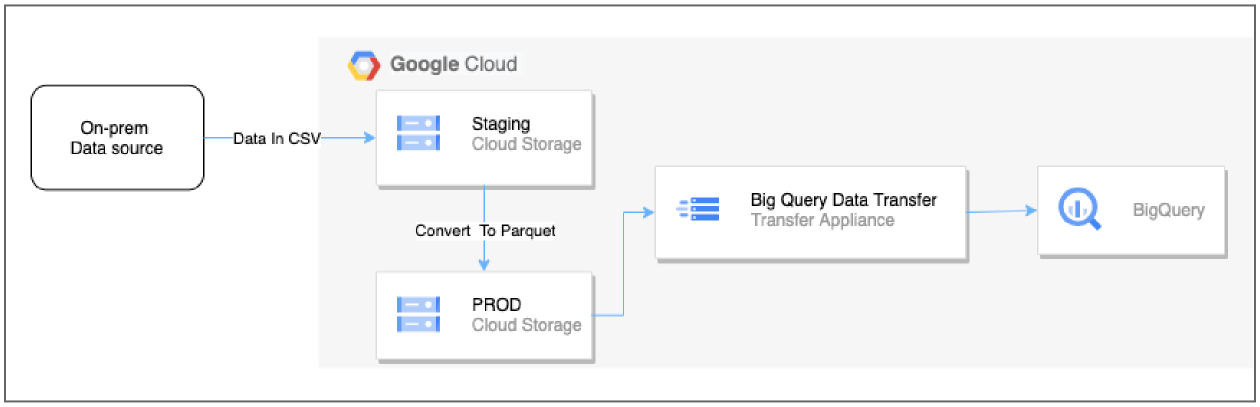
Using BigQuery Data Transfer Service

Here is a working solution which explains a working demo for data load from Google Cloud Storage to BigQuery Data Transfer Service.



Here are the following Steps that load the data to Big query from GCS using Big Query.

Prerequisites

You must have billing enabled in your GCP project. If not sure, follow these instructions.

APIs to be enabled

BigQuery API
BigQuery Data Transfer API

IAM Roles and Permissions

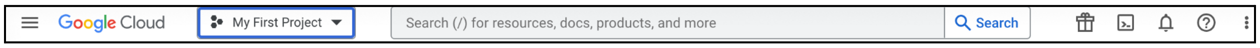
The only predefined IAM roles with all the BigQuery and Cloud Storage permissions are **BigQuery Admin** (roles/bigquery.admin) and **Storage Admin** (roles/storage.admin)

Follow these instructions to grant the IAM roles.

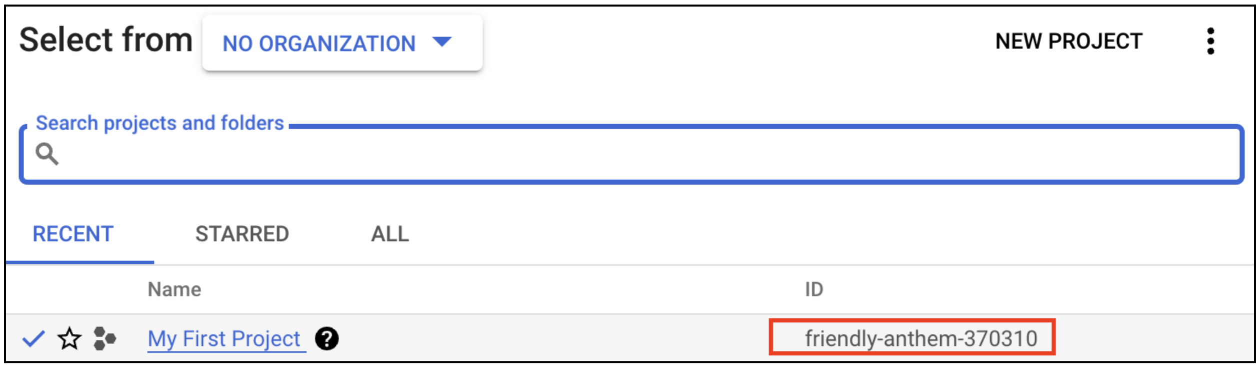
Steps to setup development environment on Cloud Shell Editor

Prerequisites: Gather project ID

On the GCP Console top bar, click on project name (besides Google Cloud logo) e.g. My First Project is referred to in the image below.

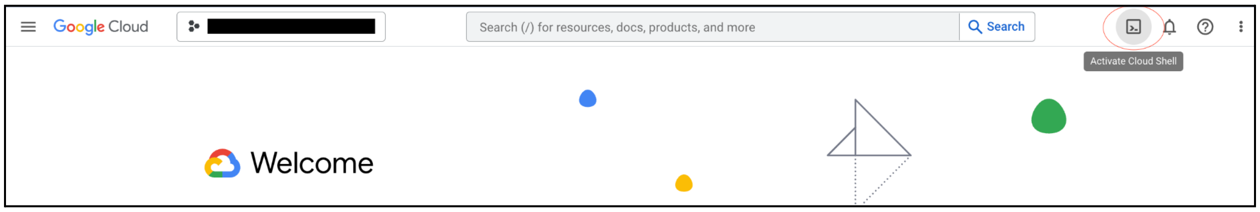


Pop up will appear which contains Project **Name** and Project **ID.** Save the project ID for further usage.

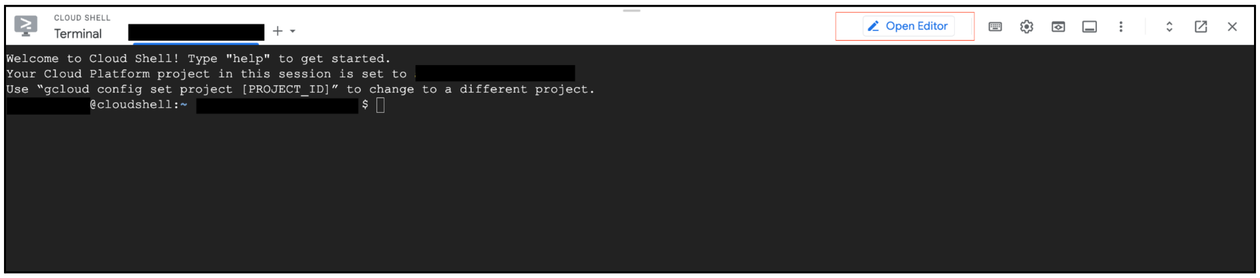


Name	ID
My First Project	friendly-anthem-370310

You can use Cloud Shell Editor to run lab's commands. On the GCP Console, on top right click on **Activate Cloud Shell**.

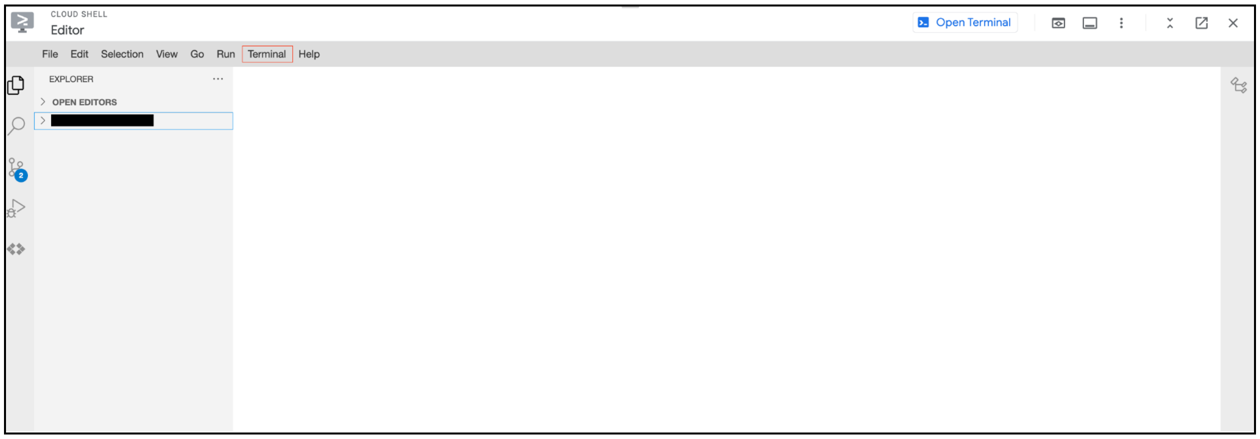


Cloud Shell terminal will open at the bottom of the screen. Click on the **Open Editor** button to access the Cloud Shell Editor.



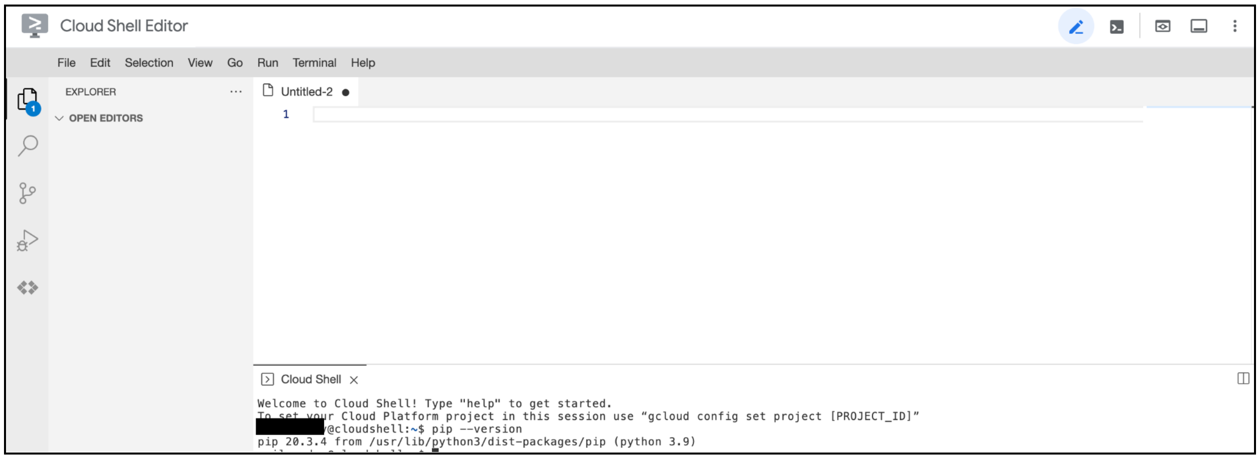
```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to [redacted]
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
[redacted]@cloudshell:~ [redacted] $
```

Cloud Shell Editor will be opened refer to below screenshot. Click on **Terminal** to open terminal to run various operations.



