**Final Project-ETL**

CIS 9440 - Data Warehousing for Analytics

Final Project Milestone 3

Group Number - 17

Student(s) - Yan San, Tung-Ting Wang, Dongdong Song

We add the subscription to Revenue fact table as a measure and delete the member dimension.

Currently, we have four fact tables: covid\_fact, rev\_sub\_fact, trend fact, and NFLX\_Stock\_Price\_Fact; and two conformed dimension tables: region\_dim and date\_dim.

We use python to extract, transform and load the data. The following are the step description for ETL processes:

1.The following codes explain how we extract the Covid-19 data from Google public data and set the extract function named Extract\_Covid\_Data.

### 1a. extract COVID data

def Extract\_Covid\_Data():

# SQL query to run in BigQuery to extract Hacker News story data

sql\_query = """

SELECT date, country\_name, SUM(new\_confirmed) as new\_confirmed

FROM `bigquery-public-data.covid19\_open\_data.covid19\_open\_data`

WHERE new\_confirmed IS NOT NULL

AND date BETWEEN '2020-01-01' AND '2021-04-19'

AND country\_name IN (SELECT country\_name FROM `bigquery-public-data.covid19\_open\_data.covid19\_open\_data` WHERE country\_name LIKE 'United States'

or country\_name LIKE 'Canada'

or country\_name LIKE 'United Kingdom'

or country\_name LIKE 'Ireland'

or country\_name LIKE 'Denmark'

or country\_name LIKE 'Finland'

or country\_name LIKE 'Norway'

or country\_name LIKE 'Sweden'

or country\_name LIKE 'Netherlands'

or country\_name LIKE 'France'

or country\_name LIKE 'Germany'

or country\_name LIKE 'South Africa'

or country\_name LIKE 'Brazil'

or country\_name LIKE 'Argentina'

or country\_name LIKE 'Chile'

or country\_name LIKE 'Colombia'

or country\_name LIKE 'Mexico'

or country\_name LIKE 'Australia'

or country\_name LIKE 'New Zealand'

or country\_name LIKE 'Japan'

or country\_name LIKE 'India'

or country\_name LIKE 'South Korea')

GROUP BY date, country\_name

ORDER BY country\_name , date

"""

#Those countries above are the most crucial members in their regions. Therefore, we select those countries’ names to represent the region.

# store extracted data in new dataframe

Covid\_df = client.query(sql\_query).to\_dataframe()

# validate that >0 stories have been extracted and return dataframe

if len(Covid\_df) > 0:

print(len(Covid\_df), "Covid-19 data is extracted")

return Covid\_df

else:

print("Extraction FAILED")

The following codes show how we clean the data which we extracted from Google public dataset.

### 1b.Clean COVID data

def Clean\_Covid\_Data(df):

# check if table exists in your database

try:

table\_id = 'CIS\_9440\_project.covid\_fact'

table = client.get\_table(table\_id)

print("Covid Fact table exists, number of rows: ", table.num\_rows)

print("now filtering for only new data")

# if the table is not already in the database, clean all data

except:

print("Covid Fact table is not in database, cleaning all data")

# drop rows with null values

if df.isnull().sum()[4] > 0:

df.dropna()

print(df.isnull().sum()[4], "null values dropped")

else:

print("Covid data has no null values")

# drop duplicate values

if len(df[df.duplicated()]) > 0:

df.duplicated(keep = 'first')

print(len(df[df.duplicated()])/2.0, "duplicate values dropped")

else:

print("Covid data has no duplicate rows")

print("Covid data cleaning successful")

return df

The following codes show how we create the Covid\_fact table and create a function called Create\_Covid\_Fact.

### 1c. Create COVID\_fact

def Create\_Covid\_Fact(df, region\_dim):

#create date\_id

df['date\_id'] = df['date'].apply(lambda x: x.strftime("%Y%m%d"))

# create region\_id column #left -> df's column #right->region\_dim's column, inner join

df = df.merge(region\_dim, left\_on='region\_name' , right\_on='region\_name',

how='inner')

# drop unneeded columns

for c in ['date','region\_name']:

df.drop(c, axis = 1, inplace=True)

return df

These are executing codes that are used to transfer the raw data to the normalized Covid fact table.

####.COVID related

#download covid dataset from bigquery

covid\_raw = dsf.Extract\_Covid\_Data();

#add new column

dsf.Append\_Region(covid\_raw)

### Create covid fact

covid\_fact = dsf.Create\_Covid\_Fact(covid\_transformed, region\_dim)

#upload the dataset to my bigquery, making it easier to manipulate

dsf.load\_df\_to\_bigquery(covid\_raw, 'covid\_append\_column')

#Using SQL to transform the data and downloading the dataset

covid\_transformed = dsf.Transform\_Country\_Region()

### add region column (since we don’t have a “region” column in the “covid\_raw” table.)

def Append\_Region(df):

df.insert(0, 'region\_name', 0)

covid\_region = []

L = 'Latin America'

US = 'United States and Canada'

EMA = 'Europe, Middle East, and Africa'

AP = 'Asia-Pacific'

for name in df['country\_name']:

if name == 'United States of America':

covid\_region.append(US)

elif name == 'Canada':

covid\_region.append(US)

elif name == 'United Kingdom' :

covid\_region.append(EMA)

elif name == 'Ireland':

covid\_region.append(EMA)

elif name =='Denmark':

covid\_region.append(EMA)

elif name =='Finland':

covid\_region.append(EMA)

elif name =='Norway':

covid\_region.append(EMA)

elif name =='Sweden' :

covid\_region.append(EMA)

elif name =='Netherlands':

covid\_region.append(EMA)

elif name =='France':

covid\_region.append(EMA)

elif name =='Germany' :

covid\_region.append(EMA)

elif name == 'South Africa':

covid\_region.append(EMA)

elif name == 'Argentina':

covid\_region.append(L)

elif name == 'Brazil':

covid\_region.append(L)

elif name == 'Chile':

covid\_region.append(L)

elif name =='Colombia' :

covid\_region.append(L)

elif name =='Mexico' :

covid\_region.append(L)

elif name == 'Australia':

covid\_region.append(AP)

elif name == 'New Zealand' :

covid\_region.append(AP)

elif name == 'Japan' :

covid\_region.append(AP)

elif name == 'India':

covid\_region.append(AP)

elif name == 'South Korea':

covid\_region.append(AP)

df['region\_name'] = covid\_region

### Transform the format “country” into “region” by selecting region\_name and grouping by region\_name

def Transform\_Country\_Region():

sql\_query = '''

SELECT date, region\_name, SUM(new\_confirmed) as new\_confirmed

FROM `baruch-cis.CIS\_9440\_project.covid\_append\_column`

GROUP BY date, region\_name

ORDER BY region\_name, date

'''

transform\_df = client.query(sql\_query).to\_dataframe()

if len(transform\_df) > 0:

print(len(transform\_df), "transformation data is extracted")

