DEMO

* Ecco lo scenario: ho due file CSV relativi al set di dati sui taxi di New York, che ho già utilizzato in una delle precedenti demo. Uno contiene i dati su tutti i viaggi in taxi giallo da gennaio 2020 (580 MB), mentre l'altro contiene i dati da febbraio 2020 (580 MB). Scaricati da qui:

<https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page>

Credenziali:

**Server**: datasatdemo-ondemand.sql.azuresynapse.net

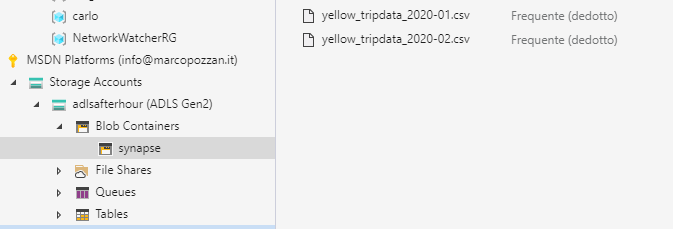
**User**: sqladminuser

**Pwd**: Pa$$w0rdissima

* Creiamo il database

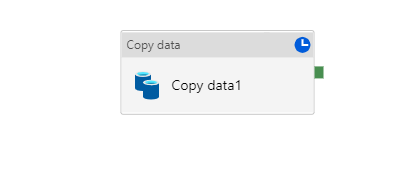
CREATE DATABASE DataSatDB

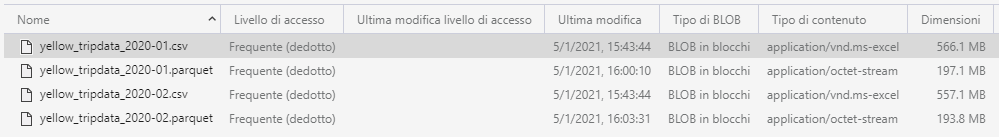
* Carichiamo i file nel data lake di synapse **adlsaftherhour**



* Creo una pipeline per la generazione di file parquet dai file csv

Mostrare la creazione di una pipeline di copy data . La pipeline **testconvertCSV2Parquet** già funziona





* Prima di creare le viste verifico il tipo dei dati con la store procedure in modo da evitare il infered schema e impostare noi il giusto tipo di dati

--Verifica del tipo di dati

EXEC sp\_describe\_first\_result\_set N'SELECT \* FROM

    OPENROWSET(

        BULK N''https://adlsafterhour.dfs.core.windows.net/synapse/DataSat/yellow\_tripdata\_2020-01.parquet'',

        FORMAT = ''PARQUET''

        ) as test'

* Creo 4 viste su SQL pool Servless una per mese e due per csv e due per parquet. Nelle viste farò attenzione a fare le seguenti cose
* Sto usando il blocco WITH per definire in modo esplicito i tipi di dati, se non lo facessi, tutte le colonne dei tuoi caratteri verranno automaticamente impostate su VARCHAR (8000) e, di conseguenza, una memoria più costosa.
* Come puoi notare, ho rinominato i nomi delle mie colonne generiche dai file CSV, quindi ora sembrano più leggibili.
* Ho anche trasformato le colonne DateTime solo al tipo di data, poiché non ho bisogno della parte del tempo per questa demo.
* In questo modo, abbiamo ridotto di conseguenza la cardinalità e l'intera dimensione del modello di dati (BEST PRACTICES DI POWER BI). Controlliamo quanti record contiene ciascuno di questi file?

DROP VIEW IF EXISTS trip202001\_csv;

GO

CREATE VIEW trip202001\_csv AS

SELECT

VendorID

,cast(tpep\_pickup\_datetime as DATE) tpep\_pickup\_datetime

,cast(tpep\_dropoff\_datetime as DATE) tpep\_dropoff\_datetime

,passenger\_count

,trip\_distance

,RateCodeID

,store\_and\_fwd\_flag

,PULocationID

,DOLocationID

,payment\_type

,fare\_amount

,extra

,mta\_tax

,tip\_amount

,tolls\_amount

,improvement\_surcharge

,total\_amount

,congestion\_surcharge

FROM

    OPENROWSET(

        BULK N'https://adlsafterhour.dfs.core.windows.net/synapse/DataSat/yellow\_tripdata\_2020-01.csv',

