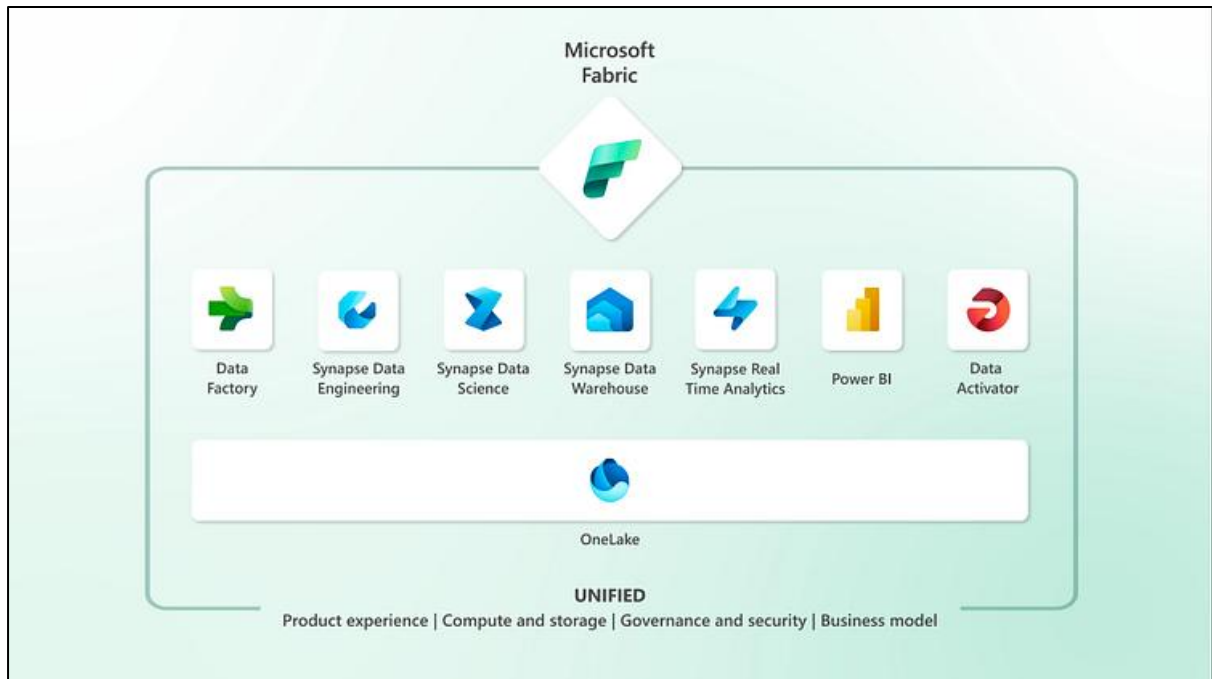


Microsoft Fabric End to End Project



Before getting started, I'd like to introduce some of the important building blocks of Microsoft Fabric.

Microsoft Fabric Terminologies:

- Workspaces: Collaborative environments where you can create and manage items like lakehouses, warehouses, and reports. All data is stored in OneLake and accessed through workspaces.
- The OneLake catalog in Microsoft Fabric helps users easily find and access various data sources within their organization.
- Fabric workloads: To create items in a workload. You have the option to create different items in a workload e.g. for storing, processing, and analyzing data.
- Fabric also supports a data mesh architecture, allowing decentralized data ownership while maintaining centralized governance.
- The default storage format in OneLake is Delta-Parquet.

Fabric Workspace

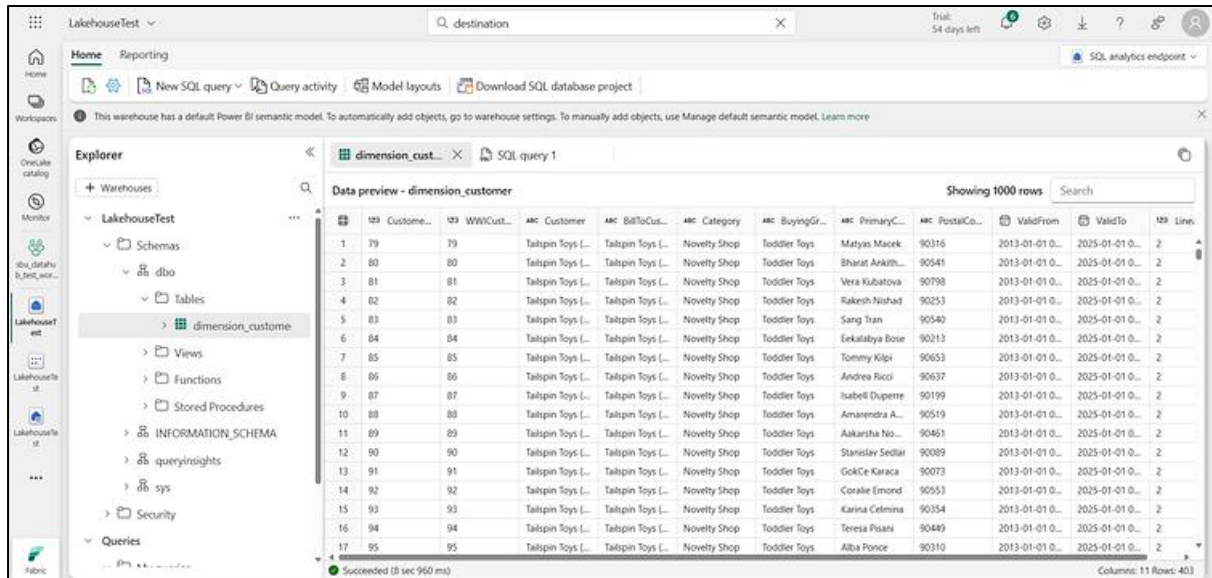
Step 1: Sign in to the [Microsoft Fabric portal](#)

Step 2: Create a new **Workspace**.

