

Get started with data warehouses in Microsoft Fabric

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/get-started-data-warehouse/1-introduction>

Introduction

Completed

Relational data warehouses are at the center of most enterprise business intelligence (BI) solutions. They provide a structured, SQL-based environment where organizations store, query, and analyze business data at scale.

Microsoft Fabric provides a fully managed data warehouse with full transactional T-SQL capabilities, including the ability to create tables and insert, update, and delete data. Because warehouse data is stored in Delta format on OneLake, it integrates seamlessly with other Fabric workloads. This makes the data warehouse a core component of your end-to-end analytics solution and part of the intelligent data foundation that supports Copilot experiences and AI-driven insights across the platform.

Suppose you work at a retail organization that stores structured business data across multiple systems. Your team needs to centralize this data for analytics and reporting using familiar SQL tools. You need to understand when a Fabric data warehouse is the right choice, how to create one, and how to query and transform data using T-SQL.

In this module, you explore the fundamentals of dimensional modeling with fact and dimension tables, then discover what makes Fabric's data warehouse unique, including full T-SQL support with MERGE capabilities, Copilot-assisted query authoring, and seamless integration with OneLake. You practice querying data, structuring tables, and explore the security and monitoring features that keep your warehouse governed and performant. By the end of this module, you'll know how to build a warehouse that serves both human analysts and AI-powered experiences.

2. Understand data warehouses

Understand data warehouses

Completed

A data warehouse is a centralized, structured store designed for analytical queries and reporting. Unlike operational databases that handle day-to-day business transactions, a data warehouse consolidates data from multiple sources into a format optimized for analysis.

Building a modern data warehouse typically involves:

- **Data ingestion** - Moving data from source systems into the warehouse.
- **Data storage** - Storing the data in a format optimized for analytics.
- **Data processing** - Transforming the data into a format ready for consumption by analytical tools.
- **Data analysis and delivery** - Analyzing the data to gain insights and delivering them to the business.

Design a data warehouse

Data warehouses contain tables organized in a schema optimized for multidimensional modeling. In this approach, you group numerical data related to events by different attributes. For instance, you can analyze the total amount paid for sales orders that occurred on a specific date or at a particular store.

Tables in a data warehouse

You organize data warehouse tables to support efficient analysis of large amounts of data. This organization, known as dimensional modeling, involves structuring tables into fact tables and dimension tables.

Fact tables contain the numerical data that you want to analyze. Fact tables typically have a large number of rows and are the primary source of data for analysis. For example, a fact table might contain the total amount paid for sales orders that occurred on a specific date or at a particular store.

Dimension tables contain descriptive information about the data in the fact tables. Dimension tables typically have a few rows and provide context for the data in the fact tables. For example, a dimension table might contain information about the customers who placed sales orders.

In addition to attribute columns, a dimension table contains a unique key column that uniquely identifies each row in the table. In fact, it's common for a dimension table to include two key columns:

- A *surrogate key* is a unique identifier for each row in the dimension table. It's often an integer value that the database management system generates automatically when you insert a new row.
- An *alternate key* is often a natural or business key that identifies a specific instance of an entity in the transactional source system - such as a product code or a customer ID.

You need both surrogate and alternate keys in a data warehouse, because they serve different purposes. Surrogate keys are specific to the data warehouse and help maintain consistency and accuracy. Alternate keys are specific to the source system and help maintain traceability between the data warehouse and the source system.

Special types of dimension tables

Special types of dimensions provide additional context and enable more comprehensive data analysis.

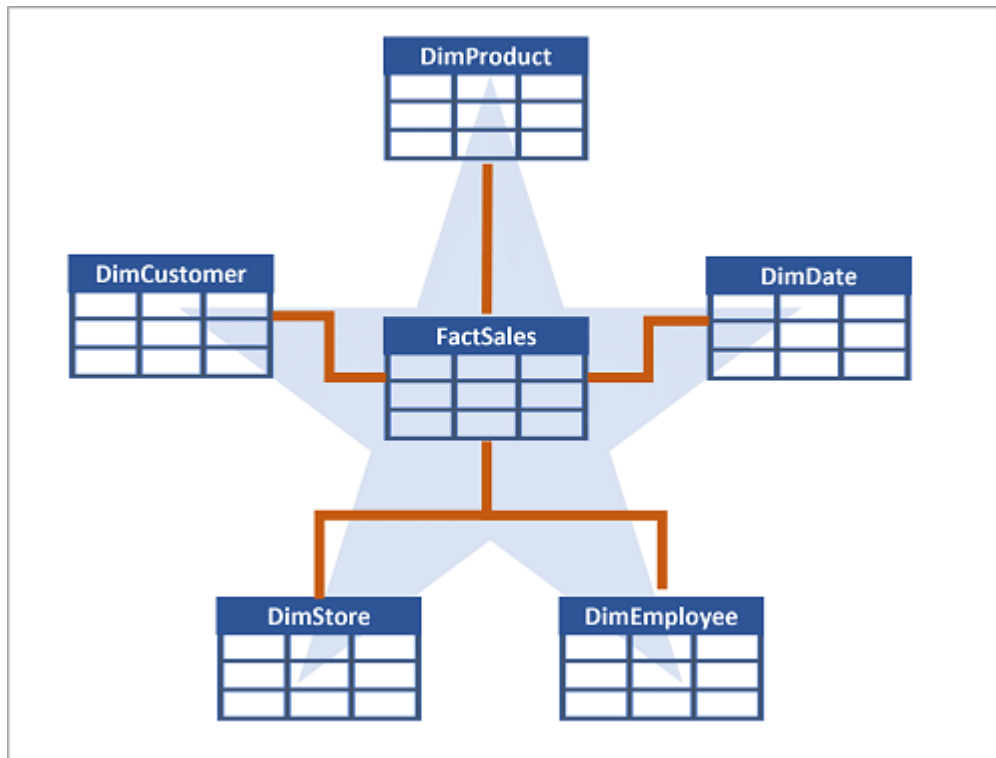
Time dimensions provide information about the time period in which an event occurred. This table enables data analysts to aggregate data over temporal intervals. For example, a time dimension might include columns for the year, quarter, month, and day of a sales order.

