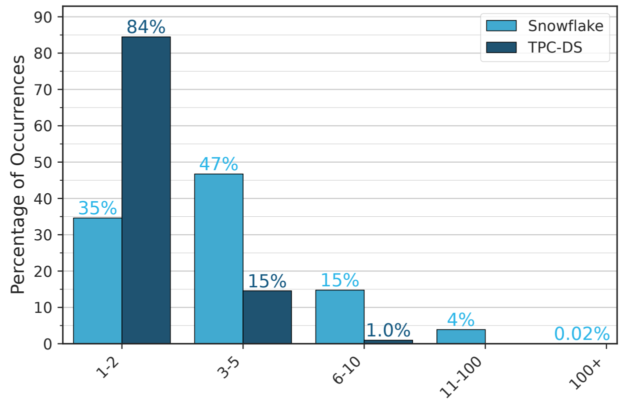

Figure 9: Maximum Nesting Depth of Filter Predicates per Query

Table 12: Selectivity Distribution of Filters for Snowflake (SF) and TPC-DS (DS)

Bucket	SF (%)	DS (%)	Bucket	SF (%)	DS (%)
0	13	3	(0.50, 0.80]	5.9	3.4
(0, 0.05]	38	60	(0.80, 0.90]	2.8	1
(0.05, 0.10]	4	7	(0.90, 0.95]	1	0.4
(0.10, 0.20]	4	16	(0.95, 1)	6	4
(0.20, 0.50]	5.6	7.2	1	19	1

The broader selectivity patterns in real-world workloads highlight the need for adaptive query optimization strategies. High-selectivity filters benefit from techniques such as indexing, clustering, and pruning to reduce unnecessary data access. In contrast, mixed-selectivity filters require dynamic predicate reordering to minimize intermediate results, while moderate-selectivity filters (selectivity ≈ 0.5) introduce challenges like branching mispredictions. These findings underline the importance of benchmarks to capture the broader variability of selectivity patterns seen in production environments, ensuring that evaluation conditions reflect the complexities of real-world workloads.

3.3.5 Discussion. Our findings highlight that real-world analytical workloads pose unique challenges for filter processing. The extensive use of text columns, prevalence of complex predicates, and frequent occurrence of deeply nested filtering conditions demand solutions that go beyond traditional numeric and shallow-filter assumptions. Recognizing the prominence of text-based filters and diverse selectivity patterns can guide system designers toward more adaptable optimization methods. Leveraging compression techniques for textual data, employing dynamic predicate evaluation to reduce intermediate results, and adjusting planning algorithms for deeper nesting levels can enhance system robustness and better address the intricacies of production workloads.

3.4 GROUP BY

For the *GROUP BY* clause, we focus on two key aspects: the distribution of the number of grouping keys and the unique key cardinalities of group-by operators. This complements our previous discussion of the most common aggregation functions in Section 3.1.2.

Table 13: Aggregation Group By Keys

	0	1-2	3-5	6-10	10+
Snowflake (%)	7	53	26	7	7
TPC-DS (%)	8	56	28	7	0

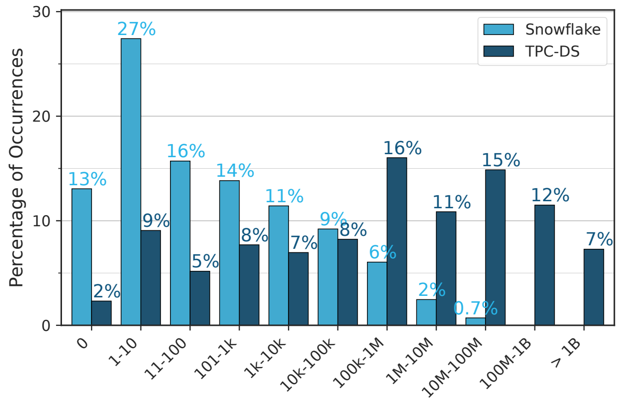

Figure 10: Number of Distinct GroupBy Keys

3.4.1 Distribution of Number of Grouping Keys. Table 13 reveals that only 7% of aggregating queries compute scalar aggregates. The majority (86%) computes grouped aggregates involving 1–10 grouping keys, with 7% exceeding 10 keys, and another 7% even surpassing 100 keys. Aside from its lack of queries with a large number of grouping keys, TPC-DS captures the distribution of smaller groupings reasonably well. This highlights the complexity of real-world queries, where compact key representations, such as single integers, may not suffice. Aggregation algorithms must efficiently handle scenarios with larger numbers of grouping keys.