        FORMAT = 'CSV',

        PARSER\_VERSION='2.0',

HEADER\_ROW = TRUE

)

WITH(

VendorID INT,

tpep\_pickup\_datetime DATETIME2,

tpep\_dropoff\_datetime DATETIME2,

passenger\_count INT,

trip\_distance DECIMAL(10,2),

RateCodeID INT,

store\_and\_fwd\_flag VARCHAR(10),

PULocationID INT,

DOLocationID INT,

payment\_type INT,

fare\_amount DECIMAL(10,2),

extra DECIMAL(10,2),

mta\_tax DECIMAL(10,2),

tip\_amount DECIMAL(10,2),

tolls\_amount DECIMAL(10,2),

improvement\_surcharge DECIMAL(10,2),

total\_amount DECIMAL(10,2),

congestion\_surcharge DECIMAL(10,2)

)

AS [trip202001\_csv]

WHERE tpep\_pickup\_datetime < '20200207'

DROP VIEW IF EXISTS trip202002\_csv;

GO

CREATE VIEW trip202002\_csv AS

SELECT

VendorID

,cast(tpep\_pickup\_datetime as DATE) tpep\_pickup\_datetime

,cast(tpep\_dropoff\_datetime as DATE) tpep\_dropoff\_datetime

,passenger\_count

,trip\_distance

,RateCodeID

,store\_and\_fwd\_flag

,PULocationID

,DOLocationID

,payment\_type

,fare\_amount

,extra

,mta\_tax

,tip\_amount

,tolls\_amount

,improvement\_surcharge

,total\_amount

,congestion\_surcharge

FROM

    OPENROWSET(

        BULK N'https://adlsafterhour.dfs.core.windows.net/synapse/DataSat/yellow\_tripdata\_2020-02.csv',

        FORMAT = 'CSV',

        PARSER\_VERSION='2.0',

HEADER\_ROW = TRUE

)

WITH(

VendorID INT,

tpep\_pickup\_datetime DATETIME2,

tpep\_dropoff\_datetime DATETIME2,

passenger\_count INT,

trip\_distance DECIMAL(10,2),

RateCodeID INT,

store\_and\_fwd\_flag VARCHAR(10),

PULocationID INT,

DOLocationID INT,

payment\_type INT,

fare\_amount DECIMAL(10,2),

extra DECIMAL(10,2),

mta\_tax DECIMAL(10,2),

tip\_amount DECIMAL(10,2),

tolls\_amount DECIMAL(10,2),

improvement\_surcharge DECIMAL(10,2),

total\_amount DECIMAL(10,2),

congestion\_surcharge DECIMAL(10,2)

)

AS [trip202002\_csv]

WHERE tpep\_pickup\_datetime < '20200207'

DROP VIEW IF EXISTS trip202001\_parquet;

GO

CREATE VIEW trip202001\_parquet AS

SELECT

VendorID

,cast(tpep\_pickup\_datetime as DATE) tpep\_pickup\_datetime

,cast(tpep\_dropoff\_datetime as DATE) tpep\_dropoff\_datetime

,passenger\_count

,trip\_distance

,RateCodeID

,store\_and\_fwd\_flag

,PULocationID

,DOLocationID

,payment\_type

,fare\_amount

,extra

,mta\_tax

,tip\_amount

,tolls\_amount

,improvement\_surcharge

,total\_amount

,congestion\_surcharge

FROM

    OPENROWSET(

        BULK N'https://adlsafterhour.dfs.core.windows.net/synapse/DataSat/yellow\_tripdata\_2020-01.parquet',

        FORMAT = 'PARQUET'

) WITH(

VendorID INT,

tpep\_pickup\_datetime DATETIME2,

tpep\_dropoff\_datetime DATETIME2,

passenger\_count INT,

trip\_distance DECIMAL(10,2),

RateCodeID INT,

store\_and\_fwd\_flag VARCHAR(10),

PULocationID INT,

DOLocationID INT,

payment\_type INT,

fare\_amount DECIMAL(10,2),

extra DECIMAL(10,2),

mta\_tax DECIMAL(10,2),

tip\_amount DECIMAL(10,2),

tolls\_amount DECIMAL(10,2),

improvement\_surcharge DECIMAL(10,2),

total\_amount DECIMAL(10,2),

congestion\_surcharge DECIMAL(10,2)