Slowly changing dimensions track changes to dimension attributes over time, like changes to a customer's address or a product's price. They're significant in a data warehouse because they enable you to analyze and understand changes to data over time. Slowly changing dimensions ensure that data stays up-to-date and accurate, which is important for making good business decisions.

Data warehouse schema designs

In most transactional databases used in business applications, the data is *normalized* to reduce duplication. In a data warehouse however, the dimension data is denormalized* to reduce the number of joins required to query the data.

Often, a data warehouse uses a *star schema*, in which a fact table relates directly to the dimension tables, as shown in this example:

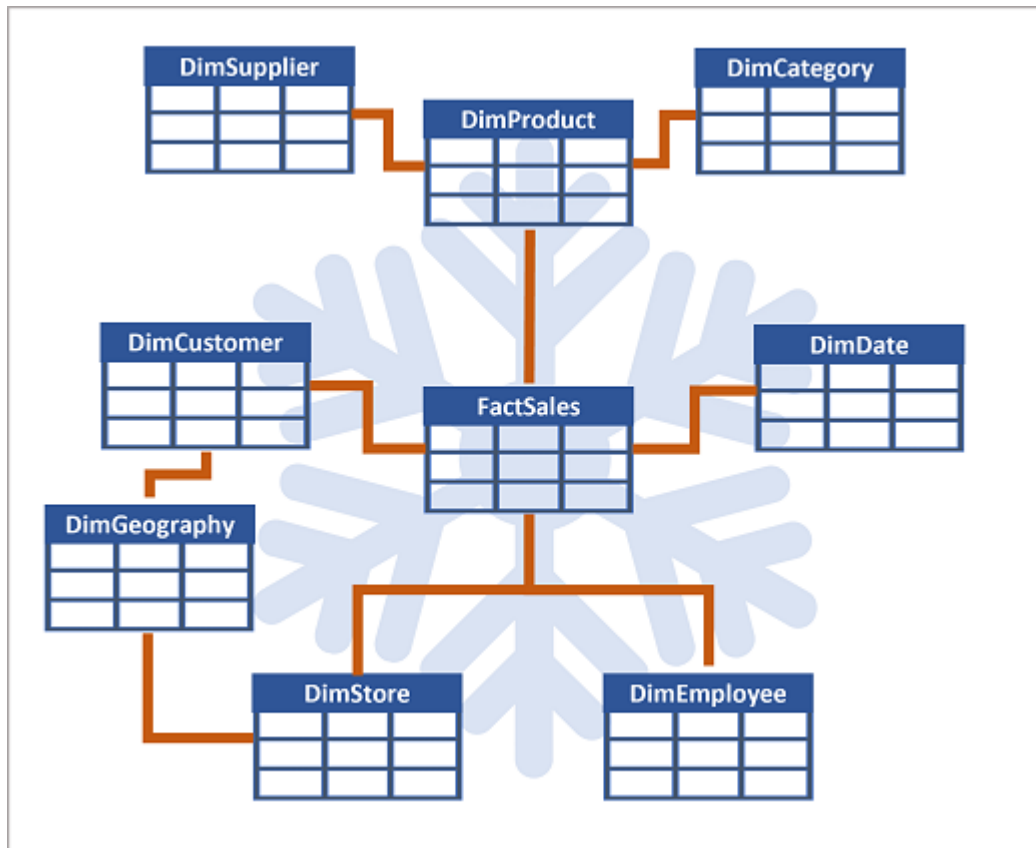


You can use dimension attributes to group fact table numbers at different levels. For example, you could find the total sales revenue for a whole region or just for one customer. You can store the information for each level in the same dimension table.

Tip

See [What is a star schema?](#) for more information on designing star schemas for Fabric.

If there are lots of levels or attributes shared by different things, it might make sense to use a *snowflake schema* instead. Here's an example:



In this case, the **DimProduct** table splits (normalizes) into separate dimension tables for product categories and suppliers.

- Each row in the **DimProduct** table contains key values for the corresponding rows in the **DimCategory** and **DimSupplier** tables.

A **DimGeography** table contains information on where customers and stores are located.

- Each row in the **DimCustomer** and **DimStore** tables contains a key value for the corresponding row in the **DimGeography** table.

3. Understand data warehouses in Fabric

<https://learn.microsoft.com/en-us/training/modules/get-started-data-warehouse/3-understand-data-warehouse-fabric>

Understand data warehouses in Fabric

Completed

Now that you understand data warehouse fundamentals, let's explore what Microsoft Fabric provides for data warehousing.

