Serverless SQL pool in Azure Synapse Analytics

Every Azure Synapse Analytics workspace comes with serverless SQL pool endpoints that you can use to query data in the Azure Data Lake (Parquet, Delta Lake, delimited text formats), Azure Cosmos DB, or Dataverse.

Serverless SQL pool is a query service over the data in your data lake. It enables you to access your data through the following functionalities:

- A familiar T-SQL syntax to query data in place without the need to copy or load data into a specialized store. To learn more, see the T-SQL support section.
- Integrated connectivity via the T-SQL interface that offers a wide range of business intelligence and ad-hoc querying tools, including the most popular drivers. To learn more, see the Client tools section.

Serverless SQL pool is a distributed data processing system, built for large-scale data and computational functions. Serverless SQL pool enables you to analyze your Big Data in seconds to minutes, depending on the workload. Thanks to built-in query execution fault-tolerance, the system provides high reliability and success rates even for long-running queries involving large data sets.

Serverless SQL pool is serverless, hence there's no infrastructure to setup or clusters to maintain. A default endpoint for this service is provided within every Azure Synapse workspace, so you can start querying data as soon as the workspace is created.

There is no charge for resources reserved, you are only being charged for the data processed by queries you run, hence this model is a true pay-per-use model.

If you use Apache Spark for Azure Synapse in your data pipeline, for data preparation, cleansing or enrichment, you can query external Spark tables you've created in the process, directly from serverless SQL pool. Use Private Link to bring your serverless SQL pool endpoint into your managed workspace VNet.

Serverless SQL pool benefits

If you need to explore data in the data lake, gain insights from it or optimize your existing data transformation pipeline, you can benefit from using serverless SQL pool. It is suitable for the following scenarios:

- Basic discovery and exploration - Quickly reason about the data in various formats (Parquet, CSV, JSON) in your data lake, so you can plan how to extract insights from it.
- Logical data warehouse – Provide a relational abstraction on top of raw or disparate data without relocating and transforming data, allowing always up-to-date view of your data. Learn more about creating logical data warehouse.
- Data transformation - Simple, scalable, and performant way to transform data in the lake using T-SQL, so it can be fed to BI and other tools, or loaded into a relational data store (Synapse SQL databases, Azure SQL Database, etc.).

Different professional roles can benefit from serverless SQL pool:

- Data Engineers can explore the lake, transform and prepare data using this service, and simplify their data transformation pipelines. For more information, check this tutorial.
- Data Scientists can quickly reason about the contents and structure of the data in the lake, thanks to features such as OPENROWSET and automatic schema inference.
- Data Analysts can explore data and Spark external tables created by Data Scientists or Data Engineers using familiar T-SQL language or their favorite tools, which can connect to serverless SQL pool.
- BI Professionals can quickly create Power BI reports on top of data in the lake and Spark tables.

How to start using serverless SQL pool

Serverless SQL pool endpoint is provided within every Azure Synapse workspace. You can create a workspace and start querying data instantly using tools you are familiar with.

Make sure that you are applying the best practices to get the best performance.

Client tools

Serverless SQL pool enables existing SQL ad-hoc querying and business intelligence tools to tap into the data lake. As it provides familiar T-SQL syntax, any tool capable to establish TDS connection to SQL offerings can connect to and query Synapse SQL. You can connect with Azure Data Studio and run ad-hoc queries or connect with Power BI to gain insights in a matter of minutes.

T-SQL support

Serverless SQL pool offers T-SQL querying surface area, which is slightly enhanced/extended in some aspects to accommodate for experiences around querying semi-structured and unstructured data. Furthermore, some aspects of the T-SQL language aren't supported due to the design of serverless SQL pool, as an example, DML functionality is currently not supported.

- Workload can be organized using familiar concepts:
- Databases - serverless SQL pool endpoint can have multiple databases.
- Schemas - Within a database, there can be one or many object ownership groups called schemas.
- Views, stored procedures, inline table value functions
- External resources – data sources, file formats, and tables

Security can be enforced using:

- Logins and users
- Credentials to control access to storage accounts
- Grant, deny, and revoke permissions per object level
- Azure Active Directory integration

Supported T-SQL:

- Full SELECT surface area is supported, including a majority of SQL functions
- CETAS - CREATE EXTERNAL TABLE AS SELECT
- DDL statements related to views and security only

Serverless SQL pool has no local storage, only metadata objects are stored in databases. Therefore, T-SQL related to the following concepts isn't supported:

- Tables
- Triggers
- Materialized views
- DDL statements other than ones related to views and security
- DML statements

Note

Serverless SQL pool queries have a timeout. For more information on query timeout that may affect your workload, see **serverless SQL pool system constraints**. Currently you can't change the timeout.

Extensions

In order to enable smooth experience for in place querying of data residing in files in data lake, serverless SQL pool extends the existing OPENROWSET function by adding following capabilities:

Query multiple files or folders

Query PARQUET file format

Query DELTA format

Various delimited text formats (with custom field terminator, row terminator, escape char)

Azure Cosmos DB analytical store

Read a chosen subset of columns

Schema inference

filename function

filepath function

Work with complex types and nested or repeated data structures

Security

Serverless SQL pool offers mechanisms to secure access to your data.

Azure Active Directory integration and multi-factor authentication

Serverless SQL pool enables you to centrally manage identities of database user and other Microsoft services with Azure Active Directory integration. This capability simplifies permission management and enhances security. Azure Active Directory (Azure AD) supports multi-factor authentication (MFA) to increase data and application security while supporting a single sign-on process.

Authentication

Serverless SQL pool authentication refers to how users prove their identity when connecting to the endpoint. Two types of authentication are supported:

- **SQL Authentication**

  This authentication method uses a username and password.

- **Azure Active Directory Authentication**:

  This authentication method uses identities managed by Azure Active Directory. For Azure AD users, multi-factor authentication can be enabled. Use Active Directory authentication (integrated security) whenever possible.

Authorization

Authorization refers to what a user can do within a serverless SQL pool database, and is controlled by your user account's database role memberships and object-level permissions.

If SQL Authentication is used, the SQL user exists only in serverless SQL pool and permissions are scoped to the objects in serverless SQL pool. Access to securable objects in other services (such as Azure Storage) can't be granted to SQL user directly since it only exists in scope of serverless SQL pool. The SQL user needs to use one of the supported authorization types to access the files.

If Azure AD authentication is used, a user can sign in to serverless SQL pool and other services, like Azure Storage, and can grant permissions to the Azure AD user.

