Microsoft

Azure Synapse Analytics
SQL on-demand

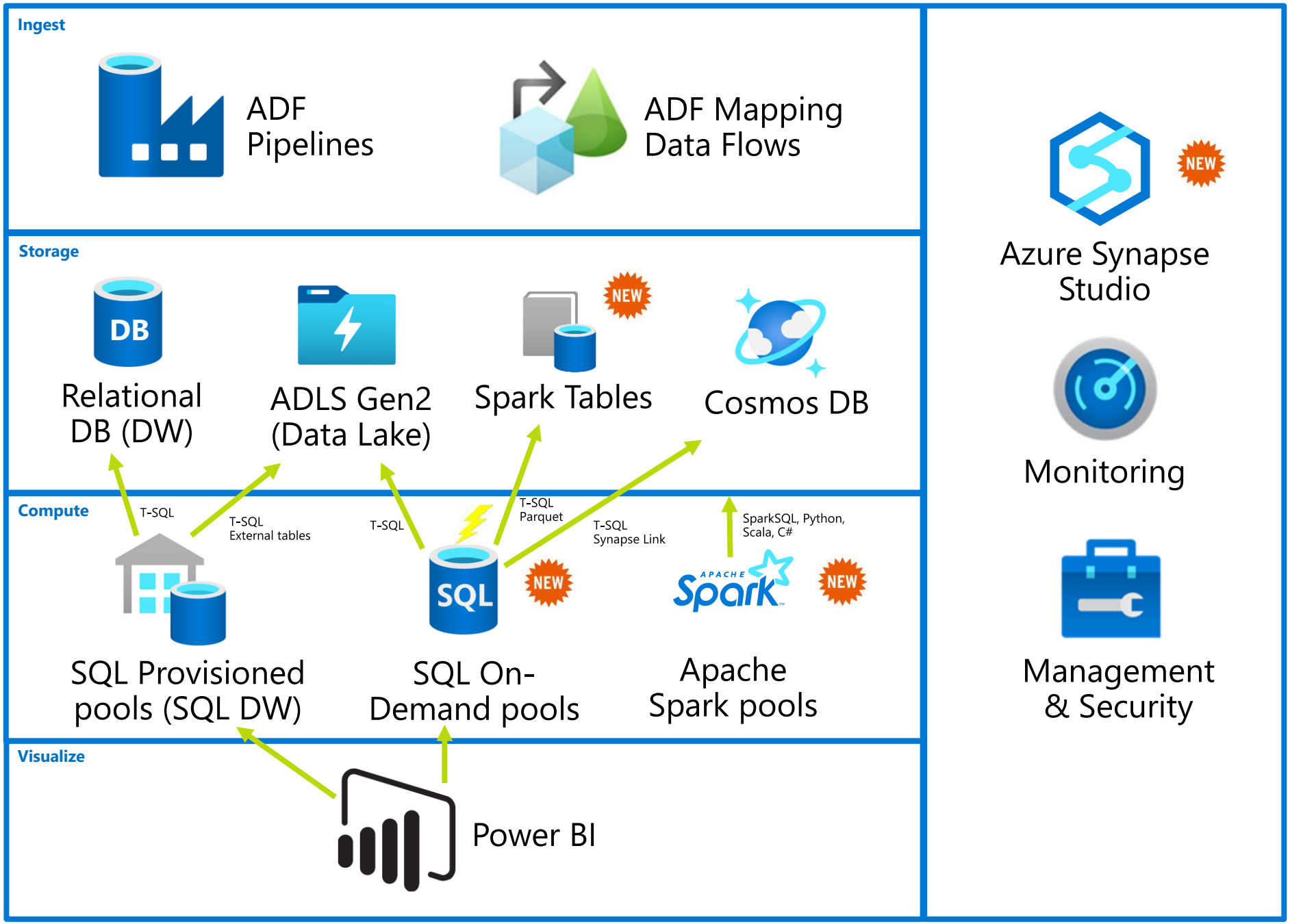
Azure Synapse Analytics SQL serverless pool

Andrea Benedetti

Sr. Cloud Architect / Data & AI Engineer | Microsoft Italia

/in/abenedetti @anBenedetti https://github.com/anbened

Azure Synapse Analytics (workspaces)



Azure Synapse Studio

Monitoring

Management & Security

SQL On-demand pools will be renamed to Serverless SQL pools

SQL Provisioned pools/SQL pools will be renamed to Dedicated SQL pools

3 main scenarios that SQL on-demand is great for

Basic discovery and exploration

Quickly view the data → extract insights

Logical data warehouse

Relational abstraction on top of raw

Always up-to-date view

T-SQL → blurring the line between a relational database and a data lake

Data transformation

Simple, scalable, and performant way to transform data in the lake using T-SQL

- For example, using the Copy activity in Azure Data Factory you can convert CSV files in the data lake (via T-SQL views in SQL on-demand) to Parquet files in the data lake

Who will benefit from?

Data Engineers

can explore the lake, then transform the data in ad-hoc queries or build a logical data warehouse with reusable queries

Data Scientists

can explore the lake to build up context about the contents and structure of the data in the lake and ultimately contribute to the work of the data engineer.

- Features such as OPENROWSET and automatic schema inference are useful in this scenario.

Data Analysts

can explore data (created by Data Scientists / Data Engineers) using familiar T-SQL language or their favorite tools that support connection to SQL on-demand

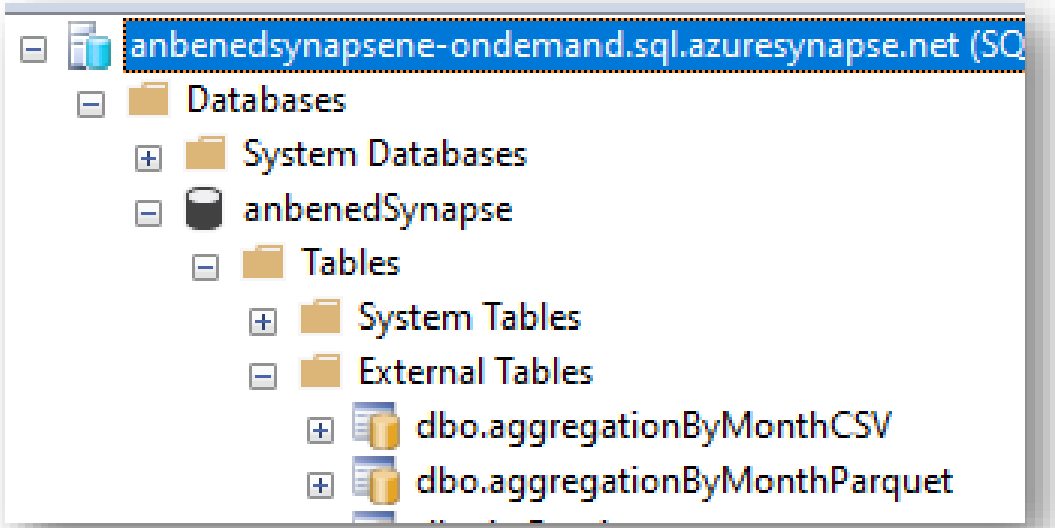
BI Professionals

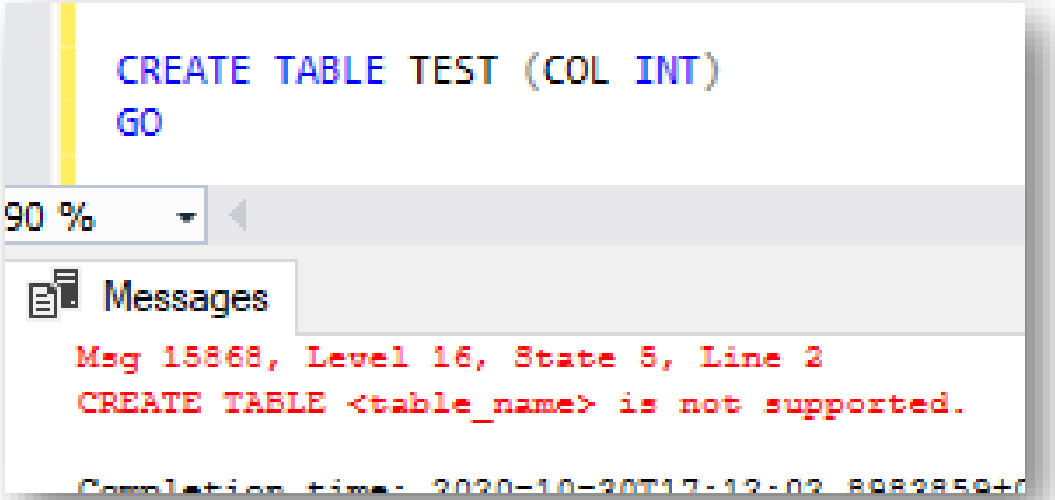
can quickly create Power BI reports on top of data in the lake

