REPORT

TPC-DS Benchmark for Power BI/Direct Lake (Microsoft Fabric)

January 2024

David Mariani
CTO, Co-Founder,
AtScale

Albert Zhou
Senior Solutions Engineer

ATSCALE

Introduction

In the rapidly evolving world of data analytics, the need for robust and efficient Business Intelligence (BI) tools is paramount. Power BI, a leader in the BI space, is a subject of interest for many organizations aiming to leverage its capabilities to the fullest. The introduction of Microsoft's Direct Lake architecture on Fabric presents a new frontier in this regard. This report provides an in-depth analysis of the TPC-DS Benchmark for Power BI/Direct Lake. The focus of this benchmark is to critically evaluate Microsoft's claims about Power BI's enhanced capabilities in the context of scalability, cost-effectiveness, and manageability.

This paper is organized into sections in order to deliver a holistic understanding of Power BI's performance on the Direct Lake (Microsoft Fabric) platform. The following sections examine the following:

1. What is Microsoft Fabric and Power BI/Direct Lake?
2. Purpose of Study
3. Benchmarking Testing Methodology
4. Test Results for Power BI/Direct Lake
5. Comparative Test Results for Power BI/Direct Lake versus AtScale on Snowflake
6. Power BI/Direct Lake Considerations
7. Conclusion

What Is Microsoft Fabric And Power BI/Direct Lake?

According to Microsoft, “Microsoft Fabric is an all-in-one analytics solution for enterprises that covers everything from data movement to data science, Real-Time Analytics, and business intelligence. It offers a comprehensive suite of services, including data lake, data engineering, and data integration, all in one place.”

Power BI/Direct Lake is a feature within Microsoft's Power BI suite that integrates Power BI into Microsoft Fabric’s OneLake lakehouse platform. With Power BI/Direct Lake,

customers can create “semantic models” (datasets) directly from files within the lakehouse to avoid the need for importing or copying data into Power BI for analysis.

In this paper, we will utilize the TPC-DS industry standard benchmark to evaluate the suitability for deploying Power BI/Direct Lake to deliver fast, scalable enterprise business intelligence.

Purpose Of Study

Why This Benchmark?

The benchmark was initiated to test Microsoft's assertion that its Direct Lake architecture on Fabric is ideally suited for enabling Power BI to offer a direct query experience without the need for imports. This claim, if validated, positions Power BI as a more agile and efficient tool in the BI space, especially for organizations dealing with large-scale data.

Key Questions to be Answered

The benchmark focused on three key considerations:

1. **Is Power BI/Direct Lake a suitable alternative to Import Mode and DirectQuery for delivering performant queries at a reasonable enterprise data scale?**

   Before Microsoft Fabric and Direct Lake, Power BI offered two distinct methods for querying data: Import Mode and DirectQuery.

   In Import Mode, data is imported and stored within Power BI, allowing for high-speed analytics and visualizations, but with limitations on data volume and the need for regular data refreshes.

   DirectQuery, on the other hand, leaves the data in its original source, querying it directly for each report interaction. This method offers real-time insights and avoids data duplication but can result in slower performance due to the reliance on the source system's response time.

   Microsoft’s solution to delivering the best of both worlds, Power BI/Direct Lake, is marketed as the “perfect” solution that can deliver both fast and real-time performance without the need to import data into Power BI, as illustrated in Microsoft’s diagram below:

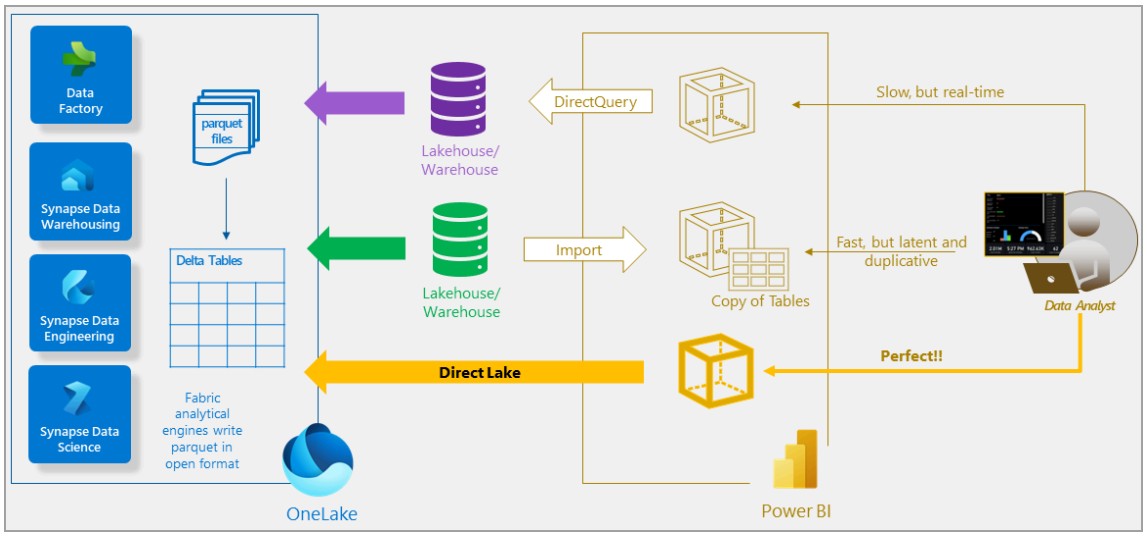


Illustration #1: Power BI/Direct Lake (source)

In this paper, we will set out to test this assertion that Power BI/Direct Lake (Fabric) can deliver fast queries without the limits of Import Mode and DirectQuery.

2. Is Power BI/Direct Lake financially viable for organizations of various sizes?

Microsoft Fabric (and Power BI/Direct Lake) have a similar cost profile to Power BI Premium's capacity pricing model, Microsoft's enterprise-grade offering for Power BI. The cost of Microsoft Fabric is determined by the amount of dedicated cloud resources allocated to an organization, such as CPU, memory, and storage. This means that pricing is based on the scale and performance needs of the organization or use case, rather than the number of individual users.

This model can be cost-effective for large organizations with many users, as it allows unlimited access to Power BI's full suite of tools and services within the purchased capacity. However, for smaller organizations or those with fewer users, the cost can be higher on a per-user basis compared to other Power BI offerings.

Understanding the specific data and analytics needs of an organization is crucial in evaluating whether Power BI Premium's cost profile aligns with its value proposition. We conducted performance and scalability tests to determine the limits and scalability factors that drive sizing and costs.

3. Is Microsoft Fabric and Power BI/Direct Lake a viable choice as a lakehouse platform?

A lakehouse architecture is a relatively new data management strategy that combines the benefits of data lakes and data warehouses, and aims to provide a

unified platform for both large-scale data storage and advanced analytics. In a lakehouse, data is stored in a raw, unstructured form similar to a data lake, allowing for high scalability and flexibility in handling diverse data types and is particularly beneficial for organizations dealing with massive volumes of data from a variety of data sources.

The key advantage of a lakehouse architecture is its ability to support both machine learning and BI workloads. It enables advanced analytics and AI directly on the raw data without the need for separate data silos or extensive ETL (Extract, Transform, Load) processes typically associated with traditional data warehouses.

Many cloud-based data platform vendors, including Databricks and Snowflake, offer tools and services to make a lakehouse architecture work for larger enterprises. Although not the intent of this paper, we will comment on Microsoft Fabric's suitability as a generalized lakehouse platform.

Key Findings

As you will see in the data below, the results of the benchmark reveal some critical insights that customers need to consider when evaluating whether Power BI/Direct Lake is a suitable solution for their use cases. In our analysis, we conclude the following:

1. **Direct Lake serves as a "lazy load" Import Mode alternative with all the same drawbacks and limitations as Import Mode**
2. **Direct Lake is fast on very small data but stumbles with larger data and higher user concurrency**
3. **Upon a data or model refresh, Direct Lake creates a substantial "cold cache" performance hit**
4. **Power BI Web is missing critical modeling features and is poorly suited for multi-user collaboration**
5. **Fabric Lakehouse is not suitable for serving as a general purpose Lakehouse due to its lack of basic data management features**

We will dive into each of these conclusions with specifics and supporting data in the following sections.

Testing Methodology

Why TPC-DS Benchmark?

The TPC-DS (Transaction Processing Performance Council Decision Support) benchmark is a widely recognized standard for evaluating the performance of data processing systems, particularly in the context of decision support systems, which include database warehouses and big data systems. Designed by the Transaction Processing Performance Council, the TPC-DS benchmark simulates a decision support system that encompasses several real-world scenarios, such as customer relationship management, inventory, sales, and product returns.