3.4.2 Grouping Key Cardinalities. Figure 10 illustrates the distribution of key cardinalities for grouping keys. Our production workload predominantly produces small to medium numbers of groups, with the vast majority (70%) of queries generating fewer than 1,000 groups, 40% fewer than 10 groups, and 13% not producing a single group at all. Only 27% of queries handle medium key cardinalities (1K–1M groups), and group sizes exceeding 100M are rare.

In contrast, TPC-DS tends to represent higher key cardinalities, with 45% of queries producing over 1M groups, while smaller cardinalities of ten or fewer groups are covered by only 11% of queries. This difference highlights how TPC-DS skews toward large cardinality scenarios, underrepresenting the more common small-to-medium cardinalities seen in real-world workloads.

These findings underscore the importance of designing aggregation strategies that efficiently handle both moderate and extremely large numbers of distinct groups. Effective systems must balance performance across the full spectrum of cardinalities while focusing on the specific challenges posed by small-to-medium groups, which dominate production workloads.

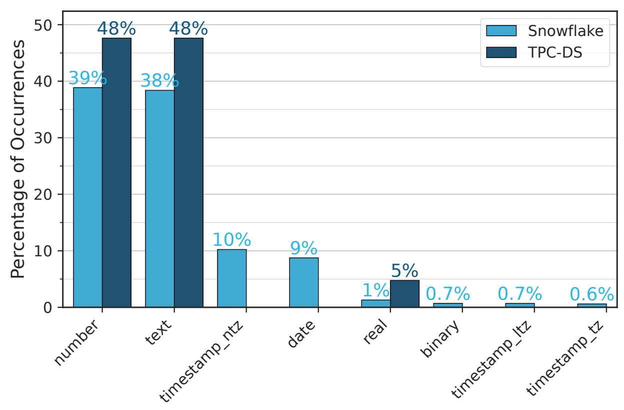

Figure 11: OrderBy Column Logical Type (Occurrences)

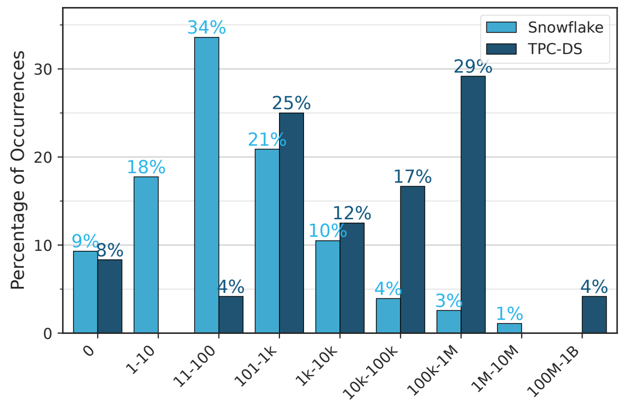

Figure 12: Distribution of Number of Sorted Rows

3.5 ORDER BY

For the *ORDER BY* clause, we analyze the logical column types used as sort keys and the distribution of sorted rows. Figure 11 shows that sorting spans a diverse range of data types, with numeric (39%) and text (38%) columns leading, while timestamp columns account for 11.3%. While our workload is largely aligned with TPC-DS, it includes sorting on timestamp columns, which TPC-DS lacks. Figure 12 reveals that 82% of sort operations handle fewer than 1,000 rows, reflecting a focus on small-scale sorting. However, there is a long tail of queries sorting millions of rows, underscoring the need for scalable sorting algorithms capable of efficiently processing both small and large datasets.

3.6 LIMIT

The *LIMIT* clause often serves as a mechanism to control result sizes, but our analysis shows that real-world queries can request far larger subsets than commonly assumed. Although LIMIT is frequently perceived as a way to fetch only a few rows, the actual usage patterns reflect more complex scenarios and practical considerations.

3.6.1 *Limit Without Order By.*

Table 14 reports the distribution of limits applied without an *ORDER BY* clause. While 1–10 rows comprise 6% of these queries, a striking 71% request between 1M and 10M rows, and 5% even exceed 1 billion rows. This result may seem surprising, as limits are generally expected to be small.