Run the following commands on the cloud shell editor terminal to check if you have PIP already installed.

```
pip --version
```



```
Welcome to Cloud Shell! Type "help" to get started.
To set your Cloud Platform project in this session use "gcloud config set project [PROJECT_ID]"
[redacted]@cloudshell:~$ pip --version
pip 20.3.4 from /usr/lib/python3/dist-packages/pip (python 3.9)
```

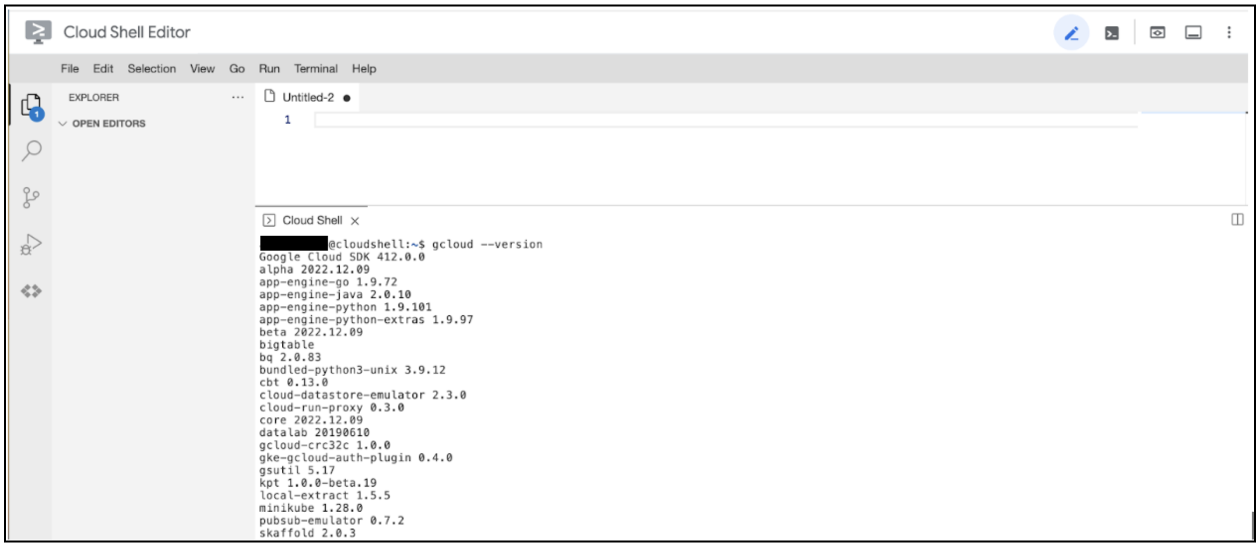
Note: If it is not installed, follow these instructions to [Install](#) PIP according to the operating system you are using.

To run the Python scripts to interact with GCP resources from cloud shell editor, you will need to set up a development environment using the following steps.

I. Run the following commands on cloud shell editor to check if you have the Google Cloud SDK installed.

```
gcloud --version
```

If you get an output similar to the screenshot below, your SDK installation is done.



```
[redacted]@cloudshell:~$ gcloud --version
Google Cloud SDK 412.0.0
alpha 2022.12.09
app-engine-go 1.9.72
app-engine-java 2.0.10
app-engine-python 1.9.101
app-engine-python-extras 1.9.97
beta 2022.12.09
bigtable
bq 2.0.83
bundled-python3-unix 3.9.12
cbt 0.13.0
cloud-datastore-emulator 2.3.0
cloud-run-proxy 0.3.0
core 2022.12.09
datalab 20190610
gcloud-crc32c 1.0.0
gke-gcloud-auth-plugin 0.4.0
gsutil 5.17
kpt 1.0.0-beta.19
local-extract 1.5.5
minikube 1.28.0
pubsub-emulator 0.7.2
skaffold 2.0.3
```

Note:If it is not installed, follow these instructions to Install and Initialize Google Cloud CLI according to the operating system you are using.

II. Run the following command on the cloud shell editor terminal to set the current working GCP project

```
gcloud config set project PROJECT_ID
```

Make sure you replace PROJECT_ID with the ID of your GCP Project.

Download data for testing

For this workshop we will be using a public Kaggle Dataset on Netflix Movies and TV Shows.

File Size: 3.4 MB
Number of Rows: 8807
Number of Columns: 12

Link to download the dataset: https://www.kaggle.com/datasets/shivamb/netflix-shows

Note: You may have to login or sign up to Kaggle to download the data.
Save it to your computer, extract the zip file and rename it as `netflix_titles.csv`

Create GCS buckets and upload the data

Create two Google Cloud Storage Buckets

- synopsys-staging
- synopsys-prod

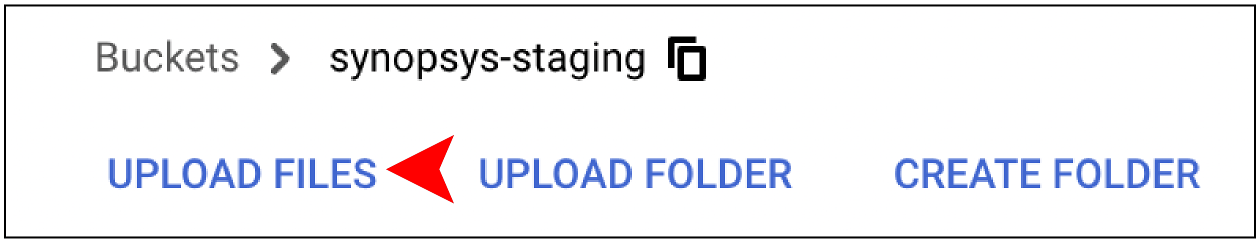
To create the buckets:

1. In the Google Cloud console, go to the Cloud Storage **Buckets** page.
   Go to Buckets
2. Click **Create bucket**.
3. On the **Create a bucket** page, enter your bucket information. To go to the next step, click **Continue**.
   - For **Name your bucket**, you must enter values that are globally unique. Since 'synopsys-staging' and 'synopsys-prod' are already taken, add a suffix to make it unique. Eg 'synopsys-staging-323' or 'synopsys-prod-982'
   - For **Choose where to store your data**, select **Location type** as **Region** and select the region that is closest to you from the dropdown list.

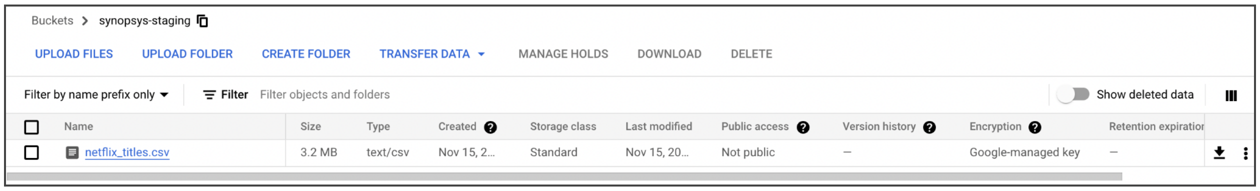
Note: You must replace 'synopsys-staging' and 'synopsys-prod' with the bucket names you created for the rest of the tutorial.

Leave the other options as Default and click CREATE.

Navigate to your staging bucket and click the UPLOAD FILES button to upload the 'netflix_titles.csv' file.



This is how your bucket should look once the file is uploaded.



Name	Size	Type	Created	Storage class	Last modified	Public access	Version history	Encryption	Retention expiratior
netflix_titles.csv	3.2 MB	text/csv	Nov 15, 2...	Standard	Nov 15, 20...	Not public	—	Google-managed key	—

Steps to load CSV data from GCS to BigQuery

The following steps will be used to transfer the CSV file to another GCS Bucket after converting it to Parquet.

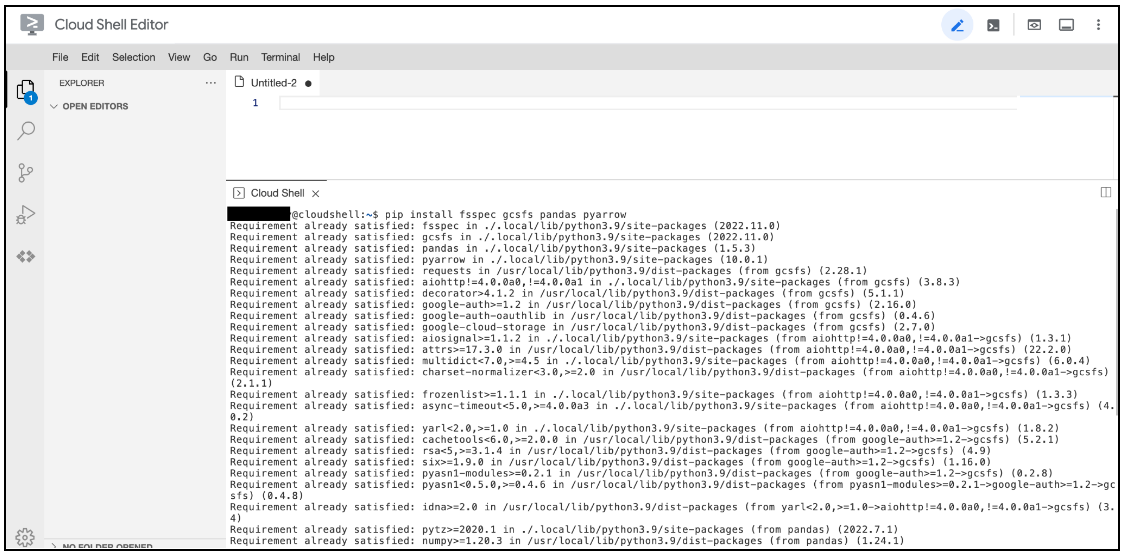
What is Parquet?