Usage Patterns

Plese, keep in mind that not exist a «physical» TABLE in sql on-demand

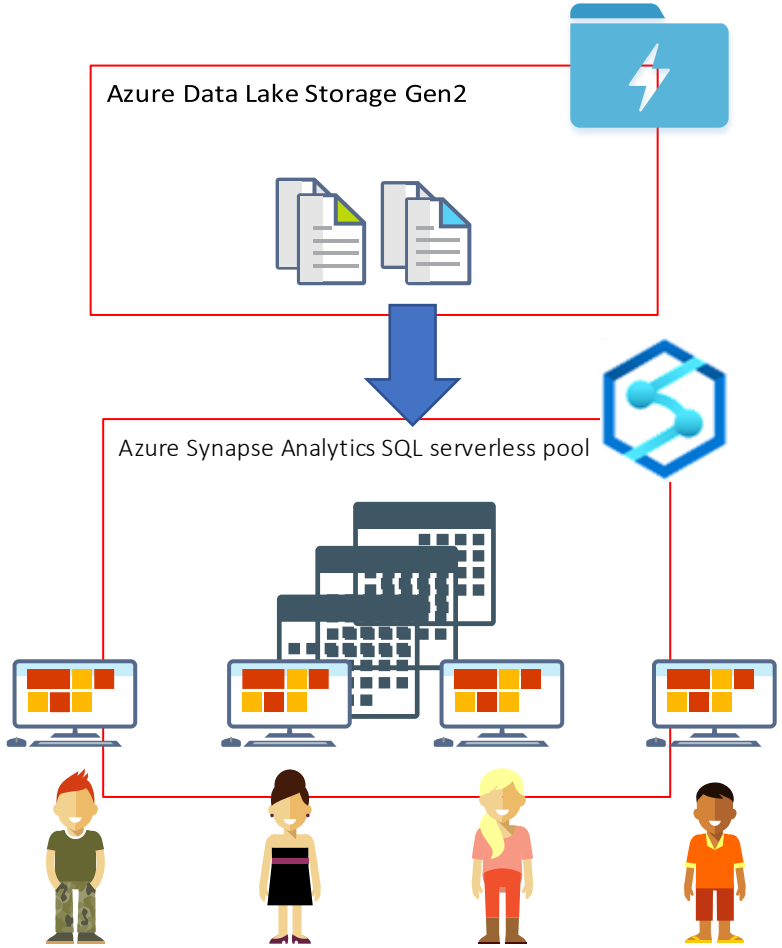
Only EXTERNAL TABLE



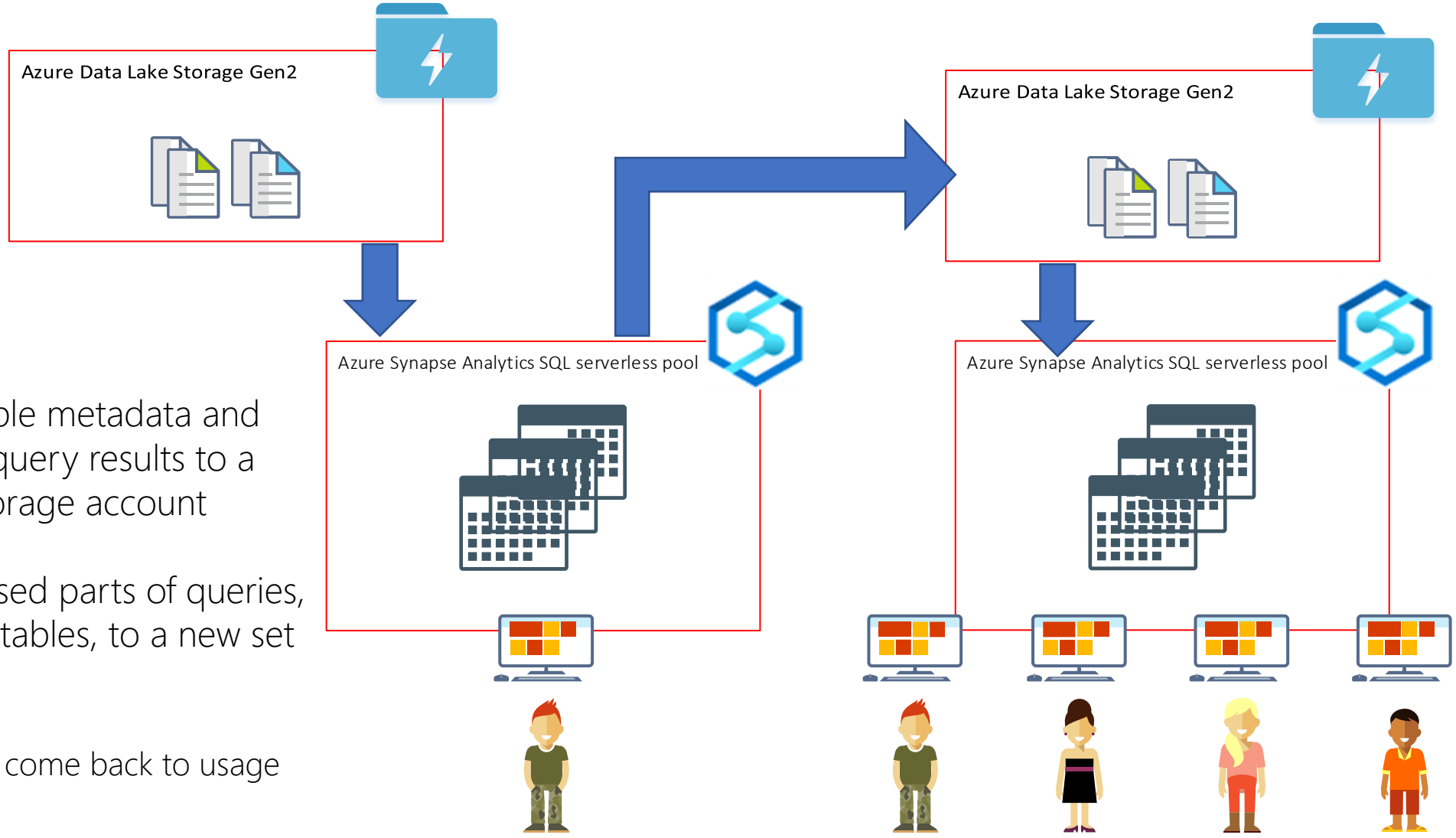


1) Discover / explore data in data lake

- Explore files with T-SQL
- Create External Tables / Views over files
- Join disparate data if needed
- Logical DWH by creating a relational abstraction on top of raw
- You can transform your data to satisfy whichever model you want
- Quickly create (Power BI) reports on top of data in the lake
- …



2) Store query results to storage



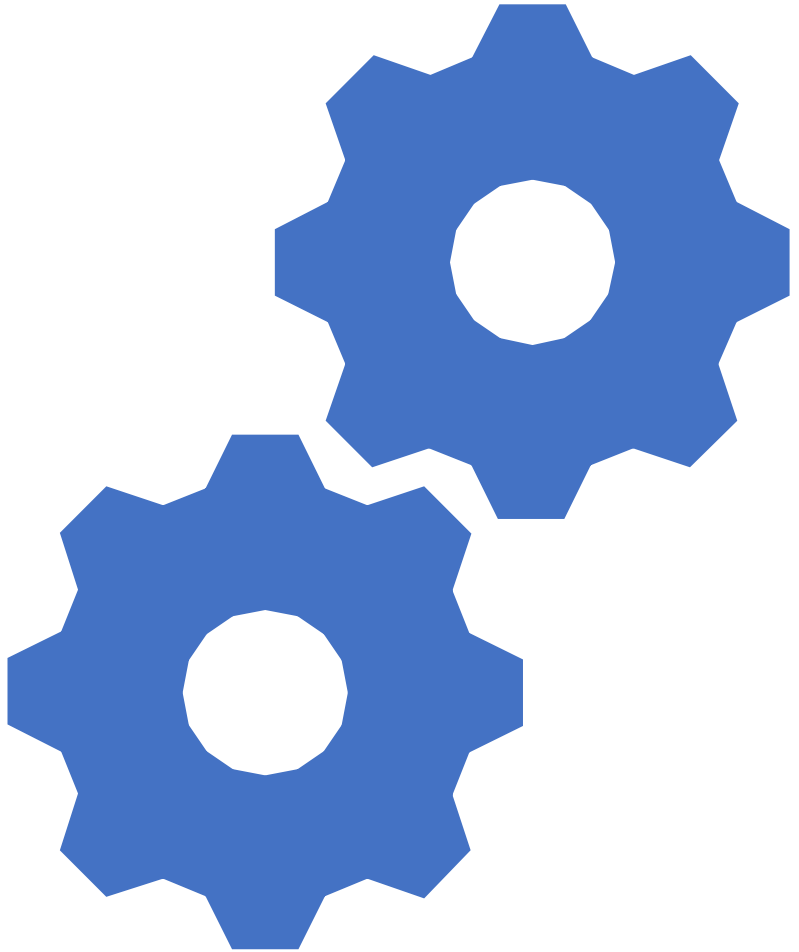
- To create external table metadata and exports the SELECT query results to a set of files in your storage account
- To store frequently used parts of queries, like joined reference tables, to a new set of files
- When stored… you can come back to usage pattern (1)

Tools

- Tools
  - SQL Server Management Studio
  - Azure Data Studio
  - Azure Synapse Studio
  - Any tool/library that uses standard SQL can access SQL on-demand

- Limitations
  - Max query duration 30min
  - ~10TB max data that can be processed per query
  - Use desktop tools (ADS, SSMS) instead of Synapse Studio if you are returning multi-GB of data per query.
    - Web interface is not designed for huge data exports.

Setup

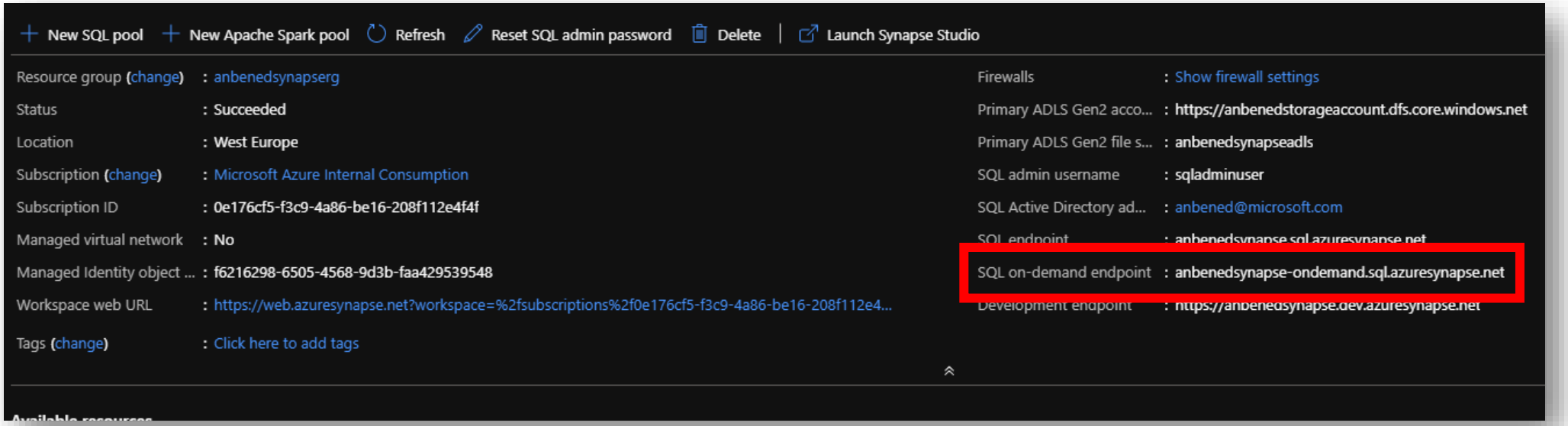


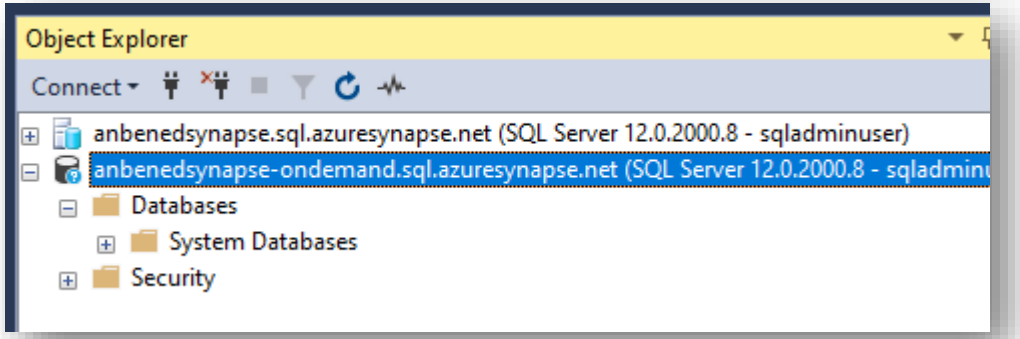
Setup

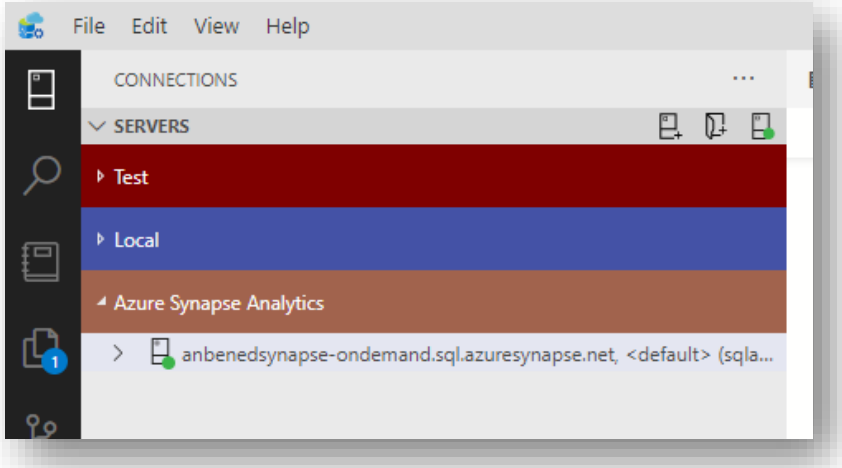
- Just 1 thing → provisioning Azure Synapse Analytics