Describe a Fabric data warehouse

A Fabric data warehouse is a fully managed, enterprise-scale relational database built on OneLake. It provides full transactional T-SQL capabilities, including DDL statements (CREATE, ALTER, DROP) and DML statements (INSERT, UPDATE, DELETE, MERGE), with full ACID compliance for data consistency.

Data is stored in open Delta format on OneLake, which means other Fabric workloads can access the same data without duplication. You use T-SQL to create tables, load data, build views and stored procedures, and perform transformations, all within a familiar SQL experience.

Key capabilities include:

- **Full T-SQL support** - Write DDL and DML statements, including MERGE for upsert scenarios, using familiar SQL Server syntax.
- **Fully managed** - No infrastructure to configure. Compute scales automatically and independently from storage.
- **OneLake integration** - Warehouse data is stored in Delta format and accessible by other Fabric workloads without duplication.
- **Cross-database querying** - Query data across warehouses and lakehouses without copying data. Use three-part naming (database.schema.table) to join warehouse tables with lakehouse tables in a single query.
- **Familiar tooling** - Connect with SQL Server Management Studio (SSMS), Azure Data Studio, or any SQL client through standard TDS connections.
- **Copilot assistance** - Copilot for Data Warehouse generates SQL queries from natural language, provides code completion as you type, and can explain or fix existing queries in the SQL editor.

Warehouse vs SQL analytics endpoint

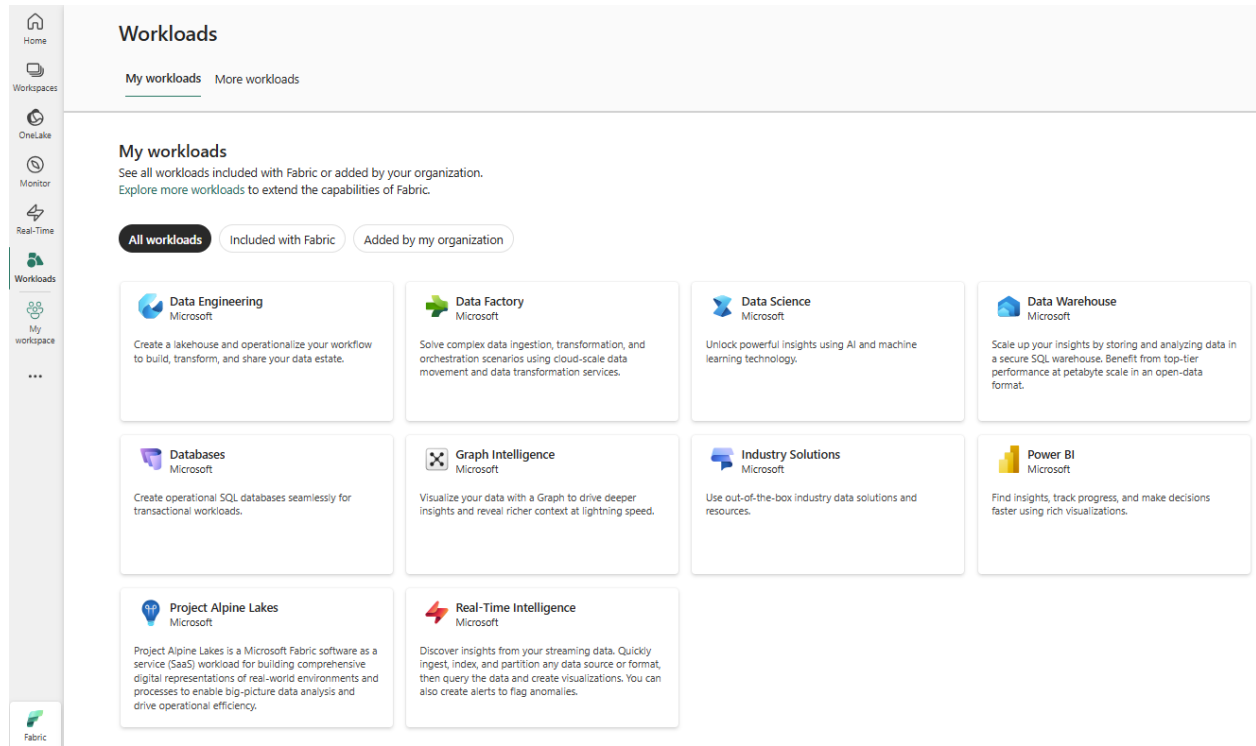
Fabric workspaces can contain two types of SQL-based items that serve different purposes.

Capability	Warehouse	SQL analytics endpoint
Read data	Yes	Yes
Write data (INSERT, UPDATE, DELETE, MERGE)	Yes	No
Create tables (DDL)	Yes	No
Create views and stored procedures	Yes	Yes
Data source	Native warehouse tables	Lakehouse Delta tables

Use a warehouse when you need full read/write T-SQL capabilities. Use the SQL analytics endpoint when you need read-only SQL access to lakehouse data.

Create a data warehouse

You can create a data warehouse in Fabric from the **create hub** or within a **workspace**. After creating an empty warehouse, you can add tables, views, and other objects.



Once your warehouse is created, you can start creating tables and loading data using the SQL query editor in the Fabric portal.

Ingest data into a warehouse

There are several ways to load data into a Fabric data warehouse:

- **COPY INTO** - Bulk load data from external files (CSV, Parquet) in Azure storage into warehouse tables.
- **OPENROWSET** - Query files directly from external storage or OneLake locations for ad hoc analysis or ingestion, without creating tables first.
- **Pipelines and Dataflows** - Use Data Factory pipelines or Dataflows Gen2 for orchestrated data movement and transformation.
- **Cross-database queries** - Query lakehouse tables directly from the warehouse using three-part naming, without copying data.