Access to storage accounts

A user that is logged into the serverless SQL pool service must be authorized to access and query the files in Azure Storage. serverless SQL pool supports the following authorization types:

- **Shared access signature (SAS)** provides delegated access to resources in storage account. With a SAS, you can grant clients access to resources in storage account, without sharing account keys. A SAS gives you granular control over the type of access you grant to clients who have the SAS: validity interval, granted permissions, acceptable IP address range, acceptable protocol (https/http).
- **User Identity** (also known as "pass-through") is an authorization type where the identity of the Azure AD user that logged into serverless SQL pool is used to authorize access to the data. Before accessing the data, Azure Storage administrator must grant permissions to Azure AD user for accessing the data. This authorization type uses the Azure AD user that logged into serverless SQL pool, therefore it's not supported for SQL user types.
- **Workspace Identity** is an authorization type where the identity of the Synapse workspace is used to authorize access to the data. Before

accessing the data, Azure Storage administrator must grant permissions to workspace identity for accessing the data.

Access to Azure Cosmos DB

You need to create server-level or database-scoped credential with the Azure Cosmos DB account read-only key to access the Azure Cosmos DB analytical store.

Query CSV files

```
select top 10 *

from openrowset(

    bulk    'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases/latest/ecdc_cases.csv',

    format = 'csv',

 parser_version = '2.0',

    firstrow = 2 ) as rows
```

There are some additional options that can be used to adjust parsing rules to custom CSv format:

- ESCAPE_CHAR = 'char' Specifies the character in the file that is used to escape itself and all delimiter values in the file. If the escape character is followed by a value other than itself, or any of the delimiter values, the escape character is dropped when reading the value. The ESCAPE_CHAR parameter will be applied whether the FIELDQUOTE is or isn't enabled. It won't be used to escape the quoting character. The quoting character must be escaped with another quoting character. Quoting character can appear within column value only if value is encapsulated with quoting characters.
- FIELDTERMINATOR ='field_terminator' Specifies the field terminator to be used. The default field terminator is a comma ("**,**")
- ROWTERMINATOR ='row_terminator' Specifies the row terminator to be used. The default row terminator is a newline character: **\r\n**.

Query Delimted text files

```
SELECT  *
```

```
FROM OPENROWSET(
        BULK 'csv/population-unix-hdr-quoted/population.csv',
        FORMAT = 'CSV', PARSER_VERSION = '2.0',
        FIELDTERMINATOR =',',
        ROWTERMINATOR = '0x0a',
        FIRSTROW = 2,
        FIELDQUOTE = '"'
    ) AS [r]
```

Query Parquet files

```
Select  top  10 *
from openrowset(
    bulk    'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases/latest/ecdc_cases.parquet',
    format = 'parquet')  as  rows
```

Query JSON files

```
select top 10 *
from openrowset(
        bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases/latest/ecdc_cases.jsonl',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
Go
```

Query Delta Lake files

```
SELECT TOP 10 *
FROM OPENROWSET(
    BULK 'https://sqlondemandstorage.blob.core.windows.net/delta-lake/covid/',

    FORMAT = 'delta') as rows;
```

Create and Use external tables with Synapse SQL

An external table points to data located in Hadoop, Azure Storage blob, or Azure Data Lake Storage. External tables are used to read data from files or write data to files in Azure Storage. With Synapse SQL, you can use external tables to read external data using dedicated SQL pool or serverless SQL pool.

Depending on the type of the external data source, you can use two types of external tables:

- **Hadoop external tables** that you can use to read and export data in various data formats such as CSV, Parquet, and ORC. Hadoop external tables are available in dedicated SQL pools, but they aren't available in serverless SQL pools.
- **Native external tables** that you can use to read and export data in various data formats such as CSV and Parquet. Native external tables are available in serverless SQL pools, and they are in **public preview** in dedicated SQL pools. Writing/exporting data using CETAS and the native external tables is available only in the serverless SQL pool, but not in the dedicated SQL pools.

The key differences between Hadoop and native external tables are presented in the following table:

External table type	**Hadoop**	**Native**
Dedicated SQL pool	Available	Only Parquet tables are available in **public preview**.
Serverless SQL pool	Not available	Available
Supported formats	Delimited/CSV, Parquet, ORC, Hive RC, and RC	Serverless SQL pool: Delimited/CSV, Parquet, and Delta Lake Dedicated SQL pool: Parquet (preview)
Folder partition elimination	No	Partition elimination is available only in the partitioned tables created on Parquet or CSV formats that are synchronized from Apache Spark pools. You might create external tables on Parquet partitioned folders, but the partitioning columns will be inaccessible and ignored, while the partition elimination will not

External table type	Hadoop	Native
		be applied. Do not create external tables on Delta Lake folders because they are not supported. Use Delta partitioned views if you need to query partitioned Delta Lake data.
File elimination (predicate pushdown)	No	Yes in serverless SQL pool. For the string pushdown, you need to use Latin1_General_100_BIN2_UTF8 collation on the VARCHAR columns to enable pushdown.
Custom format for location	No	Yes, using wildcards like /year=*/month=*/day=* for Parquet or CSV formats. Custom folder paths are not available in Delta Lake. In the serverless SQL pool you can also use recursive wildcards /logs/** to reference Parquet or CSV files in any sub-folder beneath the referenced folder.
Recursive folder scan	Yes	Yes. In serverless SQL pools must be specified /** at the end of the location path. In Dedicated pool the folders are always scanned recursively.
Storage authentication	Storage Access Key(SAK), AAD passthrough, Managed identity, Custom application Azure AD identity	Shared Access Signature(SAS), AAD passthrough, Managed identity, Custom application Azure AD identity.
Column mapping	Ordinal - the columns in the external table definition are mapped to the columns in the underlying Parquet files by position.	Serverless pool: by name. The columns in the external table definition are mapped to the columns in the underlying Parquet files by column name matching. Dedicated pool: ordinal matching. The columns in the external table definition are mapped to the columns in the underlying Parquet files by position.
CETAS	Yes	CETAS with the native tables as a target works

External table type	Hadoop	Native
(exporting/transformation)		only in the serverless SQL pool. You cannot use the dedicated SQL pools to export data using native tables.

Note

The native external tables are the recommended solution in the pools where they are generally available. If you need to access external data, always use the native tables in serverless pools. In dedicated pools, you should switch to the native tables for reading Parquet files once they are in GA. Use the Hadoop tables only if you need to access some types that are not supported in native external tables (for example - ORC, RC), or if the native version is not available.

External tables in dedicated SQL pool and serverless SQL pool

You can use external tables to:

- Query Azure Blob Storage and Azure Data Lake Gen2 with Transact-SQL statements.
- Store query results to files in Azure Blob Storage or Azure Data Lake Storage using CETAS.
- Import data from Azure Blob Storage and Azure Data Lake Storage and store it in a dedicated SQL pool (only Hadoop tables in dedicated pool).