One of the key features of the TPC-DS benchmark is its comprehensive set of queries and data schema that simulates complex and realistic business analytics tasks while testing the system's ability to process large volumes of data, execute queries of varying complexity, and handle concurrent data loads and queries. The benchmark measures performance in terms of query throughput and execution speed, providing a standardized metric to compare the effectiveness of different hardware and software configurations in handling decision support workloads.

The TPC-DS is highly regarded for its thoroughness and relevance to real-world business data operations, making it a critical benchmarking tool for organizations looking to evaluate and optimize their data processing capabilities.

TPC-DS Data Schema

The table below lists the TPC-DS generated tables and their number of rows used for the benchmark. In order to evaluate how Power BI/Direct Lake behaved across different data sizes, we used three data scales for the test: 100GB, 1TB and 10TB.

TPC-DS TABLE NAME	# OF ROWS (100GB)	# OF ROWS (1TB)	# OF ROWS (10TB)
CALL_CENTER	30	42	54
CATALOG_PAGE	20,400	30,000	40,000
CATALOG_RETURNS	14,404,374	143,996,756	1,440,033,112

CATALOG_SALES	143,997,065	1,439,980,416	14,399,964,710
CUSTOMER	2,000,000	12,000,000	65,000,000
CUSTOMER_ADDRESS	1,000,000	6,000,000	32,500,000
CUSTOMER_DEMOGRAPHICS	1,920,800	1,920,800	1,920,800
DATE_DIM	73,049	73,049	73,049
HOUSEHOLD_DEMOGRAPHICS	7,200	7,200	7,200
INCOME_BAND	20	20	20
INVENTORY	399,330,000	783,000,000	1,311,525,000
ITEM	204,000	300,000	402,000
PROMOTION	1,000	1,500	2,000
REASON	55	65	70 §
SHIP_MODE	20	20	20
STORE	402	1,002	1,500
STORE_RETURNS	28,795,080	287,999,764	2,879,898,629
STORE_SALES	287,997,024	2,879,987,999	28,800,239,865
TIME_DIM	86,400	86,400	86,400
WAREHOUSE	15	20	25
WEB_PAGE	2,040	3,000	4,002
WEB_RETURNS	7,197,670	71,997,522	720,020,485
WEB_SALES	72,001,237	720,000,376	7,199,963,324
WEB_SITE	24	54	78

Illustration #2: TPC-DS Tables (source)

TPC-DS Queries Used

The TPC-DS benchmark toolset generates a diverse set of complex queries, simulating real-world decision support systems. For the benchmark testing, we selected a representative set of 20 queries from the 99 TPC-DS generated query set to keep the run time and costs of running the benchmarks within reason, without having to downsize data size. The queries were chosen in no particular order and were selected to eliminate redundancy and to ensure the usage of most tables.

SPECIAL NOTE:
We had to exclude queries 48, 50, 71 and 88 from the test because Power BI Direct Lake currently lacks support for Calculated Columns which is required for these queries. The absence of Calculated Column support also required creating bespoke DAX calculations for most of the TPC-DS queries since many of the metrics required specific filtering.

The following 20 TPC-DS queries were selected for the test. The queries highlighted in red were excluded from the test due to the inability to model them in Power BI/Direct Lake because of the lack of Calculated Columns support.

TPC-DS QUERY NUMBER	TPC-DS QUERY DESCRIPTION
2	Report the ratios of weekly web and catalog sales increases from one year to the next year for each week. That is, compute the increase of Monday, Tuesday, ... Sunday sales from one year to the following.
7	Compute the average quantity, list price, discount, and sales price for promotional items sold in stores where the promotion is not offered by mail or a special event. Restrict the results to a specific gender, marital and educational status.
13	Calculate the average sales quantity, average sales price, average wholesale cost, total wholesale cost for store sales of different customer types (e.g., based on marital status, education status) including their household demographics, sales price and different combinations of state and sales profit for a given year.
15	Report the total catalog sales for customers in selected geographical regions or who made large purchases for a given year and quarter.

26	Computes the average quantity, list price, discount, sales price for promotional items sold through the catalog channel where the promotion was not offered by mail or in an event for given gender, marital status and educational status.
31	List counties where the percentage growth in web sales is consistently higher compared to the percentage growth in store sales in the first three consecutive quarters for a given year.
33	What is the monthly sales figure based on extended price for a specific month in a specific year, for manufacturers in a specific category in a given time zone. Group sales by manufacturer identifier and sort output by sales amount, by channel, and give total sales.
42	For each item and a specific year and month calculate the sum of the extended sales price of store transactions.
48	Calculate the total sales by different types of customers (e.g., based on marital status, education status), sales price and different combinations of state and sales profit.
50	For each store count the number of items in a specified month that were returned after 30, 60, 90, 120 and more than 120 days from the day of purchase.
52	Report the total of extended sales price for all items of a specific brand in a specific year and month.
53	Find the ID, quarterly sales and yearly sales of those manufacturers who produce items with specific characteristics and whose average monthly sales are larger than 10% of their monthly sales.
55	For a given year, month and store manager calculate the total store sales of any combination of all brands.
56	Compute the monthly sales amount for a specific month in a specific year, for items with three specific colors across all sales channels. Only consider sales of customers residing in a specific time zone. Group sales by item and sort output by sales amount.
60	What is the monthly sales amount for a specific month in a specific year, for items in a specific category, purchased by customers residing in a specific time zone. Group sales by item and sort output by sales amount.
61	Find the ratio of items sold with and without promotions in a given month and year. Only items in certain categories sold to customers living in a specific time zone are considered.
71	Select the top revenue generating products, sold during breakfast or dinner time for one month managed by a given manager across all three sales channels.
88	How many items do we sell between specific times of a day in certain stores to customers with one dependent count and two or less vehicles registered or two dependents with four or fewer vehicles registered or three dependents and five or less vehicles registered. In one row break the counts into sales from 8:30 to 9, 9 to 9:30, 9:30 to 10 ... 12 to 12:30
96	Compute a count of sales from a named store to customers with a given number of dependents made in a specified half-hour period of the day.

98	Report on items sold in a given 30-day period, belonging to the specified category.

Illustration #3: TPC-DS Queries Selected for the Test (source)

Data and Model Preparation Procedure

The following section breaks out the steps we took to create the testing framework.

1. **Generate TPC-DS Data, Load in Lakehouse & Create Direct Lake Tables**

   The procedure for generating and loading TPC-DS data for the Power BI/Direct Lake benchmark begins by running the TPC-DS benchmark tool "dsdgen" to create the TPC-DS raw CSV files for three distinct scale factors: 100GB, 1TB, and 10TB. This range of data sizes ensures a comprehensive evaluation of the performance and scalability across varying levels of data intensity.

   We encountered the following difficulties when loading data to Fabric:

   1. As of this writing, Fabric Lakehouses do not support schemas for lakehouse tables. Without the ability to organize tables into separate schemas, it was necessary to create three separate Fabric Lakehouses for each data size (100GB, 1TB, 10TB). This drastically complicated the rest of our benchmarking tasks.
   2. Once generated, we loaded the raw TPC-DS files into the Fabric Lakehouse using one of two methods. For smaller files, we used the built-in Fabric Lakehouse data uploader, but larger files (i.e. store_sales, catalog_sales, web_sales) failed with a "too many blocks error". As a result, for larger files, we had to employ a two-step process that used the Azure azcopy utility to first upload the larger files into an ADLS directory after which we then created Fabric Lakehouse Shortcuts to point to those files.

   Once the raw data files were loaded into the Lakehouse, we created a Fabric Lakehouse notebook to transform the raw TPC-DS files into Lakehouse tables in a parquet format with the proprietary option of "v-order" enabled, using the following function:

Python

```
def loadData(folder, tableName):

 # Create the Dataframe

 df =
spark.read.format('csv').options(header=True).options(sep='|').options(inferSchema=T
rue).load("Files/" + folder + "/" + tableName + ".csv")


 # Write to vordered delta

 delta_table_path = "Tables/" + tableName

 spark.conf.set("spark.sql.parquet.vorder.enabled", "true")

 df.write.format("delta").mode("overwrite").option("overwriteSchema",
True).save(delta_table_path)


 # Count the number of rows

 row_count = df.count()

 diff = row_count - dic_tpcds_rows[tableName][rows_index]

 new_row = {'table': tableName, 'numrows': row_count, 'checksum': diff}

 df_stats.loc[len(df_stats)] = new_row
```

Illustration #4: Python code for creating Direct Lake tables

2. Create TPC-DS Semantic Model in Power BI Web

For modeling the TPC-DS schema in the Power BI/Direct Lake benchmark, we had to use the Fabric Lakehouse Semantic Model web editor. As of this writing, it is not possible to use Power BI Desktop to author semantic models in Power BI/Direct Lake. We set out to create a semantic model that would accommodate the 20 target TPC-DS generated queries we chose for the test.