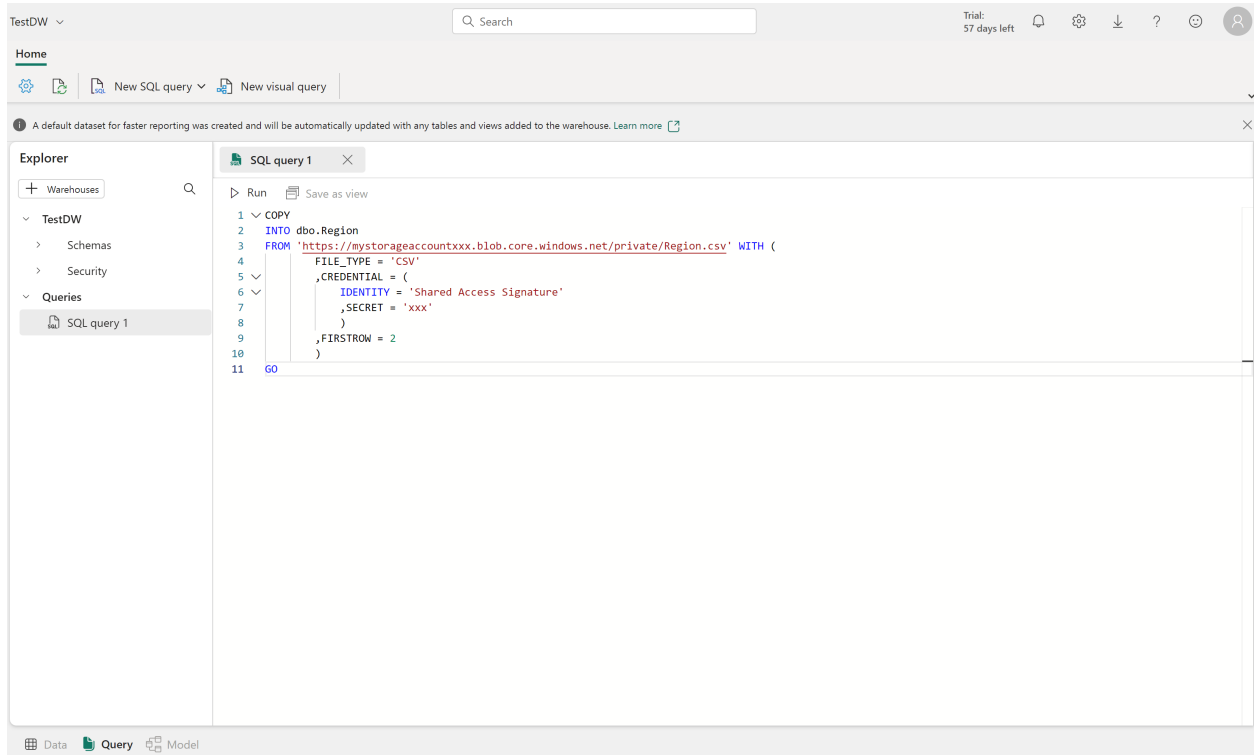
You can use the `COPY INTO` T-SQL command to bulk load data from files. For example, the following statement loads data from a CSV file into a table:

```
COPY INTO dbo.Region
FROM 'https://mystorageaccount.blob.core.windows.net/data/Region.csv'
WITH (
```

```

FILE_TYPE = 'CSV',
CREDENTIAL = (
    IDENTITY = 'Shared Access Signature',
    SECRET = 'xxx'
),
FIRSTROW = 2
)
GO

```



Tip

If you have tables in a lakehouse that you want to query from your warehouse without making changes, use cross-database querying instead. You don't need to copy the data.

Create tables and load data

After creating a warehouse and choosing an ingestion method, the next step is to define your tables and load data into them.

You create tables using T-SQL `CREATE TABLE` statements. Define columns with appropriate data types for analytics workloads.

```

CREATE TABLE dbo.DimCustomer
(
    CustomerKey INT NOT NULL,
    CustomerAltKey NVARCHAR(10) NOT NULL,
    CustomerName NVARCHAR(100) NOT NULL,

```

```

        Region NVARCHAR(50) NULL
    );
GO

CREATE TABLE dbo.FactSales
(
    SalesKey INT NOT NULL,
    CustomerKey INT NOT NULL,
    ProductKey INT NOT NULL,
    DateKey INT NOT NULL,
    SalesAmount DECIMAL(10,2) NOT NULL,
    Quantity INT NOT NULL
);
GO

```

Choose data types that balance precision with storage efficiency. Use `INT` for key columns, `NVARCHAR` for text that may include special characters, and `DECIMAL` for financial values that require precision.

Use staging tables for data loading

A common pattern in data warehousing is to land raw data in staging tables before transforming and loading it into final dimension and fact tables. Staging tables mirror the structure of your source data and act as a temporary holding area.