- SQL on-demand is immediately available for your workspace
- SQL pools can be configured to adapt to team or organizational requirements and constraints
  - https://github.com/Azure/azure-synapse-analytics/blob/master/docs/quickstart-create-a-sqlpool.md

SQL on-demand Endpoint

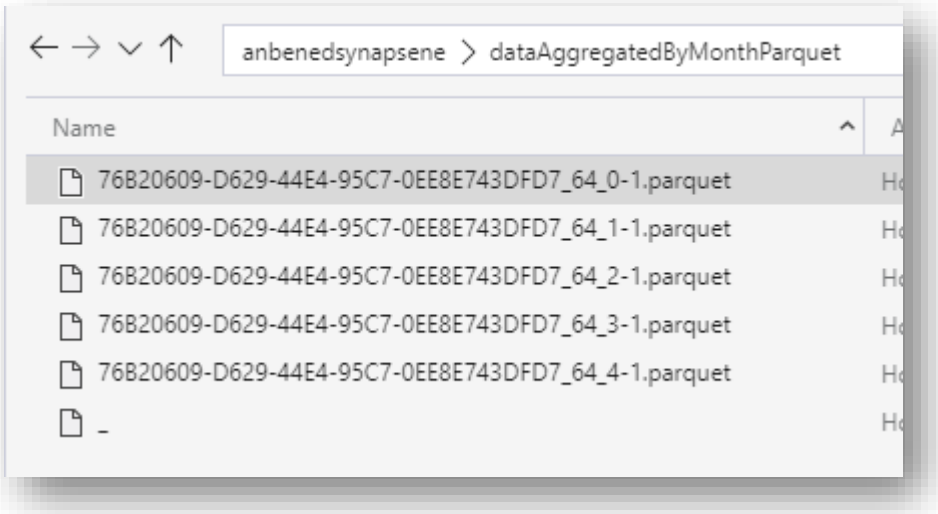






Demo

1. Azure Synapse Studio
2. Accessing Data



Supported File Formats and Concerns

- Currently, **CSV** (including TSV), **Apache Parquet**, and **JSON** (semi-structured) format are supported in SQL on-demand

- For performance perspective, it will be recommended to use Apache **Parquet** (columnar-base format), but there exist another reason for using Apache Parquet in SQL on-demand
  - The schema for underlying files can be auto-detected (inferred) in SQL
    - However, currently, this schema inference works only for PARQUET format
  - When you use CSV, you should specify all columns in schema description by WITH clause in OPENROWSET

Querying different file formats

Overview

Use OPENROWSET function to access data stored in various file formats

Benefits

Enables you to read CSV, parquet, and JSON files

Provides unified T-SQL interface for all file types

Use standard SQL language to transform and analyze returned data

- Use JSON functions to get the data from underlying files.
- Use JSON functions to get data from PARQUET nested types

```
SELECT TOP 10 *
FROM OPENROWSET(
        BULK 'https://XYZ.blob.core.windows.net/csv/taxi/*.csv',
        FORMAT = 'CSV')
WITH (
          country_code VARCHAR(4),
          country_name VARCHAR(50),
          year INT,
          population INT
          ) AS nyc
```

```
SELECT TOP 10 *
FROM OPENROWSET(
        BULK 'https://XYZ.blob.core.windows.net/csv/taxi/*.parquet',
        FORMAT = 'PARQUET') AS nyc
```

```
SELECT TOP 10 *
   JSON_VALUE(jsonContent, '$.countryCode') AS country_code,
   JSON_VALUE(jsonContent, '$.countryName') AS country_name,
   JSON_VALUE(jsonContent, '$.year')  AS year
   JSON_VALUE(jsonContent, '$.population')  AS population
FROM  OPENROWSET(
      BULK 'https://XYZ.blob.core.windows.net/json/taxi/*.json',
      FORMAT='CSV',
      FIELDTERMINATOR ='0x0b',
      FIELDQUOTE = '0x0b',
      ROWTERMINATOR = '0x0b'
   )
   WITH ( jsonContent varchar(MAX)   ) AS json_line
```

	country_code	country_name	year	population
1	LU	Luxembourg	2017	594130

Notes for CSV files

- Parser version 2.0 supports following formats only:
  2019-10-15 13:20:11
  and
  1998-03-10

```
BULK N'https://anbenedstorageaccount.blob.c
        FORMAT = 'CSV',
        PARSER_VERSION='2.0',
        FIELDTERMINATOR =':',
```

- PARSER_VERSION=2.0 is very strict in respect to datetime2 support (we are working on enhancing it)

- Parser version 2.0 is much faster than version 1.0

```
SELECT count(*)
FROM
    OPENROWSET(
        BULK 'json/books/*.json',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT='CSV',
        FIELDTERMINATOR ='0x0b',
        FIELDQUOTE = '0x0b',
        ROWTERMINATOR = '0x0b'
    )
    WITH (
        content varchar(8000)
    ) AS books;
--> ~ 30 seconds
```

```
SELECT count(*)
FROM
    OPENROWSET(
        BULK 'json/books/*.json',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT='CSV',
        PARSER_VERSION='2.0',
        FIELDTERMINATOR ='0x0b',
        FIELDQUOTE = '0x08',
        ROWTERMINATOR = '0x04'
    )
    WITH (
        content varchar(8000)
    ) AS books;
--> ~ 15 seconds
```

Use a View to encapsulate logic

```
CREATE VIEW VW_PROV AS
(
    SELECT * FROM OPENROWSET
        (
            BULK N'https://anbenedstorageaccount.blob.core.windows.net/anbenedsynapseadls/dpc-covid19-ita-province.csv',
            FORMAT = 'CSV',
            PARSER_VERSION='2.0',
            FIELDTERMINATOR =',',
            ROWTERMINATOR = '\n',
            FIRSTROW = 2 --> header row
        )
        WITH
        (
            data varchar(50),stato varchar(50),codice_regione varchar(50),denominazione_regione varchar(50),
            codice_provincia varchar(50),denominazione_provincia varchar(50),sigla_provincia varchar(50),
            lat varchar(50),long varchar(50),totale_casi varchar(50),note varchar(50)
        ) as _rows
)
GO

SELECT * FROM VW_PROV
WHERE
    denominazione_regione = 'Lombardia'
```

Results | Messages

	data	stato	codice_regione	denominazione_regione	codice_provincia	denominazione_provincia	sig
1	2020-02-24T18:00:00	ITA	03	Lombardia	012	Varese	VA
2	2020-02-24T18:00:00	ITA	03	Lombardia	013	Como	CO
3	2020-02-24T18:00:00	ITA	03	Lombardia	014	Sondrio	SO
4	2020-02-24T18:00:00	ITA	03	Lombardia	015	Milano	MI
5	2020-02-24T18:00:00	ITA	03	Lombardia	016	Bergamo	BG

Check where you run your queries

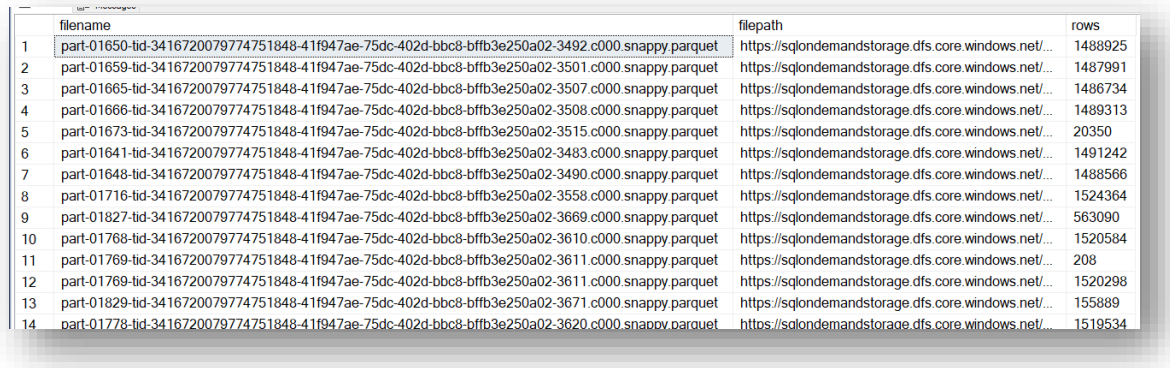
```
/* Check Synapse on-demand */


if db_name() = 'master'
    throw 50001, 'This script cannot be executed in master database. Create new database and run the script there.', 1;


if SERVERPROPERTY('EngineEdition') <> 11
    throw 50001, 'This script must be executed on Azure Synapse - SQL serverless endpoint.', 1;
```

Metadata information (using file metadata in queries)

```
/* OPENROWSET can return extra metadata information beyond FileName and FilePath when using SQL OnDemand?
NOT yet, but in development */
SELECT TOP  100
nyc.filename() AS [filename],
nyc.filepath() as [filepath],
COUNT_BIG(*) AS [rows]
FROM
    OPENROWSET(
        BULK 'parquet/taxi/year=*/month=*/*.parquet',
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT='PARQUET'
    ) AS nyc
/* WHERE nyc.filecreateddatetime() > '2020-07-01 13:00:00' */
GROUP BY
    nyc.filename(), nyc.filepath()
GO
```



	filename	filepath	rows
1	part-01650-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3492.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	1488925
2	part-01659-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3501.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	1487991
3	part-01665-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3507.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	1486734
4	part-01666-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3508.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	1489313
5	part-01673-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3515.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	20350
6	part-01641-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3483.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	1491242
7	part-01648-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3490.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	1488566
8	part-01716-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3558.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	1524364
9	part-01827-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3669.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	563090
10	part-01768-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3610.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	1520584
11	part-01769-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3611.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	208
12	part-01769-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3611.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	1520298
13	part-01829-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3671.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	155889
14	part-01778-tid-3416720079774751848-41f947ae-75dc-402d-bbc8-bffb3e250a02-3620.c000.snappy.parquet	https://sqlondemandstorage.dfs.core.windows.net/...	1519534

Schema inference

Overview

OPENROWSET will automatically determine columns and types of data stored in external file.

