



Implementing Analytics Solutions Using Microsoft Fabric

Exam Ref DP-600

Daniil Maslyuk
Johnny Winter
Štěpán Rešl

FREE SAMPLE CHAPTER |



Exam Ref DP-600 Implementing Analytics Solutions Using Microsoft Fabric

**Daniil Maslyuk
Johnny Winter
Štěpán Rešl**

Exam Ref DP-600 Implementing Analytics Solutions Using Microsoft Fabric

Published with the authorization of Microsoft Corporation by:
Pearson Education, Inc.

Copyright © 2025 by Pearson Education, Inc.

Hoboken, New Jersey

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/permissions.

No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-13-533602-1

ISBN-10: 0-13-533602-3

Library of Congress Control Number: 2024912313

\$PrintCode

TRADEMARKS

Microsoft and the trademarks listed at <http://www.microsoft.com> on the “Trademarks” webpage are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

WARNING AND DISCLAIMER

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author, the publisher, and Microsoft Corporation shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the programs accompanying it.

SPECIAL SALES

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

CREDITS

EDITOR-IN-CHIEF
Brett Bartow

EXECUTIVE EDITOR
Loretta Yates

ASSOCIATE EDITOR
Shourav Bose

DEVELOPMENT EDITOR
Songlin Qiu

MANAGING EDITOR
Sandra Schroeder

SENIOR PROJECT EDITOR
Tracey Croom

COPY EDITOR
Linda Laflamme

INDEXER
Timothy Wright

PROOFREADER
Donna E. Mulder

TECHNICAL EDITOR
Nuric Ugarte

EDITORIAL ASSISTANT
Cindy Teeters

COVER DESIGNER
Twist Creative, Seattle

COMPOSITOR
codeMantra

GRAPHICS
codeMantra

FIGURE CREDIT
Figure 2.38 The Apache
Software Foundation

Contents at a glance

	<i>Introduction</i>	<i>xiii</i>
CHAPTER 1	Plan, implement, and manage a solution for data analytics	1
CHAPTER 2	Prepare and serve data	61
CHAPTER 3	Implement and manage semantic models	143
CHAPTER 4	Explore and analyze data	261
CHAPTER 5	Exam DP-600: Implementing Analytics Solutions Using Microsoft Fabric updates	299
	<i>Index</i>	<i>305</i>

Contents

Introduction	xiii
<i>Organization of this book</i>	<i>xiii</i>
<i>Preparing for the exam</i>	<i>xiii</i>
<i>Microsoft certifications</i>	<i>xiv</i>
<i>Access the exam updates chapter and online references</i>	<i>xiv</i>
<i>Errata, updates, & book support</i>	<i>xv</i>
<i>Stay in touch</i>	<i>xv</i>
 Chapter 1 Plan, implement, and manage a solution for data analytics	 1
Skill 1.1: Plan a data analytics environment	1
Identify requirements for a solution, including components, features, performance, and capacity stock-keeping units (SKUs)	2
Recommend settings in the Fabric admin portal	4
Choose a data gateway type	14
Create a custom Power BI report theme	20
Skill 1.2: Implement and manage a data analytics environment.	25
Implement workspace- and item-level access controls for Fabric items	26
Implement data sharing for workspaces, warehouses, and lakehouses	28
Manage Sensitivity Labels in semantic models and lakehouses	30
Configure Fabric-enabled workspace settings	33
Manage Fabric capacity	37
Skill 1.3: Manage the analytics development lifecycle	39
Implement version control for a workspace	40
Create and manage a Power BI Desktop project (.pbip)	42
Plan and implement deployment solutions	44
Perform impact analysis of downstream dependencies from lakehouses, data warehouses, dataflows, and semantic models	47

Deploy and manage semantic models by using the XMLA endpoint	50
Create and update reusable assets, including Power BI template (.pbit) files, Power BI data source (.pbids) files, and shared semantic models	52
Chapter summary	55
Thought experiment	57
Thought experiment answers	58

Chapter 2 Prepare and serve data 61

Skill 2.1: Create objects in a lakehouse or warehouse	61
Ingest data by using a data pipeline, dataflow, or notebook	62
Create and manage shortcuts	74
Implement file partitioning for analytics workloads in a lakehouse	76
Create views, functions, and stored procedures	79
Enrich data by adding new columns or tables	84
Skill 2.2: Copy data	87
Choose an appropriate method for copying data from a Fabric data source to a lakehouse or warehouse	87
Copy data by using a data pipeline, dataflow, or notebook	89
Implement fast copy when using dataflows	95
Add stored procedures, notebooks, and dataflows to a data pipeline	99
Schedule data pipelines	105
Schedule dataflows and notebooks	106
Skill 2.3: Transform data	108
Implement a data cleansing process	109
Implement a star schema for a lakehouse or warehouse, including Type 1 and Type 2 slowly changing dimensions	111
Implement bridge tables for a lakehouse or warehouse	113
Denormalize data	116
Aggregate or de-aggregate data	117
Merge or join data	120
Identify and resolve duplicate data, missing data, or null values	122

Convert data types by using SQL or PySpark	125
Filter data	128
Skill 2.4: Optimize performance	132
Identify and resolve data loading performance bottlenecks in dataflows, notebooks, and SQL queries	133
Implement performance improvements in dataflows, notebooks, and SQL queries	133
Identify and resolve issues with the structure or size of Delta table files (including V-Order and optimized writes)	135
Chapter summary	137
Thought experiment	139
Thought experiment answers	141

Chapter 3 Implement and manage semantic models 143

Skill 3.1: Design and build semantic models	143
Choose a storage mode, including Direct Lake	144
Identify use cases for DAX Studio and Tabular Editor 2	145
Implement a star schema for a semantic model	148
Implement relationships, such as bridge tables and many-to-many relationships	148
Write calculations that use DAX variables and functions, such as iterators, table filtering, windowing, and information functions	150
Implement calculation groups, dynamic strings, and field parameters	231
Design and build a large-format semantic model	236
Design and build composite models that include aggregations	238
Implement dynamic row-level security and object-level security	240
Validate row-level security and object-level security	246
Skill 3.2: Optimize enterprise-scale semantic models	248
Implement performance improvements in queries and report visuals	248
Improve DAX performance by using DAX Studio	249
Optimize a semantic model by using Tabular Editor 2	250
Implement incremental refresh	251

Chapter summary	256
Thought experiment.....	258
Thought experiment answers	260
Chapter 4 Explore and analyze data	261
Skill 4.1: Perform exploratory analytics	261
Implement descriptive and diagnostic analytics	261
Integrate prescriptive and predictive analytics into a visual or report	266
Profile data	270
Skill 4.2: Query data by using SQL	281
Query a lakehouse in Fabric using SQL queries or the visual query editor	282
Query a warehouse in Fabric using SQL queries or the visual query editor	288
Connect to and query datasets by using the XMLA endpoint	290
Chapter summary	295
Thought experiment.....	296
Thought experiment answers	297
Chapter 5 Exam DP-600: Implementing Analytics Solutions Using Microsoft Fabric updates	299
The purpose of this chapter	299
About possible exam updates	300
Impact on you and your study plan	300
News and commentary about the exam objective updates.....	300
Updated technical content.....	301
Objective mapping	301
<i>Index</i>	<i>305</i>

Acknowledgments

Daniil Maslyuk

I'd like to acknowledge the team at Pearson who made this book happen, including Loretta Yates, who trusted us to write this book, and Shourav Bose, who managed the project. Songlin, Nuric, and Linda, the editors, made this book a better read. And I'd like to thank my co-authors, Johnny and Štěpán, without whom I wouldn't have been able to write this book.

Johnny Winter

I'd like to thank my co-authors Daniil and Štěpán for their ongoing advice and support, and Daniil in particular for inviting me to join the project. Thanks to our sponsoring editor Shourav for his support and patience. Special thanks to my wife Amanda for being supportive and allowing me the time and space to complete my contributions to the book, which often ate into our spare time. I'd also like to thank my employers, Advancing Analytics, a great bunch of colleagues and, dare I say it, friends. They have supported me in attending Fabric community events, given me the opportunity to get hands on with Fabric and learn the platform end to end, and supported me with my Fabric Analytics Engineer certification.

Štěpán Rešl

I want to acknowledge my friends and colleagues from DataBrothers for their endless support and encouragement in co-authoring this book and for allowing me to share my thoughts and ideas: Adam, Róza, Míra, Janek, and Lukáš. I especially want to thank my brother Matěj, who stood behind me all the time and helped me organize everything in my work schedule so that I could participate in this book. I also would like to thank my co-authors Daniil and Johnny for their support, cheerful mindset, and advice during this whole project.

About the Authors

DANIIL MASLYUK is an independent business intelligence consultant, trainer, and speaker who specializes in Microsoft Power BI. Daniil blogs at xxlbi.com and tweets as @DMaslyuk.

JOHNNY WINTER is a data and analytics consultant who has been working with business intelligence software since 2007, specializing in the Microsoft data platform since 2016. He's a self-confessed business intelligence geek, and in his spare time runs the website and YouTube channel *Greyskull Analytics*, where he likes to nerd out about all things analytics.

ŠTĚPÁN REŠL is a lead technical consultant and a Microsoft MVP in the Data Platform category. As a technical consultant, Štěpán focuses on assisting medium and large organizations in deploying and maintaining their data solutions. He is also a speaker and co-organizer of conferences. In his spare time, he runs a blog called *DataMeerkat*, where he focuses on topics related to data analytics.

Introduction

This book covers all the skills measured in the exam DP-600: Implementing Analytics Solutions Using Microsoft Fabric. In each chapter, you'll find a combination of step-by-step instructional content and related high-level theoretical material. The aim is to show you the settings you need to select and buttons you need to click to carry out the tasks required, as well as to cover key concepts that you need to understand when designing an analytics solution. Ultimately, we cover not only the how, but also the why.

This book covers every major topic area found on the exam, but it does not cover every exam question. Only the Microsoft exam team has access to the exam questions, and Microsoft regularly adds new questions to the exam, making it impossible to cover specific questions. You should consider this book a supplement to your relevant real-world experience and other study materials. If you encounter a topic in this book that you do not feel completely comfortable with, use the "Need more review?" links you'll find in the text to find more information and take the time to research and study the topic.

Organization of this book

This book is organized by the "Skills measured" list published for the exam. The "Skills measured" list is available for each exam on the Microsoft Learn website: microsoft.com/learn. Each chapter in this book corresponds to a major topic area in the list, and the technical tasks in each topic area determine a chapter's organization. If an exam covers six major topic areas, for example, the book will contain six chapters.

Preparing for the exam

Microsoft certification exams are a great way to build your resume and let the world know about your level of expertise. Certification exams validate your on-the-job experience and product knowledge. Although there is no substitute for on-the-job experience, preparation through study and hands-on practice can help you prepare for the exam. This book is *not* designed to teach you new skills.

We recommend that you augment your exam preparation plan by using a combination of available study materials and courses. For example, you might use the *Exam Ref* and another study guide for your at-home preparation and take a Microsoft Official Curriculum course for the classroom experience. Choose the combination that you think works best for you. Learn more about available classroom training, online courses, and live events at microsoft.com/learn.

Note that this *Exam Ref* is based on publicly available information about the exam and the authors' experience. To safeguard the integrity of the exam, authors do not have access to the live exam.

Microsoft certifications

Microsoft certifications distinguish you by proving your command of a broad set of skills and experience with current Microsoft products and technologies. The exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop, or implement and support, solutions with Microsoft products and technologies both on-premises and in the cloud. Certification brings a variety of benefits to the individual and to employers and organizations.

MORE INFO ALL MICROSOFT CERTIFICATIONS

For information about Microsoft certifications, including a full list of available certifications, go to microsoft.com/learn.

Access the exam updates chapter and online references

The final chapter of this book, "Exam DP-600: Implementing Analytics Solutions Using Microsoft Fabric updates," will provide information about changes to content, such as additions of new exam topics, removal of content from the exam objectives, and revised mapping of exam objectives to chapter content. The chapter will be made available from the link below as exam updates are released.

Throughout this book are addresses to webpages that the author has recommended you visit for more information. Some of these links can be very long and painstaking to type, so we've shortened them for you to make them easier to visit. We've also compiled them into a single list that readers of the print edition can refer to while they read.

The URLs are organized by chapter and heading. Every time you come across a URL in the book, find the hyperlink in the list to go directly to the webpage.

Download the Exam Updates chapter and the URL list at MicrosoftPressStore.com/ERDP600/downloads.

Errata, updates, & book support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

MicrosoftPressStore.com/ERDP600/errata

If you discover an error that is not already listed, please submit it to us at the same page.

For additional book support and information, please visit *MicrosoftPressStore.com/Support*.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *support.microsoft.com*.

Stay in touch

Let's keep the conversation going! We're on X/Twitter: *twitter.com/MicrosoftPress*.

Prepare and serve data

For data to be used to get answers, it needs to be modeled, prepared, cleaned, orchestrated, and provided so that the right users can get to it when they need it. As part of the preparation process, you often need to decide what transformation tool to use, whether to duplicate the data, how the partitions should be created, which transformations to use, and how to control the transformations to ensure the correct continuity of individual operations between individual states. At the same time, you want to undertake these steps via the most efficient means possible, ensuring optimal usage of your Microsoft Fabric resources.

Skills covered in this chapter:

- Skill 2.1: Create objects in a lakehouse or warehouse
- Skill 2.2: Copy data
- Skill 2.3: Transform data
- Skill 2.4: Optimize performance

Skill 2.1: Create objects in a lakehouse or warehouse

With data coming from various sources and in multiple formats, you need somewhere to store it so that it can then be processed into the appropriate form, format, and style of storage most suitable for its subsequent use. You need to get the data into a unified OneLake environment, which then makes it possible to use no-code, low-code, and even full-code transformations to process the data. Once the data is in OneLake, it is not duplicated in multiple places, because OneLake uses shortcuts to point to specific locations of data rather than creating additional instances of that data when it's needed by other items.

