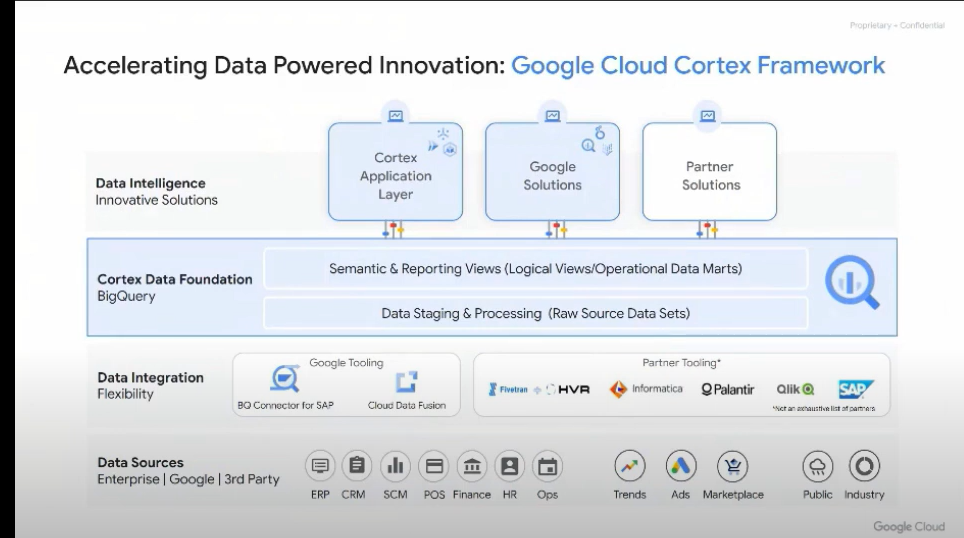
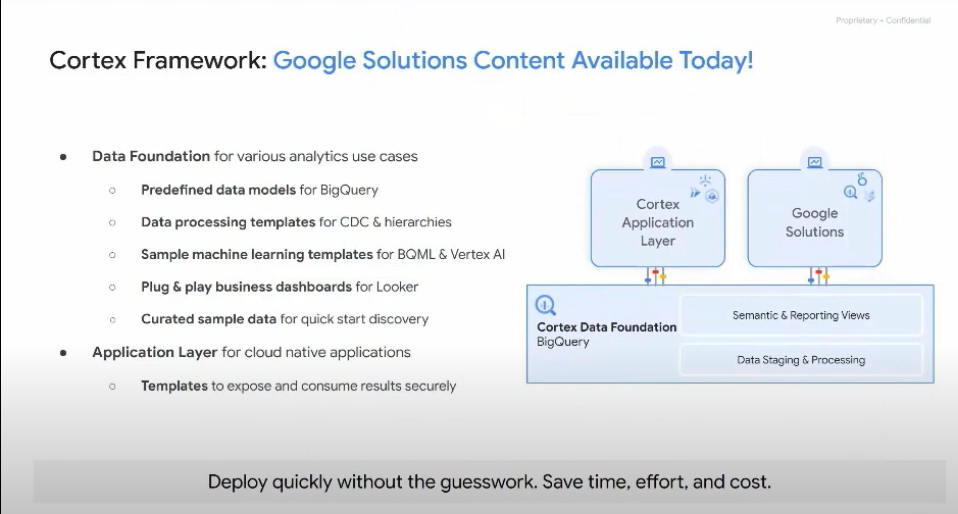
**Google Cloud Cortex Framework**