)

 as [trip202001\_parquet ]

WHERE tpep\_pickup\_datetime < '20200107'

DROP VIEW IF EXISTS trip202002\_parquet;

GO

CREATE VIEW trip202002\_parquet AS

SELECT

VendorID

,cast(tpep\_pickup\_datetime as DATE) tpep\_pickup\_datetime

,cast(tpep\_dropoff\_datetime as DATE) tpep\_dropoff\_datetime

,passenger\_count

,trip\_distance

,RateCodeID

,store\_and\_fwd\_flag

,PULocationID

,DOLocationID

,payment\_type

,fare\_amount

,extra

,mta\_tax

,tip\_amount

,tolls\_amount

,improvement\_surcharge

,total\_amount

,congestion\_surcharge

FROM

    OPENROWSET(

        BULK N'https://adlsafterhour.dfs.core.windows.net/synapse/DataSat/yellow\_tripdata\_2020-02.parquet',

        FORMAT = 'PARQUET'

) WITH(

VendorID INT,

tpep\_pickup\_datetime DATETIME2,

tpep\_dropoff\_datetime DATETIME2,

passenger\_count INT,

trip\_distance DECIMAL(10,2),

RateCodeID INT,

store\_and\_fwd\_flag VARCHAR(10),

PULocationID INT,

DOLocationID INT,

payment\_type INT,

fare\_amount DECIMAL(10,2),

extra DECIMAL(10,2),

mta\_tax DECIMAL(10,2),

tip\_amount DECIMAL(10,2),

tolls\_amount DECIMAL(10,2),

improvement\_surcharge DECIMAL(10,2),

total\_amount DECIMAL(10,2),

congestion\_surcharge DECIMAL(10,2)

)

 as [trip202002\_parquet]

WHERE tpep\_pickup\_datetime < '20200207'

* Creo 2 viste su SQL pool Servless una che mette assieme i due file csv e una i parquet

--creo una vista che unisce i due mesi

DROP VIEW IF EXISTS trip2020\_csv;

GO

CREATE VIEW trip2020\_csv AS

select \*

from dbo.trip202001\_csv

WHERE (tpep\_pickup\_datetime  >= '20200101' AND tpep\_pickup\_datetime   < '20200107')

union all

select \*

from dbo.trip202002\_csv

WHERE (tpep\_pickup\_datetime  >= '20200201' AND tpep\_pickup\_datetime   < '20200207')

--creo una vista che unisce i due mesi

DROP VIEW IF EXISTS trip2020\_parquet;

GO

CREATE VIEW trip2020\_parquet AS

select \*

from dbo.trip202001\_parquet

WHERE (tpep\_pickup\_datetime  >= '20200101' AND tpep\_pickup\_datetime   < '20200107')

union all

select \*

from dbo.trip202002\_parquet

WHERE (tpep\_pickup\_datetime  >= '20200201' AND tpep\_pickup\_datetime   < '20200207')

* Verifichiamo che il numero di righe sia uguale

--Verifico cardinalità

Select count(\*)

from trip2020\_csv

--Verifico cardinalità

Select count(\*)

from trip2020\_parquet

* Creiamo ora le tabelle esterne

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Pa$$w0rdissima'

CREATE DATABASE SCOPED CREDENTIAL SqlOnDemandTest

WITH IDENTITY='SHARED ACCESS SIGNATURE',

SECRET = '?sv=2019-12-12&ss=bfqt&srt=sco&sp=rwdlacupx&se=2021-01-30T05:45:52Z&st=2021-01-03T21:45:52Z&spr=https&sig=WJEdg94PkWgM7a4Ka5sJlCN1ARKheVFewn%2F5Wbh%2BwGY%3D'

GO

CREATE EXTERNAL DATA SOURCE DS\_Taxi WITH (

    LOCATION = 'https://adlsafterhour.blob.core.windows.net',

    CREDENTIAL = SqlOnDemandTest

);

