Serverless SQL Pool

Azure Synapse SQL is a distributed query system, offers both serverless and dedicated resource models

* Serverless SQL pool is serverless, resources are not dedicated, hence there's no infrastructure to setup or clusters to maintain.
* Serverless SQL Pool uses Distributed SQL Engine “Polaris”.
* It can be used to query data in the lake also write processed data back to Data Lake
* It is used to directly query Live data using T-SQL
* The architecture is similar to dedicated SQL Pool, It has control node and various compute nodes.
* Compute node can communicate with various sources like Data Lake, CosmosDB, Spark Table
* Serverless SQL pool has no local storage, only metadata objects are stored in databases.
* It is True pay-per-use model as resources are not dedicated but allocated based on need of query.
* You can use following tools for querying Data: Azure Synapse Studio, Azure Data Studio ,SSMS

**Use Cases:**

1. **Data Exploration:**

* Using Azure Synapse studio you can easily browse the data in data lake and get insights into it.
* You can explore know the data set and then apply projections, filtering, grouping, and most of the operation over the data as if the data were in a regular SQL Server table.

1. **Data Transformation:**

* Serverless SQL pool enables you to execute a SELECT statement over the data in the lake and store the results back to the data lake in a specified format.

1. **Logical data warehouse:**

* you can start creating objects (such as VIEWs and External Tables) that provide you with a SQL metadata layer over the data in the lake.
* Clients can feel these objects as regular SQL Objects and don’t even know that the underlying data is in the data lake. This makes serverless SQL pool a powerful lightweight layer between the data lake and a client.
* You can connect from BI tools like Power BI or Azure Analysis Service, from integration tools like Synapse Pipelines or Azure Data Factory, and you can also programmatically connect using any of the popular languages like C# or Python.

**Feature Support:**

* Schema, Views, Stored Procedures, Functions,
* External Tables and Other Resources

**Not supported:**

* Tables
* Triggers
* DML

**Query Data in Data Lake options:**

* External Table
  + Can be used to read /write data to data lake
* OPENROWSET
  + Can write select query without creating any external entities
  + Can create View and stored procedure using those queries
  + Supports Virtual Functions

Query Data in data Lake

* Serverless Pool supports querying CSV, JSON, and Parquet file formats directly.
* **OPENROWSET(bulk..)** function is used to read data files in storage. It supports built in BULK Provider that enables data from a file to be read and returned as a rowset.
* The OPENROWSET function can be referenced in the FROM clause of a query with or without DATA\_SOURCE option.
* **DATA\_SOURCE** Option enables you to configure location of the storage account in the data source and specify the authentication method that should be used to access storage.

**Ref:** [**https://docs.microsoft.com/en-us/azure/synapse-analytics/sql/develop-openrowset**](https://docs.microsoft.com/en-us/azure/synapse-analytics/sql/develop-openrowset)

**Lab 1: Link Sample Dataset for nyc yellowtaxi**

Home Hub🡪Knowledge centre🡪Browse Gallery🡪Select Dataset Tab🡪Select Any Dataset🡪Continue🡪Add Dataset

Data Hub🡪Sample Dataset🡪Observe the newly added Dataset🡪Right Click🡪New SQL Script🡪Select TOP 100 rows

*Note: You can also use New Notebook option and work with it using Spark Cluster*

**Lab 2: Get data from sample dataset using serverless pool**

Go to sample dataset🡪Right click🡪New SQL Script

SELECT

    TOP 100 \*

FROM

    OPENROWSET(

        BULK     'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/puYear=\*/puMonth=\*/\*.parquet',

        FORMAT = 'parquet'

    ) AS [result];

**Lab 3: Access subset of files and folders. Get data for year 2020**

SELECT

    TOP 100 \*

FROM

    OPENROWSET(

        BULK     'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/puYear=2020/puMonth=\*/\*.parquet',

        FORMAT = 'parquet'