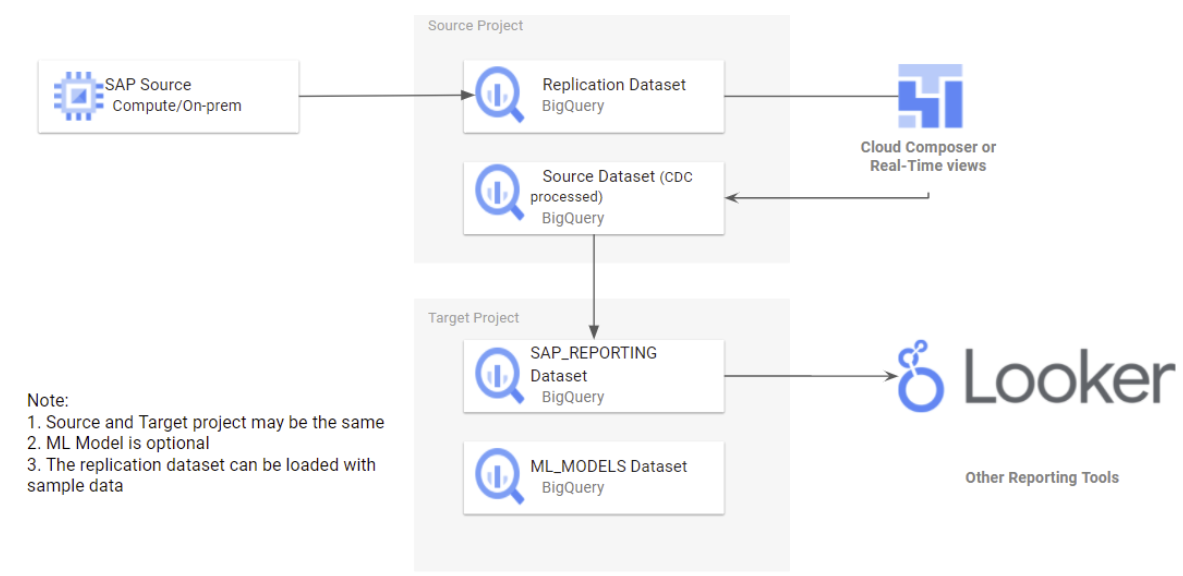
****

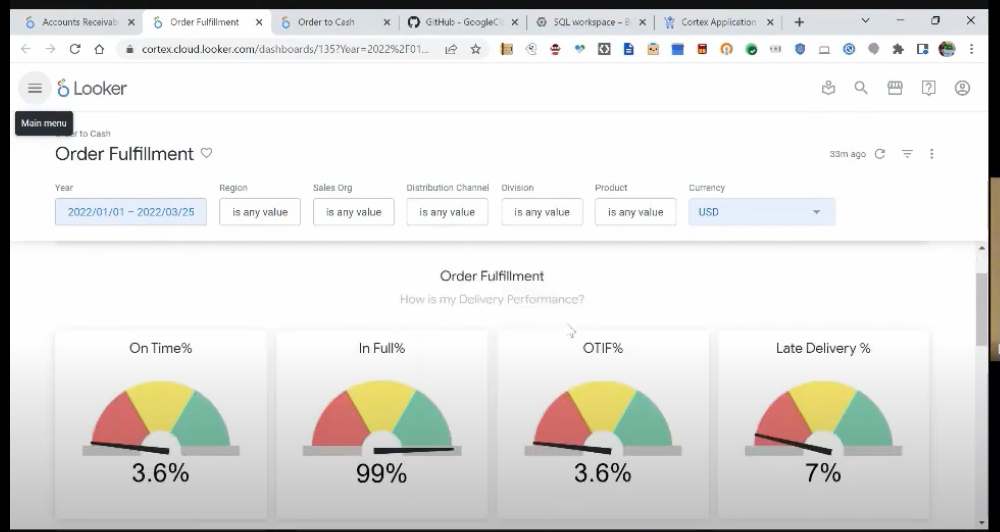
### **SAP connector:**

[**Choose your BigQuery Connector for SAP installation and configuration guide**](https://cloud.google.com/solutions/sap/docs/bq-connector/latest/install-config)

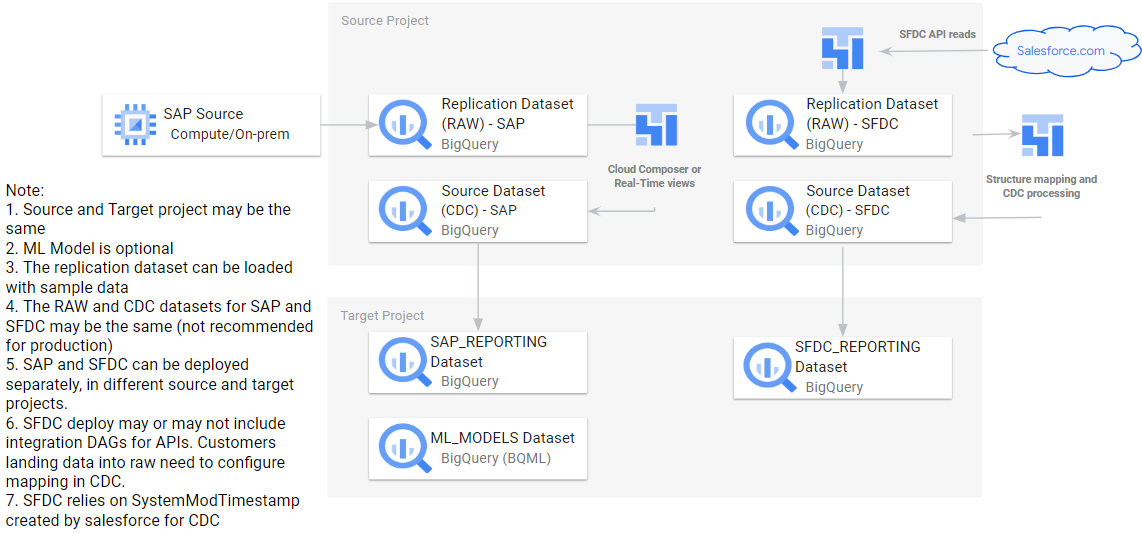
****

**SAP to BigQuery :-**

****

****

**SAP to BigQuery & SFDC to BigQuery :-**

****

**What your team should know :-**

1) General Google Cloud Platform (Cloud Shell, IAM, Cloud Build).

2) General Data and Analytics (BigQuery, Cloud Composer or Airflow, SQL).

3) General SAP concepts.

**Deployment Prerequisites :-**

**Storage buckets :-**

- New bucket for DAGs

- Separate bucket for logs

**Permissions : -**

**-** With own account :

- Service Usage Consumer

- Storage Object Viewer for the Cloud Build bucket and Writer to the right output buckets.

- Cloud Build Editor

- Project Viewer or Storage Object Viewer

**-** Cloud Build :

- Access to output buckets

- BigQuery Editor and Job User.