Apache Parquet is an open source, column-oriented data file format designed for efficient data storage and retrieval. It provides efficient data compression and encoding schemes with enhanced performance to handle complex data in bulk. We will be converting the large CSV file to Parquet for faster and more efficient processing.

Step 1: CSV to Parquet

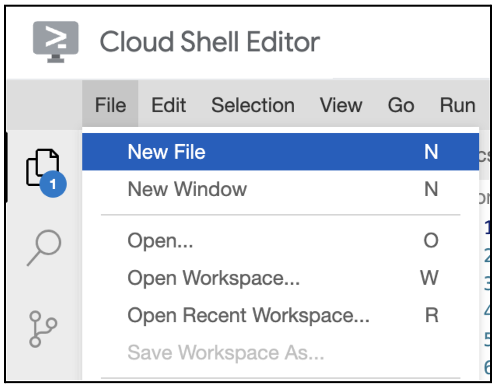
a. Open cloud shell Editor.
b. Run the following command on your cloud shell Editor. terminal to install the prerequisite packages

```
pip install fsspec gcsfs pandas pyarrow
```



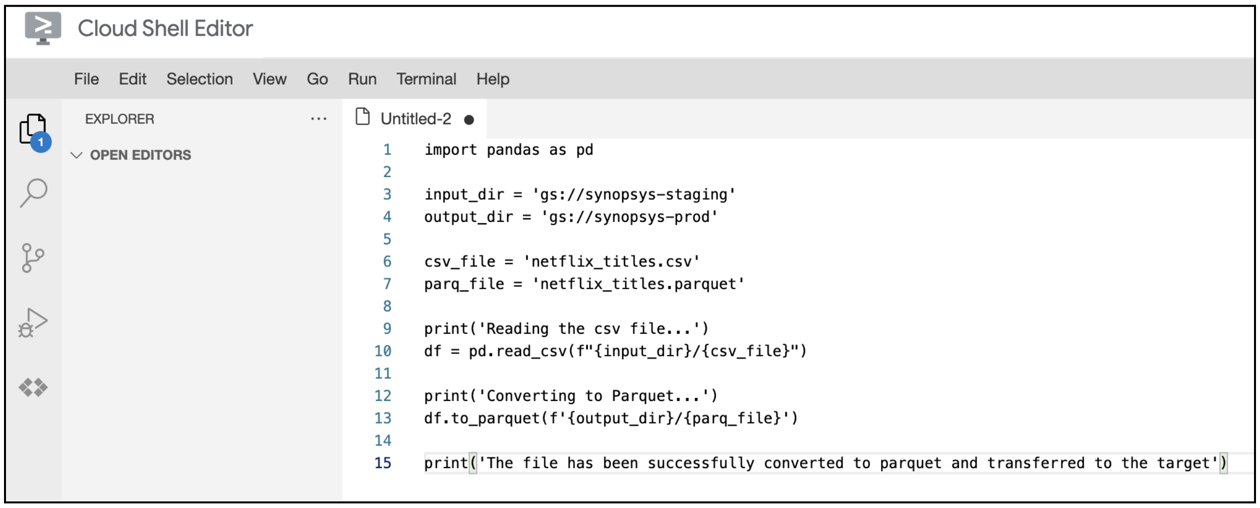
```
[redacted]@cloudshell:~$ pip install fsspec gcsfs pandas pyarrow
Requirement already satisfied: fsspec in ./.local/lib/python3.9/site-packages (2022.11.0)
Requirement already satisfied: gcsfs in ./.local/lib/python3.9/site-packages (2022.11.0)
Requirement already satisfied: pandas in ./.local/lib/python3.9/site-packages (1.5.3)
Requirement already satisfied: pyarrow in ./.local/lib/python3.9/site-packages (10.0.1)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from gcsfs) (2.28.1)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in ./.local/lib/python3.9/site-packages (from gcsfs) (3.8.3)
Requirement already satisfied: decorator>4.1.2 in /usr/local/lib/python3.9/dist-packages (from gcsfs) (5.1.1)
Requirement already satisfied: google-auth>=1.2 in /usr/local/lib/python3.9/dist-packages (from gcsfs) (2.16.0)
Requirement already satisfied: google-auth-oauthlib in /usr/local/lib/python3.9/dist-packages (from gcsfs) (0.4.6)
Requirement already satisfied: google-cloud-storage in /usr/local/lib/python3.9/dist-packages (from gcsfs) (2.7.0)
Requirement already satisfied: aiosignal>=1.1.2 in ./.local/lib/python3.9/site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->gcsfs) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.9/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->gcsfs) (22.2.0)
Requirement already satisfied: multidict<7.0,>=4.5 in ./.local/lib/python3.9/site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->gcsfs) (6.0.4)
Requirement already satisfied: charset-normalizer<3.0,>=2.0 in /usr/local/lib/python3.9/dist-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->gcsfs)
(2.1.1)
Requirement already satisfied: frozenlist>=1.1.1 in ./.local/lib/python3.9/site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->gcsfs) (1.3.3)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in ./.local/lib/python3.9/site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->gcsfs) (4.
0.2)
Requirement already satisfied: yarl<2.0,>=1.0 in ./.local/lib/python3.9/site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->gcsfs) (1.8.2)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from google-auth>=1.2->gcsfs) (5.2.1)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.9/dist-packages (from google-auth>=1.2->gcsfs) (4.9)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.9/dist-packages (from google-auth>=1.2->gcsfs) (1.16.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.9/dist-packages (from google-auth>=1.2->gcsfs) (0.2.8)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.9/dist-packages (from pyasn1-modules>=0.2.1->google-auth>=1.2->gc
sfs) (0.4.8)
Requirement already satisfied: idna>=2.0 in /usr/local/lib/python3.9/dist-packages (from yarl<2.0,>=1.0->aiohttp!=4.0.0a0,!=4.0.0a1->gcsfs) (3.
4)
Requirement already satisfied: pytz>=2020.1 in ./.local/lib/python3.9/site-packages (from pandas) (2022.7.1)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.9/dist-packages (from pandas) (1.24.1)
```

c. Create a new file in cloud shell editor. Refer below screenshot for reference.

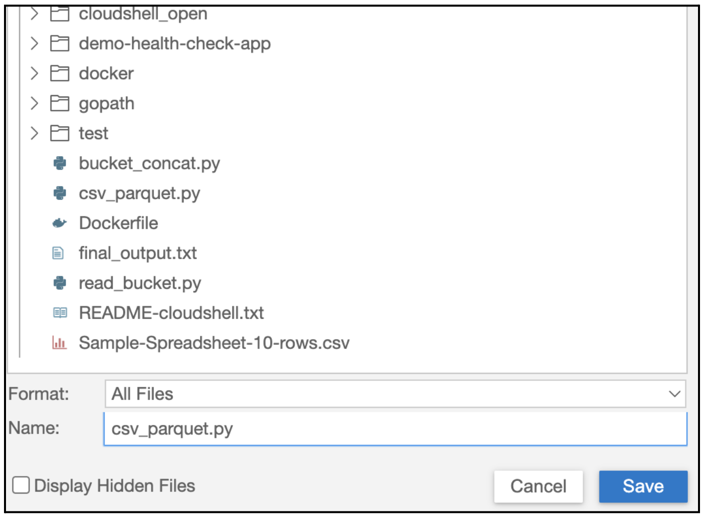


d. Copy and paste below python code in the newly created file. Save the file with the name ‘csv_parquet.py’.

```
import pandas as pd

input_dir = 'gs://synopsys-staging'
output_dir = 'gs://synopsys-prod'

csv_file = 'netflix_titles.csv'
parq_file = 'netflix_titles.parquet'

print('Reading the csv file...')
df = pd.read_csv(f"{input_dir}/{csv_file}")

print('Converting to Parquet...')
df.to_parquet(f'{output_dir}/{parq_file}')

print('The file has been successfully converted to parquet and transferred to the
target')
```



```
import pandas as pd

input_dir = 'gs://synopsys-staging'
output_dir = 'gs://synopsys-prod'

csv_file = 'netflix_titles.csv'
parq_file = 'netflix_titles.parquet'

print('Reading the csv file...')
df = pd.read_csv(f"{input_dir}/{csv_file}")

print('Converting to Parquet...')
df.to_parquet(f'{output_dir}/{parq_file}')

print('The file has been successfully converted to parquet and transferred to the target')
```

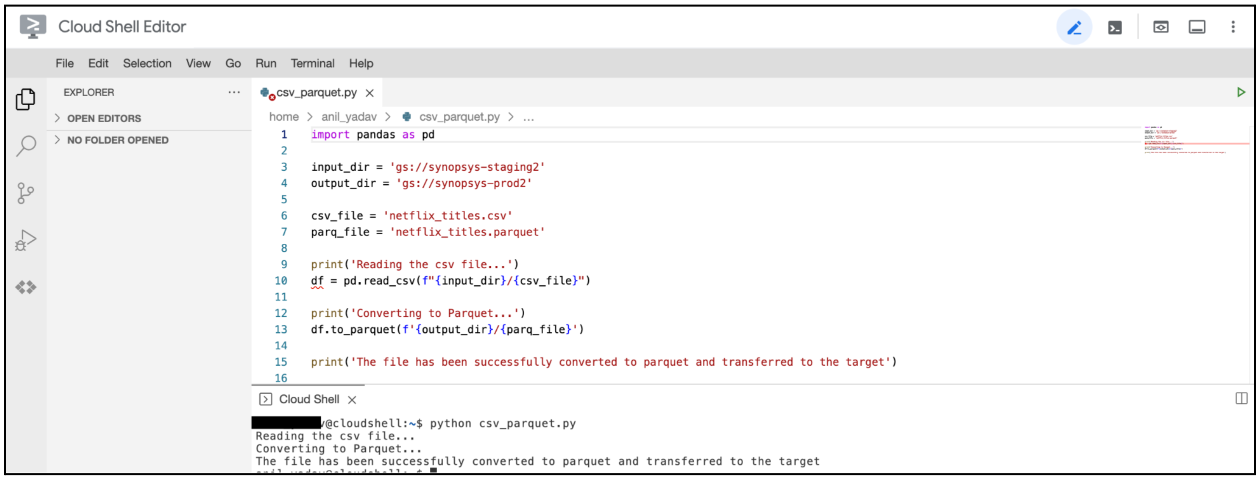

cloudshell_open
demo-health-check-app
docker
gopath
test
bucket_concat.py
csv_parquet.py
Dockerfile
final_output.txt
read_bucket.py
README-cloudshell.txt
Sample-Spreadsheet-10-rows.csv