This skill covers how to:

- Ingest data by using a data pipeline, dataflow, or notebook
- Create and manage shortcuts
- Implement file partitioning for analytics workloads in a lakehouse
- Create views, functions, and stored procedures
- Enrich data by adding new columns or tables

Ingest data by using a data pipeline, dataflow, or notebook

Fabric provides three basic ways to retrieve data from existing storage and systems: *data pipelines*, *dataflows*, and *notebooks*. Each item uses a different user approach and targets different types of users.

Data pipelines

A *data pipeline* is an item from the Data Factory experience that acts as an orchestration component. It can run other items and services and be scheduled to run at specific times. To start creating a pipeline, select **New > Data pipeline**, give the pipeline a name, then select **Create**. In Figure 2-1, you can see the blank canvas of the data pipeline editor.

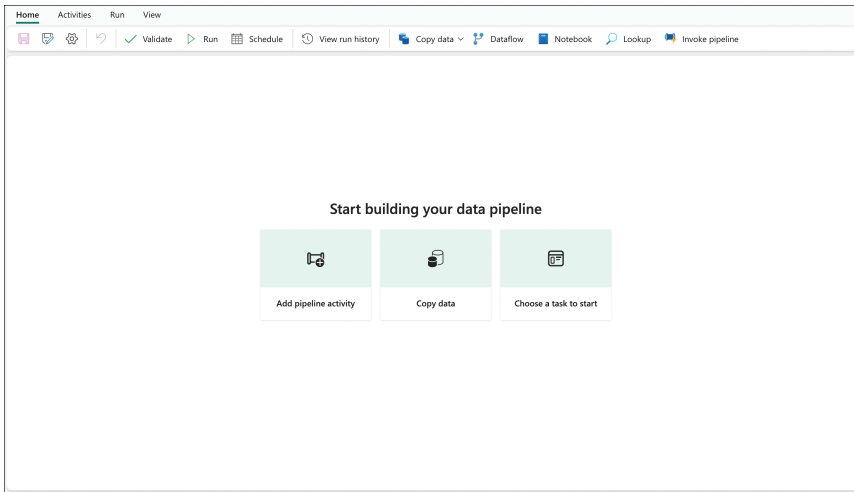


FIGURE 2-1 Blank data pipeline canvas in the data pipeline editor

Within the Fabric services, the **Copy Activity** feature uses the **Data Movement service** and allows you to get data from child nodes, bring it to the Fabric environment, and save it. You can specify the data be saved in the original data format or first converted to another format or directly to Delta Parquet tables within Lakehouse Explorer. You can set up **Copy Activity** by selecting either:

- **Add pipeline activity** (works on the canvas)
- **Copy data** (launches a wizard)

Neither method requires code from you to retrieve the data and convert it to the desired format. In addition to lakehouses, **Copy Activity** can also work and ingest data from other Fabric items, such as warehouses.

NOTE CSV FILE SAMPLE

If you want to try the following examples on your own, you can find the CSV file at github.com/tirnovar/dp-600/blob/main/data/sales.

EXAMPLE OF HOW TO INGEST DATA TO LAKEHOUSE BY PIPELINE

To upload data, open or create a pipeline in the workspace. In this pipeline, you can ingest your data by following these steps:

1. Select **Copy data** from the blank canvas (Figure 2-2).

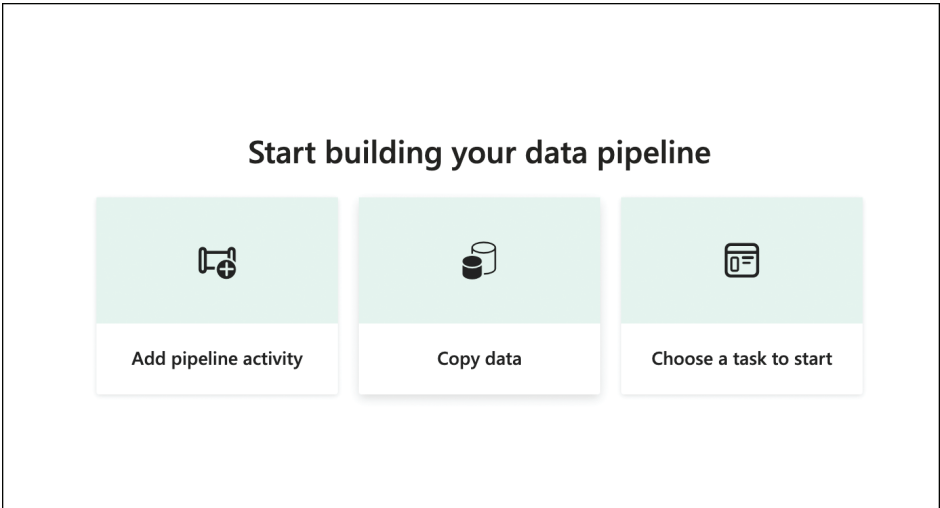


FIGURE 2-2 Quick actions appear only if the pipeline is empty.

2. Search and choose data sources, such as **Azure Data Lake Storage Gen2** shown in Figure 2-3.

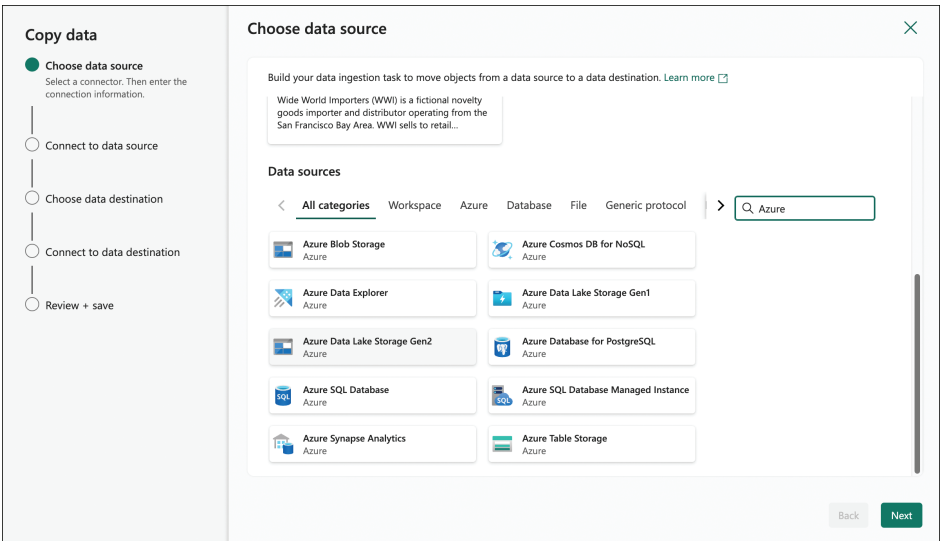


FIGURE 2-3 Filtered data sources

3. Select **Create a new connection** (Figure 2-4), fill in your URL, and sign in.

Azure Data Lake Storage Gen2
Learn more

☐ Existing connection
 ☒ Create new connection

Connection settings

URL * ⓘ
Example: https://contosoadlscdm.dfs.core.windows.net/file...

Connection credentials

Connection
Create new connection

Connection name
https://.dfs.core.windows.net/

Authentication kind
Organizational account

You are not signed in. Please sign in.
Sign in

FIGURE 2-4 Create a new connection. Blank fields in the wizard preview require values in the proper format to help you.

- Choose data to import. For example, Figure 2-5 shows a file named Sales selected.

Copy data

- Choose data source
- Connect to data source**
Select, preview, and choose the data.
- Choose data destination
- Connect to data destination
- Review + save

Connect to data source

Select a file system or file

Search

- dummy-data
- assets
- employees
- sales**

Preview data: sales

☐ Schema agnostic (binary copy) ⓘ

File format
DelimitedText

Column delimiter
Comma (,)

Row delimiter
Line feed (\n)

☒ First row as header ⓘ

> Advanced

	abc: ORDERNUMBER	abc: QUANTITYORDERED	abc: PRICEEACH	abc: ORDERLINENUMBER	abc: SALES	abc: ...
1	10107	30	95.7	2	2871	2/24
2	10121	34	81.35	5	2765.9	5/7/
3	10134	41	94.74	2	3884.34	7/1/
4	10145	45	83.26	6	3746.7	8/25
5	10159	49	100	14	5205.27	10/1

Back

Next

FIGURE 2-5 CSV preview of a selected file

- Select **Lakehouse** as a data destination. You can use the search feature to quickly find a specific data destination (Figure 2-6).

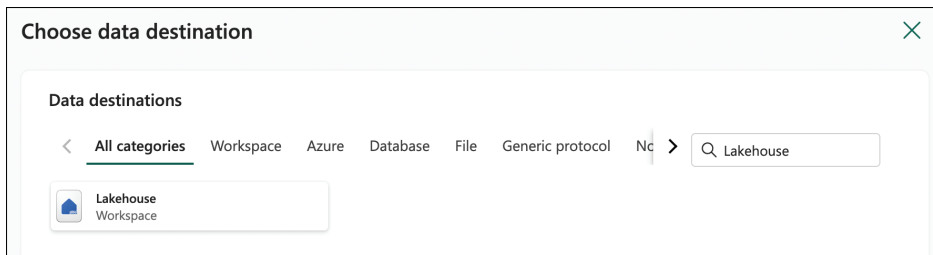


FIGURE 2-6 Searching for lakehouse in data destinations

6. Select the existing lakehouse or create a new one (Figure 2-7).

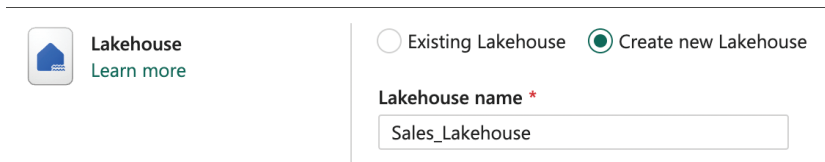


FIGURE 2-7 Creating a new lakehouse

7. Map data to columns in a table or create a new table and define column names and their data types (Figure 2-8).

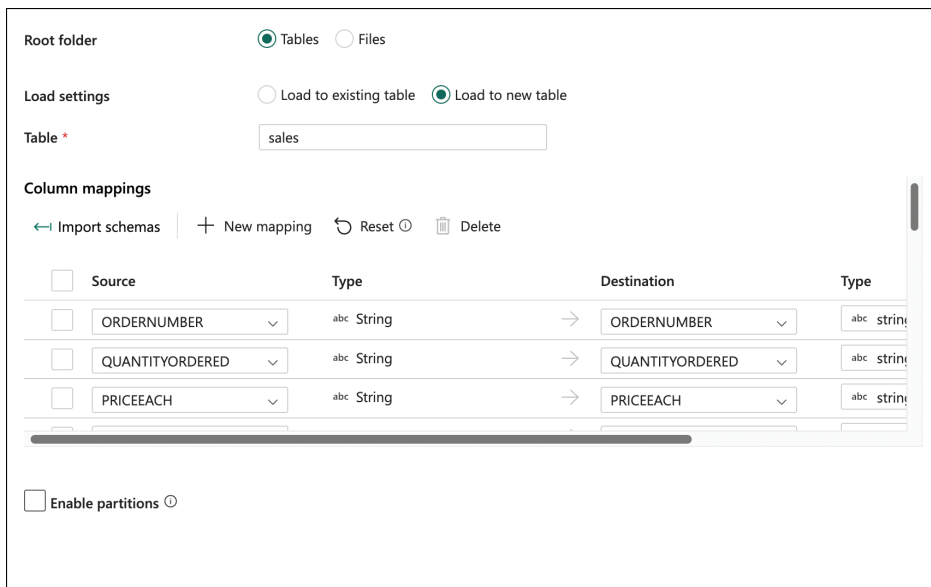


FIGURE 2-8 Preview of column mapping

8. Select **Save + Run**.

The pipeline will be immediately blocked among the items to be launched. After a while, the status of the ongoing data migration and the operation's result will be displayed.

Dataflows

An item that uses Power Query Online, DataFlow Gen2 allows you to use all existing data connectors, including a connection to on-premises data using an on-premises data gateway. Figure 2-9 shows how DataFlow Gen2 looks when you open it for the first time.

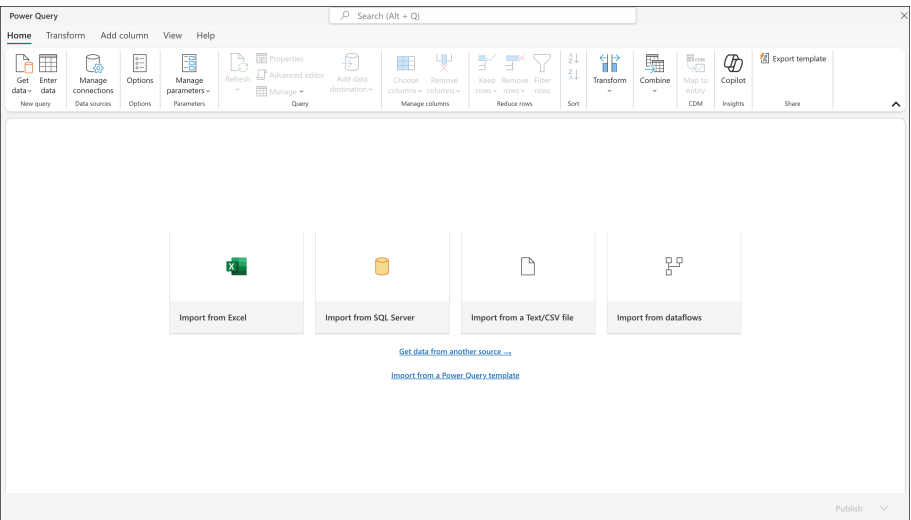


FIGURE 2-9 Empty canvas of DataFlow Gen2

While working with data, DataFlow Gen2 uses two additional items for staging: An automatically generated lakehouse serves as **Staging Storage**, and a warehouse serves as **Staging Compute** and can be used to write to the data destination. Figure 2-10 illustrates the process.