However, we encountered the following difficulties:

1. We could not model four of the selected queries (Q48, Q50, Q71, Q88) because Calculated Columns are currently not supported in Power BI/Direct Lake web modeling experience and these queries require data to be returned as columns in the result set. As a result, we had to remove and disqualify these queries from the test.

2. We had to manually create three distinct semantic models for each data size (100GB, 1TB, 10TB) due to the following:
    a. Fabric Lakehouses do not currently support schemas (2 or 3 part namespaces) as mentioned in Step 1 so we had to create a distinct Fabric Lakehouse for each data size.
    b. Power BI/Direct Lake does not currently allow semantic models to be shared across Fabric Lakehouses so we had to manually create a separate semantic model for each of the three Lakehouses.

After the completion of this step, we had three distinct Lakehouses with three distinct Lakehouse models as you see in the image below:

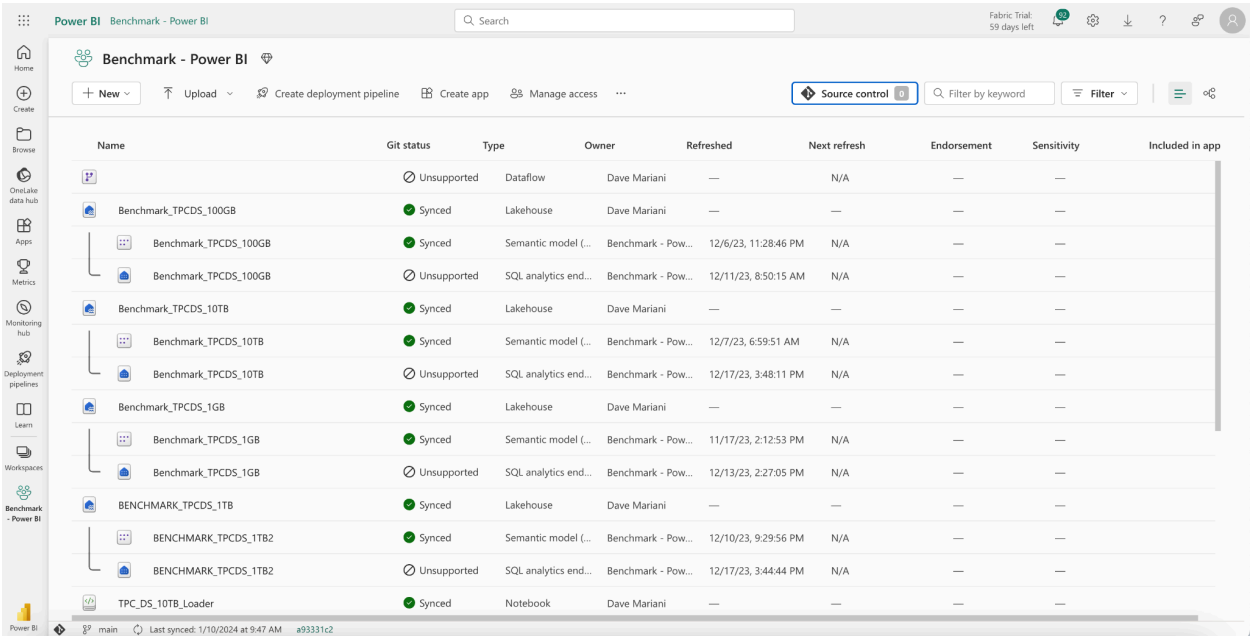


Illustration #5: Direct Lake Benchmark Workspace

3. Create Power BI Report and Reconcile Query Results

In this step, we re-created the 16 TPC-DS generated SQL queries in a Power BI report so we could extract each of the DAX queries for our JMeter suite. Using a 1GB version of the semantic model we created above, we defined a table visual for each of the TPC-DS queries in a Power BI Web workbook, one sheet for each of the modeled 16 queries.

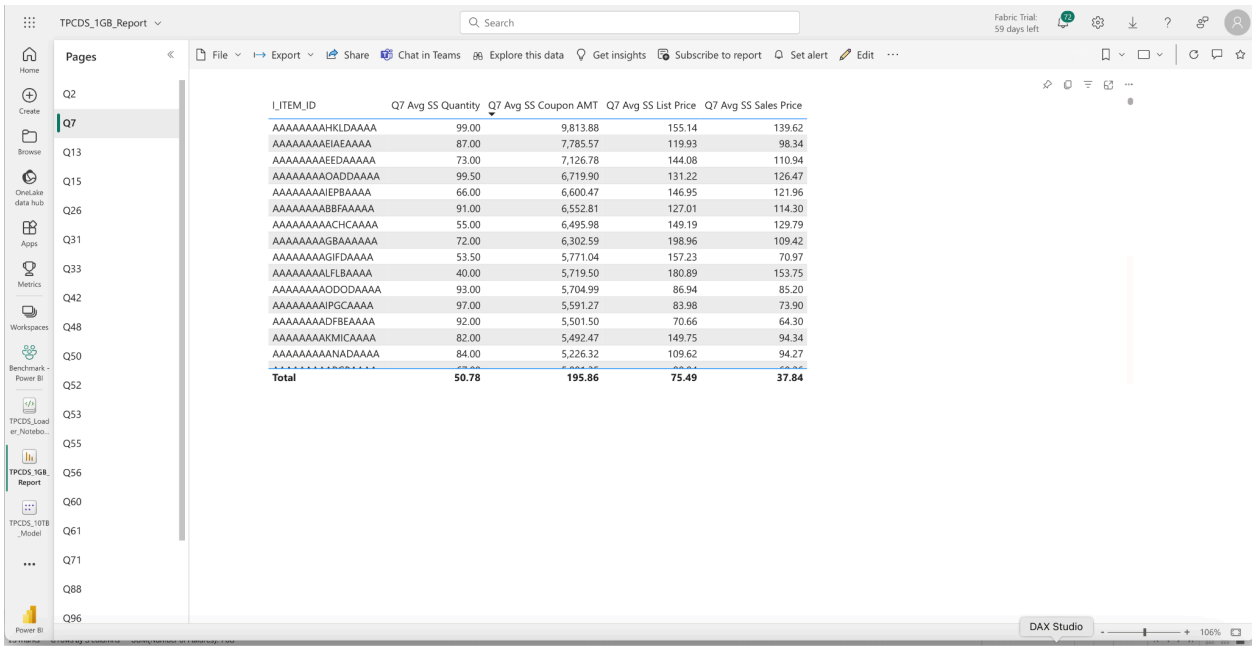


Illustration #6: Power BI Workbook for each TPC-DS Query

We then reconciled each query result against the original TPC-DS SQL queries from our Snowflake data warehouse. This reconciliation process is key to ensuring the accuracy and reliability of the Power BI model.

Once data was reconciled, we then downloaded the Power BI Web report as a PBIX file and opened it in Power BI Desktop, using the Power BI Performance Analyzer to capture the DAX queries for each of the 16 queries.

