

Design semantic models for scale in Microsoft Fabric

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/design-semantic-models-scale/1-introduction>

Introduction

Completed

Semantic models are the foundation of analytics in Microsoft Fabric. They define how data is structured, calculated, and consumed across reports, dashboards, and AI experiences. A model that works for a small team in Power BI Desktop doesn't automatically serve hundreds of users across multiple data stores. When data volumes grow, teams expand, and consumption patterns shift, the design decisions behind the model need to change.

Suppose an organization is scaling its analytics platform in Microsoft Fabric. Their data lives across lakehouses and warehouses, and their existing semantic models were built in Power BI Desktop for small teams. Now those models need to handle larger datasets, more concurrent users, and broader consumption patterns. The models work at their current size, but they weren't designed for scale.

In this module, you make the design decisions that prepare a semantic model for scale. You start by choosing the right storage mode for how data flows into the model. Then you design star schema relationships for clarity and performance. Next, you design calculations that stay performant and maintainable as data volumes and team size grow. Finally, you configure settings that control how the model handles large datasets, concurrent queries, and external tool access.

By the end of this module, you're able to design semantic models that use the right storage mode, follow star schema best practices, include scalable calculation patterns, and are configured for growing data volumes and consumption demands. Models designed for scale also benefit AI consumption, because AI demands the same things from a model: current data, clear relationships, descriptive structures, and the capacity to handle additional query load.

2. Choose a storage mode

Choose a storage mode

Completed

The first design decision for any semantic model in Microsoft Fabric is how data flows into the model. The storage mode you choose affects query performance, data freshness, and which Fabric features are available. In Fabric, Direct Lake is the default, and for most workloads it's the right choice.

Direct Lake mode

Direct Lake is the default storage mode for semantic models created in Microsoft Fabric. Unlike Import mode, Direct Lake doesn't copy data into the model. Unlike DirectQuery, it doesn't translate queries into source SQL. Instead, Direct Lake reads Delta tables directly from OneLake into memory, which combines the speed of Import with the freshness of DirectQuery.

When a user opens a report backed by a Direct Lake semantic model, the engine loads column data from Delta Parquet files on demand. You don't need to schedule a refresh, like you do with Import mode. When the underlying Delta tables update, the model reflects those changes.

Direct Lake models automatically enable the large semantic model storage format. This setting removes the 10-GB model size limit and is a prerequisite for both query scaleout and XMLA endpoint read/write access. You don't need to enable it manually for Direct Lake models.

Direct Lake connection options

Direct Lake models can connect to data through two paths:

- **OneLake tables:** The model connects directly to Delta tables in a lakehouse or warehouse. This is the simplest path and works well when your data is in a single Fabric data store.
- **SQL analytics endpoint:** The model connects through the SQL endpoint of a lakehouse or warehouse. This path enables access to views, cross-database queries, and security features defined at the SQL layer.

Choose OneLake tables when your data is straightforward and lives in one place. Choose the SQL analytics endpoint when you need views, cross-source joins, or row-level security defined in SQL.

Fallback behavior

Some operations can cause a Direct Lake model to fall back to DirectQuery mode. Complex DAX calculations, queries that exceed available memory, or certain unsupported operations trigger this fallback. When fallback occurs, the query runs against the SQL analytics endpoint rather than reading Delta files directly.

You can configure fallback behavior in the semantic model settings:

- **Allow fallback:** Queries that can't run in Direct Lake mode fall back to DirectQuery automatically. The user gets results, but performance might decrease.
- **Disallow fallback:** Queries that can't run in Direct Lake mode return an error. This option enforces consistent performance but requires that all queries stay within Direct Lake capabilities.

For most production workloads, start with fallback allowed and monitor which queries trigger it. Then optimize those queries or data structures to reduce fallback frequency over time.

Import mode

Import mode copies data into the semantic model and stores it in a compressed, in-memory format. Queries run against the local copy, which makes Import the fastest storage mode for query performance. However, the data is only as current as the last refresh.

Import mode is the right choice when:

- Your data source is outside Fabric (on-premises databases, third-party APIs, flat files).
- Query performance is the top priority and near-real-time freshness isn't required.
- You need features that aren't yet supported in Direct Lake.