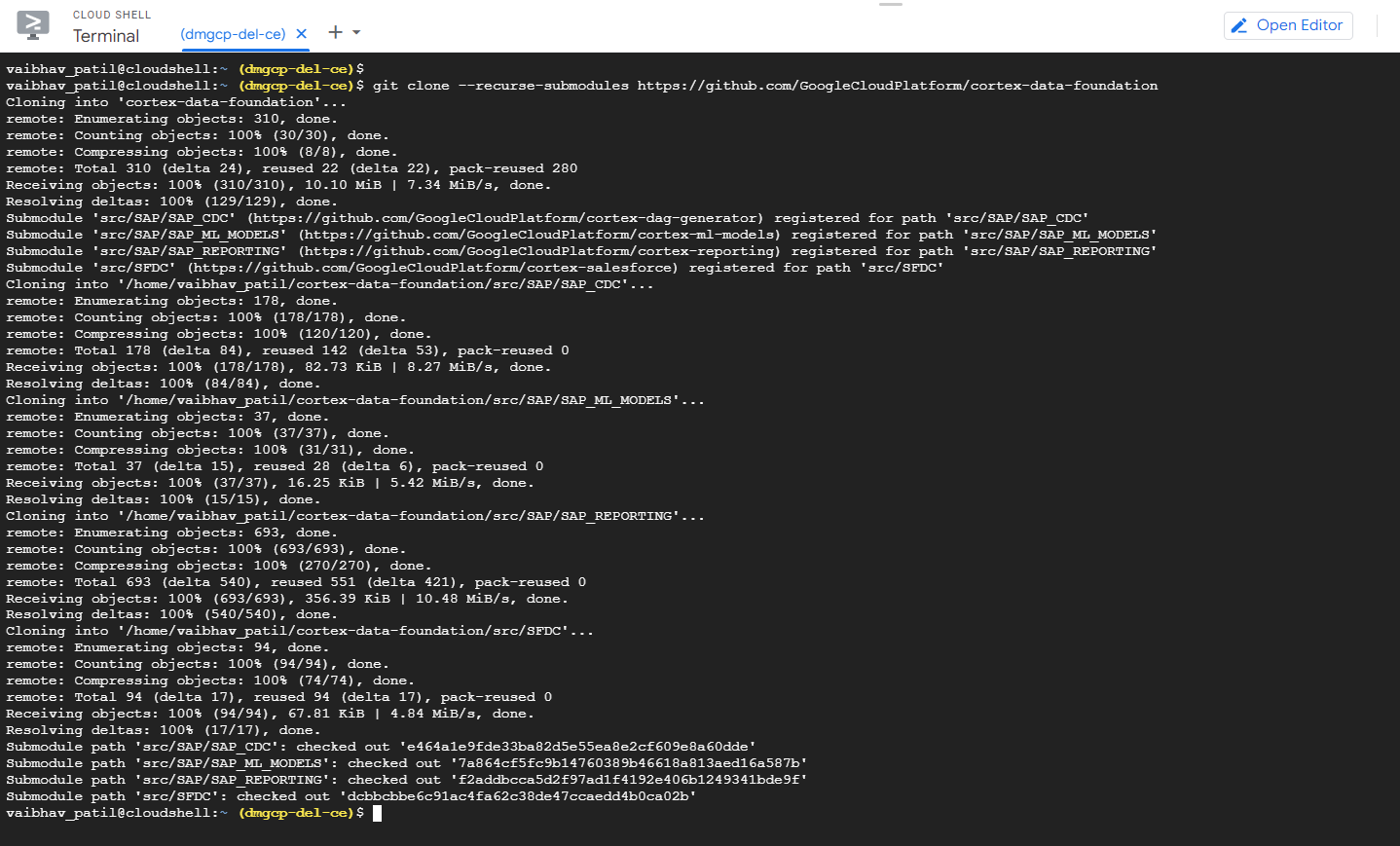
# 

# **Steps to setup Cortex Framework :-**

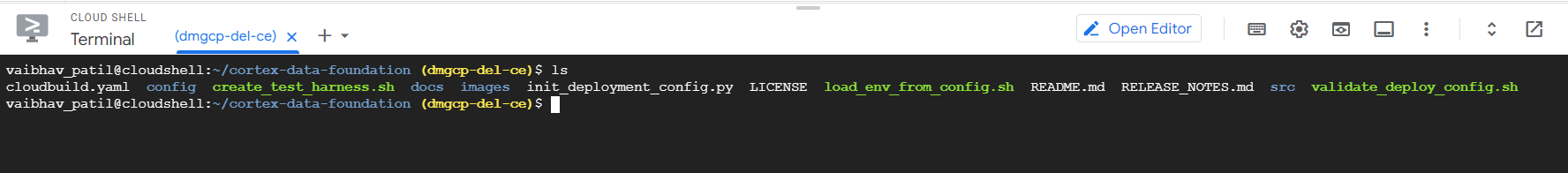
**Step 1:-** Open Cloud shell and run the command below to clone this repository.