Benefits

No need to up-front analyze file structure to query the file

OPENROWSET identifies columns and their types based on underlying file metadata.

Perfect solution for data exploration where schema is unknown.

Currently available only for parquet files.

```
SELECT TOP 10 *
FROM OPENROWSET(
        BULK 'https://XYZ.blob.core.windows.net/csv/taxi/*.parquet',
        FORMAT = 'PARQUET') AS nyc
```

	country_code	country_name	year	population
1	LU	Luxembourg	2017	594130

Inline defined result schema

Overview

Specify columns and types at query time.

Benefits

Define result schema at query time in WITH clause.

No need for external format files.

Explicitly define exact return types, their sizes, and collations.

Improve performance by column elimination in parquet files.

```
SELECT TOP 10 *
FROM OPENROWSET(
        BULK 'https://XYZ.blob.core.windows.net/csv/taxi/*.csv',
        FORMAT = 'CSV')
WITH (
        country_code VARCHAR(4),
        country_name VARCHAR(50),
        year INT,
        population INT
        ) AS nyc
```

	country_code	country_name	year	population
1	LU	Luxembourg	2017	594130

Demo

1. Inferred Type

Check inferred data types

- Schema inference helps you quickly write queries and explore data without knowing file schemas

```
EXEC sp_describe_first_result_set
  N'
    SELECT
        TerritoryID, Name, CountryRegionCode, [Group], SalesYTD, SalesLastYear, CostYTD, CostLastYear, rowguid, ModifiedDate
    FROM
        OPENROWSET(
            BULK ''https://anbenedstorageaccount.dfs.core.windows.net/anbenedsynapseadls/Sales_SalesTerritory_20200723.parquet'',
            FORMAT=''PARQUET''
        ) AS [r]
  ';
```

00 %

Results | Messages

	is_hidden	column_ordinal	name	is_nullable	system_type_id	system_type_name	max_length	precision	scale	collation_name	user_
1	0	1	TerritoryID	1	56	int	4	10	0	NULL	NUL
2	0	2	Name	1	167	varchar(8000)	8000	0	0	SQL_Latin1_General_CP1_CI_AS	NUL
3	0	3	CountryRegionCode	1	167	varchar(8000)	8000	0	0	SQL_Latin1_General_CP1_CI_AS	NUL
4	0	4	Group	1	167	varchar(8000)	8000	0	0	SQL_Latin1_General_CP1_CI_AS	NUL
5	0	5	SalesYTD	1	108	numeric(38,18)	17	38	18	NULL	NUL
6	0	6	SalesLastYear	1	108	numeric(38,18)	17	38	18	NULL	NUL
7	0	7	CostYTD	1	108	numeric(38,18)	17	38	18	NULL	NUL
8	0	8	CostLastYear	1	108	numeric(38,18)	17	38	18	NULL	NUL
9	0	9	rowguid	1	167	varchar(8000)	8000	0	0	SQL_Latin1_General_CP1_CI_AS	NUL
10	0	10	ModifiedDate	1	42	datetime2(7)	8	27	7	NULL	NUL

Check inferred data types

- Schema inference can be used with OPENROWSET/view
  - currently supported formats (CSV, parquet) do NOT have max character column length metadata
    - schema inference defaults to 8000 for character columns
  - large character columns hinder performance
    - especially when used in DISTINCT, JOIN, WHERE, GROUP BY, ORDER BY;
    - if performance is not good enough for you, you might want to explicitly specify schema as specified in best practices
- If you reference the same external table in your query twice, query optimizer will know that you are referencing the same object twice, while 2 same OPENROWSETs will not be recognized as the same object
  - For this reason, at this moment, in such cases better execution plans could be generated when using external tables instead of OPENROWSETs

Customize parsing

Overview

Uses OPENROWSET function to access data from various types of CSV files.

Benefits

Ability to read CSV files with custom format

- With or without header row
- Handle any new-line terminator (Windows or Unix style)
- Use custom field terminator and quote character
- Read UTF-8 and UTF-18 encoded files
- Use only a subset of columns by specifying column position after column types

```
SELECT *
FROM OPENROWSET(
    BULK 'https://XYZ.blob.core.windows.net/csv/population/population.csv',
    FORMAT = 'CSV',
    FIELDTERMINATOR =',',
    ROWTERMINATOR = '\n'
)
WITH (
    [country_code] VARCHAR (5) 2,
    [country_name] VARCHAR (100) 4,
    [year] smallint 7,
    [population] bigint 9
) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017
```

Second, fourth, seventh and ninth columns are returned

	country_code	country_name	year	population
1	LU	Luxembourg	2017	594130

Querying multiple files

Overview

Uses OPENROWSET function to access data from multiple files or folders using wildcards in path

Benefits

Offers reading multiple files/folders through usage of wildcards

Offers reading specific file/folder

Supports use of multiple wildcards

```
SELECT YEAR(pickup_datetime) as [year],
        SUM(passenger_count) AS passengers_total,
        COUNT(*) AS [rides_total]
FROM OPENROWSET(
       BULK 'https://XYZ.blob.core.windows.net/csv/taxi/year=*/month=1/*.parquet',
       FORMAT = 'PARQUET')  AS nyc
GROUP BY YEAR(pickup_datetime)
ORDER BY YEAR(pickup_datetime)
```

	year	passengers_total	rides_total
1	2001	14	10
2	2002	29	16
3	2003	22	16
4	2008	378	188
5	2009	594	353
6	2016	102093687	61758523
7	2017	184464988	113496932
8	2018	86272771	53925040
9	2019	37	29
…	2020	6	6

Querying partitioned data – dynamic file pruning

Overview

Uses OPENROWSET function to access data partitioned in sub-folders

Benefits

Use filepath() function to access actual values from file paths.

Eliminate sub-folders/partitions before the query starts execution

Query Spark/Hive partitioned data sets

```
SELECT
    r.filepath(1) AS [year]
   ,r.filepath(2) AS [month]
   ,COUNT_BIG(*) AS [rows]
FROM OPENROWSET(
      BULK 'https://XYZ.blob.core.windows.net/year=*/month=*/*.parquet',
      FORMAT = 'PARQUET') AS [r]
WHERE r.filepath(1) IN ('2017')
   AND r.filepath(2) IN ('10', '11', '12')

GROUP BY    r.filepath() ,r.filepath(1) ,r.filepath(2)
ORDER BY    filepath
```

year	month	rows
2017	10	9768815
2017	11	9284803
2017	12	9508276

Rich surface area

- External tables
- Views
- Stored procedures
- Inline TVF

- T-SQL query language
- Windows aggregate functions
- Cross-database queries
- Federated queries
  - ADLS, CosmosDB
- SQL permission model
  - SQL/Azure AD auth

(Azure Synapse Analytics SQL serverless pool)

External Tables

External Tables

- Used to read data from files or write data to files in Azure Storage
  - Query data in Azure Blob Storage or Azure Data Lake Storage with T-SQL
  - Store query results to files in Azure Blob Storage or Azure Data Lake Storage using CETAS

- Steps:
  - CREATE EXTERNAL DATA SOURCE
  - CREATE EXTERNAL FILE FORMAT
  - CREATE EXTERNAL TABLE

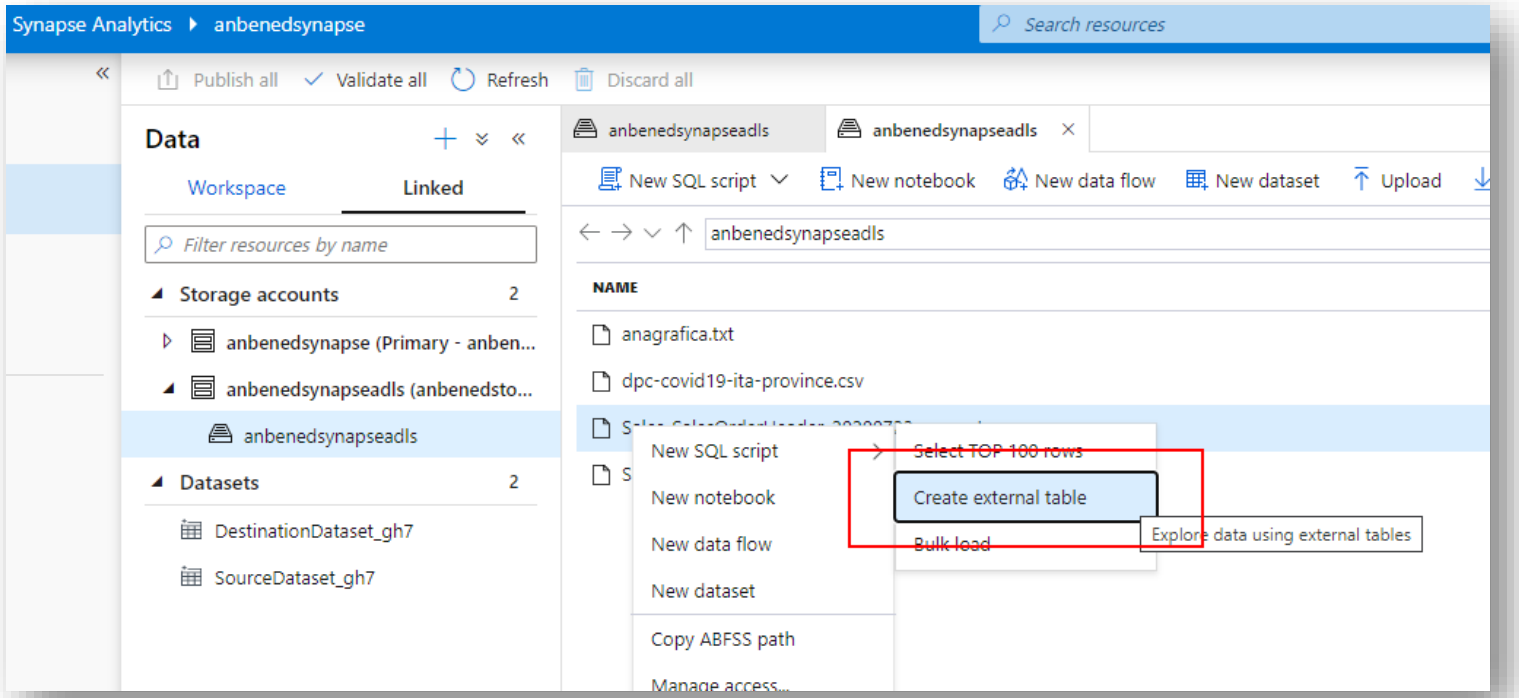
```
CREATE EXTERNAL TABLE abc
WITH
(
   LOCATION = 'myFolder',
   DATA_SOURCE = myDS,
   FILE_FORMAT = myFF
)
AS
          SELECT ...
GO
```