Format: All Files
Name: csv_parquet.py

e. Change the values for input_dir and output_dir to the staging and production buckets if necessary.
f. Change the values for csv_file and parq_file accordingly with the proper extensions if necessary.
g. Run the script using the following command and you'll find the new parquet file on the prod bucket

```
python csv_parquet.py
```

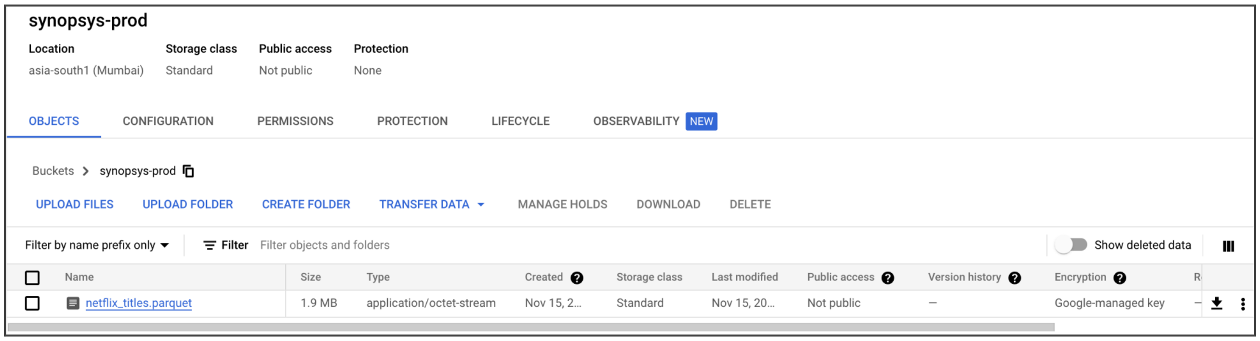
This is how your output should look like.


Cloud Shell Editor
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
NO FOLDER OPENED
csv_parquet.py
home > anil_yadav > csv_parquet.py > ...
Cloud Shell

```
import pandas as pd

input_dir = 'gs://synopsys-staging2'
output_dir = 'gs://synopsys-prod2'

csv_file = 'netflix_titles.csv'
parq_file = 'netflix_titles.parquet'

print('Reading the csv file...')
df = pd.read_csv(f"{input_dir}/{csv_file}")

print('Converting to Parquet...')
df.to_parquet(f'{output_dir}/{parq_file}')

print('The file has been successfully converted to parquet and transferred to the target')
```

```
[redacted]@cloudshell:~$ python csv_parquet.py
Reading the csv file...
Converting to Parquet...
The file has been successfully converted to parquet and transferred to the target
```

Once the file is successfully converted and transferred, you'll find it in the 'synopsys-prod' bucket.



Location	Storage class	Public access	Protection
asia-south1 (Mumbai)	Standard	Not public	None

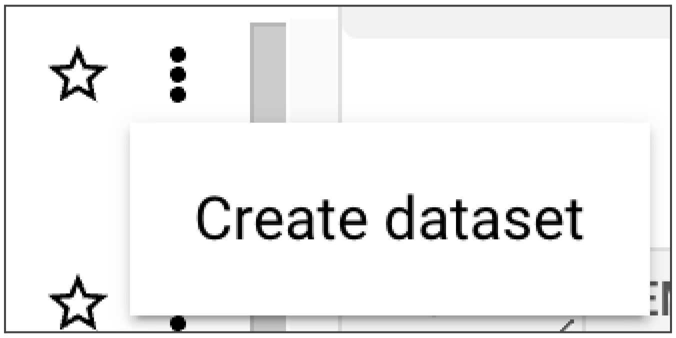
Name	Size	Type	Created	Storage class	Last modified	Public access	Version history	Encryption
netflix_titles.parquet	1.9 MB	application/octet-stream	Nov 15, 2...	Standard	Nov 15, 20...	Not public	—	Google-managed key

Step 2: GCS to BQ

Now that we have the Parquet file ready on a GCS bucket, we will be transferring it to BigQuery using the BigQuery Data Transfer Service.

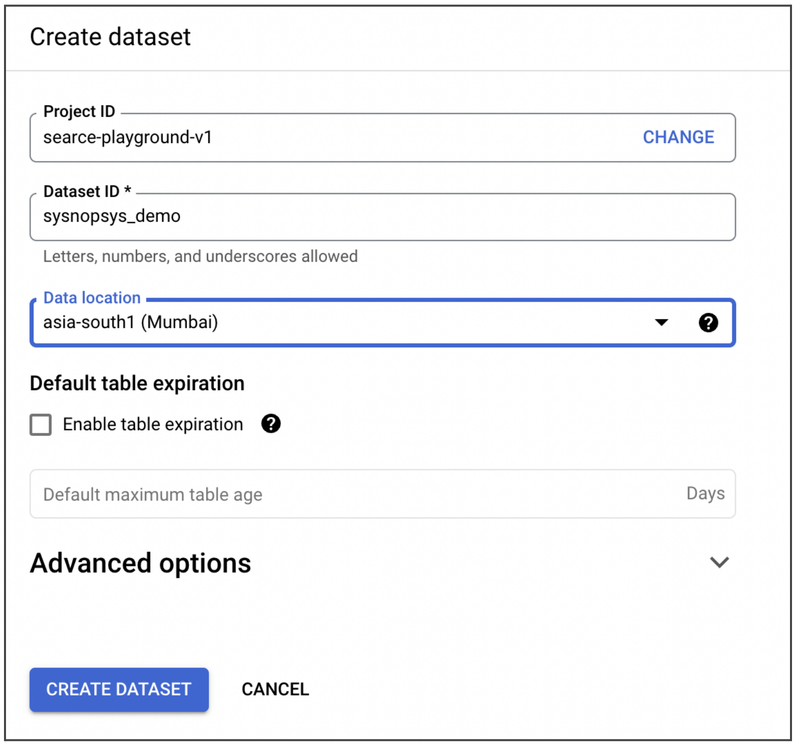
First we need to **create a dataset in BigQuery**.

A. Open the BigQuery page in the Google Cloud console.
B. Go to the BigQuery page
C. Click on SQL Workshop.
D. In the Explorer panel, select the project where you want to create the dataset.
E. Expand the Actions (three vertical dots) option next to your project name and click Create dataset.



F. On the Create dataset page:
   a. For Dataset ID, enter a unique dataset name, in this case we use `synopsys_demo`.
   b. For Data location, choose a geographic location for the dataset. **This location should be the same as the one you used for creating the GCS buckets.**

Leave the other options as Default and click CREATE DATASET.


Create dataset

Project ID: searce-playground-v1 — CHANGE

Dataset ID *: sysnopsys_demo

Letters, numbers, and underscores allowed

Data location: asia-south1 (Mumbai)

Default table expiration

☐ Enable table expiration

Default maximum table age — Days

Now we need to create an empty table in that dataset.

Run the following query in the Query **Editor** to create a new table 'netflix_title' in our Dataset. Make sure you replace PROJECT_ID with the ID of your GCP project.

Copy and paste the below query in SQL Workshop editor.

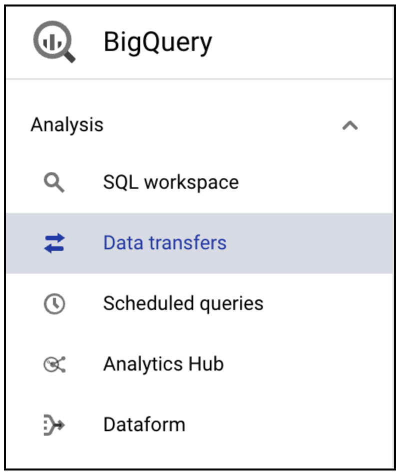
```
CREATE TABLE `PROJECT_ID.synopsys_demo.netflix_titles`
(
show_id STRING,
type STRING,
title STRING,
director STRING,
`cast` STRING,
country STRING,
date_added STRING,
release_year INTEGER,
rating STRING,
duration STRING,
listed_in STRING,
description STRING
);
```

Note: We are using backticks (`) for `cast` because it is a reserved keyword in BigQuery

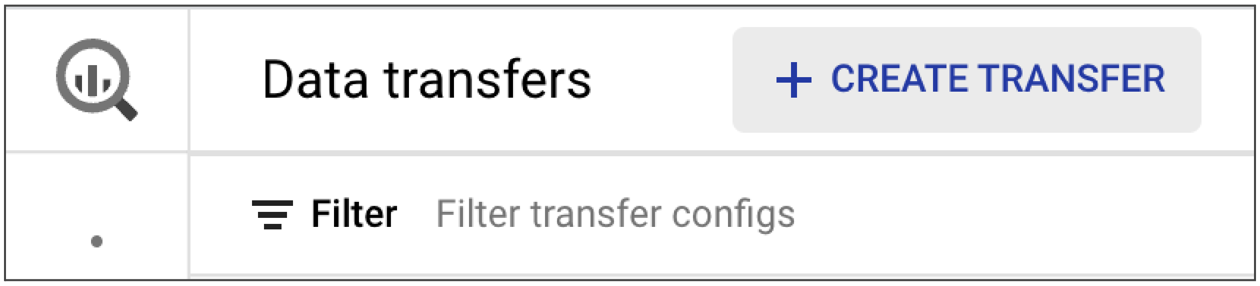
There are two ways to run the BigQuery Transfer. One is via using the console, and the other is using the CLI.

