Microsoft

Modern Power BI Architecture Choices for Reporting On Azure Databricks

A benchmark comparing the performance of different Power BI storage modes on Azure Databricks data sources

Author
Liping Huang
CEO, Data Leaps

Table of Contents

1. Executive Summary

Many enterprise Power BI semantic models use Azure Databricks as a data source. When building these semantic models, Power BI developers and architects are faced with the question of which storage mode to use. Several different factors need to be taken into account when making this choice – cost, security, ease of development and ease of tuning – but out of all of them report performance is probably the most important. Reports that are slow to load are a common cause of dissatisfaction among end users. The purpose of this white paper is to help Power BI developers and architects understand which Power BI semantic model storage mode will result in the best performance for their reports.

The recommendations in this white paper are grounded in empirical evidence gathered from a series of benchmark tests. The goal of these benchmarks was to compare query performance under both single-user and multi-user load scenarios, while also accounting for platform-specific optimizations that a well-architected deployment would typically employ. This benchmark represents a point-in-time comparison as of June and July 2026. Given the pace of new features and optimizations being introduced to both Power BI and Databricks, the relative performance of both DirectQuery and Direct Lake storage modes is likely to evolve quickly. This white paper is intended to serve as a guide for customers to conduct their own performance testing using their own data, rather than as a definitive or permanent verdict on either platform. It is important to note that this benchmark measures the end-user query experience within Power BI, rather than raw database execution speed.

By isolating variables such as data volume, cache scenarios, and test queries, this study seeks to equip architects, BI leads, and decision-makers with practical, evidence-based insights into performance differences and trade-offs among the architectural patterns.

1.1 Key Findings

- **No single storage mode wins universally**, and the optimal choice depends heavily on cache state, filter scenario, data volume, and query type.
- **Direct Lake is highly competitive at smaller to medium data volumes** (SF10, SF100), frequently matching or beating DirectQuery.
- **Direct Lake is the strongest overall performer at scale in Warm and Hot Cache scenarios,** which are the scenarios most commonly encountered by end users in real-world usage. For the Warm Cache scenario, Direct Lake is only outperformed by the Composite Model at Scale Factor 1000 when no filter is applied, and at all data volumes for queries that can be resolved using the aggregation tables.
- **DirectQuery is the slower pattern in Warm Cache scenarios,** even taking into account the overhead resulting from clearing the Power BI semantic model cache discussed later in this paper. However, it becomes more competitive, and in some cases the fastest pattern, at Scale Factor 1000 (when compute was scaled up to an X-large SQL warehouse and compared against F128), particularly in Cold Cache and load test conditions.
- **Direct Lake over Mirrored Unity Catalog tables degrades significantly at higher data volumes**, becoming the slowest pattern at SF100 and SF1000, with latency at SF1000 exceeding what's acceptable for

normal Power BI interactions. It also failed to complete load testing at SF1000 due to exceeding memory limits on F128.

- **Composite Model with aggregations delivers the fastest and most consistent performance for queries that can be resolved by aggregation tables**, across all data volumes and cache scenarios, but this advantage disappears for queries that fall through to the underlying DirectQuery source.
- **Distinct Count emerges as the most challenging query type for both Direct Lake and Direct Lake over Mirrored,** especially at SF1000 when no filter is applied in cold and warm cache tests, where DirectQuery and the Composite Model outperform them.

2. Scope

This benchmark focuses on the query performance of the following patterns:

- DirectQuery on Databricks (All tables in DirectQuery mode)
- Composite Model on Databricks (Dimension Tables in Dual mode, Fact Tables in DirectQuery mode and aggregation tables in import mode)
- Direct Lake over Mirrored Unity Catalog tables (All tables are mirrored from Databricks)
- Direct Lake on OneLake over persistent Delta tables in Fabric (All tables are persistent as Delta tables in Fabric, and not accessed through shortcuts)

Out of Scope

- Any scenarios involving security features such as row-level security or column-level security
- UI/UX comparisons
- Semantic layer location/placement discussion
- Cost-performance analysis, as the actual price paid for Power BI and Databricks could vary considerably among customers.
- Direct Lake over Shortcut tables from ADLS Gen2
- Direct Lake on SQL in Fabric. Between the two Direct Lake variants, Direct Lake on OneLake is chosen for this benchmark because it is the newer storage mode that supports multiple sources and tighter integration with native OneLake security.
- Fallback to DirectQuery, as this behavior only applies to Direct Lake on SQL in Fabric
- Aggregation tables built in DirectQuery mode rather than Import mode, because building aggregations in Import mode is common practice. This decision was also made to ensure a clear and consistent comparison between Direct Lake and DirectQuery. Both patterns were tested with all original tables kept in their respective native mode (Direct Lake and DirectQuery), rather than allowing DirectQuery to benefit from aggregations while Direct Lake did not.

- Composite Models where Fact Tables are in Direct Lake mode and Dimension Tables are in Import mode. This configuration was considered out of scope because Dimension Tables are typically small in size relative to Fact Tables, meaning that switching Dimension Tables from Direct Lake to Import mode would be unlikely to yield a meaningful difference in query performance. As a result, testing this pattern would not add additional insight to this benchmark.
- Custom aggregations in Import mode, where the built-in aggregation feature is not used, but an aggregation table is instead included in the model, with DAX logic in measures to redirect queries to the aggregation table wherever possible.

3. Test Methodology Overview

3.1 Testing philosophy

To ensure a fair comparison across all tested patterns, identical dataset volumes and schemas, identical report and semantic model logic, and identical test queries were used throughout. Additionally, this benchmark was designed to be fully reproducible: the testing scripts, test setup, data generation and customization steps, and optimizations applied are all documented in detail, allowing the tests to be independently replicated and validated.

3.2 Tool used for performance testing

- Tool used for data generation
  - https://github.com/microsoft/LakeBench v1.0.1
  - The data-generation and customization script is provided in Appendix 1.
- Tool used for load testing
  - https://github.com/microsoft/fabric-toolbox/tree/main/tools/FabricLoadTestTool
  - Customizations were made to the load testing notebook; see Section 7.3 for the rationale.
  - The customized load-testing script is provided in Appendix 2.
- Tools used for monitoring
  - Power BI Performance Analyzer
  - DAX Studio
  - Semantic Link Labs
  - Fabric Capacity Metrics App
  - Databricks query history

3.3 Metrics captured

- The following metrics were captured during the single-user tests. See the DAX Studio documentation for definitions.
    - Storage Engine Duration
    - Formula Engine Duration
    - Total DirectQuery Duration
    - Timeline Total Duration
    - Storage Engine CPU
    - Storage Engine Query Count
    - Total Duration
    - Total CPU Duration
    - VertiPaq Cache Matches
- The following metrics were captured during the load tests. See Fabric Load Test Tool documentation for performance metrics details.
    - Loadtest_id
    - Query number
    - Query
    - Visual name
    - Duration
    - Start_time
    - Rows
    - Concurrent_threads
    - Thread_id

4. Dataset Description

4.1 Source dataset and data generation

The dataset used in this benchmark is a subset of the TPC-DS dataset, comprising 2 Fact Tables and 8 Dimension Tables. The dataset was generated using LakeBench (v1.0.1), an open-source data generation tool available at https://github.com/microsoft/LakeBench.

4.2 Data volume

The following data volume tiers were tested during performance testing:

- Scale Factor 10 (SF10) (approximately 26 million rows for the largest fact table after customization)
- Scale Factor 100 (SF100) (approximately 260 million rows for the largest fact table after customization)
- Scale Factor 1000 (SF1000) (approximately 2.6 billion rows for the largest fact table after customization)

4.3 Tables and columns

- Fact Tables
    - Store_sales (Largest fact table)
    - Catalog_sales (second fact table)
- Dimension Tables
    - Catalog_page
    - Customer_address
    - Customer_demographics
    - Date_dim
    - Item
    - Promotion
    - Ship_mode
    - Store

- Table row count

Table/Row Count	SF10	SF100	SF1000
store_sales*	26,206,837	262,082,396	2,620,785,279
catalog_sales*	14,257,451	142,557,716	1,425,579,810
catalog_page	12,000	20,400	30,000
customer_address	250,000	1,000,000	6,000,000
customer_demographics	1,920,800	1,920,800	1,920,800
date_dim*	2,191	2,191	2,191
item	102,000	204,000	300,000
promotion	500	1,000	1,500
ship_mode	20	20	20
store	102	402	1,002

Table 4.3.1 Row counts by scale factor

* The row counts shown in the table are after customization; the datasets cover 2021–2026.

- Columns
  - Power BI modeling best practices recommend including only the columns required to support reporting and query needs within the semantic model. To better reflect real-world scenarios, however, some additional columns beyond those used directly in the test queries were retained in the model.
  - To explore the full set of columns included in the model, see the test model in Appendix 3.
- Data types
  - All data types in the test dataset conformed to the TPC-DS v4.0.0 specification.

4.4 Relationships

- All relationships were based on the TPC-DS v4.0.0 specification. Relationships that originally used d_date_sk used d_date_sk_1 instead; see Section 4.5 for the rationale.
- To prevent ambiguous paths between tables in Power BI, the following relationships were deactivated
  - 'catalog_sales'[cs_ship_addr_sk] ∞←1 'customer_address'[ca_address_sk]
  - 'catalog_sales'[cs_ship_cdemo_sk] ∞←1 'customer_demographics'[cd_demo_sk]
  - 'promotion'[p_item_sk] ∞←1 'item'[i_item_sk]
- All relationships with Fact Tables were built with "Assume Referential Integrity" (only relationships with Fact Tables matter in the test queries)
- All queries used one-to-many relationships; the test model contains no many-to-many relationships.

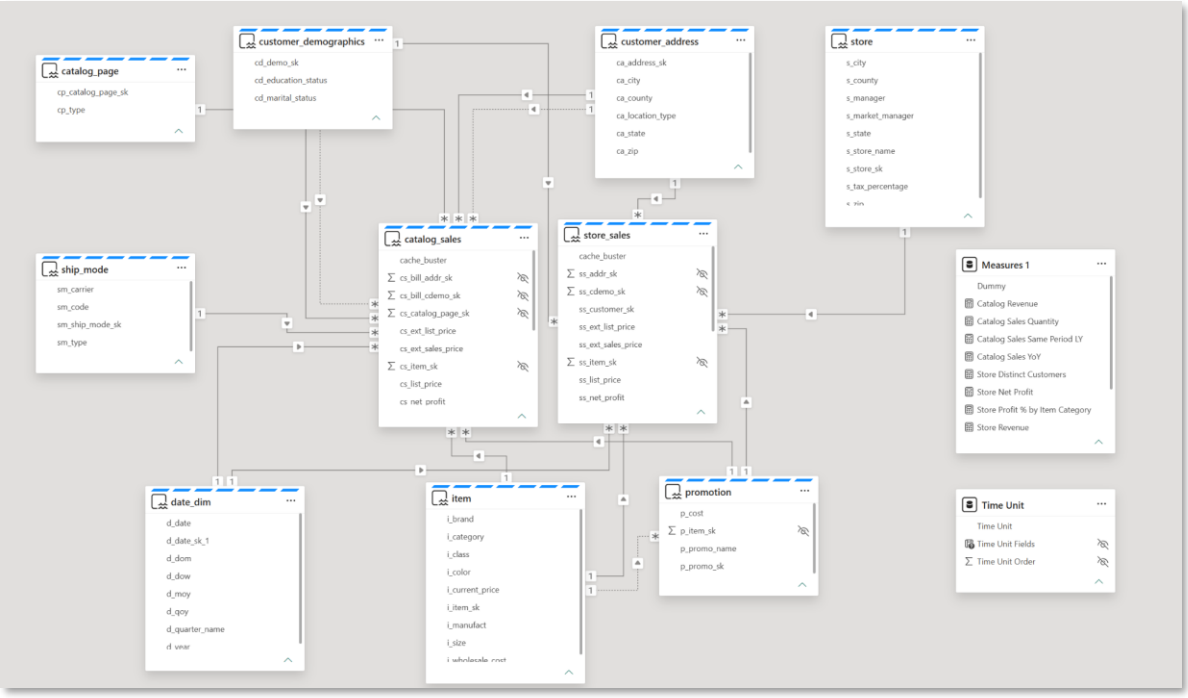


Figure 4.4.1 Test Model Diagram for the Direct Lake model

4.5 Customizations

The following customizations were made to Fact Tables:

- All nulls in both Fact Tables were removed so that "Assume Referential Integrity" could be set to "True" for relationships involving Fact Tables, allowing Power BI to generate inner join SQL queries, which are more efficient.
- A cache_buster column, initialized with a value of 1 and used to minimize cache reuse during load testing, was added to both Fact Tables. See Section 7.3 (Load Test Setup) for the rationale.

The following customizations were made to the date_dim tables:

- To support time intelligence DAX calculations such as YoY and YTD (with the performance test conducted in mid-2026), an additional column, d_date_sk_1, was added to the date_dim table. This column replaces d_date_sk as the primary key for the date_dim table.
- All rows containing dates before 2021 and after 2026 were removed, as data in both Fact Tables, based on their relationship with d_date_sk_1, is only available for this period.

All customizations are included in the data generation and customization script in Appendix 1.

4.6 Delta Lake table format details

- OPTIMIZE, VACUUM, and ANALYZE TABLE were run on all tables across all patterns.
- Fabric Direct Lake tables had V-Order enabled.
- Fabric Direct Lake Fact Tables were partitioned and Z-ordered.
- Databricks DirectQuery tables used Liquid Clustering.
- Direct Lake over Mirrored Unity Catalog tables were Z-ordered instead of liquid clustered, because Fabric did not support mirrored Unity Catalog tables containing V2 checkpoints at the time of testing.

	Store_sales	Catalog_sales
Direct Lake over Mirrored	ZORDER BY (ss_sold_date_sk, ss_addr_sk)	ZORDER BY (cs_sold_date_sk, cs_bill_addr_sk)
Direct Lake on OneLake	Partitioned by ["ss_sold_date_sk"] ZORDER BY (ss_addr_sk)	Partitioned by ["cs_sold_date_sk"] ZORDER BY (cs_bill_addr_sk)

DirectQuery on Databricks	LIQUID CLUSTER BY (ss_sold_date_sk, ss_addr_sk)	LIQUID CLUSTER BY (cs_sold_date_sk, cs_bill_addr_sk)
Composite Model on Databricks	Same as DirectQuery on Databricks	Same as DirectQuery on Databricks

Table 4.6.1 Delta Table Data Layout Details

	SF	Store_sales File Count (File Size)	Catalog_sales File Count (File Size)
Direct Lake over Mirrored	10	16 (0.90 GB)	16 (0.77 GB)
	100	110 (12.12 GB)	108 (9.87 GB)
	1000	1,237 (124.89 GB)	1,012 (100.67 GB)
Direct Lake on OneLake	10	1,823 (1.47 GB)	1,835 (1.24 GB)
	100	1,823 (14.04 GB)	1,836 (12.02 GB)
	1000	1,823 (123.00 GB)	1,836 (106.94 GB)
DirectQuery on Databricks	10	19 (1.14 GB)	16 (0.90 GB)
	100	253 (12.68 GB)	167 (10.10 GB)
	1000	2,321 (132.55 GB)	1,712 (104.18 GB)
Composite Model on Databricks	10	Same as DirectQuery on Databricks	
	100		
	1000		

Table 4.6.2 Fact Table File Counts and Sizes

5. Optimization Techniques Applied

5.1 Databricks-side optimizations

- A Serverless SQL Warehouse was used.
- Disk caching was used for warm cache testing, and result caching was used for hot cache testing.
- OPTIMIZE and VACUUM were run on all tables.
- ANALYZE TABLE COMPUTE STATISTICS FOR ALL COLUMNS was run on all tables.
- Primary Key constraints with the RELY clause were added to all Dimension Tables.
- Liquid Clustering was enabled on both Fact Tables for DirectQuery scenarios.
- Predictive Optimization was enabled on all tables.
- For load testing, the SQL Warehouse scaling setting was configured with a minimum and maximum of 10 clusters to ensure maximum parallelism.