    ) AS [result];

|  |  |
| --- | --- |
| Access all csv files in taxi Folder | 'csv/taxi/\*.csv' |
| Access subset of files in Folder  Reads the 2017 NYC Yellow Taxi data files from the csv/taxi | 'csv/taxi/yellow\_tripdata\_2017-\*.csv' |
| Read all files from a specific folder | 'csv/taxi/' |
| Read all files from multiple foldersRead all files from all folders located in the csv folder that have names starting with t and ending with i. | 'csv/t\*i/' |
| Use wildcard at folder level and file levelread files with 2017 data only, from all folders which names start with t and end with i. | 'csv/t\*i/yellow\_tripdata\_2017-\*.csv’ |
| Traverse Folder Reciusivelyread all files from all folders and subfolders located in the csv folder. | csv/taxi/\*\*' |
|  | puYear=\*/puMonth=\*/\*.parquet'  'puYear=2020/puMonth=\*/\*.snappy.parquet' |

*Note:*

*Unlike Hadoop and PolyBase, serverless SQL pool doesn't return subfolders unless you specify /\*\* at the end of path. Just like Hadoop and PolyBase, it doesn't return files for which the file name begins with an underline (\_) or a period (.).*

Query partitioned Data

* **Filename** and **Filepath**and   return file names and path in the result set.
* Filename and Filepath can be used to filter the data to be read in specified path.

**Filename**

* returns the file name that the row originates from.

**Filepath**

* When called without parameter, returns the full file path that a row originates from.
* When called with parameter, it returns part of path that matches the wildcard on position specified in the parameter.

**Example:**

**Lab 4:Get filename and file path for specified location and get number of rows in each file**

SELECT

     nyc.filename() AS [filename],

     nyc.filepath() AS [filepath],

    COUNT\_BIG(\*) AS [rows]

FROM

    OPENROWSET(

        BULK     'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/puYear=\*/puMonth=\*/\*.parquet',

        FORMAT = 'parquet'

    ) AS [nyc]

    --WHERE  nyc.filepath(1) IN (‘2018’,’2019’)

GROUP BY nyc.filename(),nyc.filepath()

**Lab5: Create Publicly available Data source for Newyork city Yellow Taxi**

CREATE EXTERNAL DATA SOURCE YellowTaxi

WITH ( LOCATION = 'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/')

*Note: Before creating Data Source you need to* ***create database*** *in serverless pool*

**Lab 6 : Get Total number of rows passangercount wise based on particular year 2018 ans month 1,2,3 in YellowTaxi**

**Use filepath function to filter year as 2018 and 1,2,3 months**

SELECT

    YEAR(tpepPickupDateTime),

    passengerCount,

    COUNT(\*) AS cnt

FROM

    OPENROWSET(

        BULK 'puYear=\*/puMonth=\*/\*.snappy.parquet',

        DATA\_SOURCE = 'YellowTaxi',

        FORMAT='PARQUET'

    )

WITH (

        tpepPickupDateTime DATETIME2,

        passengerCount INT

    ) AS nyc

    WHERE nyc.filepath(1) = 2018

    AND nyc.filepath(2) IN (1, 2, 3)

   -- AND tpepPickupDateTime BETWEEN CAST('1/1/2018' AS datetime) AND CAST('3/31/2018' AS datetime)

GROUP BY

    passengerCount,

    YEAR(tpepPickupDateTime)

ORDER BY

    YEAR(tpepPickupDateTime),

    passengerCount;

Note:You can Explicitly specify what columns you want to read from the file using the **WITH** clause

**Assignment: Query top 100 rows for year 2017**

**Lab 7: Query publicly available CSV File**

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

**Lab 8: Query CSV File using DATA\_SOURCE option**

**create external data** source covid

with ( location = 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc\_cases' );

select top 10 \*

from openrowset(

bulk 'latest/ecdc\_cases.csv',

data\_source = 'covid',

format = 'csv',

parser\_version ='2.0',

firstrow = 2

) as rows

***Note:***

*firstrow: Skip first row in CSV file*

*HEADER\_ROW=TRUE : will result in reading column names from the header row in file*

FIELDTERMINATOR =’\t’ : Specifies the field terminator to be used.Here ‘Tab’ is used

For publicly available file or if Azure AD identity can access the file you need not provide SAS Token .

For protected files You can use Azure AD passthrough for Azure AD logins and SAS token for SQL logins.

