Data Warehouse- Fabric

A **Warehouse** is a core component of Microsoft Fabric's **OneLake architecture**, enabling **structured, scalable, and fast querying** of large datasets using **familiar T-SQL** — ideal for business analysts, data engineers, and Power BI users.

🔧 Creating a Warehouse in Fabric

1. **Go to Microsoft Fabric workspace**
2. Click **"New" → "Warehouse"**
3. Give it a name (e.g., `SalesWarehouse`)
4. Use the **SQL editor** or **pipelines** to load data

You can now query Lakehouse tables from a Warehouse, but only if the Lakehouse table is a managed table.

☑ Key Update: Lakehouse-to-Warehouse Cross Querying Now Supported (with Conditions)

◇ Condition:

- ✅ The Lakehouse table **must be a managed table**
- ❌ External or shortcut-based tables **cannot** be queried directly from a Warehouse

🧠 What is a Managed Table in Lakehouse?

A **managed table** is one where:

- The data is **stored and controlled inside** the Lakehouse (OneLake)
- Created by Dataflows Gen2, Notebooks, Pipelines, or SQL using `CREATE TABLE` without external location

🔗 How Cross Querying Works Now

📍 From Warehouse (T-SQL), you can now do:

```
sql
CopyEdit
SELECT TOP 10 * FROM LakehouseName.Schema.TableName
```

🔧 Syntax:

```
sql
CopyEdit
SELECT * FROM MyLakehouse.dbo.sales
```

🔄 Cross query behavior:

- It reads the **Lakehouse managed table** from its Delta storage
- You can **JOIN** it with existing **Warehouse tables**

Now create tables in warehouse, the below syntax will create the table in warehouse the tables that are already exist in lakehouse

```
CREATE TABLE Pragya_Warehouse.Gold.Fact_Sales
AS SELECT * FROM SilverLakehouse.dbo.sales_managed;
```

Dimensional Modeling

Dimensional modeling is a data modeling technique used primarily for designing data warehouses and analytical systems. It organizes data into structures that are easy to understand, fast to query, and optimized for reporting and analysis.

🧱 Core Concepts of Dimensional Modeling

There are **two main types of tables**:

1. ☑ Fact Table

- Contains **measurable, quantitative data** (facts) about a business process.
- Typically has **foreign keys** pointing to dimension tables.
- Columns: metrics (e.g., SalesAmount, Quantity), foreign keys (e.g., CustomerID, ProductID)

2. ☑ Dimension Table

- Contains **descriptive attributes** related to business entities (who, what, when, where).
- Columns: text or categorical data (e.g., Customer Name, Product Category, Region)

🧩 Common Dimensional Modeling Techniques

1. Star Schema

- Fact table in the center, directly connected to dimension tables.
- Preferred for performance and simplicity.

2. Snowflake Schema

- Dimensions are **normalized** (i.e., broken into sub-dimensions).
- Slightly more complex but saves storage.

3. Slowly Changing Dimensions (SCD)

- Handles historical changes in dimension data (e.g., if a customer changes address).

Both **Star Schema** and **Snowflake Schema** are popular **dimensional modeling** techniques used in data warehousing and BI tools like **Power BI** and **Microsoft Fabric**. They help organize data to make querying and reporting more efficient and understandable.

★ Star Schema

◇ Definition:

A **Star Schema** has a central **Fact Table** connected directly to multiple **Dimension Tables**, resembling a star shape.

🧱 Structure:

- **Fact Table**: Contains numeric metrics (e.g., SalesAmount, Quantity)
- **Dimension Tables**: Contain descriptive attributes (e.g., Product Name, Customer Name)

📊 Example:

```
              DimProduct
                   |
DimDate — FactSales — DimCustomer
                   |
              DimStore
```

✅ Pros:

- Simple and easy to understand
- Fast query performance
- Ideal for BI tools like Power BI

❌ Cons:

- Dimension tables can have redundant data
- Not normalized

❄️ Snowflake Schema

🔹 Definition:

A **Snowflake Schema** is a more **normalized** version of a Star Schema — dimension tables are **split into sub-dimensions**, resembling a snowflake.

🧱 Structure:

- Dimension tables may have **relationships to other dimension tables**
- Reduces redundancy but increases complexity

Example:

```
          DimCategory
                |
            DimProduct
                |
DimDate — FactSales — DimCustomer — DimRegion
                |
            DimStore
```

Pros:

- Saves storage space (less redundancy)
- Better data integrity

Cons:

- More complex joins
- Slower queries compared to star schema
- Not as intuitive for end users

What is a Slowly Changing Dimension?

Slowly Changing Dimensions (SCDs) are a key concept in **dimensional modeling** that deal with how to handle **changes in dimension data over time** — for example, when a customer's address or a product's category changes.

A **dimension** is "slowly changing" when its attributes can change over time. For example:

CustomerID	Name	City
101	John Smith	New York

If John moves to Chicago, how do you track that change?

This is where **SCD types** come in.

🧩 Types of Slowly Changing Dimensions (Most Common)

Type	Name	What it does
SCD Type 1	**Overwrite** the old data	Keeps only the latest data
SCD Type 2	**Track history** using new rows	Maintains full history
SCD Type 3	Track **limited history** in extra columns	Keeps previous + current values only

🔧 Type 1 – Overwrite (No History)

Simple update: Replace old data with new values.

```
-- Before
101, John Smith, New York

-- After Update
101, John Smith, Chicago
```

✅ Easy
❌ Loses historical data

🔁 Type 2 – Full History (Add Row with New Surrogate Key)

Insert a new row for every change, and track using:

- Surrogate key (e.g., `CustomerSK`)
- Effective date columns
- Active flag

CustomerSK	CustomerID	Name	City	IsActive	StartDate	EndDate
1	101	John Smith	New York	No	2022-01-01	2023-01-01
2	101	John Smith	Chicago	Yes	2023-01-01	NULL

✅ Keeps full history
❌ More complex ETL

🧮 Type 3 – Partial History (Extra Columns)

Stores **previous value in a separate column**:

CustomerID	Name	CurrentCity	PreviousCity
101	John Smith	Chicago	New York

✅ Easy to compare current vs previous
❌ Cannot track more than one change

📦 SCD in Microsoft Fabric

In **Microsoft Fabric**, you can implement SCD logic in:

- **Dataflows Gen2** using conditional transformations
- **Notebooks** using PySpark (especially for SCD Type 2)
- **SQL Pipelines** for warehouse-based updates

🧠 When to Use Which?

Business Need	Use SCD Type
Don’t care about history	Type 1
Need full historical tracking	Type 2
Only need last change	Type 3

🧩 What is the Visual Query Editor in Microsoft Fabric?

The **Visual Query Editor** is a **no-code, drag-and-drop interface** that allows you to clean, shape, and transform your data **without writing any code** — similar to Power Query in Power BI.

It is mainly used in:

✅ **Dataflows Gen2**
✅ **In Datawarehouse**
✅ **Power BI (Transform Data)**
✅ **Fabric Data Pipelines (when transforming data)**

In Fabric **Warehouses**, the **Visual Query Editor** lets you visually explore and transform data from your warehouse tables using a **no-code, SQL-powered interface** — great for:

- Building complex T-SQL queries visually
- Creating quick views, filters, joins
- Previewing and validating results interactively