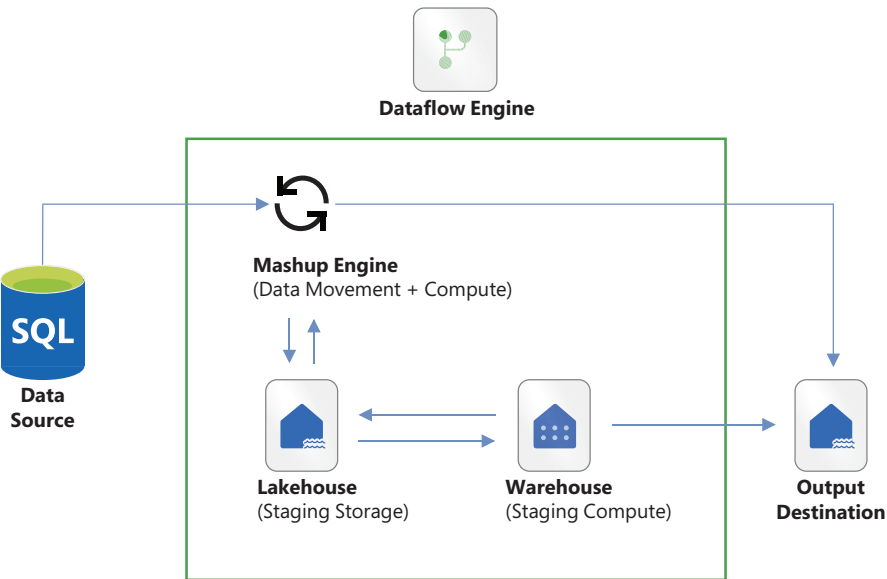


FIGURE 2-10 Dataflow engine schema

When data enters the dataflow engine, the mashup engine based on staging can distribute the data to Staging Storage and Staging Compute or directly transform it and then save it to a destination. You can set data destinations within dataflows separately for each query. Your current destination choices are lakehouse, warehouse, **Azure SQL** database, and Azure Data Explorer (Kusto). If queries have disabled staging, these items are unused, and everything is calculated in memory (Figure 2-11), which on a smaller data sample can better impact the consumed CUs capacity and, simultaneously, the speed. If you have a larger sample of data or a sample requiring more transformations and even combining data from different sources, then the impact can be precisely the opposite.

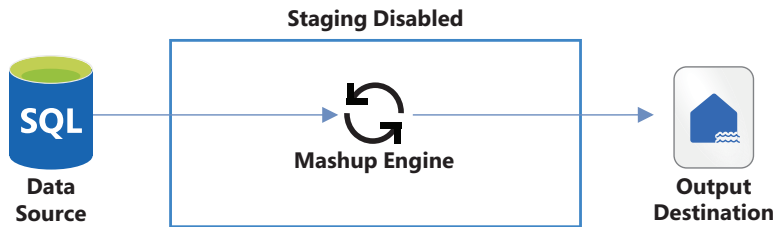


FIGURE 2-11 Disabled staging

Because dataflows use Power Query Online, you can create transformations using a graphical interface or directly with the M language in **Advanced editor**, **Script view**, or **Step script**.

NEED MORE REVIEW? M FORMULA LANGUAGE

For a definition of M formula language, please visit learn.microsoft.com/powerquery-m.

EXAMPLE OF HOW TO INGEST DATA TO LAKEHOUSE BY DATAFLOWS

To open a new DataFlow Gen2:

1. Select the “Get data from another source” link in DataFlow Gen2.
2. Search for **Azure Data Lake Storage Gen2** (Figure 2-12) and select it from the **New sources** section.
3. Create a new connection in a new window (Figure 2-13), select **Next**, and then select **Create**.

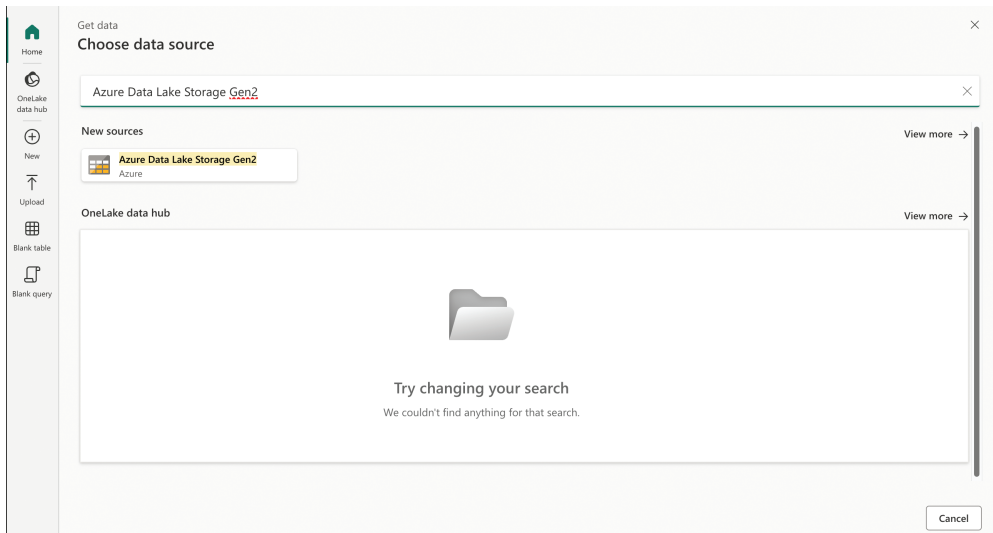


FIGURE 2-12 Choose a data source wizard

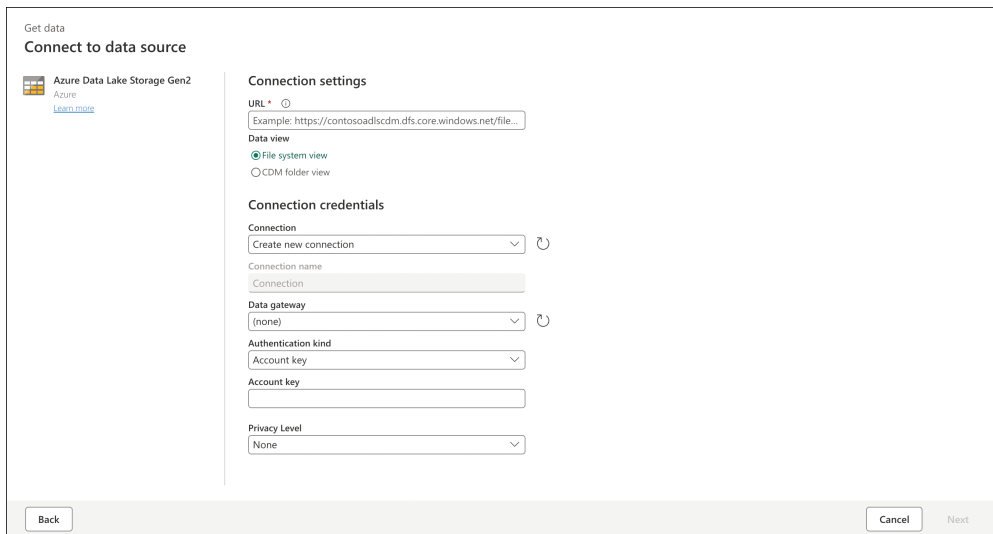


FIGURE 2-13 Connection settings

4. Filter the files. If you have the file names, you can filter by the **Name** column in Figure 2-14; otherwise, you can use the column **Folder Path** to select the data container/folder destination.
5. If you are selecting just one file, you can directly select the value **[Binary]** in the Content column, and Power Query will extract data for you. Otherwise, use the **Combine** icon next to the column name (two arrows pointing down), set the file origin if necessary, and select **OK**. Figure 2-15 shows a preview of the data.

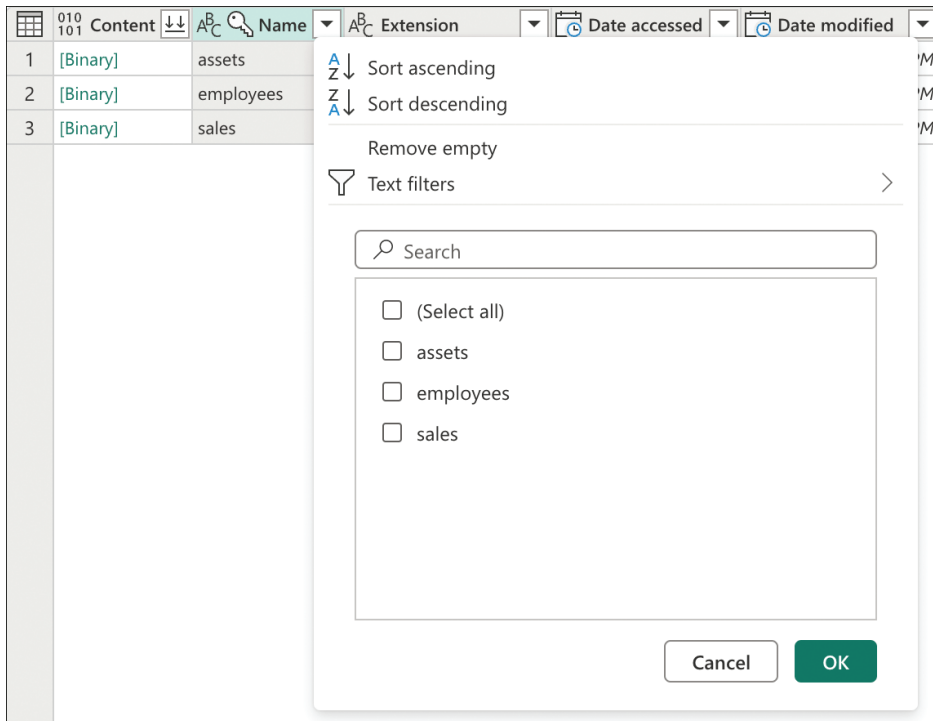


FIGURE 2-14 Filter using the Name column.

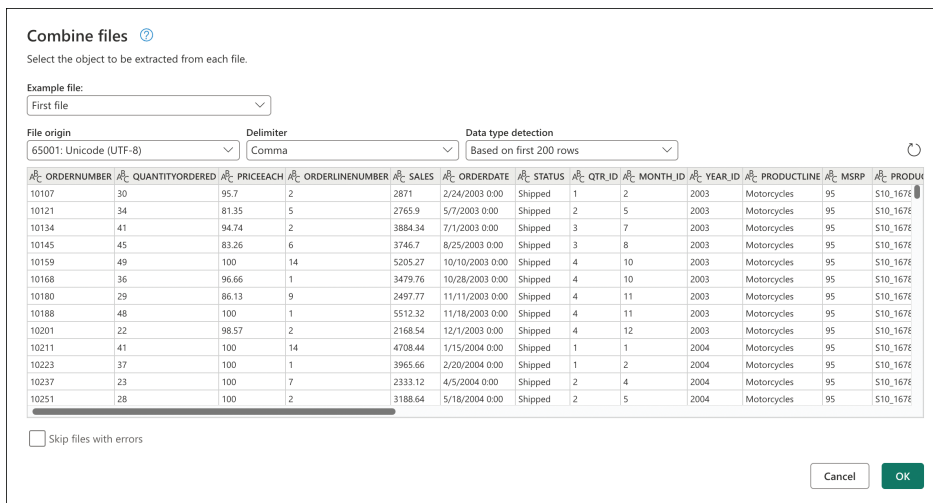


FIGURE 2-15 Data preview

6. Prepare the data, select the **plus** icon (right corner) to add a data destination, and then choose **Lakehouse** (Figure 2-16).

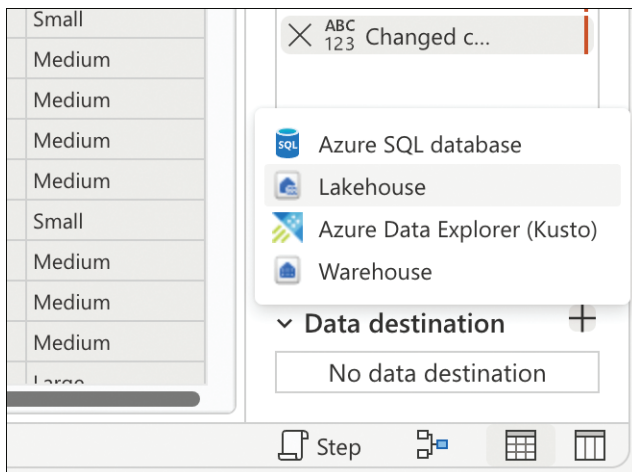


FIGURE 2-16 Possible data destinations

7. Create your connection for all lakehouses, or use the one you already have.
8. Search for your lakehouse (Figure 2-17), and choose a table to insert data or create a new one.