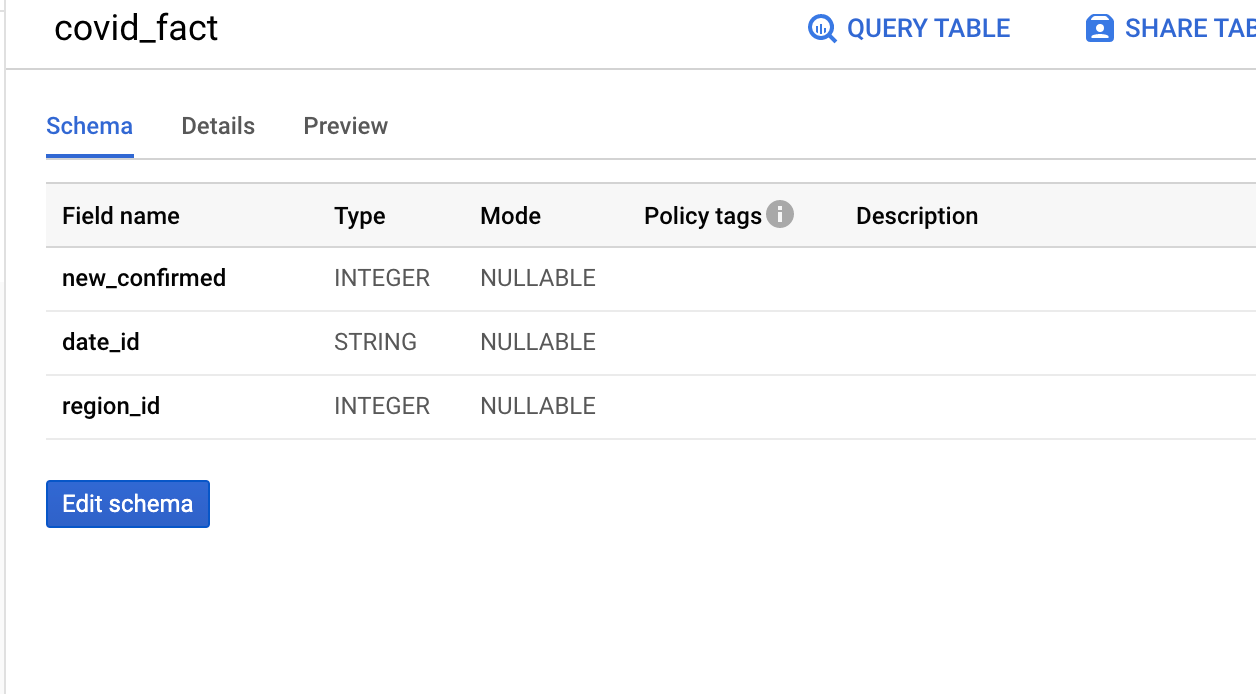
return transform\_df

else:

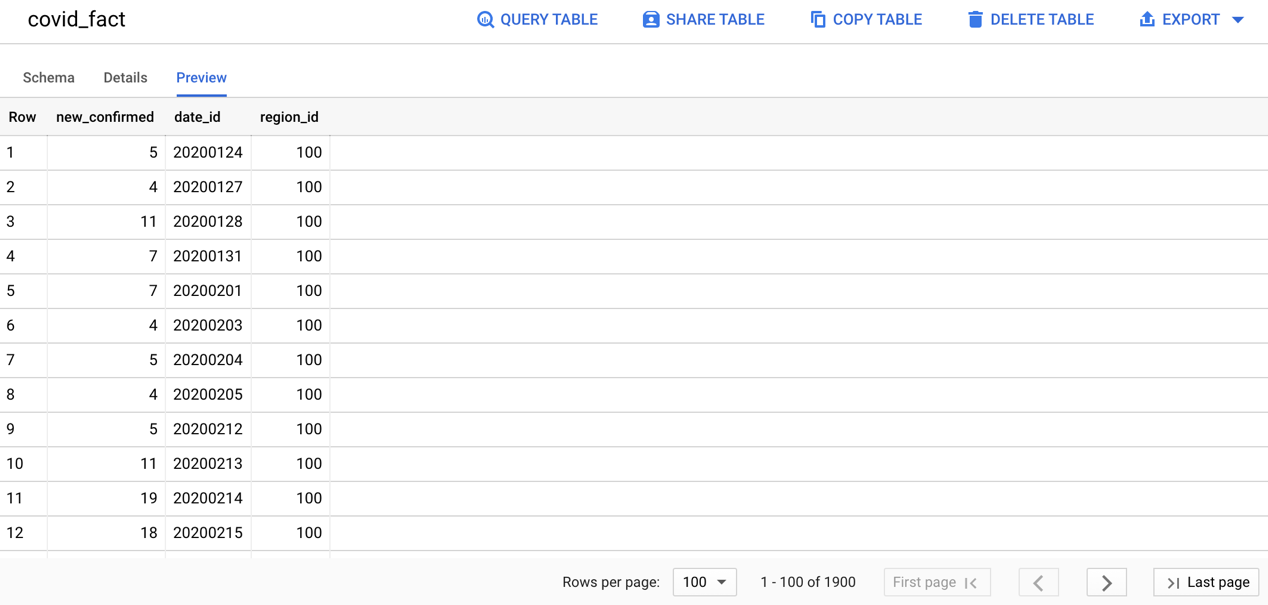
print("Transformation FAILED")

The screenshots are the Fact tables shown on the big query, including the covid\_fact schema and preview.

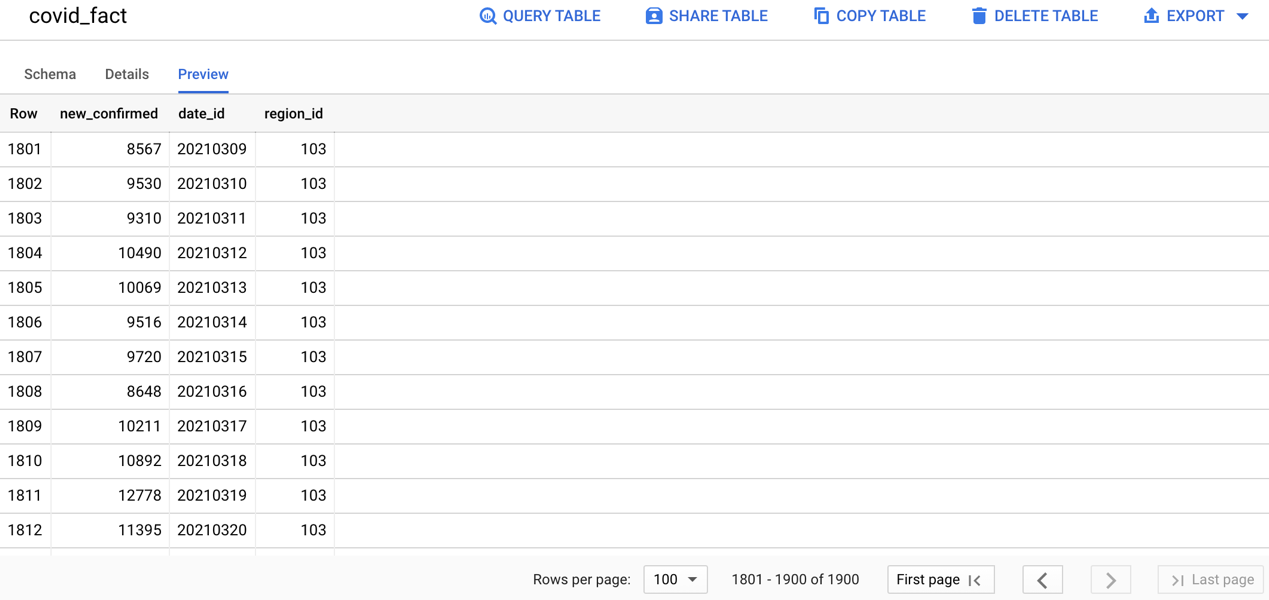
covid schema:



covid fact1:



covid fact2:



2. The following codes explain how we extract the Netflix data from its revenue and subscription file which we get from Netflix investor relations and set the extract function named Extract\_rev\_sub\_Data.

rev\_sub schema:

### 2a.extract rev\_sub

def Extract\_rev\_sub\_Data():

# SQL query to run in BigQuery to extract revenue and subscribers related data

sql\_query = """

select \*

from `baruch-cis.CIS\_9440\_project.sub\_rev\_2016\_2020`

order by region\_name;

"""

# store extracted data in new dataframe

rev\_sub\_df = client.query(sql\_query).to\_dataframe()

# validate that >0 stories have been extracted and return dataframe

if len(rev\_sub\_df) > 0:

print(len(rev\_sub\_df), "rev\_sub extracted")

return rev\_sub\_df

else:

print("rev\_sub extraction FAILED")

The following codes show how we create the ‘NFLX\_rev\_sub\_Fact’ fact table and create a function called Create\_NFLX\_rev\_sub\_Fact.