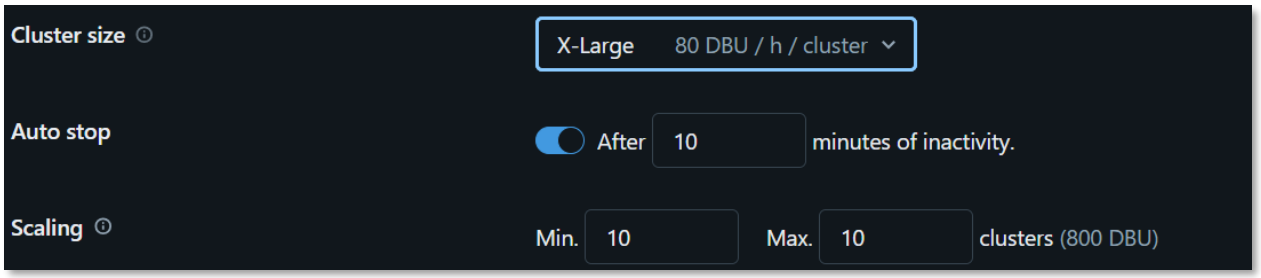


Figure 5.1.1 Databricks SQL Warehouse Scaling during the Load Test

Delta-level optimization details are provided in the data generation and customization script in Appendix 1.

5.2 Fabric Direct Lake optimizations

- V-Order was enabled.
- ZSTD compression was applied to Parquet files.
- Optimize Write for Delta Tables was enabled.
- Target file size for Optimize Write for Delta Tables was set to 1 GB.
- OPTIMIZE and VACUUM were run on all tables.
- ANALYZE TABLE COMPUTE STATISTICS FOR ALL COLUMNS was run on all tables.
- Fact Tables were partitioned and Z-ordered.

Target file sizing for optimize write set to 1 GB was taken from the readHeavyForPBI resource profile from the Fabric documentation. Due to the size of the dataset and the partitioning strategy of this test, the test did not achieve a 1 GB file size. This test was using the most recent, generally available (GA) runtime version of Fabric at the time of test, which is Runtime 1.3. Beginning with Runtime 2.0, which was in Public Preview at the time of writing, Adaptive Target File Size is enabled by default. Delta-level optimization details are provided in the data generation and customization script in Appendix 1.

5.3 Power BI / Semantic model-level optimizations

- The number of visuals on the report page was minimized, in line with DirectQuery best practices.
- An "Apply All Slicers" button was added to reduce the number of queries sent to Databricks under the DirectQuery and Composite Model patterns.
- Assume Referential Integrity was enabled for all relationships involving Fact Tables.
- Large storage format was enabled for all test models.
- Query scale-out was enabled for all test models.
- For load testing, the following evaluation configuration settings were applied to the test capacity (F128), set to their maximum values for both DirectQuery and Composite Model patterns to ensure maximum parallelism:
    - Maximum connections per data source: 75
    - Maximum parallelism per query: 12
- For the Databricks Composite Model, aggregation tables were configured for both Fact Tables.

The test model/report and aggregation table configurations are provided in Appendix 3.

6. Test Scenarios

For each data volume and each pattern, 3 filter scenarios, 3 cache scenarios and 5 queries were run during the single user test. In addition, an auxiliary "Hot Cache 0" measurement was collected for Databricks DirectQuery and Composite Model patterns.

6.1 Filter Scenarios

- **Scenario 1 – No Filter Selected in Slicers**: The report page loads with no slicers applied (Cold Cache for Scenario 1 only).
- **Scenario 2 – Filter on Aggregation Slicers**: The report page loads with slicers applied only on dimensions included in the aggregation table, allowing queries to be served by the aggregation layer whenever possible.
- **Scenario 3 – Filter on All Slicers**: The report page loads with all available slicers applied.

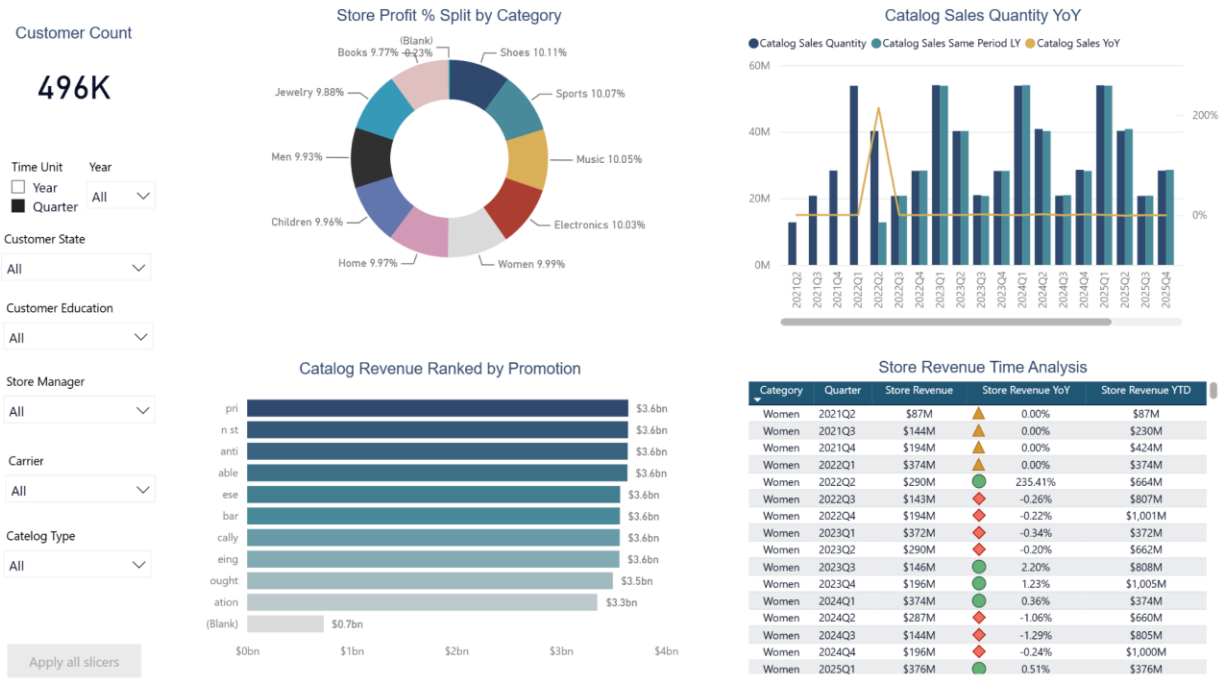


Figure 6.1.1 Test Report Page

6.2 Cache Scenarios

- **Cold Cache**: This represents worst-case performance. For Fabric Direct Lake and Mirrored patterns, this means the data required to answer the query is not resident in memory and must be paged in before the query can execute. In practice, it means a clearValues refresh was run on the test model followed by a full refresh using Semantic Link Labs. For Databricks DirectQuery and Composite Model patterns, this means the SQL warehouse has been restarted and result caching is disabled.
- **Warm Cache**: For Fabric Direct Lake and Mirrored patterns, this means all the parts of the semantic model required by the query are already resident in memory. For Databricks DirectQuery and Composite Model patterns, this means the SQL warehouse has the required data cached on disk, with result caching disabled.
- **Hot Cache**: This represents the fastest scenario. For Fabric Direct Lake and Mirrored patterns, this means all the parts of the semantic model required by the query are already resident in memory, and the semantic model has the data required by the query in its cache. For Databricks DirectQuery and Composite Model patterns, this means the SQL warehouse has the required data cached on disk with result caching enabled, and the Power BI semantic model has the query result cached.
- **Hot Cache 0**: When initially evaluating warm cache performance, surprisingly poor performance was observed for DirectQuery scenarios. Therefore, a Hot Cache 0 experiment was introduced to investigate the behavior. This test left disk caching and result caching fully enabled on the SQL warehouse but cleared the semantic model cache before each DAX query was executed. Performance was also poor; Section 8.1.4 explains the cause.

6.3 Test Queries

Based on common Power BI use cases, five representative DAX queries were designed for this performance test:

- **Query 1** – Distinct Count
- **Query 2** – Percentage Share
- **Query 3** – Rank based on sum, implemented as a visual calculation
- **Query 4** – Year-over-Year (YoY) calendar-based time intelligence on the smaller fact table
- **Query 5** – Year-over-Year (YoY) and Year-to-Date (YTD) calendar-based time intelligence on the largest fact table

These DAX queries were generated by interacting with the Power BI test report page and captured by Performance Analyzer. The full text of the test DAX queries is provided in Appendix 4.

7. Test Setup and Configuration

7.1 Environment specifications

- Azure region: Southeast Asia
- Fabric Capacity: F128
- Databricks SQL Warehouse size:
    - SF10: Medium (serverless)
    - SF100: Large (serverless)
    - SF1000: X-Large (serverless)

7.2 Single-user test setup

- Power BI report interactions were tested across three filter scenarios.
- Performance Analyzer was used to capture the queries generated by each Power BI report interaction.
- The captured queries were then run in DAX Studio with Server Timings traces enabled.
- Cold cache testing was not performed for the filtered scenarios (Scenarios 2 and 3). In real-world usage, a report page is typically loaded before a filter is applied, so filtered interactions do not naturally occur against a cold cache.

- For each combination of query, filter scenario, and cache scenario, at least three iterations were run. For warm cache testing on the filtered scenarios (Scenarios 2 and 3), 4 iterations were run instead, since the first iteration is not fully warm. Because some of the data required to answer the query (i.e., the filter columns) is not yet resident in memory and must first be paged in, the results of this first iteration were excluded from the analysis.

7.2.1 Cache-clearing methodology between runs

- Cold Cache
    - Fabric: A clearValues refresh was run, followed by a full refresh using Semantic Link Labs. Queries were then run using the "Clear on Run" option in DAX Studio, ensuring no Power BI semantic model cache was used. See the clear-cache notebook in Appendix 5.
    - Databricks: The SQL warehouse was restarted to clear the disk cache, and a dummy table-update query was run to make Databricks register the table as updated, thereby disabling the result cache. Queries were then run using the "Clear on Run" option in DAX Studio, ensuring no Power BI semantic model engine cache was used. See the dummy update query in Appendix 6.
- Warm Cache
    - Fabric: Queries were run using the "Clear on Run" option in DAX Studio, ensuring no Power BI semantic model cache was used.
    - Databricks: A pre-warm query (CACHE SELECT * FROM table_name;) was run to populate the SQL warehouse disk cache, followed by a dummy table-update query to disable the result cache. Queries were then run using the "Clear on Run" option in DAX Studio, ensuring that semantic model caches were cleared prior to each execution. See the pre-warm and dummy update queries in Appendix 6.
- Hot Cache
    - Fabric: Queries were run **without** the "Clear on Run" option in DAX Studio, allowing the Power BI semantic model cache to be used.
    - Databricks: Queries were executed **without** using DAX Studio's "Clear on Run" option, allowing caches from previous executions at both the Power BI and Databricks layers to be potentially reused.
- Hot Cache 0
    - Databricks: Queries were run using the “Clear on Run” option in DAX Studio, clearing the Power BI Storage Engine cache before each execution. Databricks-side caches were not cleared.

7.3 Load test setup

- Interactions were made to Power BI test report page, and Performance Analyzer was used to capture interactions with the test report.
- The JSON file generated by Performance Analyzer was uploaded to the load testing tool to run load tests for 3 data volumes across 4 different patterns.
- Concurrent user counts tested: 20 (In practice, BI solutions rarely experience more than a handful of genuinely concurrent users, and a load of 20 concurrent users represents a reasonable upper bound for most real-world scenarios).
- For each data volume and each pattern, 3 iterations were run.
- The load test for SF1000 using Direct Lake over Mirrored Unity Catalog tables could not be completed, as it exceeded the available memory limit on the F128 capacity as shown in Figure 7.3.1. As a result, load testing results for this configuration are excluded from the analysis. This is expected considering the results for single-user Warm Cache tests of Direct Lake over Mirrored Unity Catalog tables as shown in Section 8.1.2.
- To ensure accurate load testing results, the load testing tool was customized to randomize the value used in the cache_buster < [random value] filter condition applied to fact table queries. This customization was necessary to prevent Databricks result caching from being triggered. If identical queries were issued across concurrent users and iterations, Databricks would serve subsequent requests from the result cache rather than executing them against the cluster. Since cached results do not consume cluster compute slots, this would understate real concurrency load and undermine the validity of the load test.
- Load tests were run within the F128 capacity limit, i.e., no results were skewed due to throttling.

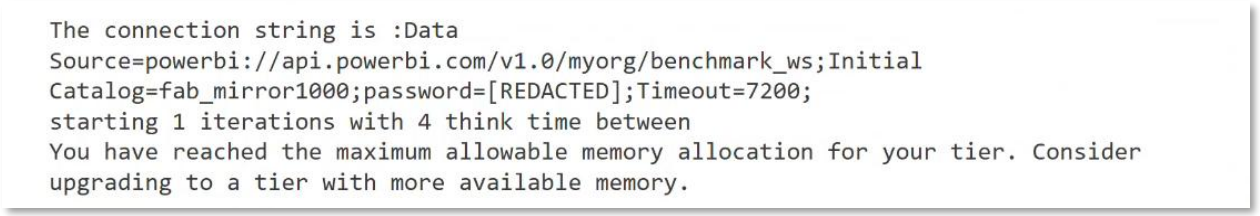


Figure 7.3.1 SF1000 Direct Lake over Mirrored Unity Catalog load-test memory error

Test report, Performance Analyzer-generated JSON, and customized load testing tool are provided in Appendices 2, 3, and 7.

7.4 Control variables

- Identical dataset volume and schema
- Identical report/semantic model logic
- Identical test queries

8. Test Results and Analysis

8.1 Single-User Performance Results

- All results presented in this section are reported in milliseconds.
- Results show the Cold Cache Total Duration, as well as the Warm Cache and Hot Cache Total Duration and Storage Engine Duration for each query, based on DAX Studio Server Timings traces.
- The result analysis notebook and accompanying Power BI report are provided in Appendix 8.
- No throttling of any kind occurred during the single-user test.
- As discussed in Section 8.1.4, clearing the Power BI semantic model cache and rebuilding the DirectQuery metadata cache means that the Total Duration metric for DirectQuery and some Composite Model scenarios carries an overhead of approximately 4 seconds. Therefore, to allow for a fairer comparison for Warm Cache scenarios, Storage Engine Duration for queries has been added as an additional metric. Total Duration is the appropriate comparison metric for Cold Cache scenarios. For more information on what the Total Duration and Storage Engine Duration metrics mean, see Section 9.1.

8.1.1 Cold Cache

8.1.1.1 Scenario 1 (No Filter Selected in Slicers)

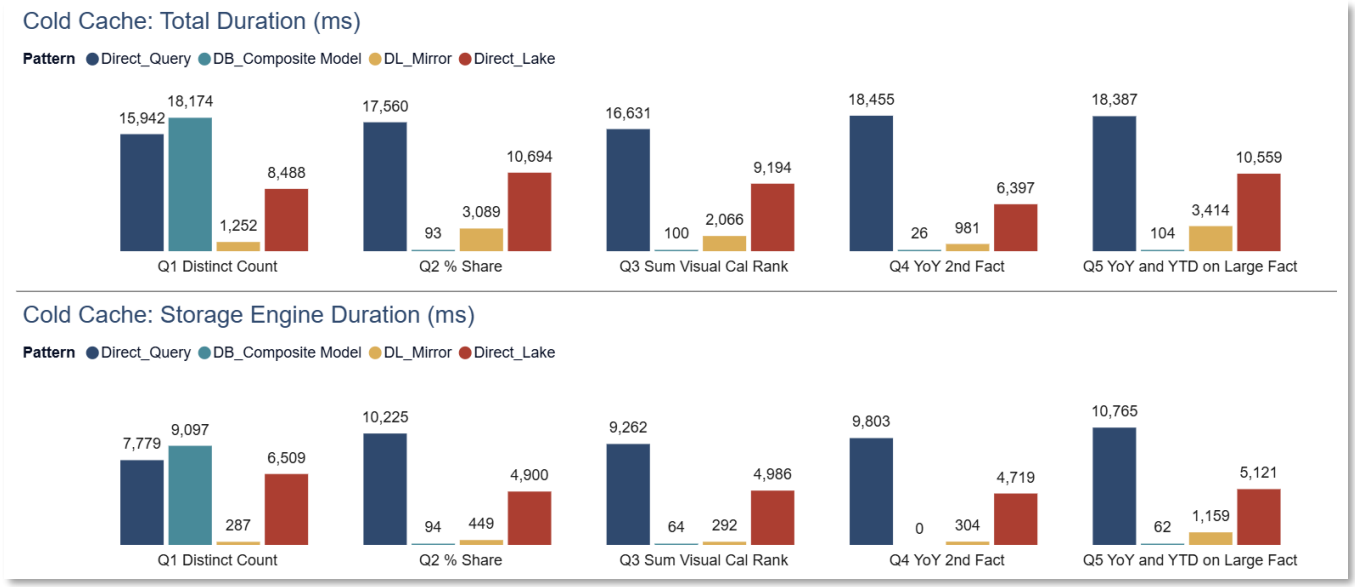
Scenario 1 (No Filter Selected in Slicers) is the only filter scenario in which cold cache performance was evaluated. Cold cache query performance varies by data volume, as summarized below:

- **"Cold Cache"** means something different for the Composite Model because its aggregation tables operate in Import mode: while the Power BI semantic model cache is cleared and Import mode tables can be paged in and out of memory like Direct Lake tables, there is no easy way to ensure that they have been paged out of memory before any testing. Because Query 1 (Distinct Count) does not utilize the aggregation tables, its result reflects a "Cold Cache" performance that is more consistent with the other cold cache scenarios tested. The following findings describe the performance excluding the Composite Model results for Query 2-5.
- All analysis is based on the Total Duration metric. Storage Engine Duration is shown purely for consistency with other caching scenarios.
- At **Scale Factor 10**, Direct Lake over Mirrored outperforms Direct Lake, which in turn outperforms DirectQuery.
- At **Scale Factor 100**, results vary by query: DirectQuery is fastest for Query 2 (% Share) and Query 5 (time intelligence on the largest fact table); Direct Lake is fastest for Query 1 (Distinct Count); and Direct Lake

over Mirrored is fastest for Query 3 (visual calculation for Rank) and Query 4 (simple time intelligence on the second fact table).