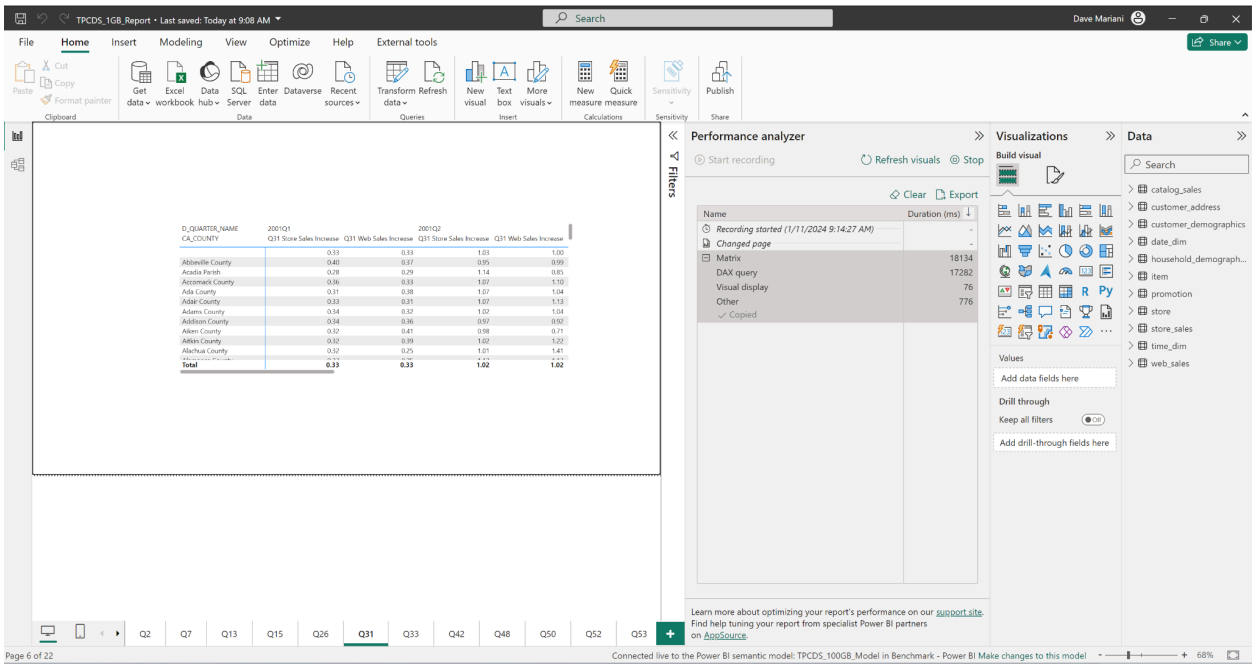


Illustration #7: Power BI Performance Analyzer

We used this Power BI helper to capture each worksheet's DAX query which we then "stringified" using this tool in order to make it suitable as a payload in the Power BI REST API, executeQueries.

4. Create JMeter Test Suite

For simulating user concurrency and to measure performance and latency in a controlled testing environment, we chose to use the open-source tool Apache JMeter to define, execute, and capture the results of our test.

Using the stringified and escaped DAX queries we captured above, we created a JMeter JMX (Java Management Extensions) test file that calls the Power BI executeQueries REST API to run each of the 16 queries in 4 concurrency test scenarios:

1. 1 user/thread (16 queries total)
2. 5 concurrent users/threads (80 total queries)
3. 25 concurrent users/threads (400 total queries)
4. 50 concurrent users/threads (800 total queries)

Additionally, we added a "Constant Throughput Timer" to our JMeter test file (for Power BI only) in order to keep query requests under the 120 queries per minute threshold that the executeQueries API enforces. After several attempts, we needed

to throttle our queries to 60 per minute to eliminate throttling errors.

> **SPECIAL NOTE:**
> The Power BI REST API enforces a maximum rate of 120 requests per minute. To avoid throttling errors in our test, we had to throttle the Power BI queries in JMeter to 60 queries per minute. Since query input is throttled, this may have given Power BI an unfair advantage under the concurrency tests.

5. **Run JMeter Test Suite and Capture Results**

   To run the JMeter suite, we needed to first generate and capture a user authentication token. As of this writing, Power BI REST API does not support service principle tokens. You can generate and capture the token using the executeQueries API test tool. For our test, we extended the one hour token expiration using the steps documented here.

   Using the command line version of JMeter, we ran each test scenario for all three data sizes (100GB, 1TB, 10TB) and captured the results using JMeter's CSV file output option, and loaded the data into a database for reporting purposes. Using Tableau, we built each of the visualizations below.

> **SPECIAL NOTE:**
> We only throttled queries (1 per second) for Power BI/Direct Lake and not AtScale on Snowflake. By doing so, we gave Power BI/Direct Lake an advantage over AtScale on Snowflake by effectively limiting concurrency load for Power BI/Direct Lake to one query per second maximum.

Test Configurations

Power BI can be purchased using a reserved capacity pricing option or a "pay as you go" demand-based pricing option which allows the pausing of the capacity to avoid ongoing charges. Power BI Premium and Microsoft Fabric offer a number of pricing tiers that are based on different levels of capacity, with higher tiers offering more computing capacity,

higher table row limits and memory. For Power BI/Direct Lake, we tested using the trial capacity level offered during the preview period. The trial capacity (FT1) equates to a Power BI Premium capacity level of P1 or a Microsoft Fabric capacity level of F64 which includes 64 capacity units (CUs) and 25GBs or RAM.

Snowflake also charges on a capacity basis using data warehouse capacity sizes (i.e. Small, Medium, Large, etc.) that equate to an allocation of compute and memory capacity, counted as "credits". Snowflake clusters can be configured to automatically pause when not being used to avoid further charges. For the Snowflake and AtScale on Snowflake scenarios, warehouse capacity sizes were chosen for each respective data size.

For this benchmark, we used the on-demand pricing option for Power BI capacity to match the same on-demand pricing model for Snowflake data warehouses. The table below shows the capacity and Snowflake data warehouse sizes used for each platform:

Vendor	Data Size	Configuration	Compute Cost per Hour (min)
Power BI/Fabric	All	F64/FT1/P1	$12.98[1] ($.22/min)
	TBD[2]	F256/P3	$51.91 ($.87/min)
AtScale on Snowflake	100GB	Medium (4 credits/hour)	$8.00 ($.13/min)
	1TB/10TB	1X-Large (16 credits/hour)	$32.00 ($.53/min)

Illustration #8: Data Platform Capacity Sizes Used for the Test

SPECIAL NOTE:

For the Power BI/Direct Lake tests, as noted, we used the F64/FT1/P1 trial capacity level which limits memory to 25GB. We would have preferred to use the F256/P3 capacity instead which has 100GB of RAM and higher row limits. However, we were unable to capture results on this capacity level due to the following error: "You have reached the maximum allowable memory allocation for your tier. Consider upgrading to a tier with more available memory." As of this writing, we have an open support ticket with Microsoft to resolve this issue. Upon

resolution, we will update our Power BI benchmark results using this larger capacity level.

For the 10TB data size level that we were able to complete on the F256/P3 capacity, we found that the F256/P3 capacity results were 25% better than the F64/FT1/P1 capacity results but with the same high degree of query failures and the same fallback behavior to DirectQuery that we saw in the F64/FT1/P1 trial capacity level.

Cost Computations

We calculated the compute costs for each platform by multiplying the cumulative end-to-end run time for each query as reported by JMeter for the concurrency test by the cluster compute cost per hour like so:

SequentialRunTimeMinutes / 60 * ComputeCostPerHour

The method for measuring compute time is different in this benchmark as compared to prior AtScale benchmarks. Specifically, each query's run time was summed as if the queries were run sequentially, not concurrently. In contrast, prior AtScale benchmarks measured concurrent run times as simply: end time - start time for the entire suite. This change was necessary because we had to normalize the effect of Power BI's query throttling, even though throttling Power BI and not throttling Snowflake may have benefited Power BI's results comparatively.

We explicitly excluded storage costs from our calculations. We found that storage cost was nominal across all platforms and given that it's a fixed cost, it was not subject to variation in our testing scenarios.

Test Results - Power BI

Query Performance

We found that Power BI/Direct Lake does well with small data but struggles with larger data and concurrency, resulting in query timeouts. As you can see in the chart below, Power BI/Direct Lake performs well for the 100GB data size, but performance and query timeouts increase dramatically for larger data sizes (1TB, 10TB).