### 2b. Create rev Fact

def Create\_NFLX\_rev\_sub\_Fact(df, region\_dim):

# create date\_id column

new\_dateform=[] #transform string to date form

for c in df['date']:

new\_dateform.append(c)

df['date'] = new\_dateform

print(df['date'])

df['date\_id'] = df['date'].apply(lambda x: x.strftime("%Y%m%d"))

# create region\_id column #left -> df’s column #right->region\_dim’s column

df = df.merge(region\_dim, left\_on='region\_name' , right\_on='region\_name',

how='inner')

# drop unneeded columns

for c in ['fiscal\_year', 'fiscal\_qtr','date','region\_name']:

df.drop(c, axis = 1, inplace=True)

return df

These are executing codes which we used to extract data and transfer the raw data to the normalized ‘NFLX\_rev\_sub\_Fact’ fact table and then load it into the big query.

####. extract sub\_rev

rev\_sub = dsf.Extract\_rev\_sub\_Data()

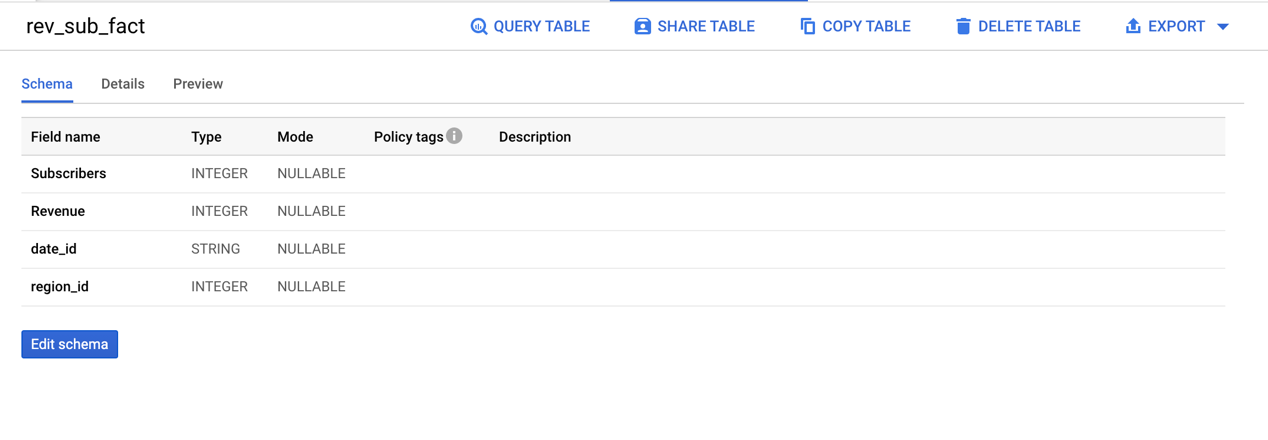
###.Create rev\_sub fact

rev\_sub\_fact = dsf.Create\_NFLX\_rev\_sub\_Fact(rev\_sub, region\_dim)

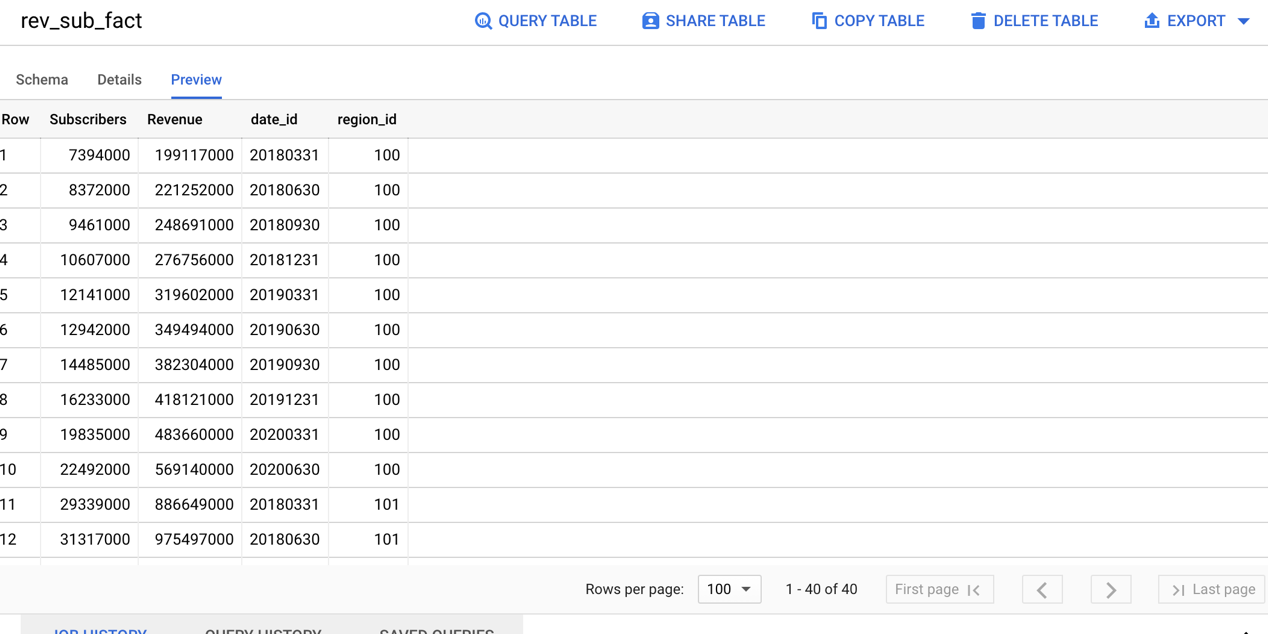
dsf.load\_df\_to\_bigquery(df= rev\_sub\_fact, table\_name= 'rev\_sub\_fact')

The screenshots are the Fact tables shown on the big query, including the Rev\_sub\_fact schema and preview.