--Creaione del formato di importazione

DROP EXTERNAL FILE FORMAT exCsv

GO

CREATE EXTERNAL FILE FORMAT exCsv

WITH (

    FORMAT\_TYPE = DELIMITEDTEXT,

    FORMAT\_OPTIONS (FIELD\_TERMINATOR =',',Encoding = 'UTF8',First\_Row= 2)

);

DROP EXTERNAL FILE FORMAT exParquet

GO

CREATE EXTERNAL FILE FORMAT exParquet

WITH (

    FORMAT\_TYPE = PARQUET,

    DATA\_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'

);

--Crea una copia locale nello storage di Synapse synapse/DataSat/ext

DROP EXTERNAL TABLE trip\_2020\_csv\_ext

GO

CREATE EXTERNAL TABLE trip\_2020\_csv\_ext

WITH (

    LOCATION = 'synapse/DataSat/trip\_2020\_csv\_ext',

    DATA\_SOURCE = DS\_Taxi,

    FILE\_FORMAT = exCsv

)

AS

SELECT \*

from dbo.trip2020\_parquet

go

DROP EXTERNAL TABLE taxi\_parquet

GO

CREATE EXTERNAL TABLE trip\_2020\_parquet\_ext

WITH (

    LOCATION = 'synapse/DataSat/trip\_2020\_parquet\_ext',

    DATA\_SOURCE = DS\_Taxi,

    FILE\_FORMAT = exParquet

)

AS

SELECT \*

from dbo.trip2020\_parquet

* Test 1: **(Collegamento in modalità importata su CSV)**

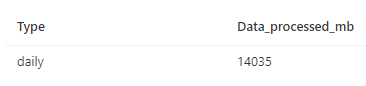
Credenziali:

**Server**: datasatdemo-ondemand.sql.azuresynapse.net

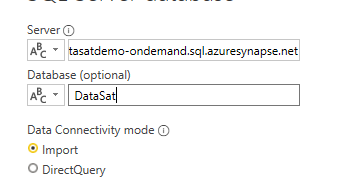
**User**: sqladminuser

**Pwd**: Pa$$w0rdissima

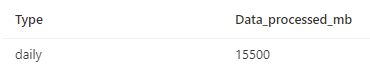
* ----------------------------------------Passo 1
  + Prima devo segnarmi i mega a cui sono arrivato
    - SELECT \* FROM sys.dm\_external\_data\_processed
    - WHERE type = 'daily'



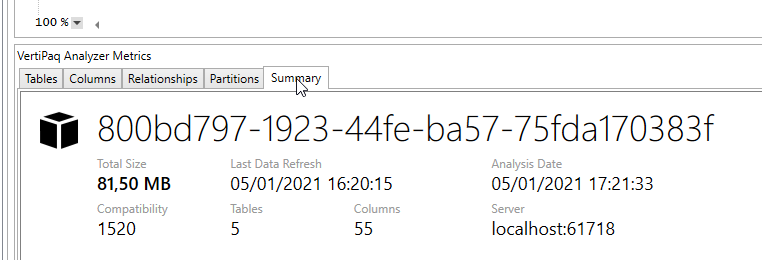
* + Ci colleghiamo in modalità importata a power bi e lo puntiamo alla vista 2020csv che crea la union tra i due mesi (**Oppure partiamo dal file già caricato Demo 1 - Import CSV**)



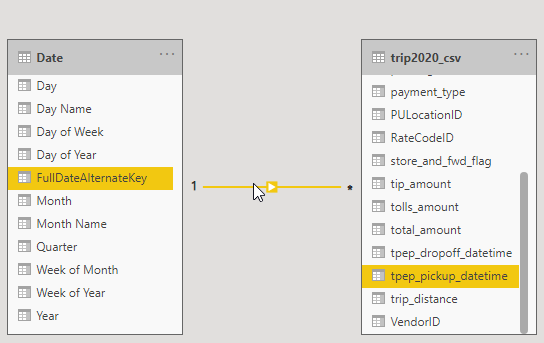
Poi verifichiamo quanto mi è costato il caricamento quindi 15500-14035= 1465 MB = 1.465 GB