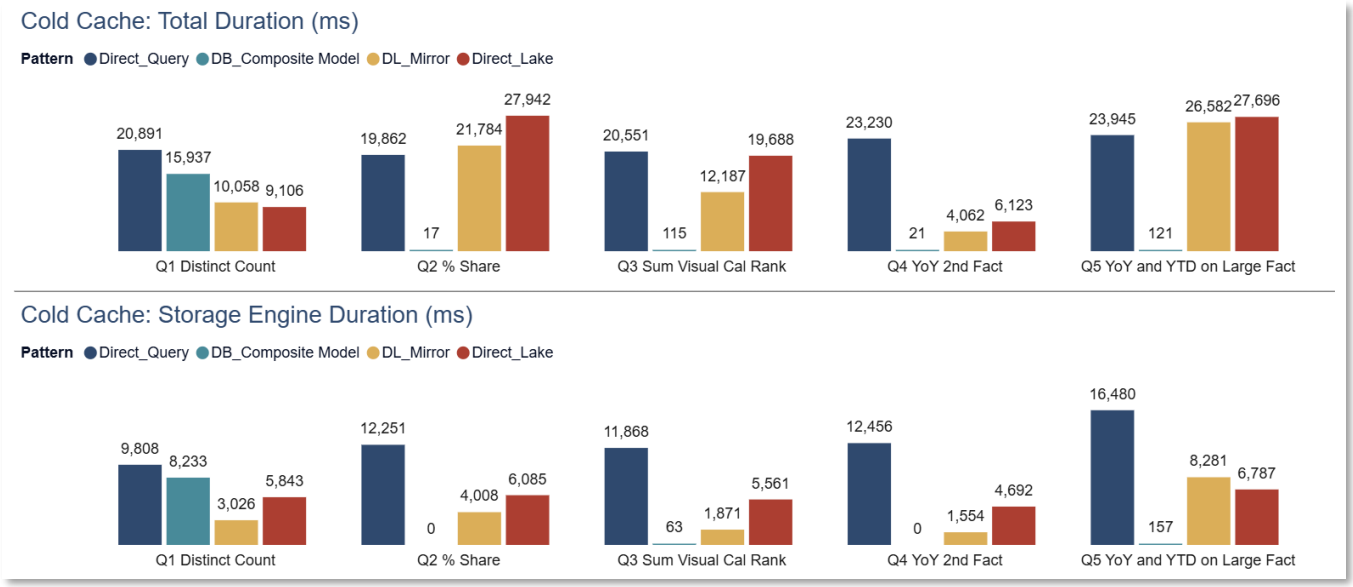
- At **Scale Factor 1000**, DirectQuery outperforms Direct Lake across all queries except Query 4 (simple time intelligence on the second fact table), while Direct Lake over Mirrored is the slowest across all queries.

Duration (ms; lower is better)



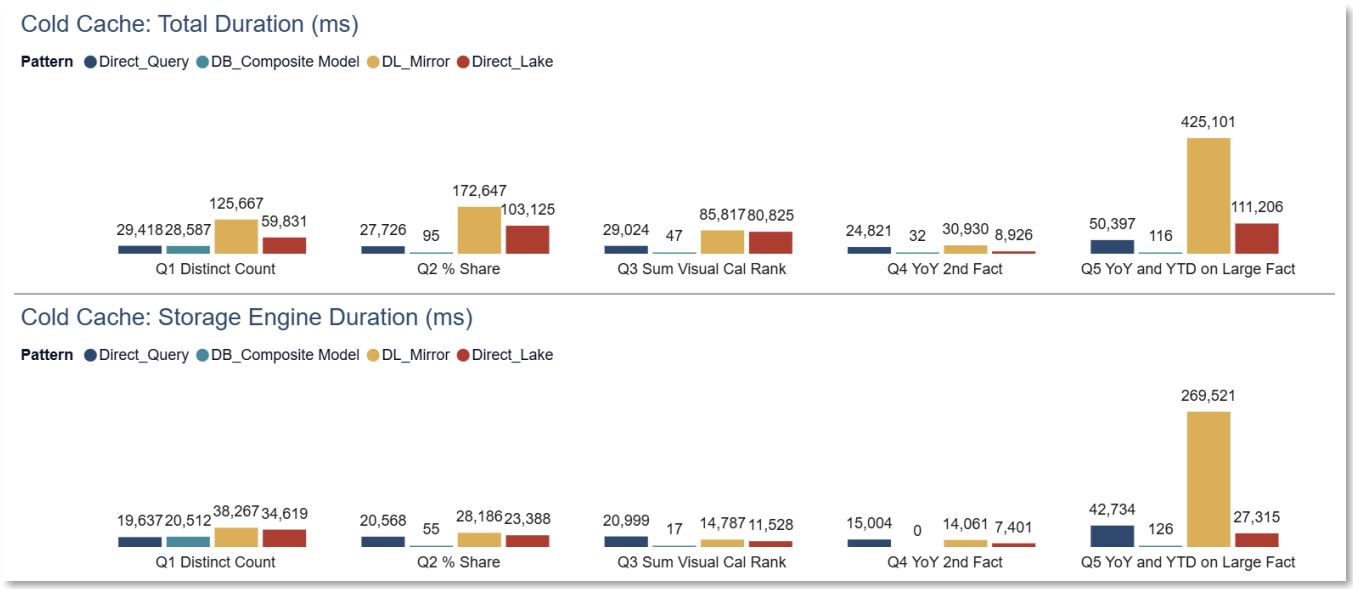
Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.1.1 Cold Cache Results Scenario 1 Scale Factor 10



Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.1.2 Cold Cache Results Scenario 1 Scale Factor 100



Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

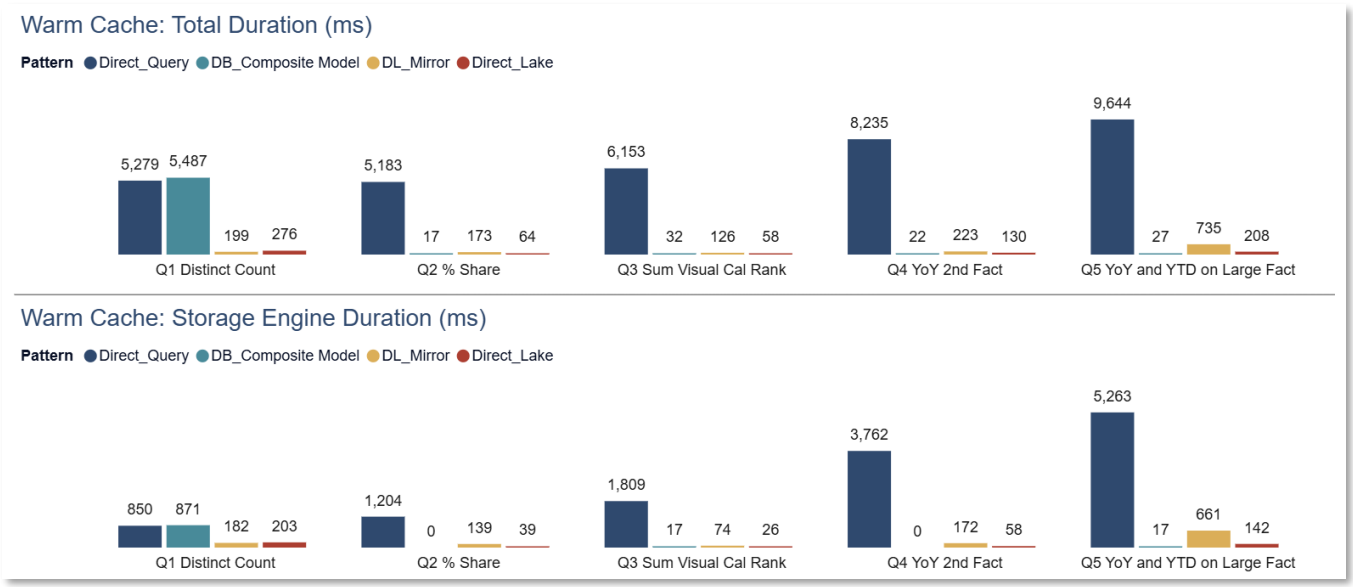
Figure 8.1.1.3 Cold Cache Results Scenario 1 Scale Factor 1000

8.1.2 Warm Cache

8.1.2.1 Scenario 1 (No Filter Selected in Slicers)

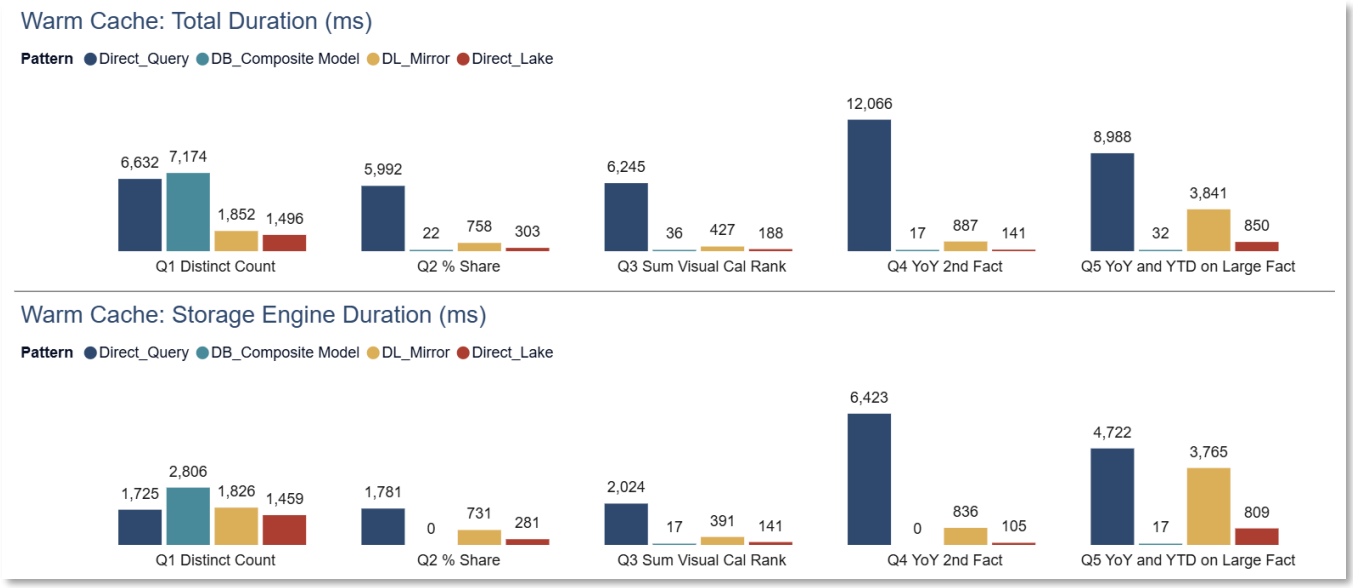
- All analysis is based on the Storage Engine Duration metric.
- In Scenario 1, where no filter is selected in the slicers, Queries 2–5 hit the Import mode aggregation tables for the Composite Model. As a result, Composite Model performance across Queries 2–5 is very fast (<100 ms) across all data volumes. Since Query 1 does not utilize the aggregation tables, Composite Model performance for this query is very close to that of DirectQuery.
- At **Scale Factor 10**, Direct Lake, Direct Lake over Mirrored, and the Composite Model (except for Query 1, which does not hit the aggregation tables) all achieve sub-second performance.
- At **Scale Factor 100**, Direct Lake and the Composite Model achieve sub-second performance except for Query 1 (Distinct Count). Direct Lake over Mirrored also achieves sub-second performance, except for Query 1 (Distinct Count) and Query 5 (time intelligence on the largest fact table).
- At **Scale Factor 1000**, excluding the Composite Model, DirectQuery is fastest for Query 1 (Distinct Count) and Query 2 (% Share), while Direct Lake is fastest across the remaining queries.

Duration (ms; lower is better)



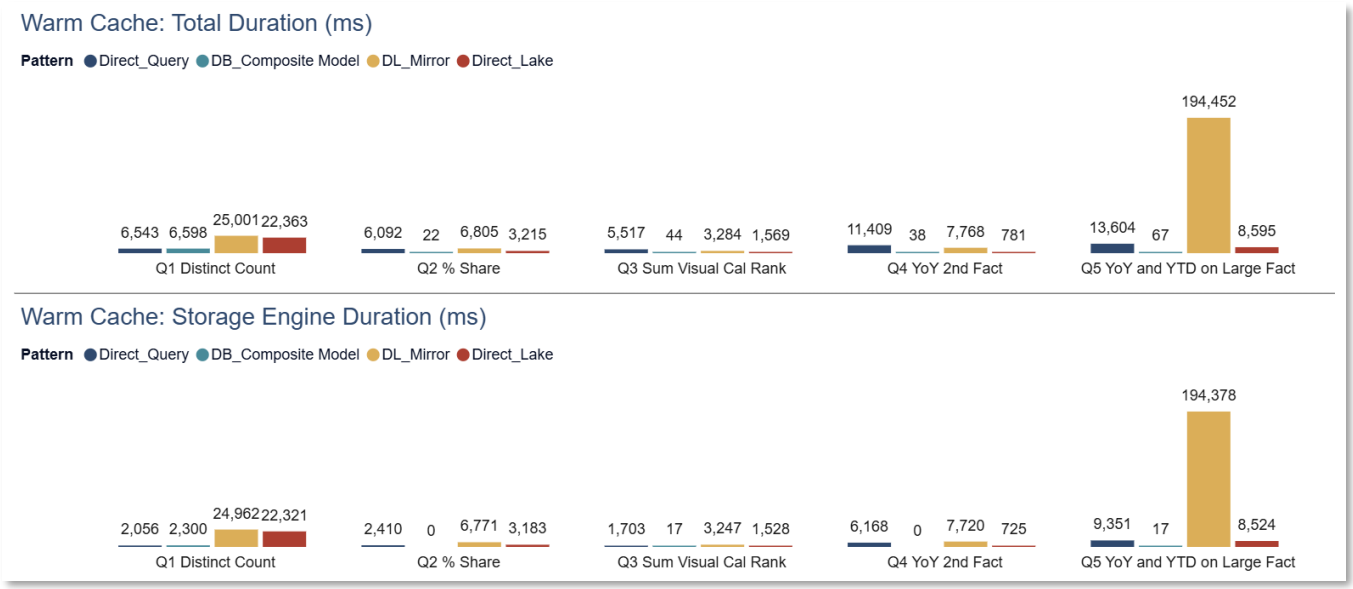
Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.2.1 Warm Cache Results Scenario 1 Scale Factor 10



Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.2.2 Warm Cache Results Scenario 1 Scale Factor 100