Using the Console - BQ Transfer

Open the Data Transfer Dashboard from BigQuery

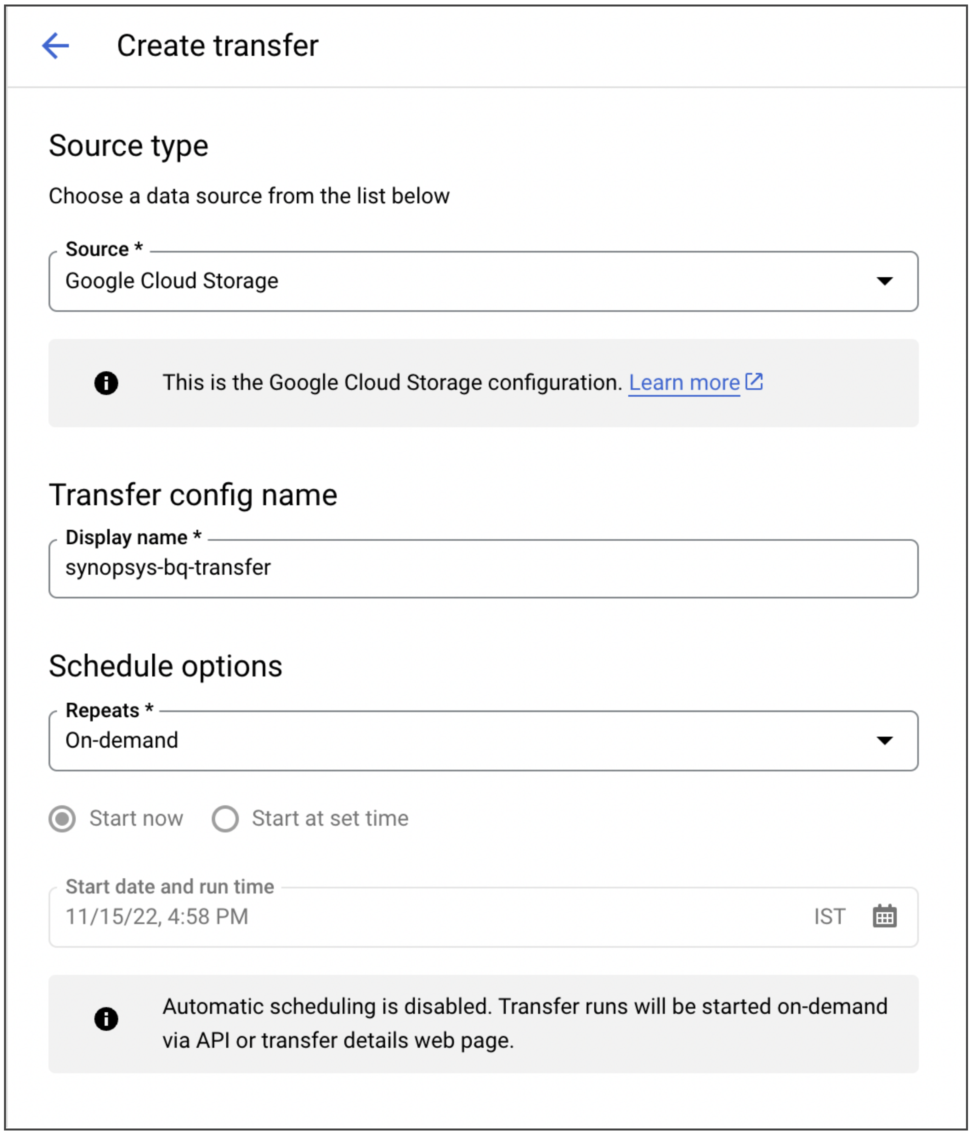


Select “Create Transfer”.



Enter the configuration as follows:

Source: **Google Cloud Storage** as our source file is in a GCS bucket.
Display name: **synopsys-bq-transfer**
Repeats: **On-demand**



Create transfer

Source type

Choose a data source from the list below

Source *: Google Cloud Storage

This is the Google Cloud Storage configuration. Learn more

Transfer config name

Display name *: synopsys-bq-transfer

Schedule options

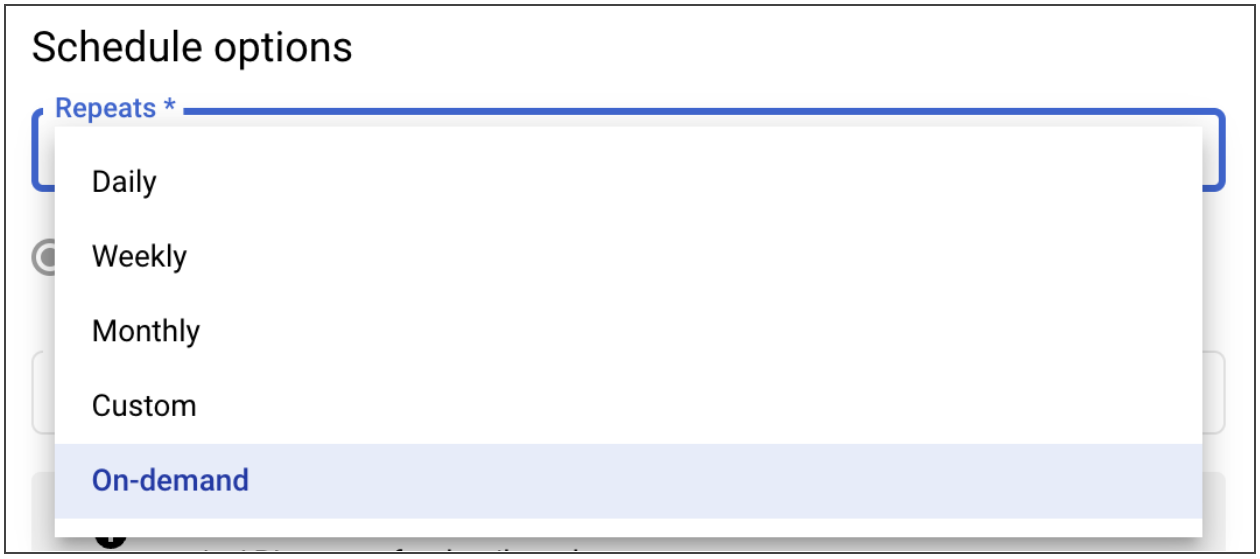
Repeats *: On-demand

[✓ Start now] Start at set time

Start date and run time: 11/15/22, 4:58 PM IST

Automatic scheduling is disabled. Transfer runs will be started on-demand via API or transfer details web page.

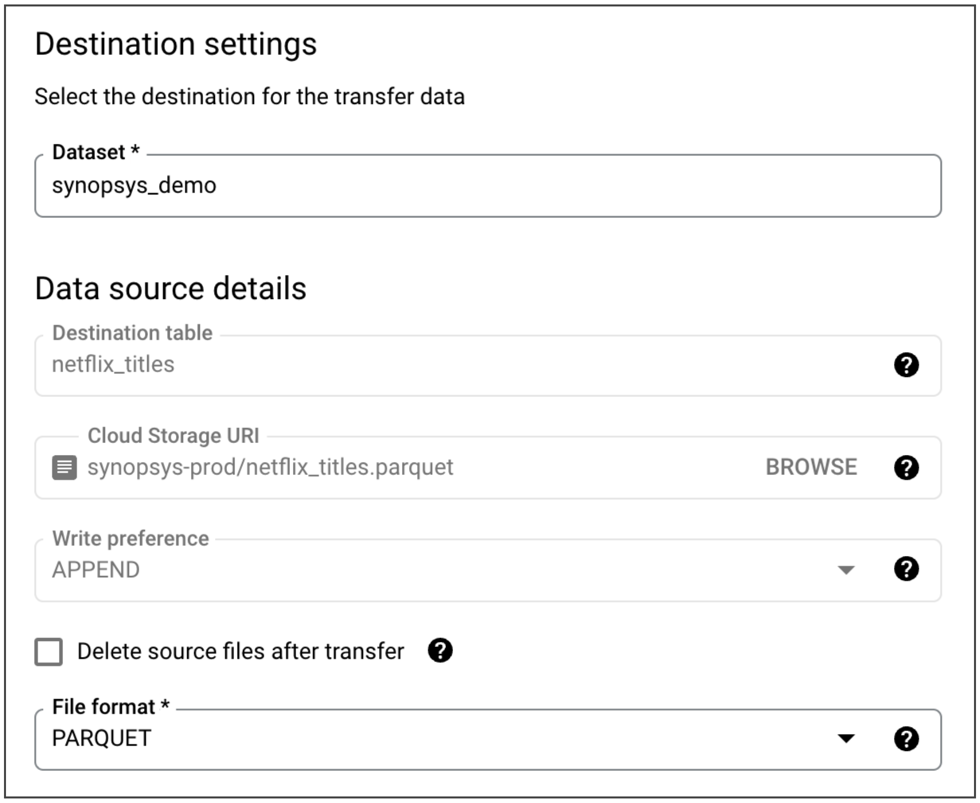
If you want the transfer to be repeated according to a fixed Schedule, you can select one of the other options.



Schedule options

Repeats *

In this case, we are only going to make a one time transfer so we can go with **On-demand.** This way we can run the transfer whenever we want.



Destination settings

Select the destination for the transfer data

Dataset *: synopsys_demo

Data source details

Destination table: netflix_titles

Cloud Storage URI: synopsys-prod/netflix_titles.parquet

Write preference: APPEND

[ ] Delete source files after transfer

File format *: PARQUET

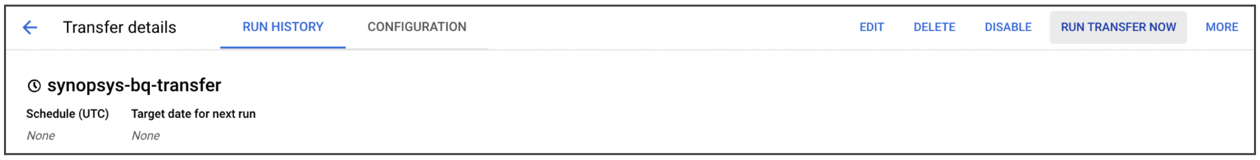
Dataset: **synopsys_demo** name of the dataset we created.
Destination table: **netflix_titles** name of the table we created.
Cloud Storage URI: **synopsys-prod/netflix_titles.parquet** GCS URI of the source file. You can use the BROWSE option to select the file.