* + vediamo il costo **(GBFinali-GBIniziali)** **1.465 GB /1000 \*5. = 0.007325$. Se aggiorno il modello 1 volta al giorno spendo (0.21$)**
  + Possiamo guardare con DAX Studio (**view metrics**) quanto sta occupando il modello per circa **12 milioni di righe cioè 81.50mb**



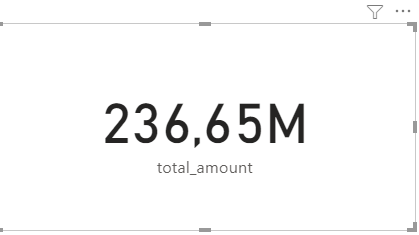
* + Avrò bisogno di una tabella delle **dimensioni della data** appropriata per testare diversi scenari e la creerò usando Power Query. Questa tabella di date sarà in modalità di importazione in tutti gli scenari, quindi non dovrebbe influire sulla quantità di dati elaborati dal pool SQL Serverless. Nel file **DimDate.m** c’è lo script



* + Creiamo una griglia dove mettiamo **fullDatealternatekey** e **total\_amount**



Creo anche una card con il totale di **total\_amount**

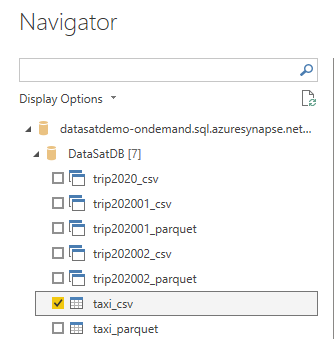


Mettiamo anche uno slicer su **fullDatealternatekey**



Posso creare quanti visual voglio ma non pagherò più nienete perché i costi sono in memoria.

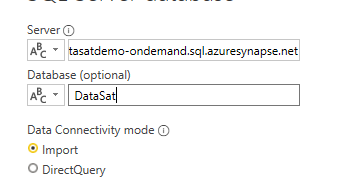
* ----------------------------------------Passo 2
  + Ora ripetiamo la stessa esperienza di collegarsi in modalità importata ma alla **tabella esterna** **trip\_2020\_csv\_ext**. Prima però verifichiamo quanto ammontano i MB usati **15505MB**



Poi verifichiamo quanto mi è costato il caricamento quindi 15787-15505= 282 MB = **0,282GB** (esattamente la dimensione del file csv nella cartella ext… quindi non legge tutti i csv)



* + vediamo il costo **(GBFinali-GBIniziali)** **0,282GB /1000 \*5. = 0.00141$. Se aggiorno il modello 1 volta al giorno spendo (0.04$)!!!!! YEAH grande risparmio con le tabelle esterne**
* ----------------------------------------Passo 3
  + Ci colleghiamo in modalità importata a power bi e lo puntiamo alla vista **dbo.2020parquet** che crea la union tra i due mesi (**Oppure partiamo dal file già caricato Demo 1 - Import Parquet**)



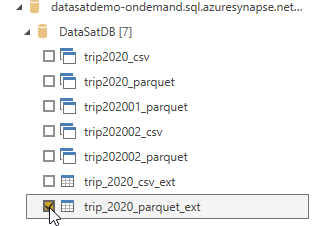
* + Prima di eseguire il caricamento vedo quanti mega ho bruciato 5716 MB



Poi verifichiamo quanto mi è costato il caricamento quindi 6310-5716 = 594 MB = 0.594 GB



* + vediamo il costo **(GBFinali-GBIniziali)** **0.594** **GB /1000 \*5. = 0.00297$. Se aggiorno il modello 1 volta al giorno spendo (0.08$)**
* ----------------------------------------Passo 4
  + Ora ripetiamo la stessa esperienza di collegarsi in modalità importata ma alla **tabella esterna** **dbo.** **trip\_2020\_parquet\_ext**. Prima però verifichiamo quanto ammontano i MB usati **15505MB**



Poi verifichiamo quanto mi è costato il caricamento quindi 13630-13457= 173 MB = **0,173GB** (esattamente la dimensione del file parquet nella cartella ext… quindi non legge tutti i parquet)