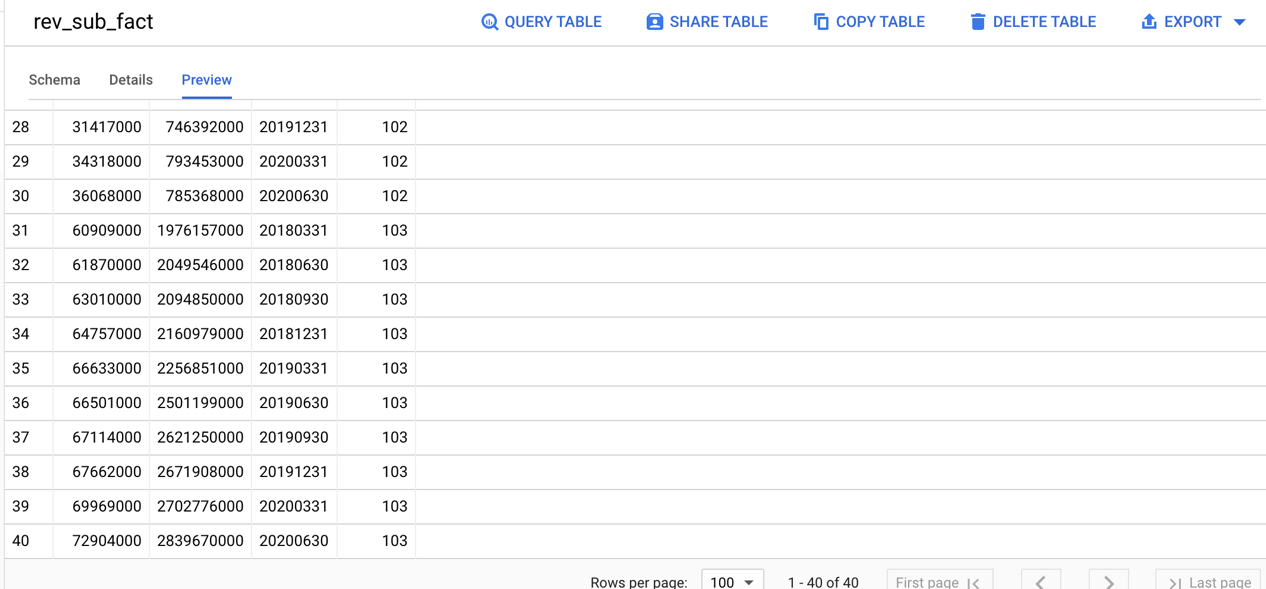
Rev\_sub\_fact schema



rev\_sub fact1:



rev\_sub fact2:



3. The following codes explain how we extract the Netflix trend data from Google trends ( we download the trend data manually and save them as CSV files in local devices) . Then we set the extract function named Extract\_Trend\_Data.

###3a.extract trend data

def Extract\_Trend\_Data():

# SQL query to run in BigQuery to extract Hacker News story data

sql\_query = """

SELECT Date, region, avg(netflix\_Trend) as popularity

FROM `baruch-cis.CIS\_9440\_project.trend\_5years`

GROUP BY Date, region

ORDER BY region, Date;

"""

# store extracted data in new dataframe

Google\_trend\_df = client.query(sql\_query).to\_dataframe()

# validate that >0 stories have been extracted and return dataframe

if len(Google\_trend\_df) > 0:

print(len(Google\_trend\_df), "trend extracted")

return Google\_trend\_df

else:

print("trend extraction FAILED")

The following code show the how we make trend\_Fact table and create a function called Create\_trend\_Fact

###3b.create trend fact

def Create\_trend\_Fact(df, region\_dim):

# create date\_id column

new\_dateform=[] #transform string to date frame

for c in df['Date']:

new\_dateform.append(c)

df['Date'] = new\_dateform

print(df['Date'])

df['date\_id'] = df['Date'].apply(lambda x: x.strftime("%Y%m%d"))

# create region\_id column #left -> df's column #right->region\_dim's column！

df = df.merge(region\_dim, left\_on='region' , right\_on='region\_name',

how='inner')

# drop unneeded columns

for c in ['Date','region']:

df.drop(c, axis = 1, inplace=True)

return df

These are executing codes which we used to extract data and transfer the raw data to the normalized trend\_fact fact table and then load to the big query.

###. extract trend data

trend\_region = dsf.Extract\_Trend\_Data()

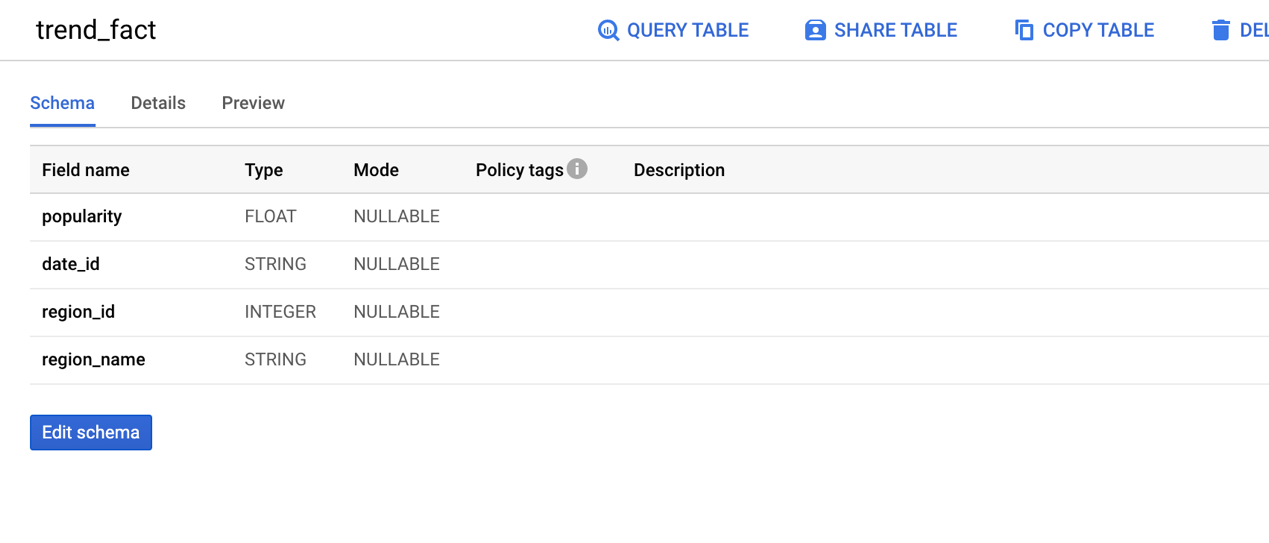
###. create trend fact

trend\_fact = dsf.Create\_trend\_Fact(trend\_region, region\_dim)

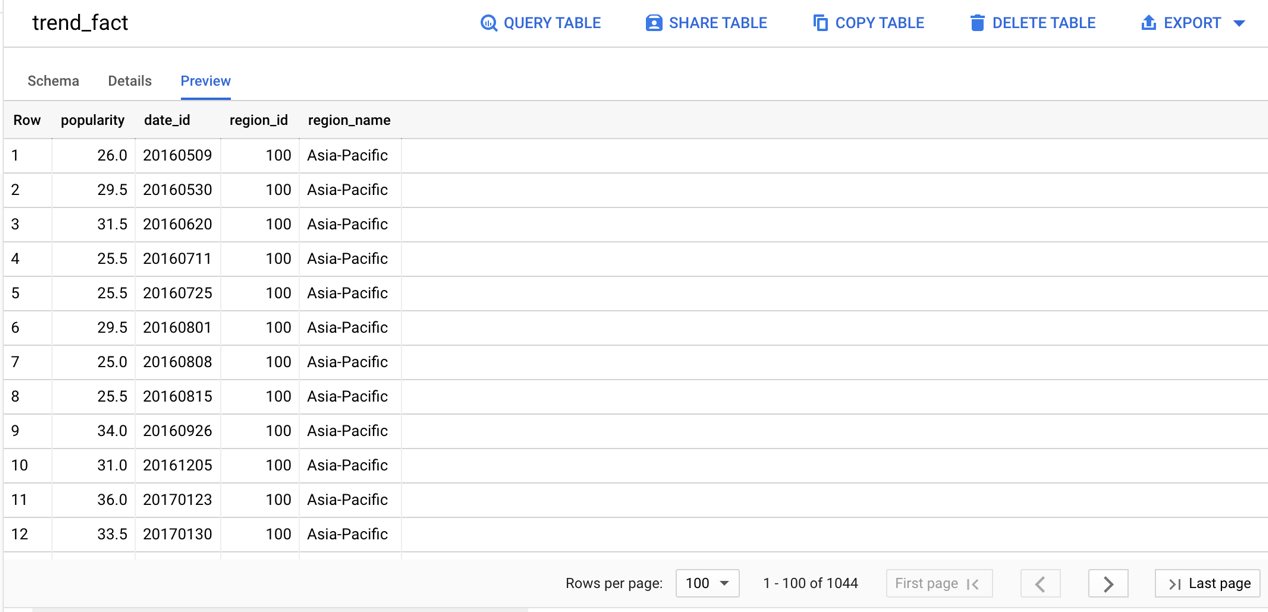
## load to big query

dsf.load\_df\_to\_bigquery(df= trend\_fact, table\_name= 'trend\_fact')