Lakehouse: Ingest data into the lakehouse

Step 1: Create a Lakehouse

Step 2: Create a Data Flow Gen 2 item and import the data

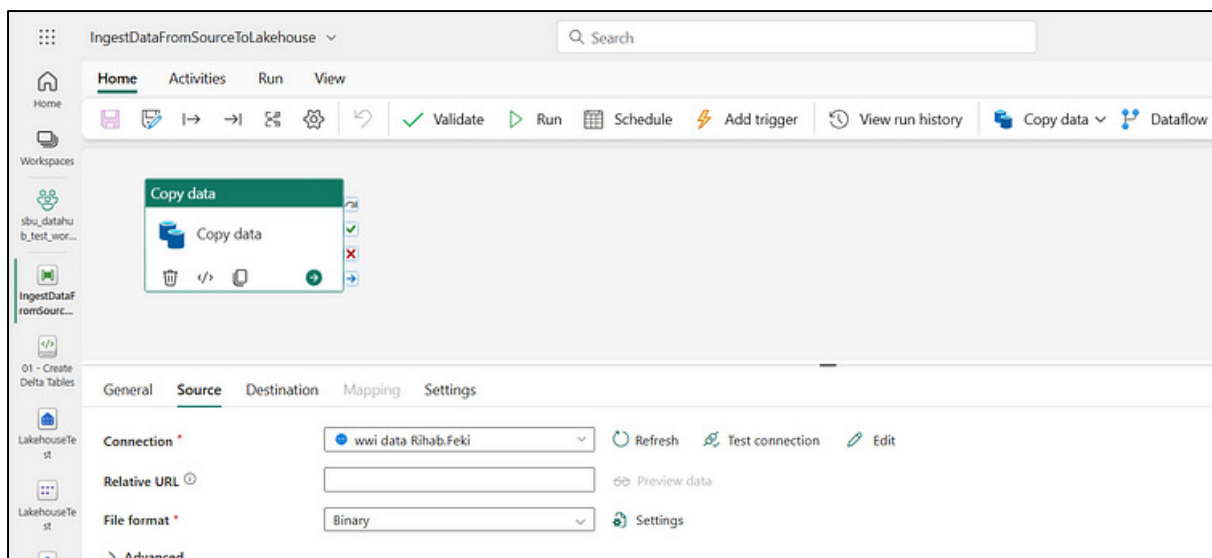


The screenshot shows the LakehouseTest interface with a data preview for 'dimension_customer'. The table has 11 columns: ID, CustomerID, WWCustomerID, Customer, BillToCustomer, Category, BuyingGroup, PrimaryContact, PostalCode, ValidFrom, ValidTo, and Line. The data is displayed in a grid with 17 rows visible.

ID	CustomerID	WWCustomerID	Customer	BillToCustomer	Category	BuyingGroup	PrimaryContact	PostalCode	ValidFrom	ValidTo	Line
1	79	79	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Matyas Maciek	90316	2013-01-01 0...	2025-01-01 0...	2
2	80	80	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Bharat Ankith...	90541	2013-01-01 0...	2025-01-01 0...	2
3	81	81	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Vera Kubatova	90798	2013-01-01 0...	2025-01-01 0...	2
4	82	82	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Rakesh Nishad	90253	2013-01-01 0...	2025-01-01 0...	2
5	83	83	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Sang Tran	90540	2013-01-01 0...	2025-01-01 0...	2
6	84	84	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Eekalabya Bose	90213	2013-01-01 0...	2025-01-01 0...	2
7	85	85	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Tommy Kilpi	90653	2013-01-01 0...	2025-01-01 0...	2
8	86	86	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Andrea Ricci	90637	2013-01-01 0...	2025-01-01 0...	2
9	87	87	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Isabell Duperré	90199	2013-01-01 0...	2025-01-01 0...	2
10	88	88	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Amanendra A...	90519	2013-01-01 0...	2025-01-01 0...	2
11	89	89	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Aakarsha Nis...	90461	2013-01-01 0...	2025-01-01 0...	2
12	90	90	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Stanislav Sedlar	90089	2013-01-01 0...	2025-01-01 0...	2
13	91	91	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Gökçe Karaca	90073	2013-01-01 0...	2025-01-01 0...	2
14	92	92	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Coralie Emmond	90553	2013-01-01 0...	2025-01-01 0...	2
15	93	93	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Karina Celmina	90354	2013-01-01 0...	2025-01-01 0...	2
16	94	94	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Teresa Pisani	90449	2013-01-01 0...	2025-01-01 0...	2
17	95	95	Talpin Toys L...	Talpin Toys L...	Novelty Shop	Toddler Toys	Alba Ponce	90310	2013-01-01 0...	2025-01-01 0...	2

Step 3: Create a Data pipeline to ingest the data

- Add a Copy Data activity and make the following configurations:
- Source: In the connection filed choose an http operator and copy there the link to the source data Then choose Binary as the file format.

