Designing and Implementing Enterprise-Scale Analytics Solutions Using Microsoft Azure and Microsoft Power BI

Exam Ref DP-500

Daniil Maslyuk
Justin Frébault

Microsoft

Exam Ref DP-500 Designing and Implementing Enterprise-Scale Analytics Solutions Using Microsoft Azure and Microsoft Power BI

Daniil Maslyuk
Justin Frébault

has two implications. First, accessing the data for OLAP queries is faster with columnar format, because the data we want to query (the whole column) is physically stored next to each other. With a CSV or JSON file, the data of the same column would be physically scattered. Second, the data is generally more homogenous in a column than in a row. Indeed, usually a row contains multiple data types, so there will be a mix of data. In a column the data has the same type and can be similar or even identical sometimes. This leads to greater potential for compression. That is why Parquet files are recommended for analytics workloads.

> ***NOTE*** **DELTA**
>
> **In 2020, a new file format named Delta was released open source. It extends Parquet and includes ACID transactions. It is compatible with Synapse Analytics and is the recommended file format if you work with Spark pools.**

How to query Parquet files

To review how to query a Parquet file, let's go back to our Synapse Studio:

1. Navigate to **Develop** > **+** > **SQL script**.
2. Make sure it is attached to **Built-in**, as a reminder, this is your serverless pool.
3. Using `OPENROWSET`, select the top 10 rows of your taxi dataset in a Parquet format:

```
SELECT
 TOP 10 *
FROM
OPENROWSET(
BULK 'https:// your-storage.dfs.core.windows.net/ your-filestore /synapse/
workspaces/green_tripdata_2022-01.parquet',
FORMAT = 'PARQUET'
) AS [result]
```

4. Select **Run**.

You should get the same results as before, as shown earlier in Figure 2-11.

Query relational data sources in dedicated or serverless SQL pools, including querying partitioned data sources

SQL pools, whether dedicated or serverless, will test your knowledge of T-SQL. Here you will learn how to query data, both with dedicated and serverless, and we will also consider partitioned data.

To review how to use SQL dedicated pools, let's go back to our Synapse Studio and:

1. Go to the **Manage** menu.
2. Under **Analytics pools** select **SQL pools** > **+ New**.
3. Set **Dedicated SQL Pool Name** to a name of your choice

4. Change the **Performance** level to DW100c; the default is DW1000c.
5. Select **Review + create > Create**.

The deployment of the dedicated SQL pool will take a few minutes. Next let's look at how to ingest the data into the dedicated pool:

6. Go to **Develop** > **+** > **SQL script**.
7. Set the **Connect to** option to your dedicated SQL pool, not to **Built-in**.
8. You can ingest from the data lake to your SQL dedicated pool with the `COPY` statement. You will need to replace the file address with your own address:

```
COPY INTO dbo.TaxiTrips
FROM 'https://your-storage.dfs.core.windows.net/your-filestore/synapse/workspaces/
green_tripdata_2022-01.parquet'
WITH
(
FILE_TYPE = 'PARQUET',
MAXERRORS = 0,
IDENTITY_INSERT = 'OFF',
AUTO_CREATE_TABLE = 'ON'
)
```

9. You can now run a `SELECT` on the newly created table in your dedicated pool. You will see that the table has been populated with the data from the Parquet file.

```
SELECT TOP 10 * FROM dbo.TaxiTrips
```

10. Select **Run**.

You should now see the first 10 rows of the table, as shown in Figure 2-12.

VendorID	lpep_pickup_d...	lpep_dropoff_...	store_and_fwd...	RatecodeID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount
2	164150222400...	164150244400...	N	1	42	42	2	0.86	5
1	164155400300...	164155446500...	N	1	75	239	1	1.4	7.5
2	164157484400...	164157512400...	N	1	74	41	2	0.84	5
2	164158892200...	164158921000...	N	1	255	255	1	0.58	5
1	164163993800...	164164217000...	N	1	55	181	1	0	35.88
2	164165803000...	164165957600...	N	4	16	265	2	22.99	109.5
2	164167201000...	164167252000...	N	1	74	42	1	1	7
2	164172666000...	164172695300...	N	1	7	7	1	0.56	5
1	164175019800...	164175177100...	N	1	7	17	1	0	24.38
2	164180474500...	164180525200...	N	1	74	152	1	1.2	7.5

FIGURE 2-12 Results from querying the populated table with Synapse Dedicated Pool

11. Don't forget to pause your dedicated SQL pool to limit the cost of this exercise.

Oftentimes, though, Parquet files are partitioned. To review and reproduce this use case, we'll use the SQL serverless pool. So follow these steps:

1. Navigate to https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page and download the February and March 2022 Green Taxi Trip Records.
2. In Synapse Studio, select **Data** > **Linked** > **Azure Data Lake Storage Gen2**.
3. Open your primary data store.