**Lab 9: Query Csv File and specify columns you want to read from file.**

select top 10 \*

from openrowset(

bulk 'latest/ecdc\_cases.csv',

data\_source = 'covid',

format = 'csv',

parser\_version ='2.0',

firstrow = 2

) with (

date\_rep date 1,

cases int 5,

geo\_id varchar(6) 8

) as rows

*Note: The WITH clause allows you to specify columns that you want to read from files.*

*The numbers after a data type in the WITH clause represent column location, known as a column index in the CSV file.*

*use ordinal numbers to pick the subset of columns from original file*

*If HEADER\_ROW = TRUE is used, then column binding is done by column name instead of ordinal position*

**Collation and Conversion Errors:**

Refer: https://techcommunity.microsoft.com/t5/azure-synapse-analytics-blog/always-use-utf-8-collations-to-read-utf-8-text-in-

**Collation:**

A collation is a property of string types in SQL Server, Azure SQL, and Synapse SQL that defines how to compare and sort strings. In addition, it describes the encoding of string data.

If a collation name in Synapse SQL ends with UTF8, it represents the strings encoded with the UTF-8 encoding schema

**You might face conversion issues** if you use OPENROWSET without WITH clause or OPENROWSET/External table that return VARCHAR column without UTF8 collation

.

**Check default collation**

SELECT DATABASEPROPERTYEX(DB\_NAME(), 'Collation') AS Collation;

**Change the collation of database :**

ALTER DATABASE mydb

COLLATE Latin1\_General\_100\_BIN2\_UTF8;

Query JSON

* The easiest way to see the content of your JSON file is to provide the file URL to the OPENROWSET function, specify **csv FORMAT**, and set the values of **0x0b** for the **fieldterminator** and **fieldquote** variables.
* If you need to read line-delimited JSON files, then this is enough.
* If you have classic JSON file, you would need to set values **0x0b** for **rowterminator**.

|  |
| --- |
| 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 |
| **--Use Relative Path in OPENROWSET by creating data source**  create external data source covid  with (location = 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc\_cases');  go  select top 10 \*  from  openrowset(  bulk 'latest/ecdc\_cases.jsonl',  data\_source = 'covid',  format = 'csv',  fieldterminator ='0x0b',  fieldquote = '0x0b'  ) with (doc nvarchar(max)) as rows  go  **--Use Row Terminator**  select top 10 \*  from  openrowset(  bulk 'latest/ecdc\_cases.json',  data\_source = 'covid',  format = 'csv',  fieldterminator ='0x0b',  fieldquote = '0x0b',  **rowterminator** = '0x0b' --> You need to override rowterminator to read classic JSON  ) with (doc nvarchar(max)) as rows |
| **--Query JSON files using JSON\_VALUE**  --To extract individual values from the JSON, you can use the JSON\_VALUE function  select  JSON\_VALUE(doc, '$.date\_rep') AS date\_reported,  JSON\_VALUE(doc, '$.countries\_and\_territories') AS country,  JSON\_VALUE(doc, '$.cases') as cases,  doc  from  openrowset(  bulk 'latest/ecdc\_cases.jsonl',  data\_source = 'covid',  format = 'csv',  fieldterminator ='0x0b',  fieldquote = '0x0b'  ) with (doc nvarchar(max)) as rows  order by JSON\_VALUE(doc, '$.geo\_id') desc |

Create metadata objects in serverless pools

**Create Database**

Data Hub🡪Workspace🡪+SQL Database🡪Serverless🡪Database name:serverlessDemoDb🡪Create

OR

Create database serverlessDemoDb

**Create Credentials:**

Azure Synapse serverless SQL pool accesses the storage to read the files using credentials. There are 3 types of credentials that are supported:

* Azure Active Directory pass-through
* Managed Identity
* Shared access signature (SAS)

|  |  |
| --- | --- |
| **Azure Active Directory pass-through**  Identity of logged in User is used | CREATE DATABASE SCOPED CREDENTIAL [sqlondemand]  WITH IDENTITY='User Identity' |
| **Managed Identity**  Manged identity for synapse workspace is created in AD   You can grant access to services to this identity explicitly | CREATE DATABASE SCOPED CREDENTIAL [sqlondemand]  WITH IDENTITY='Managed Identity' |
| **Shared access signature (SAS).**  SAS enables you to give a time bound permissions at a storage account file system or directory level | 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%2FJX2lguw%3D' |