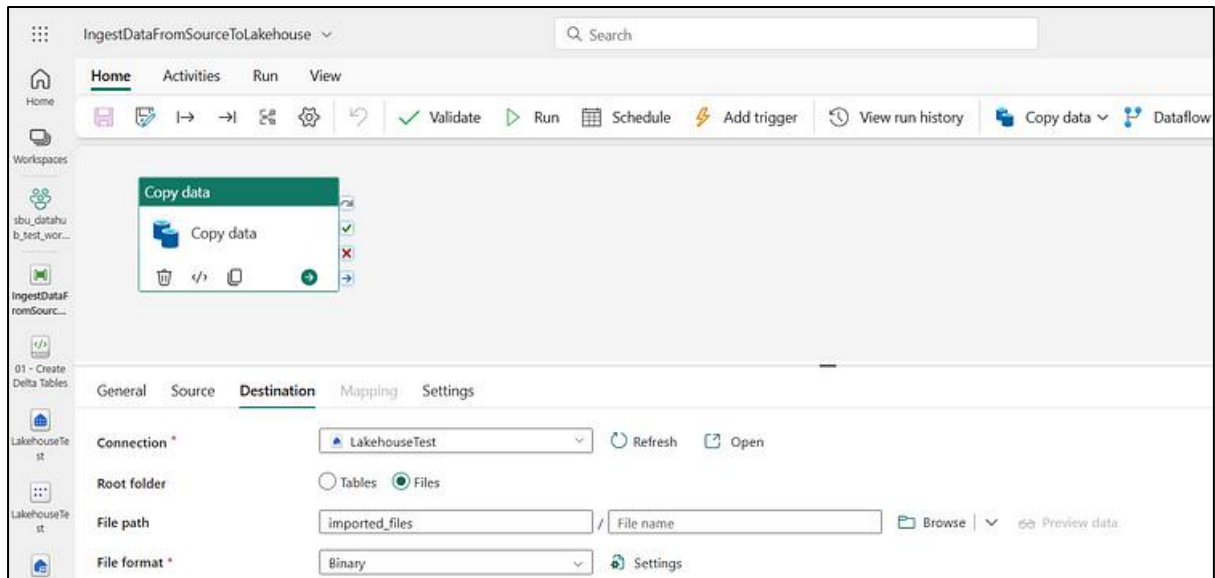


The screenshot shows the 'Copy data' activity configuration in the LakehouseTest interface. The 'Source' tab is selected, and the 'Connection' is set to 'wwi data Rihab.Feki'. The 'File format' is set to 'Binary'. The 'Destination' tab is also visible, showing the 'LakehouseTest' connection and the 'dimension_customer' table.

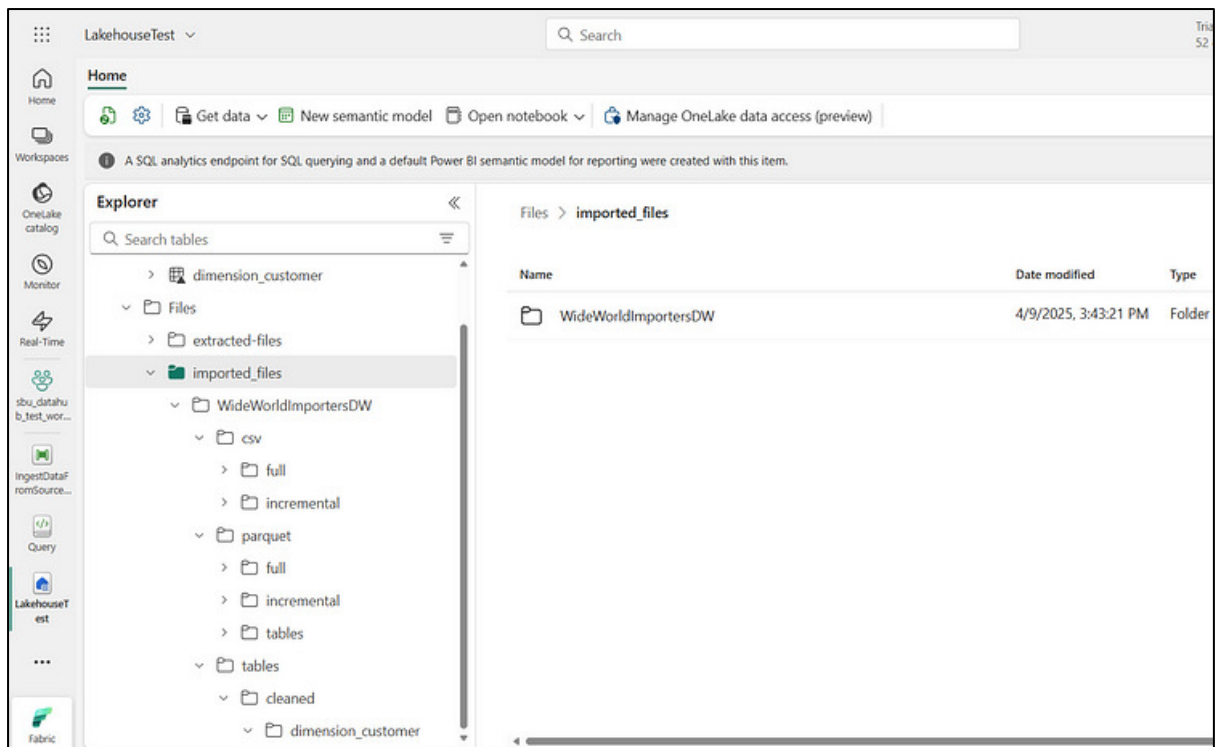
Connection	Relative URL	File format
wwi data Rihab.Feki		Binary

Source configuration

Destination: Choose in the connection the Lakehouse which was created in the previous step. Select the root folder as files. Choose the Binary file format as well.



The Save the configuration and run the pipeline and when it is successfully run, check the files in the Lakehouse.



Prepare and transform data in the Lakehouse

From the previous steps, raw data was ingested from the source to the Files section of the Lakehouse. Now it is possible to transform that data and prepare it for creating Delta tables.

Step 1: Create Delta Tables

- In the workspace, click on open Notebook and select a new Notebook.
- Before writing data as Delta lake tables in the Tables section of the Lakehouse, Two Fabric features are used (V-order and Optimize Write) for optimized data writing and for improved reading performance. To enable these features, add the following configuration in the Notebook:

```
spark.conf.set("spark.sql.parquet.vorder.enabled", "true")
spark.conf.set("spark.microsoft.delta.optimizeWrite.enabled", "true")
spark.conf.set("spark.microsoft.delta.optimizeWrite.binSize", "1073741824")
```