Tip

When using Import mode, connect to views instead of raw tables, include only necessary columns, and use appropriate data types to reduce model size. Learn more about [techniques to reduce data loaded into Import models](#).

DirectQuery mode

DirectQuery sends queries directly to the data source at query time. No data is stored in the model, which makes DirectQuery suitable for real-time data scenarios and very large datasets that can't be imported.

The tradeoff is performance. Every report interaction generates a query against the source system. DirectQuery works best when:

- Real-time data is required and even short refresh delays aren't acceptable.
- Source data volumes are too large to import, and the data source is outside Fabric.
- Governance requirements mandate that data stays at the source.

Tip

For more information, see [DirectQuery model guidance](#).

Composite mode

Composite mode combines storage modes within a single model. Some tables use Import, while others use DirectQuery or Direct Lake. This provides flexibility for scenarios where different tables have different performance and freshness needs.

For example, a large fact table might stay in Direct Lake while a small reference table from an external source uses Import. Composite mode also enables many-to-many relationships between tables from different data sources.

Use composite mode when:

- You need data from both Fabric and non-Fabric sources in the same model.
- Some tables require real-time data while others benefit from cached performance.
- You need to combine Direct Lake tables with Import tables for cross-source analysis.

Choose the right storage mode

The following table summarizes when to choose each mode:

Mode	Data location	Query speed	Data freshness	Best for
Direct Lake	OneLake (Delta tables)	Fast	Near real-time	Fabric-native workloads (default)
Import	In-model cache	Fastest	Refresh-dependent	Non-Fabric sources, max performance
DirectQuery	Source system	Depends on source system	Near real-time	Real-time requirements, very large external data
Composite	Mixed	Varies	Mixed	Cross-source scenarios, hybrid requirements

Storage mode also affects AI consumption. When Copilot or data agents query a semantic model, they return answers based on whatever data the model currently reflects. Direct Lake's near-real-time freshness means AI queries return current results without waiting for a scheduled refresh. For models that serve both human users and AI, storage mode choice directly affects the quality of both experiences.

In Fabric, start with Direct Lake. Move to another mode only when your specific scenario requires it.

3. Design star schema for semantic models

<https://learn.microsoft.com/en-us/training/modules/design-semantic-models-scale/3-implement-star-schema>