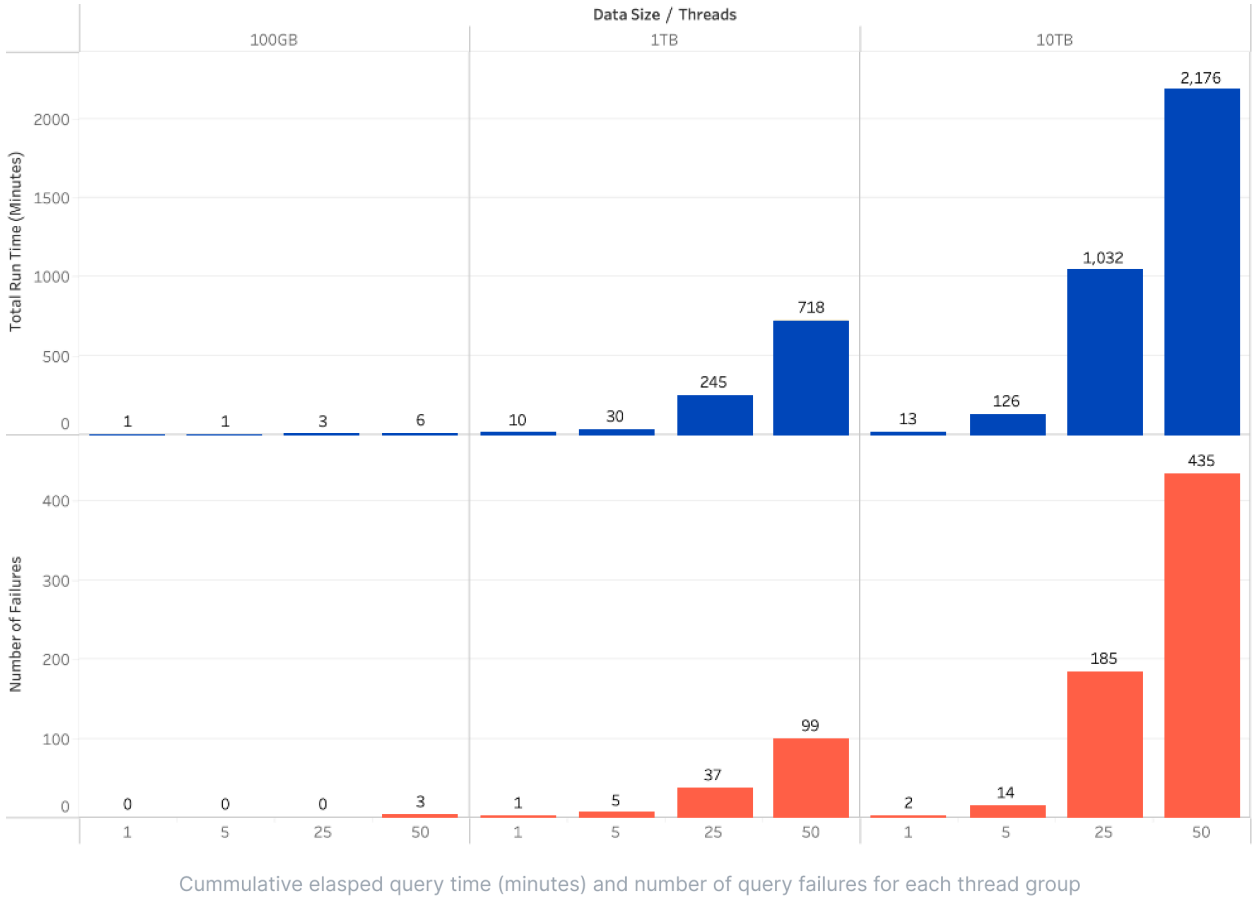
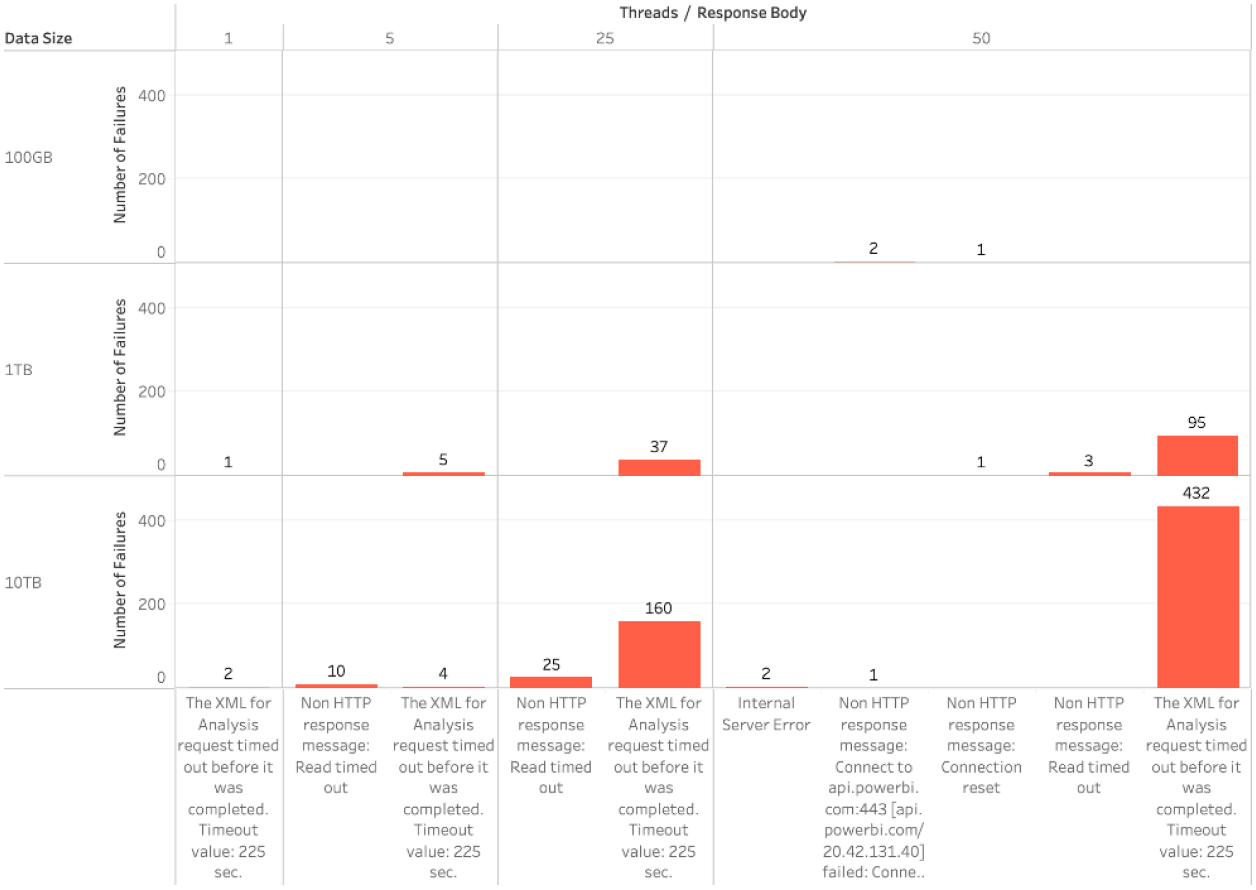


Illustration #9: Power BI/Direct Lake Query Performance by Thread Group

Upon investigation, we found that the 1TB and 10TB data sizes triggered Power BI/Direct Lake's "*fallback*" behavior to DirectQuery due to data either not fitting in memory or tables exceeding the row limits for the capacity (see Illustration #9 for threshold limits). While the "fallback" to DirectQuery mode prevented queries from failing outright, customers should be concerned that query performance may be inconsistent or erratic depending on data and query profiles. See the "Fallback" section in this Microsoft article for more information.

Query Failures and Their Nature

As you can see in Illustration #9, we experienced a high degree of query failures as data size and user concurrency increased. Since we were using the Power BI REST API, we wanted to ensure that the failures were not due to REST API scalability limitations, since we intended to measure Power BI/Direct Lake engine performance, not API performance. As you can see in Illustration #10, the majority of query failures were due to query timeouts with the following message: **"The XML for Analysis request timed out before it was completed. Timeout value: 225 sec."**



Cummulative elasped query time (minutes) and number of query failures for each thread group

Illustration #10: Power BI/Direct Lake Query Failures by Type

As of this writing, there is no way of increasing the query timeouts for Power BI/Direct Lake via the Power BI REST API. However, a query run time of 3 minutes and 45 seconds is not acceptable for interactive analysis, and increasing timeouts would have negatively impacted Power BI/Direct Lake results even more.

The “Cold Cache” Effect

When semantic models are refreshed, Power BI/Direct Lake shows a noticeable performance hit as data is loaded in memory. This “cold cache” effect is most acute when data is small enough to fit in memory. As you can see in Illustration #11, when queries fallback to DirectQuery mode due to data size or row limits, the “cold cache” effect is less predictable.