* + vediamo il costo **(GBFinali-GBIniziali)** **0,173GB** **/1000 \*5. = 0.0008$. Se aggiorno il modello 1 volta al giorno spendo (0.02$)!!!!! YEAH grande risparmio con le tabelle esterne**
* Test 2: **(Collegamento in modalità DQ su CSV)**
* ----------------------------------------Passo 1
  + Ci colleghiamo DQ al nostro database partendo dal file di Power BI (**Demo 1 - Impot csv.pbix**) con i dati importati da CSV e cancelliamo la tabella ricaricandola in DQ con i dati presi dalla vista **dbo.trip2020\_csv**
  + Prendiamo nota dei mega prima di effettuare il caricamento **32069 MB**



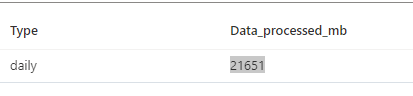
* + Ora eseguo l’aggiornamento dei dati e vediamo che la query ha bruciato (35825-32069)/1000 = 3,7 GB, che è **0,0118$. Più del test 1 0.007325$. perché consuma solo 1.4gb** Tieni presente che quando utilizzi DirectQuery, ogni oggetto visivo genererà una query separata per l'origine dati sottostante. Come puoi concludere, aumentando il numero degli elementi visivi nell'area di disegno del report, aumenterai anche i costi.
  + **Un'altra cosa importante da tenere a mente**: questi costi sono per utente! Quindi, se hai 10 utenti che eseguono lo stesso report in parallelo, dovresti moltiplicare i costi per 10, poiché verrà generata una nuova query per ogni oggetto visivo e per ogni utente.
* ----------------------------------------Passo 2
  + Ora partendo dal file di prima sostituiamo la vista con la tabella estesa **dbo.trip2020\_csv\_ext e vediamo prima** dell’aggiornamento e dopo l’aggiornamento i valori dei mega byte
  + Prendiamo nota dei mega prima di effettuare il caricamento **36658 MB**



* + Ora eseguo l’aggiornamento dei dati e vediamo che la query ha bruciato (37228-36658)/1000 = 0,5GB, che è **0,0028$. Più del**
* ----------------------------------------Passo 3
  + Ci colleghiamo al file del passo 1 (**Demo 2 - DQ csv.pbix**) con i dati presi dalla vista **dbo.trip2020\_parquet**
  + Prendiamo nota dei mega prima di effettuare il caricamento **21192MB**



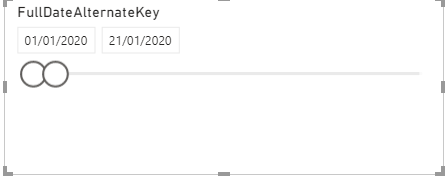
* + Ora eseguo l’aggiornamento dei dati e vediamo che la query ha bruciato (21513-21192)/1000 = 0,3 GB, che è **0,001$**
* ----------------------------------------Passo 4
  + Ci colleghiamo al file del passo 3 con i dati presi dalla vista dalla tabella estesa **dbo.trip\_2020\_parquet\_ext**
  + Prendiamo nota dei mega prima di effettuare il caricamento **21651MB**



* + Ora eseguo l’aggiornamento dei dati e vediamo che la query ha bruciato (21666-21651)/1000 = 0,015 GB, che è **0,000075$**
* Test 3: **(Collegamento in modalità DQ su CSV e filtraggio dati)**
  + Ora, prendo il file voglio controllare cosa succede se seleziono un intervallo di date specifico all'interno del mio filtro dei dati, ad esempio **tra il 1 ° gennaio e il 4 gennaio**
  + Prima di spostare lo slicer vedere i megabyte 41812



* + Poi spostiamo lo slicer **tra il 1 ° gennaio e il 4 gennaio** e vedere i megabyte



* Ora eseguo l’aggiornamento dei dati e vediamo che la query ha bruciato (45568-41812)/1000 = 3,7 GB, che è **0,01878$ la prima cosa che noto è che la query per la griglia mi è costata esattamente lo stesso!** La cosa strana è che, se guardo la query SQL generata per recuperare i dati, posso vedere che il motore era abbastanza intelligente da applicare il filtro della data nella clausola WHERE **Vedere con il profiler!!!!! Che sono applicati i where MA…..**