Per definition of an external table

The table object does NOT own the underlying data

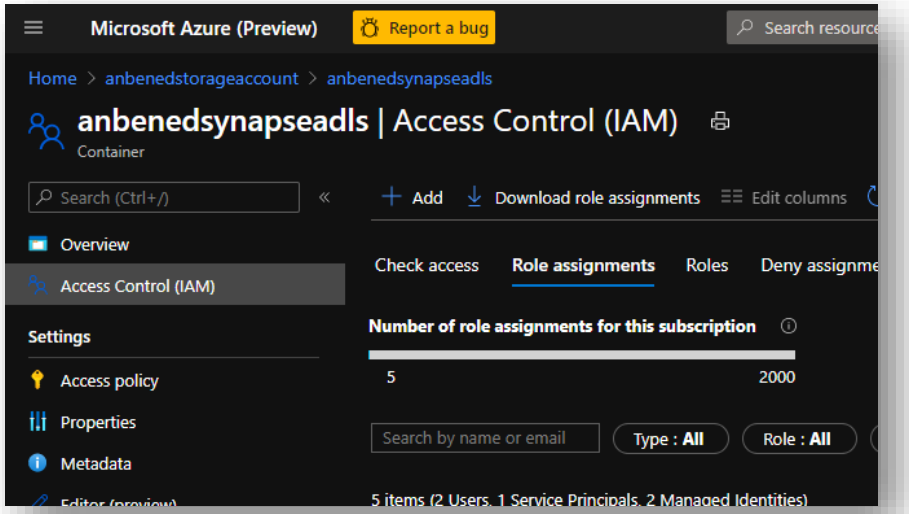
External Tables

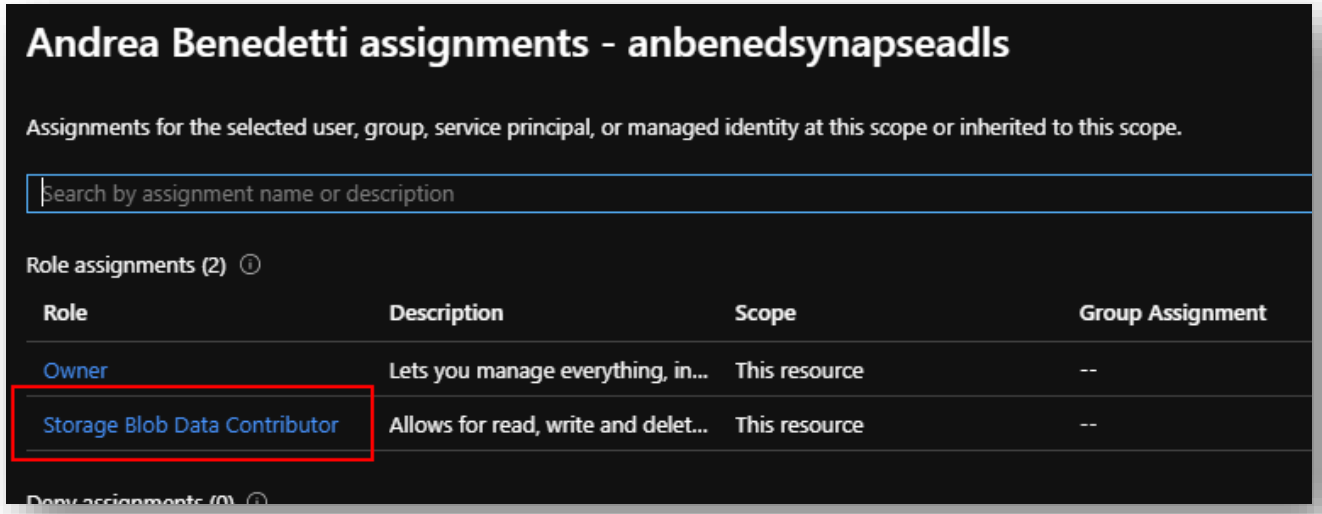
- You can create them without writing any lines of codes
- We'll create a link definition between your Data Lake and the db
  - This takes advantage of the Synapse Analytics engine (MPP system)



Security Tip

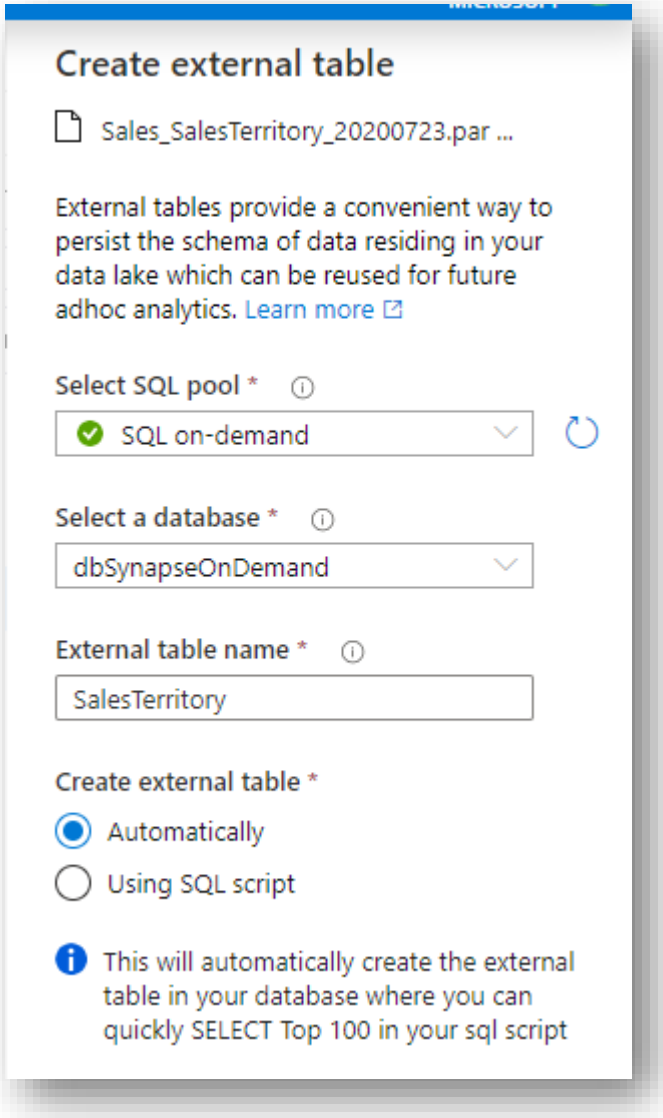
- Grant user / group 'Storage Blob Data Contributor' role on the storage account you're trying to query





External Tables

- Azure Synapse Analytics can create the external table for us
- We just need to provide the name of the database that we want to use, external table name, and the automatic option



CETAS

CREATE EXTERNAL TABLE AS SELECT

CETAS

Overview

Create external tables as select (CETAS) enables you to easily transform data and store the results of query on Azure storage

Benefits

Select any data set and store it in parquet format.

Pre-calculate and store results of query and store them permanently on Azure storage.

Use saved data using external table.

Improve performance of your reports by permanently storing the result based on current snapshot of data as parquet files.

```
-- copy CSV dataset into parquet data set
CREATE EXTERNAL TABLE parquet.Population
WITH(
      LOCATION = '/parquet/population',
      DATA_SOURCE = MyAzureStorage,
      FILE_FORMAT = MyAzureParquetFormat  )
AS
SELECT *
FROM csv.Population

-- pre-create report using new parquet data-set
CREATE EXTERNAL TABLE parquet.PopulationByMonth2017
WITH(
      LOCATION = '/parquet/population/bymonth/2017',
      DATA_SOURCE = MyAzureStorage,
      FILE_FORMAT = MyAzureParquetFormat  )
AS
SELECT month = p.month, population = COUNT ( p.population )
FROM parquet.Population p
WHERE p.year = 2017
GROUP BY p.month

-- Reporting tools can now directly read data from pre-created report
SELECT *
FROM parquet.PopulationByMonth2017
```

CETAS - Note

- ORDER BY clause in SELECT is not supported for CETAS
- LOBs can't be used with CETAS

- At this time DROP TABLE don’t delete folder / files
  - 2 separate process: one to drop the table and another one to drop ADLS file

Demo

1. Write data

Statistics for external tables

Statistics

- The more SQL on-demand knows about your data, the faster it can execute queries against it
  - The SQL on-demand query optimizer is a cost-based optimizer
  - DQP (Distributed Query Processor) takes into consideration numerous variables, including number and sizes of files, partitions and statistics, combine all relevant information to explore viable execution plans and ultimately pick the one with the lowest estimated cost

Statistics

- For CSV files, we need to drop and create statistics manually
- For Parquet files, automatic recreation of statistics is turned on and, when I query my data, I can see the statistics created if they didn't exist

Statistics

```
CREATE EXTERNAL FILE FORMAT [QuotedCsvWithHeaderFormat]
WITH
(
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS
        (
            FIELD_TERMINATOR = N',',
            STRING_DELIMITER = N'"',
            FIRST_ROW = 2,
            USE_TYPE_DEFAULT = False
        )
)
GO

CREATE EXTERNAL TABLE dbo.itaProvince
(
    data varchar(50),stato varchar(50),codice_regione varchar(50),
    denominazione_regione varchar(50), codice_provincia varchar(50),
    denominazione_provincia varchar(50),sigla_provincia varchar(50),
    lat varchar(50),long varchar(50),totale_casi int,note varchar(5000)
)
WITH(
        LOCATION = 'anbenedsynapsene/dpc-covid19-ita-province.csv',
        DATA_SOURCE = SqlOnDemandDemoNE,
        FILE_FORMAT = QuotedCsvWithHeaderFormat
)
GO

SELECT * FROM itaProvince
GO
```

```
CREATE STATISTICS stat_codice_provincia
ON dbo.itaProvince(codice_provincia)
WITH FULLSCAN, NORECOMPUTE;
```

Statistics

```sql
--> statistics info
SELECT
    s.name AS statistics_name,
    c.name AS column_name,
    sc.stats_column_id, *
FROM sys.stats AS s
INNER JOIN sys.stats_columns AS sc ON s.object_id = sc.object_id AND s.stats_id = sc.stats_id
INNER JOIN sys.columns AS c ON sc.object_id = c.object_id AND c.column_id = sc.column_id
WHERE
    s.object_id = OBJECT_ID('itaProvince');


--> most recent statistics for the table
SELECT
    name AS stats_name,
    STATS_DATE(object_id, stats_id) AS statistics_update_date
FROM sys.stats
WHERE object_id = OBJECT_ID('itaProvince');



--> last time statistics were updated on each table
SELECT
    sm.[name] AS [schema_name],
    tb.[name] AS [table_name],
    co.[name] AS [stats_column_name],
    st.[name] AS [stats_name],
    STATS_DATE(st.[object_id],st.[stats_id]) AS [stats_last_updated_date]
FROM
    sys.objects ob
    JOIN sys.stats st ON  ob.[object_id] = st.[object_id]
    JOIN sys.stats_columns sc ON  st.[stats_id] = sc.[stats_id]
        AND st.[object_id] = sc.[object_id]
    JOIN sys.columns co ON  sc.[column_id] = co.[column_id]
        AND sc.[object_id] = co.[object_id]
    JOIN sys.types  ty ON  co.[user_type_id] = ty.[user_type_id]
    JOIN sys.tables tb ON  co.[object_id] = tb.[object_id]
    JOIN sys.schemas sm ON  tb.[schema_id] = sm.[schema_id]
WHERE
    st.[user_created] = 1;
```