Note

When used in conjunction with the **CREATE TABLE AS SELECT** statement, selecting from an external table imports data into a table within the **dedicated** SQL pool.

If performance of Hadoop external tables in the dedicated pools do not satisfy your performance goals, consider loading external data into the Datawarehouse tables using the **COPY statement**.

For a loading tutorial, see **Use PolyBase to load data from Azure Blob Storage**.

You can create external tables in Synapse SQL pools via the following steps:

1. CREATE EXTERNAL DATA SOURCE to reference an external Azure storage and specify the credential that should be used to access the storage.
2. CREATE EXTERNAL FILE FORMAT to describe format of CSV or Parquet files.

3. CREATE EXTERNAL TABLE on top of the files placed on the data source with the same file format.

Folder partition elimination

The native external tables in Synapse pools are able to ignore the files placed in the folders that are not relevant for the queries. If your files are stored in a folder hierarchy (for example - **/year=2020/month=03/day=16**) and the values for **year**, **month**, and **day** are exposed as the columns, the queries that contain filters like year=2020 will read the files only from the subfolders placed within the **year=2020** folder. The files and folders placed in other folders (**year=2021** or **year=2022**) will be ignored in this query. This elimination is known as **partition elimination**.

The folder partition elimination is available in the native external tables that are synchronized from the Synapse Spark pools. If you have partitioned data set and you would like to leverage the partition elimination with the external tables that you create, use the partitioned views instead of the external tables.

File elimination

Some data formats such as Parquet and Delta contain file statistics for each column (for example, min/max values for each column). The queries that filter data will not read the files where the required column values do not exist. The query will first explore min/max values for the columns used in the query predicate to find the files that do not contain the required data. These files will be ignored and eliminated from the query plan. This technique is also known as filter predicate pushdown and it can improve the performance of your queries. Filter pushdown is available in the serverless SQL pools on Parquet and Delta formats. To leverage filter pushdown for the string types, use the VARCHAR type with the Latin1_General_100_BIN2_UTF8 collation.

Security

User must have SELECT permission on an external table to read the data. External tables access underlying Azure storage using the database scoped credential defined in data source using the following rules:

- Data source without credential enables external tables to access publicly available files on Azure storage.
- Data source can have a credential that enables external tables to access only the files on Azure storage using SAS token or workspace Managed Identity - For examples, see the Develop storage files storage access control article.

CREATE EXTERNAL DATA SOURCE

External data sources are used to connect to storage accounts. The complete documentation is outlined here.

Syntax for CREATE EXTERNAL DATA SOURCE

- Hadoop
- Native

External data sources without TYPE=HADOOP are generally available in serverless SQL pools and in public preview in dedicated pools.

syntaxsql

```
CREATE EXTERNAL DATA SOURCE <data_source_name>
WITH
(   LOCATION         = '<prefix>://<path>'
    [, CREDENTIAL = <database scoped credential> ]
)
[;]
```

Arguments for CREATE EXTERNAL DATA SOURCE

data_source_name

Specifies the user-defined name for the data source. The name must be unique within the database.

Location

LOCATION = '<prefix>://<path>' - Provides the connectivity protocol and path to the external data source. The following patterns can be used in location:

External Data Source	Location prefix	Location path
Azure Blob Storage	wasb[s]	<container>@<storage_account>.blob.core.windows.net
Azure Blob Storage	http[s]	<storage_account>.blob.core.windows.net/<container>/subfolders
Azure Data Lake	http[s]	<storage_account>.azuredatalakestore.net/webhdfs/v1

External Data Source	Location prefix	Location path
Store Gen 1		
Azure Data Lake Store Gen 2	http[s]	<storage_account>.dfs.core.windows.net/<container>/subfolders

https: prefix enables you to use subfolder in the path.

Credential

CREDENTIAL = <database scoped credential> is optional credential that will be used to authenticate on Azure storage. External data source without credential can access public storage account or use the caller's Azure AD identity to access files on storage.

- In dedicated SQL pool, database scoped credential can specify custom application identity, workspace Managed Identity, or SAK key.
- In serverless SQL pool, database scoped credential can specify workspace Managed Identity, or SAS key.

TYPE

TYPE = HADOOP is the option that specifies that Java-based technology should be used to access underlying files. This parameter can't be used in serverless SQL pool that uses built-in native reader.

Example for CREATE EXTERNAL DATA SOURCE

- Hadoop
- Native

The following example creates an external data source in serverless or dedicated SQL pool for Azure Data Lake Gen2 that can be accessed using SAS credential:
SQL

```
CREATE DATABASE SCOPED CREDENTIAL [sqlondemand]
WITH IDENTITY='SHARED ACCESS SIGNATURE',
SECRET = 'sv=2018-03-28&ss=bf&srt=sco&sp=rl&st=2019-10-
14T12%3A10%3A25Z&se=2061-12-
```

```
31T12%3A10%3A00Z&sig=KlSU2ullCscyTS0An0nozEpo4tO5JAgGBvw%2F
JX2lguw%3D'
GO

CREATE EXTERNAL DATA SOURCE SqlOnDemandDemo WITH (
    LOCATION = 'https://sqlondemandstorage.blob.core.windows.net',
    CREDENTIAL = sqlondemand
);
```

Note

The SQL users needs to have proper permissions on database scoped credentials to access the data source in Azure Synapse Analytics Serverless SQL Pool. **Access external storage using serverless SQL pool in Azure Synapse Analytics**.

The following example creates an external data source for Azure Data Lake Gen2 pointing to the publicly available New York data set:

SQL

```
CREATE EXTERNAL DATA SOURCE YellowTaxi
WITH ( LOCATION =
'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/')
```

CREATE EXTERNAL FILE FORMAT

Creates an external file format object that defines external data stored in Azure Blob Storage or Azure Data Lake Storage. Creating an external file format is a prerequisite for creating an external table. The complete documentation is here.

By creating an external file format, you specify the actual layout of the data referenced by an external table.

Syntax for CREATE EXTERNAL FILE FORMAT

Syntaxsql

```
-- Create an external file format for PARQUET files.
CREATE EXTERNAL FILE FORMAT file_format_name
WITH (
    FORMAT_TYPE = PARQUET
    [ , DATA_COMPRESSION = {
        'org.apache.hadoop.io.compress.SnappyCodec'
     | 'org.apache.hadoop.io.compress.GzipCodec'      }
    ]);

--Create an external file format for DELIMITED TEXT files
CREATE EXTERNAL FILE FORMAT file_format_name
WITH (
    FORMAT_TYPE = DELIMITEDTEXT
    [ , DATA_COMPRESSION = 'org.apache.hadoop.io.compress.GzipCodec' ]
    [ , FORMAT_OPTIONS ( <format_options> [ ,...n  ] ) ]
    );

<format_options> ::=
{
    FIELD_TERMINATOR = field_terminator
    | STRING_DELIMITER = string_delimiter
    | FIRST_ROW = integer
    | USE_TYPE_DEFAULT = { TRUE | FALSE }
    | ENCODING = {'UTF8' | 'UTF16'}
    | PARSER_VERSION = {'parser_version'}
}
```

Arguments for CREATE EXTERNAL FILE FORMAT

file_format_name- Specifies a name for the external file format.