Step 2: Read the data

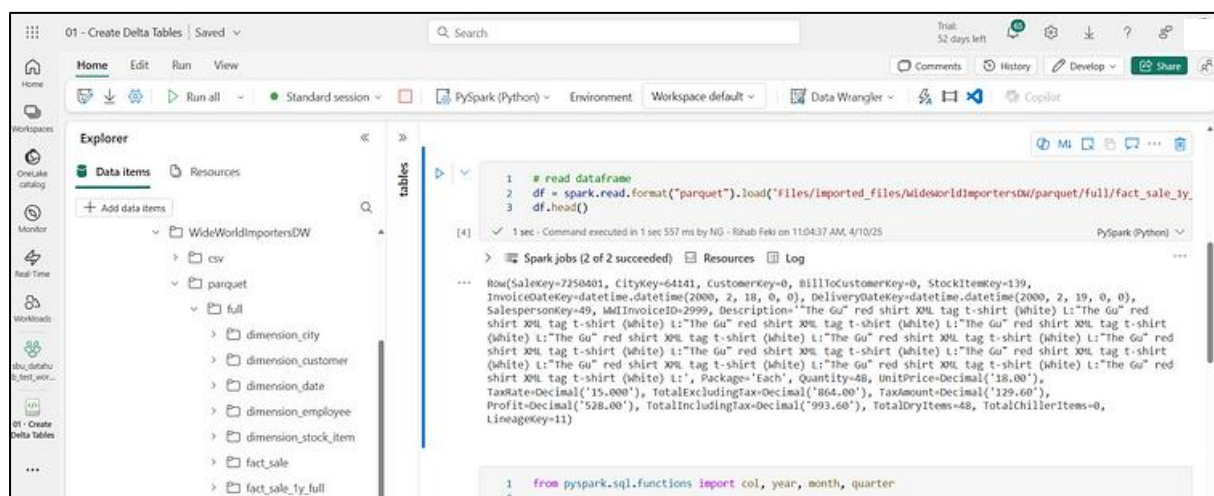
PySpark will be used to read raw data from the Files section of the Lakehouse, and add more columns for different date parts as part of the transformation.

Next, raw data is read from the Files section of the Lakehouse, and more columns are added for different date parts as part of the transformation.

```
# read data as a dataframe
```

```
df =
```

```
spark.read.format("parquet").load("Files/imported_files/WideWorldImportersDW/parquet/full/fact_sale_by_act_sale_1y_full")
df.head()
```



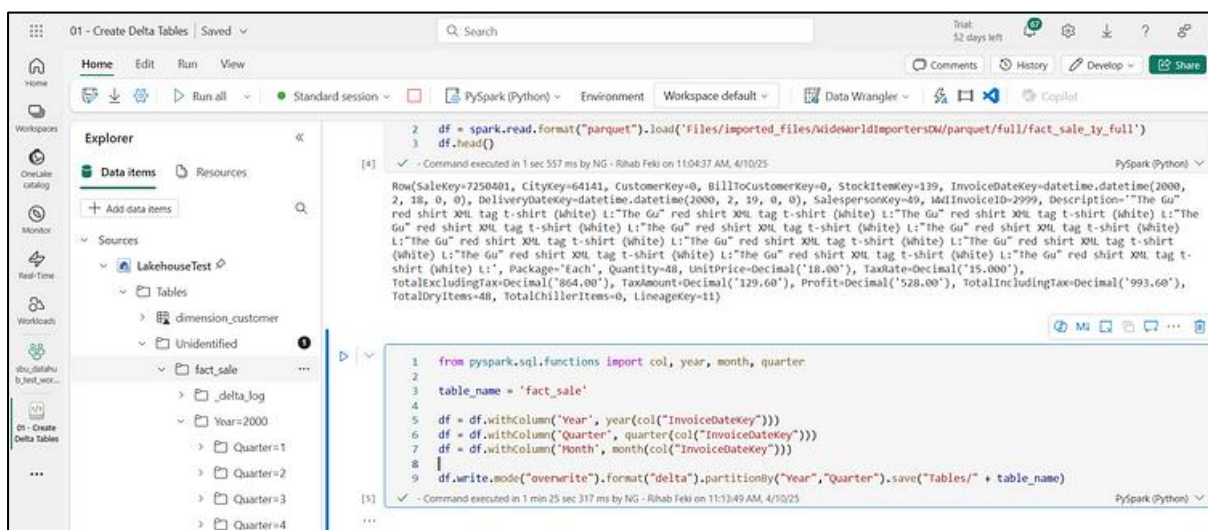
Finally, partition By Spark API is used to partition the data before writing it as Delta table format based on the newly created data part columns (Year and Quarter)

```
from pyspark.sql.functions import col, year, month, quarter
```

```
table_name = 'fact_sale'
```

```
df = df.withColumn('Year', year(col("InvoiceDateKey")))
df = df.withColumn('Quarter', quarter(col("InvoiceDateKey")))
df = df.withColumn('Month', month(col("InvoiceDateKey")))
```

```
df.write.mode("overwrite").format("delta").partitionBy("Year", "Quarter").save("Tables/" +
table_name)
```



Data partitioning and writing fact table to Tables

After the fact tables load, data for the rest of the dimensions is loaded. The following cell creates a function to read raw data from the Files section of the Lakehouse for each of the table names passed as a parameter. Note that the script drops the column named Photo in this example because the column isn't used.

```
from pyspark.sql.types import *
```