Results | Messages

	statistics_name	column_name	stats_column_id	object_id	name	stats_id	auto_created	user_created
1	stat_codice_provincia	codice_provincia	1	981578535	stat_codice_provincia	2	0	1

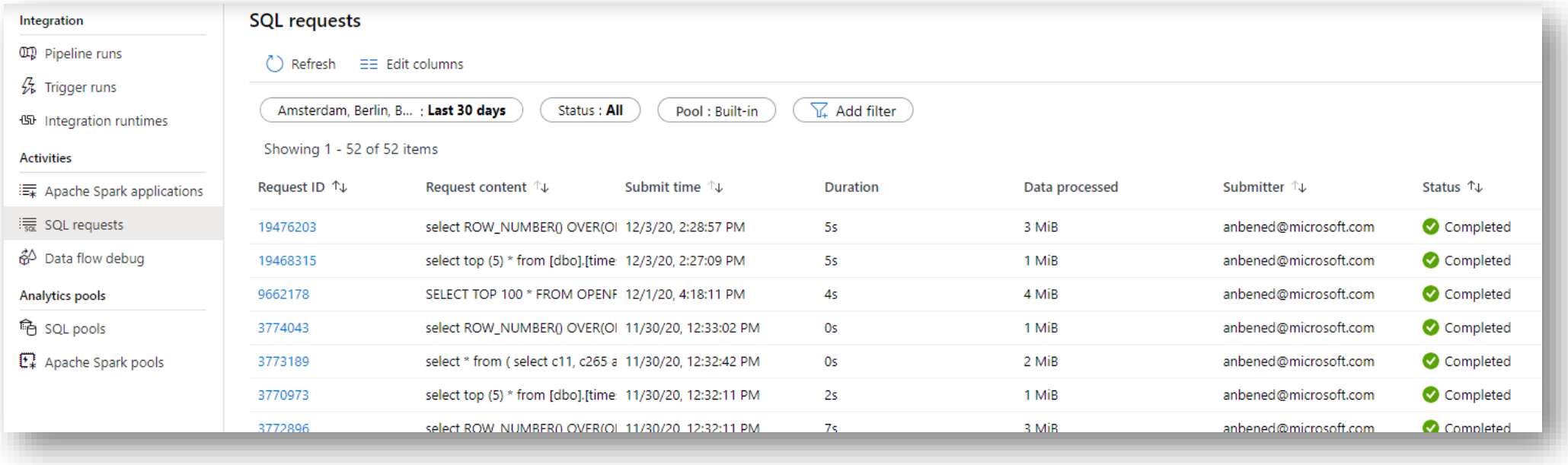
	stats_name	statistics_update_date
1	stat_codice_provincia	2020-10-26 14:06:43.813

	schema_name	table_name	stats_column_name	stats_name	stats_last_updated_date
1	dbo	itaProvince	codice_provincia	stat_codice_provincia	2020-10-26 14:06:43.813

Log

Logs Retention

- Log retention = 60 days




```
select * from sys.dm_exec_requests_history order by start_time desc
```

Best practices for SQL on-demand

Best Practices

- Minimize latency: **colocate** your Azure storage account and your SQL on-demand endpoint
  - Storage accounts and endpoints provisioned during workspace creation are located in the same region

- Optimal performance: if you access other storage accounts with SQL on-demand, make sure they're in the **same region**
  - Different region = increased latency for the data's network transfer between the remote region and the endpoint's region

Best Practices

- Optimal performance → **same region**

Different region → Workspace WE; Storage NE

Execution time: ~31 sec

```
PRINT 'Start time: ' + CAST(SYSDATETIMEOFFSET() as varchar)
SELECT TOP 10 *,
       JSON_VALUE(content,'$._id') as ID,
       JSON_VALUE(content,'$.type') as TypeObj,
       JSON_VALUE(content,'$.title') as Title
FROM json.Books
GO
```

```
Start time: 2020-10-21 18:52:43.0529942 +0
Statement ID: {3A9FC555-8769-46D2-86C0-F301AA7624C5} | Query has
h: 0xA3D53FC7C24A0A9C | Distributed request ID: {D0DF16F3-0B0E-4
670-9677-222065823D5F}. Total size of data scanned is 25 megabyt
es, total size of data moved is 1 megabytes, total size of data
written is 0 megabytes.
(10 rows affected)
Total execution time: 00:00:30.517
```

Same region → Workspace NE; Storage NE

Execution time: ~13 sec

```
PRINT 'Start time: ' + CAST(SYSDATETIMEOFFSET() as varchar)
SELECT TOP 10 *,
       JSON_VALUE(content,'$._id') as ID,
       JSON_VALUE(content,'$.type') as TypeObj,
       JSON_VALUE(content,'$.title') as Title
FROM json.Books
GO
```

```
Start time: 2020-10-22 13:36:30.9970461 +0
Statement ID: {291E82F7-C734-4A36-A3B1-B76E765C4325} | Query has
h: 0xA3D53FC7C24A0A9C | Distributed request ID: {3314CA9A-C27B-4
FCB-B2A3-254AA49B0D4B}. Total size of data scanned is 25 megabyt
es, total size of data moved is 1 megabytes, total size of data
written is 0 megabytes.
(10 rows affected)
Total execution time: 00:00:12.961
```

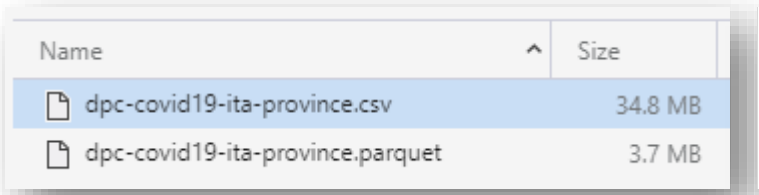
Best Practices

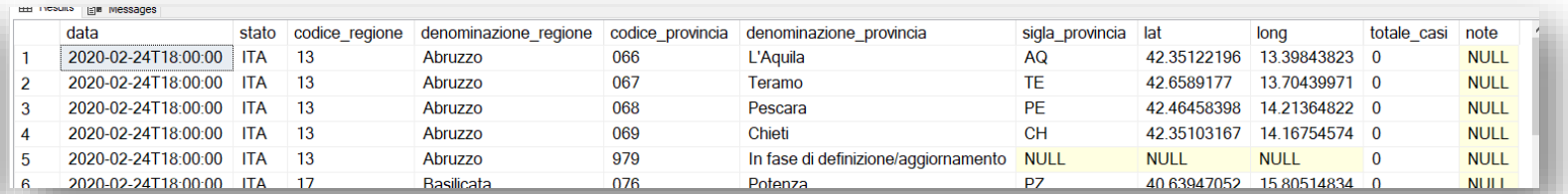
- Multiple applications and services might access your storage account
  - **don't stress the storage** with other workloads during query execution

- If possible, you can **prepare files** for better performance
  - Convert CSV and JSON → Parquet (data scanned reduced)
    - Simple idea: copy data in ADF

  - Single large file → multiple smaller files
  - CSV file size below 10 GB
  - Equally sized files for a single OPENROWSET path / external table LOCATION
  - Partition your data by storing partitions to different folders or file names

Best Practices (csv vs. parquet)





	data	stato	codice_regione	denominazione_regione	codice_provincia	denominazione_provincia	sigla_provincia	lat	long	totale_casi	note
1	2020-02-24T18:00:00	ITA	13	Abruzzo	066	L'Aquila	AQ	42.35122196	13.39843823	0	NULL
2	2020-02-24T18:00:00	ITA	13	Abruzzo	067	Teramo	TE	42.6589177	13.70439971	0	NULL
3	2020-02-24T18:00:00	ITA	13	Abruzzo	068	Pescara	PE	42.46458398	14.21364822	0	NULL
4	2020-02-24T18:00:00	ITA	13	Abruzzo	069	Chieti	CH	42.35103167	14.16754574	0	NULL
5	2020-02-24T18:00:00	ITA	13	Abruzzo	979	In fase di definizione/aggiornamento	NULL	NULL	NULL	0	NULL
6	2020-02-24T18:00:00	ITA	17	Basilicata	076	Potenza	PZ	40.63947052	15.80514834	0	NULL

Statement test	Format	Duration	Rows	Data Scanned	Data Moved
SELECT *	CSV	~20 sec	441259	35 MB	38 MB
	Parquet	~18 sec	441259	4 MB	38 MB
COUNT(*)	CSV	~0 sec	1	35 MB	1 MB
	Parquet	~0 sec	1	1 MB	1 MB
SELECT sum(cast(totale_casi as int))	CSV	~1 sec	1	35 MB	1 MB
	Parquet	~0 sec	1	2 MB	1 MB
SELECT data, codice_provincia, totale_casi	CSV	~8 sec	441259	35 MB	14 MB
	Parquet	~6 sec	441259	2 MB	14 MB

Pricing

Pricing

*Does **Azure Synapse SQL On Demand** pricing have provision for Azure reserved capacity (e.g. 1 year or 3 year reserved) like how it is present in Azure SQL Pool or it is flat as per the rate described above?*

At the moment, there is no reserved capacity payment model for the SQL On-demand.

*Suppose if the data queried through **Azure Synapse SQL On Demand** is 1 MB , so price will be calculated according to pro-rate usage or any rounding logic involved?e.g. with a rate of €4.217 per TB, 1 MB data query cost will be € 0.000004217?*

The floor for charging is actually 10 MB, so the minimum charge is for 10 MB, but otherwise the math you're showing above is good.

For **Azure Synapse SQL On Demand**, the pricing is €4.217 per TB of data processed in West EU region.

Pricing example

- How to calculate price for Azure Synapse SQL (Serverless) as below?