FORMAT_TYPE = [PARQUET | DELIMITEDTEXT]- Specifies the format of the external data.

- PARQUET - Specifies a Parquet format.
- DELIMITEDTEXT - Specifies a text format with column delimiters, also called field terminators.

FIELD_TERMINATOR = *field_terminator* - Applies only to delimited text files. The field terminator specifies one or more characters that mark the end of each field (column) in the text-delimited file. The default is the pipe character ('|').

Examples:

- FIELD_TERMINATOR = '|'
- FIELD_TERMINATOR = ' '
- FIELD_TERMINATOR = '\t'

STRING_DELIMITER = *string_delimiter* - Specifies the field terminator for data of type string in the text-delimited file. The string delimiter is one or more characters in length and is enclosed with single quotes. The default is the empty string ("").

Examples:

- STRING_DELIMITER = '"'
- STRING_DELIMITER = '*'
- STRING_DELIMITER = ','

FIRST_ROW = *First_row_int* - Specifies the row number that is read first and applies to all files. Setting the value to two causes the first row in every file (header row) to be skipped when the data is loaded. Rows are skipped based on the existence of row terminators (/r/n, /r, /n).

USE_TYPE_DEFAULT = { TRUE | **FALSE** } - Specifies how to handle missing values in delimited text files when retrieving data from the text file.

Note

Please note that USE_TYPE_DEFAULT=true is not supported for FORMAT_TYPE = DELIMITEDTEXT, PARSER_VERSION = '2.0'.

TRUE - If you're retrieving data from the text file, store each missing value by using the default value's data type for the corresponding column in the external table definition. For example, replace a missing value with:

- 0 if the column is defined as a numeric column. Decimal columns aren't supported and will cause an error.
- Empty string ("") if the column is a string column.
- 1900-01-01 if the column is a date column.

FALSE - Store all missing values as NULL. Any NULL values that are stored by using the word NULL in the delimited text file are imported as the string 'NULL'.

Encoding = {'UTF8' | 'UTF16'} - Serverless SQL pool can read UTF8 and UTF16 encoded delimited text files.

DATA_COMPRESSION = *data_compression_method* - This argument specifies the data compression method for the external data.

The PARQUET file format type supports the following compression methods:

- DATA_COMPRESSION = 'org.apache.hadoop.io.compress.GzipCodec'
- DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'

When reading from PARQUET external tables, this argument is ignored, but is used when writing to external tables using CETAS.

The DELIMITEDTEXT file format type supports the following compression method:

- DATA_COMPRESSION = 'org.apache.hadoop.io.compress.GzipCodec'

PARSER_VERSION = 'parser_version' Specifies parser version to be used when reading CSV files. The available parser versions are 1.0 and 2.0. This option is available only in serverless SQL pools.

Example for CREATE EXTERNAL FILE FORMAT

The following example creates an external file format for census files:

SQL

```
CREATE EXTERNAL FILE FORMAT census_file_format
WITH
(
   FORMAT_TYPE = PARQUET,
   DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
)
```

CREATE EXTERNAL TABLE

The CREATE EXTERNAL TABLE command creates an external table for Synapse SQL to access data stored in Azure Blob Storage or Azure Data Lake Storage.

Syntax for CREATE EXTERNAL TABLE

syntaxsql

```
CREATE EXTERNAL TABLE { database_name.schema_name.table_name |
schema_name.table_name | table_name }
   ( <column_definition> [ ,...n ] )
   WITH (
      LOCATION = 'folder_or_filepath',
      DATA_SOURCE = external_data_source_name,
      FILE_FORMAT = external_file_format_name
```

```
        [, TABLE_OPTIONS =
N'{"READ_OPTIONS":["ALLOW_INCONSISTENT_READS"]}' ]
        [, <reject_options> [ ,...n ] ]
    )
[;]

<column_definition> ::=
column_name <data_type>
    [ COLLATE collation_name ]

<reject_options> ::=
{
    | REJECT_TYPE = value,
    | REJECT_VALUE = reject_value,
    | REJECT_SAMPLE_VALUE = reject_sample_value,
    | REJECTED_ROW_LOCATION = '/REJECT_Directory'
}
```

Arguments CREATE EXTERNAL TABLE

{ database_name.schema_name.table_name | *schema_name.table_name* | *table_name }*

The one to three-part name of the table to create. For an external table, Synapse SQL pool stores only the table metadata. No actual data is moved or stored in Synapse SQL database.

<column_definition>, ...*n*]

CREATE EXTERNAL TABLE supports the ability to configure column name, data type, and collation. You can't use the DEFAULT CONSTRAINT on external tables.

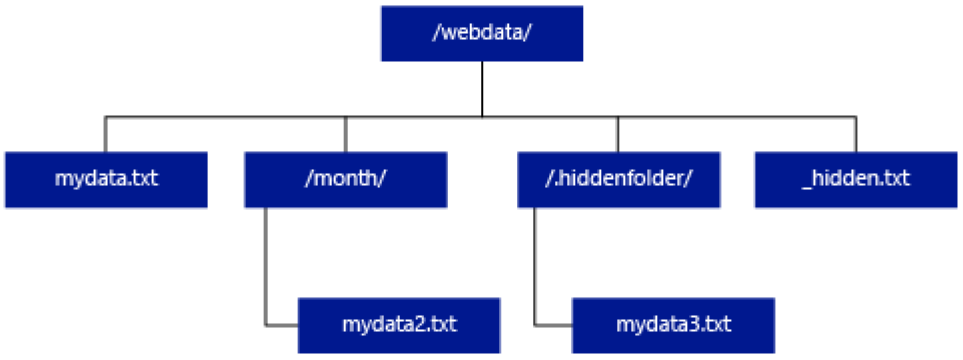
Important

The column definitions, including the data types and number of columns, must match the data in the external files. If there's a mismatch, the file rows will be rejected when querying the actual data. See reject options to control rejected rows behavior.

When reading from Parquet files, you can specify only the columns you want to read and skip the rest.

LOCATION = '*folder_or_filepath*'

Specifies the folder or the file path and file name for the actual data in Azure Blob Storage. The location starts from the root folder. The root folder is the data location specified in the external data source.