Further investigation revealed that some business intelligence (BI) tools set large default limits to ensure that desktop data visualization applications receive a sufficient amount of data. For instance, a limit value of 1,000,001 rows appears frequently, corresponding to a default configuration in a popular BI tool. Such queries can be considered “BI Tool Extracts“, where the aim is to cap retrieval sizes to prevent overwhelming the client environment. Compared to the narrower range observed in TPC-DS benchmarks (exclusively 11-100), real-world workloads exhibit a skew toward larger limits, particularly in the 1M–10M row bucket with a long tail of queries requesting over 1 billion rows.

Table 14: Limit Distribution Without ORDER BY

	0	1-10	11-100	101-1K
Snowflake (%)	1	6	3	10
TPC-DS (%)	0	0	100	0
	100K-1M	**1M-10M**	**10M-100M**	**>1B**
Snowflake (%)	0.7	71	2	5
TPC-DS (%)	0	0	0	0

Table 15: Limit Distribution With ORDER BY

	0	1-10	11-100	101-1K
Snowflake (%)	60	1	4	26
TPC-DS (%)	26	0	74	0
	1K-10K	**10K-100K**	**100K+**	
Snowflake (%)	3	5	0.3	
TPC-DS (%)	0	0	0	

3.6.2 *Limit With Order By (TopK Queries).*

Table 15 shows that 60% of queries combining LIMIT and ORDER BY specify a limit of $k = 0$, effectively turning these queries into metadata retrieval requests. In addition, 31% handle between 1 and 1,000 rows, while a long tail of 9% involves more than 1,000 rows. This indicates that top-k queries, often considered lightweight operations, must also adapt to a range of scenarios from returning no data at all to delivering extensive result sets. Such variability requires robust top-k execution strategies capable of quickly returning small amounts of metadata or scaling to process large ordered subsets without incurring substantial overhead. This also gives optimization opportunities for systems to recognize and efficiently handle hidden metadata queries.

3.7 UNION ALL

In this subsection, we examine the usage of *UNION ALL* operators and their unique challenges in real-world analytical queries.

3.7.1 *Distribution of Union All Operators.*

Figure 13 shows that among queries containing at least one *UNION ALL* operator, 97% include 1–10 *UNION ALL* operators, 2% include 11–20 operators, and 0.7% exceed 20 operators. While most queries remain relatively simple, there is a clear tail of more complex queries that chain

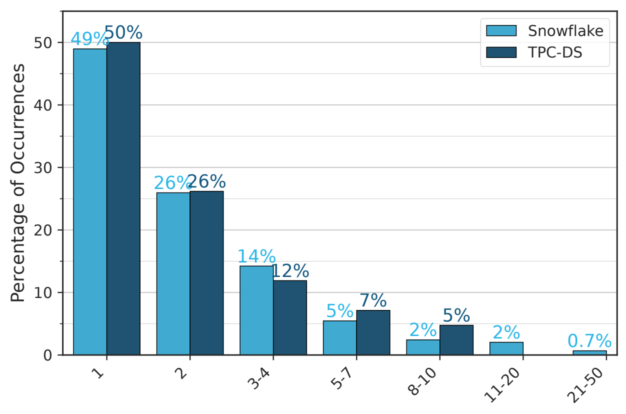

Figure 13: Number of Union Alls

together a large number of *UNION ALL* operators. TPC-DS queries follow a similar distribution but do not surpass 10 *UNION ALL* operators. Real-world workloads, however, reveal a need for handling scenarios that extend well beyond such limits, emphasizing the importance of ensuring efficient plan execution even for queries with extensive query blocks. This complexity highlights opportunities to evaluate advanced optimization strategies, including inter-operator parallelism, where multiple independent sub-plans can be processed simultaneously to improve performance.

3.7.2 Distribution of Number of Input Links. Table 16 presents the distribution of input links per *UNION ALL* operator. In our workload, slightly less than 90% of *UNION ALL* operators have 1–10 input links, and 10% have 11–100 input links, leaving a small but critical fraction of operators with more than 100 input links. This one-percent tail of highly complex operators indicates that systems must efficiently handle cases where numerous input streams merge into a single result set.

Table 16: Distribution of Union All Input Links

	1-2	**3-5**	**6-10**	**11-100**	**101-1000**	**1000+**
Snowflake (%)	41	31	18	10	0.4	0.001
TPC-DS (%)	33	43	12	12	0	0

Such complexity in *UNION ALL* usage suggests that real-world analytical systems benefit from strategies enabling concurrent processing of independent inputs. Efficient parallelization and careful resource management ensure that even large-scale *UNION ALL* operations remain performant.