Query Scenario: 30 execution times /month; 0.5 TB per execution times

Calculation: 30 * 0.5 * 4.217 = 63.255 € (Monthly cost)

Pricing Tips

- Charging is per data processed

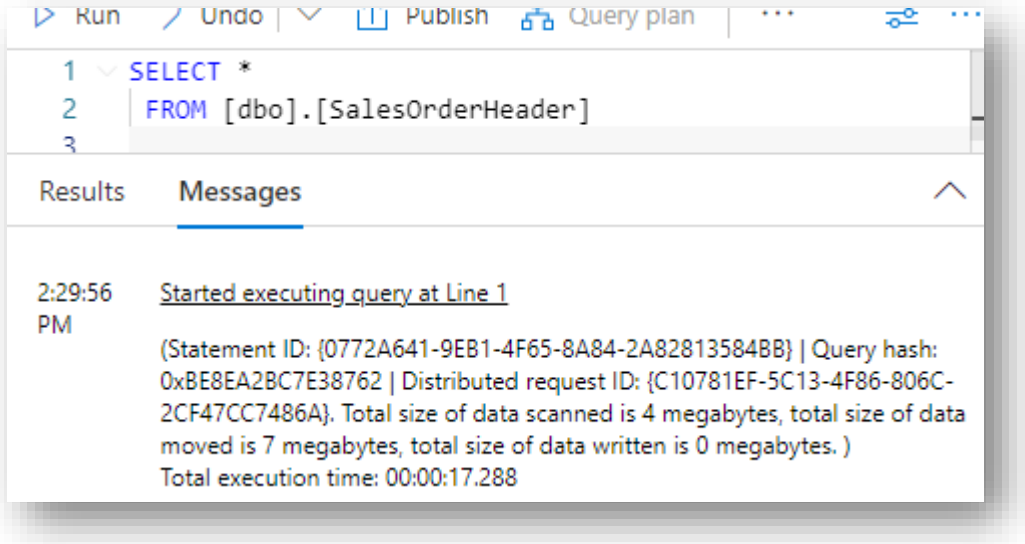
- If data is in Parquet format which is compressed, it will be cheaper
- Also, Parquet format is columnar so you will be charged only for columns you need in your query, not all columns, making it even cheaper

- In case of CSV, data is not only uncompressed, but SQL on-demand would need to read whole rows for you to extract columns you target

Billing: “total processed” volume

Each query:

- Total size of data scanned is XX megabytes
- Total size of data moved is XX megabytes
- Total size of data written is XX megabytes

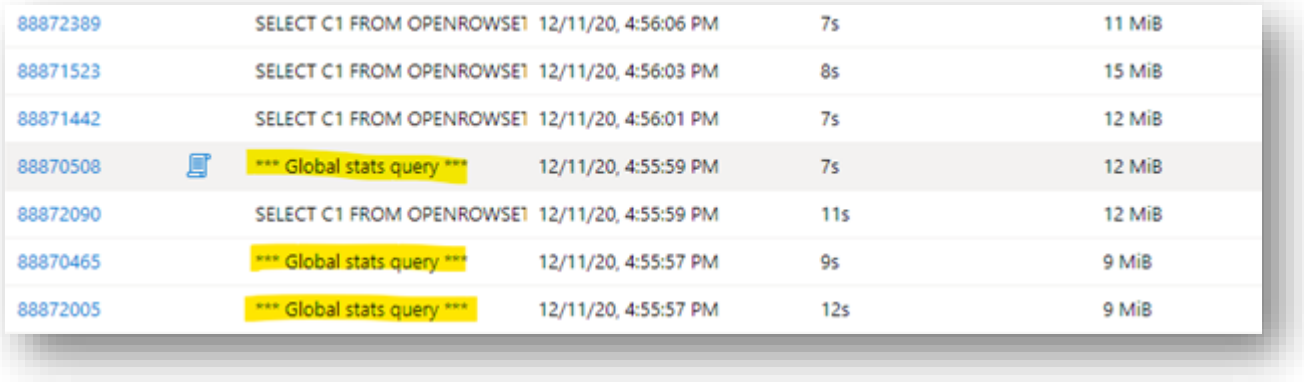


Total data processed = data scanned + data moved + data written

- Data processed = data stored internally while executing query
  - Data read (compressed data + metadata reads) + intermediate results (data shuffled, uncompressed format always)
  - + data transferred to node you connected to before returning results to client
    - (this transfer is accounted for as data processed also)
  - + in general case: autostats and read-ahead.

Billing & global stats query

- Global stats query = queries that system automatically executes to figure out what are the statistics in the data
  - You are charged for that query as well
  - Without statistics execution plan would be suboptimal and would lead to more data processed by the user query itself and worst performance



88872389	SELECT C1 FROM OPENROWSET	12/11/20, 4:56:06 PM	7s	11 MiB
88871523	SELECT C1 FROM OPENROWSET	12/11/20, 4:56:03 PM	8s	15 MiB
88871442	SELECT C1 FROM OPENROWSET	12/11/20, 4:56:01 PM	7s	12 MiB
88870508	*** Global stats query ***	12/11/20, 4:55:59 PM	7s	12 MiB
88872090	SELECT C1 FROM OPENROWSET	12/11/20, 4:55:59 PM	11s	12 MiB
88870465	*** Global stats query ***	12/11/20, 4:55:57 PM	9s	9 MiB
88872005	*** Global stats query ***	12/11/20, 4:55:57 PM	12s	9 MiB

Demo

1. Monitoring & costs

Cost control

Cost control

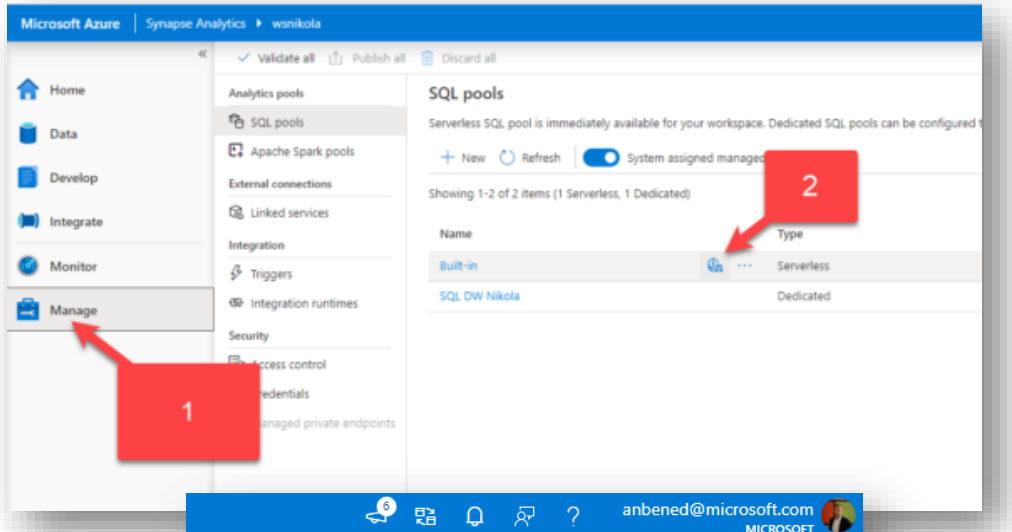
```
exec sp_set_data_processed_limit
    @type = N'daily', @limit_tb = 5
exec sp_set_data_processed_limit
    @type= N'weekly', @limit_tb = 10
exec sp_set_data_processed_limit
    @type= N'monthly', @limit_tb = 50
```

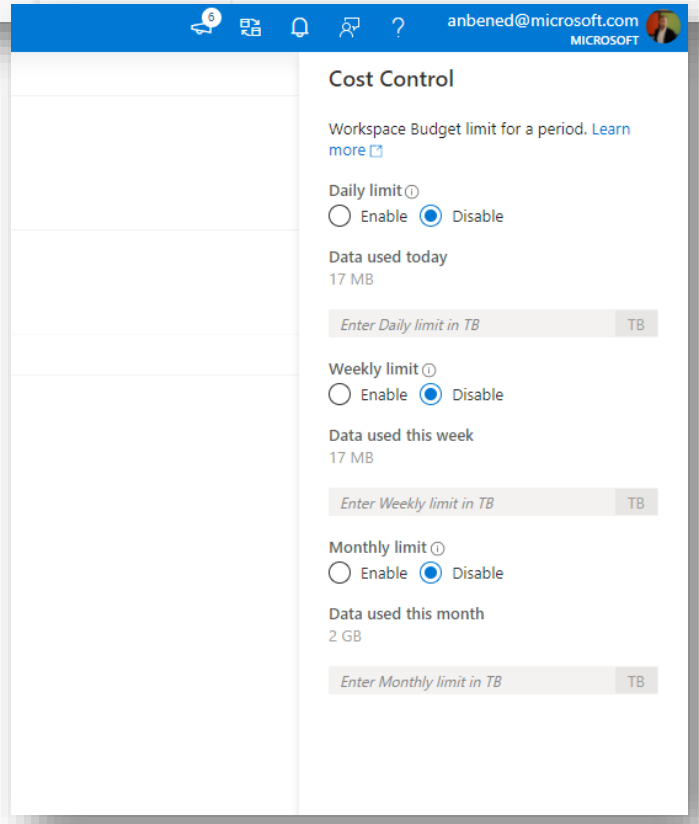
```
SELECT * FROM sys.dm_external_data_processed
```

0 %

Results | Messages

	type	data_processed_mb
1	daily	17
2	weekly	17
3	monthly	3065





Notes

- There is **no cache** yet in SQL on-demand so the queries won’t run faster after the first run

- There is **no way for a user to pre-allocate more resources** to SQL on-demand at this moment

- Everything related to reading files from storage might have an impact on query performance
  - SQL on-demand allows you to query files in your Azure storage accounts. It doesn't have local storage or ingestion capabilities. So, all files that the query targets are external to SQL on-demand.

Synapse Analytics Notes

- Separation of state (data, metadata and transactional logs) and compute
- Queries against data loaded into SQL Analytics tables are 2-3X faster compared to queries over external tables
- Warm-up for first on-demand SQL query takes about 30-40 seconds
- Provisioned SQL may give you better and more predictable performance due to resource reservation
- Each SQL pool can currently only access tables created within its pool (there is one database per pool), while on-demand SQL can not yet query a database
- You can only run OPENROWSET statement from SQL on-demand

Conclusion

Conclusion

Synapse is integrated environment for Azure data analytics

Serverless Synapse SQL scenarios

- Logical data warehouse
- Azure storage data analysis with rich T-SQL language

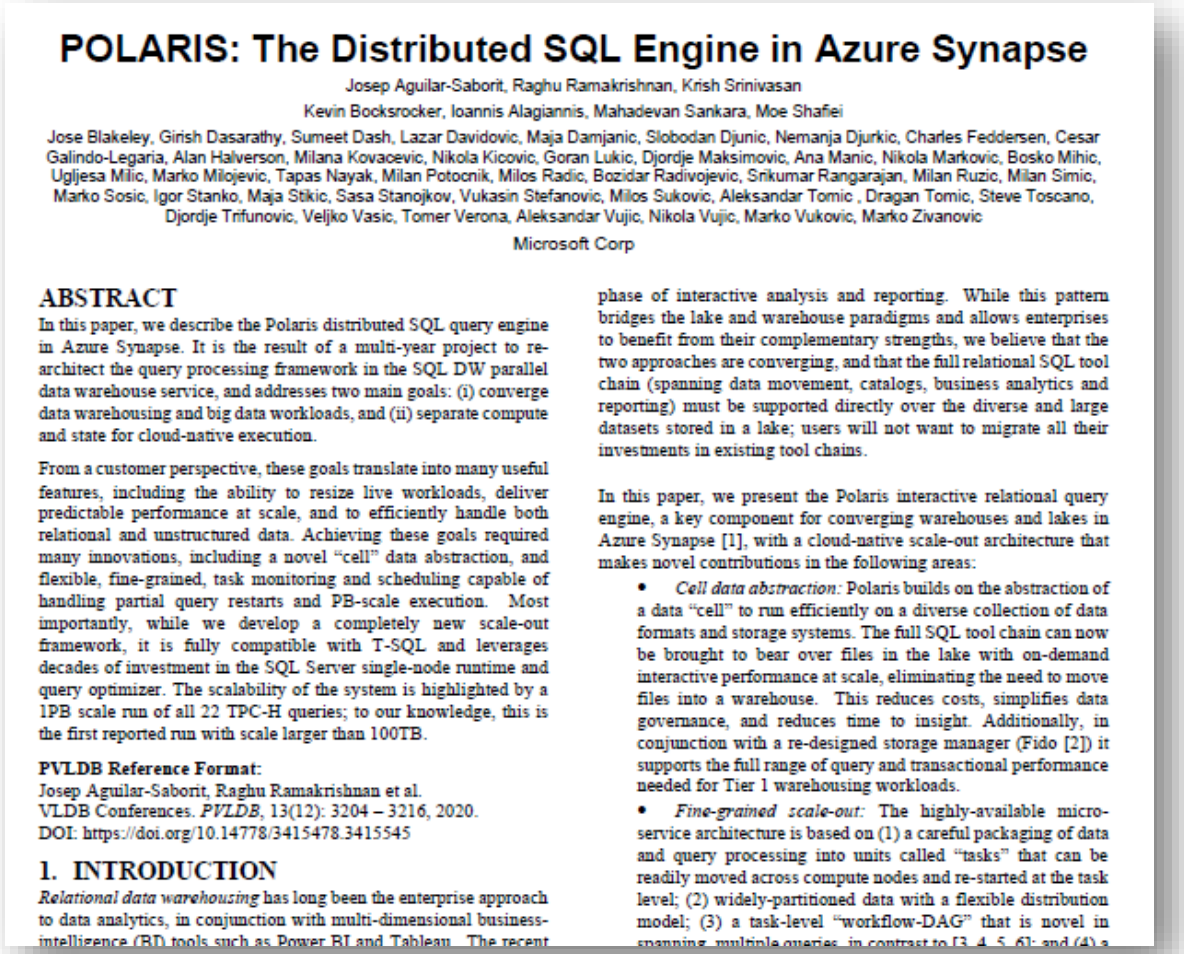
Serverless Synapse SQL workload patterns