After loading data into staging tables using `COPY INTO` or pipelines, you transform and insert it into your dimensional model:

```

INSERT INTO dbo.FactSales (SalesKey, CustomerKey, ProductKey, DateKey, SalesAmount, Quantity)
SELECT
    s.OrderID,
    c.CustomerKey,
    p.ProductKey,
    d.DateKey,
    s.Amount,
    s.Qty
FROM dbo.StgSales AS s
INNER JOIN dbo.DimCustomer AS c ON s.CustomerID = c.CustomerAltKey
INNER JOIN dbo.DimProduct AS p ON s.ProductID = p.ProductAltKey
INNER JOIN dbo.DimDate AS d ON s.OrderDate = d.DateValue;
GO

```

This pattern keeps your source data intact while you apply business rules and key lookups during the load process.

Understand table clones

You can create zero-copy table clones in a Fabric data warehouse. Clones copy table metadata while still referencing the same underlying data files in OneLake. The data itself isn't duplicated, which

keeps storage costs low.

The following code example shows you how to create a clone with T-SQL:

```
--Clone creation within the same schema  
CREATE TABLE dbo.Employee AS CLONE OF dbo.EmployeeUSA;
```

Table clones are useful for development and testing, data recovery after a failed release, and preserving data at specific points in time for historical reporting.

Tip

For more information, see the [Clone table in Microsoft Fabric](#) documentation.

Now that you understand Fabric's warehouse capabilities, let's explore how to query and transform your data.

4. Query and transform data

<https://learn.microsoft.com/en-us/training/modules/get-started-data-warehouse/4-query-transform-data>

Query and transform data

Completed

Now that you know how to create a warehouse and ingest data, you can start exploring and shaping that data for analytics.

Raw data rarely arrives in the exact format you need for analysis. You might need to join tables, filter rows, aggregate values, or restructure data before it's useful for reporting. A Fabric data warehouse gives you two tools for this work: the SQL query editor for T-SQL and the Visual query editor for a no-code approach.

Query data with the SQL query editor

The **SQL query editor** provides a query experience that includes IntelliSense, code completion, syntax highlighting, client-side parsing, and validation. This will feel familiar if you have experience writing T-SQL in SQL Server Management Studio (SSMS) or Azure Data Studio (ADS).

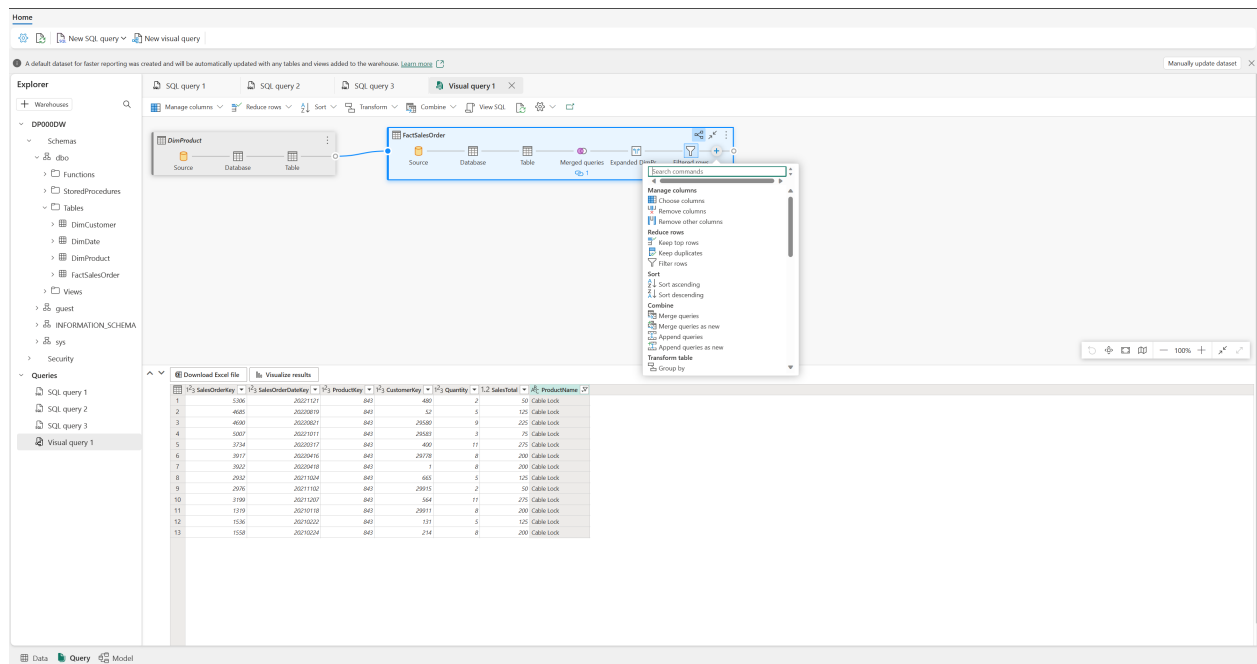
To create a new query, use the **New SQL query** button in the menu. Copilot for Data Warehouse is available in the editor to help generate queries from natural language, complete code as you type,

and explain or fix existing queries.

Query data with the Visual query editor

The *Visual query editor* provides an experience similar to the [Power Query online diagram view](#). Use the **New visual query** button to create a new query.

Drag a table from your data warehouse onto the canvas to get started. You can then use the **Transform** menu at the top of the screen to add columns, filters, and other transformations. You can also use the (+) button on the visual itself to perform similar actions.



Transform data with views and stored procedures

Beyond ad hoc queries, you can save transformation logic as reusable objects in the warehouse.