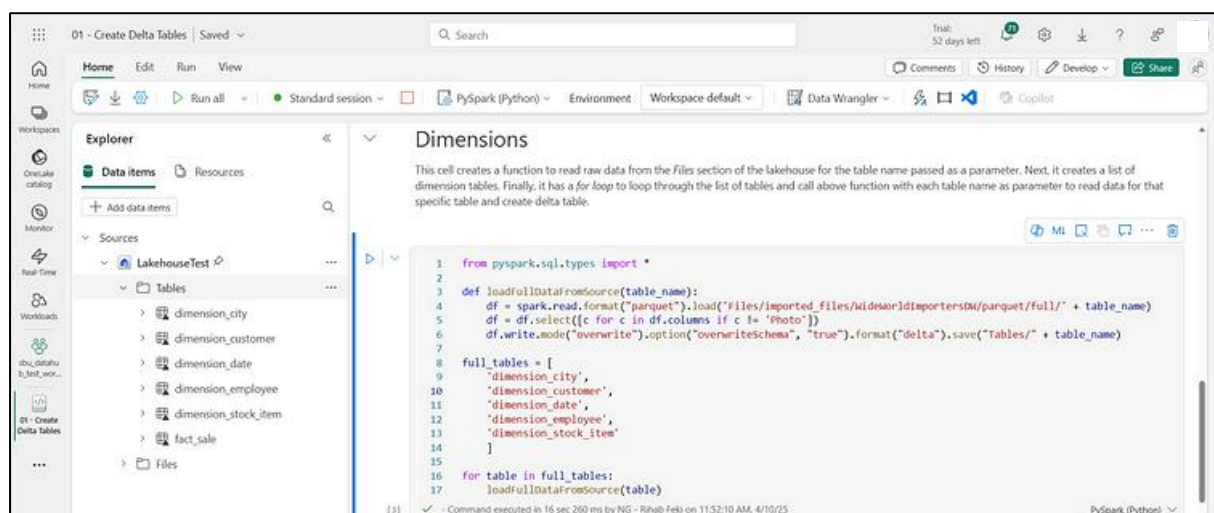
```
def loadFullDataFromSource(table_name):
    df =
spark.read.format("parquet").load('Files/imported_files/WideWorldImportersDW/parquet/full/'
+ table_name)
    df = df.select([c for c in df.columns if c != 'Photo'])
    df.write.mode("overwrite").option("overwriteSchema", "true").format("delta").save("Tables/"
+ table_name)("Tables/" + table_name)
```

Next, Creates a list of dimension tables and loop through the list of tables and create a Delta table for each table name that's read from the input parameter.

```
full_tables = [  
    'dimension_city',  
    'dimension_customer',  
    'dimension_date',  
    'dimension_employee',  
    'dimension_stock_item'  
]
```

```
for table in full_tables:  
    loadFullDataFromSource(table)
```

To validate the created tables, right-click and select refresh on the LakehouseTest. The tables appear, as shown below:



At this stage, we are done loading the data as Tables and can move to the next step to make data transformation depending on the business needs.

Data Transformation — Business aggregates

Approach #1 — Use PySpark

PySpark will be used to join and aggregate data for generating business aggregates.

With the following code, three different Spark dataframes are created, each referencing an existing Delta table.

```
df_fact_sale = spark.read.table("LakehouseTest.fact_sale")
df_dimension_date = spark.read.table("LakehouseTest.dimension_date")
df_dimension_city = spark.read.table("LakehouseTest.dimension_city")
```

In this next step, the tables are joined using the dataframes created earlier, doing group by to generate aggregation, renaming few of the columns and finally writing it as delta table in the *Tables* section of the Lakehouse.

```
sale_by_date_city = df_fact_sale.alias("sale") \
.join(df_dimension_date.alias("date"), df_fact_sale.InvoiceDateKey ==
df_dimension_date.Date, "inner") \
.join(df_dimension_city.alias("city"), df_fact_sale.CityKey == df_dimension_city.CityKey,
"inner") \
.select("date.Date", "date.CalendarMonthLabel", "date.Day", "date.ShortMonth",
"date.CalendarYear", "city.City", "city.StateProvince", "city.SalesTerritory",
"sale.TotalExcludingTax", "sale.TaxAmount", "sale.TotalIncludingTax", "sale.Profit") \
.groupBy("date.Date", "date.CalendarMonthLabel", "date.Day", "date.ShortMonth",
"date.CalendarYear", "city.City", "city.StateProvince", "city.SalesTerritory") \
.sum("sale.TotalExcludingTax", "sale.TaxAmount", "sale.TotalIncludingTax", "sale.Profit") \
.withColumnRenamed("sum(TotalExcludingTax)", "SumOfTotalExcludingTax") \
.withColumnRenamed("sum(TaxAmount)", "SumOfTaxAmount") \
.withColumnRenamed("sum(TotalIncludingTax)", "SumOfTotalIncludingTax") \
.withColumnRenamed("sum(Profit)", "SumOfProfit") \
.orderBy("date.Date", "city.StateProvince", "city.City")
```


Approach #2 — Use Spark SQL

Spark SQL is used to join and aggregate data for generating business aggregates.

In this cell, temporary Spark view is created by joining three tables, group by is done to generate aggregation, and a few of the columns are renamed.

```
%%sql
CREATE OR REPLACE TEMPORARY VIEW sale_by_date_employee
AS
SELECT
    DD.Date, DD.CalendarMonthLabel
, DD.Day, DD.ShortMonth, CalendarYear Year
, DE.PreferredName, DE.Employee
, SUM(FS.TotalExcludingTax) SumOfTotalExcludingTax
, SUM(FS.TaxAmount) SumOfTaxAmount
, SUM(FS.TotalIncludingTax) SumOfTotalIncludingTax
, SUM(Profit) SumOfProfit
FROM LakehouseTest.fact_sale FS
INNER JOIN LakehouseTest.dimension_date DD ON FS.InvoiceDateKey = DD.Date
INNER JOIN LakehouseTest.dimension_Employee DE ON FS.SalespersonKey =
DE.EmployeeKey
GROUP BY DD.Date, DD.CalendarMonthLabel, DD.Day, DD.ShortMonth, DD.CalendarYear,
DE.PreferredName, DE.Employee
ORDER BY DD.Date ASC, DE.PreferredName ASC, DE.Employee ASC
```

In the following cell, we read from the temporary Spark view created in the previous cell and finally write it as a Delta table in the Tables section of the Lakehouse

```
sale_by_date_employee = spark.sql("SELECT * FROM sale_by_date_employee")
sale_by_date_employee.write.mode("overwrite").format("delta").option("overwriteSchema",
"true").save("Tables/aggregate_sale_by_date_employee")
```

to verify the creation of the Delta Table, right click on the Lakehouse and you should be able to see the newly created table:

The screenshot displays the Databricks workspace interface. The top navigation bar shows the workspace name "02 - Data Transformation - Business Aggregates" and a search bar. The left sidebar contains the "Explorer" panel with "Data items" and "Resources" tabs. Under "Data items", the "LakehouseTest" lakehouse is expanded, showing a "Tables" folder containing several tables, including "aggregate_sale_by_date_city". The main notebook area shows a PySpark (Python) session with the following code:

```
1 sale_by_date_city = df_fact_sale.alias("sale") \
2 .join(df_dimension_date.alias("date"), df_fact_sale.InvoiceDateKey == df_dimension_date.Date, "inner") \
3 .join(df_dimension_city.alias("city"), df_fact_sale.CityKey == df_dimension_city.CityKey, "inner") \
4 .select("date.Date", "date.CalendarMonthLabel", "date.Day", "date.ShortMonth", "date.CalendarYear", "city.city", "city.St
5 .groupBy("date.Date", "date.CalendarMonthLabel", "date.Day", "date.ShortMonth", "date.CalendarYear", "city.city", "city.St
6 .sum("sale.TotalExcludingTax", "sale.TaxAmount", "sale.TotalIncludingTax", "sale.Profit")\
7 .withColumnRenamed("sum(TotalExcludingTax)", "SumOfTotalExcludingTax")\
8 .withColumnRenamed("sum(TaxAmount)", "SumOfTaxAmount")\
9 .withColumnRenamed("sum(TotalIncludingTax)", "SumOfTotalIncludingTax")\
10 .withColumnRenamed("sum(Profit)", "SumOfProfit")\
11 .orderBy("date.Date", "city.StateProvince", "city.city")
12
```

Below the code, the execution results are shown. The first command executed successfully in 765 ms. The second command, which writes the table as a Delta table, executed successfully in 32 seconds.

```
1 # write as a delta table the joined and transformed table
2 sale_by_date_city.write.mode("overwrite").format("delta").option("overwriteSchema", "true").save("Tables/aggregate_sale_by
3
```

The execution results show that the table was successfully created as a Delta table.

Build a report with Power BI

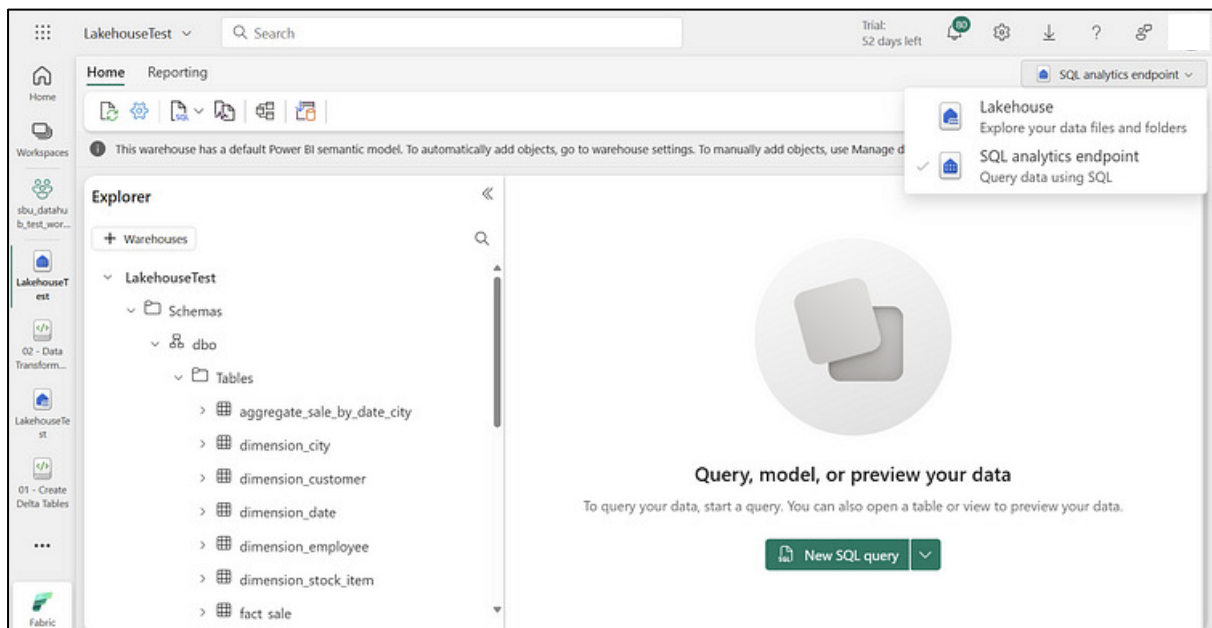
Power BI is natively integrated in the whole Fabric experience.

DirectLake is a groundbreaking new capability that allows you to analyze very large semantic models in Power BI. With DirectLake, you load parquet-formatted files directly from a data lake into memory without needing to query a data warehouse or Lakehouse endpoint, and without needing to import or duplicate data into a Power BI semantic model.

DirectLake mode is the ideal choice for analyzing very large semantic models and semantic models with frequent updates at the source.

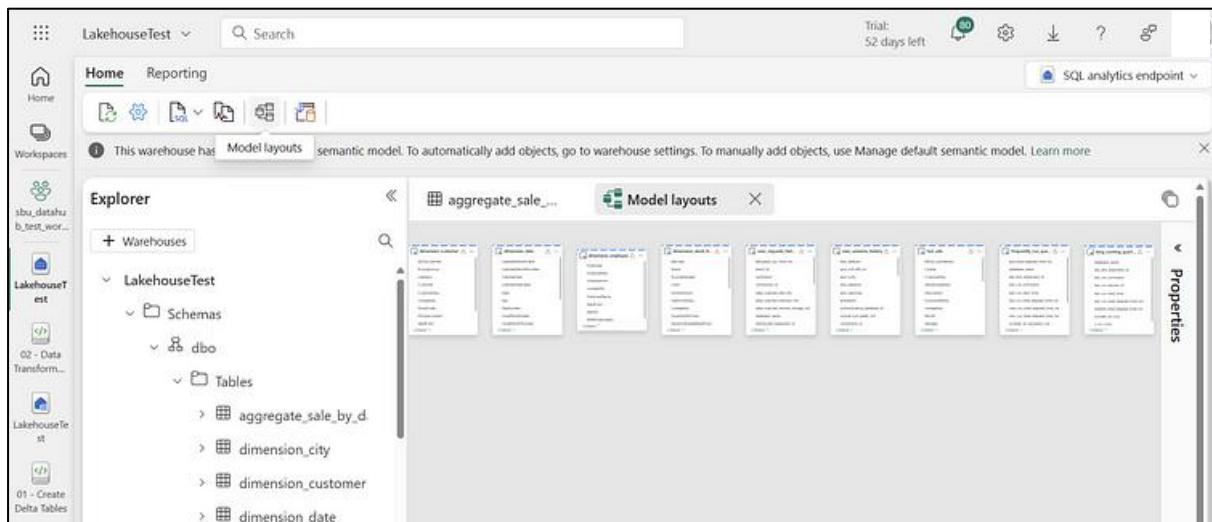
Step 1: Data modeling

Navigate to the Lakehouse and select at the top right button “**SQL analytics endpoint**”