4. With the **+ New folder** command, create a folder structure in your data store representing YEAR/MONTH, as shown in Figure 2-13.

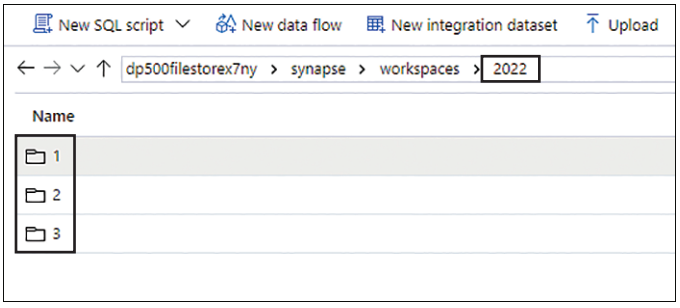

FIGURE 2-13 The file store hierarchy in Synapse Studio

5. Upload the datasets in their corresponding folders; you should have one Parquet file per month.
6. To leverage the partitioning in your query, try to count how many records are in each partition, using the `OPENROWSET` function, the Parquet format, and the `*` wildcard.
7. Run the query for Partitions 1 and 2:

```
SELECT
COUNT(*)
FROM
OPENROWSET(
BULK 'https:// your-storage.dfs.core.windows.net/ your-filestore /synapse/
workspaces/2022/*/*.parquet',
FORMAT = 'PARQUET'
) AS TaxiTrips
WHERE TaxiTrips.filepath(1) IN ('1', '2')
```

8. Run the query for Partition 1:

```
SELECT
COUNT(*)
FROM
OPENROWSET(
BULK 'https://your-storage.dfs.core.windows.net/your-filestore/synapse/
workspaces/2022/*/*.parquet',
FORMAT = 'PARQUET'
) AS TaxiTrips
WHERE TaxiTrips.filepath(1) IN ('1')
```

9. Check that the results are different.

You now know how to query a Parquet file containing partitions.

Use a machine learning PREDICT function in a query

Finally, Synapse SQL dedicated pools can be used to consume a machine learning model. For example, with our taxi dataset, we could have a model to predict the duration of the trip, based

on the pickup location and the time of day. Having the ability to score, or consume, the model directly in Synapse Analytics is useful because we don't need to move the data outside of our data analytics platform. To score our model, we will use the PREDICT function.

However, a Synapse SQL dedicated pool doesn't have the ability to train a machine learning model. So to use the PREDICT function, we will need to have the model trained outside of Synapse SQL. We could, for example, train the model in a Synapse Spark pool or in another product like Azure Machine Learning.

The trained model will need to be converted to the ONNX (Open Neural Network Exchange) format. ONNX is an open source standard format, with the very purpose of enabling exchange of models between platforms.

Load a model in a Synapse SQL dedicated pool table

The model has to be stored in a dedicated SQL pool table, as a hexadecimal string in a `varbinary(max)` column. For instance, such a table could be created with this:

```
CREATE TABLE [dbo].[Models]
(
[Id] [int] IDENTITY(1,1) NOT NULL,
[Model] [varbinary](max) NULL,
[Description] [varchar](200) NULL
)
WITH
(
DISTRIBUTION = ROUND_ROBIN,
HEAP
)
GO
```

where **Model** is the `varbinary(max)` column storing our model, or models, as hexadecimal strings.

Once the table is created, we can load it with the COPY statement:

```
COPY INTO [Models] (Model)
FROM '<enter your storage location>'
WITH (
FILE_TYPE = 'CSV',
CREDENTIAL=(IDENTITY= 'Shared Access Signature', SECRET='<enter your storage key here>')
)
```

We now have a model ready to be used for scoring.

Scoring the model

Finally, the PREDICT function will come into play to score the model. Like any machine learning scoring, you will need the input data to have the same format as the training data.