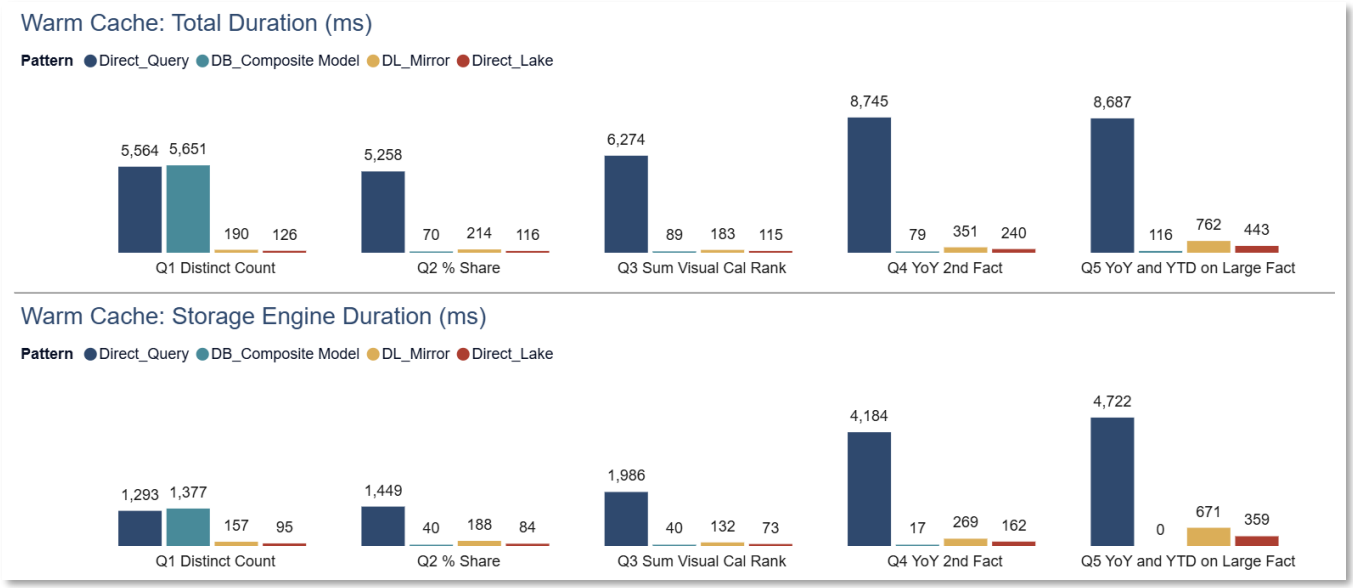
Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.2.3 Warm Cache Results Scenario 1 Scale Factor 1000

8.1.2.2 Scenario 2 (Filter on Aggregation Slicers)

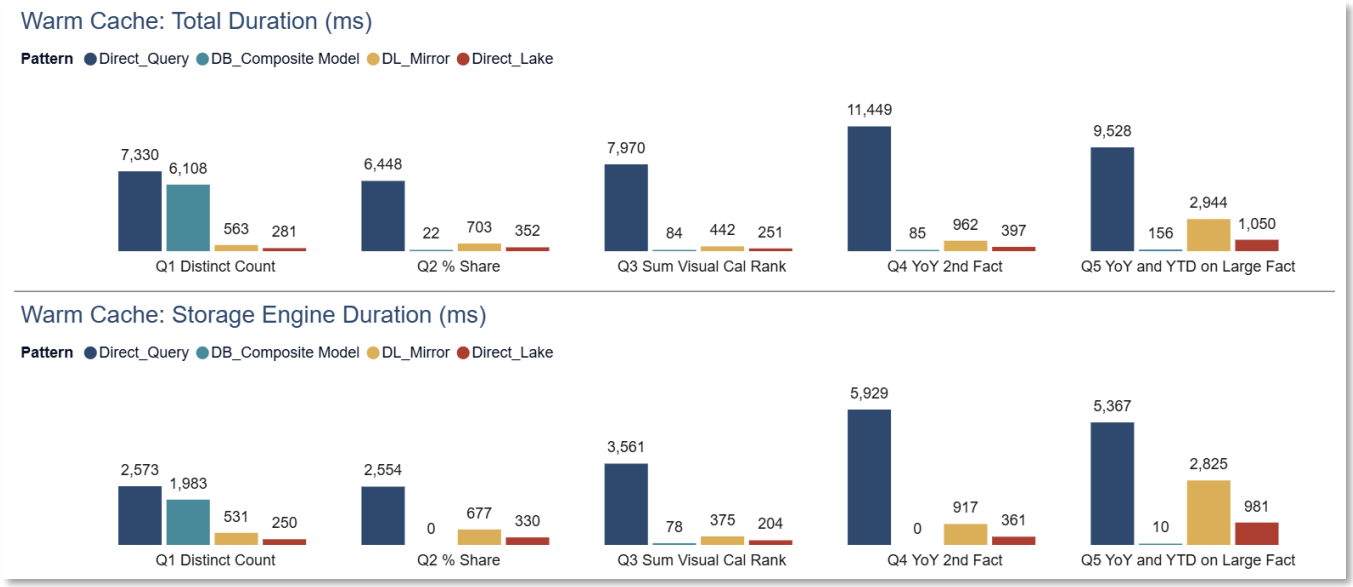
- All analysis is based on the Storage Engine Duration metric.
- In Scenario 2, where slicers are filtered on group-by columns present in the aggregation tables, Queries 2–5 hit the aggregation tables in Import mode for the Composite Model. As a result, Composite Model performance across Queries 2–5 is very fast (<100 ms) across all data volumes. Since Query 1 does not utilize the aggregation tables, Composite Model performance for this query is very close to that of DirectQuery.
- At **Scale Factor 10**, Direct Lake and Direct Lake over Mirrored achieve sub-second Storage Engine Duration across all queries. The Composite Model also achieves sub-second performance for Queries 2–5; Query 1 exceeds one second because it does not utilize the aggregation tables.
- At **Scale Factor 100**, Direct Lake maintains sub-second Storage Engine Duration across all queries. Direct Lake over Mirrored also remains below one second for Queries 1–4, with Query 5 (time intelligence on the largest fact table) being the only exception.
- At **Scale Factor 1000**, excluding the Composite Model, Direct Lake delivers the lowest Storage Engine Duration for Queries 1 through 4, while DirectQuery is slightly faster for Query 5. Direct Lake over Mirrored is consistently the slowest pattern across all queries. At this data volume, the Storage Engine Duration for Direct Lake over Mirrored for Queries 1, 2, and 5 becomes excessively high, making it unsuitable for interactive Power BI experiences.

Duration (ms; lower is better)



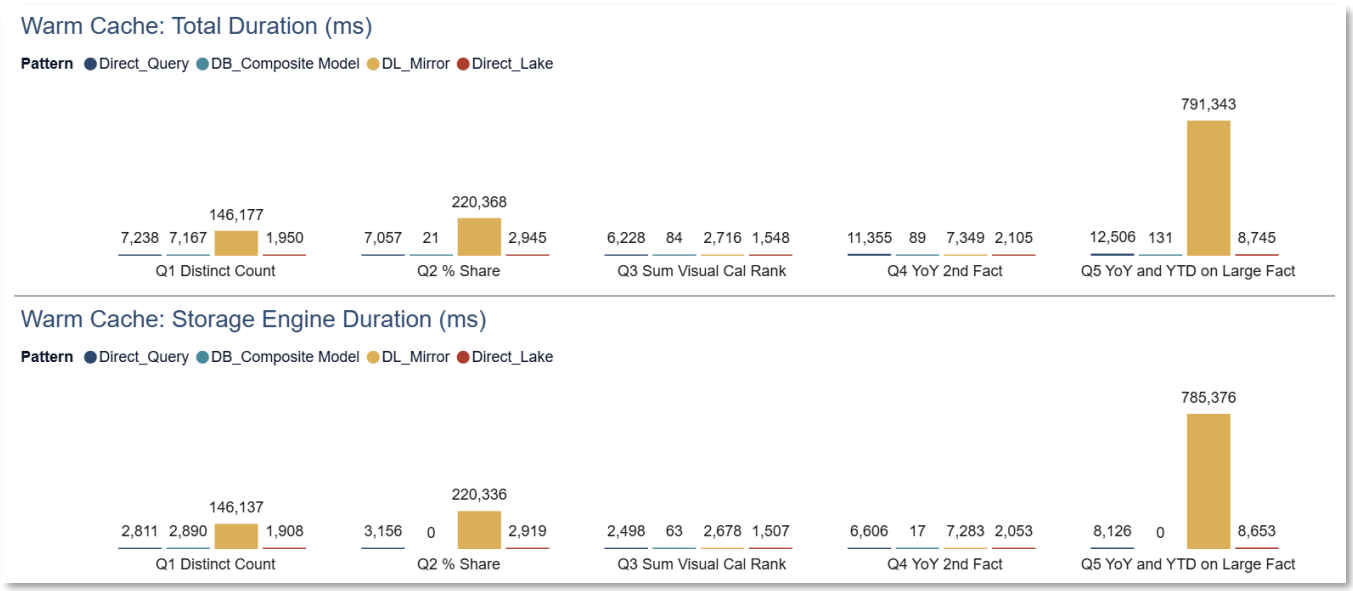
Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.2.4 Warm Cache Results Scenario 2 Scale Factor 10



Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.2.5 Warm Cache Results Scenario 2 Scale Factor 100



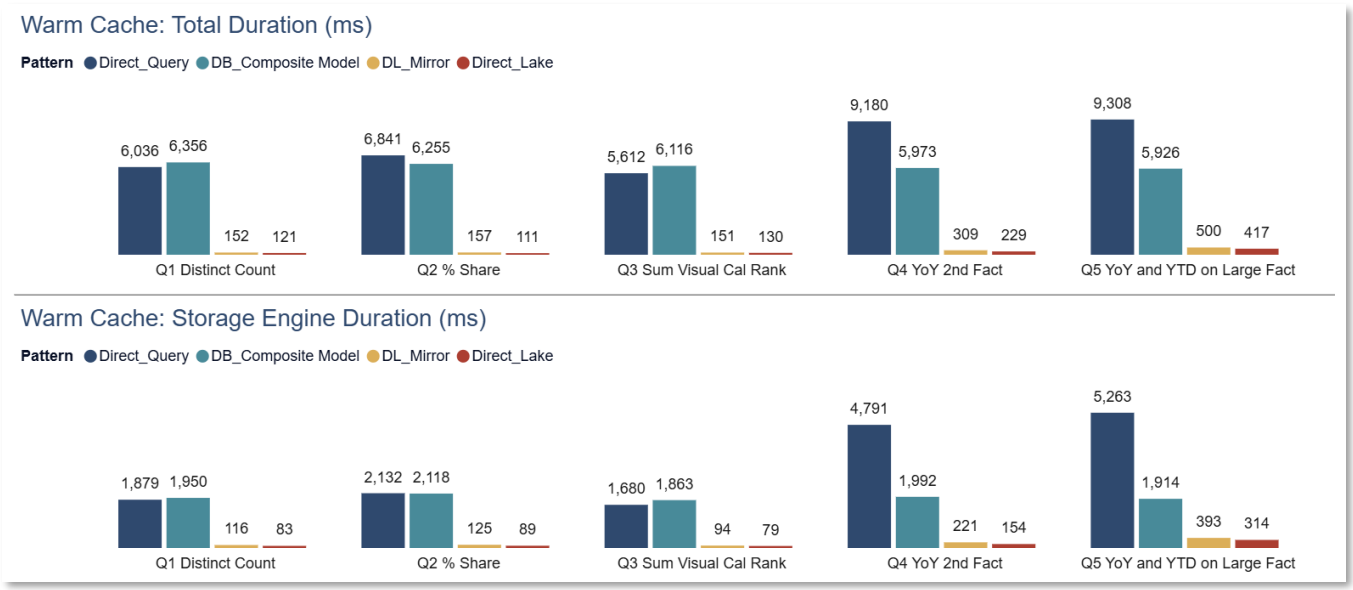
Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.2.6 Warm Cache Results Scenario 2 Scale Factor 1000

8.1.2.3 Scenario 3 (Filter on All Slicers)

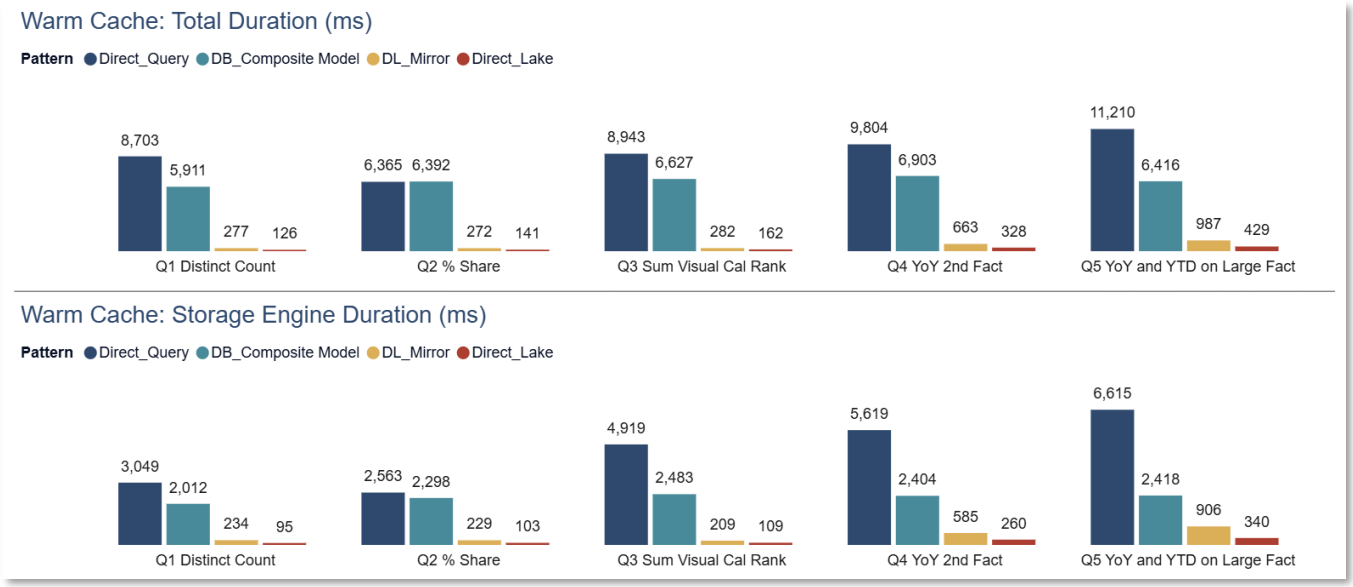
- In Scenario 3, where all slicers are filtered, the Composite Model requires more granular data than is present in the aggregation tables to resolve the query. As a result, Composite Model performance is similar to, and in some cases slightly faster than, DirectQuery across all queries and all data volumes.
- At **Scale Factor 10** and **Scale Factor 100**, both Direct Lake and Direct Lake over Mirrored achieve sub-second performance across all queries.
- At **Scale Factor 1000**, Direct Lake is fastest across all queries, maintaining sub-second performance. Direct Lake over Mirrored is the slowest across all queries at this data volume, with latency too high to be acceptable for normal Power BI interactions.