Tuttavia, sembra che la vista sottostante scansiona l'intera porzione di dati all'interno del file CSV! Quindi, non c'è alcun vantaggio in termini di risparmio se si utilizza il filtro dei dati per limitare il volume di dati, poiché l'intero file CSV verrà comunque scansionato **... (lo slicer non incide in niente!!!!)**

* Test 4: (**Collegamento in modalità DQ su CSV e tabelle aggregate usando le aggregazioni di Power BI**)
  + Il prossimo test ci mostrerà cosa succede se creiamo una tabella aggregata e la memorizziamo in modalità DirectQuery all'interno di Power BI. (**anche qui non cambia nulla legge sempre tutti i file dietro perché è una vista e non una tabella esterna**)

--Creo vista per aggregazione per csv

DROP VIEW IF EXISTS trip2020\_csv\_agg;

GO

CREATE VIEW trip2020\_csv\_agg AS

select SUM(total\_amount) as total\_amount,tpep\_pickup\_datetime

from dbo.trip2020\_csv

group by tpep\_pickup\_datetime

--Creo vista per aggregazione per parquet

DROP VIEW IF EXISTS trip2020\_parquet\_agg;

GO

CREATE VIEW trip2020\_parquet\_agg AS

select SUM(cast(total\_amount as decimal)) as total\_amount,tpep\_pickup\_datetime

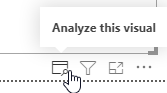
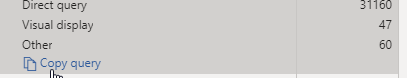
from dbo.trip2020\_parquet

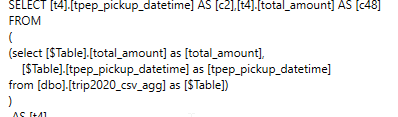
group by tpep\_pickup\_datetime





* + **Ora attivo l’analizzatore e clicco il refresh solo della griglia** . Catturo la query che ne esce e la porto su DAX studio dove vedo

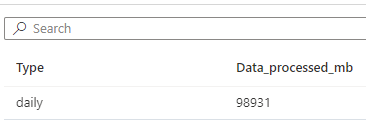
* + **Lancio l’aggiornamento** e la query matchia le aggregazioni (mostrare solo per la griglia) ma la quantità di dati trasferiti 94917-91161= **3,756GB** per la griglia è sempre quella di leggersi **dbo.trip2020\_csv anche se va sulla aggregazione questo perché comunque dietro le quinte si legge tutti i file come per il caso del filtro del test 3** 



* + Allora!!!!! Provo a portare la tabella aggregata in memoria 😊 e vediamo così il test 5
* Test 5: (**Collegamento in modalità DQ su CSV e tabelle aggregate usando le aggregazioni di Power BI**)
  + Mi aspetto che oltre ad essere più veloce avrò un costo minore perché essendo che è stata aggiunta in memoria al prossimo refresh non dovrò più sostenere questo costo come nel test3. Se ora aggiorno il report



* + Prendo nota dei mega **98931MB**



* + Infatti quando aggiorno sarà quello di recuperare i dati della tabella delle corse dei taxi e a aggregarla **100183**-**98931= 1.252GB = 0,00626$ ma se clicco nella griglia e nella card non abbiamo costi**
  + Sostituiamo la vista che aggrega e mettiamo la tabella **dbo.trip\_2020\_csv\_ext** **104463-104273= 0.19GB = 0,00095$ ma se clicco nella griglia e nella card non abbiamo costi**
  + Sostituiamo la vista che aggrega e mettiamo la tabella **dbo.trip2020\_parquet** **105052-104945= 0.10GB = 0,00053$ ma se clicco nella griglia e nella card non abbiamo costi**
  + Sostituiamo la vista che aggrega e mettiamo la tabella **dbo.trip2020\_parquet** **105058-105052= 0.006GB = 0,00003$ ma se clicco nella griglia e nella card non abbiamo costi**