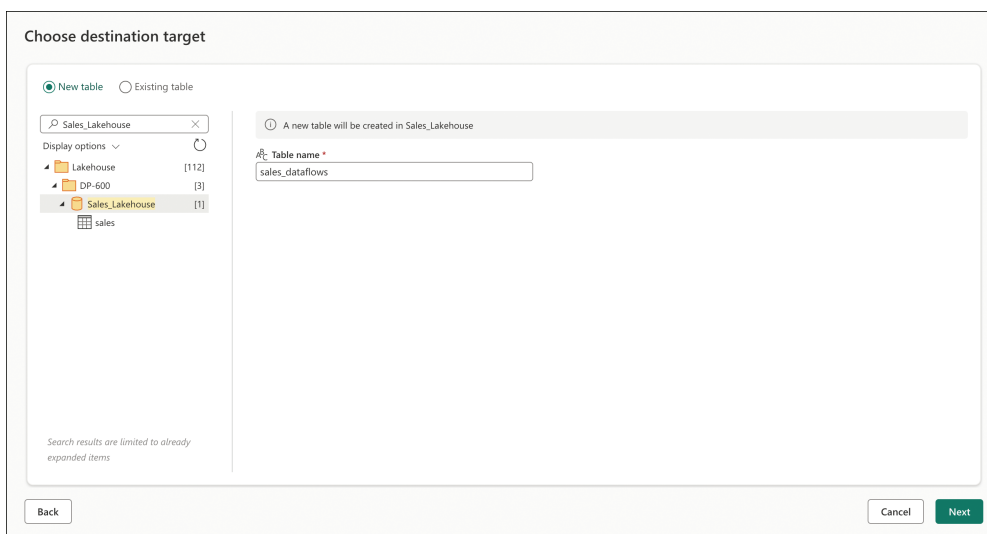


FIGURE 2-17 Destination target

9. Disable **Use automatic settings**. Select **Replace** and **Dynamic schema**. Set data types of all inserted columns or select columns that will be used with selected data types as a schema for the new table. You can see the full settings in Figure 2-18.

Choose destination settings

☐ Use automatic settings

Update method

Existing data → New data → Append → Replace

Schema options on publish

Existing schema → Dynamic schema → Fixed schema

Column mapping

Source	Source type	Destination	Destination type
<input checked="" type="checkbox"/> ORDERNUMBER	1,2 Decimal number	ORDERNUMBER	Decimal number
<input checked="" type="checkbox"/> QUANTITYORDERED	123 Whole number	QUANTITYORDERED	Whole number
<input checked="" type="checkbox"/> PRICEEACH	\$ Currency	PRICEEACH	Currency
<input checked="" type="checkbox"/> ORDERLINENUMBER	123 Whole number	ORDERLINENUMBER	Whole number
<input checked="" type="checkbox"/> SALES	123 Whole number	SALES	Whole number

Back Cancel Save settings

FIGURE 2-18 Destination settings

10. Disable staging of the query by right-clicking a query in the **Queries** pane and toggling off **Enable staging**.
11. Publish the dataflow, and refresh it.

The table is created and filled with data as soon as the dataflow is updated. You can update your dataflow manually, or you can set regular updates. Because users often need to branch out individual transformation processes, timing all items separately can be problematic. That's why using the orchestration capabilities of a data pipeline is good, as they can also run dataflows.

Notebooks

Notebooks are items that are used primarily for creating Apache Spark jobs and machine learning experiments. A notebook itself does not allow you to perform data transformations using the UI. Instead, you must use one of the supported scripting languages:

- PySpark (Python)
- Spark SQL
- Spark (Scala)
- SparkR (R)

You can use these languages in individual code cells that can be executed independently regardless of their order or run sequentially. If a notebook is started using a pipeline, for example, then all cells are executed in their order. Individual code cells can reuse previous cells' variables and outputs, so combining individual scripting languages is possible to obtain a result. In addition to these languages, you can also use Markdown notepads. However, it is possible to create notes in code cells according to the rules of the chosen language.

Notebooks are extended with the **Data Wrangler** tool, which allows you to perform transformation and explorer operations with data using a graphical interface similar to Power Query. It currently allows editing data loaded as **pandas DataFrame** and **Spark DataFrame**.

Notebooks allow the use of many libraries, which are ready-made collections of code for the user. You can use three types of libraries:

- **Built-in** These are pre-installed libraries for each Fabric Spark runtime, according to its settings. For specific details, consult “Apache Spar Runtimes in Fabric” at learn.microsoft.com/en-us/fabric/data-engineering/runtime.
- **Public** These libraries are stored in public repositories like PyPI or Conda. Public libraries must be installed within individual notebook runs or in advance in the runtime via a custom environment or workspace default environment.
- **Custom** These are libraries created within the organization or provided by any developer. You can use .whl libraries for Python, .jar for Java, or .tar.gz for R.

You can use the code below to inline call pieces of libraries for notebook purposes. The first line imports the full library, and the second imports only specifically named functions from a library:

```
import {name-of-package-from-library} [as {user-defined-name-of-package}]
from {name-of-package-from-library} import {name-of-function}
```

Also, thanks to libraries, notebooks can get data from a large number of source locations and can also get it to a lot of destinations. Thus, notebooks use Fabric capacity for their operation, and the admin should monitor this use of capacity to prevent a possible shortage.

NEED MORE REVIEW? DATA WRANGLER

To find more information about Data Wrangler, please read “How to accelerate Data Prep with Data Wrangler in Microsoft Fabric” at learn.microsoft.com/fabric/data-science/data-wrangler.

EXAMPLE OF HOW TO INGEST DATA TO A LAKEHOUSE BY A NOTEBOOK

Open a blank notebook, and follow these steps:

1. Insert the following code into the first cell:

```
azure_data_lake_gen2_name = "<name-of-your-ADLG2>"
blob_container_name = "<container-name>"
file_name = "<file-name>"
path = f'abfss://{blob_container_name}@{azure_data_lake_gen2_name}.dfs.core.windows.net/{file_name}'
```

2. Fill variables by your content.
3. Create a new **Code cell**.
4. Insert the following code:

```
df = spark.read.format('csv').options(header='True', inferSchema='True').load(path)
```

5. Add a lakehouse by selecting **Lakehouse** in Explorer.
6. Select **Add** in the left of the window (Figure 2-19).

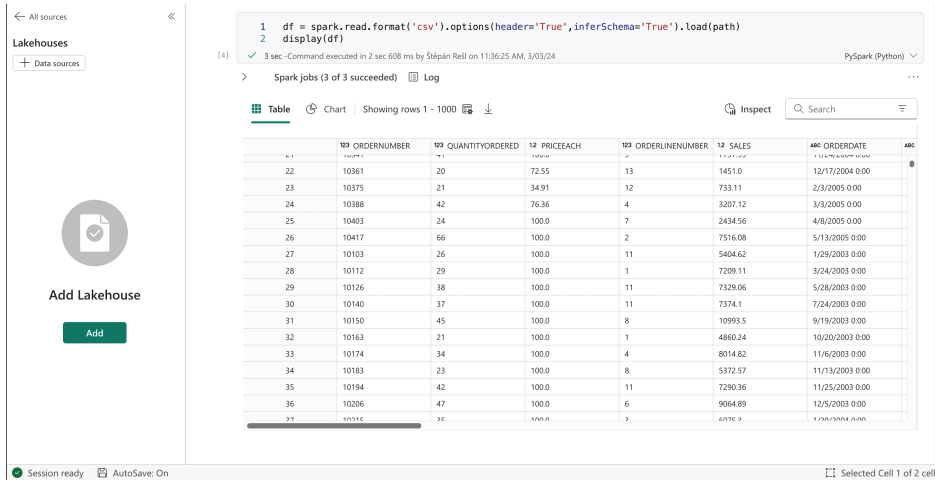


FIGURE 2-19 Preview of data in a notebook

- In the popup, choose if you want to create a new lakehouse or use an existing one (Figure 2-20).

Add Lakehouse

☐ New lakehouse
 ☒ Existing lakehouse

FIGURE 2-20 Add lakehouse options

- Your decision in step 7 will stop the current Spark session. You need to confirm this by selecting the **Stop now** button.
- Create a new **Code cell**, and insert the following code:


```
df.write.mode("overwrite").format("delta").saveAsTable('salesByNotebook')
```
- Select **Run all**.

Once selected, the lakehouse will create a new table named `salesbyNotebook` with the schema defined by the data frame. In addition, you can use the function `saveAsTable` to save; the input would look like `save('Table/salesByNotebook')`. There is often no need to overwrite all data stored in tables, so you can use `mode('append')` just to add new rows. If you want to save data not as a table but as a file, you can use `save('Files/<name-of-folder-for-files>')`. The result would then look like:

```
df.write.format("csv").save("Files/SalesData")
df.write.format("parquet").save("Files/SalesData")
```

Create and manage shortcuts

Shortcuts are objects in OneLake that point to other storage locations. They appear as folders in OneLake; any experience or service with access to OneLake can use them. OneLake shortcuts behave similarly to Microsoft Windows shortcuts. They're independent objects from the target to which they are just pointing. If you delete a shortcut, the target remains unaffected. The shortcut can break if you move, rename, or delete a target path.

Shortcuts can be created in **lakehouse** or **KQL (Kusto Query Language)** databases, and you can use them as data directly in OneLake. Any Fabric service can use them without necessarily copying data directly from a data source. Shortcuts can be created as:

- Table shortcut
- File shortcut

Thanks to the ability to create shortcuts with data stored directly in OneLake, you can reuse data between lakehouses stored in different workspaces. These shortcuts can be generated from a **lakehouse**, **warehouse**, or **KQL** database. They can also access data for notebook transformations or other Fabric items.

EXAMPLE OF HOW TO CREATE A SHORTCUT INSIDE A LAKEHOUSE

You can create a shortcut if you own a lakehouse by following these steps:

1. Open Lakehouse Explorer.
2. Right-click a directory within the **Explorer** pane, or select the **ellipsis** icon that appears when you hover over the Tables or Files main folder.
3. Select **New shortcut** (Figure 2-21).

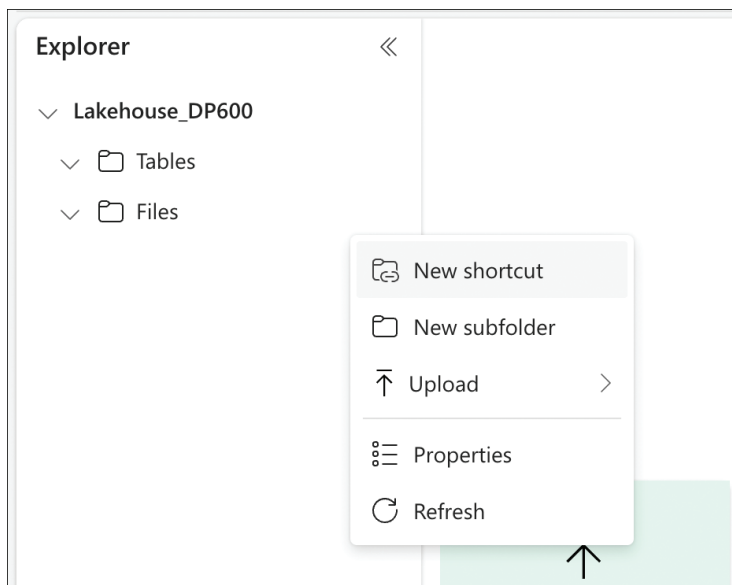


FIGURE 2-21 Creating a new shortcut

4. Select a source of data, such as **Azure Data Lake Storage Gen2** (Figure 2-22).

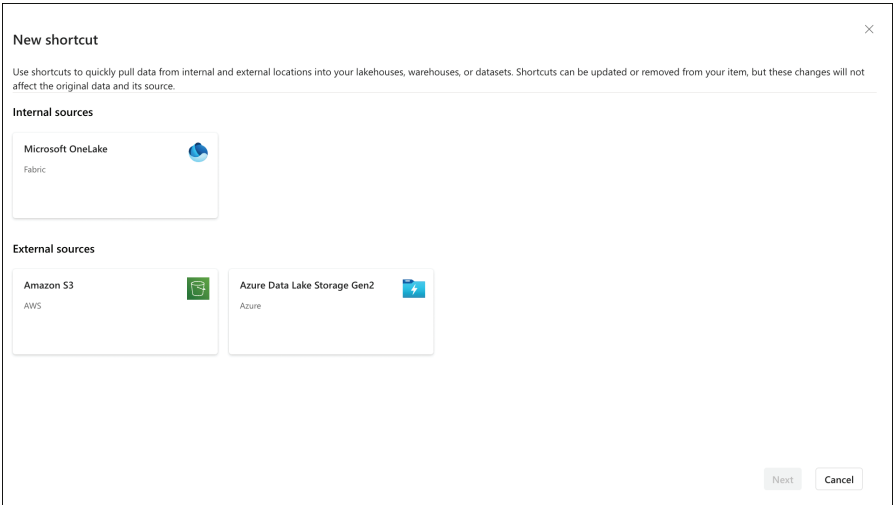


FIGURE 2-22 Shortcut wizard

5. Fill in the connection settings.
6. Name the shortcut, and set the subpath to your data.
7. Select the new shortcut folder to preview the data (Figure 2-23).

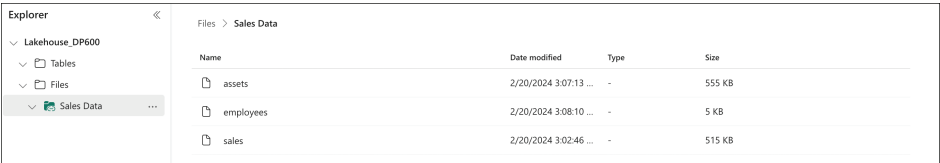


FIGURE 2-23 Data preview

As you can see in Figure 2-24, shortcuts are indicated by a folder icon similar to the one used for Tables and Files but with an added link symbol. This icon is attached to the original icon and can recognize data connected as shortcuts. To delete a shortcut, select the **ellipsis** icon displayed after hovering over the shortcut name and select the **Delete** option.