Duration (ms; lower is better)



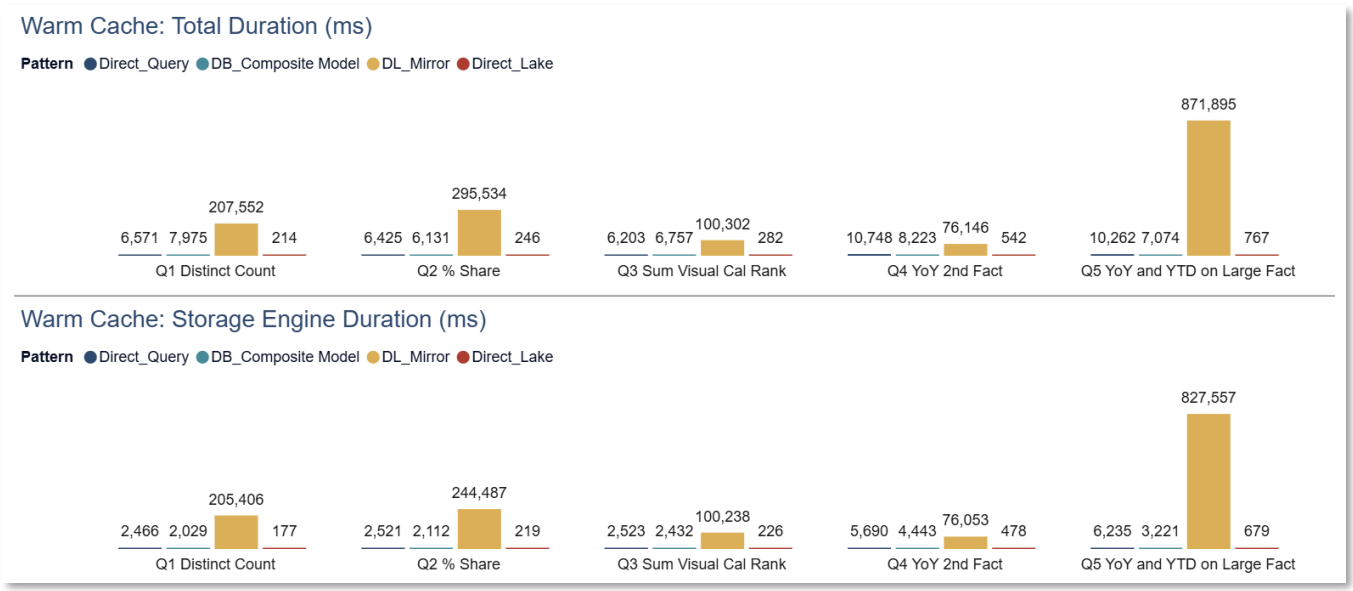
Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.2.7 Warm Cache Results Scenario 3 Scale Factor 10



Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.2.8 Warm Cache Results Scenario 3 Scale Factor 100



Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

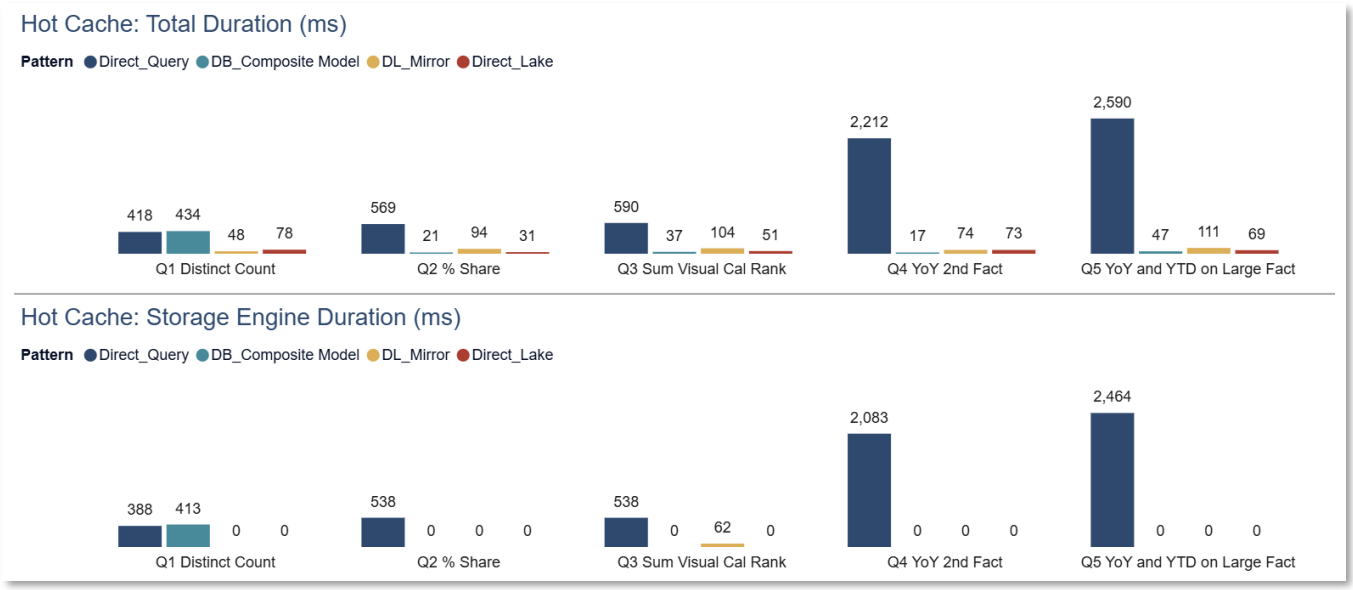
Figure 8.1.2.9 Warm Cache Results Scenario 3 Scale Factor 1000

8.1.3 Hot Cache

Scenario 1 (No Filter Selected in Slicers)

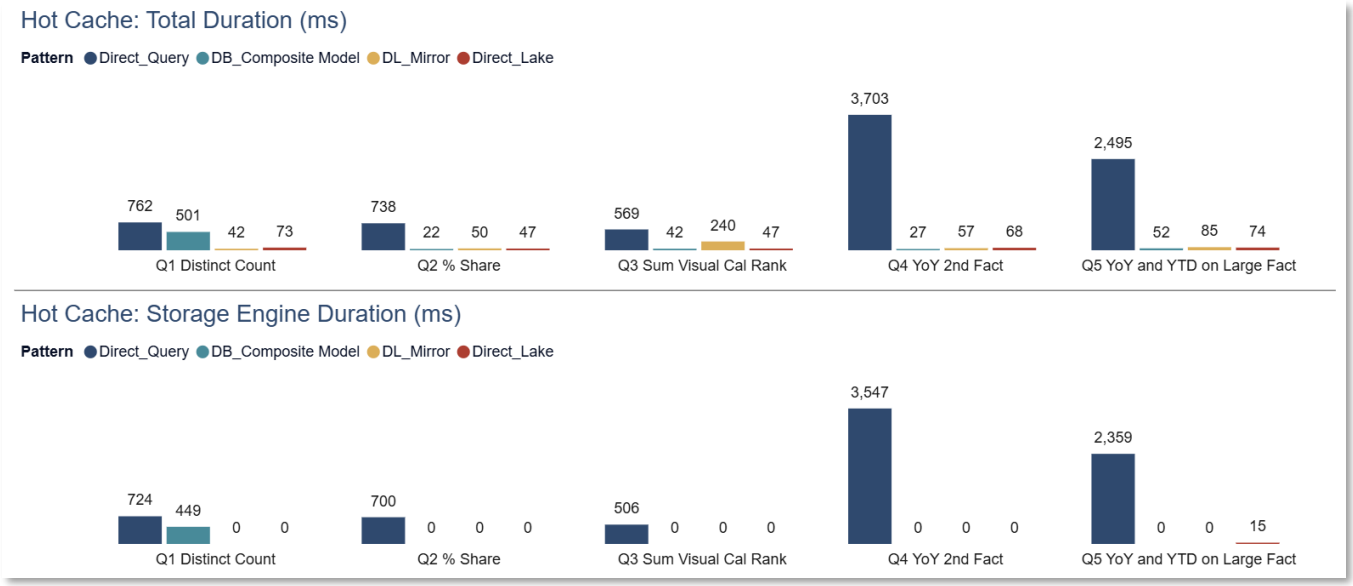
- Since DAX queries could be answered from the Power BI semantic model cache in this scenario, it is expected that the Storage Engine Duration metric is 0 or negligible for all patterns except DirectQuery and that the Total Duration metric is very similar to the Storage Engine Duration metric.
- In Scenario 1, with no slicer filters applied, Queries 2–5 in the Composite Model can be answered directly from the Import-mode aggregation tables rather than the underlying Fact Tables. As a result, Composite Model performance for Queries 2–5 remains extremely fast (<100 ms) across all data volumes. Since Query 1 does not utilize the aggregation tables, Composite Model performance for this query is very close to that of DirectQuery.
- Performance is consistent across all data volumes. The Composite Model delivers the best performance for all queries except Query 1 (Distinct Count), with Direct Lake a close second, followed by Direct Lake over Mirrored. DirectQuery is generally the slowest, although it still achieves sub-second response times for Queries 1–3.

Duration (ms; lower is better)



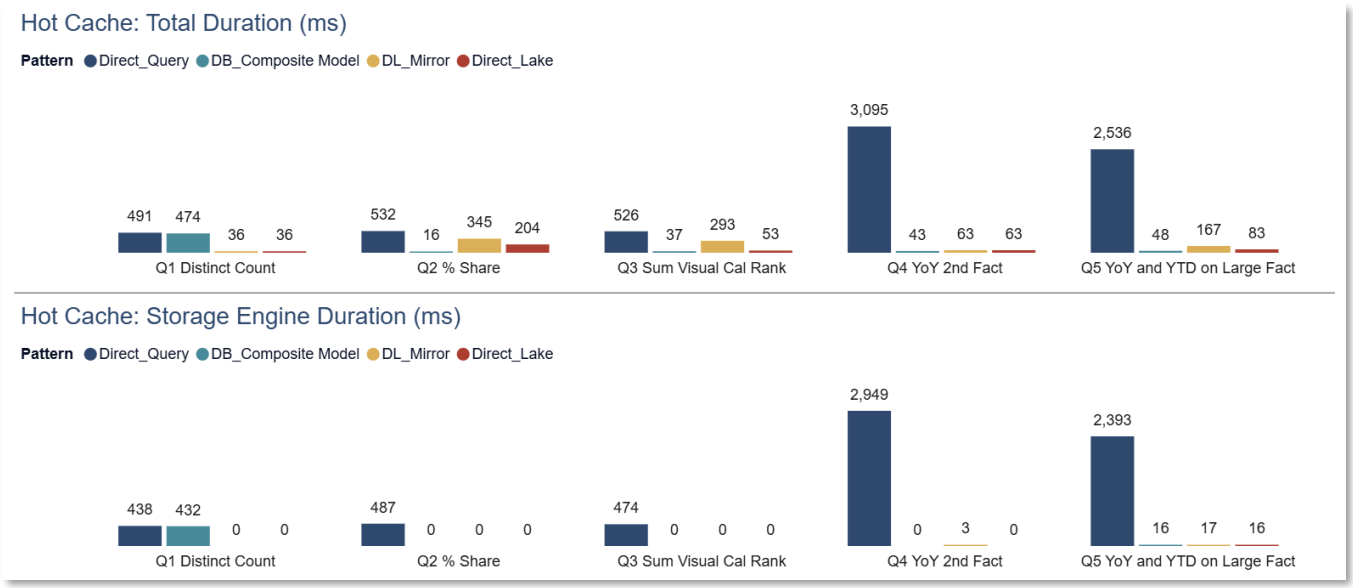
Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.3.1 Hot Cache Results Scenario 1 Scale Factor 10



Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.3.2 Hot Cache Results Scenario 1 Scale Factor 100



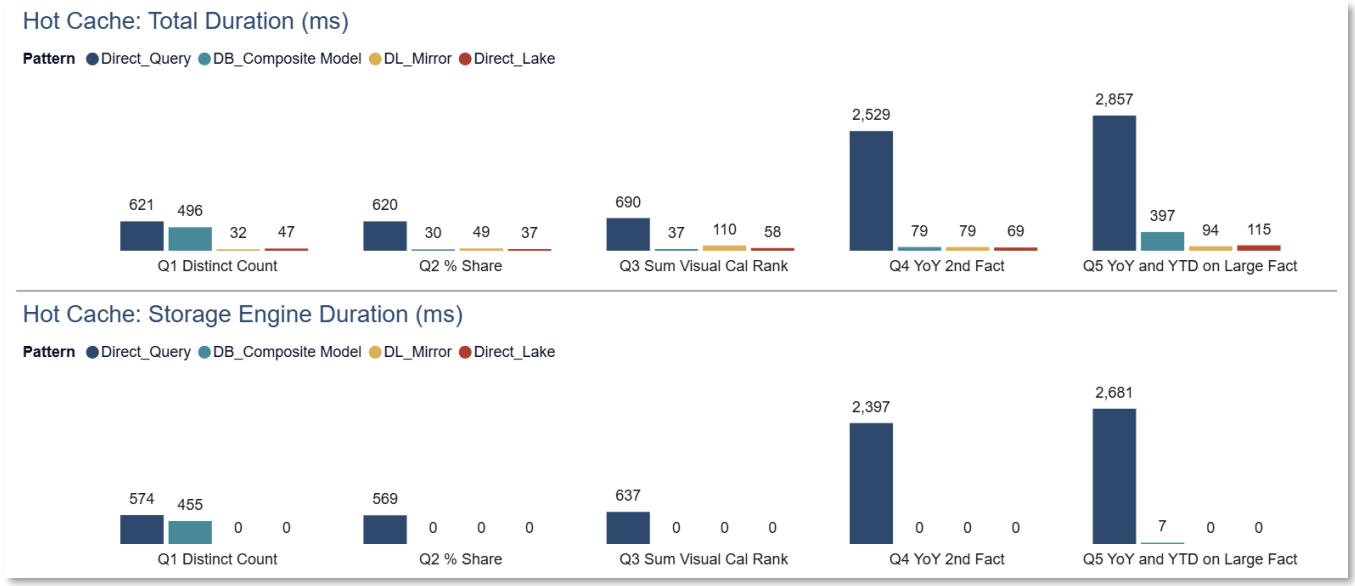
Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.3.3 Hot Cache Results Scenario 1 Scale Factor 1000

Scenario 2 (Filter on Aggregation Slicers)

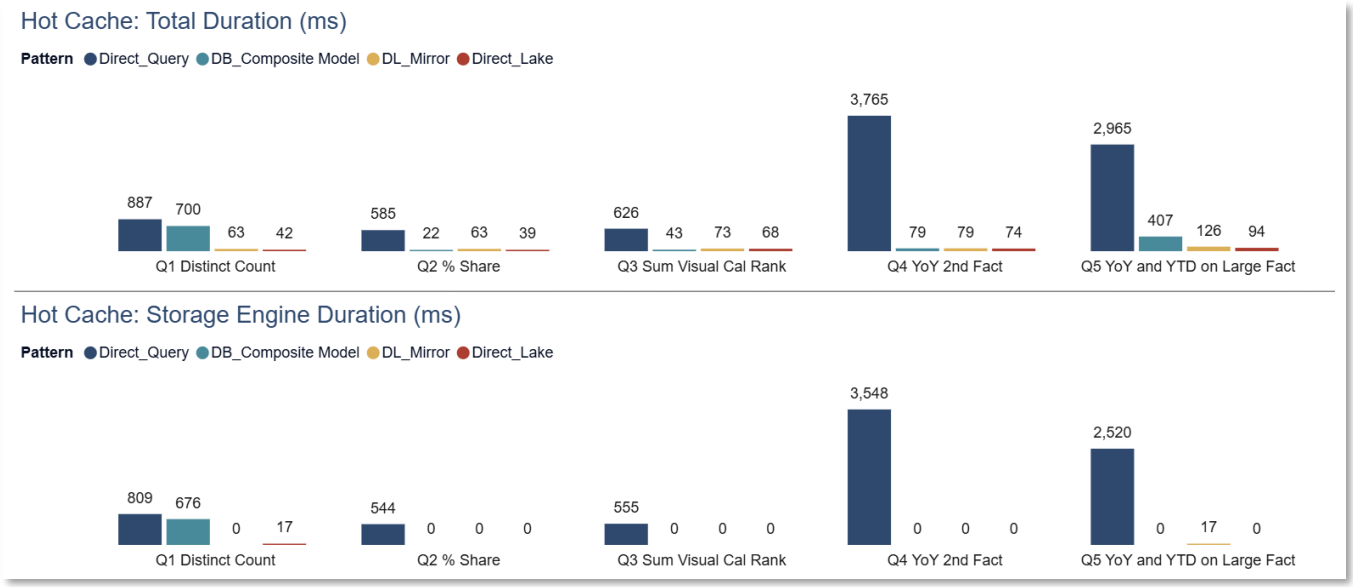
- In Scenario 2, the slicers filter on dimensions supported by the aggregation tables, allowing Queries 2–5 in the Composite Model to be answered directly from Import-mode aggregations. As a result, Composite Model performance for Queries 2–5 remains extremely fast (<100 ms) across all data volumes. Query 1 does not benefit from the aggregation tables and therefore exhibits performance similar to DirectQuery.
- Performance is consistent across all data volumes. Direct Lake is fastest across all queries, followed by Direct Lake over Mirrored. DirectQuery is generally the slowest, although it still delivers sub-second response times for Queries 1–3. The one exception is Query 5 (time intelligence on the largest fact table) at Scale Factor 1000, where Direct Lake over Mirrored becomes the slowest-performing pattern observed in this scenario.