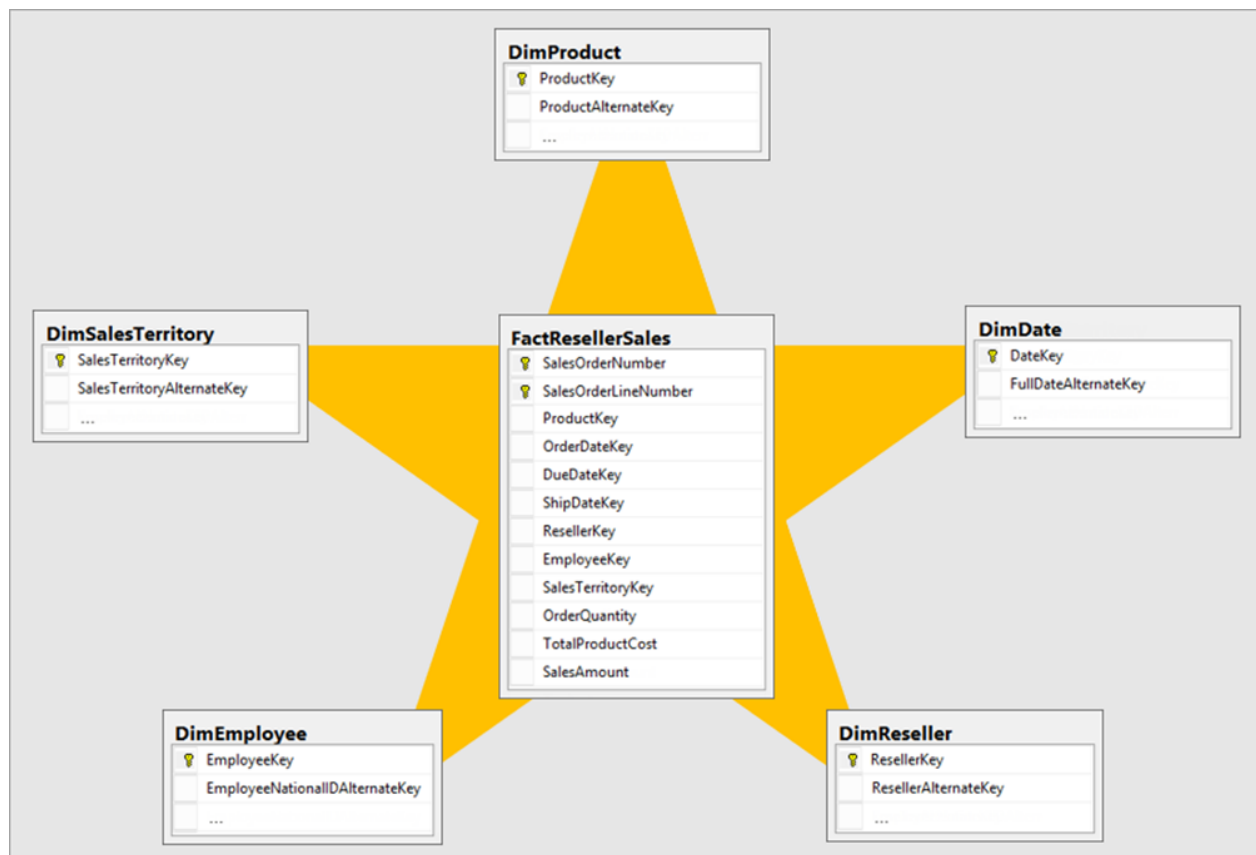
Design star schema for semantic models

Completed

You chose how data flows into your semantic model. Now design the star schema that organizes it for clear, performant queries. A star schema connects fact tables to dimension tables through relationships, creating the filter paths that reports and AI consumption depend on. If you're familiar with building star schema in Power BI Desktop, this unit focuses on the relationship design decisions that matter as models grow in complexity and scale.

Star schema in a semantic model

In a star schema, fact tables store measurable business events (such as sales transactions, order lines, and web visits), and dimension tables provide the descriptive context (such as product details, customer information, and date attributes). Dimension tables filter fact tables through relationships, which allows users to slice metrics by any descriptive attribute.



In a Fabric semantic model, this pattern provides clean filter propagation for both reports and AI consumption. When Copilot or a data agent generates a natural language query, a well-organized star schema gives the AI clear paths to the right data. Ambiguous or circular relationships confuse both report consumers and AI tools.

How storage mode affects relationships

Relationships in a semantic model behave differently depending on the storage mode. Understanding these differences is essential for designing star schema that performs well across different scenarios.

Direct Lake relationships

In Direct Lake mode, the engine reads relationships directly from the Delta table metadata. Relationships perform best when:

- Dimension key columns have low cardinality relative to fact table rows.
- Referential integrity is maintained in the source data. When referential integrity holds, the engine uses INNER joins instead of LEFT OUTER joins, which improves query performance.
- Columns used in relationships are indexed in the underlying Delta tables.

Note

If a query involves a relationship that causes the model to exceed memory limits or use unsupported operations, Direct Lake falls back to DirectQuery, and the relationship behavior changes to match DirectQuery semantics.

Cross-source relationships

Fabric semantic models can connect tables from different data stores. A fact table from a lakehouse can have a relationship to a dimension table from a warehouse, or to a table accessed through a SQL analytics endpoint. These cross-source connections use composite model capabilities.

When tables come from different sources, the storage mode for each table determines how the relationship works at query time. The engine resolves each side independently and joins the results.

Relationship types

One-to-many relationships

One-to-many is the most common relationship type in a star schema. One unique value in a dimension table relates to many rows in a fact table. For example, one product row in the Product dimension matches thousands of order rows in the Sales fact table.

Configure one-to-many relationships with the filter direction flowing from the dimension (the "one" side) to the fact table (the "many" side). This is the standard star schema filter pattern.

Many-to-many relationships

Many-to-many relationships are required when neither table has unique values for the relationship column. Use a bridge table to resolve these relationships. A bridge table sits between two tables and holds unique combinations of the keys from each side.

For example, if a customer can have multiple accounts and an account can belong to multiple customers, a Customer-Account bridge table resolves the relationship. The bridge table has one-to-many relationships to both the Customer and Account tables.

Filter direction

In most star schema implementations, use single-direction filtering from dimension to fact. This provides predictable filter propagation and avoids ambiguity in query results.