Views define a saved query that you can reference like a table. Use views to standardize how analysts access data, such as combining fact and dimension tables into a reporting-friendly format or filtering rows to a specific business context. For example:

Stored procedures contain T-SQL logic that you can execute on demand. Use stored procedures for repeatable transformation tasks, like loading staging data into final tables or applying business rules.

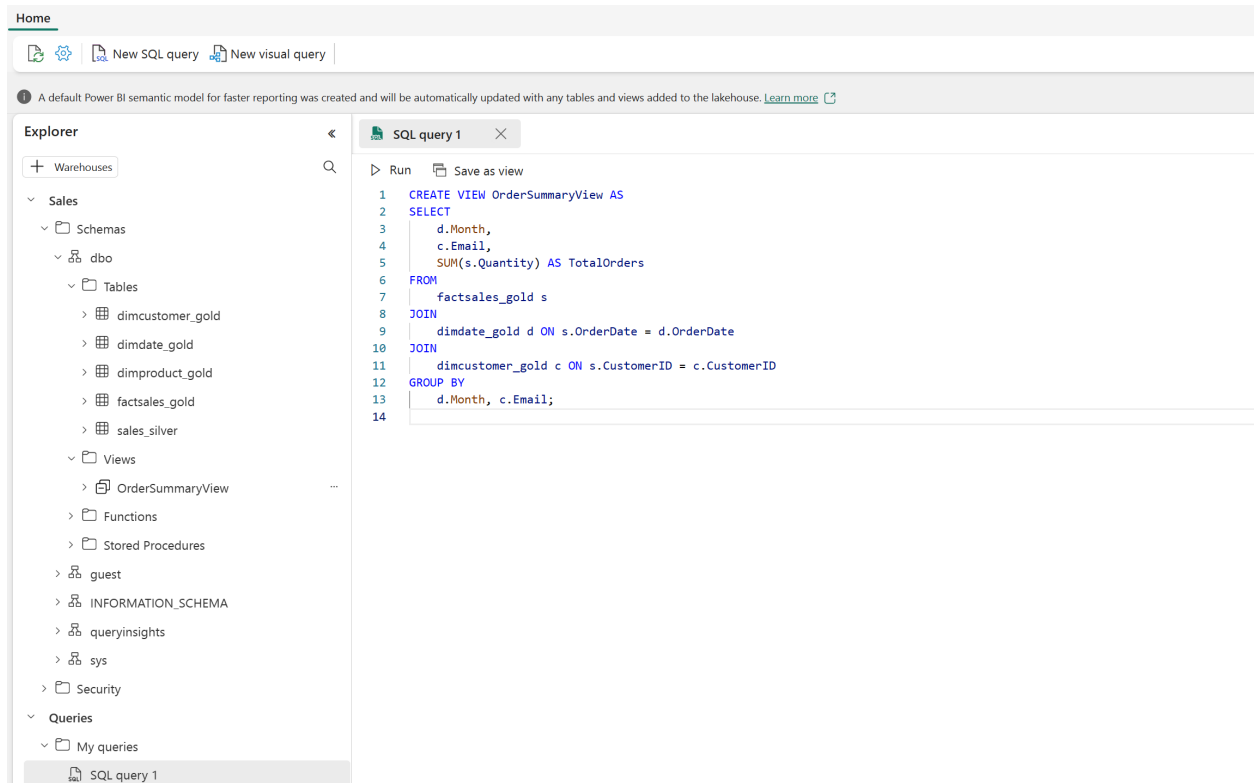
Views and stored procedures also help make your data more accessible to AI-powered tools. Copilot and Fabric IQ data agents can query views just like tables, so standardizing data access through well-named views improves the accuracy of natural language queries.

The following code shows a sample view and the screenshot shows how to use the code in the warehouse SQL query editor:

```

CREATE VIEW dbo.vw_SalesByRegion
AS
SELECT
    c.Region,
    SUM(f.SalesAmount) AS TotalSales,
    COUNT(f.OrderID) AS OrderCount
FROM dbo.FactSales AS f
INNER JOIN dbo.DimCustomer AS c
    ON f.CustomerKey = c.CustomerKey
GROUP BY c.Region;

```



5. Model data in a warehouse

<https://learn.microsoft.com/en-us/training/modules/get-started-data-warehouse/5-model-data>

Model data in a warehouse

Completed

Without data modeling, every consumer has to figure out which tables relate to each other, write their own aggregation logic, and guess at column meanings. Data modeling solves this problem by embedding structure, business logic, and documentation directly into the warehouse. In a Microsoft

Fabric warehouse, you prepare data for clarity, define relationships between tables, standardize access through views and measures, and publish semantic models for reporting. These modeling choices affect every downstream experience, including T-SQL queries, Power BI reports, and AI-driven natural language analytics.

Prepare data for consumption

Before you define relationships or add calculations, you need to clean up what consumers see. Raw warehouse tables often contain staging tables, surrogate key columns, and internal flags that are meant for ETL processing, not for analysis. These objects create noise when consumers browse the data. Preparing the warehouse for consumption means surfacing only what's relevant and making it understandable.

In the model view, you can take several steps to improve the consumer experience:

- **Hide internal objects** like staging tables, surrogate key columns, and ETL artifacts that clutter the field list.
- **Rename columns** to use business-friendly names where the warehouse column names are technical or abbreviated. For example, rename `CustRgn` to `Customer Region`.
- **Add descriptions** to tables and columns so that consumers understand what the data represents without referring to external documentation.