4 DISCUSSION AND OUTLOOK

In this section, we reflect on our analysis of real-world queries and explore how TPC-DS, while suitable for standard BI scenarios, diverges from the complexities observed in Snowflake's workloads. These include metadata queries, deeply nested expressions, and complex joins, along with uncommon but critical query patterns that pose significant challenges for query processing.

4.1 Extending TPC-DS for Modern Analytical Workloads

Our analysis of analytical queries run by BI tools shows that TPC-DS effectively captures the core characteristics of analytical workloads, providing a strong foundation for evaluating database performance in standard BI scenarios. Its well-structured query set reflects the thoughtful design of the committee, covering a broad range of SQL features, data transformations, and query complexities typical of analytical use cases. However, as its primary focus is on standard scenarios, some of the complexities observed in real-world production systems, such as those discussed in our workload, naturally fall outside its scope. Identifying these gaps presents opportunities to complement TPC-DS by extending benchmarks to cover additional dimensions of modern workloads.

4.1.1 Metadata Queries. Our analysis reveals that metadata queries, such as those accessing schema definitions or table statistics, are a critical component of real-world workloads but are absent in TPC-DS. These queries play a vital role in query planning, caching, and metadata indexing. Incorporating metadata-heavy queries into benchmarks would enable a more comprehensive evaluation of database systems, particularly for cloud-native platforms where such queries are frequent.

4.1.2 Expressions. Production workloads frequently feature deeply nested expressions, especially in SELECT and WHERE clauses, with some exceeding depths of 100. While TPC-DS is designed to evaluate typical workloads, it does not account for such deeply nested expressions, limiting its ability to test query optimizers under these conditions. Expanding benchmarks to include these patterns would better reflect the complexity of modern workloads.

4.1.3 Joins. Our analysis also shows that production workloads have a long tail of join complexity, with some queries featuring over 500 joins and 3.3% involving 30 or more joins. TPC-DS underrepresents such scenarios and focuses primarily on inner joins, whereas real-world workloads highlight the importance of outer joins and text-based join keys. Extending the benchmark to include diverse join types, exploding joins, and text-key joins would enhance its coverage of real-world challenges.

4.1.4 Aggregations. Aggregations, such as *anyvalue* and those involving text-based columns, are common in Snowflake workloads but are less emphasized in TPC-DS. Adding queries that include these patterns would improve the benchmark's alignment with production workloads.

4.1.5 Limit and Top-K. Real-world workloads frequently involve limit queries with substantial result sizes, where 78% of limits return over 1M rows, and 34.3% of top-k queries involve $k > 100$. TPC-DS does not account for these scenarios. Including large-result limits in benchmarks would better test database system performance under such high-output conditions.

4.1.6 Union All. Our findings show that production workloads often exceed the limits of UNION ALL operators in TPC-DS, with 2.7% of queries containing more than 10 operators and 0.4% exceeding 100 input links. Including such patterns in benchmarks would

Table 17: Components of Top 1% Outlier Queries

	Expressions	Aggregates	Filters	Projection	Joins	Union All	Scan
mean	1413	337	305	357	76	23	75
10%	524	67	89	187	35	9	34
50%	908	232	178	269	58	17	58
90%	2584	740	511	584	131	47	139
99%	7953	2100	2957	1682	352	127	348

provide a more accurate representation of real-world inter-operator parallelism and input variability.

4.2 Unusual but Not Uncommon Query Patterns

In this section, we highlight query patterns that are both distinctive and occur frequently enough to warrant closer attention. While a single query featuring thousands of expressions or several hundred UNION ALL operators represents an edge case, our focus is on identifying groups of queries that consistently stand out across multiple dimensions. To achieve this, we employ Isolation Forests [18], a tree-based anomaly detection technique that isolates outliers by recursively partitioning the dataset and identifying queries that are easier to separate from the rest. We then analyze the top 1% of outliers to understand the patterns they exhibit. Given the large scale at which cloud providers operate, this 1% still represents a substantial number of queries. Table 17 summarizes the statistics of these unusual yet common workloads.

As shown in the table, even at lower percentiles, the number of expressions, predicates, and joins is already substantial, indicating that a notable fraction of queries are inherently complex and resource-intensive. At the higher end of the spectrum, the 99th percentile reveals extremely complex queries that could significantly influence system performance. These findings reinforce the need for robust query processing and resource management strategies capable of efficiently handling diverse and challenging workloads.