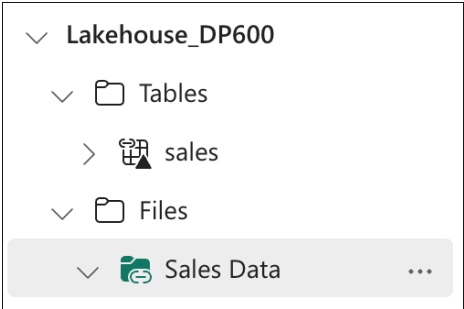


FIGURE 2-24 Icon previews

Implement file partitioning for analytics workloads in a lakehouse

A crucial technique in data management, *data partitioning* involves dividing a large dataset into smaller, more manageable subsets known as *partitions*. Each partition holds a portion of the data, which can be stored and processed independently.

Partitions are represented as folders that contain Parquet files that all meet the same partition condition. A partition condition uses data in a selected column (or columns) because multiple partitions are supported. Based on them, you can create a partition folder with an exact name pattern: `<partition-name>=<value>`. For example, Figure 2-25 shows a preview of COUNTRY partitions. Note that a partition folder must contain at least one file; empty partition folders are automatically removed.

The image is a screenshot of a file explorer interface showing a directory named 'sales (file view)'. It contains a table with four columns: 'Name', 'Date modified', 'Type', and 'Size'. The table lists ten folders, each representing a country partition: COUNTRY=Australia, COUNTRY=Austria, COUNTRY=Belgium, COUNTRY=Canada, COUNTRY=Denmark, COUNTRY=Finland, COUNTRY=France, COUNTRY=Germany, COUNTRY=Ireland, and COUNTRY=Ireland. Each folder is marked as a 'Folder' type and contains '1 items'. The 'Date modified' column shows timestamps for each folder, all starting with '3/10/2024 8:29:51 ...'.

Name	Date modified	Type	Size
COUNTRY=Australia	3/10/2024 8:29:51 ...	Folder	1 items
COUNTRY=Austria	3/10/2024 8:29:52 ...	Folder	1 items
COUNTRY=Belgium	3/10/2024 8:29:53 ...	Folder	1 items
COUNTRY=Canada	3/10/2024 8:29:52 ...	Folder	1 items
COUNTRY=Denmark	3/10/2024 8:29:53 ...	Folder	1 items
COUNTRY=Finland	3/10/2024 8:29:52 ...	Folder	1 items
COUNTRY=France	3/10/2024 8:29:51 ...	Folder	1 items
COUNTRY=Germany	3/10/2024 8:29:53 ...	Folder	1 items
COUNTRY=Ireland	3/10/2024 8:29:54 ...	Folder	1 items

FIGURE 2-25 Deployed COUNTRY partitions

Not every column can be used as a partition column, because partition columns must have one of the following data types:

- String
- Integer
- Boolean
- DateTime

If a column contains empty values, one more partition with a condition equal to `__HIVE_DEFAULT_PARTITION__` will be created.

Delta tables are also filed by composition so that the same principle can be applied to them. However, the **Copy Activity** options within pipelines, DataFlow Gen2, and notebooks currently allow you to create partitions using **Copy Activity** (only for tables) and notebook (for tables and files).

EXAMPLE OF IMPLEMENTING PARTITIONS BY COPY ACTIVITY IN A PIPELINE

Open the new data pipeline in the same workspace as a lakehouse, which will be used as a data destination, and then follow these steps:

1. Add a **Copy Activity** by selecting **Add pipeline activity** > **Copy data**. Under **Source**, select **Sample dataset**, select **Browse**, and then choose a dataset, such as NYC Taxi – Green (Parquet), as shown in Figure 2-26.

The screenshot shows the 'Source' tab of a configuration window. It has five tabs: 'General', 'Source' (active), 'Destination', 'Mapping', and 'Settings'. Under 'Data store type', there are three radio buttons: 'Workspace', 'External', and 'Sample dataset' (which is selected). Below this, under 'Sample dataset', there is a dropdown menu showing 'NYC Taxi - Green (Parquet)', a 'Browse' button with a folder icon, and a 'Preview data' button with a document icon.

FIGURE 2-26 Inserting a sample dataset as a data store type

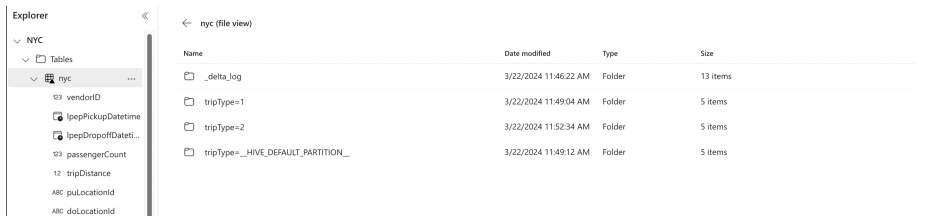
2. Under **Destination**, select a destination **lakehouse**.
3. Open **Advanced options**.
4. Enable **partitions**.
5. Add partitions columns (Figure 2-27).

The screenshot shows the 'Destination' tab of the configuration window. It has five tabs: 'General', 'Source', 'Destination' (active), 'Mapping', and 'Settings'. Under 'Data store type', there are two radio buttons: 'Workspace' (selected) and 'External'. Below this, under 'Workspace data store type', there is a dropdown menu showing 'Lakehouse'. Under 'Lakehouse', there is a dropdown menu showing 'NYC', and next to it are 'Refresh', 'Open', and 'New' buttons. Under 'Root folder', there are two radio buttons: 'Tables' (selected) and 'Files'. Under 'Table name', there is a dropdown menu showing '(New) nyc', and next to it are 'Refresh', 'Preview data', and 'New' buttons. Under 'Advanced', there is a section for 'Table action' with two radio buttons: 'Append' and 'Overwrite' (selected). Below this is 'Enable partitions' with a checked checkbox. Under 'Partition columns', there is a '+ Add column' button and a 'Refresh' button. Below these are two dropdown menus: the first shows 'tripType' and the second shows 'paymentType', each with a '+' and a trash icon to its right.

FIGURE 2-27 Enabling partitions and assigned columns from the data source

6. Select **Run**.

This run's result will look the same in the Lakehouse Explorer as the run without partitions. The difference occurs when you select the created table's **ellipsis** and select **View files**. The result will then look similar to Figure 2-28.



Name	Date modified	Type	Size
.delta_log	3/22/2024 11:46:22 AM	Folder	13 items
tripType=1	3/22/2024 11:49:04 AM	Folder	5 items
tripType=2	3/22/2024 11:52:34 AM	Folder	5 items
tripType=_HIVE_DEFAULT_PARTITION_	3/22/2024 11:49:12 AM	Folder	5 items

FIGURE 2-28 Implemented partitions on a table with a blank value

EXAMPLE OF IMPLEMENTING PARTITIONS USING FABRIC NOTEBOOKS

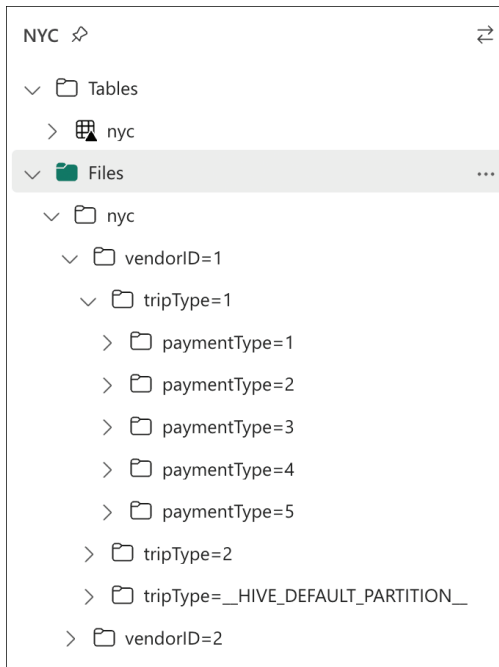
Create and open a new notebook in a workspace where is also a lakehouse that might be used as a data destination, and then follow these steps:

1. Connect to the lakehouse with the data ingested by the previous pipeline.
2. Delete all default cells, and create one new **Code cell**.
3. Insert the following code:

```
df = spark.sql("SELECT * FROM nyc")
df.write.partitionBy("vendorID", "tripType", "paymentType").mode("overwrite").
parquet("Files/nyc")
```

4. Select **Run**.

The function `partitionBy` from step 3 creates partitions based on the column names inserted, which in this case are `vendorID`, `tripType`, and `paymentType`. These appear both in Lakehouse Explorer (Figure 2-29) and a notebook's Lakehouse Preview.



Name	Date modified	Type	Size
vendorID=1		Folder	
tripType=1		Folder	
paymentType=1		Folder	
paymentType=2		Folder	
paymentType=3		Folder	
paymentType=4		Folder	
paymentType=5		Folder	
tripType=2		Folder	
tripType=_HIVE_DEFAULT_PARTITION_		Folder	
vendorID=2		Folder	

FIGURE 2-29 Preview of partitions in Lakehouse Explorer

Create views, functions, and stored procedures

Fabric lakehouses that use SQL analytic endpoints and warehouses both support the creation of views, functions, and stored procedures. Three components are integral parts of a **SQL** database:

- **View** A virtual table whose content is pre-defined by a query
- **Function** A user-defined function that accepts parameters, performs an action, such as a complex calculation, and returns the result as a scalar value or a table
- **Stored procedure** A block of T-SQL code stored in a database that the client can execute

All of them can be created using the SQL query editor either directly in the SQL analytic endpoint interface via the SQL query of the mentioned items or by using SQL Management Studio or Azure Data Studio. You can create views and stored procedures using templates also. To access them, hover over their respective folder, and then select the **ellipsis** icon that appears (Figure 2-30). Note that the **Refresh** option in the resulting menu refreshes only the preview of the data, not the data itself.

<div>▼ Lakehouse</div> <div>▼ Schemas</div> <div>▼ dbo</div> <div> > Tables</div> <div> > Views</div> <div> > Functions</div> <div> > Stored Procedur...</div> <div> > guest</div> <div> > INFORMATION_SCHE...</div> <div> > queryinsights</div> <div> > sys</div>	1	10361	20
	2	10361	26
	3	10139	49
	4	10270	31
	5	10139	20
	6	10139	20
	7	10270	44
	8	10361	25
	9	10270	32
	New stored procedure		21
	Refresh		33
			20
	13	10361	24
	14	10361	23
	15	10288	28

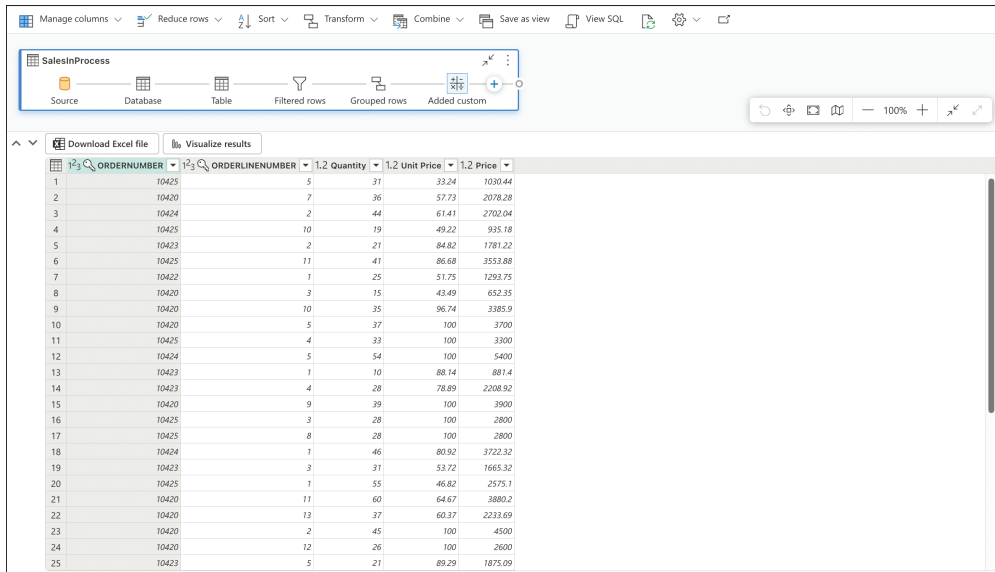
FIGURE 2-30 Quick options for creating a Stored Procedure

NOTE CSV FILE SAMPLE

The data in the following examples are created from the CSV file found at github.com/tirnovar/dp-600/blob/main/data/sales.

Views

You can create a view in two ways: using a **SQL command** in the SQL query editor or a **visual query**, which is a no-code option that uses Diagram view in Power Query and is shown in Figure 2-31.



The screenshot shows the Power Query Online interface. At the top, there's a toolbar with options like 'Manage columns', 'Reduce rows', 'Sort', 'Transform', 'Combine', 'Save as view', 'View SQL', and a settings icon. Below the toolbar is a diagram view showing a flow from 'Source' to 'Database' to 'Table' to 'Filtered rows' to 'Grouped rows' to 'Added custom'. The main area displays a data preview table with 25 rows and 5 columns: ORDERNUMBER, ORDERLINENUMBER, Quantity, Unit Price, and Price. The table contains numerical data for each row.

	ORDERNUMBER	ORDERLINENUMBER	Quantity	Unit Price	Price
1	10425	5	31	33.24	1030.44
2	10420	7	36	57.73	2078.28
3	10424	2	44	61.41	2702.04
4	10425	10	19	49.22	935.18
5	10423	2	21	84.82	1781.22
6	10425	11	41	86.68	3553.88
7	10422	1	25	51.75	1293.75
8	10420	3	15	43.49	652.35
9	10420	10	35	96.74	3385.9
10	10420	5	37	100	3700
11	10425	4	33	100	3300
12	10424	5	54	100	5400
13	10423	1	10	88.14	881.4
14	10423	4	28	78.89	2208.92
15	10420	9	39	100	3900
16	10425	3	28	100	2800
17	10425	8	28	100	2800
18	10424	1	46	80.92	3722.32
19	10423	3	31	53.72	1665.32
20	10425	1	55	46.82	2575.1
21	10420	11	60	64.67	3880.2
22	10420	13	37	60.37	2233.69
23	10420	2	45	100	4500
24	10420	12	26	100	2600
25	10423	5	21	89.29	1875.09

FIGURE 2-31 Diagram experience with data preview of a visual query

You can open the entire Power Query Online window, but you risk using an untranslatable operation. Power Query provides you with a data preview during each transformation step, allowing you to navigate your data easily and see what is happening. If you only use operations that Power Query can convert to SQL, you can save your results by selecting the **Save as view** button. (If you use a nontranslatable operation, an information banner will immediately tell you.) The **Save as view** popup is shown in Figure 2-32.