Duration (ms; lower is better)



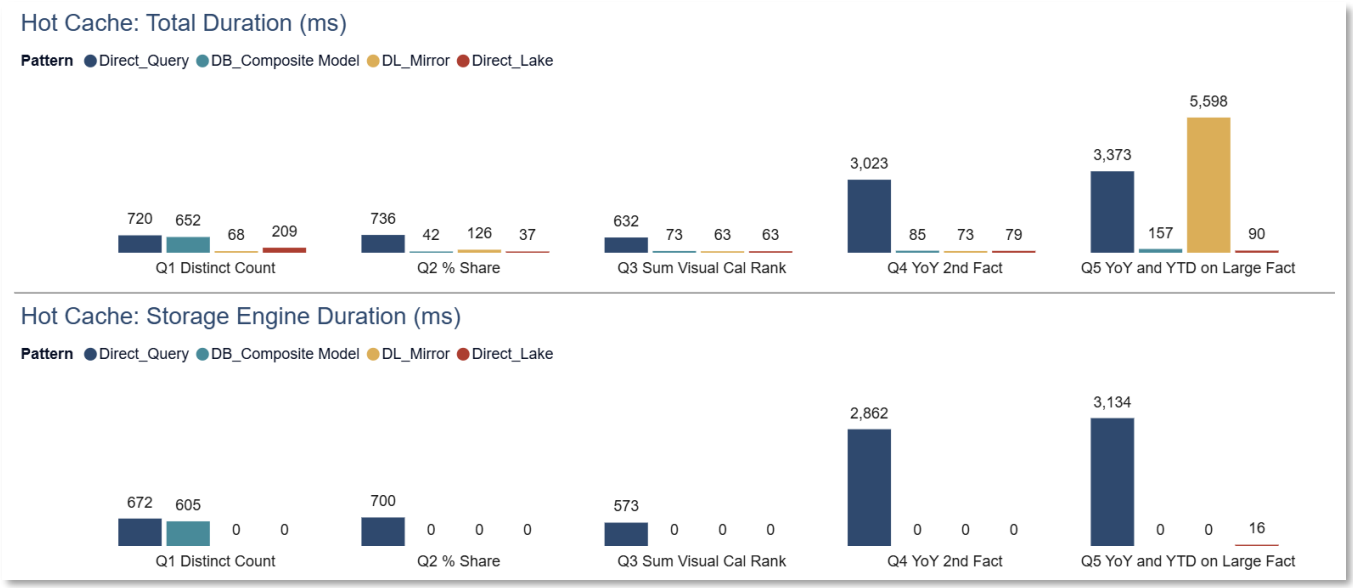
Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.3.4 Hot Cache Results Scenario 2 Scale Factor 10



Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.3.5 Hot Cache Results Scenario 2 Scale Factor 100



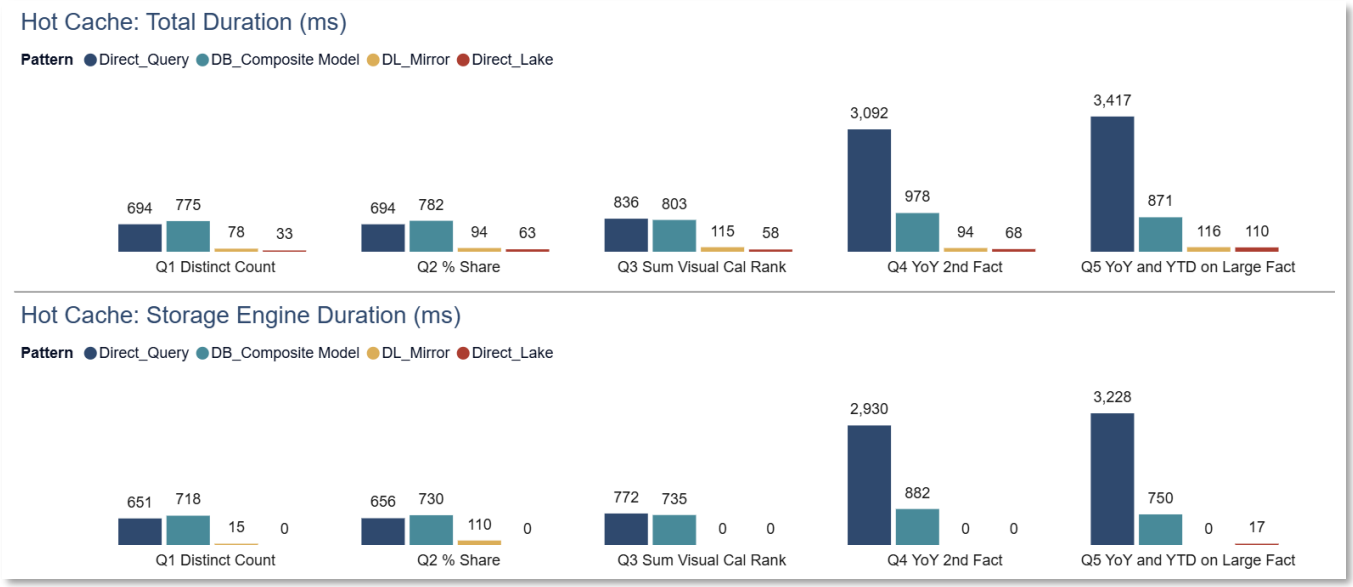
Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.3.6 Hot Cache Results Scenario 2 Scale Factor 1000

Scenario 3 (Filter on All Slicers)

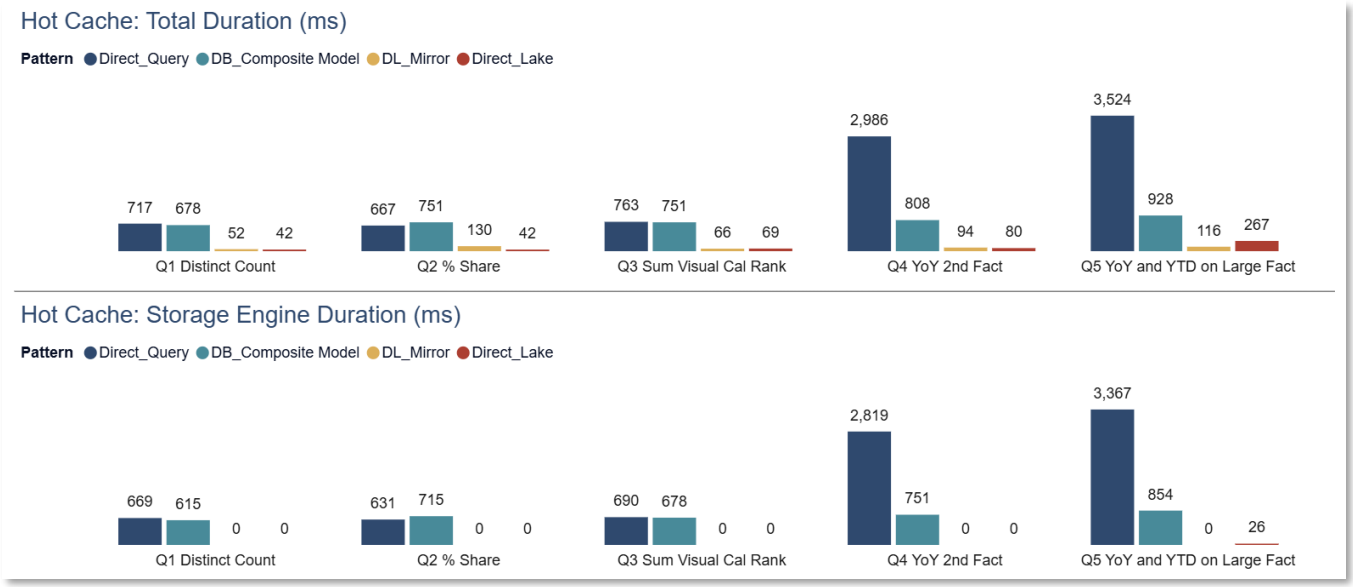
- In Scenario 3, with all slicers applied, the Composite Model requires more granular data than is present in the aggregation tables to resolve the query. As a result, Composite Model performance is similar to, and in some cases slightly faster than, DirectQuery across all queries and all data volumes.
- At **Scale Factor 10** and **Scale Factor 100**, Direct Lake delivers the fastest performance across all queries, followed by Direct Lake over Mirrored. DirectQuery is generally the slowest approach, although it still achieves sub-second performance for Queries 1–3.
- At **Scale Factor 1000**, Direct Lake remains the fastest across all queries. Direct Lake over Mirrored is generally the slowest, with the exception of Query 3 (visual calculation for Rank) and Query 4 (simple time intelligence on the second fact table).

Duration (ms; lower is better)



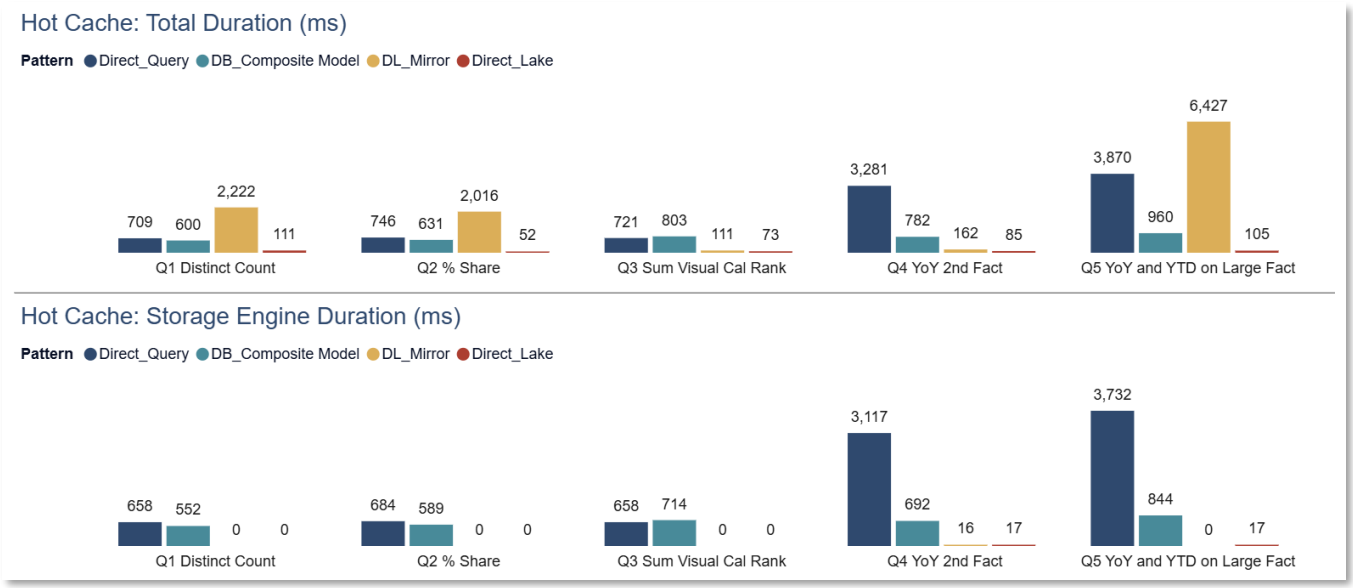
Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.3.7 Hot Cache Results Scenario 3 Scale Factor 10



Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.3.8 Hot Cache Results Scenario 3 Scale Factor 100



Note: Due to independent Y-axis auto-scaling, Total Duration and Storage Engine Duration bars may appear similar in length even when the underlying values are different. Duration = 0 means the time is too small to be captured.

Figure 8.1.3.9 Hot Cache Results Scenario 3 Scale Factor 1000

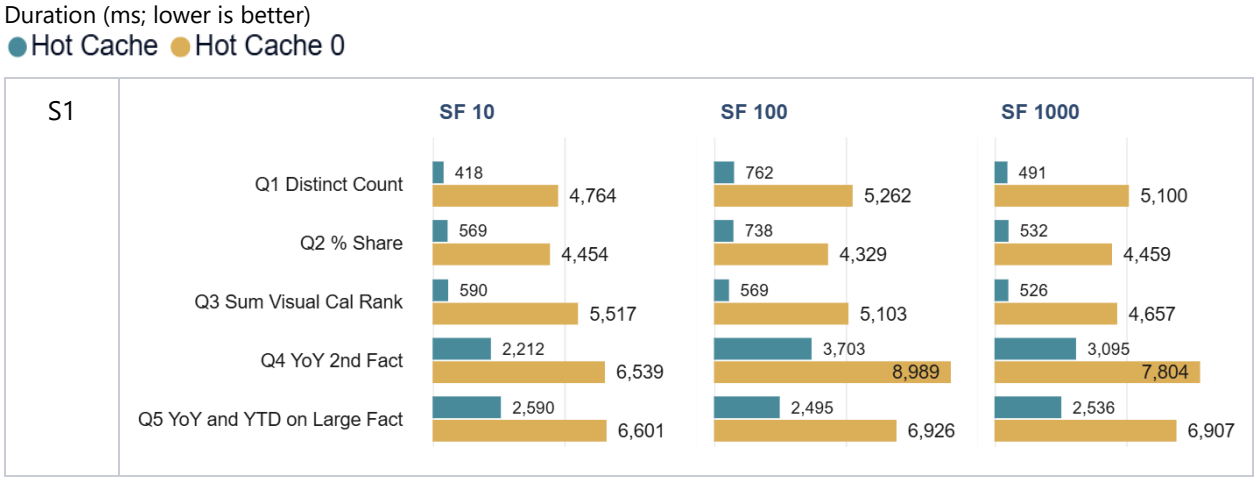
8.1.4 Hot Cache vs Hot Cache 0

As mentioned earlier, the initial performance results for Warm Cache on DirectQuery indicated surprisingly poor performance. Therefore, an additional "Hot Cache 0" test was performed to investigate the behavior. This test left SQL warehouse disk caching and result set caching enabled, but it cleared the Power BI semantic model cache before each DAX query using the ClearCache command. As shown in Table 8.1.4.1, Hot Cache 0 is consistently about 4 seconds slower than Hot Cache. The results were surprisingly similar to the Warm Cache configuration and therefore were investigated more deeply.

After investigation, the root cause of the poor performance of DirectQuery in Warm Cache and the Hot Cache 0 mode was identified. DirectQuery against Databricks and other data sources uses the Power Query engine internally, and the Power Query engine needs to fetch metadata (for example, tables, columns, and data types) from the data source so that it can generate queries against it.

Retrieving this metadata can be expensive. To reduce this overhead, the Analysis Services and Power Query engines maintain metadata caches that are normally reused across query executions. However, the ClearCache command clears these metadata caches in addition to other internal caches. As a result, subsequent queries must re-fetch metadata from the data source, introducing additional latency and causing the poor performance observed in the Warm Cache and Hot Cache 0 scenarios.

The Warm Cache and Hot Cache 0 results are affected by the repeated rebuilding of metadata caches following each ClearCache operation, an overhead that is rarely encountered in production environments. Consequently, the total query durations for these tests are not fully representative of typical DirectQuery or Composite Model performance. The tests also captured the amount of time the DAX queries spent in the Power BI Storage Engine and these timings provide an acceptable way of comparing the performance across different storage modes. An even better methodology would rebuild the metadata cache after ClearCache and before measuring DAX query execution time.



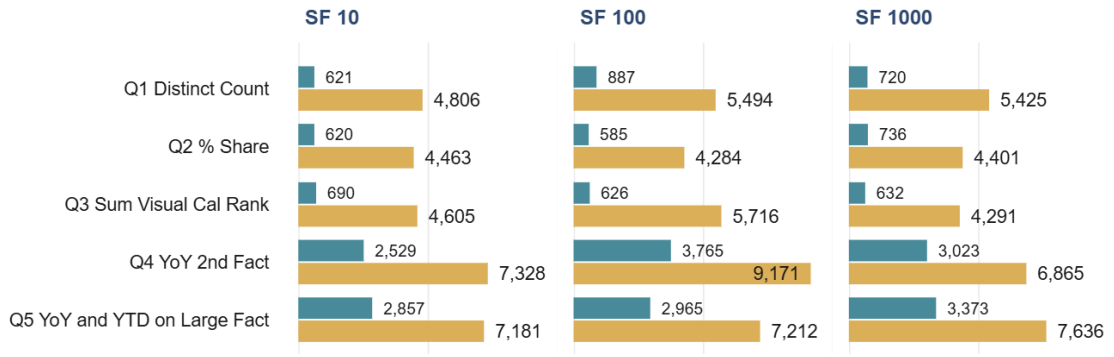
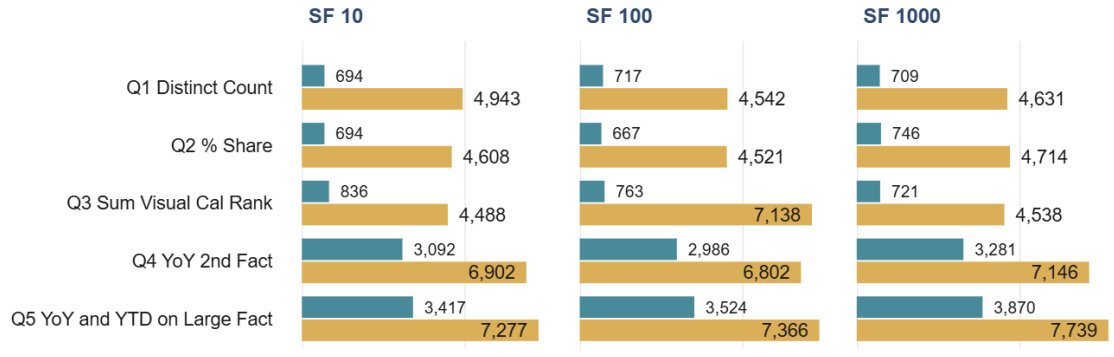
S2	
S3	

Table 8.1.4.1 DirectQuery Total Duration: Hot Cache vs. Hot Cache 0

8.2 Load test results

- All results presented in this section are reported in milliseconds.
- P50 query duration was calculated per query, per data volume, per pattern, across 20 concurrent users, meaning 50% of measured query executions were faster and 50% were slower.
- Results shown represent the average P50 query duration across 3 iterations per query, per data volume, per pattern.
- Results shown exclude query duration for slicer interactions.
- The result analysis notebook and accompanying Power BI report are provided in Appendix 8.
- No throttling occurred during the load test.