5 RELATED WORK

In order to position our findings in the broader research context, we now discuss two areas of related work: first, recent studies analyzing production database workloads to better understand their real-world properties; second, studies focused on benchmarking, which assess whether existing database benchmarks accurately reflect the complexity and patterns observed in practice.

5.1 Workload Analysis

Several recent studies have investigated real-world SQL workloads to better understand their characteristics. Vuppalapati et al. [31] published Snowset, a dataset containing statistics for 70 million queries collected over a 14-day period at Snowflake. They identified several research opportunities based on query characteristics, workload patterns, and system performance metrics. In contrast, our study focuses on query characteristics such as operator distribution, complexity, and usage patterns, emphasizing core relational operators (e.g., table scans, joins, aggregations). We also compare our findings with TPC-DS, highlighting gaps where it does not fully capture real-world analytical workloads.

Similarly, van Renen et al. [26] conducted an extensive workload analysis, publishing Redset, a dataset containing anonymized query logs. Their work primarily addresses data characteristics, including table properties, data skew, row-count distributions, common-value frequencies, and null-value fractions, alongside workload-level features such as elasticity and repeating queries. Our study complements their findings by investigating how analytical SQL queries are structured, composed, and executed in practice, emphasizing query-level characteristics and real-world query patterns. From a sample of 667M BI queries we find that 31 % of all statements are catalog metadata queries, reads outnumber writes 25 : 1, and 78 % of `LIMIT` clauses return at least one million rows – query-level patterns absent from the Redset analysis and critical for tuning.

Durner et al. introduced TracEx [12], a workload exploration tool designed for detailed analysis and comparison of query traces from multiple database systems. Using TracEx, they compared the Snowset workload with TPC-DS and concluded that TPC-DS lacked comprehensive coverage, highlighting the need for additional, more representative queries – a conclusion consistent with our findings.

Comparably, Boissier et al. [5] analyzed SQL workload traces from a live SAP ERP system to extract fine-grained access patterns and study data relevance. Their work highlights differences between real-world workloads and the synthetic OLTP benchmarks TPC-C and TPC-E, providing a similar conclusion to ours.

Aleyasen et al. [1, 2] employ their profiling tool qInsight to examine 40 production workloads drawn from large enterprises covering a broad range of workload metrics. Their study quantifies data volume and variety, inspects logical design (e.g., primary-key and foreign-key usage), reviews physical layout choices (partitioning and compression), and measures query mix and throughput. The authors surface several unexpected patterns, such as minimal foreign-key enforcement and the frequent use of non-traditional data types, that challenge conventional data-warehousing best practices. While the work also targets query properties, it does not compare them with synthetic benchmarks, but rather argues for workload-aware database replatforming.

Further highlighting gaps in current benchmarks, Vogelsgesang et al. [29] analyzed query complexity and expression evaluation bottlenecks on a set of more than 60k real-world BI repositories of Tableau customers. Their findings demonstrate that simpler query structures may hide significant complexities in expression evaluation, while some queries possess operator complexities that are not represented in standard benchmarks. While their work explores data-type distributions and operator frequency, our analysis emphasizes comprehensive coverage of query structure, focusing explicitly on language properties and real-world distribution of relational operators and constructs.

Jain et al. conducted multiple studies on generalized SQL workload analysis. In [16], they advocated for managing database workloads independently of specific database implementations to optimize efficiency in cloud and hybrid database systems. In another study [15], Jain et al. analyzed multi-year workloads from a database-as-a-service platform, noting the prevalence of ad-hoc and exploratory queries written mostly by non-experts.

Tsai et al. [25] examined challenges associated with benchmarking SaaS applications, specifically addressing scalability testing and proposing novel metrics for assessing these systems. Additionally, automated workload analysis remains essential in self-tuning database systems, including workload-driven partitioning and replication strategies (Curino et al. [7]), robust workload forecasting (Ma et al. [19]), and predictive analysis of SQL query properties (Zolaktaf et al. [34]).

Stokely et al. [22] published and analyzed a three-year dataset of hourly VM-demand across multiple Snowflake workloads. Using this data, they characterize demand patterns and derive effective optimizations to balance compute capacity commitments and costs for cloud customers.

5.2 Benchmarks

Beyond workload characterization, another closely related research direction investigates the limitations of traditional benchmarks – particularly their ability to accurately capture cloud-specific workload characteristics – and proposes new benchmarks designed specifically to reflect realistic database usage and performance. Binnig et al. [4] emphasize that benchmarks for cloud databases should account for critical properties such as scalability, elasticity, fault tolerance, and pay-as-you-go pricing. They also argue that cloud benchmarks must ensure results remain comparable across providers and service configurations.