Unlike Hadoop external tables, native external tables don't return subfolders unless you specify /** at the end of path. In this example, if LOCATION='/webdata/', a serverless SQL pool query, will return rows from mydata.txt. It won't return mydata2.txt and mydata3.txt because they're located in a subfolder. Hadoop tables will return all files within any sub-folder.

Both Hadoop and native external tables will skip the files with the names that begin with an underline (_) or a period (.).

DATA_SOURCE = *external_data_source_name*

Specifies the name of the external data source that contains the location of the external data. To create an external data source, use CREATE EXTERNAL DATA SOURCE.

FILE_FORMAT = *external_file_format_name*

Specifies the name of the external file format object that stores the file type and compression method for the external data. To create an external file format, use CREATE EXTERNAL FILE FORMAT.

Reject Options

Note

Rejected rows feature is in Public Preview. Please note that rejected rows feature works for delimited text files and PARSER_VERSION 1.0.

You can specify reject parameters that determine how service will handle *dirty* records it retrieves from the external data source. A data record is considered 'dirty' if actual data types don't match the column definitions of the external table.

When you don't specify or change reject options, service uses default values. This information about the reject parameters is stored as additional metadata when you create an external table with CREATE EXTERNAL TABLE statement. When a future SELECT statement or SELECT INTO SELECT statement selects data from the external table, service will use the reject options to determine the number of rows that can be rejected before the actual query fails. The query will return (partial) results until the reject threshold is exceeded. It then fails with the appropriate error message.

REJECT_TYPE = **value**

This is the only supported value at the moment. Clarifies that the REJECT_VALUE option is specified as a literal value.

value

REJECT_VALUE is a literal value. The query will fail when the number of rejected rows exceeds *reject_value*.

For example, if REJECT_VALUE = 5 and REJECT_TYPE = value, the SELECT query will fail after five rows have been rejected.

REJECT_VALUE = *reject_value*

Specifies the number of rows that can be rejected before the query fails.

For REJECT_TYPE = value, *reject_value* must be an integer between 0 and 2,147,483,647.

REJECTED_ROW_LOCATION = *Directory Location*

Specifies the directory within the External Data Source that the rejected rows and the corresponding error file should be written. If the specified path doesn't exist, service will create one on your behalf. A child directory is created with the name "*rejectedrows". The* "" character ensures that the directory is escaped for other data processing unless explicitly named in the location parameter. Within this directory, there's a folder created based on the time of load submission in the format YearMonthDay_HourMinuteSecond_StatementID (Ex. 20180330-173205-559EE7D2-196D-400A-806D-3BF5D007F891). You can use statement id to correlate folder with query that generated it. In this folder, two files are written: error.json file and the data file.

error.json file contains json array with encountered errors related to rejected rows. Each element representing error contains following attributes:

Attribute	**Description**
Error	Reason why row is rejected.
Row	Rejected row ordinal number in file.
Column	Rejected column ordinal number.
Value	Rejected column value. If the value is larger than 100 characters, only the first 100 characters will be displayed.
File	Path to file that row belongs to.

TABLE_OPTIONS

TABLE_OPTIONS = *json options* - Specifies the set of options that describe how to read the underlying files. Currently, the only option that is available is "READ_OPTIONS":["ALLOW_INCONSISTENT_READS"] that instructs the external table to ignore the updates that are made on the underlying files, even if this might cause some inconsistent read operations. Use this option only in special cases where you have frequently appended files. This option is available in serverless SQL pool for CSV format.

Permissions CREATE EXTERNAL TABLE

To select from an external table, you need proper credentials with list and read permissions.

Example CREATE EXTERNAL TABLE

The following example creates an external table. It returns the first row:

SQL
```sql
CREATE EXTERNAL TABLE census_external_table
(
   decennialTime varchar(20),
   stateName varchar(100),
   countyName varchar(100),
   population int,
   race varchar(50),
   sex    varchar(10),
   minAge int,
   maxAge int
)
WITH (
    LOCATION = '/parquet/',
    DATA_SOURCE = population_ds,
    FILE_FORMAT = census_file_format
)
GO

SELECT TOP 1 * FROM census_external_table
```

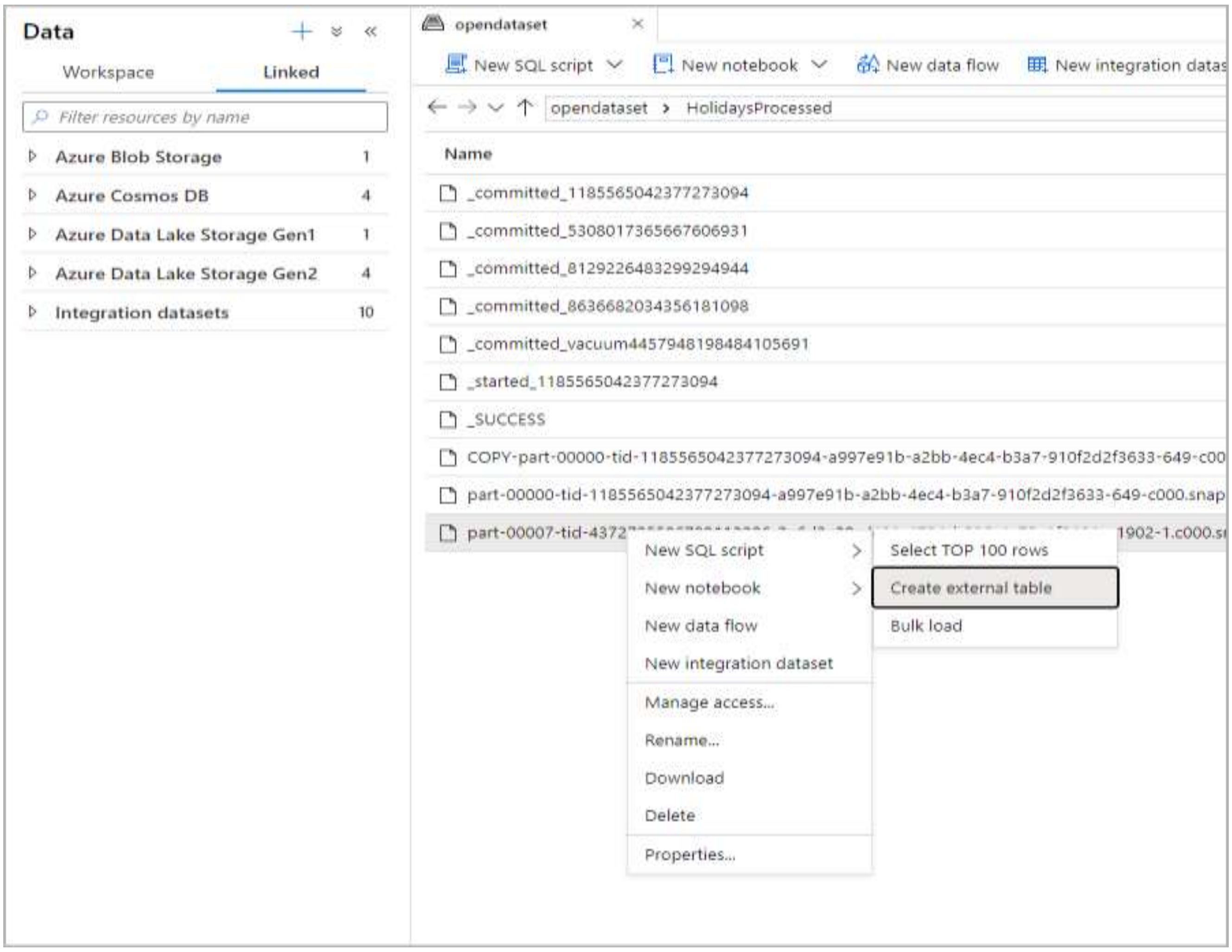
Create and query external tables from a file in Azuredl