**{** git clone --recurse-submodules <https://github.com/GoogleCloudPlatform/cortex-data-foundation> **}**

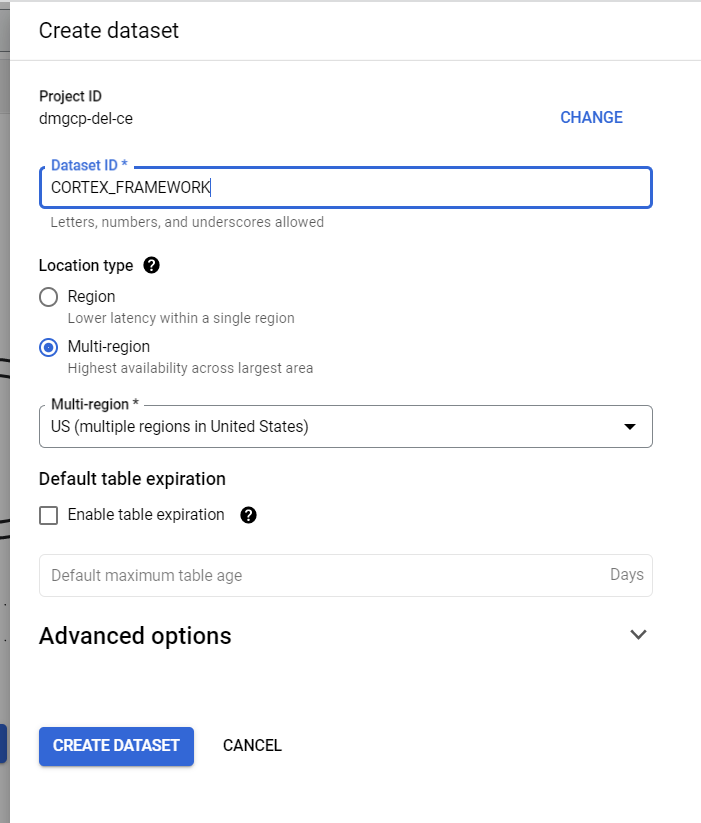




Once the command runs, see the below snap **cortex-data-foundation** directory.

This directory has all of the framework codes.

**Step 2:-** Create a dataset in Bigquery by passing the dataset name and region. ( Make sure while creating dataset region must be same on source and target side)



Total four datasets we will need to create RAW data ,CDC data ,Reporting view and ML Model dataset.

**Step 3:-** Run the below command by passing the parameters.

gcloud builds submit **--project** <execution project, likely the source> \

--substitutions \

**\_PJID\_SRC=**<project for landing raw data>**,\_PJID\_TGT=**<project to deploy user-facing views>**,\_DS\_CDC=**<BQ dataset to land the result of CDC processing - must exist before deployment>**,\_DS\_RAW=**<BQ dataset to land raw data from replication - must exist before deployment>**,\_DS\_REPORTING=**<BQ dataset where Reporting views are created, will be created if it does not exist>**,\_DS\_MODELS=**<BQ dataset where ML views are created, will be created if it does not exist>**,\_GCS\_BUCKET=**<Bucket for logs - Cloud Build Service Account needs access to write here>**,\_TGT\_BUCKET=**<Bucket for DAG scripts - don’t use the actual Airflow bucket - Cloud Build Service Account needs access to write here>**,\_TEST\_DATA=true,\_DEPLOY\_CDC=true,\_GEN\_EXT=true**

**Parameters :-**

**\_PJID\_SRC:-** Project where the source dataset is and the build will run.

**\_PJID TGT:-** Target project for user-facing datasets (reporting and ML datasets) -

**\_DS\_CDC:-** Dataset that works as a source for the reporting views, and target for the records processed DAGS. If using test data, create an empty dataset.

**\_DS RAW:-**: Used by the CDC process, this is where the replication tool lands the data

from SAP. If using test data, create an empty dataset.

**\_DS\_REPORTING:-** Name of the dataset that is accessible to end users for reporting,

where views and user-facing tables are deployed

**-DS\_ML**:- Name of the dataset that stages results of Machine Learning algorithms or

BQML models.

**\_GCS BUCKET:-** Bucket for logs.

**\_TGT\_BUCKET:-** Bucket where DAGS will be generated for the customer to copy.

**\_TEST\_DATA:-** Set to true if the customer wants the DATASET\_REPL to be filled by

Avoid using the actual airflow bucket. tables with sample data, mimicking a replicated dataset.

\_**DEPLOY\_CDC:-** Generate the DAG files into a target bucket based on the tables in

settings.yaml. If using test data, set to true.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Optional Parameters**

**\_LOCATION:-** Location where the BigQuery dataset and GCS buckets are (Options: US, ASIA or EU)

**\_MANDT:-** Default mandant or client for SAP. For test data, keep the default value.

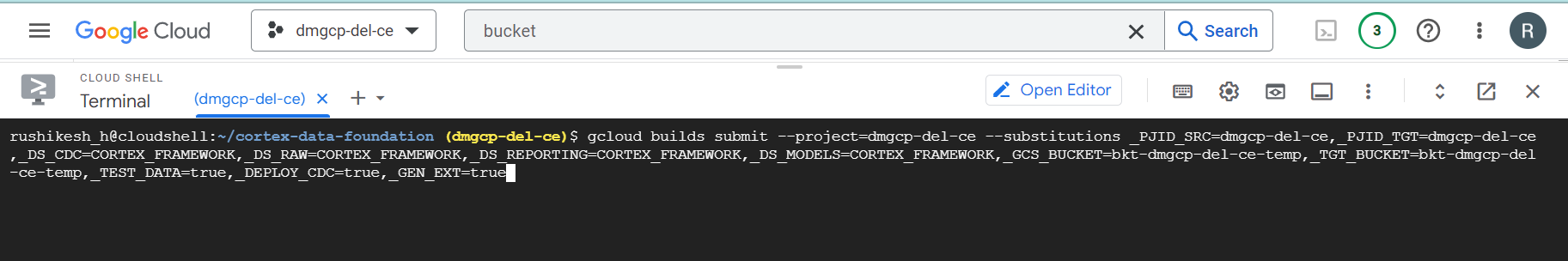
**\_sql-flavour:-** S4 or ECC. See the documentation for options. For test data, keep the default value.