As described in Section 7.3 (Load Test Setup), the load testing tool was customized to randomize the filter values applied to fact table queries to prevent Databricks result caching and ensure a meaningful concurrency test. The same approach was used for the Fabric Direct Lake load test. As a result, each query reached the VertiPaq engine with a different filter value, preventing effective reuse of the Power BI semantic model cache. Consequently, the load test results are more representative of Warm Cache behavior than Hot Cache behavior observed in the single-user tests.

Query Name	Query Number
Distinct Count	5, 13, 23
Percentage Share	2, 10, 20
Visual Calculation for Rank	4, 12, 22
Simple Time Intelligence for the Second Fact Table	1, 9, 19
Time Intelligence for the largest Fact Table	6, 14, 24

Table 8.2.1 Query Number and Names during Load Test

- **Composite Model** query performance varies considerably depending on whether the query can be resolved using the aggregation tables. For queries that can leverage the aggregation tables, namely Queries 2, 4, 6, 9, and 12, the load test achieves sub-second performance across all data volumes. The remaining queries perform similarly to, or slightly faster than, DirectQuery.
- Databricks **DirectQuery** performance is similar across different data volumes.
- Some outliers were observed in the **DirectQuery** and **Composite Model** results, where individual queries took an exceptionally long time to complete: Query 5 for the Composite Model and Query 1 for DirectQuery at Scale Factor 10, and Query 1 for DirectQuery at Scale Factor 100.

- **Direct Lake over Mirrored** performs comparably to Direct Lake at Scale Factor 10 but is considerably slower than Direct Lake at Scale Factor 100. No results are available at Scale Factor 1000, as the load test exceeded the available memory limit on the F128 capacity.
- **Direct Lake** performs better than DirectQuery and is comparable to Composite Model and Direct Lake over Mirrored at Scale Factor 10. At Scale Factor 100, Direct Lake is faster than DirectQuery and Direct Lake over Mirrored. Results relative to the Composite Model are mixed because it is slower on queries that can be resolved using the aggregation tables in Composite Model. It is much slower than DirectQuery and the Composite Model at Scale Factor 1000. The likely reasons for this are discussed below in section 9.2; it is also possible that a different data layout in the Delta tables could have improved concurrency specifically for this scenario.

Duration (ms; lower is better)

Pattern	1	2	4	5	6	9	10	12	13	14	19	20	22	23	24
Direct_Query	23,596	1,946	1,622	1,592	8,172	5,029	1,798	1,537	1,684	7,644	5,268	1,541	1,600	1,571	7,814
DB_Composite Model	9,534	147	167	14,397	127	191	2,465	75	1,862	3,300	2,012	1,864	2,048	1,662	2,766
DL_Mirror	5,198	2,776	1,334	958	2,886	348	212	191	198	839	715	229	212	226	952
Direct_Lake	5,277	5,067	3,104	1,970	4,138	328	1,764	309	195	1,046	2,351	180	218	172	911

Table 8.2.2 Average P50 query duration at Scale Factor 10

Pattern	1	2	4	5	6	9	10	12	13	14	19	20	22	23	24
Direct_Query	22,765	2,701	2,077	1,938	6,420	4,262	1,863	1,687	1,788	5,736	4,392	1,796	2,006	1,757	5,443
DB_Composite Model	4,656	88	164	3,551	156	124	1,821	79	1,672	4,949	2,082	1,748	1,826	1,613	2,259
DL_Mirror	20,436	36,571	12,773	12,333	29,984	3,205	4,355	1,135	1,067	6,043	5,060	1,063	460	600	6,389
Direct_Lake	5,628	8,960	5,539	2,495	7,678	306	1,459	285	167	846	2,375	163	223	147	643

Table 8.2.3 Average P50 query duration at Scale Factor 100

Pattern	1	2	4	5	6	9	10	12	13	14	19	20	22	23	24
Direct_Query	11,610	3,578	2,429	3,286	8,158	4,478	1,940	1,842	1,889	6,015	4,612	1,876	2,054	1,766	5,881
DB_Composite Model	5,212	106	161	3,721	179	118	2,002	75	1,749	5,518	2,216	1,826	2,043	1,806	2,555
Direct_Lake	32,041	103,212	59,968	48,878	123,968	32,625	91,201	40,524	31,891	72,345	50,251	52,541	40,807	64,540	116,863

Table 8.2.4 Average P50 query duration at Scale Factor 1000

9. Discussion

9.1 Power BI Query Duration vs Databricks Query History

This white paper compares query performance from the perspective of Power BI query duration. It is worth noting, however, that a difference exists between Power BI query duration and the query duration reported in Databricks Query History and system tables. This difference can be attributed to two factors: the difference between Storage Engine duration and the duration reported by Databricks Query History, and the contribution of other operations on the Power BI side that are required for DirectQuery models.

9.1.1 Power BI Storage Engine Duration vs Databricks Query History

A difference generally exists between Power BI Storage Engine duration and the duration reported in Databricks Query History. Databricks Query History measures the full lifecycle of a query within the Databricks cluster, whereas Power BI Storage Engine duration measures only the time Power BI spends waiting for the DirectQuery source to return raw data rows. The difference between the two can be attributed to network latency and/or overhead introduced by Power BI's Databricks connector/ODBC driver.

The magnitude of this difference varies by scenario. Figure 9.1.1.1 shows Databricks Query History versus Power BI Storage Engine duration for Scale Factor 10, Scenario 3 (Filter on All Slicers), Query 1 (Distinct Count), Warm Cache, Run 3, where the difference between the two is approximately half a second.

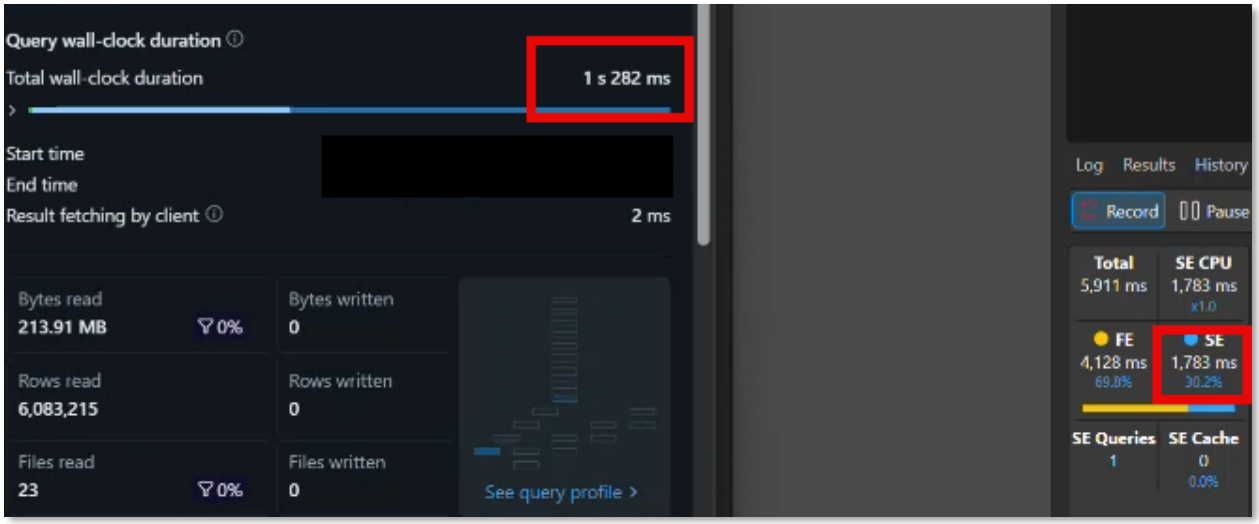


Figure 9.1.1.1 Query History vs Storage Engine Duration for SF10 Scenario 3 Query 1 Warm Cache Run 3

Figure 9.1.1.2 shows the same comparison for Scale Factor 10, Scenario 3 (Filter on All Slicers), Query 4 (simple time intelligence on the second fact table), Hot Cache 0, Run 1, where the difference between the two is approximately 2.7 seconds.

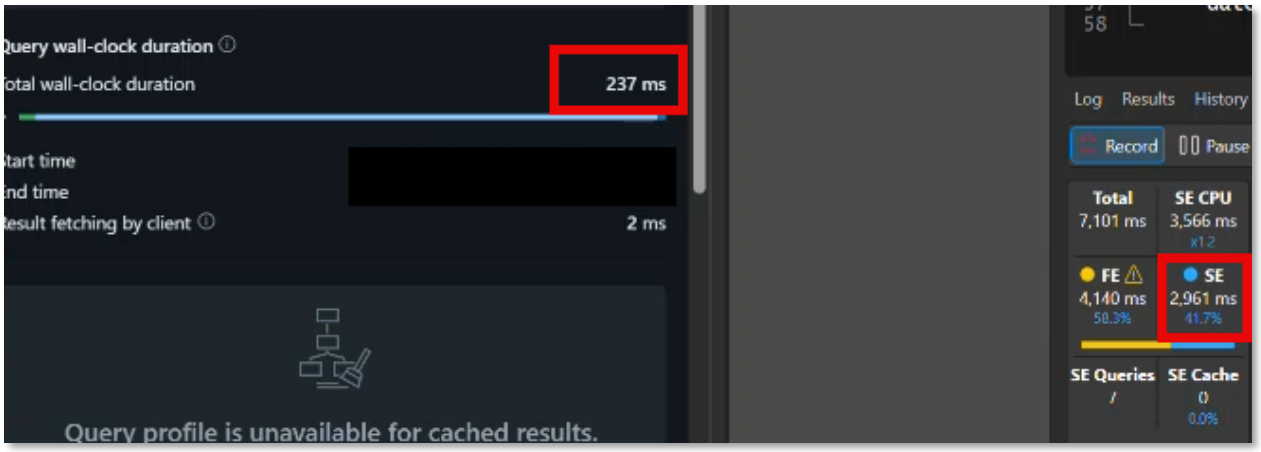


Figure 9.1.1.2 Query History vs Storage Engine Duration for SF10 Scenario 3 Query 4 Hot Cache 0 Run 1

9.1.2 Power BI Formula Engine Duration

The Power BI Formula Engine collaborates with the Storage Engine to plan queries and perform complex calculations. In DAX Studio (the tool used in these tests) the amount of time spent in the Formula Engine for a query is calculated as the total duration of a query minus the amount of time spent in the Storage Engine. In the Cold Cache, Warm Cache, and Hot Cache 0 scenarios, where the Power BI semantic model cache and DirectQuery metadata caches are cleared prior to running the query, this Formula Engine Duration metric includes time spent on additional activities such as rebuilding the DirectQuery metadata cache. Although this work contributes to the Formula Engine Duration metric in DAX Studio, strictly speaking it should be considered a Storage Engine operation. As a result, Formula Engine Duration is higher in these test scenarios than in the Hot Cache scenario and reflects performance that users may only rarely encounter in practice.

As discussed in Section 8.1.4, rebuilding the DirectQuery metadata cache after the Power BI semantic model cache has been cleared added an overhead of approximately 4 seconds. This explains why the Formula Engine timings are so high for scenarios where DirectQuery storage was used and where the Power BI semantic model cache had been cleared.

For example, Figure 9.1.2.1 shows that Formula Engine duration exceeds 7 seconds in the Cold Cache scenario in DirectQuery, and approximately 4 seconds in the Warm Cache and Hot Cache 0 scenarios, for Scale Factor 10, Scenario 1 (No Filter Selected in Slicers). In the Hot Cache scenario, where the Power BI semantic model cache is utilized and DirectQuery metadata caches are already populated, Formula Engine duration is negligible relative to Storage Engine duration. This pattern holds consistently across the remaining data volumes and filter scenarios. Full results comparing DirectQuery Formula Engine duration versus Storage Engine duration across all data volumes and filter scenarios are provided in Appendix 9.

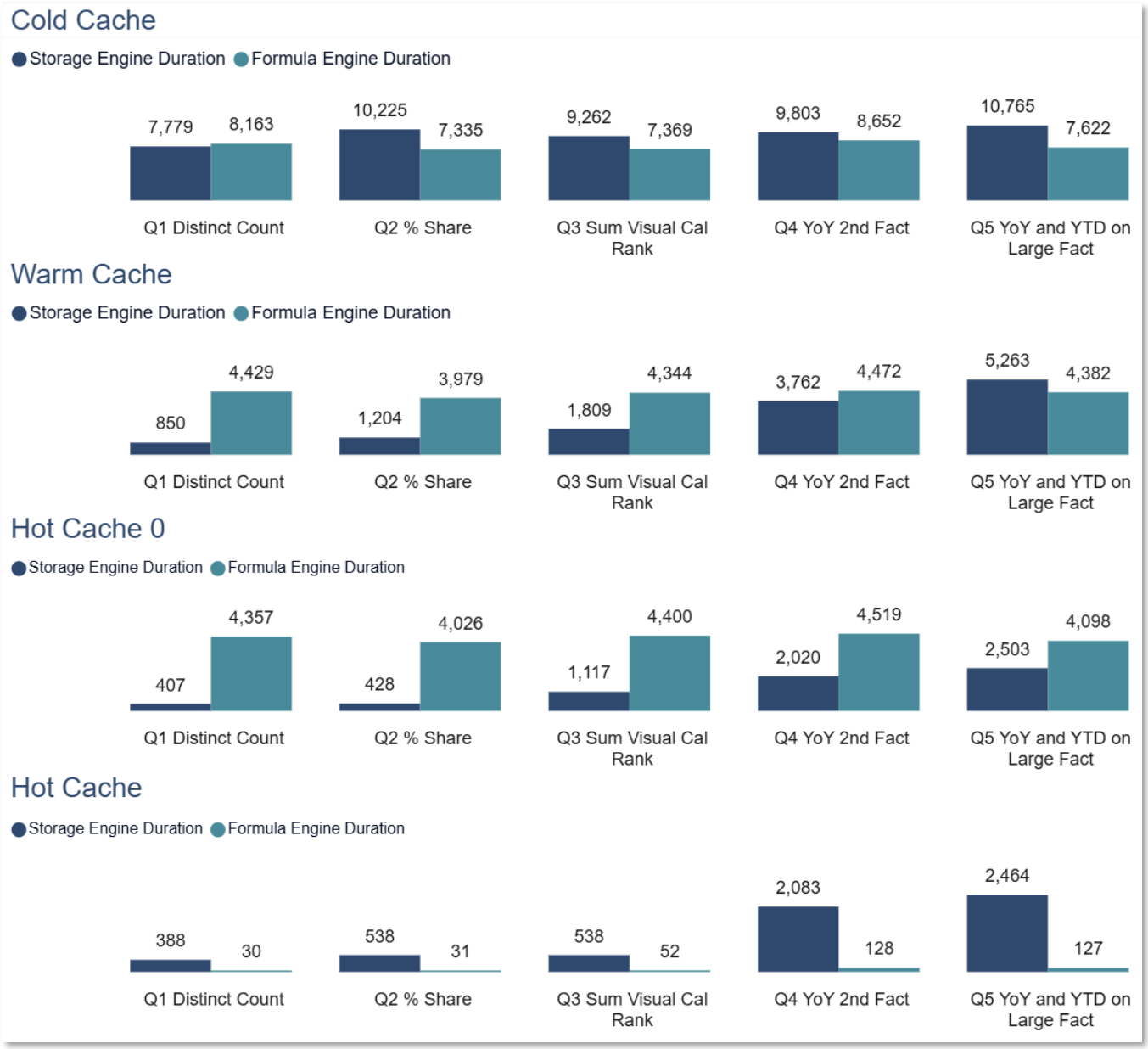


Figure 9.1.2.1 DirectQuery Formula Engine Duration vs Storage Engine Duration for SF10 Scenario 1

9.2 Sizing Consideration

If the performance tests described in this paper had been run using a different Fabric capacity size or Databricks SQL Warehouse cluster size, the results would likely have differed.

F128 was selected for this test based on the size of the Scale Factor 1000 dataset. The largest fact table at Scale Factor 1000 contains 2.6+ billion rows, which exceeds the row count limit for Direct Lake supported by F64 (1.5 billion rows). Because different Fabric capacity sizes may run on different underlying hardware, running this performance test on a different capacity size could yield different results. In addition, Semantic Model Scale-Out is more likely to be used on F256+ capacities and this would have had an impact on the load test results, especially for SF1000.

It is worth noting that capacity size remains an important consideration for pure DirectQuery models. Although no data is stored within Power BI itself, some compute resources are still consumed, and the number of connections available back to the data source varies depending on the capacity size selected.