These steps matter beyond just tidiness. Copilot in Power BI and Fabric IQ data agents rely on table names, column names, and descriptions to interpret natural language questions and generate accurate SQL or DAX. A column named `Customer Region` with a description like "Geographic region of the customer's primary address" produces better natural language results than `CustRgn` with no description.

With clean, well-named tables in place, you're ready to define how those tables connect to each other.

Understand relationships between tables

A relationship is a logical connection between two tables that enables filtering, grouping, and aggregation across those tables. In a star schema, relationships connect fact tables to dimension tables through shared key columns.

For example, a `CustomerKey` column that exists in both `FactSales` and `DimCustomer` establishes the link that enables analysis of sales by customer attributes like region, segment, or account type.

Each relationship has two important properties.

- **Cardinality** describes how rows in the two tables correspond. In a star schema, fact-to-dimension relationships are typically many-to-one, meaning many fact rows map to a single dimension row.

- **Cross-filter direction** determines which way filters propagate between the tables. Single direction, where the dimension filters the fact table, is the standard setting for most star schema designs because it keeps filter behavior predictable and performant.

Without defined relationships, every consumer who wants to combine data across tables needs to write explicit JOIN logic. Relationships eliminate that repetition by encoding the connection once. When you create a semantic model from the warehouse, these relationships inform how Power BI, Copilot, and Fabric IQ data agents interpret the data. Data agents, for example, use relationships to generate accurate joins when translating natural language questions into SQL.

Note

Most data warehouses use dimensional modeling. Relationships can be created to shape a **star schema**, which is an ideal model for analytics. For more information, see the [Design dimensional models in Microsoft Fabric](#) module.

Standardize data access with views and measures

Now that your tables are clean and connected, the next step is to give consumers reliable, consistent ways to query and calculate against that data. Without standardization, each team writes its own join logic, applies its own filters, and defines its own formulas, which leads to conflicting results.

Views provide this consistency for T-SQL consumers. A view encapsulates join logic, filters, and column selections into a reusable query that consumers reference like a table. For example, a view that joins fact and dimension tables, filters to completed orders, and surfaces only the columns analysts need gives every T-SQL consumer a reliable starting point. Views also serve as stable data sources for reports. Rather than building reports directly against base tables that might change, you can point reports at views that present a consistent shape.

Measures provide the same consistency for DAX calculations. A measure is a reusable DAX expression that defines a calculation like a total, average, ratio, or count. You create measures directly in the warehouse model view by selecting a table and adding a new measure. For example, a **Total Sales** measure that sums the **SalesAmount** column ensures every consumer uses the same calculation.

Because the measure definition lives with the data, it becomes the single source of truth for that metric. When the business changes how it calculates revenue, you update the measure in one place rather than tracking down every report that contains its own formula.

Together, views and measures cover both sides of consumption: views standardize how T-SQL consumers access and query data, while measures standardize how business calculations appear in reports and dashboards.

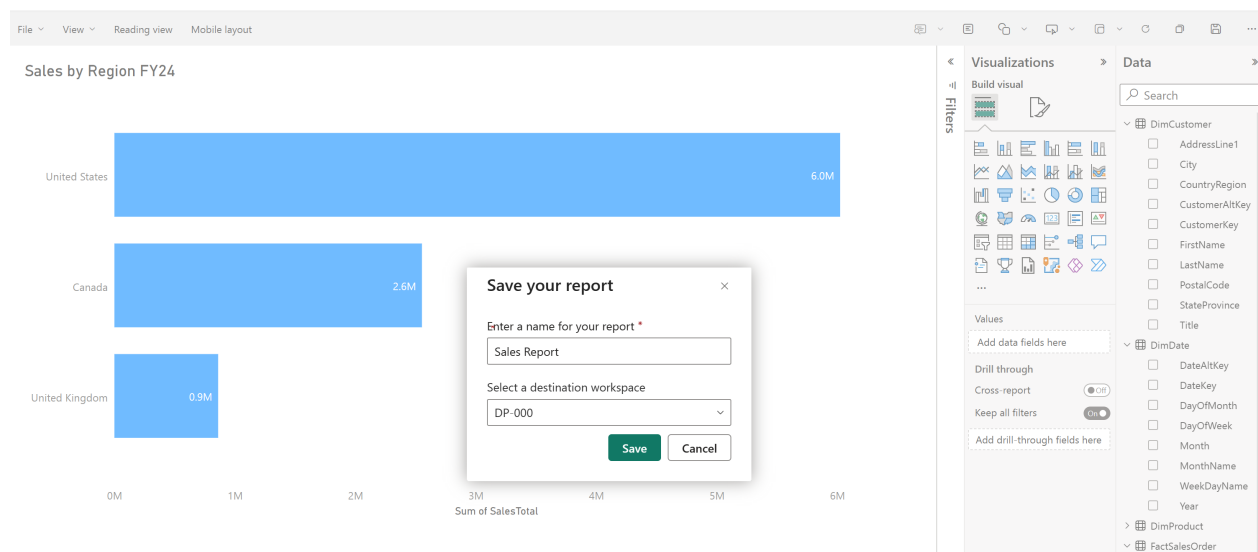
Tip

DAX formulas and advanced measure design are covered in depth in later modules. For views and stored procedures, see the previous unit on querying and transforming data.