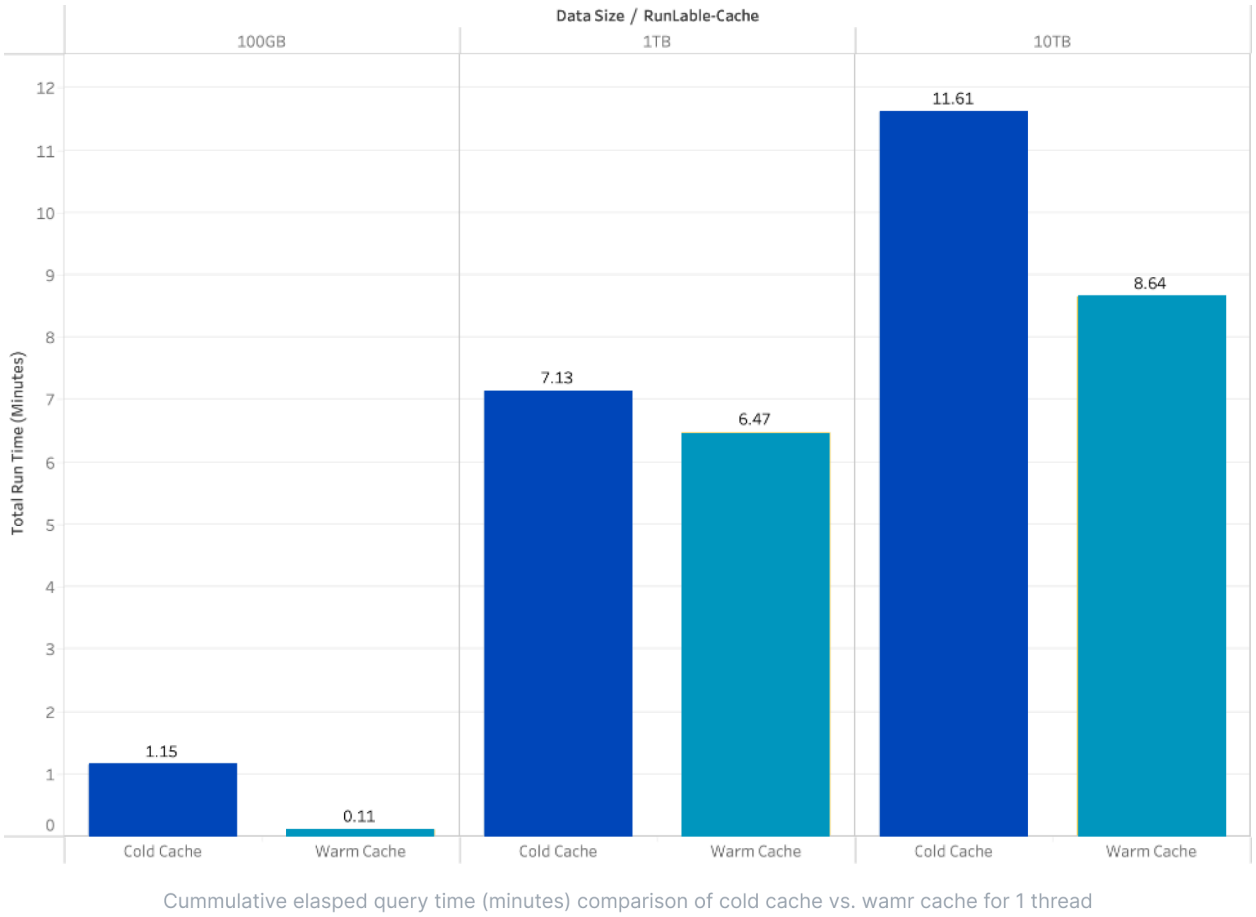


Illustration #11: Power BI/Direct Lake Cold Cache Effect

Customers using Power BI/Direct Lake need to consider cache warming strategies to mitigate this “lazy load” behavior to avoid a poor end-user experience after a model or data refresh.

Test Results - Comparison with AtScale on Snowflake

There are alternatives for providing users with a performant, direct query connection in Power BI without using Power BI/Direct Lake.

AtScale's semantic layer platform provides a BI tool agnostic option for connecting Power BI, Excel, Tableau, Looker, and many other tools to cloud data platforms like Snowflake, Databricks, Google BigQuery, Amazon Redshift, and more. With the AtScale semantic layer platform, there is no requirement to move data out of your chosen data platform nor subject Power BI users to a degraded Power BI experience that comes with a DirectQuery interface.

In this section, we will compare the TPC-DS results of Power BI/Direct Lake with AtScale on Snowflake.

Query Performance Comparison

As you can see in the charts below, Power BI/Direct Lake is faster than AtScale on Snowflake for the 100GB data size. However, Power BI/Direct Lake is substantially slower with query timeouts increasing with data size and concurrency. Furthermore, AtScale on Snowflake shows a modest, non-linear performance impact as data size increases by 10x and even 100x.

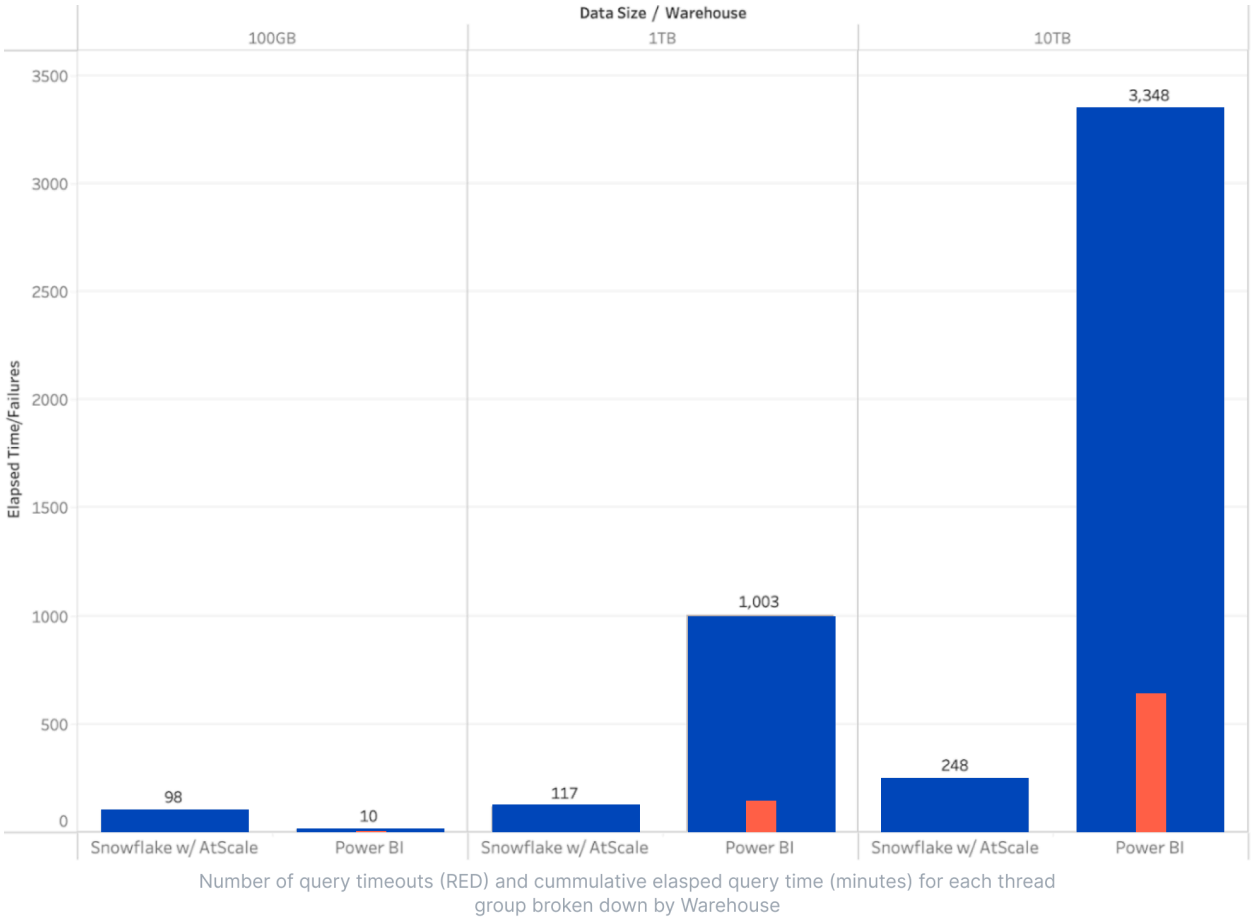


Illustration #12: Query Performance Comparison of Power BI/Direct Lake and AtScale on Snowflake

Compute Cost Comparison

As you can see in Illustration #13, Power BI/Direct Lake is substantially cheaper at the 100GB data size but much more expensive at larger data sizes and concurrencies. Note that cost statistics for Power BI/Direct Lake are incomplete due to the high degree of query timeouts at larger data sizes and concurrencies. If queries were left to complete on their own, costs and run times would likely have increased even more.