**Create external data sources**

 External data source is used to define the location of the data

|  |  |
| --- | --- |
| **Use public storage account** | CREATE EXTERNAL DATA SOURCE YellowTaxi  WITH ( LOCATION = 'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/') |
| **Use CREDENTIAL to access protected account** | 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%2FJX2lguw%3D'  **CREATE EXTERNAL DATA SOURCE SqlOnDemandDemo WITH (**  **LOCATION = 'https://sqlondemandstorage.blob.core.windows.net',**  **CREDENTIAL = sqlondemand**  **);** |

**Lab 10:Create credential using SAS AND Data Source for YellowTaxi and sqlondemand asShown above.**

**Create File Format**

**Lab 10: Create file format for population.csv in sqlondemand datasource**

CREATE EXTERNAL FILE FORMAT QuotedCsvWithHeaderFormat

WITH (

FORMAT\_TYPE = DELIMITEDTEXT,

FORMAT\_OPTIONS ( FIELD\_TERMINATOR = ',', STRING\_DELIMITER = '"', FIRST\_ROW = 2 )

);

GO

**Lab 11: Create file format for parquet file**

CREATE EXTERNAL FILE FORMAT ParquetFormat WITH ( FORMAT\_TYPE = PARQUET );

**Create external tables**

* External tables are useful when you want to control access to external data in serverless SQL pools
* They are commonly used in PolyBase activities
* External table can access public storage file or protected storage files using credentials.

Prerequisites:

1. Create Credentials
2. Create Data source
3. Create File Format

**Lab 12: Create External Table for population.csv present in location as ('csv/population/population.csv')**

USE [mydbname];

GO

CREATE EXTERNAL TABLE populationExternalTable

(

[country\_code] VARCHAR (5) COLLATE Latin1\_General\_BIN2,

[country\_name] VARCHAR (100) COLLATE Latin1\_General\_BIN2,

[year] smallint,

[population] bigint

)

WITH (

LOCATION = 'csv/population/population.csv',

DATA\_SOURCE = **SqlOnDemandDemo**,

FILE\_FORMAT = QuotedCsvWithHeaderFormat

);

***Note:*** Collation is a set of rules that tell database engine how to compare and sort the character data in SQL Server.

SELECT SERVERPROPERTY('collation') gives you collation for sql server.

**Lab 13:Create External Table for Yellow Taxi for year 2019**

CREATE EXTERNAL TABLE YellowTaxi2019 (

    [vendorID] varchar(8000),

    [tpepPickupDateTime] datetime2(7),

    [tpepDropoffDateTime] datetime2(7),

    [passengerCount] int,

    [tripDistance] float,

    [puLocationId] varchar(8000),

    [doLocationId] varchar(8000),

    [startLon] float,

    [startLat] float,

    [endLon] float,

    [endLat] float,

    [rateCodeId] int,

    [storeAndFwdFlag] varchar(8000),

    [paymentType] varchar(8000),

    [fareAmount] float,

    [extra] float,

    [mtaTax] float,

    [improvementSurcharge] varchar(8000),

    [tipAmount] float,

    [tollsAmount] float,

    [totalAmount] float

    )

    WITH (

    LOCATION = '/puYear=2019/puMonth=\*/\*.parquet',

    DATA\_SOURCE = [YellowTaxi],

    FILE\_FORMAT = [ParquetFormat ]

    )

Go

Select top 100 \* from YellowTaxi2019

**Lab 13: Query using External Table**

USE [mydbname];

GO

SELECT

country\_name, population

FROM populationExternalTable

WHERE

[year] = 2019

ORDER BY

[population] DESC;

**Reference:**

<https://docs.microsoft.com/en-us/azure/synapse-analytics/sql/query-data-storage>

**Create View**

* Views will allow you to reuse queries that you create.
* Views are also needed if you want to use tools such as Power BI to access the data in conjunction with serverless SQL pools.

|  |
| --- |
| USE [mydbname];  GO    DROP VIEW IF EXISTS populationView;  GO    CREATE VIEW populationView AS  SELECT \*  FROM OPENROWSET(  BULK 'csv/population/population.csv',  DATA\_SOURCE = 'SqlOnDemandDemo',  FORMAT = 'CSV',  FIELDTERMINATOR =',',  ROWTERMINATOR = '\n'  )  WITH (  [country\_code] VARCHAR (5) COLLATE Latin1\_General\_BIN2,  [country\_name] VARCHAR (100) COLLATE Latin1\_General\_BIN2,  [year] smallint,  [population] bigint  ) AS [r]; |
| USE [mydbname];  GO    SELECT  country\_name, population  FROM populationView  WHERE  [year] = 2019  ORDER BY  [population] DESC; |