Poggi et al. [21] introduced the Elasticity Test, an extension to existing TPC benchmarks designed explicitly for cloud environments. Their benchmark emphasizes elasticity and evaluates database responsiveness under realistic big-data scenarios, including metrics based on service-level agreements.

Van Renen et al. [27] compared traditional benchmarks (TPC-DS, TPC-H) with real-world query workloads on cloud databases based on the Snowset dataset. They identified several shortcomings and proposed the Cloud Analytics Benchmark (CAB), enabling users to evaluate system performance and cost-effectiveness more accurately in cloud scenarios.

Zhang et al. [32] proposed HyBench, a benchmark designed specifically for hybrid transactional and analytical processing (HTAP) workloads. It targets hybrid transaction/analytical scenarios, whereas our work exclusively addresses analytical queries and their detailed query characteristics.

Gruenheid et al. introduced DIAMetrics [10], a comprehensive benchmarking suite developed at Google to evaluate database systems at scale. DIAMetrics offers broad compatibility and versatility across multiple database systems and industrial use-cases. Tang et al. propose CSDBen [33], a benchmark that focuses on cloud-native storage services. They discuss where TPC-C and YCSB benchmarks fall short and enable users to generate I/O traces that accurately resemble real-world I/O traces.

Collectively, these benchmarks underscore the ongoing effort to develop more representative testing and benchmarking methodologies. Our work supports this effort by providing new insights into the shape and structure of real-world analytical workloads, which the research community can use to enhance existing benchmarks or revisit cost models for optimal instance configuration [17].

6 CONCLUSION

In this paper, we analyzed a real-world sample of queries executed by popular BI tools on the Snowflake platform, uncovering key insights into modern analytical workloads and identifying critical gaps in existing benchmarks such as TPC-DS.

Workloads. Our analysis revealed that while most queries are simple, a significant tail of complex queries with thousands of operators and diverse configurations poses unique challenges for database systems. Even rare query patterns (e.g., 0.1% of the workload) are common enough to influence system design, emphasizing the need for robust handling of complex workloads at scale.

Query Patterns. By examining SQL components such as SELECT, WHERE, GROUP BY, and LIMIT clauses, we identified prevalent operator and expression properties, including frequent conditional logic and deeply nested expressions. These findings enable researchers to evaluate new database algorithms and optimization strategies based on realistic workload characteristics. For example, the dominance of text processing and temporal functions underscores the need for optimizing predicate evaluation and text-oriented processing algorithms.

Insights on TPC-DS. While TPC-DS captures a broad range of query complexities and operator configurations, our findings highlight the following gaps: 1) Metadata queries essential for query planning and caching, 2) High query complexities such as deeply nested expressions and large join graphs, and 3) Diverse selectivity patterns and operator types, particularly conditional logic and text processing.

Benchmark Evolution. The observed differences between Snowflake and TPC-DS workloads emphasize the need to evolve benchmarks to better reflect the diversity and complexity of modern analytical workloads. Incorporating metadata queries, complex join patterns, and mixed selectivity filters would enable a more comprehensive evaluation framework for database systems.

By addressing these gaps, this work contributes to the development of robust, scalable, and adaptable database systems capable of meeting the demands of real-world analytical workloads.

ACKNOWLEDGMENTS

We thank the Snowflake team for their support and collaboration throughout this project. In particular, we are grateful to Hossein Ahmadi, Allison Lee, Rudi Leibbrandt, Abdul Q Munir, Berni Schiefer, Martin Schoenert and Changbin Song for their invaluable support. Some phrasing in the paper was refined with OpenAI ChatGPT (GPT o3-pro mode, June 2025). All technical content was authored and verified by the human authors.

REFERENCES

[1] Amirhossein Aleyasen. 2022. *Overcoming barriers in data warehouse replatforming.* Ph.D. Dissertation. University of Illinois at Urbana-Champaign.