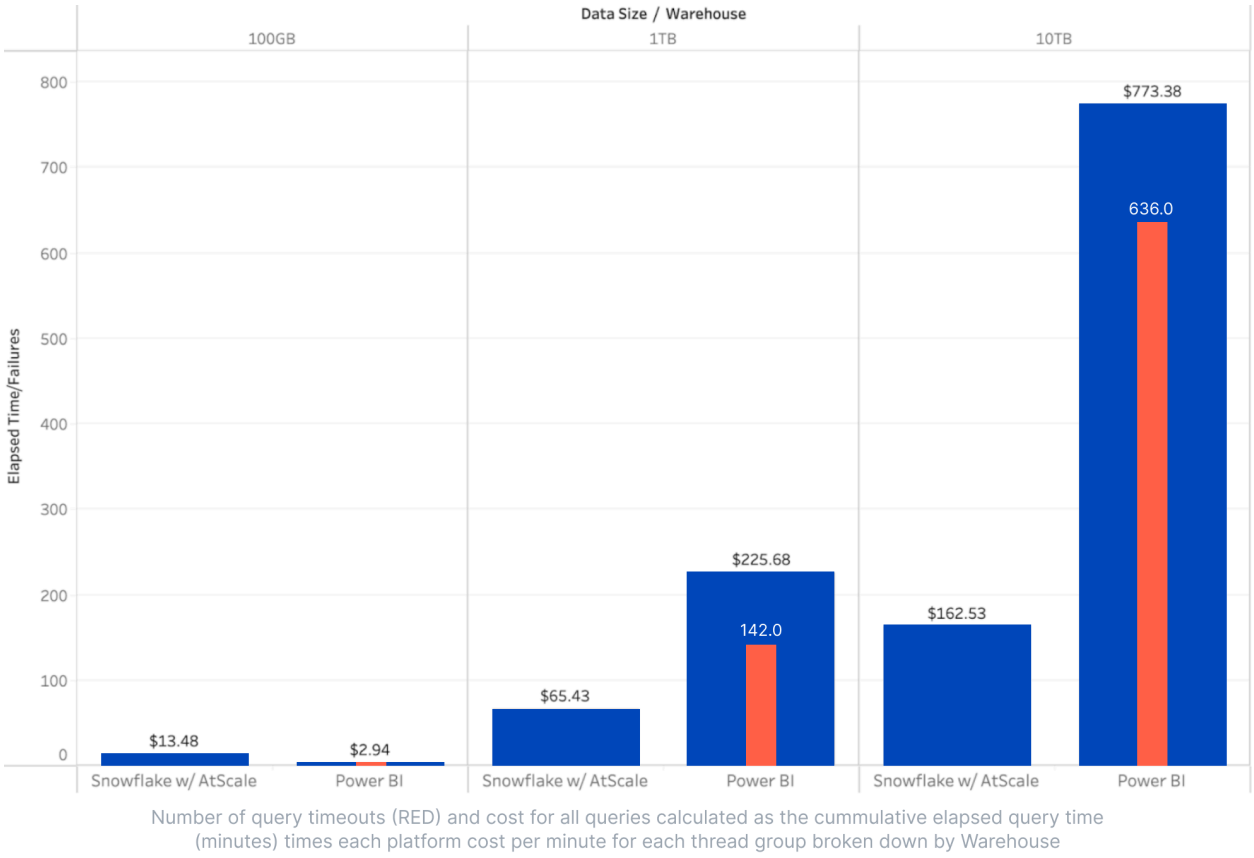


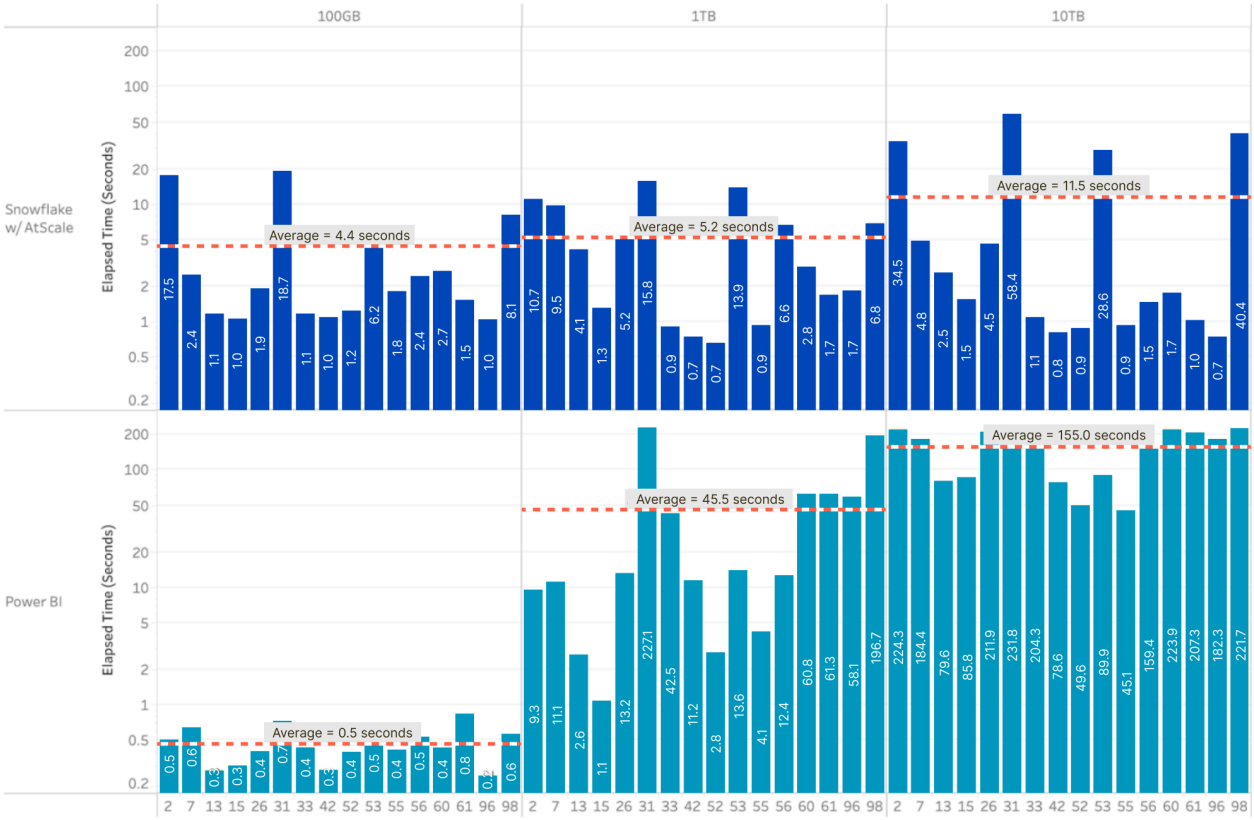
Illustration #13: Compute Cost Comparison of Power BI/Direct Lake and AtScale on Snowflake

Average Query Time Comparison

For end users, what matters the most is how fast their visualizations and dashboards render. In Illustration #14, you can see that Power BI/Direct Lake returns queries in about 500 milliseconds for the 100GB data size - a very good response time.

However, as data gets larger and concurrency increases, average query response times suffer due to DirectQuery fallback, subjecting users to inconsistent performance that depends on the size (what fits in memory) and the complexity (number of rows in a table) of data.

In contrast, while the average query time for AtScale on Snowflake has a floor of 4.4 seconds, query times stay level as data size and user concurrency grow, providing users with acceptable and consistent interactive performance.



Average elapsed time (seconds) for each TPC-DS query for all thread groups (1, 5, 25, 50) broken down by warehouse

Illustration #14: Power BI/Direct Lake & AtScale on Snowflake Average Query Time Comparison

While not an apples-to-apples comparison because Power BI/Direct Lake capacities are fixed, Illustration #15 shows the average query time for AtScale on a multi-cluster Snowflake data warehouse. You can see that query times are improved and even more consistent over the data ranges because Snowflake's multi-cluster feature reduces query queuing.