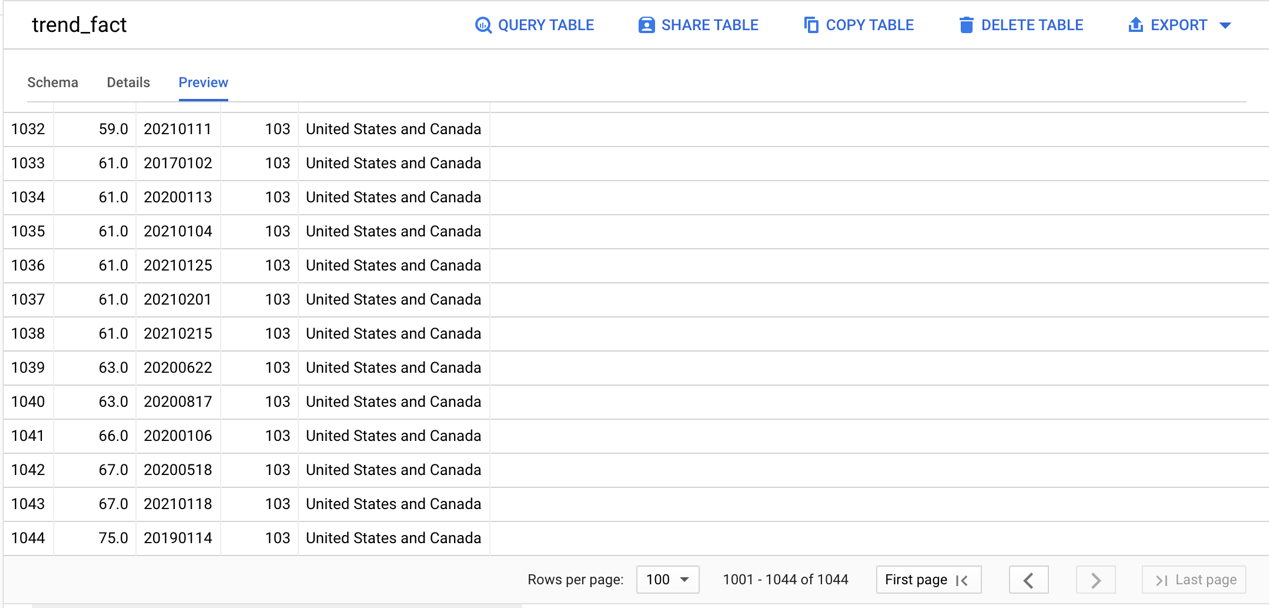
The screenshots are the Fact tables shown on the big query, including the trend\_fact schema and preview.



Trend fact1:



Trend fact 2 :



4. For this part, we use the yfinance package to export the Netflix stock price from yahoo finance from 2016 to 2021 and transfer the date to date\_id and extract the date\_id and Adj Close and save at the local.

### 4 Create stock fact

import pandas as pd

import numpy as np

import pandas\_datareader as web

import datetime

try:

import yfinance as yf

except:

print("installation unsuccessful")

def create\_stock\_df(stock, start, end):

# get the data from yahoo finance

df = yf.download(stock,

start=start,

end=end,

progress=False)

# add extra columns for day, stock title,

# simple moving average, and closing price average difference

df['day'] = range(1, len(df) + 1)

df['stock'] = stock

df = df.reset\_index()

return df

netflix\_2016\_df = create\_stock\_df("NFLX",

start = '2016-01-01',

end = '2021-04-23')

df= pd.DataFrame(netflix\_2016\_df)

df['date\_id'] = df['Date'].apply(lambda x: x.strftime("%Y%m%d"))

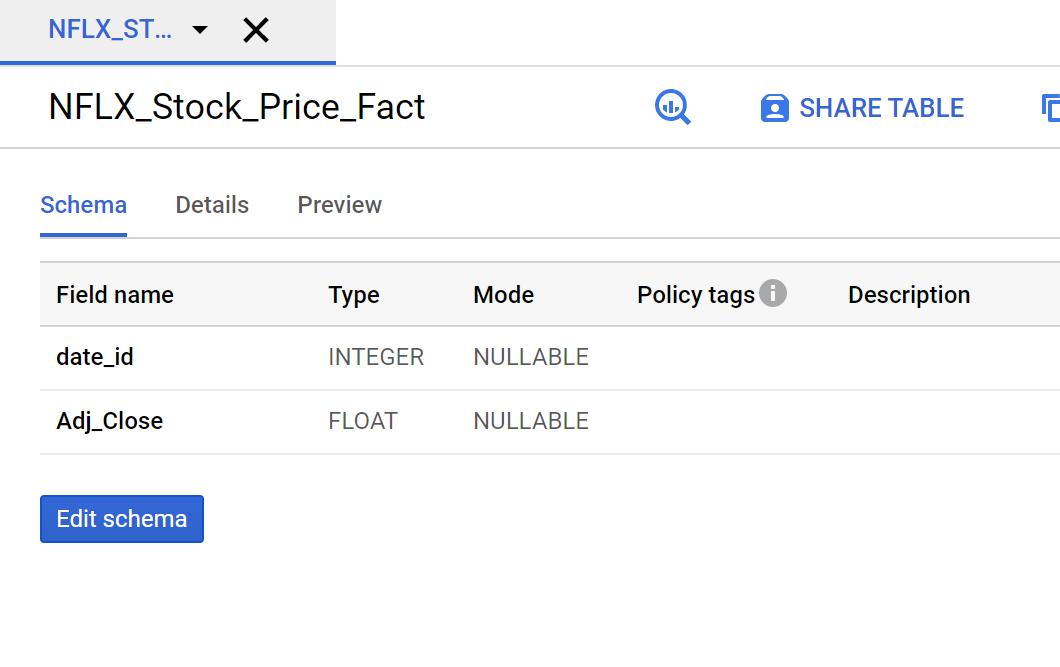
df2 = df[["date\_id", "Adj Close"]]