Write preference: **APPEND** as we will be adding the data to an existing table.
File format: **PARQUET** as our source file is in Parquet format.

Leave the other options as Default and click the SAVE button.

Run the Transfer job

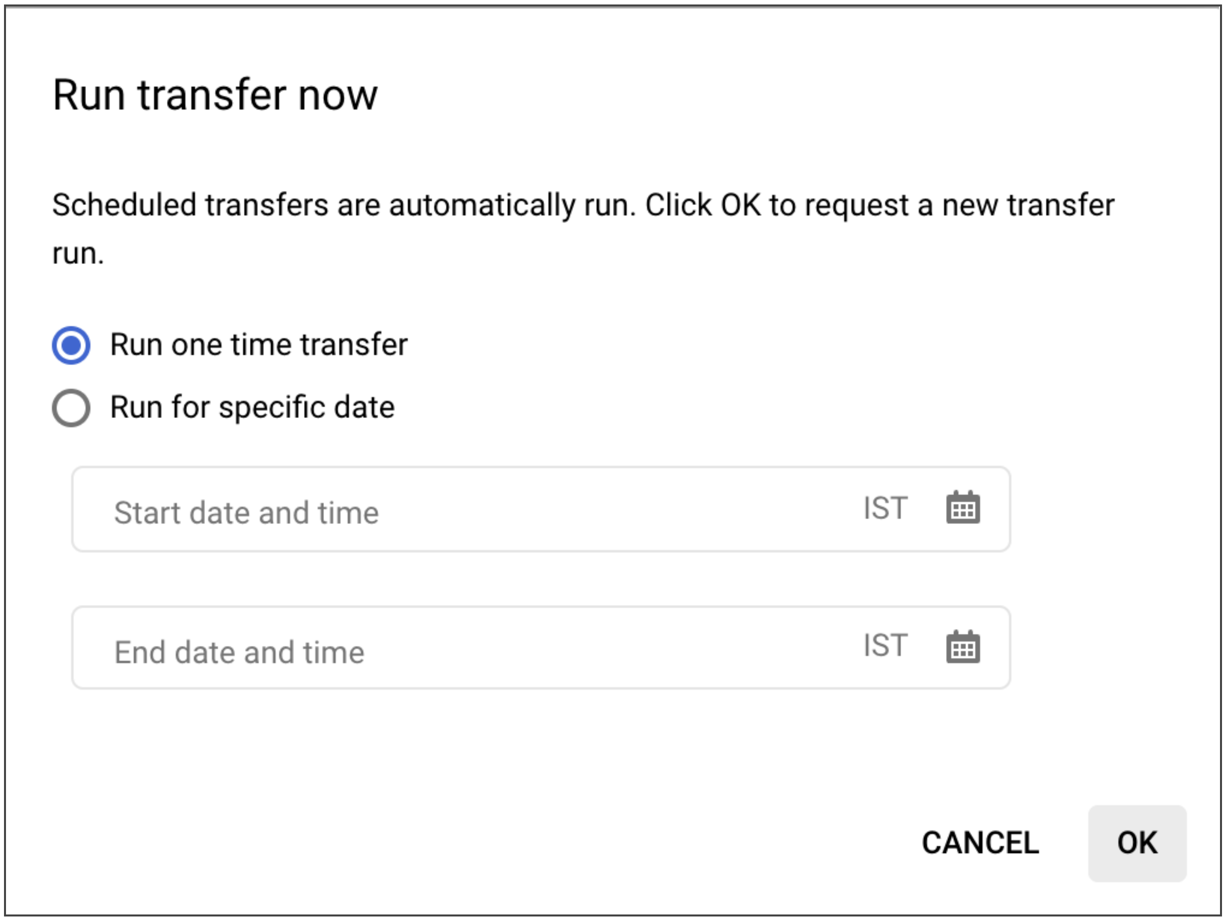
Click the RUN TRANSFER NOW button on the top right corner to begin the transfer process.



synopsys-bq-transfer

Schedule (UTC)	Target date for next run
None	*None*

We are only going to run it for one time, so select the first option and click OK



Run transfer now

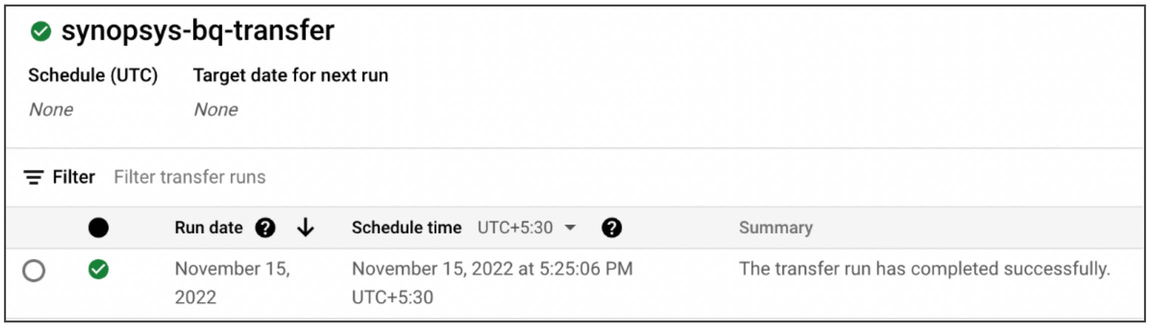
Scheduled transfers are automatically run. Click OK to request a new transfer run.

[✓ Run one time transfer] Run for specific date

Start date and time IST

End date and time IST

Once the transfer is complete, you can find the resulting table on BQ



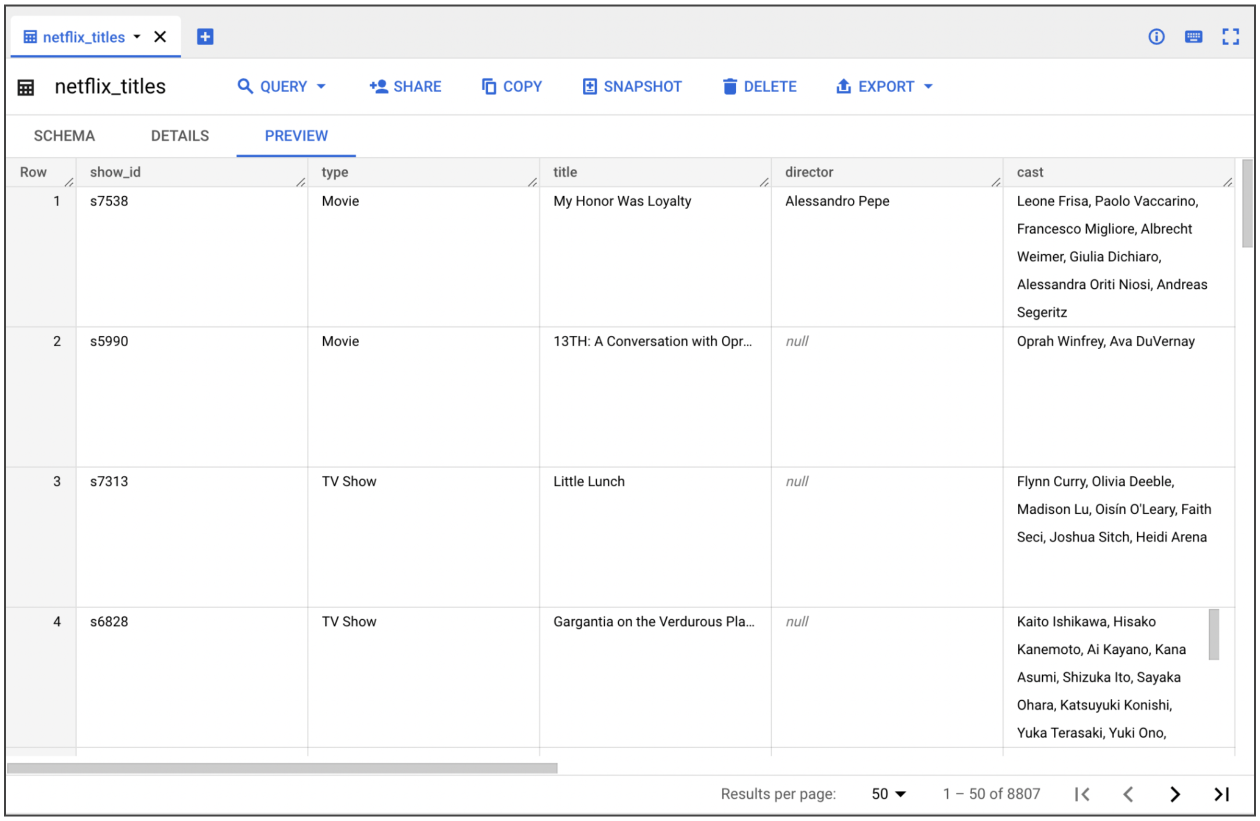
synopsys-bq-transfer

Schedule (UTC): *None*
Target date for next run: *None*

Filter   Filter transfer runs

	●	Run date	Schedule time UTC+5:30	Summary
○	✓	November 15, 2022	November 15, 2022 at 5:25:06 PM UTC+5:30	The transfer run has completed successfully.