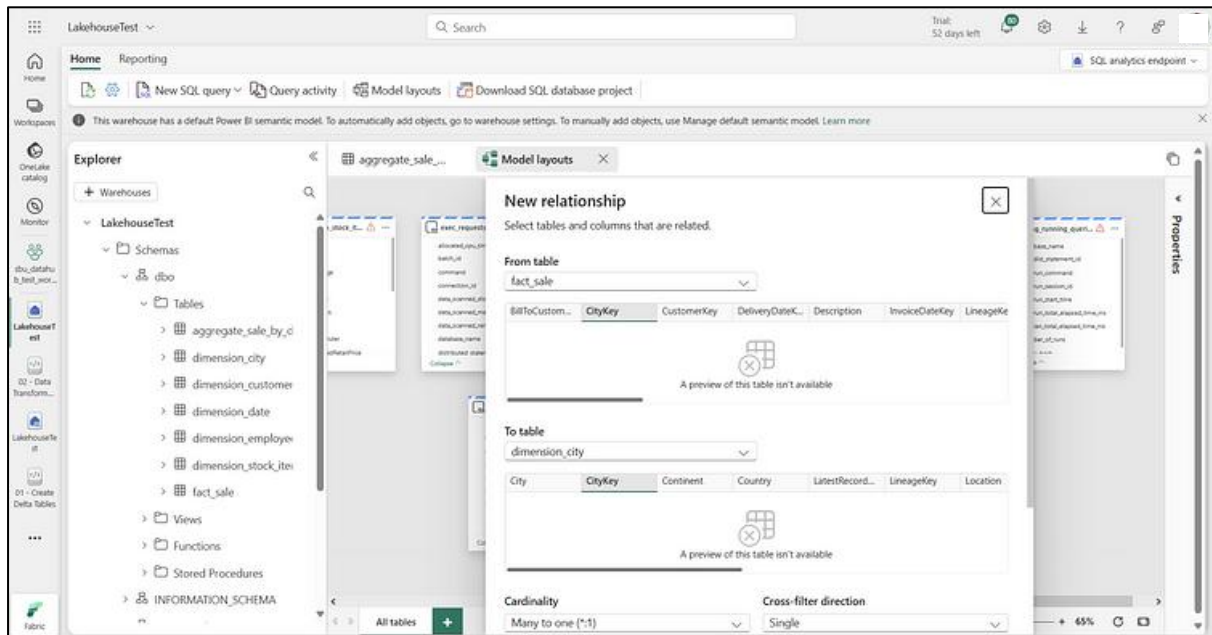
For this data model, the relationship between different tables need to be defined so that you can create reports and visualizations based on data coming across different tables.

Initially, click on the “**model layouts**” button, as shown below:

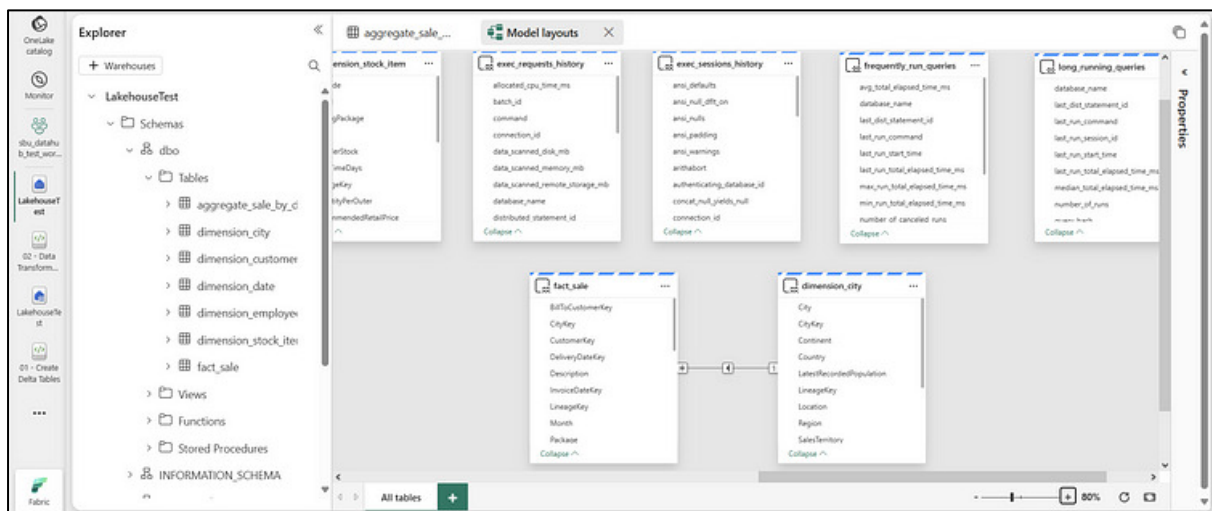


At this stage, the tables are disconnected and needs to be joined in the next step.