df2

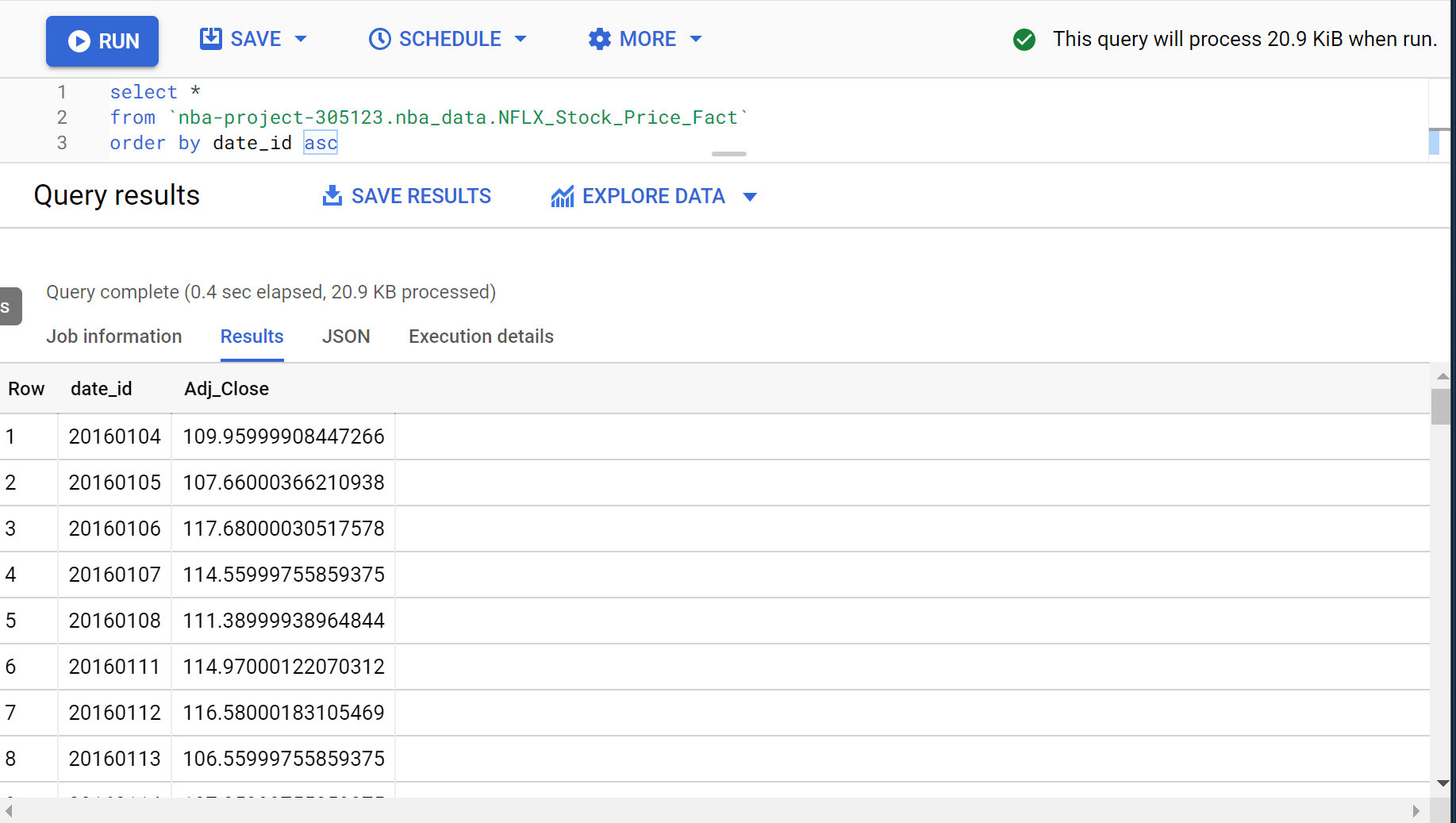
df2.to\_csv(r'C:\Users\Jeff Song\OneDrive\Desktop\Spring 2021 Course\CIS 9440 Data Warehousing\finalproject\NFLX\_Stock\_Price\_Fact.csv', index=False)

Then we upload the csv file Netflix\_stock\_price\_fact to big query manually and order the date by asc.

Stock Schema



Stock fact:



5. For this part we create the date dimension to match the date in fact tables up load to big query.

### 5.create date dimension

def create\_date\_dimension():

sql\_query = """

SELECT

CONCAT (FORMAT\_DATE("%Y",d),FORMAT\_DATE("%m",d),FORMAT\_DATE("%d",d)) as date\_id,

d AS full\_date,

EXTRACT(YEAR FROM d) AS year,

EXTRACT(WEEK FROM d) AS year\_week,

EXTRACT(DAY FROM d) AS year\_day,

EXTRACT(YEAR FROM d) AS fiscal\_year,

FORMAT\_DATE('%Q', d) as fiscal\_qtr,

EXTRACT(MONTH FROM d) AS month,

FORMAT\_DATE('%B', d) as month\_name,

FORMAT\_DATE('%w', d) AS week\_day,

FORMAT\_DATE('%A', d) AS day\_name,

FROM (

SELECT

\*

FROM

UNNEST(GENERATE\_DATE\_ARRAY('2016-01-01', '2021-05-01', INTERVAL 1 DAY)) AS d )

"""

# store extracted data in new dataframe

date\_df = client.query(sql\_query).to\_dataframe()

# validate that >0 stories have been extracted and return dataframe

if len(date\_df) > 0:

print("date dimension created")

return date\_df

else:

print("date dimension FAILED")

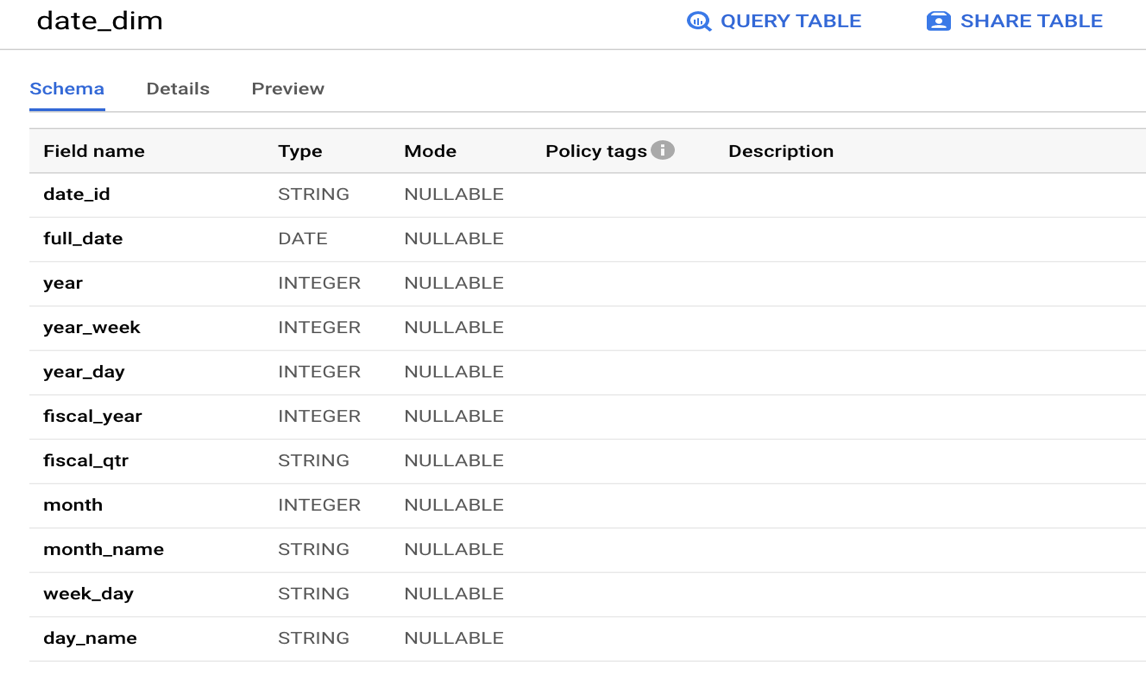
These are executing codes for creating the date dimension to big query