[2] Amirhossein Aleyasen, Mark Morcos, Lyublena Antova, Marc Sugiyama, Dmitri Korablev, Jozsef Patvarczki, Rima Mutreja, Michael Duller, Florian M. Waas, and Marianne Winslett. 2022. Intelligent Automated Workload Analysis for Database Replatforming. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 2273–2285. https://doi.org/10.1145/3514221.3526050

[3] Nikos Armenatzoglou, Sanuj Basu, Naga Bhanoori, Mengchu Cai, Naresh Chainani, Kiran Chinta, Venkatraman Govindaraju, Todd J. Green, Monish Gupta, Sebastian Hillig, Eric Hotinger, Yan Leshinksy, Jintian Liang, Michael McCreedy, Fabian Nagel, Ippokratis Pandis, Panos Parchas, Rahul Pathak, Orestis Polychroniou, Foyzur Rahman, Gaurav Saxena, Gokul Soundararajan, Sriram Subramanian, and Doug Terry. 2022. Amazon Redshift Re-invented. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 2205–2217. https://doi.org/10.1145/3514221.3526045

[4] Carsten Binnig, Donald Kossmann, Tim Kraska, and Simon Loesing. 2009. How is the weather tomorrow? towards a benchmark for the cloud. In *Proceedings of the Second International Workshop on Testing Database Systems* (Providence, Rhode Island) *(DBTest '09)*. Association for Computing Machinery, New York, NY, USA, Article 9, 6 pages. https://doi.org/10.1145/1594156.1594168

[5] Martin Boissier, Carsten Alexander Meyer, Timo Djürken, Jan Lindemann, Kathrin Mao, Pascal Reinhardt, Tim Specht, Tim Zimmermann, and Matthias Uflacker. 2016. Analyzing Data Relevance and Access Patterns of Live Production Database Systems. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (Indianapolis, Indiana, USA) *(CIKM '16)*. Association for Computing Machinery, New York, NY, USA, 2473–2475. https://doi.org/10.1145/2983323.2983336

[6] ClickHouse. 2024. ClickBench: a Benchmark For Analytical Databases. github.com/ClickHouse/ClickBench. Accessed: 2025-07-10.

[7] Carlo Curino, Evan Jones, Yang Zhang, and Sam Madden. 2010. Schism: a workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.* 3, 1–2 (Sept. 2010), 48–57. https://doi.org/10.14778/1920841.1920853

[8] Bjoern Daase and Rudi Leibbrandt. 2024. Query Acceleration: Smarter Join Decisions. https://www.snowflake.com/engineering-blog/query-acceleration-smarter-join-decisions/ Accessed: 2024-12-02.

[9] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 215–226. https://doi.org/10.1145/2882903.2903741

[10] Shaleen Deep, Anja Gruenheid, Kruthi Nagaraj, Hiro Naito, Jeff Naughton, and Stratis Viglas. 2022. DIAMETRICS: benchmarking query engines at scale. *Commun. ACM* 65, 12 (Nov. 2022), 105–112. https://doi.org/10.1145/3567464

[11] Markus Dreseler, Martin Boissier, Tilmann Rabl, and Matthias Uflacker. 2020. Quantifying TPC-H choke points and their optimizations. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1206–1220.

[12] Dominik Durner, Lennart Espe, Jana Giceva, and Anja Gruenheid. 2024. TracEx: Understanding and Analyzing Database Traces. In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*. www.cidrdb.org.

[13] Anurag Gupta, Deepak Agarwal, Derek Tan, Jakub Kulesza, Rahul Pathak, Stefano Stefani, and Vidhya Srinivasan. 2015. Amazon redshift and the case for simpler data warehouses. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1917–1923.

[14] Surabhi Gupta and Karthik Ramachandra. 2021. Procedural extensions of SQL: understanding their usage in the wild. *Proc. VLDB Endow.* 14, 8 (April 2021), 1378–1391.

[15] Shrainik Jain, Dominik Moritz, Daniel Halperin, Bill Howe, and Ed Lazowska. 2016. SQLShare: Results from a Multi-Year SQL-as-a-Service Experiment. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 281–293. https://doi.org/10.1145/2882903.2882957

[16] Shrainik Jain, Jiaqi Yan, Thierry Cruanes, and Bill Howe. 2019. Database-Agnostic Workload Management. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p110-jain-cidr19.pdf

[17] Viktor Leis and Maximilian Kuschewski. 2021. Towards cost-optimal query processing in the cloud. *Proc. VLDB Endow.* 14, 9 (May 2021), 1606–1612. https://doi.org/10.14778/3461535.3461549

[18] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*. 413–422. https://doi.org/10.1109/ICDM.2008.17

[19] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. 2018. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 631–645. https://doi.org/10.1145/3183713.3196908

[20] Sergey Melnik, Andrey Gubarev, Jingjing Long, Geoffrey Romer, Sriram Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. Dremel: Interactive Analysis of Web-Scale Datasets. In *Proceedings of the 36th International Conference on Very Large Data Bases (VLDB)*. VLDB Endowment, 330–339.