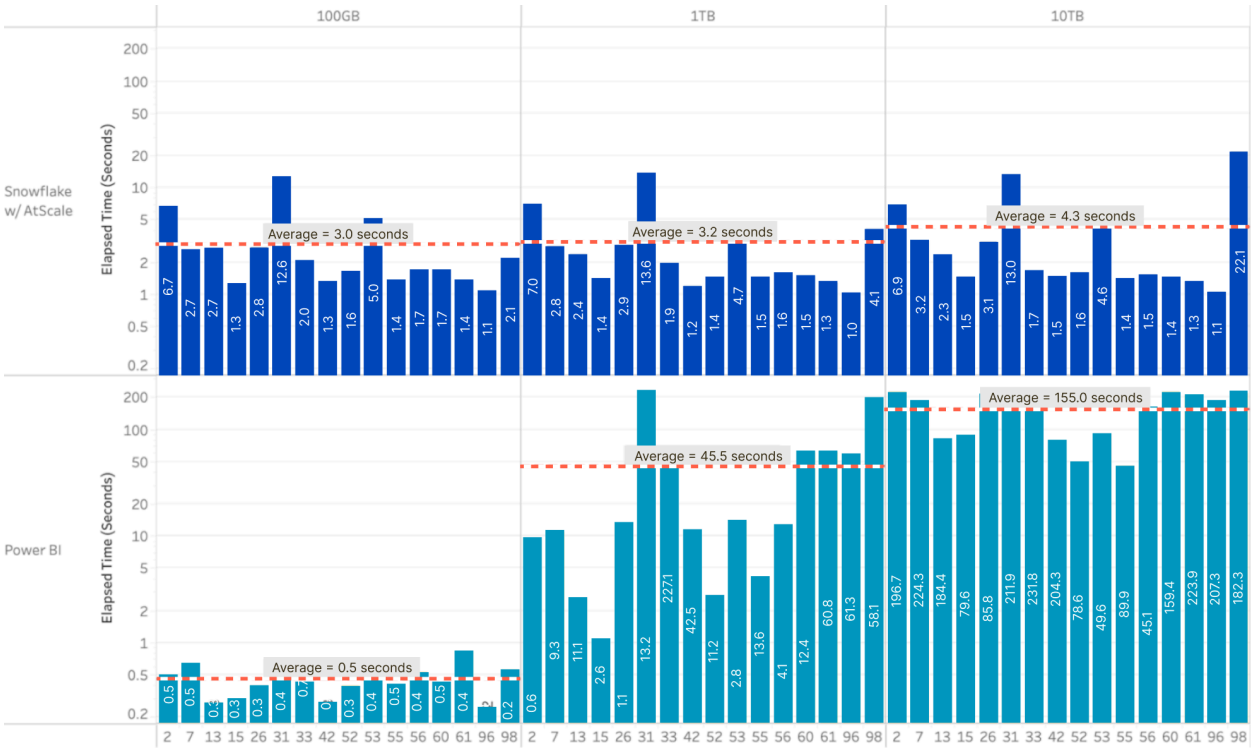


Illustration #15: Power BI/Direct Lake & AtScale on Snowflake Average Query Time Comparison for a Multi-Clustered Snowflake Warehouse

Power BI/Direct Lake Considerations

While conducting this benchmarking test, we worked closely with the Power BI on the Microsoft Fabric ecosystem. Overall, while there are some good new features, it appears that the Microsoft Fabric Lakehouse is purpose-built to provide an alternative to Power BI Import Mode and Power BI DirectQuery connections. In our experience, we found the following shortcomings that should be top of mind for those considering Power BI/Direct lake on Fabric:

1. **Lakehouse Limitations**

   We found that the Microsoft OneLake Fabric Lakehouse is immature and not yet ready for use as an alternative to platforms like Snowflake or Databricks for the following reasons:

1. Microsoft Fabric Lakehouse only supports a single part namespace for tables. Although the lakehouse supports file directories, lakehouse tables, which are required for Direct Lake, cannot be organized into catalogs or schemas (a 3-part namespace would be catalog/schema/table). Since we had three data sizes (100GB, 1TB, 10TB), we had to create three separate lakehouses and three separate semantic models. In our opinion, forcing users to store all their tables in one location is not suitable for enterprise use cases.
2. In order to make Power BI/Direct Lake performant, we needed to duplicate data already in ADLS in a Microsoft proprietary format ("v-order"). Besides data duplication, the requirement to create proprietary tables required extra data engineering to load and format the data. If the Microsoft Fabric Lakehouse was suitable for use as a general purpose lakehouse, this may not be a big issue since these tables may be useful across multiple use cases. However, given OneLake's current limitations and lack of integration with non-Azure tools, most organizations will need to add another layer of data processing to make PowerBI/Direct Lake work.
3. OneLake lacks integration support for tooling outside of Azure. Over time, we expect that OneLake will become more ecosystem friendly. However, as of this writing, integration with other data platforms like Databricks is limited to sharing files via ADLS through Fabric Shortcuts. This means that data cataloging and governance features like Databricks Unity Catalog are not yet supported.

2. Semantic Modeling Limitations

As of this writing, Microsoft Fabric's new web-based modeling tool is required to create semantic models that take advantage of Direct Lake. The new web-based modeling experience introduces several feature gaps compared to Power BI Desktop. For example, we encountered the following shortcomings in the Direct Lake semantic modeling experience:

1. **No support for Calculated Columns**. For many models, including TPC-DS, it is sometimes necessary to create calculations when defining a dimension. For example, in the TPC-DS model, we needed to create a "sales price tier" for some of the queries that look like this (in SQL):

```
Unset

CASE WHEN "SS_SALES_PRICE" > 200 THEN '200 and More'

WHEN "SS_SALES_PRICE"BETWEEN 150 AND 200 THEN '150-200'

WHEN "SS_SALES_PRICE" BETWEEN 100 AND 150 THEN '100-150'

WHEN "SS_SALES_PRICE" BETWEEN 50 AND 100 THEN ' 50-100'

ELSE ' 50 and Less' END
```

Illustration #16: Sales Price Tier Calculated Column (SQL)

In Power BI Desktop, this feature is available, but, as of this writing, it is not available in Power BI's Web modeling experience. As a result, this forced us to disqualify 4 (out of 20) TPC-DS queries for our test. *Customers should consider this a serious shortcoming, especially if they have existing models defined in Power BI Desktop that require Calculated Columns*.

2. **No way to share models or modeling components with other modelers**. Power BI Web modeling experience does not currently offer the ability to create or use a library of shared modeling components, like dimensions, metrics, and calculations. For our use case, due to the lakehouse namespace limitation, this required that we manually re-create semantic models which was a manual, error prone process.
3. **No way to edit a Power BI Semantic Model outside the web-based workspace**. Power BI Desktop is a richer modeling experience than the Power BI web-based modeler, but once a model is edited in Power BI Desktop, or the third-party tool, Tabular Editor, or has its properties changed in SSMS, it is no longer editable in the Power BI web-based interface (see below).

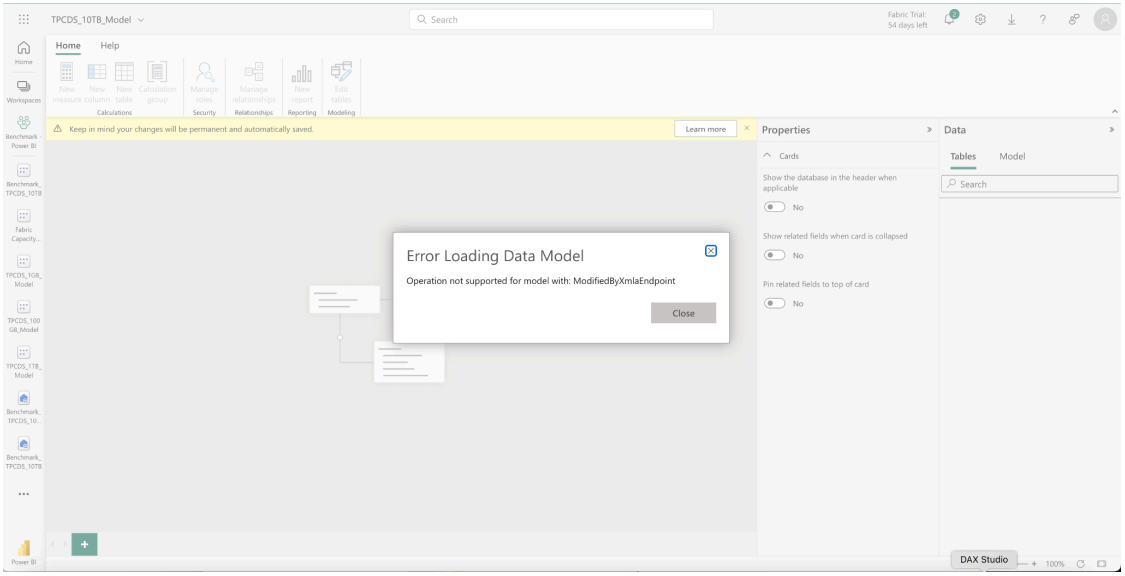


Illustration #17: Power BI/Direct Lake Web Modeler After Editing in Tabular Editor or SSMS

This made creating and editing our models difficult and once we did save a model outside of Power BI Web, our model became inaccessible.

3. Scalability and Stability

As you can see in Illustrations #14 and #15, Power BI/Direct Lake performs well for small data - 100GB in our case. However, once a dataset exceeded Direct Lake capacity limits (i.e. memory, number of rows, etc.), the fallback to DirectQuery made the experience inconsistent and unstable.

For example, queries timed out at 1TB & 10TB data levels in large numbers - up to 49% of queries timed out at the 10TB level. In addition, even at the 100GB level, a handful of queries failed when concurrency hit the 50 users. During our tests, we also encountered locking errors ("The operation was canceled because of locking conflicts.") when working on the model and querying at the same time.

Conclusion

The TPC-DS Benchmark for Power BI/Direct Lake presents a mixed picture. While there are areas where Power BI/Direct Lake shows promise, particularly with smaller datasets, it faces significant challenges in scalability, cost efficiency, and handling large-scale, concurrent data operations. The comparison with AtScale on Snowflake further highlights these gaps. Organizations considering Power BI/Direct Lake should carefully evaluate their data and operational needs to ensure that Power BI/Direct Lake can deliver results now and into the future.