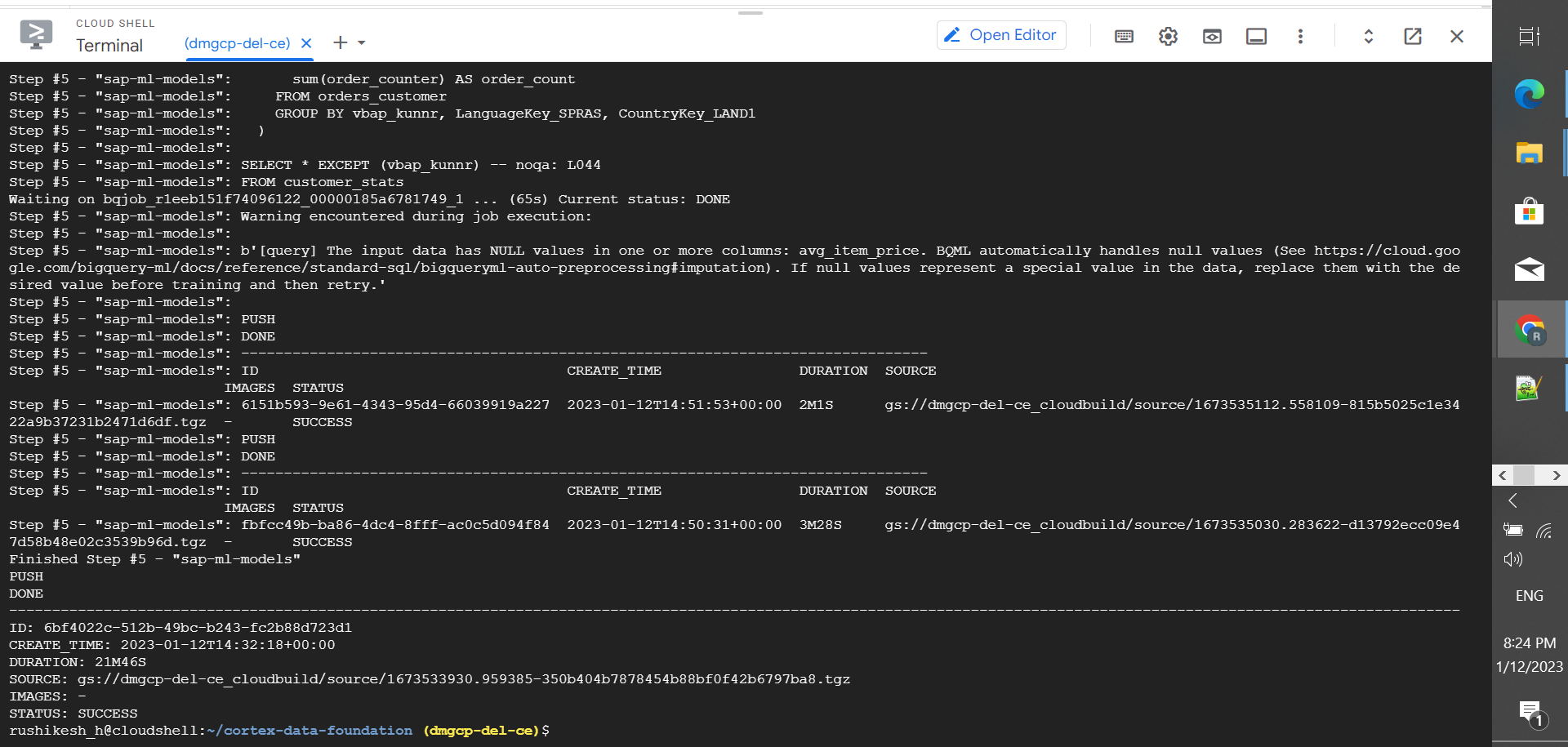
**Run the command for deployment:-**

gcloud builds submit --project=dmgcp-del-ce --substitutions \_PJID\_SRC=dmgcp-del-ce,\_PJID\_TGT=dmgcp-del-ce,\_DS\_CDC=CDC\_DATASET,\_DS\_RAW=RAW\_DATASET,\_DS\_REPORTING=REPORTING\_DATAEST,\_DS\_MODELS=ML\_MODEL\_DATASET,\_GCS\_BUCKET=BUCKET\_TO\_STORE\_LOG,\_TGT\_BUCKET=BUCKET\_TO\_STORE\_DAG,\_TEST\_DATA=true,\_DEPLOY\_CDC=true,\_GEN\_EXT=true

Recently used command:

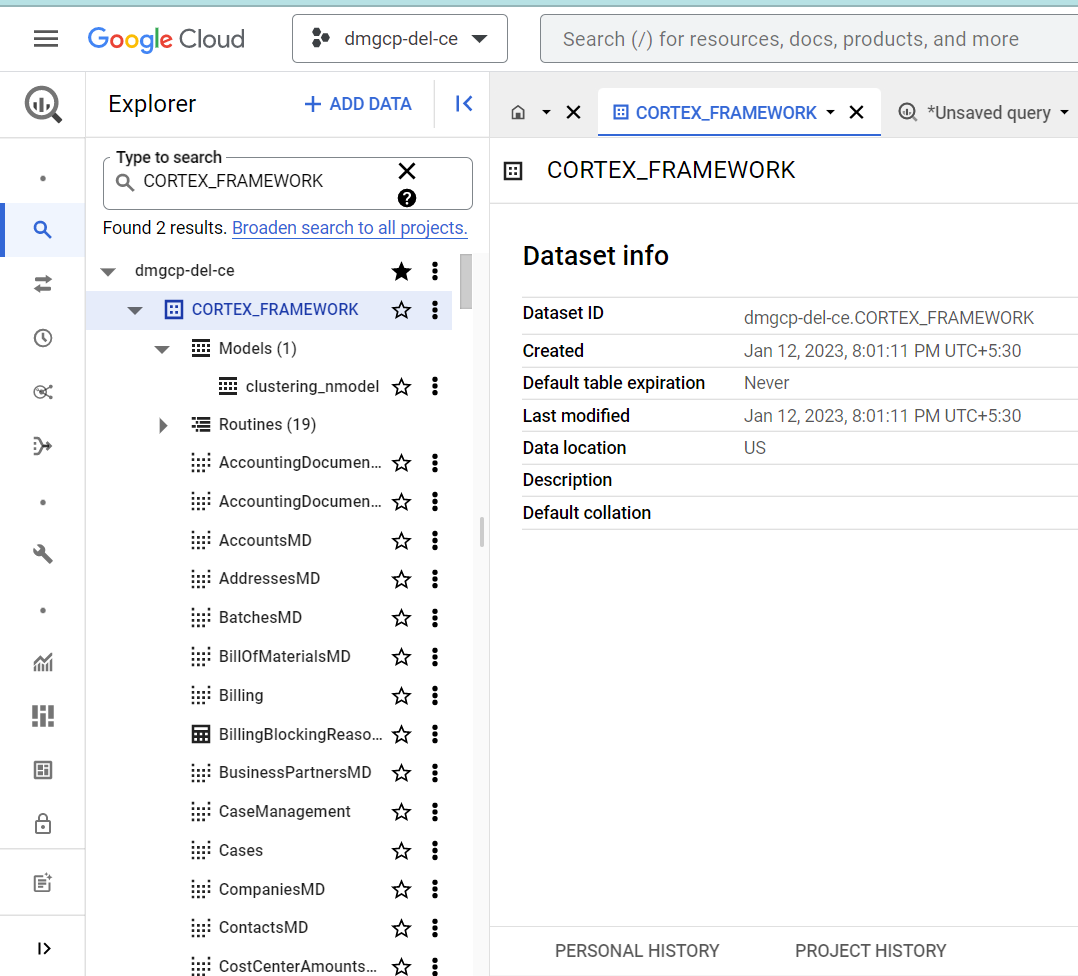
gcloud builds submit --project=dmgcp-del-ce --substitutions \_LOCATION=us-central1,\_MANDT=210,\_PJID\_SRC=dmgcp-del-ce,\_PJID\_TGT=dmgcp-del-ce,\_DS\_CDC=NEW\_CORTEX\_FRAMEWORK\_TEST\_CDC,\_DS\_MODELS=NEW\_CORTEX\_FRAMEWORK\_TEST\_ML,\_DS\_RAW=NEW\_CORTEX\_FRAMEWORK\_TEST\_RAW,\_DS\_REPORTING=NEW\_CORTEX\_FRAMEWORK\_TEST\_VIEW,\_SQL\_FLAVOUR=s4,\_GCS\_BUCKET=a0de9ffb-9a67-439c-ac9e-bdbc813ab010,\_TGT\_BUCKET=e2413a6b-44bd-4e02-b200-78c5de722a6a,\_DEPLOY\_SAP=true,\_DEPLOY\_SFDC=false,\_TEST\_DATA=true,\_GEN\_EXT=true





**Step 4:-**

After the execution in bigquery dataset tables ,views and functions are created.



**Note:-** For more information go through the [link](https://github.com/GoogleCloudPlatform/cortex-data-foundation) .

**If we are deployed to the cortex framework,we need to make some changes as per requirement .**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* **If we are not using test data we need some configurations. (\_TEST\_DATA=false )**

**Step 1:- DD03L for SAP metadata**

If you are not planning on deploying test data, and if you are planning on generating CDC DAG scripts during deployment, make sure table **DD03L** is replicated from SAP in the source project.

This table contains metadata about tables, like the list of keys, and is needed for the CDC generator and dependency resolver to work. This table will also allow you to add tables not currently covered by the model to generated CDC scripts, like custom or Z tables.

### **Step 2:-**

### Need this configuration if test data is false and need External Public Tables.

* Navigate to [BigQuery > Analytics Hub](https://console.cloud.google.com/bigquery/analytics-hub)
* Click Search Listings. Search for "**NOAA Global Forecast System**"
* Click Add dataset to project. When prompted, keep "**noaa\_global\_forecast\_system**" as the name of the dataset. If needed, adjust the name of the dataset and table in the FROM clauses in **weather\_daily.sql**.
* Repeat the listing search for Dataset "**OpenStreetMap Public Dataset**".
* Adjust the FROM clauses containing **bigquery-public-data.geo\_openstreetmap.planet\_layers** in **postcode.sql**.