To view the BQ Table :
Go to the Explorer Panel, first expand the project and then expand the dataset you created, and then select the table
Click on Preview, this is how the BQ table should look like



netflix_titles

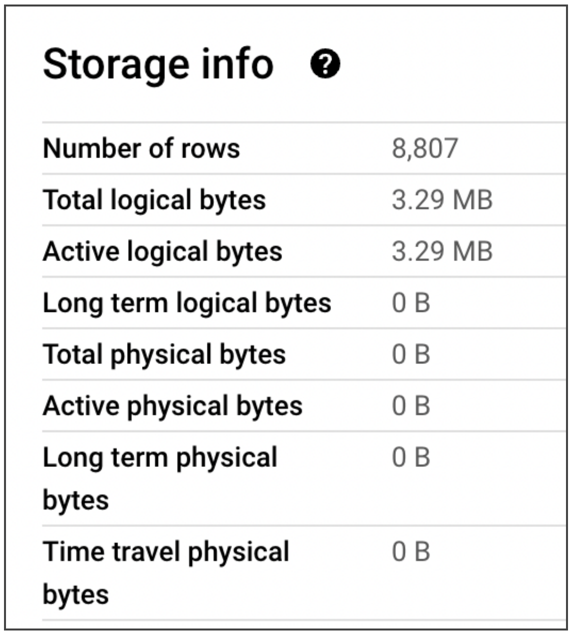
QUERY   SHARE   COPY   SNAPSHOT   DELETE   EXPORT

SCHEMA   DETAILS   PREVIEW

Row	show_id	type	title	director	cast
1	s7538	Movie	My Honor Was Loyalty	Alessandro Pepe	Leone Frisa, Paolo Vaccarino, Francesco Migliore, Albrecht Weimer, Giulia Dichiaro, Alessandra Oriti Niosi, Andreas Segeritz
2	s5990	Movie	13TH: A Conversation with Opr...	*null*	Oprah Winfrey, Ava DuVernay
3	s7313	TV Show	Little Lunch	*null*	Flynn Curry, Olivia Deeble, Madison Lu, Oisín O'Leary, Faith Seci, Joshua Sitch, Heidi Arena
4	s6828	TV Show	Gargantia on the Verdurous Pla...	*null*	Kaito Ishikawa, Hisako Kanemoto, Ai Kayano, Kana Asumi, Shizuka Ito, Sayaka Ohara, Katsuyuki Konishi, Yuka Terasaki, Yuki Ono,

Results per page: 50   1 – 50 of 8807

From the DETAILS tab, you can see that we have successfully added all the 8,807 rows and the table size is 3.29 MB.



Storage info

Number of rows	8,807
Total logical bytes	3.29 MB
Active logical bytes	3.29 MB
Long term logical bytes	0 B
Total physical bytes	0 B
Active physical bytes	0 B
Long term physical bytes	0 B
Time travel physical bytes	0 B

Method 2: Using CLI

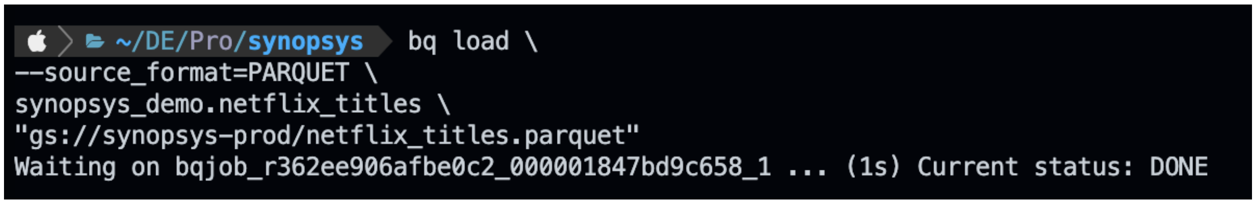
The transfer can also be done using the 'bq load' CLI command

```
bq load \
--source_format=PARQUET \
synopsys_demo.netflix_titles \
"gs://<bucket-name>/netflix_titles.parquet"
```

Note : Replace `<bucket-name>` `with your prod bucket name`
The parameters used are:

1. The file format of the source file, in this case it is **PARQUET**
2. The destination table in the form 'DATASET.TABLE_NAME', in this case, its **synopsys_demo.netflix_titles**
3. The URI of the source file, in this case its **gs://synopsys-prod/netflix_titles.parquet**

Once the job has successfully run, you'll find the following output.



```
~/DE/Pro/synopsys  bq load \
--source_format=PARQUET \
synopsys_demo.netflix_titles \
"gs://synopsys-prod/netflix_titles.parquet"
Waiting on bqjob_r362ee906afbe0c2_000001847bd9c658_1 ... (1s) Current status: DONE
```

The disadvantage of using this method is that you can't schedule the transfers to run at particular times.

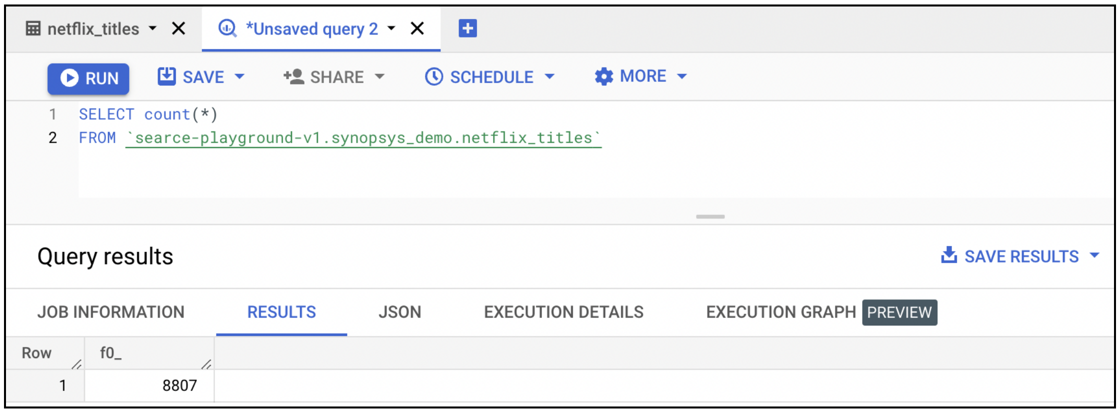
Step 3 : BQ sample Queries

a. Total number of records

Query

```
SELECT count(*)
FROM `<project_id>.synopsys_demo.netflix_titles`
```

Results



```
SELECT count(*)
FROM `searce-playground-v1.synopsys_demo.netflix_titles`
```

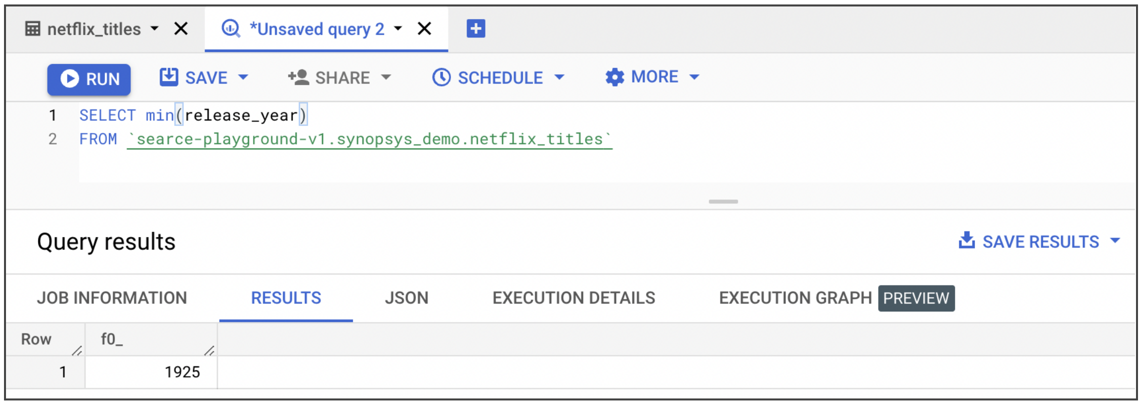
Row	f0_
1	8807

b. Find the year in which the oldest title was released

Query

```
SELECT min(release_year)
FROM `<project_id>.synopsys_demo.netflix_titles`
```

Results



```
SELECT min(release_year)
FROM `searce-playground-v1.synopsys_demo.netflix_titles`
```

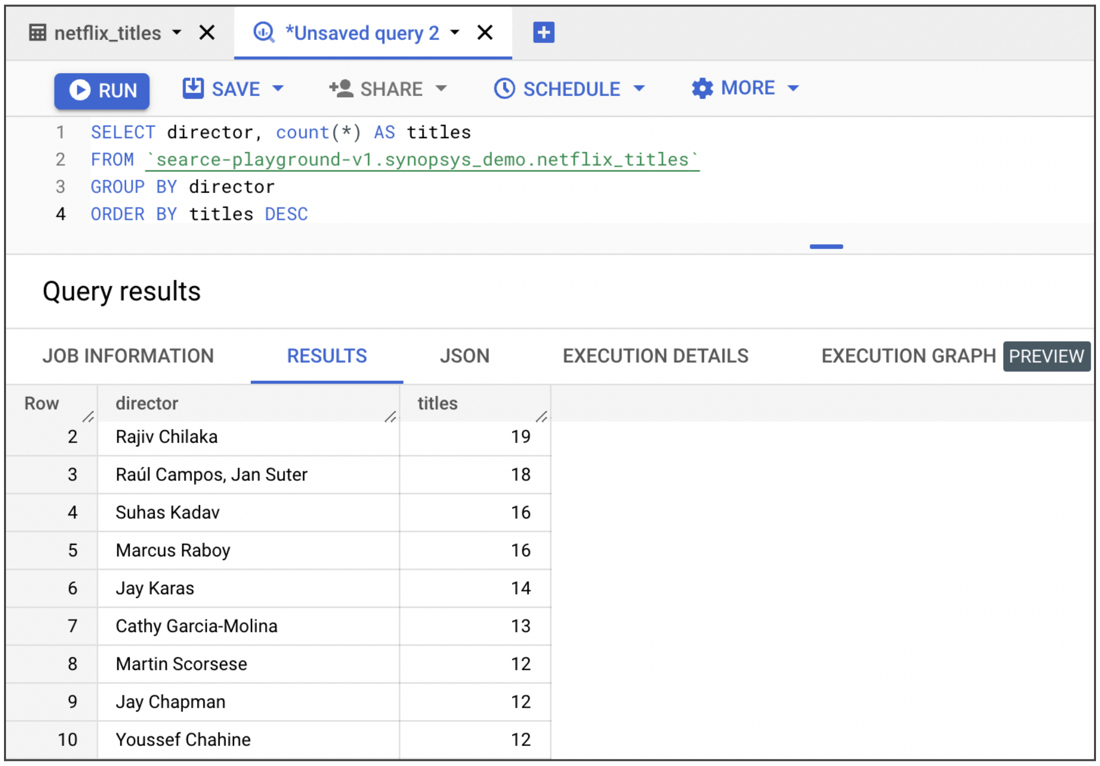
Row	f0_
1	1925

c. Number of titles from each director ordered from greatest to the least

Query

```
SELECT director, count(*) AS titles
FROM `<project_id>.synopsys_demo.netflix_titles`
GROUP BY director
ORDER BY titles DESC
```