###5. create date\_dim

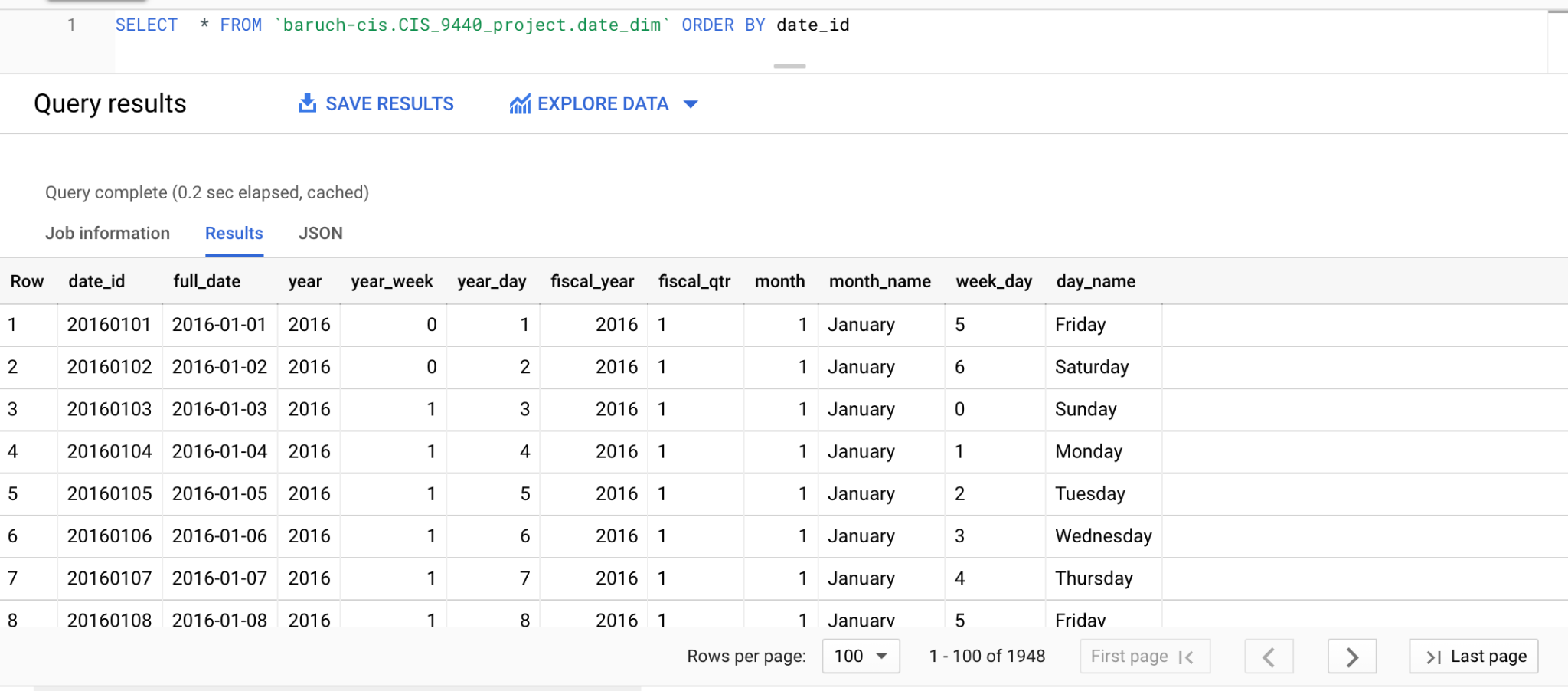
date\_dim = dsf.create\_date\_dimension()

dsf.load\_df\_to\_bigquery(df= date\_dim, table\_name= 'date\_dim')

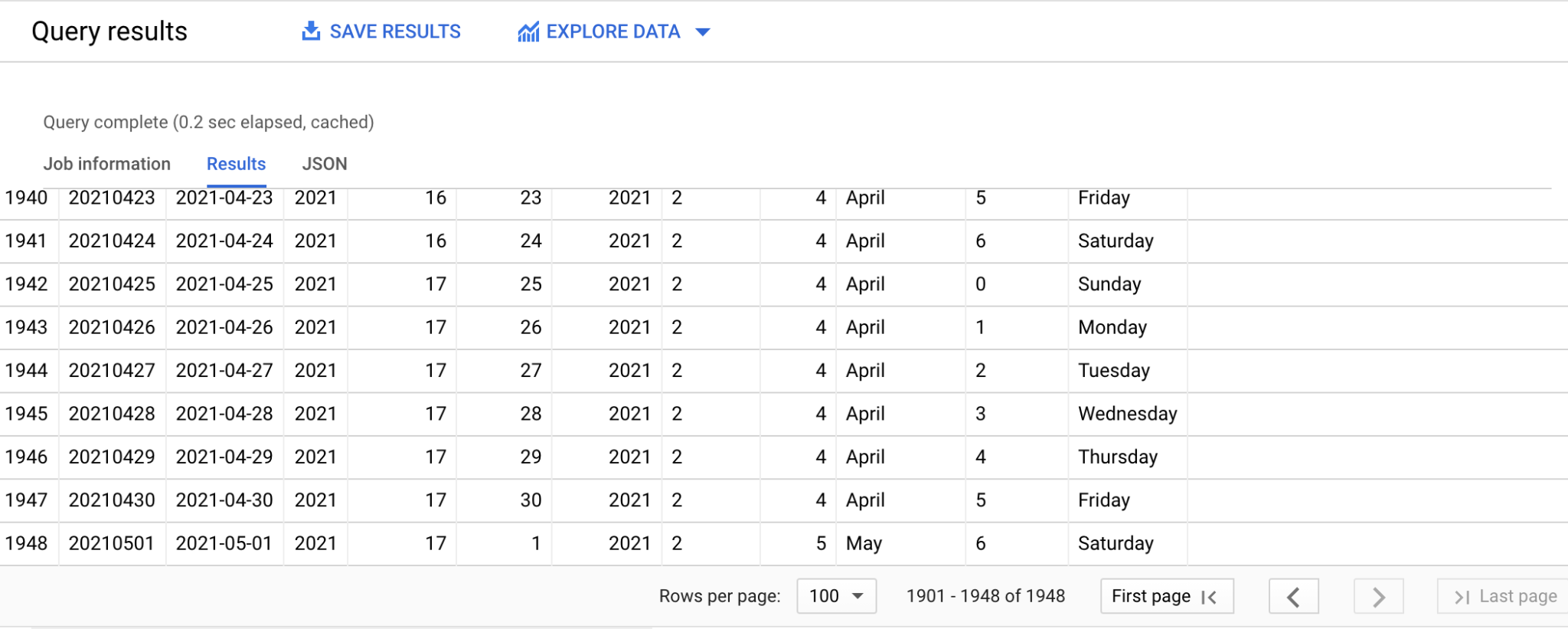
The screenshots are the Fact tables shown on the big query, including the date\_dim schema and preview.



Date dim 1:



Date dim 2:



6. The following codes are about region dim, since in real situations, the dataset Popularity trend and Covid 19 are only recorded by country, however, Netflix's revenue and subscription are documented by region. We have to make a function to transfer the country to the region in which they belong and set the region as a dimension table.

Follow the format of Netflix’s revenue and subscription dataset. We put the countries into four groups: Latin America, United and Canada; Europe, Middle East, and Africa; and Asia-Pacific.