[21] Nicolas Poggi, Víctor Cuevas-Vicenttín, Josep Berral, Thomas Fenech, Gonzalo Gómez Sánchez, Davide Brini, Alejandro Montero, David Carrera, Umar Farooq Minhas, José Blakeley, Donald Kossmann, Raghu Ramakrishnan, and Clemens Szyperski. 2020. *Benchmarking Elastic Cloud Big Data Services Under SLA Constraints*. 1–18. https://doi.org/10.1007/978-3-030-55024-0_1

[22] Murray Stokely, Neel Nadgir, Jack Peele, and Orestis Kostakis. 2025. Shaved Ice: Optimal Compute Resource Commitments for Dynamic Multi-Cloud Workloads. In *Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering, ICPE 2025, Toronto, ON, Canada, May 5-9, 2025*, Marin Litoiu, Evgenia Smirni, Alessandro Vittorio Papadopoulos, and Katinka Wolter (Eds.). ACM, 124–135. https://doi.org/10.1145/3676151.3719353

[23] Transaction Processing Performance Council (TPC). 2021. TPC BENCHMARK^{TM} DS Standard Specification Version 3.2.0. www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DS_v3.2.0.pdf. Accessed: 2025-07-10.

[24] Transaction Processing Performance Council (TPC). 2022. TPC BENCHMARK^{TM} H Standard Specification Version 3.0.1. www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf. Accessed: 2025-07-10.

[25] Wei-Tek Tsai, Yu Huang, and Qihong Shao. 2011. Testing the scalability of SaaS applications. In *2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*. 1–4. https://doi.org/10.1109/SOCA.2011.6166245

[26] Alexander van Renen, Dominik Horn, Pascal Pfeil, Kapil Vaidya, Wenjian Dong, Murali Narayanaswamy, Zhengchun Liu, Gaurav Saxena, Andreas Kipf, and Tim Kraska. 2024. Why TPC is Not Enough: An Analysis of the Amazon Redshift Fleet. *Proc. VLDB Endow.* 17, 11 (Aug. 2024), 3694–3706.

[27] Alexander van Renen and Viktor Leis. 2023. Cloud Analytics Benchmark. *Proc. VLDB Endow.* 16, 6 (Feb. 2023), 1413–1425. https://doi.org/10.14778/3583140.3583156

[28] Aditya Vashistha and Shrainik Jain. 2015. Measuring Query Complexity in SQL-Share Workload. https://uwescience.github.io/sqlshare/pdfs/Jain-Vashistha.pdf Accessed: 2025-07-10.

[29] Adrian Vogelsgesang, Michael Haubenschild, Jan Finis, Alfons Kemper, Viktor Leis, Tobias Muehlbauer, Thomas Neumann, and Manuel Then. 2018. Get Real: How Benchmarks Fail to Represent the Real World. In *Proceedings of the Workshop on Testing Database Systems* (Houston, TX, USA) *(DBTest '18)*. Association for Computing Machinery, New York, NY, USA, Article 1, 6 pages. https://doi.org/10.1145/3209950.3209952

[30] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building an elastic query engine on disaggregated storage. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 449–462.

[31] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building An Elastic Query Engine on Disaggregated Storage. In *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, Ranjita Bhagwan and George Porter (Eds.). USENIX Association, 449–462. https://www.usenix.org/conference/nsdi20/presentation/vuppalapati

[32] Chao Zhang, Guoliang Li, and Tao Lv. 2024. HyBench: A New Benchmark for HTAP Databases. *Proc. VLDB Endow.* 17, 5 (May 2024), 939–951. https://doi.org/10.14778/3641204.3641206

[33] Jiashu Zhang, Wen Jiang, Bo Tang, Haoxiang Ma, Lixun Cao, Zhongbin Jiang, Yuanyuan Nie, Fan Wang, Lei Zhang, and Yuming Liang. 2023. CDSBen: Benchmarking the Performance of Storage Services in Cloud-Native Database System at ByteDance. *Proc. VLDB Endow.* 16, 12 (Aug. 2023), 3584–3596. https://doi.org/10.14778/3611540.3611549

[34] Zainab Zolaktaf, Mostafa Milani, and Rachel Pottinger. 2020. Facilitating SQL Query Composition and Analysis. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 209–224. https://doi.org/10.1145/3318464.3380602