To create a relationship, do the following: From the fact_sale table, drag the CityKey field and drop it on the CityKey field in the dimension_city table . The New relationship dialog box appears, as below:

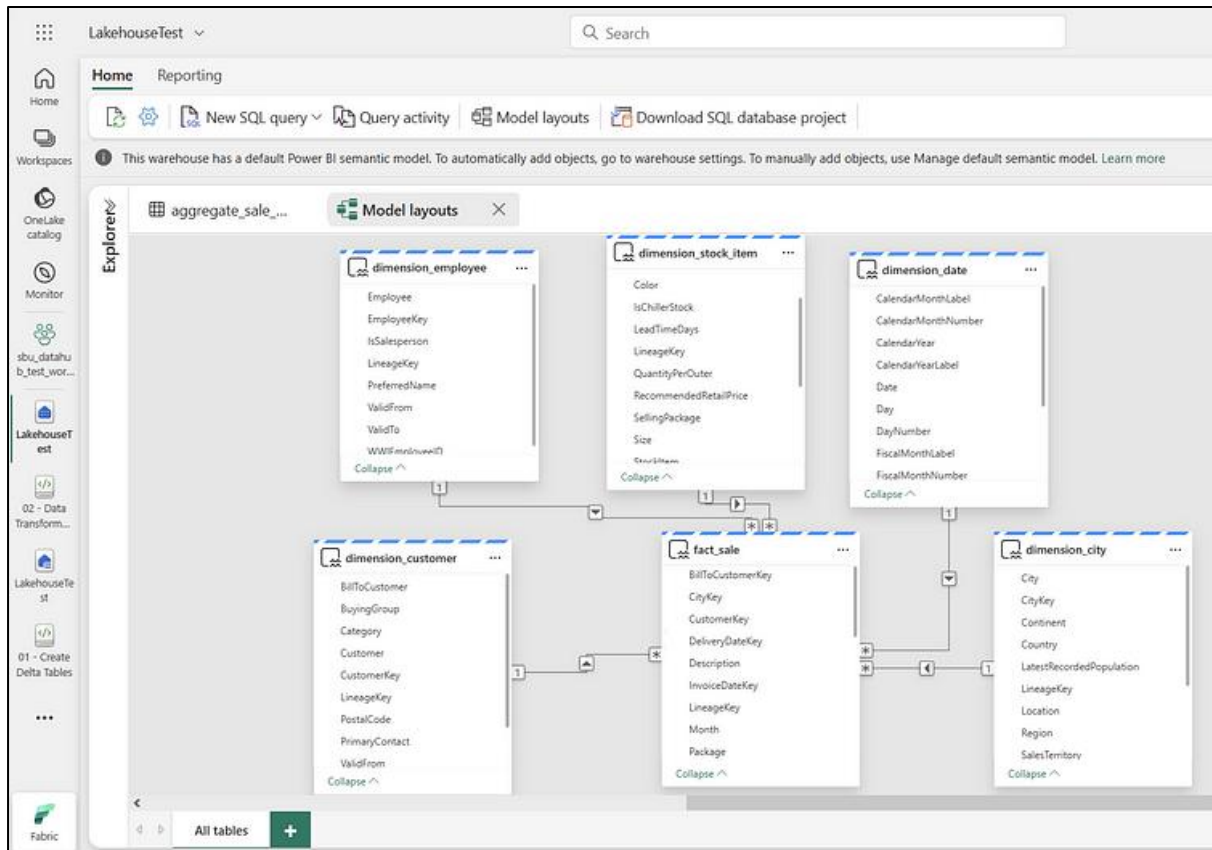


Note: When defining relationships, make sure you have a many to one relationship from the fact_sale table to the dimension_* tables and not vice versa.



Next, add these relationships with the same New relationship settings shown in the previous step, but with the following tables and columns:

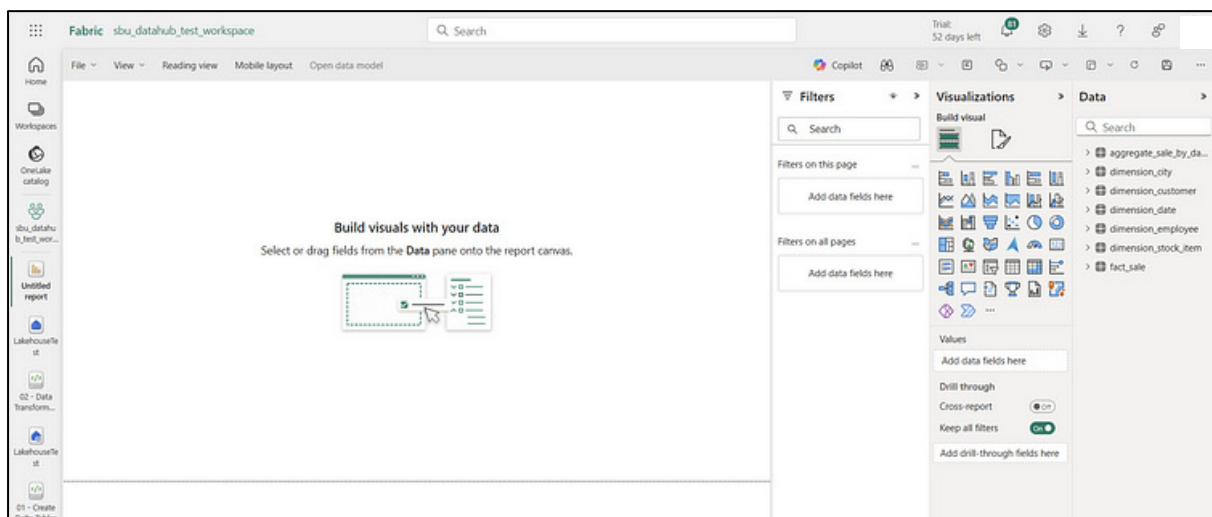
- StockItemKey(fact_sale) — StockItemKey(dimension_stock_item)
- Salespersonkey(fact_sale) — EmployeeKey(dimension_employee)
- CustomerKey(fact_sale) — CustomerKey(dimension_customer)
- InvoiceDateKey(fact_sale) — Date(dimension_date)



After adding these relationships, the data model is ready for reporting.

Step 2: Create a report

Select New report in the reporting tab to start creating reports/dashboards in Power BI, as shown below:



At this stage you can be creative and create a report that responds to business needs.