Results



```
SELECT director, count(*) AS titles
FROM `searce-playground-v1.synopsys_demo.netflix_titles`
GROUP BY director
ORDER BY titles DESC
```

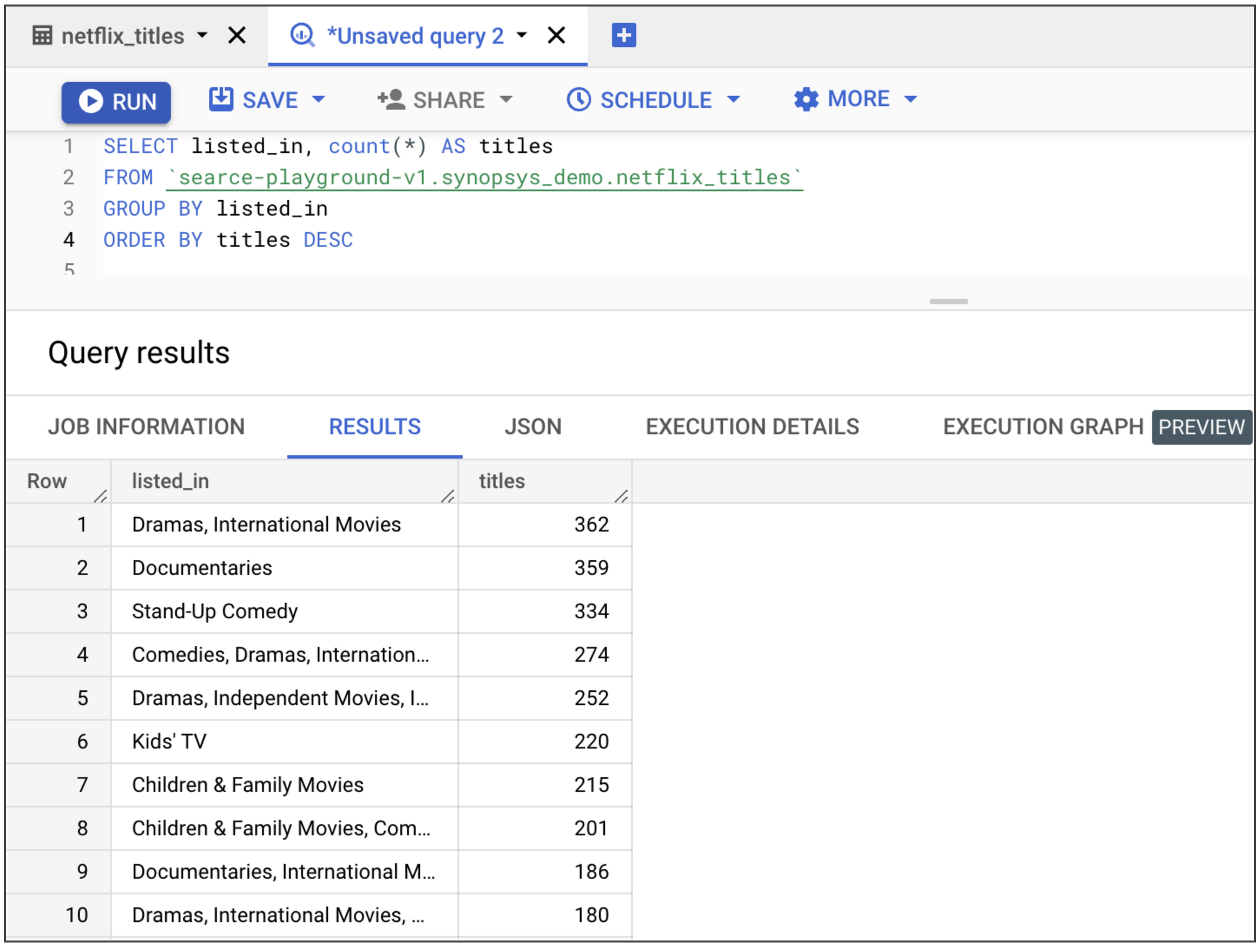
Row	director	titles
2	Rajiv Chilaka	19
3	Raúl Campos, Jan Suter	18
4	Suhas Kadav	16
5	Marcus Raboy	16
6	Jay Karas	14
7	Cathy Garcia-Molina	13
8	Martin Scorsese	12
9	Jay Chapman	12
10	Youssef Chahine	12

d. Title count for each category in listed_in ordered from highest to lowest

Query

```
SELECT listed_in, count(*) AS titles
FROM `<project_id>.synopsys_demo.netflix_titles`
GROUP BY listed_in
ORDER BY titles DESC
```

Results



```sql
SELECT listed_in, count(*) AS titles
FROM `searce-playground-v1.synopsys_demo.netflix_titles`
GROUP BY listed_in
ORDER BY titles DESC
```

Query results

Row	listed_in	titles
1	Dramas, International Movies	362
2	Documentaries	359
3	Stand-Up Comedy	334
4	Comedies, Dramas, Internation...	274
5	Dramas, Independent Movies, I...	252
6	Kids' TV	220
7	Children & Family Movies	215
8	Children & Family Movies, Com...	201
9	Documentaries, International M...	186
10	Dramas, International Movies, ...	180

Use Cases

The use cases most suited for BigQuery are the ones where the users need to perform interactive ad-hoc queries of read-only datasets.

Typically, BigQuery is used at the end of a Big Data ETL pipeline on top of processed data or in scenarios where complex analytical queries to a relational database take several seconds.

As it has a built-in cache, BigQuery works really well in cases where the data does not change often.

In addition, cases where datasets are fairly small don’t really benefit from BigQuery, with a simple query taking up to a few seconds. As such, it shouldn't be used as a regular OLTP (Online Transaction Processing) database.

BigQuery is meant and suited for OLAP (Online Analytics Processing) Analytics workloads.

Why BigQuery Transfer Service?

An alternative method to BQ Transfer Service for transferring data from GCS to BigQuery is using Cloud Dataflow.

Both methods have their advantages and disadvantages.

Method 1: Using BigQuery Data Transfer Service
Pros:

- The entire process is done on the console and no coding is involved.
- The prerequisites and IAM permissions required are minimal.
- The method is simple and straightforward. Instructions are easy to follow and less chances of getting unexpected errors.

Cons:

- Larger files (>100MB) need to be divided/sharded into multiple smaller ones.
- We cannot perform any transformations on the data before it is loaded into BQ.

Method 2: Using a Custom Dataflow Pipeline
Pros:

- Can handle large files with the appropriate configuration.
- Much more customisable.
- We can perform custom transformations on the data before it is loaded into BQ.

Cons:

- Requires many IAM permissions along with a Service Account.
- More prerequisites are required both in terms of knowledge and software packages
- Relatively more advanced and not beginner friendly, especially to someone who has no familiarity with Dataflow or Apache Beam.
- As it is a complicated process, there is a lot of room for unexpected errors and blockers to occur, which can be hard to navigate for someone who has no expertise in this.

For this tutorial we went with Method 1 because for our use case, the file size is relatively small and we did not have to make any transformations to the data before loading it to BigQuery

Best Practices

1. Use GCS as staging area before uploading to BigQuery

   In many use cases including this tutorial, it is recommended to use GCS as a staging area before transferring it to BigQuery. Here are the pros and cons of doing so.

   Pros

- Loading data is incredibly fast
- Possible to remove BQ table when not used and import it when needed (BQ table can be much bigger than even compressed data in Cloud Storage)
- You save your local storage space
- Less likely to fail during table creation (from local storage there could be networking issues, computer issues etc.)

Cons

- You pay some additional cost for storage (in the case you do not plan to access the data often e.g. once per month or once a year - you can reduce costs by using the nearline or coldline storage types in GCS)

2. Nested and Repeated Data

This is one of the most important Google BigQuery ETL best practices. Google BigQuery performs best when the data is denormalized. Instead of keeping relations, denormalize the data and take advantage of nested and repeated fields.

Nested and repeated fields are supported in Avro, Parquet, ORC, JSON (newline delimited) formats. STRUCT is the type that can be used to represent an object which can be nested and ARRAY is the type to be used for the repeated value.

3. Data Compression

The next vital Google BigQuery ETL best practice is on Data Compression. Most of the time the data will be compressed before transfer. You should consider the below points while compressing data.

- The binary Avro is the most efficient format for loading compressed data.
- Parquet and ORC format are also good as they can be loaded in parallel.
- For CSV and JSON, Google BigQuery can load uncompressed files significantly faster than compressed files because uncompressed files can be read in parallel.

Always try to convert large CSV or JSON files into file types such as Avro or Parquet before transferring to BigQuery. This will save significant time, resources, and costs.

4. Access Control and Data Encryption

Data stored in Google BigQuery is encrypted by default and keys are managed by GCP. Alternatively customers can manage keys using the Google KMS service.

To grant access to resources, BigQuery uses IAM(Identity and Access Management) to the dataset level. Tables and views are child resources of datasets and inherit permission from the dataset. There are predefined roles like bigquery.dataViewer and bigquery.dataEditor or the user can create custom roles.

Make sure all users are assigned the minimum required roles/permissions at a granular level.

5. Bulk Updates

   When you are making UPDATES to your BQ table, make them in bulk instead of multiple separate queries.

   BigQuery has quotas and limits for DML statements which are getting increased over time. As of now the limit of combined INSERT, UPDATE, DELETE and MERGE statements per day per table is 1,000.

   Note that this is not the number of rows. This is the number of the statements and as you know, one single DML statement can affect millions of rows.

   Now within this limit, you can run updates or merge statements affecting any number of rows. It will not affect any query performance, unlike many other analytical solutions.