Bi-directional filtering is sometimes necessary for many-to-many relationships or when dimension tables need to be filtered by values in the fact table. Use bi-directional filters sparingly because they can degrade query performance and create unexpected filter behavior in reports.

Referential integrity

The **Assume referential integrity** setting tells the engine to use INNER joins rather than LEFT OUTER joins when querying across a relationship. In Direct Lake and DirectQuery modes, this setting can significantly improve performance because it reduces the number of rows the engine processes.

Enable this setting when you're confident that every foreign key value in the fact table has a matching value in the dimension table. If referential integrity is violated, rows with unmatched keys silently disappear from query results.

Inactive relationships and USERELATIONSHIP

Only one active relationship can exist between two tables at a time. When you need multiple relationship paths (such as an order date and a ship date both relating to the same Date dimension), make one relationship active and the others inactive.

Use the `USERELATIONSHIP` function in DAX to activate an inactive relationship within a calculation:

```
Shipped Amount =  
CALCULATE(  
    SUM(Sales[Amount]),  
    USERELATIONSHIP(Sales[ShipDate], 'Date'[Date])  
)
```

This pattern keeps the model clean while supporting multiple analytical perspectives on the same data.

Handle snowflake schema in semantic models

Source data often arrives in a normalized snowflake schema, where dimension tables are split into multiple related tables. For example, a Product dimension might be separated into Product, Subcategory, and Category tables, each linked through foreign keys.

In a semantic model, you have two options: flatten the snowflake into a star schema, or preserve the normalized structure.

Flatten into star schema

Flattening means combining the normalized dimension tables into a single denormalized dimension table. The Product table would include Subcategory and Category columns directly, eliminating the extra tables and relationships.

Flatten when:

- The combined dimension table is still small relative to the fact table (which is almost always the case for dimensions).
- You want simpler filter paths from dimension to fact. Each filter travels through one relationship instead of a chain.

- AI consumption is a priority. Fewer tables and simpler relationships give Copilot and data agents clearer paths to the right data.

Flatten dimension tables during data preparation in lakehouses or dataflows, before the data reaches the semantic model. Use Power Query merges, SQL joins, or notebook transformations to combine the normalized tables into a single dimension.

Preserve the snowflake structure

In some cases, keeping the normalized structure makes sense:

- The dimension hierarchy has multiple levels, and flattening would create dozens of redundant columns.
- Multiple fact tables share subdimension tables (such as a shared Category table used by both Sales and Inventory facts), and denormalization would create inconsistent copies.
- Row-level security needs to be applied at a specific level in the hierarchy.

When you preserve a snowflake structure, configure the relationships carefully. Each relationship in the chain must use single-direction filtering from the outermost table toward the fact table so that filters propagate correctly. A filter on Category needs to flow through Subcategory, then through Product, and into the fact table.

Note

In most semantic model scenarios, flattening dimensions into a star schema is the better choice. Fewer tables mean fewer relationships, simpler DAX, faster queries, and better AI consumption. Preserve the snowflake structure only when there's a strong reason to keep it.

When to use composite models for cross-source scenarios

Use composite models when your star schema spans multiple Fabric data stores or includes external sources. Common scenarios include:

- Fact tables in a lakehouse with dimension tables maintained in a warehouse.
- Real-time streaming data from an eventhouse combined with historical data in a lakehouse.
- Reference data from an external source (Import) combined with Fabric-native fact tables (Direct Lake).

In these scenarios, configure the storage mode for each table independently and verify that cross-source relationships perform acceptably at your expected data volumes.

4. Design scalable calculations

Design scalable calculations

Completed

Your model is structured. Now design the calculations that keep it performant and maintainable as your data and team grow. At small scale, a model with duplicated measures and inconsistent naming still works, even if it's not ideal. At scale, it breaks. A model with hundreds of measures needs structural design decisions that prevent duplicated logic, reduce query time on large datasets, and make it possible for new team members to understand and extend the model without introducing errors.

This unit covers three patterns: calculation groups for reducing measure proliferation, DAX readability discipline for team maintainability, and aggregations for query performance on large fact tables.

Calculation groups

Calculation groups are model objects that apply the same calculation pattern across multiple measures. Instead of creating separate measures for each variation, you define the pattern once and apply it dynamically.