For Databricks, SQL Warehouse sizes of Medium Serverless (SF10), Large Serverless (SF100), and X-Large Serverless (SF1000) were selected based on the following rationale: the smallest SQL Warehouse size for each scale factor was chosen such that it could resolve the Power BI–generated test queries within 3 seconds using disk cache only (with result caching disabled), as observed in Databricks Query History. A Medium SQL Warehouse was able to resolve SF100 queries within this threshold; however, to maintain three distinct cluster sizes across the three data volumes, a Large SQL Warehouse was instead selected for SF100. Choosing a larger SQL Warehouse size could further improve Storage Engine duration; however, as discussed in Section 9.1 (Power BI Query Duration vs. Databricks Query History), the gap between Storage Engine duration and Databricks Query History, and Formula Engine duration would persist regardless of SQL Warehouse size.

It is also worth noting that during single-user testing and load testing, **Databricks SQL Warehouse sizes were scaled up** as data volume increased, whereas **Fabric compute remained fixed** at the CPU allocation provided by F128 across all data volumes. In a real-world deployment, a larger Fabric capacity size could be selected to achieve more favorable performance at SF1000 and the capacity could be scaled up and down either on a schedule or through automation to respond to load. Both a larger SQL Warehouse size and a larger Fabric capacity (F SKU) would come at a higher cost.

9.3 Data Layout Optimization for Delta Tables in Fabric

9.3.1 Liquid Clustering vs Partitioning + Z-ordering vs Z-ordering only for Fact Tables

Liquid clustered Fact Tables can be created in Fabric Runtime 1.3 and later. With Runtime 2.0 Public Preview for Spark in Fabric, Liquid Clustering is now fully supported for creating and optimizing Fact Tables directly within Fabric. To evaluate the impact of data layout optimization strategy on query performance, liquid clustered Fact Tables were created using the same clustering keys as their Databricks counterparts and compared against Fact Tables using two alternative strategies: partitioned + Z-ordered, and Z-ordered only.

Z-ordered Fact Tables were Z-ordered using the same keys as the Liquid Clustering configuration. Partitioned + Z-ordered Fact Tables were partitioned by the date-based clustering keys (ss_sold_date_sk, cs_sold_date_sk) and Z-ordered by the remaining, non-date clustering keys (ss_addr_sk, cs_bill_addr_sk).

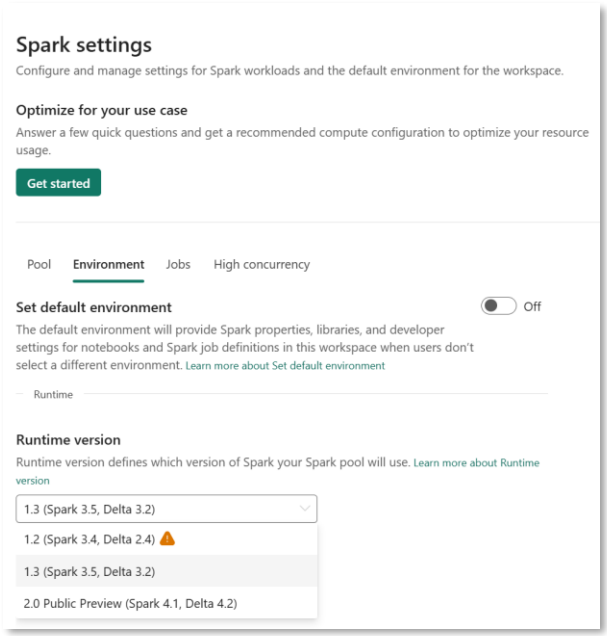


Figure 9.3.1.1 Spark Setting in the Fabric Workspace

Warm cache performance was used as the basis for this comparison. As shown in Figures 9.3.1.2 and 9.3.1.3, at **Scale Factor 1000**, for **Query 5** (time intelligence on the largest fact table) in both Scenario 1 (No Filter Selected in Slicers) and Scenario 3 (Filter on All Slicers), the **Partitioned + Z-Ordered** strategy (middle) outperforms both **Liquid Clustering (left)** and **Z-Ordering Only (right)**, while also consuming less Storage Engine CPU. The difference in Storage Engine CPU and Storage Engine Duration is more pronounced in more heavily filtered scenarios, such as Scenario 3. Based on these findings, Partitioning + Z-Ordering was selected over Liquid Clustering and Z-Ordering Only as the data layout strategy for Fact Tables in Fabric.

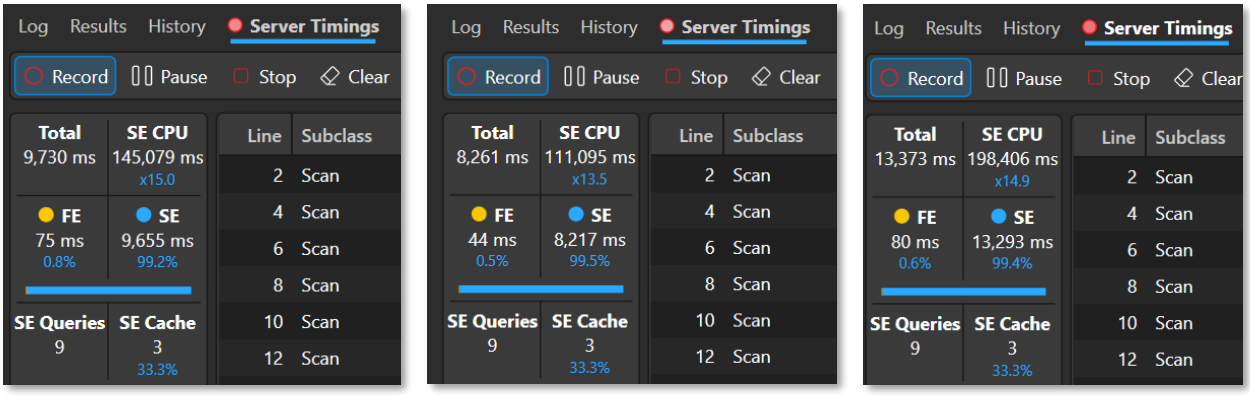


Figure 9.3.1.2 Server Timings traces for SF1000 Warm Cache Scenario 1 Query 5 for Liquid Clustering (left) vs Partitioning + Z-ordering (middle) vs Z-ordering only (right)

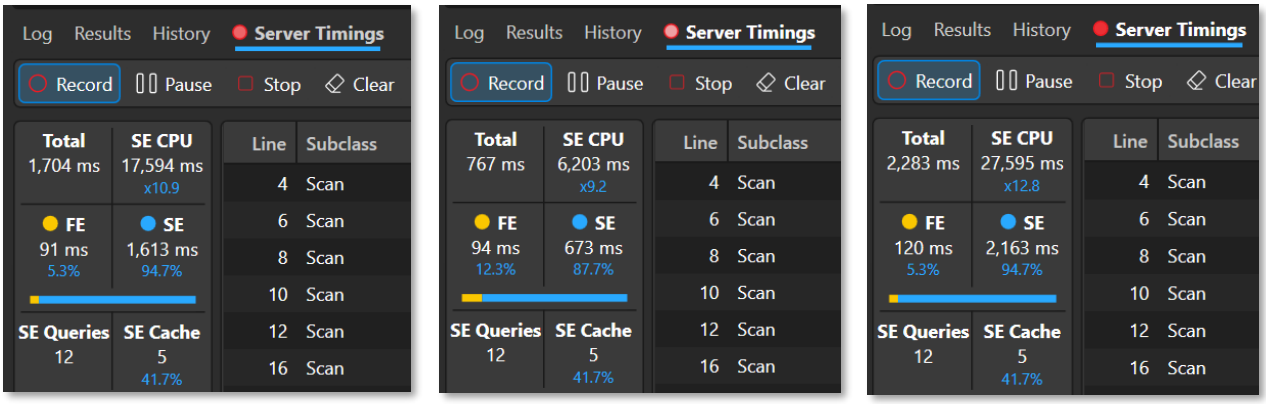


Figure 9.3.1.3 Server Timings traces for SF1000 Warm Cache Scenario 3 Query 5 for Liquid Clustering (left) vs Partitioning + Z-ordering (middle) vs Z-ordering only (right)

One thing to note is that, under the tested configuration, Partitioning + Z-Ordering showed better performance in some scenarios. However, the observed differences were influenced by file and row-group sizing behavior under the specific test configuration and should not be interpreted as a general indication that Partitioning + Z-Ordering is preferable to Liquid Clustering.

9.3.2 Row Group Size for Delta Tables

According to Fabric documentation, Direct Lake prefers row group size between 1 million and 16 millions and the recommendation is generally retain the default row-group size and verify the resulting distribution. Using the Delta Analyzer notebook developed by Phil Seamark from Azure Data CAT team, row group sizes were analyzed for each data layout strategy across data volumes.

Table 9.3.2.1 shows the percentage of row groups falling within the ideal range of 1 million to 16 million rows for the largest fact table. A value of 100% means all row groups are within the recommended range, while 0% means none are. The amended Delta Analyzer notebook, including the code used to calculate the percentage of row groups within the ideal range, is provided in Appendix 10.

Scale Factor	Liquid Clustering	Partitioning + Z-ordering	Z-ordering only
10	100%	0.00%	100%
100	100%	0.00%	96.30%
1000	98.35%	41.96%	94.35%

Table 9.3.2.1 The Percentage of Row Groups within the Ideal Range of 1 million-16 millions for the largest fact

For Partitioning + Z-Ordering, many row groups were smaller than the recommended minimum size of 1 million rows. At Scale Factors 10 and 100, all row groups were below this threshold, and even at Scale Factor 1000, more

than half (58.04%) remained below it. This suggests that the smaller row group sizes are primarily a result of table partitioning rather than the default Delta Lake row group size. As a result, manually lowering the target row group size (for example, to 128 MB) would be unlikely to increase the percentage of row groups that fall within the recommended range. Despite this, Partitioning + Z-Ordering delivered the best query performance among the three strategies.

9.4 Liquid Cluster Key Choice

In a real-world scenario, users would typically rely on automatic Liquid Clustering, which allows Delta Lake to continuously analyze historical query workloads and automatically determine the optimal columns to cluster by. Since this performance test did not generate sufficient historical query volume for Databricks to automatically select clustering keys, the Liquid Clustering keys were instead specified manually during the data generation step. The data generation notebook is provided in Appendix 1.

The Liquid Clustering keys were selected based on Databricks query performance (as observed in Query History) using the test queries generated by Power BI against SF1000, run on an X-Large SQL Warehouse in a warm cache scenario. The clustering key combinations tested are shown in the table below. These combinations were selected based on the most frequently used filters across the Power BI test queries.

Key Combination	store_sales	catalog_sales
1	["ss_sold_date_sk","ss_addr_sk"]	["cs_sold_date_sk","cs_bill_addr_sk"]
2	["ss_addr_sk","ss_cdemo_sk"]	["cs_bill_addr_sk","cs_ship_cdemo_sk"]
3	["ss_sold_date_sk","ss_cdemo_sk"]	["cs_sold_date_sk","cs_ship_cdemo_sk"]

Table 9.4.1 The Liquid Cluster Key Combination Tested

As shown in Tables 9.4.2–9.4.4, Key Combination 1 achieves performance very similar to Key Combination 3, whereas Key Combination 2 performs more slowly. Key Combination 1 was ultimately selected as the Liquid Clustering key configuration for performance testing, as its performance is comparable to, and in some cases slightly faster than Key Combination 3.

Key Combination	Filter Scenario 1	Filter Scenario 2	Filter Scenario 3
1	2.13	2.14	1.26
2	2.41	2.35	1.97
3	2.14	2.19	1.46

Table 9.4.2 The Liquid Cluster Key Combination Performance for Query 2 (seconds)

Key Combination	Filter Scenario 1	Filter Scenario 2	Filter Scenario 3
1	1.23	1.51	1.38
2	1.43	1.90	1.99
3	1.24	1.61	1.63

Table 9.4.3 The Liquid Cluster Key Combination Performance for Query 4 (seconds)

Key Combination	Filter Scenario 1	Filter Scenario 2	Filter Scenario 3
1	2.91	2.54	1.44
2	3.19	2.94	1.96
3	2.87	2.48	1.72

Table 9.4.4 The Liquid Cluster Key Combination Performance for Query 5 (seconds)

10. Conclusion

The results of this benchmark make clear that there is no simple, one-size-fits-all answer to the question of which Power BI storage mode performs best with Azure Databricks and Fabric. Performance is highly context-dependent: performance varies considerably depending on data volume, filter scenario, cache state, and the specific nature of the query being run.

Direct Lake is highly competitive and is the strongest overall performer in both Warm and Hot Cache scenarios – which are, by far, the scenarios that are most commonly encountered by end users in the real world. In fact, across the Warm Cache tests specifically, Direct Lake consistently delivers the strongest overall performance, only being outperformed by the Composite Model at Scale Factor 1000 when no filter is applied, and at all data volumes for queries that can be resolved using the aggregation tables. DirectQuery, by contrast, is generally the slower pattern in warm cache scenarios, though it becomes more competitive and in some cases the fastest pattern at Scale Factor 1000, when Databricks compute was scaled up to an X-Large SQL warehouse and compared against F128, particularly under cold cache and load tests.

Direct Lake over Mirrored Unity Catalog tables, meanwhile, degrades significantly at higher data volumes, becoming the slowest pattern at SF100 and SF1000, with latency at SF1000 exceeding what's acceptable for normal Power BI interactions. It also failed to complete load testing at SF1000 due to exceeding memory limits on F128. The Composite Model with aggregations delivers the fastest and most consistent performance for queries that can be resolved by the aggregation tables, across all data volumes and cache scenarios. However, the performance advantage disappears entirely for queries that fall through to the underlying DirectQuery source (Databricks). Distinct Count stands out as the most challenging query for both Direct Lake and Direct Lake over Mirrored, particularly at SF1000 when no filter is applied in Cold and Warm Cache tests, where DirectQuery and the Composite Model outperform them. Performance optimizations for Direct Lake over Mirrored Unity Catalog tables and distinct count measures are being worked on by the Fabric engineering team at the time of writing and will be released in the latter half of 2026. In particular the optimizations for Direct Lake over Mirrored Unity Catalog will bring performance closer to the other Direct Lake scenarios tested in this paper.

During single-user testing and load testing, Databricks SQL Warehouse sizes were scaled up as data volume increased, whereas Fabric compute remained fixed at the CPU allocation provided by F128 across all data volumes. Selecting a larger SQL warehouse or a larger Fabric capacity comes at a higher cost.

Based on the optimizations applied throughout this benchmark, it can be concluded that, unlike Import mode, where performance tuning typically requires only Power BI expertise, DirectQuery and Direct Lake require expertise in both Power BI and Databricks and Delta Lake to achieve optimal performance.

Given the complexity, readers should not treat any single result in isolation as a definitive verdict but should instead weigh the full pattern of findings against the scenarios most representative of their own workloads. That said, the optimizations applied throughout this benchmark were chosen deliberately and are documented in detail precisely so they can serve as a practical performance optimization starting point for readers building their own solutions. Rather than treating this white paper's results as prescriptive, readers are encouraged to use the methodology, optimization techniques, and testing framework presented here as a foundation for evaluating their own datasets, query patterns, and concurrency requirements, and to validate performance against their specific architecture before making a final platform decision.

11. Appendices

- Appendix 1: Data generation and customization notebooks for Databricks and Fabric.
- Appendix 2: Customized Load Testing script.
- Appendix 3: Test model and report.
- Appendix 4: Test queries used in the single-user tests, generated by Performance Analyzer.
- Appendix 5: Clear-cache notebook used to create a cold cache state for Fabric.
- Appendix 6: Pre-warm and dummy update queries used to create Cold and Warm Cache states for Databricks.
- Appendix 7: Performance Analyzer-generated JSON used for load testing.
- Appendix 8: Single-user test and load test results analysis notebooks, together with the accompanying Power BI metrics model and report.
- Appendix 9: DirectQuery Formula Engine duration versus Storage Engine duration across all data volumes and filter scenarios.
- Appendix 10: The amended Delta Analyzer notebook, including the code used to calculate the percentage of row groups within the ideal range.

Technical Reviewers

This white paper benefited from the technical expertise and thoughtful review of the following contributors.

Name	Role
Bogdan Crivat	Corporate Vice President, Azure Data Analytics
Akshai Mirchandani	Partner Group Software Engineering Manager
Christian Wade	Partner PM Architect
Chris Webb	Principal Program Manager
Philip Seamark	Principal Program Manager
Eiki Sui	Senior Program Manager
David Mitchell	Senior Cloud Solution Architect