**Note :-** During deployment, you can choose to merge changes in real time using a view in BigQuery or scheduling a merge operation in Cloud Composer (or any other instance of Apache Airflow).

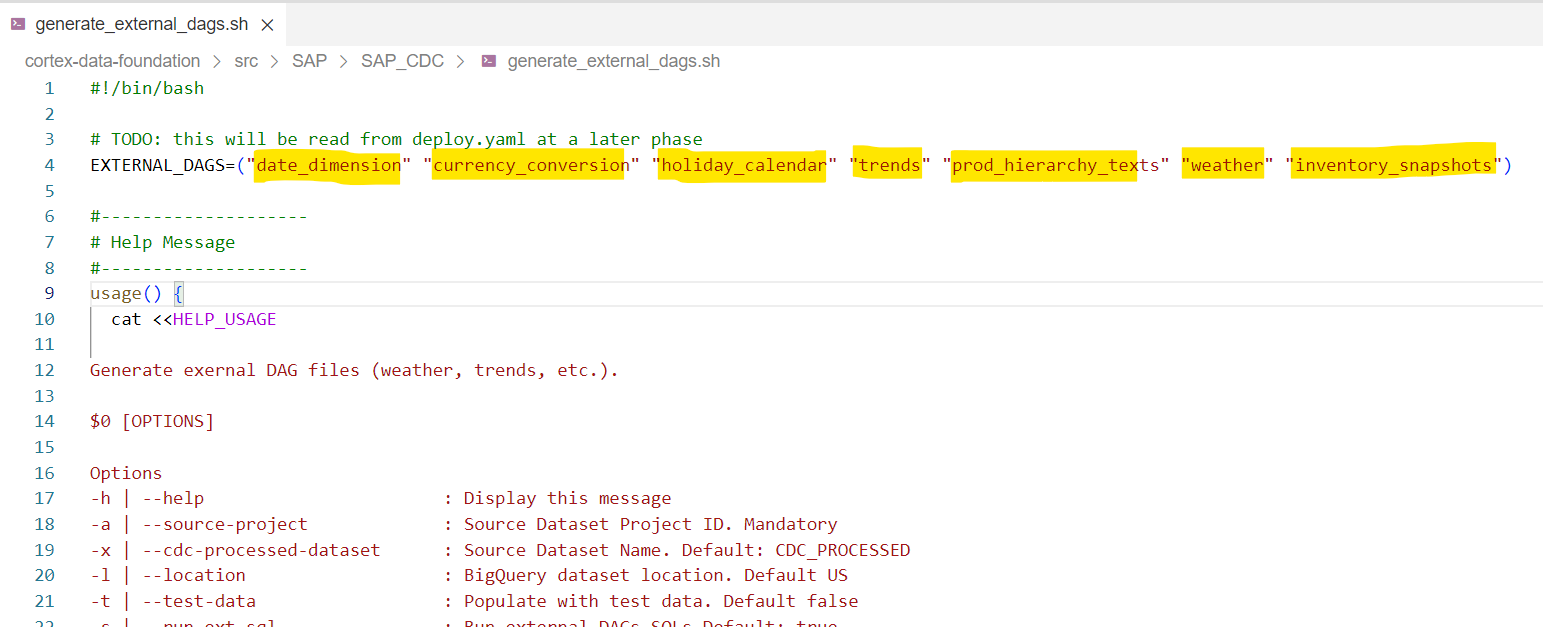
Cloud Composer can schedule the scripts to process the merge operations periodically. Data is updated to its latest version every time the merge operations execute, however, more frequent merge operations translate into higher costs.

The scheduled frequency can be customized to fit the business needs.

### **Step 3:-** Do the some changes in **generate\_external\_dags.sh**

### Path : cortex-data-foundation/src/SAP/SAP\_CDC

This file will create the external table and dags . Keep only those external dags sql that are required.



* **If we don’t need all tables :-**

**Steps 1:-**

We find the list of tables in [**setting.yaml**](https://github.com/GoogleCloudPlatform/cortex-dag-generator/blob/main/setting.yaml) file. If we do not have all of the required tables while replicating, only the views that depend on missing tables will fail to deploy.

File Path :- /src/SAP/SAP\_CDC

**Steps 2:-**

Changing the dependencies\*.txt file ,means keeping only those sql files that are required.