Using Data Lake exploration capabilities of Synapse Studio you can now create and query an external table using Synapse SQL pool with a simple right-click on the file. The one-click gesture to create external tables from the ADLS Gen2 storage account is only supported for Parquet files.

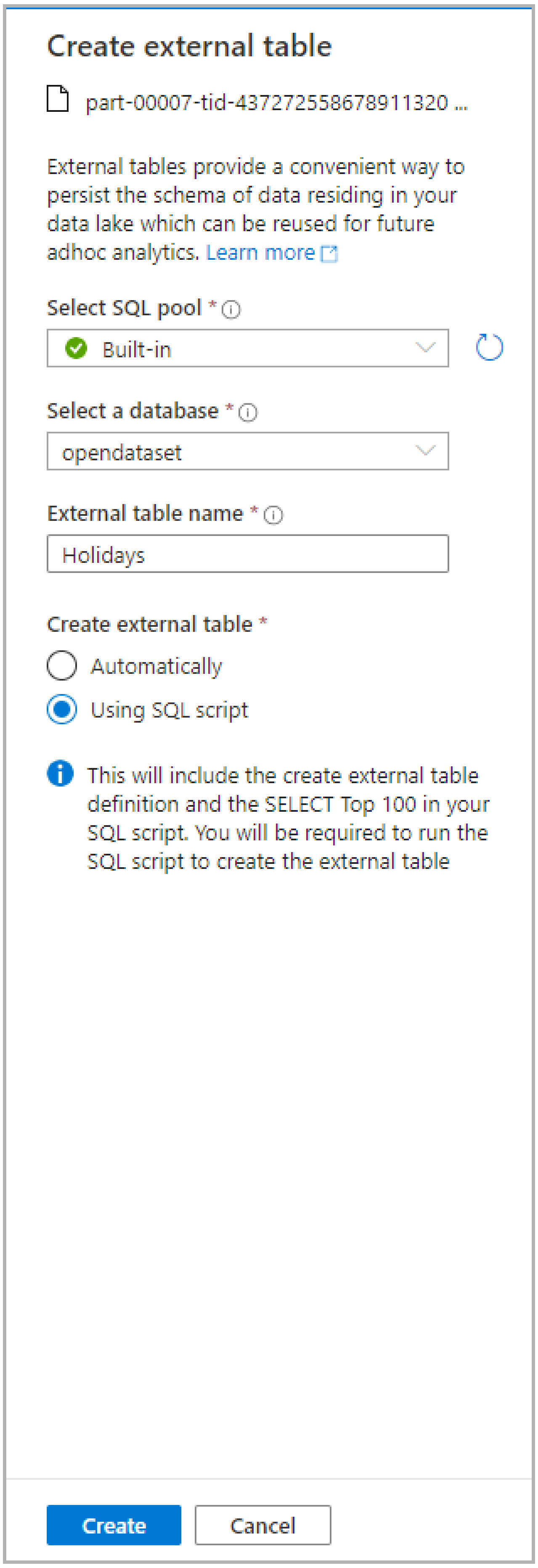
Prerequisites

- You must have access to the workspace with at least the Storage Blob Data Contributor access role to the ADLS Gen2 Account or Access Control Lists (ACL) that enable you to query the files.
- You must have at least permissions to create and query external tables on the Synapse SQL pool (dedicated or serverless).

From the Data panel, select the file that you would like to create the external table from:


Data
Workspace
Linked
Filter resources by name
Azure Blob Storage 1
Azure Cosmos DB 4
Azure Data Lake Storage Gen1 1
Azure Data Lake Storage Gen2 4
Integration datasets 10
opendataset
New SQL script
New notebook
New data flow
New integration datas
opendataset > HolidaysProcessed
Name
_committed_1185565042377273094
_committed_5308017365667606931
_committed_8129226483299294944
_committed_8636682034356181098
_committed_vacuum4457948198484105691
_started_1185565042377273094
_SUCCESS
COPY-part-00000-tid-1185565042377273094-a997e91b-a2bb-4ec4-b3a7-910f2d2f3633-649-c00
part-00000-tid-1185565042377273094-a997e91b-a2bb-4ec4-b3a7-910f2d2f3633-649-c000.snap
part-00007-tid-4372 1902-1.c000.s
New SQL script
New notebook
New data flow
New integration dataset
Manage access...
Rename...
Download
Delete
Properties...
Select TOP 100 rows
Create external table
Bulk load

A dialog window will open. Select dedicated SQL pool or serverless SQL pool, give a name to the table and select open script:


Create external table

part-00007-tid-437272558678911320 ...

External tables provide a convenient way to persist the schema of data residing in your data lake which can be reused for future adhoc analytics. Learn more