- Ad-hoc queries
- Unpredictable workloads

Behind the scenes

A high-level understanding how queries are pushed-down when using SQL on-demand

Behind the scenes



POLARIS: The Distributed SQL Engine in Azure Synapse

Josep Aguilar-Saborit, Raghu Ramakrishnan, Krish Srinivasan
Kevin Bocksrocker, Ioannis Alagiannis, Mahadevan Sankara, Moe Shafiei
Jose Blakeley, Girish Dasarathy, Sumeet Dash, Lazar Davidovic, Maja Damjanic, Slobodan Djunic, Nemanja Djurkic, Charles Feddersen, Cesar Galindo-Legaria, Alan Halverson, Milana Kovacevic, Nikola Kicovic, Goran Lukic, Djordje Maksimovic, Ana Manic, Nikola Markovic, Bosko Mihic, Ugljesa Milic, Marko Milojevic, Tapas Nayak, Milan Potocnik, Milos Radic, Bozidar Radivojevic, Srikumar Rangarajan, Milan Ruzic, Milan Simic, Marko Sosic, Igor Stanko, Maja Stikic, Sasa Stanojkov, Vukasin Stefanovic, Milos Sukovic, Aleksandar Tomic , Dragan Tomic, Steve Toscano, Djordje Trifunovic, Veljko Vasic, Tomer Verona, Aleksandar Vujic, Nikola Vujic, Marko Vukovic, Marko Zivanovic
Microsoft Corp

ABSTRACT

In this paper, we describe the Polaris distributed SQL query engine in Azure Synapse. It is the result of a multi-year project to re-architect the query processing framework in the SQL DW parallel data warehouse service, and addresses two main goals: (i) converge data warehousing and big data workloads, and (ii) separate compute and state for cloud-native execution.

From a customer perspective, these goals translate into many useful features, including the ability to resize live workloads, deliver predictable performance at scale, and to efficiently handle both relational and unstructured data. Achieving these goals required many innovations, including a novel "cell" data abstraction, and flexible, fine-grained, task monitoring and scheduling capable of handling partial query restarts and PB-scale execution. Most importantly, while we develop a completely new scale-out framework, it is fully compatible with T-SQL and leverages decades of investment in the SQL Server single-node runtime and query optimizer. The scalability of the system is highlighted by a 1PB scale run of all 22 TPC-H queries; to our knowledge, this is the first reported run with scale larger than 100TB.

PVLDB Reference Format:
Josep Aguilar-Saborit, Raghu Ramakrishnan et al.
VLDB Conferences. *PVLDB*, 13(12): 3204 – 3216, 2020.
DOI: https://doi.org/10.14778/3415478.3415545

1. INTRODUCTION

Relational data warehousing has long been the enterprise approach to data analytics, in conjunction with multi-dimensional business-intelligence (BI) tools such as Power BI and Tableau. The recent phase of interactive analysis and reporting. While this pattern bridges the lake and warehouse paradigms and allows enterprises to benefit from their complementary strengths, we believe that the two approaches are converging, and that the full relational SQL tool chain (spanning data movement, catalogs, business analytics and reporting) must be supported directly over the diverse and large datasets stored in a lake; users will not want to migrate all their investments in existing tool chains.

In this paper, we present the Polaris interactive relational query engine, a key component for converging warehouses and lakes in Azure Synapse [1], with a cloud-native scale-out architecture that makes novel contributions in the following areas:

- *Cell data abstraction:* Polaris builds on the abstraction of a data "cell" to run efficiently on a diverse collection of data formats and storage systems. The full SQL tool chain can now be brought to bear over files in the lake with on-demand interactive performance at scale, eliminating the need to move files into a warehouse. This reduces costs, simplifies data governance, and reduces time to insight. Additionally, in conjunction with a re-designed storage manager (Fido [2]) it supports the full range of query and transactional performance needed for Tier 1 warehousing workloads.
- *Fine-grained scale-out:* The highly-available micro-service architecture is based on (1) a careful packaging of data and query processing into units called "tasks" that can be readily moved across compute nodes and re-started at the task level; (2) widely-partitioned data with a flexible distribution model; (3) a task-level "workflow-DAG" that is novel in spanning multiple queries, in contrast to [3, 4, 5, 6]; and (4) a

http://www.vldb.org/pvldb/vol13/p3204-saborit.pdf

OPENROWSET in SQL On-demand is a distributed processing (NO with Spark)

Technology used is called **Polaris**

Behind the scenes

- A pool of SQL instances → quickly warmed up → driven by the new distributed query processor
  - The components have been designed to be able to query any data (from the lake as well as managed data in future), there is built-in fault tolerance, etc
  - The Polaris engine will be used in Synapse SQL gen 3 for both dedicated and serverless pools

- Data is read directly from storage, in parallel, and filtering is done at the SQL engine nodes themselves

  - We're considering adding active data sources as well, in terms of being able to push-down compute to the originating system, for situations where exhausting data into a lake is not easy/possible. But this is sometime out, as we have higher priority items to attend to first.

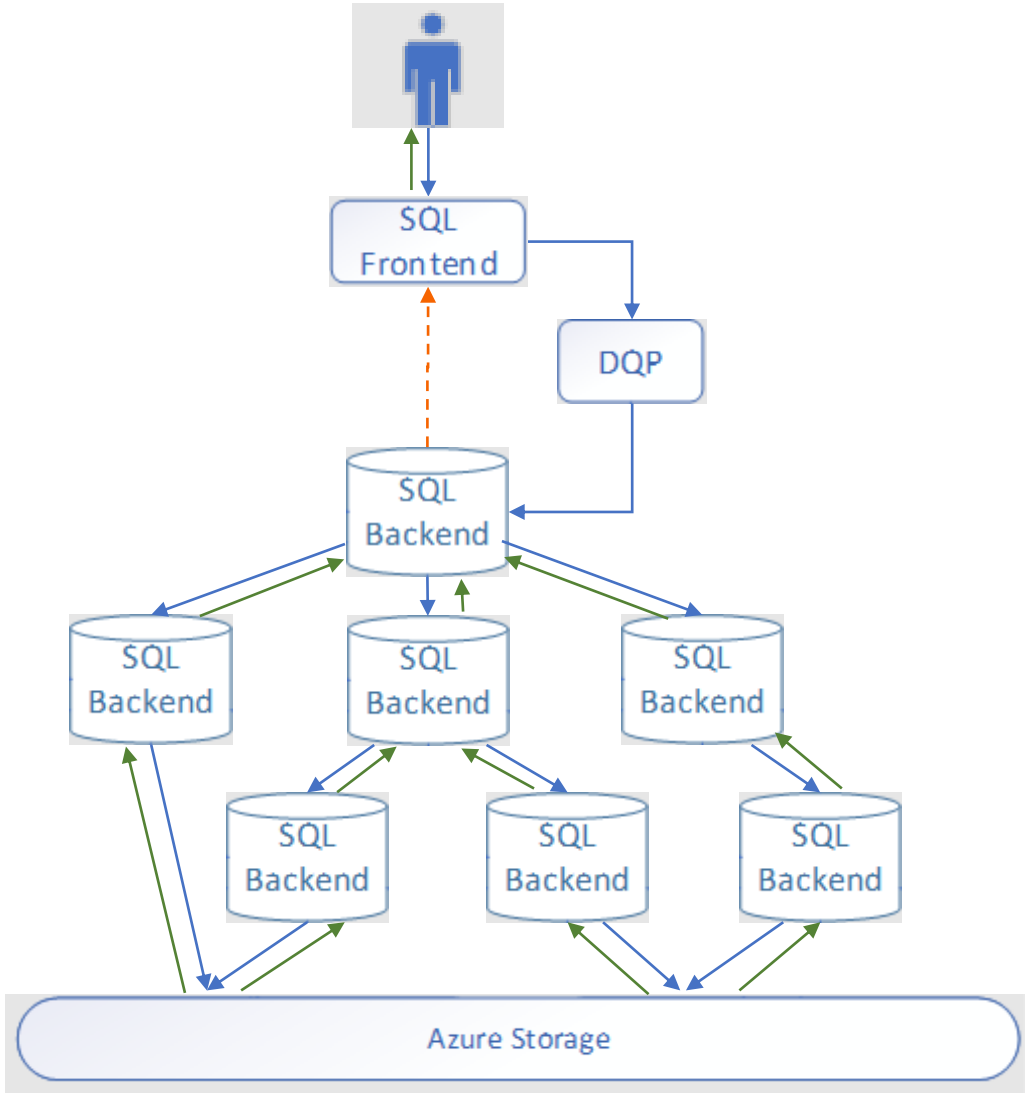
  - We also filter the data we read – for example, we eliminate some files early on based on partition elimination, and in case of Parquet we also skip reading some column segments where possible.

Behind the scenes

- If you something like a “SELECT TOP 10 * …” without order by or where, we don’t need to read the whole file
  - We will read top rows from each file

Distributed query execution flow

- SQL Frontend
  - Metadata
  - Security
  - Query simplification (filter pushdown, partition elimination, …)
- DQP – Distributed Query Processor
  - Explores viable distributed execution plans and picks one with lowest estimated cost
  - Breaks user query into T-SQL fragments (tasks)
- SQL Backend – fully stateless
  - Executes tasks
  - Propagate results to parent



Provisioned vs Serverless