Create a semantic model for Power BI reporting

With prepared tables, defined relationships, and standardized views and measures in place, the warehouse is ready for downstream reporting. Teams that query the warehouse directly by using T-SQL or connect through third-party tools can work with the warehouse model as-is. However, when you want to build interactive Power BI reports and dashboards, creating a semantic model is the next step.

Semantic models created from a Fabric warehouse use Direct Lake mode. Unlike traditional import mode, which copies data into Power BI memory, Direct Lake reads data directly from OneLake Parquet files. This means reports reflect the latest warehouse data without requiring scheduled refreshes. It also means you avoid the storage and processing overhead of maintaining a separate copy of the data.



Tip

Semantic model design and scalability patterns are covered in greater depth in [Design scalable semantic models](#). This unit focuses on modeling data in the warehouse itself.

6. Secure and monitor a warehouse

Secure and monitor a warehouse

Completed

Security and monitoring are critical aspects of managing your data warehouse. Fabric provides multiple layers of protection and visibility tools to help you control access and understand query performance.

Security

Fabric data warehouse security operates at multiple levels, from workspace access down to individual rows and columns. This design allows you to support the distinct needs of your organization by still allowing the democratization of data, but with governance.

Workspace roles

Data in Fabric is organized into *workspaces*, and workspace roles are the first layer of access control. Assign users to appropriate roles based on the level of access they need. For example, Admins have full control, while Viewers can view items but can't make changes.

Tip

For more information, see [Workspaces in Power BI](#).

Item permissions

In addition to workspace roles, you can grant **item permissions** to share individual warehouses without granting access to the entire workspace. This granularity is useful when you need to share a warehouse for downstream consumption with specific users.

Grant the following permissions as needed:

- **Read** - Allows the user to connect using the SQL analytics endpoint.
- **ReadData** - Allows the user to read data from any table or view in the warehouse.
- **ReadAll** - Allows the user to read raw parquet files in OneLake.

Note

A user connection to the SQL analytics endpoint fails without Read permission at a minimum.

Granular SQL security

For more precise access control, Fabric data warehouse supports granular security using T-SQL. These features let you restrict data visibility without changing the underlying tables:

- **Object-level security** - Control access to specific tables, views, or procedures.
- **Row-level security (RLS)** - Restrict which rows a user can see using WHERE clause predicates.
- **Column-level security (CLS)** - Restrict which columns are visible to specific users.
- **Dynamic data masking** - Mask sensitive data (such as email addresses or account numbers) from non-privileged users.

Securing your warehouse data is important for both regulatory compliance and for ensuring that AI-powered tools like Copilot and data agents operate within governed boundaries. Security policies you define in T-SQL are enforced regardless of how the data is accessed.

Tip

Row-level security, column-level security, and dynamic data masking are covered in depth in [Secure a Microsoft Fabric data warehouse](#).

Monitoring

Monitoring warehouse activity helps you identify performance issues, optimize queries, and understand usage patterns.

Query insights

Query insights provides a central location for historical query data and actionable performance information. It retains data for 30 days and helps you identify long-running queries, track performance changes over time, and understand which queries consume the most resources.

Query insights uses system views that you can query directly:

- `queryinsights.exec_requests_history` - Returns information about each completed SQL request.
- `queryinsights.long_running_queries` - Returns queries ranked by execution time.
- `queryinsights.exec_sessions_history` - Returns information about completed sessions.

Dynamic management views

You can also use *dynamic management views* (DMVs) to monitor active connections, sessions, and requests in real time. For example, use `sys.dm_exec_requests` to identify currently running queries:

```
SELECT request_id, session_id, start_time, total_elapsed_time
FROM sys.dm_exec_requests
```

```
WHERE status = 'running'  
ORDER BY total_elapsed_time DESC;
```

Important

You must be a workspace Admin to run the `KILL` command to terminate long-running sessions. Members, Contributors, and Viewers can see their own results but can't see other users' queries.

7. Exercise - Create and query a warehouse

<https://learn.microsoft.com/en-us/training/modules/get-started-data-warehouse/7-exercise>

Exercise - Create and query a warehouse

Completed

Now it's your turn to create a data warehouse in Fabric and analyze your data.

Note

You need a Microsoft Fabric trial license with the Fabric preview enabled in your tenant. See [Getting started with Fabric](#) to enable your Fabric trial license.

Launch the exercise and follow the instructions.

[Launch Exercise](#)

8. Module assessment

<https://learn.microsoft.com/en-us/training/modules/get-started-data-warehouse/8-knowledge-check>

Module assessment

Completed

9. Summary

<https://learn.microsoft.com/en-us/training/modules/get-started-data-warehouse/9-summary>

Summary

Completed

In this module, you learned how data warehouses use dimensional modeling to organize data into fact and dimension tables, and what makes a Fabric data warehouse unique. You explored querying and transforming data with T-SQL and the visual query editor, structured tables into a star schema, and applied security features like row-level security and dynamic data masking to protect your data.

Without a platform like Microsoft Fabric, building this kind of warehouse environment would require provisioning and managing dedicated SQL infrastructure, configuring separate storage and compute, and manually integrating data across siloed systems. A Fabric data warehouse eliminates that complexity by combining full T-SQL capabilities with OneLake integration in a single, governed platform that supports both traditional analytics and AI-powered experiences.

To learn advanced T-SQL transformation patterns like staging workflows, incremental loads, and MERGE-based upserts, continue to the [Transform data using T-SQL](#) module.

Learn more

- [What is Fabric Data Warehouse?](#)
- [Query the warehouse](#)