Select SQL pool *
Built-in

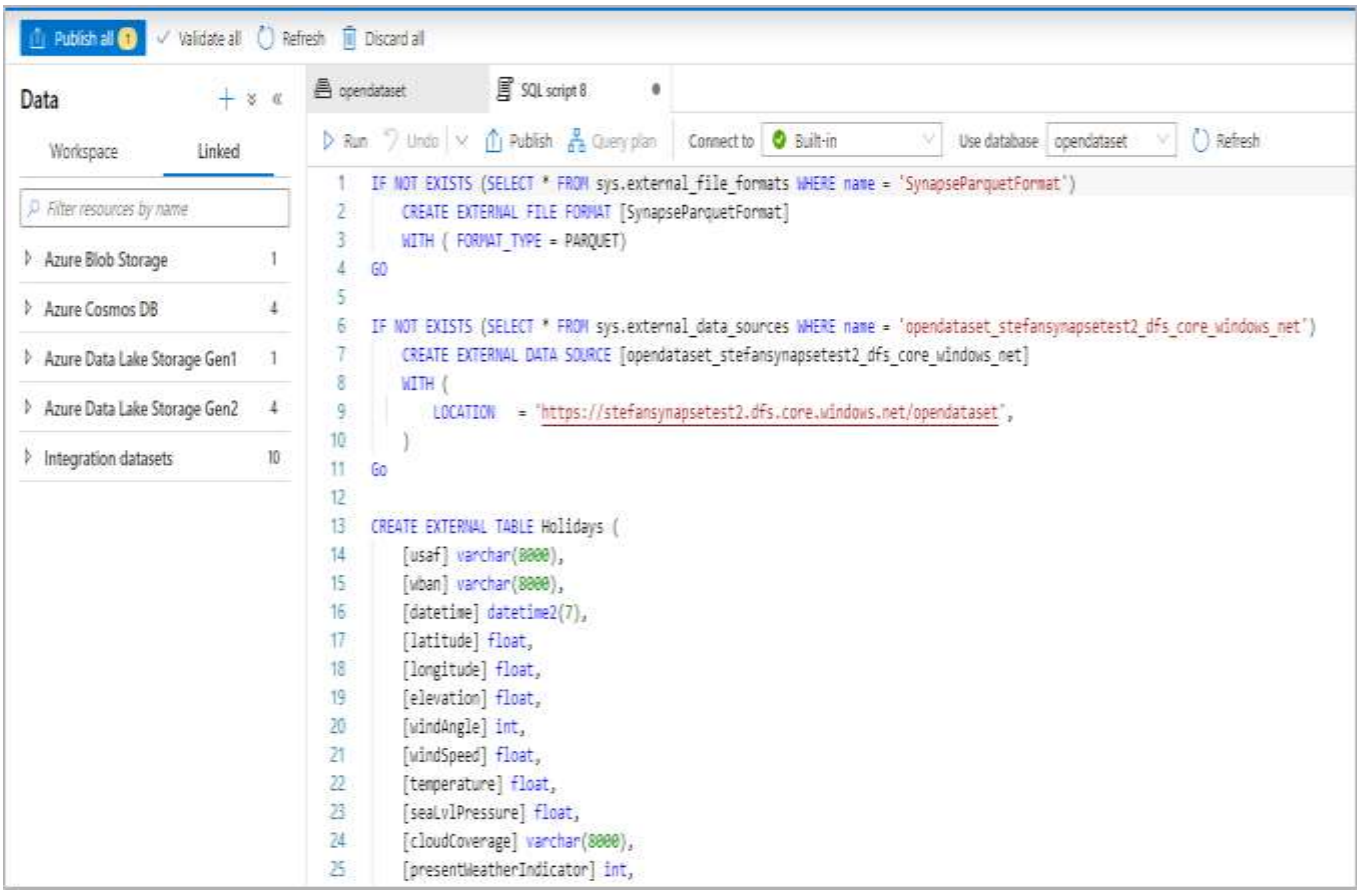
Select a database *
opendataset

External table name *
Holidays

Create external table *
○ Automatically
● Using SQL script

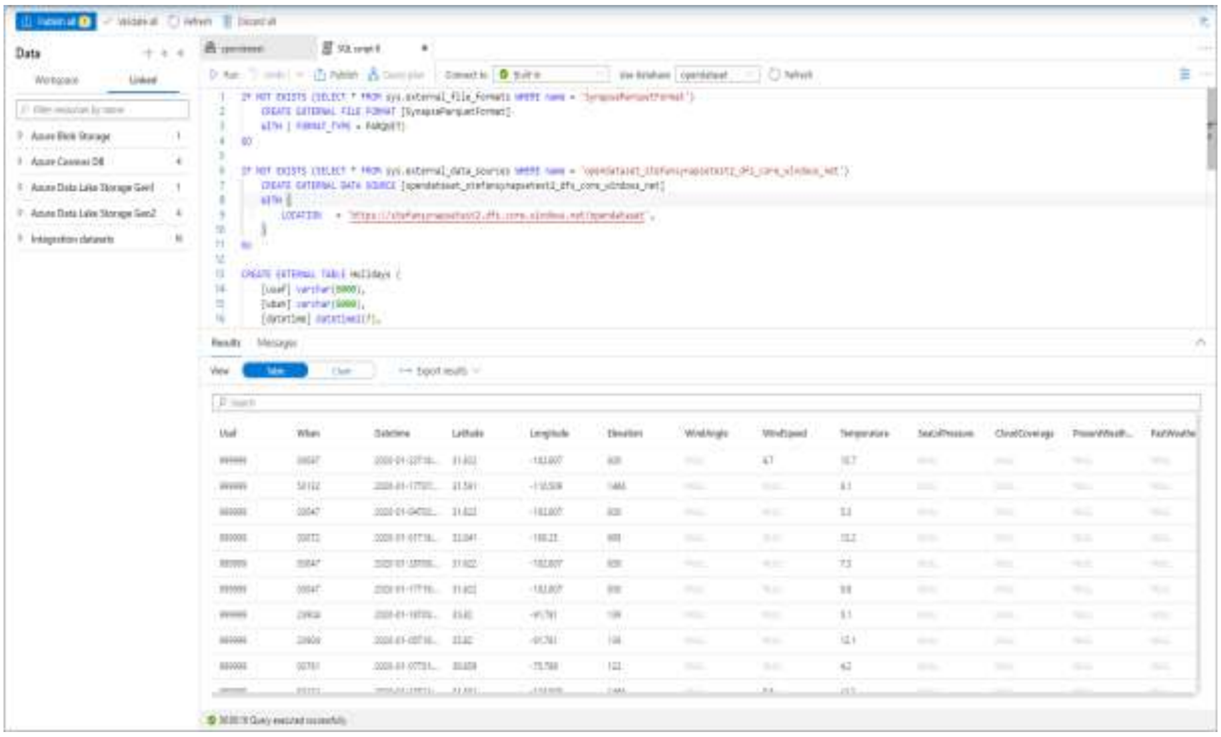
This will include the create external table definition and the SELECT Top 100 in your SQL script. You will be required to run the SQL script to create the external table