Another approach is to use the SQL query editor, where you can write and execute all your queries. These queries can also be stored as a personal queries alias, **My queries**, or as **Shared queries**, which all users with the right to access that item (SQL endpoint or warehouse) can see and potentially use if they have permission to execute SQL queries. This option contains a **Save as view** button next to the **Run** button. You can save any selected part of the code to create a new view. You can, therefore, test even more complex queries or perform different queries simultaneously. When you find a specific part of the code that suits you and returns the correct results, you can create a view from it, as shown in Figure 2-33.

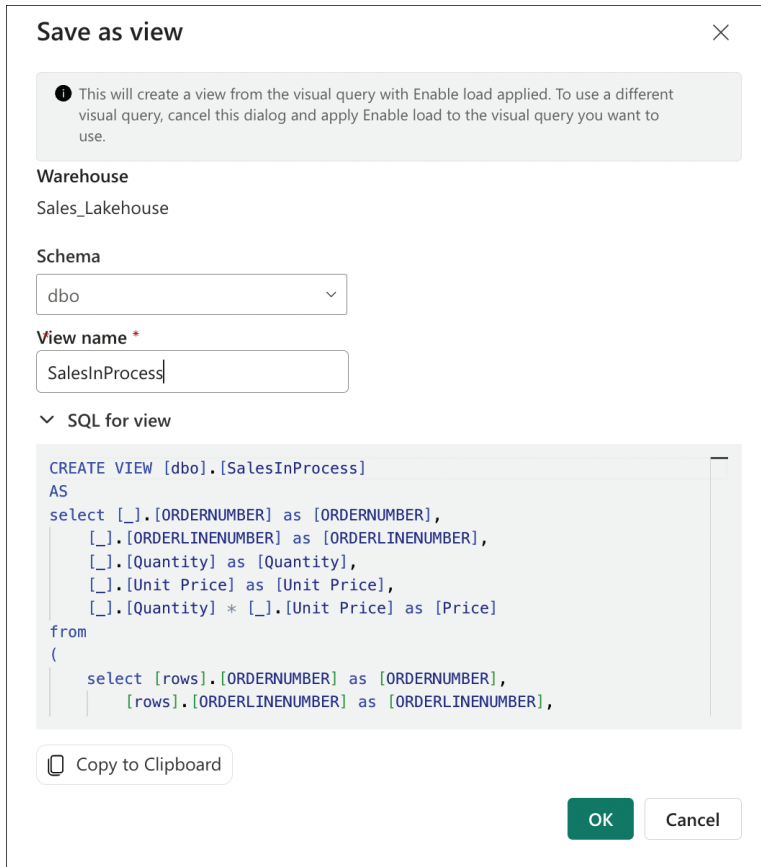


FIGURE 2-32 SQL Preview in the Save as view

SQL query

Run Save as view

```
1 SELECT
2   ... STATUS,
3   ... PRODUCTLINE,
4   ... COUNT (QUANTITYORDERED) AS 'ORDERS',
5   ... SUM (QUANTITYORDERED * UNITPRICE) AS 'ORDERED AMMOUNT',
6   ... SUM (SALES) AS 'TOTAL PRICE'
7 FROM sales WHERE
8   ... YEAR_ID = '2004'
9 GROUP BY STATUS,
10  PRODUCTLINE
```

Messages Results Open in Excel Explore this data (preview)

	ABC STATUS	ABC PRODUCTLINE	123 ORDERS	12L ORDERED AMMOUNT	12 TOTAL PRICE
1	Cancelled	Vintage Cars	9	345	28419.33
2	Resolved	Ships	6	183	16492.16
3	On Hold	Trucks and Buses	4	171	20193.29
4	Cancelled	Planes	9	311	26533.11
5	Shipped	Trucks and Buses	138	4831	509109.6
6	Cancelled	Trains	1	42	5082.42
7	Shipped	Motorcycles	164	5690	560545.23
8	Shipped	Vintage Cars	275	9605	883004.44
9	Shipped	Planes	150	5143	468552.24
10	Shipped	Ships	99	3413	292522.76
11	Cancelled	Classic Cars	15	449	53318.65
12	Shipped	Classic Cars	425	14923	1702871.52
13	On Hold	Classic Cars	2	46	6066.92
14	Cancelled	Ships	10	341	32423.05
15	Shipped	Trains	36	1261	111441.43

Succeeded (3 sec 126 ms) Columns: 5 Rows: 16

FIGURE 2-33 Selected T-SQL that will be used as a view

As with a visual query, you still need to set the view's name (Figure 2-34). You can also take another look at the code that will be used.

Save as view [X]

1 This will save the text of your SQL query as a view. Make sure the SQL syntax for the view is correct below.

Warehouse
Sales_Lakehouse

Schema
dbo

View name *
Orderes_Overview

✓ **SQL for view**

```
AS
SELECT
    STATUS,
    PRODUCTLINE,
    COUNT (QUANTITYORDERED) AS 'ORDERS',
    SUM (QUANTITYORDERED) AS 'ORDERED AMMOUNT',
    SUM (SALES) AS 'TOTAL PRICE'
FROM sales WHERE
    YEAR_ID = 2004
GROUP BY STATUS,
PRODUCTLINE
```

Copy to Clipboard

OK Cancel

FIGURE 2-34 T-SQL preview in Save as view window

Of course, you have the option to create views directly using `CREATE VIEW` using the following syntax:

```
CREATE [ OR ALTER ] VIEW [ schema_name . ] view_name [ ( column_name [ ,...n ] ) ]
AS <select_statement> [;]
<select_statement> ::=
[ WITH <common_table_expression> [ ,...n ] ]
<select_criteria>
```

NEED MORE REVIEW? T-SQL VIEWS

For more information about views, please visit learn.microsoft.com/sql/t-sql/statements/create-view-transact-sql.

Functions

Functions cannot be defined using a visual query, so you must use the T-SQL syntax directly within the SQL query option using the function syntax:

```
CREATE FUNCTION [ schema_name. ] function_name ( [ { @parameter_name [ AS ] parameter_
data_type [ = default ] } [ ,...n ]])
RETURNS TABLE
    [ WITH SCHEMABINDING ] [ AS ]
RETURN [ ( ) select_stmt [ ) ] [ ; ]
```

Alternatively, you could use another tool, such as Azure Data Studio or SQL Management Studio.

NEED MORE REVIEW? FUNCTIONS

For more information about functions, please visit learn.microsoft.com/sql/t-sql/statements/create-function-sql-data-warehouse.

Stored Procedures

Stored procedures cannot be created using a visual query either. T-SQL syntax must be used here as well:

```
CREATE [ OR ALTER ] { PROC | PROCEDURE } [ schema_name. ] procedure_name
    [ { @parameter data_type } [ OUT | OUTPUT ] ] [ ,...n ]
AS
{ [ BEGIN ] sql_statement [;] [ ,...n ] [ END ] } [;]
```

You can also create stored procedures using a shortcut that prepares the piece of code. To use this shortcut, follow these steps:

1. Open Warehouse Explorer.
2. Right-click the **Stored Procedures** folder or the **ellipsis** that appear after hovering over it.
3. Select **New stored procedure**.

To use this shortcut as a SQL query in Warehouse Explorer:

1. Open Warehouse Explorer.
2. Expand more options at the **New SQL query**.
3. Select **Stored procedure**.

Both approaches create the code shown in Figure 2-35 as a new SQL query. You can then edit your code and prepare it to do exactly what you need.

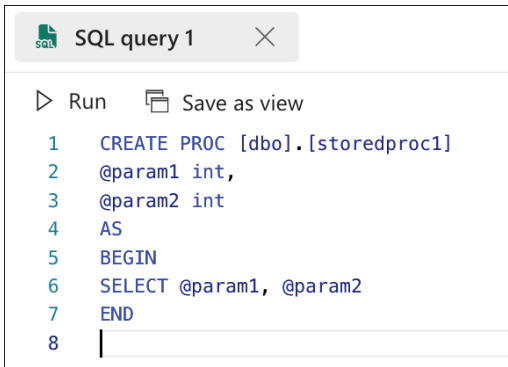


FIGURE 2-35 Create a stored procedure template

NEED MORE REVIEW? STORED PROCEDURES

For more information about stored procedures, please visit learn.microsoft.com/sql/t-sql/statements/create-procedure-transact-sql.

NOTE AZURE DATA STUDIO AND SQL MANAGEMENT STUDIO

A SQL endpoint and warehouse can be connected using Azure Data Studio or SQL Management Studio. These tools provide you with various ready-made templates for creating views, functions, and stored procedures from their environment.

Enrich data by adding new columns or tables

As you prepare the data based on the input scenarios, it may sometimes correspond to how it will need to look in the end. Often, though, additional columns or tables will need to be added and existing ones modified or removed. Microsoft Fabric, within its ingest items and T-SQL, allows you to enrich the data and thus compile the resulting views containing precisely what is needed.

Remember that Warehouse Explorer and SQL endpoint currently do *not* support ALTER TABLE ADD COLUMN within the lakehouse and warehouse data items. Therefore, extending tables with additional columns by T-SQL is impossible; instead, you must delete the table and create it again. Within Lakehouse Explorer, however, you can edit the Delta tables schema using notebooks or SparkJobs.

The `withColumn()` and `select()` functions can extend existing DataFrames. The `select()` function allows you to define the columns you want to keep from the existing DataFrame. At the same time, it allows the columns defined below to be renamed, be retyped, or to perform some function with them, such as `explode()`. This feature allows you to select the same column

twice, expanding the DataFrame with a new column. The function `withColumn()` returns a new DataFrame by adding a column or replacing the existing one with the same name.

Consider a few examples of using these functions:

```
df.select("id","name") # Will select only two columns from whole DataFrame
df.select("*",(df.UnitPrice-df.ProductionPrice).alias('ProductMargin')) # Calculates new
column based on two existing and will contain all previous columns
df.withColumn('ProductMargin', df.UnitPrice - df.ProductionPrice) # Calculates new
column based on two existingwithColumn('UnitPrice', df.UnitPrice + 10) # Replaces
current column UnitPrice with new values
```

NEED MORE REVIEW? FUNCTION EXPLODE()

For the Spark definition of function `explode()`, please visit spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.sql.functions.explode.html.

To modify the existing schema of existing tables using the schema of extended DataFrames, you can use two options parameters of the `options()` function:

- `mergeSchema` Expands the existing schema with the schema of the Spark frame being written
- `overwriteSchema` Overwrites the schema of the existing Delta table with the schema of the written Spark frame

For example, you could use:

```
df.write.mode("overwrite").option("mergeSchema", "true").saveAsTable("sales")
df.write.mode("overwrite").option("overwriteSchema", "true").saveAsTable("sales")
```

IMPORTANT REMOVING EXISTING COLUMNS BY `overwriteSchema`

Suppose an existing column is eliminated by `overwriteSchema` but not existing Parquet (like meanwhile Append mode). In that case, the data will not be discarded and will still be part of existing Parquet files. Only from the schema's point of view does the given column no longer exist, and it is not usually possible to query it.

As shown in the previous section, you can create a table using **Copy Activity**, **DataFlow Gen2**, or **notebooks** directly based on the data. You can also ingest data using a data pipeline, dataflow, or notebook. At the same time, you can create a new table using **T-SQL within the Warehouse Explorer**.