The problem calculation groups solve

Consider an organization with 50 base measures (such as Total Sales, Total Cost, Profit, and Units Sold). Each measure needs Year-to-Date, Quarter-to-Date, and Month-to-Date calculations. Without calculation groups, that's $50 \times 3 = 150$ extra measures. Add prior year comparisons and you're looking at 250+ measures to maintain.

With calculation groups, you create one group with calculation items for each time intelligence pattern. Those items apply to any measure in the model automatically.

How calculation groups work

A calculation group contains calculation items, each defining a DAX expression that modifies the current measure using `SELECTEDMEASURE()`. Here's a time intelligence calculation group:

```
// Year-to-Date
CALCULATE(
    SELECTEDMEASURE(),
    DATESYTD('Date'[Date])
)
```

```
// Quarter-to-Date
CALCULATE(
    SELECTEDMEASURE(),
    DATESQTD('Date'[Date])
)
```

```
// Month-to-Date
CALCULATE(
    SELECTEDMEASURE(),
    DATESMTD('Date'[Date])
)
```

When a user adds the calculation group to a visual, they can switch between YTD, QTD, and MTD for any measure (such as Total Sales, Profit, or Units Sold) without separate measures for each combination.

Dynamic format strings

Dynamic format strings change the display format based on the calculation item context. For example, a percentage calculation should display as a percentage, while currency calculations should display as currency, even when applied to the same base measure.

```
// In the format string expression for a YoY % calculation item:
"0.0%"
```

Dynamic format strings reduce the need for separate formatted measures and keep formatting consistent across the model.

Tip

Learn more about how to [create calculation groups in Power BI](#).

When to use calculation groups

Use calculation groups when you have three or more measures that need the same calculation pattern applied. Common use cases include time intelligence (YTD, QTD, MTD), currency conversion, and variance calculations (actual vs. budget).

DAX readability discipline

At scale with a team maintaining 200+ measures, readability is a design decision, not a personal preference. Consistent, readable DAX reduces maintenance errors and makes it easier for new team members to understand the model.

Variables

Variables store intermediate results, improve readability, and prevent the engine from evaluating the same expression multiple times:

```
Profit Margin =  
VAR TotalRevenue = SUM(Sales[Revenue])  
VAR TotalCost = SUM(Sales[Cost])  
VAR ProfitAmount = TotalRevenue - TotalCost  
RETURN  
    DIVIDE(ProfitAmount, TotalRevenue)
```

Without variables, the same `SUM(Sales[Revenue])` expression might appear three times in a complex measure. Variables evaluate the expression once and reuse the result.

Tip

Learn more about [using variables to improve DAX formulas](#).

Naming conventions

Consistent naming is critical when your model has hundreds of measures maintained by multiple people. Establish conventions for:

- **Measure names:** Use clear, descriptive names such as "Total Sales" or "YoY Revenue Growth." Avoid abbreviations that only the original author understands.
- **Variable names:** Use descriptive names that explain the intermediate value (such as `TotalRevenue` rather than `x` or `temp`).
- **Calculation group items:** Name items by what they do, not how they work (such as "Year-to-Date" rather than "DATESYTD Wrapper").

Descriptive naming also matters for AI consumption. When Copilot or a data agent queries your model, it uses measure names and descriptions to determine which calculations to include. A measure named "YoY Revenue Growth" produces better AI results than "Calc7_v2."

Tip

Copilot in Power BI can help write and explain DAX formulas. When you're working on complex measures, use Copilot to suggest improvements or explain existing logic.

Iterators vs. aggregation functions

Iterator functions (`SUMX` , `AVERAGEX` , `MAXX`) evaluate a row-by-row expression over a table.

Aggregation functions (`SUM` , `AVERAGE` , `MAX`) operate on a single column. At large data volumes, the choice matters:

- Use aggregation functions when you're summarizing a single column. They're faster because the engine can use prebuilt data structures.

- Use iterators when the calculation requires a row-level expression (such as `Quantity × UnitPrice` per row).

Note

Iterators process every row, which can affect performance on large fact tables.