The following query shows how to use the PREDICT function. It takes the model and the data as parameters.

```
DECLARE @model varbinary(max) = (SELECT Model FROM Models WHERE Id = 1);
SELECT d.*, p.Score
```

```
FROM PREDICT(MODEL = @model,
DATA = dbo.mytable AS d, RUNTIME = ONNX)
WITH (Score float) AS p;
```

NEED MORE REVIEW? **THE PREDICT FUNCTION**

The full documentation for the PREDICT function can be found here: *https://learn.microsoft.com/en-us/sql/t-sql/queries/predict-transact-sql?view=sql-server-ver15*.

Skill 2.2: Ingest and transform data by using Power BI

Power BI includes Power Query, which is an extract-transform-load (ETL) tool that uses the M language. M is a functional, case-sensitive language that, unlike DAX, does not resemble Excel formula language in any way, and differs from DAX in important ways, too. In this section, we'll look at the problems you may need to solve when working with large amounts of data in Power Query.

When the volume or number of data sources is significant, you may face performance degradation. There are tools within Power Query that will help you identify the performance problems, and later we'll review the techniques you can use to improve performance.

In addition to Power BI Desktop, Power Query is available in Power BI dataflows, and we'll review when you'd want to use dataflows and what you'd need to consider when using them.

Combining data from different data sources will lead to data privacy issues, which we'll also discuss later in this chapter.

Finally, we'll discuss how you can use Advanced Editor to write your own queries and functions, and how you can query some of the more complex data sources by using Power Query.

NOTE **COMPANION FILE**

Most of the Power Query queries shown in this chapter are available in the companion PBIX file.

This skill covers how to:

- Identify data loading performance bottlenecks in Power Query or data sources
- Implement performance improvements in Power Query and data sources
- Create and manage scalable Power BI dataflows
- Identify and manage privacy settings on data sources
- Create queries, functions, and parameters by using the Power Query Advanced Editor
- Query advanced data sources, including JSON, Parquet, APIs, and Azure Machine Learning models

Identify data loading performance bottlenecks in Power Query or data sources

Several reasons could be responsible for poor performance when connecting to data in Power BI. Power BI Desktop has a few features that can help identify those issues.

View native query

When you get data in Power BI from some data sources, like databases, Power Query will do its best to translate the transformations you perform into the native language of the data source—for example, SQL. This feature of Power Query is known as *query folding*. Most of the time, this will make getting data more efficient. For instance, if you connect to a database and get a subset of columns from a table, Power Query may only retrieve those columns from the data source instead of loading all columns and then locally removing the ones you don't want.

In some cases, it may be possible to view the query that Power Query sent to the data source to retrieve the data you wanted. For this, you need to right-click a query step in Power Query Editor and select **View Native Query**. The window that opens looks like Figure 2-14.

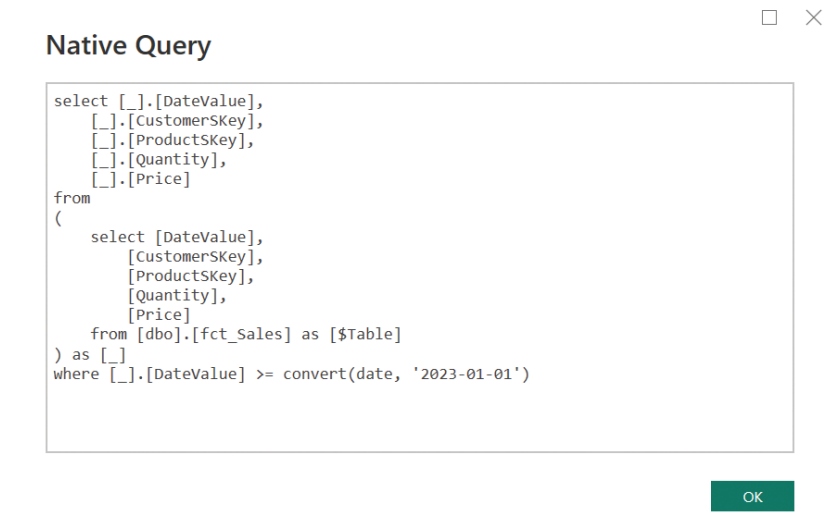

FIGURE 2-14 Native Query window

In the query shown in Figure 2-14, we connected to a SQL Server database, applied a filter, and selected a few columns. Because these operations can be translated to SQL, Power Query decided to do the transformations in the source instead of performing them after loading the whole table, which led to better performance.

You cannot edit the native query; it is provided for your information only. If you want Power BI to issue a specific query, you must provide a SQL statement when connecting to a database.