1. Select **New SQL query** to expand it.
2. Select **Table** (Figure 2-36).

Index

A

- access control
 - protected labels, 30
 - workspace-level, 26–28
- accuracy, data, 109
- ADDCOLUMNS function, 184–185
- admin portal, 4
 - Data model settings, 12
 - Endorsement and discovery settings, 10
 - Export and Sharing settings, 7–9
 - Fabric settings, 4–6
 - Git integration settings, 13–14
 - Information Protection settings, 6–8
 - Integration settings, 10–11
 - OneLake settings, 13
 - Scale-out settings, 12
 - Workspace settings, 6
- aggregate data, 117–119
- aggregation table, 238–239
- ALL function, 176–178
- ALLEXCEPT function, 179
- ALLNOBLANKROW function, 178
- ALLSELECTED function, 215–216
- analytics
 - descriptive, 261–262
 - data visualization, 262
 - reports, 263
 - summary statistics, 262
 - diagnostic, 264
 - anomaly detection, 264–265
 - cohort analysis, 264
 - hypotheses, 264
 - techniques, 264
 - exploratory, 261
 - predictive, 266, 267
 - prescriptive, 266, 269
- anomaly detection, 264–265
- apps, Microsoft Fabric Capacity Metrics, 3
- assigning, Sensitivity Labels, 30–32

- audit, data, 110
- Azure DevOps, 46–47

B

- binning, 165–168
- bottlenecks, performance, 133
- BPA (Best Practice Analyzer), 250
- bridge table, 113–116, 150
- budget, data analytics solution, 4
- bursting, 3

C

- CALCULATE function, 171–173, 211–215
- calculated column
 - circular dependency, 173–174
 - creating, 153
 - grouping values, 162
 - binning, 165–168
 - IF function, 162–163
 - List group type, 168
 - SWITCH function, 163–165
 - using Power BI interface, 165
- calculated table, 174, 204–206
- CALCULATETABLE function, 179–180
- calculation groups, 231–234
- CALENDAR function, 192–194
- CALENDARAUTO function, 193
- capacity, Fabric, 3
- CAST() function, 125–127
- cast() function, 127–128
- certification, 9
- choosing, gateway type, 14
- circular dependency, 173–174

COALESCE() function

- COALESCE() function, 125
- code
 - DAX, variables, 169–170
 - formatting, 158
- cohort analysis, 264
- column/s
 - adding, 84–87
 - calculated, 153
 - circular dependency, 173–174
 - grouping values, 162–168
 - fully qualified syntax, 155
 - measures, 206–208
 - versus calculated columns, 208
 - dynamic strings, 234
 - implicit, 231
 - Opening Profit, 220
 - using CALCULATE function, 211–215
- object-level security, 243–244
- commands
 - Fabric
 - OPTIMIZE, 135–137
 - VACUUM, 135–137
 - SQL
 - CREATE FUNCTION, 83
 - CREATE VIEW, 82
- composite model, aggregations, 238–239
- compute, bursting, 3
- conditional queries, 289–290
- consistency, data, 110
- conversion, data type, 151
 - using PySpark, 127–128
 - using SQL, 125–127
- CONVERT() function, 125–127
- copying data, 87–88
 - fast copy, 95–99
 - between lakehouses
 - using a notebook, 92–95
 - using data pipeline, 89–91
 - using DataFlow Gen2, 91–93
- corr() function, 281
- COUNT function, 209
- COUNTAX function, 209
- COUNTBLANK function, 209–210
- counting values in DAX, 208–211
- COUNTROWS function, 209
- COUNTX function, 209
- CREATE FUNCTION command, 83
- CREATE VIEW command, 82

- creating
 - calculated column, 153–154
 - data pipeline, 62
 - functions, 83
 - shortcuts, 74–75
 - stored procedures, 83–84
 - tables, 85–87
 - views, 80–82
- CROSSJOIN function, 188–189
- CU (capacity unit) seconds, 3

D

- data
 - accuracy, 109
 - aggregation, 117–119
 - audit, 110
 - completeness, 109–110, 277–279
 - consistency, 110
 - copying, 87–88
 - denormalization, 116–117
 - filtering, 128
 - using M formula language, 130–131
 - using PySpark, 131–132
 - using T-SQL, 128–130
 - joining, 120
 - merging, 120–122
 - quality, 110
 - relationships, 281
 - security, 2–3
 - sharing, 28–29
 - uniformity, 110
 - validity, 109
 - variety, 2
 - velocity, 2
 - volume, 2
- data analytics solution
 - configure Fabric-enabled workspace settings, 33–36
 - implement data sharing, 28–29
 - implement workspace- and item-level access controls for Fabric items, 26–28
 - manage Fabric capacity, 37–39
 - manage the analytics development lifecycle, 39–40
 - create and manage a Power BI Desktop project, 42–43
 - deployment rules, 46

- implement version control for a workspace, 40–42
 - plan and implement deployment solutions, 44–47
- planning, 1, 2–3
 - add a data source to a gateway, 18–19
 - budget, 4
 - choose a data gateway type, 14. *See also* gateway
 - create a custom Power BI report theme, 20–25
 - CU (capacity unit) seconds, 3
 - Fabric capacity, 3
 - Fabric SKU, 3
 - Fabric workload, 3
 - licensing, 4
 - manage data source users, 19
 - manage gateway settings, 16–17
 - manage gateway users, 17
 - recommend settings in the Fabric admin portal, 4–14. *See also* settings
 - skillsets, 3
 - tenant administration for gateways, 15–16
 - use a gateway, 19–20
- Sensitivity Labels, 30
 - assigning, 30–32
 - inheritance, 32–33
 - protected, 30
- data distribution, 274–278
- data model, 143
 - field parameters, 235–236
 - settings, 12
- data partitioning, 76. *See also* partition/s
- data pipeline
 - adding a dataflow, 102–103
 - adding a notebook, 103–104
 - adding a pipeline activity, 99–100
 - adding a stored procedure, 100–102
 - copying data between lakehouses, 89–91
 - creating, 62
 - editor, 62
 - implementing partitions by Copy Activity, 77–78
 - ingesting data to lakehouse, 63–65
 - scheduling, 105
 - Transform group, 100
- data profiling, 270
 - data pattern, 280
 - outputs, 270
 - tools, 270
- data sources, connectivity
 - Direct Lake, 145
 - DirectQuery, 144–145
- data statistics, 271–274
- data type
 - conversion, 151
 - implicit, 155–156
 - using PySpark, 127–128
 - using SQL, 125–127
 - DAX, 151
- data visualization, 262
- data warehouse, 2–3
 - bridge table, 113
 - copying data to, 87–88
 - data sharing, 28–29
 - Direct Lake, 144
 - impact analysis, 49
 - querying, 288–290
 - star schema, 111
 - denormalization of data, 116–117
 - dimension table, 111–112
 - fact table, 111–112
 - SCD (slowly changing dimension), 112–113
- Data Wrangler, 71, 275–276, 277–278
- dataflow/s, 3, 66–67
 - adding to a data pipeline, 102–103
 - checking for duplicate data, 122–123
 - copying data between lakehouses, 91–93
 - fast copy, 95–99
 - ingesting data to lakehouse, 67–71
 - performance guidelines, 133–134
 - refresh history, 97
 - scheduling, 106–108
- DataFrame, adding columns, 84–87
- DATATABLE function, 202–204
- date and time functions, 160–161
- DATEADD function, 220–222
- DATESBETWEEN function, 223–224
- DATESINPERIOD function, 224
- DATESMTD function, 218
- DATESQTD function, 218
- DATESYTD function, 218
- DAX, 143, 270
 - calculated column, creating, 153–154
 - counting values, 208–211
 - data types, 151, 155–156
 - functions
 - ADDCOLUMNS, 184–185
 - ALL, 176–178
 - ALLEXCEPT, 179
 - ALLNOBLANKROW, 178
 - ALLSELECTED, 215–216
 - AND, 153

DAX, continued

- CALCULATE, 171–173
- CALCULATETABLE, 179–180
- CALENDAR, 192–194
- CALENDARAUTO, 193
- COUNT, 209
- COUNTAX, 209
- COUNTBLANK, 209–210
- COUNTROWS, 209
- COUNTX, 209
- CROSSJOIN, 188–189
- DATATABLE, 202–204
- date and time, 160–161
- DATEADD, 220–222
- DATESBETWEEN, 223–224
- DATESINPERIOD, 224
- DISTINCT, 180
- DISTINCTCOUNT, 210–211
- EXCEPT, 198–199
- FILTER, 174–176
- FIND, 156–157
- FORMAT, 151–152
- GENERATEALL, 189–190
- GENERATESERIES, 190–192
- IF, 162–163
- IFERROR, 157
- INDEX, 229
- information, 230–231
- INTERSECT, 196–198, 226–227
- LEN, 156, 158
- LOOKUPVALUE, 161–162
- mathematical, 159–160
- NATURALINNERJOIN, 199–201
- NATURALLEFTOUTERJOIN, 201–202
- OFFSET, 229
- RANK, 228–229
- RELATED, 155
- RELATEDTABLE, 155
- ROW, 194
- row context, 170–174
- ROWNUMBER, 228–229
- SELECTCOLUMNS, 185–187
- SELECTEDVALUE, 225–226
- SUBSTITUTE, 158–159
- SUMMARIZE, 181–184
- SUMMARIZECOLUMNS, 184
- SWITCH, 163–165
- TOPN, 187–188
- TREATAS, 227–228
- UNION, 195–196
- USERNAME, 242
- USERPRINCIPALNAME, 242
- VALUES, 180
 - window, 228–230
- WINDOW, 229–230
- null values, handling, 152
- operators, 152–153
- query/ies
 - DEFINE, 293
 - EVALUATE statement, 292–293
 - ORDER BY, 293
 - parameters, 294–295
 - START AT, 294
- Time Intelligence, 216–224
- variables, 169–170
- DAX Formatter, 157–158
- DAX Studio, 146, 249–250, 291–292
- de-aggregation, 117–119
- delta file format, 2
- Delta table, 135
- denormalization of data, 116–117
- dependency, circular, 173–174
- Deployment Pipeline, perform impact analysis of
- downstream dependencies, 47–55
- deployment rules, 46
- describe() function, 262
- descriptive analytics, 261–262
 - data visualization, 262
 - reports, 263
 - summary statistics, 262
- development lifecycle, 39
 - create and manage a Power BI Desktop project, 42–43
 - implement version control for a workspace, 40–42
 - perform impact analysis of downstream dependencies, 47–55
 - plan and implement deployment solutions, 44–47
- diagnostic analytics, 264
 - anomaly detection, 264–265
 - cohort analysis, 264
 - hypotheses, 264
 - techniques, 264
- Direct Lake, 144, 145
 - advantages, 145
 - disadvantages, 145
- DirectQuery, 144
 - advantages, 144–145
 - disadvantages, 145
- disconnected tables, passing filters from, 226–230

display() function, 262, 274, 279
 DISTINCT function, 180
 DISTINCTCOUNT function, 210–211
 DP-600 Implementing Analytics Solutions Using Microsoft Fabric exam
 objective mapping, 301–303
 updates, 300–301
 dtypes() function, 271
 duplicate data, checking for
 in dataflows, 122–123
 in SQL and PySpark, 123–124
 dynamic RLS (row-level security), 241–243
 dynamic strings, 234

E

endorsement, settings, 10
 endpoint, XMLA (XML for Analysis), 10–11, 50–52, 291
 enterprise-scale semantic model, optimizing query and report performance, 248–249
 EVALUATE statement, 292–293
 evaluation context, 170–174
 Excel, data analysis, 11
 EXCEPT function, 198–199
 explicit measures, 208
 exploratory analytics, 261
 Export and Sharing settings, 7–9
 expr() function, 127–128
 extending tables, 84–87

F

Fabric
 admin portal, 4. *See also* admin portal
 capacity, 3, 37–39
 Deployment Pipelines, 44–47
 experiences, 2
 implement workspace- and item-level access controls, 26–28
 OPTIMIZE command, 135–137
 settings, 4–6
 shared semantic model, 54–55
 SKU (stock keeping unit), 3
 fast copy, 95–99
 field parameters, 235–236
 filter context, 170–174
 filter data, 128

 using M formula language, 130–131
 using PySpark, 131–132
 using T-SQL, 128–130
 filter() function, 131–132
 FILTER function, 174–176
 filters, passing from disconnected tables, 226–230
 FIND function, 156–157
 fixing implicit measures, 208
 FORMAT function, 151–152
 formatting, code, 158
 formulas. *See also* functions
 DAX, variables, 169–170
 evaluation context, 170–174
 full outer join, 120
 AND function, 153
 function/s, 79
 corr(), 281
 creating, 83
 DAX
 ADDCOLUMNS, 184–185
 ALL, 176–178
 ALLEXCEPT, 179
 ALLNOBLANKROW, 178
 ALLSELECTED, 215–216
 AND, 153
 CALCULATE, 171–173
 CALCULATETABLE, 179–180
 CALENDAR, 192–194
 CALENDARAUTO, 193
 COALESCE(), 125
 COUNT, 209
 COUNTAX, 209
 COUNTBLANK, 209–210
 COUNTRROWS, 209
 COUNTX, 209
 CROSSJOIN, 188–189
 DATATABLE, 202–204
 date and time, 160–161
 DATEADD, 220–222
 DATESBETWEEN, 223–224
 DATESINPERIOD, 224
 DATESMTD, 218
 DATESQTD, 218
 DATESYTD, 218
 DISTINCT, 180
 DISTINCTCOUNT, 210–211
 EXCEPT, 198–199
 FILTER, 174–176
 FIND, 156–157

function/s, continued

- FORMAT, 151–152
- GENERATEALL, 189–190
- GENERATESERIES, 190–192
- IF, 162–163
- IFERROR, 157
- INDEX, 229
- information, 230–231
- INTERSECT, 196–198, 226–227
- LEN, 156, 158
- LOOKUPVALUE, 159–161
- mathematical, 159–160
- NATURALINNERJOIN, 199–201
- NATURALLEFTOUTERJOIN, 201–202
- OFFSET, 229
- OPENINGBALANCEMONTH, 218
- RANK, 228–229
- RELATED, 155
- RELATEDTABLE, 155
- ROW, 194
- row context, 170–174
- ROWNUMBER, 228–229
- SELECTCOLUMNS, 185–187
- SELECTEDVALUE, 225–226
- SUBSTITUTE, 158–159
- SUMMARIZE, 181–184
- SUMMARIZECOLUMNS, 184
- SWITCH, 163–165
- Time Intelligence, 216–224
- TOPN, 187–188
- TREATAS, 227–228
- UNION, 195–196
- USERNAME, 242
- USERPRINCIPALNAME, 242
- VALUES, 180
- window, 228–230
- WINDOW, 229–230
- display(), 279
- filter context, 170–174
- Inspect(), 274–275, 278–279
- MLflow, PREDICT, 267
- options(), 85
- PySpark
 - cast(), 127–128
 - describe(), 262
 - display(), 262, 274
 - dtypes(), 271
 - expr(), 127–128
 - filter(), 131–132

- na.fill(), 125
- summary(), 271
- QuickVisualize(), 263
- select(), 84–85
- SQL
 - CAST(), 125–127
 - CONVERT(), 125–127
- Table.Profile(), 273, 277–278
- Table.Schema(), 273
- withColumn(), 84–85