Information functions for defensive patterns

Information functions like `ISBLANK`, `HASONEVALUE`, and `ISINSCOPE` create defensive patterns for measures consumed by multiple reports with different filter contexts:

```
Sales per Customer =  
IF(  
    HASONEVALUE(Customer[CustomerID]),  
    DIVIDE(SUM(Sales[Amount]), 1),  
    DIVIDE(SUM(Sales[Amount]), DISTINCTCOUNT(Sales[CustomerID]))  
)
```

These patterns prevent unexpected results when measures are used in contexts the original author didn't anticipate.

Aggregations

Aggregations are summary tables that store precalculated totals at a higher grain than the detail data. Queries hit these tables first, which improves performance on large fact tables. When a query matches an aggregation, the engine returns results from the smaller summary table rather than scanning millions of detail rows.

Aggregations as a design decision

Deciding *when* to add aggregations and *at what granularity* is a design decision. Performance monitoring and tuning are separate operational concerns, but you make the structural choice during model design.

Consider aggregations when:

- Fact tables exceed millions of rows and commonly used queries summarize data at a higher grain (such as monthly totals by region).
- Users experience slow query response times on summary-level visuals.
- Most report interactions don't need row-level detail.

How aggregation behavior differs by storage mode

In Import mode, aggregations are stored as separate hidden tables. The engine automatically routes matching queries to the aggregation table.

In Direct Lake mode, the Delta tables themselves can serve as aggregation sources. Because Direct Lake reads columnar Parquet files, the engine can handle larger data volumes without aggregations in many scenarios. Add aggregations only when query patterns confirm the need.

Tip

Learn more about [user-defined aggregations in Power BI](#).

5. Configure settings for scale

<https://learn.microsoft.com/en-us/training/modules/design-semantic-models-scale/5-enterprise-settings>

Configure settings for scale

Completed

The model is designed. Now configure the settings that control how it handles large datasets, concurrent queries, and external tool access. These settings determine whether the model can keep up as data volumes grow and more users and tools consume it.

Large semantic model storage format

The large semantic model storage format changes how the model stores and compresses data. By default, Power BI limits model uploads to 10 GB. With this setting enabled, models can grow beyond that limit on refresh. The maximum size equals the Fabric capacity size or the limit set by the capacity administrator.

Direct Lake models automatically enable this setting, so you don't need to configure it manually for those models. For Import mode models, you need to enable it explicitly.

This setting is also a prerequisite for both XMLA endpoint read/write access and query scaleout. You need to enable it first when using Import or DirectQuery storage modes.

Enable large semantic model storage format when:

- Your data volumes require models that grow beyond the 10-GB upload limit.
- You need XMLA endpoint access for external tools.
- You plan to use query scaleout for high concurrency.
- You plan to use incremental refresh with partitioned tables.

XMLA endpoint read/write access

The XMLA endpoint lets external tools connect to your semantic model. Tools like Tabular Editor, DAX Studio, and ALM Toolkit use this endpoint for development, debugging, and deployment operations that aren't available in the Fabric service interface.

XMLA endpoint read/write access requires the large semantic model storage format as a prerequisite. Once both are enabled, you can:

- Use Tabular Editor for model development and source control integration.
- Use DAX Studio for query analysis and performance tuning.
- Deploy models through CI/CD pipelines using the Analysis Services client libraries.

At scale, these external tools become essential. Manual edits through the service interface don't support the level of model development that large, team-maintained models require.

Tip

Learn more about [XMLA endpoint connectivity](#).

Query scaleout

Query scaleout distributes read queries across read-only replicas of your semantic model. When hundreds of users access the same model simultaneously, a single instance can become a bottleneck. Query scaleout adds replicas that share the query load.

When you enable query scaleout, the read replicas use a separate copy of the model. This copy synchronizes after each refresh. There can be a brief delay between the primary model finishing a refresh and the replicas reflecting the updated data.

Query scaleout requires large semantic model storage format as a prerequisite.

Enable query scaleout when:

- The model serves hundreds of concurrent users.
- Query performance degrades during peak usage periods.
- The model is backed by a Fabric capacity that supports replicas.

Tip

Learn more about [query scaleout for semantic models](#).

Direct Lake fallback configuration

Direct Lake reads Delta tables directly from OneLake into memory. Some queries can cause the model to fall back to DirectQuery mode, which changes performance characteristics. The fallback setting controls how the model handles these situations:

- **Allow fallback** (default): Queries that can't run in Direct Lake mode fall back to DirectQuery automatically. Users get results, but performance might decrease.
- **Disallow fallback**: Queries that can't run in Direct Lake mode return an error. This enforces consistent performance but requires that all queries stay within Direct Lake capabilities.