The SQL Script is autogenerated inferring the schema from the file:



```
IF NOT EXISTS (SELECT * FROM sys.external_file_formats WHERE name = 'SynapseParquetFormat')
    CREATE EXTERNAL FILE FORMAT [SynapseParquetFormat]
    WITH ( FORMAT_TYPE = PARQUET)
GO

IF NOT EXISTS (SELECT * FROM sys.external_data_sources WHERE name = 'opendataset_stefansynapsetest2_dfs_core_windows_net')
    CREATE EXTERNAL DATA SOURCE [opendataset_stefansynapsetest2_dfs_core_windows_net]
    WITH (
        LOCATION   = 'https://stefansynapsetest2.dfs.core.windows.net/opendataset',
    )
Go

CREATE EXTERNAL TABLE Holidays (
    [usaf] varchar(8000),
    [wban] varchar(8000),
    [datetime] datetime2(7),
    [latitude] float,
    [longitude] float,
    [elevation] float,
    [windAngle] int,
    [windSpeed] float,
    [temperature] float,
    [seaLvlPressure] float,
    [cloudCoverage] varchar(8000),
    [presentWeatherIndicator] int,
```

Run the script. The script will automatically run a Select Top 100 *.:

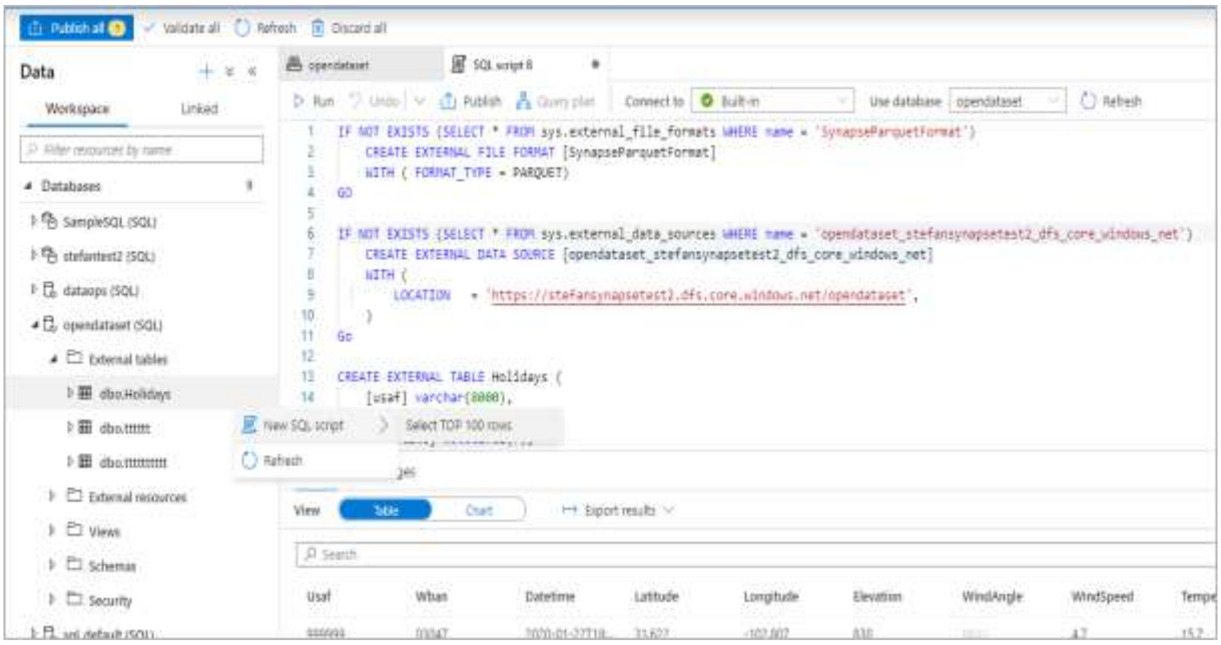


```
IF NOT EXISTS (SELECT * FROM sys.external_file_formats WHERE name = 'SynapseParquetFormat')
    CREATE EXTERNAL FILE FORMAT [SynapseParquetFormat]
    WITH ( FORMAT_TYPE = PARQUET)
GO

IF NOT EXISTS (SELECT * FROM sys.external_data_sources WHERE name = 'opendataset_stefansynapsetest2_dfs_core_windows_net')
    CREATE EXTERNAL DATA SOURCE [opendataset_stefansynapsetest2_dfs_core_windows_net]
    WITH (
        LOCATION   = 'https://stefansynapsetest2.dfs.core.windows.net/opendataset',
    )
Go

CREATE EXTERNAL TABLE Holidays (
    [usaf] varchar(8000),
    [wban] varchar(8000),
    [datetime] datetime2(7),
```

Usaf	Wban	Datetime	Latitude	Longitude	Elevation	WindAngle	WindSpeed	Temperature	SeaLvlPressure	CloudCoverage	PresentWeath...	PastWeathe...
999999	00047	2020-01-22T16:...	31.022	-102.007	820	NULL	6.7	10.7	NULL	NULL	NULL	NULL
999999	53132	2020-01-17T07:...	31.581	-110.508	1466	NULL	NULL	8.1	NULL	NULL	NULL	NULL
999999	00047	2020-01-04T02:...	31.022	-102.007	820	NULL	NULL	3.3	NULL	NULL	NULL	NULL
999999	00072	2020-01-01T16:...	33.041	-100.23	600	NULL	NULL	13.2	NULL	NULL	NULL	NULL
999999	00047	2020-01-18T00:...	31.022	-102.007	820	NULL	NULL	7.3	NULL	NULL	NULL	NULL
999999	00047	2020-01-17T16:...	31.022	-102.007	820	NULL	NULL	9.8	NULL	NULL	NULL	NULL
999999	23906	2020-01-18T05:...	33.82	-91.781	109	NULL	NULL	5.1	NULL	NULL	NULL	NULL
999999	23906	2020-01-05T16:...	33.82	-91.781	108	NULL	NULL	12.1	NULL	NULL	NULL	NULL
999999	00761	2020-01-07T03:...	30.658	-75.788	122	NULL	NULL	4.2	NULL	NULL	NULL	NULL
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

00:00:18 Query executed successfully

The external table is now created, for future exploration of the content of this external table the user can query it directly from the Data pane:



```
IF NOT EXISTS (SELECT * FROM sys.external_file_formats WHERE name = 'SynapseParquetFormat')
    CREATE EXTERNAL FILE FORMAT [SynapseParquetFormat]
    WITH ( FORMAT_TYPE = PARQUET)
GO

IF NOT EXISTS (SELECT * FROM sys.external_data_sources WHERE name = 'opendataset_stefansynapsetest2_dfs_core_windows_net')
    CREATE EXTERNAL DATA SOURCE [opendataset_stefansynapsetest2_dfs_core_windows_net]
    WITH (
        LOCATION   = 'https://stefansynapsetest2.dfs.core.windows.net/opendataset',
    )
Go

CREATE EXTERNAL TABLE Holidays (
    [usaf] varchar(8000),
```

Next steps

See the CETAS article for how to save query results to an external table in Azure Storage. Or you can start querying Apache Spark for Azure Synapse external tables.