G

- gateway, 19–20
 - adding a data source, 18–19
 - choosing, 14
 - data source users, 19
 - installation, 15–16
 - Personal mode, 15
 - settings, 16–17
 - Standard mode, 15
 - tenant administration, 15
 - users, 17
 - VNet, 16
- GENERATEALL function, 189–190
- GENERATESERIES function, 190–192
- Git integration, 13–14, 40
- governance, 4, 7, 25
- grouping values, 162
 - binning, 165–168
 - IF function, 162–163
 - List group type, 168
 - SWITCH function, 163–165
 - using Power BI interface, 165

H

- heat map, 281
- histogram, 274, 276–277
- hypotheses, 264

I

- IF function, 162–163
- IFERROR function, 157
- impact analysis of downstream dependencies, 47–49

- implementing
 - partitions
 - using Copy Activity in a pipeline, 77–78
 - using Fabric notebooks, 78
 - star schema, 148
- implicit measures, 231
- implicit type conversion, 155–156
- importing
 - data to Power BI, 144
 - libraries to notebook, 72
- inactive relationships, 224–225
- incremental refresh, 251–252
 - creating the RangeStart and RangeEnd parameters, 252
 - filtering by using the RangeStart and RangeEnd parameters, 253–254
 - policies, 254–256
 - query folding, 256
- INDEX function, 229
- information functions, 230–231
- Information Protection settings, 6–8
- ingesting data
 - fast copy, 95–99
 - to lakehouse
 - using a notebook, 72–73
 - using data pipeline, 63–65
 - using dataflows, 67–71
- inheritance, Sensitivity Label, 32–33
- inner join, 120
- Inspect() function, 274–275, 278–279
- installation, gateway, 15–16
- integration, settings, 10–11
- IntelliSense, 154
- INTERSECT function, 196–198, 226–227
- item-level access control, 26–28

J

- joins, 120, 121
- JSON, editing Power BI theme file, 23–24

L

- lakehouse/s, 2–3
 - bridge table, 113–116
 - copying data between
 - using a notebook, 92–95

- using data pipeline, 89–91
 - using DataFlow Gen2, 91–93
- copying data to, 87–88
- data partitioning, 76
- data sharing, 28–29
- Direct Lake, 144
- impact analysis, 49
- ingesting data
 - by dataflows, 67–71
 - by pipeline, 63–65
 - using a notebook, 72–73
- sample data, 89
- shortcut, creating, 74–75
- SQL queries, 282–284
- star schema, 111
 - denormalization of data, 116–117
 - dimension table, 111–112
 - fact table, 111–112
 - SCD (slowly changing dimension), 112–113
- large-format semantic model, building, 236–238
- left anti join, 120
- left outer join, 120
- LEN function, 156, 158
- library/ies
 - importing to notebook, 72
 - Matplotlib, 276–277
 - powerbiclient, 263
 - predictive analytics, 267
- licensing, Power BI, 4
- line charts, 267–268
- Lineage view, workspace, 47
- LOOKUPVALUE function, 159–161

M

- M language, 130–131, 143
- many-to-many relationship, 113, 148–150
- mathematical functions, 159–160
- Matplotlib library, 276–277
- measures, 206–208
 - versus calculated columns, 208
 - dynamic strings, 234
 - implicit, 231
 - Opening Profit, 220
 - query-level, 293
 - using CALCULATE function, 211–215
- membership, security role, 245–246
- merging data, 120–122

metadata

metadata, 144
Microsoft Fabric Capacity Metrics, 3
MLflow, 266–267

N

na.fill() function, 125
NATURALINNERJOIN function, 199–201
NATURALLEFTOUTERJOIN function, 201–202
normalization, 116
notebook/s, 3

- adding to a data pipeline, 103–104
- copying data between lakehouses, 92–95
- data distribution, 274–278
- data statistics, 271–274
- Data Wrangler tool, 71
- impact analysis, 49
- implementing partitions, 78
- ingesting data to lakehouse, 72–73
- libraries, 72
- performance, best practices, 134
- scheduling, 108
- scripting language, 71

null values, handling, 124–125, 152

O

object-level security, 243–244, 246–248
OFFSET function, 229
OneLake

- settings, 13
- shortcuts, 74

Opening Profit measure, 220
OPENINGBALANCEMONTH function, 218
operators, DAX, 152–153
OPTIMIZE command, 135–137
options() function, 85
overwriteSchema parameter, options() function, 85

P

Parquet file, 135
partition/s, 76

- column, 76
- condition, 76
- implementing

- using Copy Activity in a pipeline, 77–78
- using Fabric notebooks, 78

passing filters from disconnected tables, 226–230
.pbids file, 52–54
performance

- bottlenecks, 133
- DAX, 249–250
- notebook, best practices, 134
- query, 248–249
- report, optimizing, 248–249
- semantic model, 250
- SQL, best practices, 134

Personal mode, gateway, 15
pipeline, impact analysis, 49. *See also* data pipeline
planning a data analytics solution, 1, 2–3, 16

- add a data source to a gateway, 18–19
- budget, 4
- choose a data gateway type, 14. *See also* gateway
- create a custom Power BI report theme, 20–25
- CU (capacity unit) seconds, 3
- Fabric capacity, 3
- Fabric SKU, 3
- Fabric workload, 3
- licensing, 4
- manage data source users, 19
- manage gateway settings, 16–17
- manage gateway users, 17
- recommend settings in the Fabric admin portal, 4–14. *See also* settings
- skillsets, 3
- tenant administration for gateways, 15–16
- use a gateway, 19–20

policies

- incremental refresh, 254–256
- Sensitivity Label, 30

Power BI. *See also* DAX; semantic model

- creating a custom report theme, 20–21
- data modeling, 143
- Decomposition tree, 265
- Direct Lake connectivity, 145
- DirectQuery, 144
 - advantages, 144–145
 - disadvantages, 145
- Forecasting, 268–269
- impact analysis of downstream dependencies, 47–49
- importing data, 144
- Key influencer visual, 265
- licensing, 4
- report template, 52

- RLS (row-level security), 240
- shared semantic model, 54–55
- theme editor, 22
- theme file, editing, 23–24
- Power BI Desktop
 - creating roles, 240–241
 - IntelliSense, 154
 - Performance Analyzer, 248–249
 - View as roles window, 246–247
- Power Query, 80
- Power Query Online, 285–288
- powerbiclient library, 263
- PREDICT function, 267
- predictive analytics, 266
 - libraries, 267
 - line charts, 267–268
- prescriptive analytics, 266, 269
- primary key, 173
- production environment, 44
- protected Sensitivity Labels, 30
- PySpark
 - cast() function, 127–128
 - checking for duplicate data, 123–124
 - converting data type, 127–128
 - creating a new table, 87
 - describe() function, 262
 - display() function, 262
 - dtypes() function, 271
 - filter() function, 131–132
 - filtering data, 131–132
 - na.fill() function, 125
 - null values, handling, 125

Q

- quality, data, 110
- query/ies. *See also* SQL
 - conditional, 289–290
 - DAX
 - DEFINE, 293
 - EVALUATE statement, 292–293
 - ORDER BY, 293
 - parameters, 294–295
 - START AT, 294
 - folding, 97, 133–134, 256
 - SQL, 282–284
- QuickVisualize() function, 263

R

- RANK function, 228–229
- refresh history, dataflow, 97
- RELATED function, 155
- RELATEDTABLE function, 155
- relationship/s, 148, 281
 - inactive, 224–225
 - many-to-many, 113, 148–150
 - virtual, 226–228
- removing, duplicate data, 122–124
- report/s, 263
 - optimizing performance, 248–249
 - Power BI, 20–21
 - create a custom theme using third-party tools, 24–25
 - edit a theme JSON file, 23–24
 - .pbids file, 52–54
 - theme editor, 22
 - shared semantic model, 54–55
 - templates, 52
- right anti join, 120
- right outer join, 120
- RLS (row-level security), 240
 - dynamic, 241–243
 - validating, 246–248
- roles, creating in Power BI Desktop, 240–241
- row context, 170–174
- ROW function, 194
- ROWNUMBER function, 228–229

S

- Scale-out settings, 12
- SCD (slowly changing dimension), 112–113
- scheduling
 - data pipelines, 105
 - dataflows, 106–108
 - notebooks, 108
- scripting language, 71
- security
 - data, 2–3
 - groups, 246
 - object-level, 243–244, 246–248
 - role membership, 245–246
 - row-level, 240
 - dynamic, 241–243
 - validating, 246–248

select() function

- select() function, 84–85
- SELECTCOLUMNS function, 185–187
- SELECTEDVALUE function, 225–226
- semantic model, 11
 - bridge table, 114–116, 150
 - choose a storage mode, 144
 - deployment and management using XMLA endpoint, 50–52
 - enterprise-scale
 - optimizing DAX performance, 249–250
 - optimizing query and report performance, 248–249
 - implementing a star schema, 148
 - incremental refresh, 251–252
 - creating the RangeStart and RangeEnd parameters, 252
 - filtering by using the RangeStart and RangeEnd parameters, 253–254
 - query folding, 256
 - large-format, building, 236–238
 - optimizing, 250
 - relationships, 148
 - inactive, 224–225
 - many-to-many, 148–150
 - shared, 54–55
- Sensitivity Labels, 6–7, 30
 - assigning, 30–32
 - default settings, 30
 - inheritance, 32–33
 - protected, 30
- settings
 - Data model, 12
 - Endorsement and discovery, 10
 - Export and Sharing, 7–9
 - Fabric, 4–6
 - Fabric Capacity, 37–39
 - Fabric-enabled workspace, 33–36
 - gateway, 16–17
 - Git integration, 13–14
 - Information Protection, 6–8
 - Integration, 10–11
 - OneLake, 13
 - Scale-out, 12
 - Sensitivity Label, 30
 - Spark, 34–35
 - Workspace, 6
- shared semantic model, 54–55
- shortcuts, 74, 88, 288
 - creating, 74–75
 - creating a stored procedure, 83
- SKU (stock keeping unit)
 - Fabric, 3
 - Fabric Capacity, 35
- slicers, 235
- smoothing, 3
- Spark, settings, 34–35
- SQL
 - checking for duplicate data, 123–124
 - commands
 - CREATE FUNCTION, 83
 - CREATE VIEW, 82
 - converting data type, 125–127
 - functions, 79
 - CAST(), 125–127
 - CONVERT(), 125–127
 - null values, handling, 125
 - performance, best practices, 134
 - Power Query, 80
 - queries, 282–284
 - query editor, 80, 284
 - stored procedure, 79, 83–84
 - tables, extending, 84–87
 - views, 79, 80–82
 - visual query, 80
- standard deviation, 272
- Standard mode, gateway, 15
- star schema, 111
 - bridge table, 113–116
 - denormalization of data, 116–117
 - dimension table, 111–112
 - fact table, 111–112
 - implementing, 148
 - SCD (slowly changing dimension), 112–113
- StarterPool, 34
- statement, EVALUATE, 292–293
- stored procedure, 79
 - adding to a data pipeline, 100–102
 - creating, 83–84
- structured data, 2
- SUBSTITUTE function, 158–159
- SUMMARIZE function, 181–184
- SUMMARIZECOLUMNS function, 184
- summary() function, 271
- summary statistics, 262
- SWITCH function, 163–165

T

Table.Profile() function, 273, 277–278
 table/s. *See also* data model; relationships
 aggregation, 238–239
 bridge, 113–116, 150
 calculated, 174, 204–206
 calculated column, 153
 columns
 adding, 154–156
 calculated, 153
 fully qualified syntax, 155
 creating, 85–87
 Delta, 135
 dimension, 111–113
 disconnected, passing filters from, 226–230
 duplicating, 174
 extending, 84–87
 fact, 111–112
 measures, 206–208
 versus calculated columns, 208
 dynamic strings, 234
 implicit, 231
 Opening Profit, 220
 using CALCULATE function, 211–215
 merging, 120–122
 object-level security, 243–244
 Table.Schema() function, 273
 Tabular Editor 2, 147–148, 250
 tabular model, calculation groups, 231–234
 template, report, 52
 tenant administration, gateway, 15
 test environment, 44
 theme editor, Power BI, 22
 third-party tools, theme generator, 24–25
 Time Intelligence, 216–224
 tools
 data profiling, 270
 Data Wrangler, 71, 275–276, 277–278
 DAX Formatter, 157–158
 DAX Studio, 146, 291–292
 Tabular Editor 2, 147–148
 TOPN function, 187–188
 TREATAS function, 227–228
 troubleshooting
 Delta tables, 135–137
 performance
 dataflows, 133–134
 notebook, 134
 SQL, 134

T-SQL

 creating a new table, 85–87
 filtering data, 128–130
 functions, creating, 83
 stored procedure, creating, 83–84

U

uniformity, data, 110
 UNION function, 195–196
 unique record, 122
 unstructured data, 2
 updates, DP-600 Implementing Analytics Solutions
 Using Microsoft Fabric exam, 300–301
 USERNAME function, 242
 USERPRINCIPALNAME function, 242

V

VACUUM command, 135–137
 validating, row- and object-level security,
 246–248
 VALUES function, 180
 variables
 calculated table, 204–206
 DAX, 169–170
 version control, workspace, 40–42
 View as roles window, Power BI Desktop,
 246–247
 views, 79, 80–82
 virtual relationship, 226–228
 visual query, 80
 visual query editor, 284–290
 VNet (virtual network), gateway, 16

W

what-if scenarios, 269
 WHERE clause, 128–130
 WINDOW function, 229–230
 window functions, 228–230
 withColumn() function, 84–85
 workspace
 connecting to, 50–51
 Data model settings, 12
 data sharing, 28–29

workspace, continued

workspace, *continued*

- Fabric-enabled, 33–36
- Git integration, 40
- level access control, 26–28
- Lineage view, 47
- settings, 6
- version control, 40–42

X-Y-Z

- XMLA (XML for Analysis), 10–11
 - endpoint, 50–52
 - connecting to a dataset, 291
 - deploy and manage semantic models, 50–52
 - querying a dataset, 292–293