For models at scale, start with fallback allowed. Monitor which queries trigger it, then optimize those queries or data structures to reduce fallback frequency. Disallow fallback only when all query patterns stay within Direct Lake limits and you need guaranteed performance consistency.

OneLake integration

OneLake integration makes your semantic model data accessible as Delta tables in OneLake. When enabled, downstream Fabric items like notebooks, pipelines, and other services can read data directly from the semantic model without rebuilding it from source.

This extends the model's reach beyond reports. A semantic model with well-structured star schema and calculation logic becomes a curated data source for the broader analytics platform.

Enable OneLake integration when:

- Data engineers or data scientists need to consume semantic model data in notebooks or other Fabric items.
- You want to use the semantic model as a shared data source across Fabric.
- Downstream consumers need access to curated, business-logic-enriched data without rebuilding it from raw sources.

Note

OneLake integration currently exports Import mode tables only. Direct Lake tables, DirectQuery tables, measures, and calculation group tables can't be exported. If your model uses Direct Lake exclusively, the underlying Delta tables in OneLake are already accessible to other Fabric items directly.

Settings decision framework

The following table summarizes the key settings decisions for scale:

Setting	Default	Enable when
Large semantic model storage format	Off	Data volumes exceed 10 GB, or you need XMLA endpoint access or query scaleout

Setting	Default	Enable when
XMLA endpoint read/write	Read-only	External tools need to modify the model for development or deployment
Query scaleout	Off	High concurrency degrades query performance (requires large semantic model storage format)
Direct Lake fallback	Allowed	Change to disallowed only when all queries stay within Direct Lake limits
OneLake integration	Off	Downstream Fabric items need to consume semantic model data

Tip

These settings address scale and consumption. Other settings such as endorsement, Copilot approval, and data preparation for AI consumption are covered in separate modules. For a complete reference, see [semantic model settings in the Fabric service](#).

6. Exercise: Design a semantic model for scale in Fabric

<https://learn.microsoft.com/en-us/training/modules/design-semantic-models-scale/6-exercise>

Exercise: Design a semantic model for scale in Fabric

Completed

In this exercise, you design a semantic model for scale in Microsoft Fabric by completing the following tasks:

- Connect to Fabric data sources using Direct Lake.
- Build a star schema with fact and dimension tables.
- Configure relationships with appropriate filter direction.
- Create a calculation group for time intelligence across multiple measures.
- Configure settings for scale: enable large semantic model storage format, configure XMLA endpoint access, and enable OneLake integration.
- Verify model behavior by confirming Direct Lake mode and checking fallback configuration.

This lab takes approximately **30** minutes to complete.

Note

You need access to a Fabric-enabled workspace to complete this exercise. For information about a trial license, see [Getting started with Fabric](#).

Launch the exercise and follow the instructions.

Launch Exercise

7. Module assessment

<https://learn.microsoft.com/en-us/training/modules/design-semantic-models-scale/7-knowledge-check>

Module assessment

Completed

8. Summary

<https://learn.microsoft.com/en-us/training/modules/design-semantic-models-scale/8-summary>

Summary

Completed

In this module, you explored what changes when semantic models need to handle larger datasets, more concurrent users, and broader consumption patterns in Microsoft Fabric. The challenge was clear: models built for small teams in Power BI Desktop don't automatically handle what comes with scale.

You learned to make four critical design decisions. First, you chose Direct Lake as the default storage mode and understood when Import, DirectQuery, or composite models are the better choice. Then you designed star schema relationships for clarity and performance, including referential integrity, inactive relationships, and cross-source connections. Next, you designed scalable calculations using calculation groups to reduce measure proliferation, variables and naming conventions to support

team maintainability, and aggregations to handle large data volumes. Finally, you configured settings that control how the model handles large datasets, concurrent queries, and external tool access.

Together, these decisions prepare a semantic model for scale. They also prepare it for AI consumption, because AI demands the same things from a model that scale does: current data, clear relationships, descriptive structures, and capacity.

Learn more

- [Direct Lake overview](#)
- [Create calculation groups in Power BI](#)
- [Large semantic model storage format](#)