The steps are shown as follow:

### add region column

def Append\_Region(df):

df.insert(0, 'region\_name', 0)

covid\_region = []

L = 'Latin America'

US = 'United States and Canada'

EMA = 'Europe, Middle East, and Africa'

AP = 'Asia-Pacific'

for name in df['country\_name']:

if name == 'United States of America':

covid\_region.append(US)

elif name == 'Canada':

covid\_region.append(US)

elif name == 'United Kingdom' :

covid\_region.append(EMA)

elif name == 'Ireland':

covid\_region.append(EMA)

elif name =='Denmark':

covid\_region.append(EMA)

elif name =='Finland':

covid\_region.append(EMA)

elif name =='Norway':

covid\_region.append(EMA)

elif name =='Sweden' :

covid\_region.append(EMA)

elif name =='Netherlands':

covid\_region.append(EMA)

elif name =='France':

covid\_region.append(EMA)

elif name =='Germany' :

covid\_region.append(EMA)

elif name == 'South Africa':

covid\_region.append(EMA)

elif name == 'Argentina':

covid\_region.append(L)

elif name == 'Brazil':

covid\_region.append(L)

elif name == 'Chile':

covid\_region.append(L)

elif name =='Colombia' :

covid\_region.append(L)

elif name =='Mexico' :

covid\_region.append(L)

elif name == 'Australia':

covid\_region.append(AP)

elif name == 'New Zealand' :

covid\_region.append(AP)

elif name == 'Japan' :

covid\_region.append(AP)

elif name == 'India':

covid\_region.append(AP)

elif name == 'Korea':

covid\_region.append(AP)

df['region\_name'] = covid\_region

### Transform country into region

def Transform\_Country\_Region():

sql\_query = '''

SELECT date, region\_name, SUM(new\_confirmed) as new\_confirmed

FROM `baruch-cis.CIS\_9440\_project.covid\_append\_column`

GROUP BY date, region\_name

ORDER BY region\_name, date

'''

transform\_df = client.query(sql\_query).to\_dataframe()

if len(transform\_df) > 0:

print(len(transform\_df), "transformation data is extracted")

return transform\_df

else:

print("Transformation FAILED")

Establish create\_region\_dimension function.

###6. Create\_region\_dimension

def create\_region\_dimension(covid\_df):

unique\_region = covid\_df['region\_name'].unique().tolist()

# create blank list of dimension rows

dimension\_rows = []

# create author dimension with a surrogate key

for region\_id, region\_name in enumerate(unique\_region, start = 100):

temp\_list = [region\_id, region\_name]

dimension\_rows.append(temp\_list)

region\_dim = pd.DataFrame(data=dimension\_rows,

columns = ['region\_id', 'region\_name'])

print("region dimension create")

print(dimension\_rows)

return region\_dim

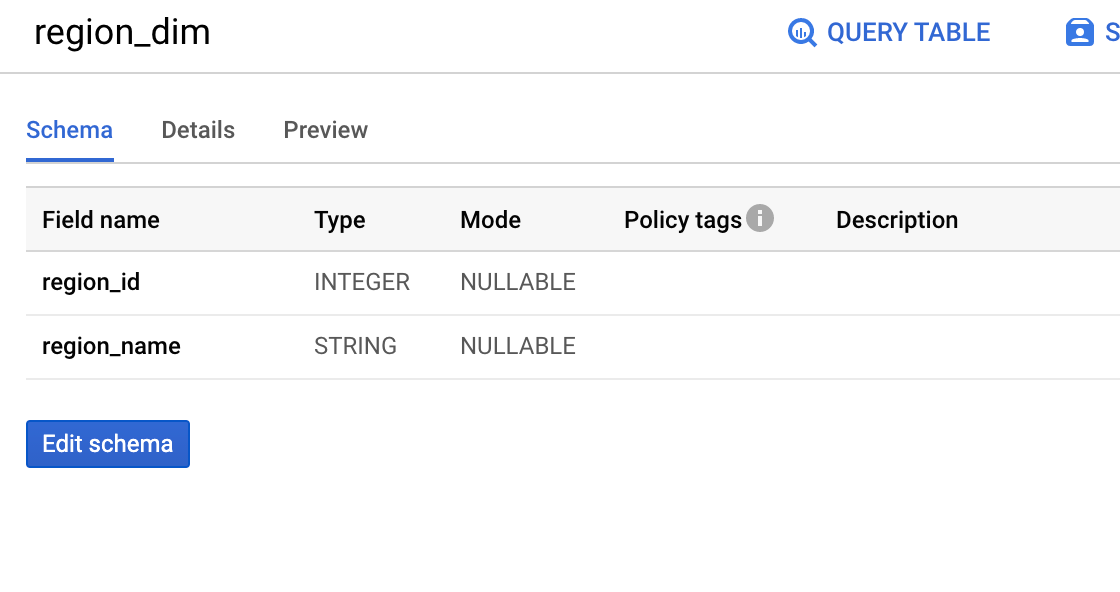
Create region\_dim and upload the table to big query.

###6. Create\_region\_dimension and upload to big query

region\_dim = dsf.create\_region\_dimension(covid\_transformed)

dsf.load\_df\_to\_bigquery(df= region\_dim, table\_name= 'region\_dim')

The following show the schema and data preview of region\_dim.



Region dim:



**Other Functions**

The function of loading the table to big query.

###.Load dim/fact to bigquery

def load\_df\_to\_bigquery(df, table\_name):

dataset\_id = '`baruch-cis.CIS\_9440\_project'

dataset\_ref = client.dataset(dataset\_id)

job\_config = bigquery.LoadJobConfig()

job\_config.autodetect = True

job\_config.write\_disposition = "WRITE\_TRUNCATE"

upload\_table\_name = 'CIS\_9440\_project.'+str(table\_name)

load\_job = client.load\_table\_from\_dataframe(df, upload\_table\_name,

job\_config=job\_config)

print("Starting job {}".format(load\_job))

We also export the tables to local.

#Export File

date\_dim.to\_excel(r'/Users/appleuser/Desktop/W4 Data Warehousing and Analytics9440/Final Project/milestone3-ETL\date\_dim.xlsx', index = False)

covid\_fact.to\_excel(r'/Users/appleuser/Desktop/W4 Data Warehousing and Analytics9440/Final Project/milestone3-ETL\covid\_fact.xlsx', index = False)

region\_dim.to\_excel(r'/Users/appleuser/Desktop/W4 Data Warehousing and Analytics9440/Final Project/milestone3-ETL\region\_dim.xlsx', index = False)

rev\_fact.to\_excel(r'/Users/appleuser/Desktop/W4 Data Warehousing and Analytics9440/Final Project/milestone3-ETL\rev\_fact.xlsx', index = False)

Another way to approach stock price data extraction and transformation.

def create\_stock\_df(stock, start, end):

# get the data from yahoo finance

df = yf.download(stock,

start=start,

end=end,

progress=False)

# add extra columns for day, stock title,

# simple moving average, and closing price average difference

df['day'] = range(1, len(df) + 1)

df['stock'] = stock

df = df.reset\_index()

return df

netflix\_2016\_df = create\_stock\_df("NFLX",

start = '2016-01-01',

end = '2021-04-23')

netflix\_2016\_df

netflix\_2016\_df.to\_csv(r'C:\Users\Colin\Desktop\stock.csv')#\_\_\_

def extract\_stock\_data():

# SQL query to run in BigQuery to extract stock data

sql\_query = """

select date, adj\_close

from \_\_\_

order by date;

"""

# store extracted data in new dataframe

stock\_df = client.query(sql\_query).to\_dataframe()

# validate that >0 stories have been extracted and return dataframe

if len(stock\_df) > 0:

print(len(stock\_df), "stock extracted")

return stock\_df

else:

print("stock extraction FAILED")

Executing code to load fact/dim to big query.

###.load fact/dim to bigquery

dsf.load\_df\_to\_bigquery(df= covid\_fact, table\_name= 'covid\_fact')

dsf.load\_df\_to\_bigquery(df= rev\_sub\_fact, table\_name= 'rev\_sub\_fact')

dsf.load\_df\_to\_bigquery(df= stock\_fact, table\_name= 'stock\_fact')

dsf.load\_df\_to\_bigquery(df= trend\_fact, table\_name= 'trend\_fact')

dsf.load\_df\_to\_bigquery(df= date\_dim, table\_name= 'date\_dim')

dsf.load\_df\_to\_bigquery(df= region\_dim, table\_name= 'region\_dim')