**Create Partitioned View:**

CREATE VIEW TaxiView

AS SELECT \*, nyc.filepath(1) AS [year], nyc.filepath(2) AS [month]

FROM

OPENROWSET(

BULK 'parquet/taxi/year=\*/month=\*/\*.parquet',

DATA\_SOURCE = 'sqlondemanddemo',

FORMAT='PARQUET'

) AS nyc

Transform Data Using CETAS

**CETAS** used to create an external table, which exports the query results to Azure Storage Blob or Azure Data Lake Storage Gen2

**It** is used to Transform data using T-SQL and Export the query result to storage.

**Syntax:**

CREATE EXTERNAL TABLE [ [database\_name . [ schema\_name ] . ] | schema\_name . ] table\_name

WITH (

LOCATION = 'path\_to\_folder',

DATA\_SOURCE = external\_data\_source\_name,

FILE\_FORMAT = external\_file\_format\_name

)

AS <select\_statement>

[;]

<select\_statement> ::=

[ WITH <common\_table\_expression> [ ,...n ] ]

SELECT <select\_criteria>

|  |
| --- |
| -- you might need to execute following statement if you never did so on current database  -- CREATE MASTER KEY  CREATE DATABASE SCOPED CREDENTIAL destination\_credential  WITH IDENTITY='SHARED ACCESS SIGNATURE',  SECRET = '' -- fill in your SAS key that will be used for CETAS destination  GO  CREATE EXTERNAL DATA SOURCE destination\_ds  WITH  (  LOCATION = 'https://<storage>.<dfs>.core.windows.net/<container>' – replace path to match root path for your CETAS destination  , CREDENTIAL = destination\_credential  )  GO  CREATE EXTERNAL FILE FORMAT parquet\_file\_format  WITH  (  FORMAT\_TYPE = PARQUET,  DATA\_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'  )  GO  -- use CETAS to export select statement with OPENROWSET result to storage  CREATE EXTERNAL TABLE aggregated\_data  WITH (  LOCATION = 'aggregated\_data/',  DATA\_SOURCE = destination\_ds,  FILE\_FORMAT = parquet\_file\_format  )  AS  SELECT decennialTime, stateName, SUM(population) AS population  FROM  OPENROWSET(BULK 'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us\_population\_county/year=\*/\*.parquet',  FORMAT='PARQUET') AS [r]  GROUP BY decennialTime, stateName  GO  -- you can query the newly created external table  SELECT \* FROM aggregated\_data |

Encapsulate Data Transformation logic using Stored Procedure

You can prepare the transformation logic using T-SQL Query and encapsulate it in stored procedure.

**Stored Procedure Benefits:**

* Reduce client -server n/w traffic
* Provide security boundary
* Eases Maintenance
* Improved Performance

|  |
| --- |
| -- this sample references external data source and external file format defined in previous section  CREATE PROCEDURE usp\_calculate\_population\_by\_year\_state  AS  BEGIN  -- Drop the table if it already exists  IF EXISTS (  SELECT \* FROM sys.external\_tables  WHERE name = ' population\_by\_year\_state '  )  DROP EXTERNAL TABLE population\_by\_year\_state  -- Create external table  CREATE EXTERNAL TABLE population\_by\_year\_state  WITH (  LOCATION = 'population\_by\_year\_state/',  DATA\_SOURCE = destination\_ds,  FILE\_FORMAT = parquet\_file\_format  )  AS  SELECT decennialTime, stateName, SUM(population) AS population  FROM  OPENROWSET(BULK 'https://azureopendatastorage.blob.core.windows.net/censusdatacontainer/release/us\_population\_county/year=\*/\*.parquet',  FORMAT='PARQUET') AS [r]  GROUP BY decennialTime, stateName  END  GO |

**Lab:Transform YellowTaxi Data and Write Result back To Data Lake using CETAS**

**Refer:** TransformAndWriteToLake.sql