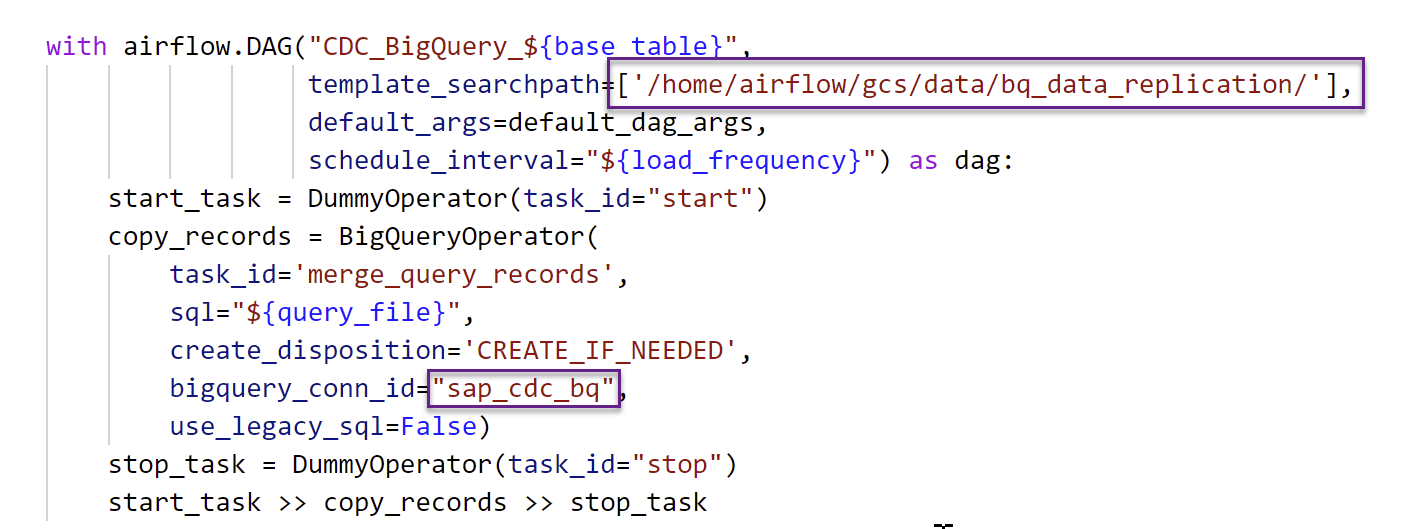
File Path :- /src/SAP/SAP\_REPORTING

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* **Changes in dag file :-**

In the dag file we need to change where we store our sql file and dag (path) and bigquery connection id .

File Path :- /src/SAP/SAP\_CDC/src/template\_dag/dag\_sql.py



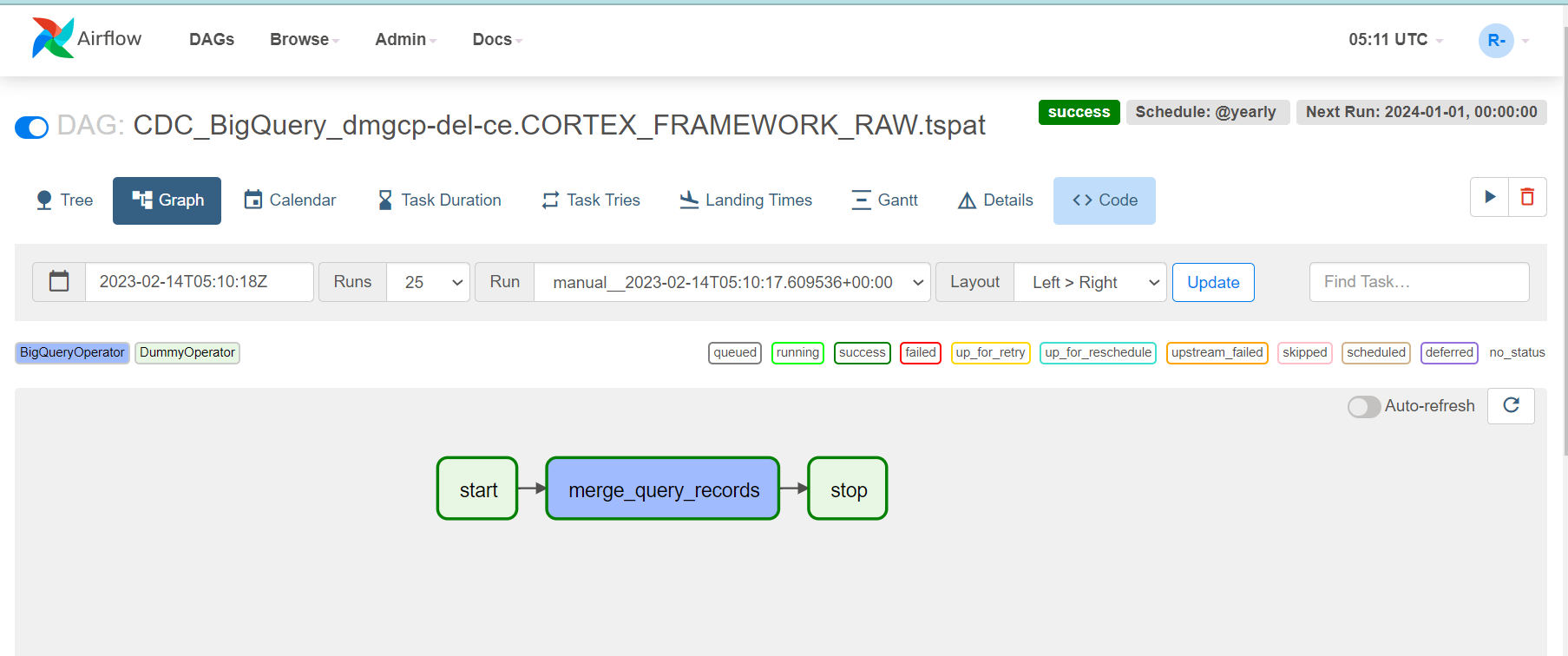
**Note :-** Do not use composer bucket to store,first store the file in another location then after move the files in composer bucket.

Move the files into the DAG bucket:-

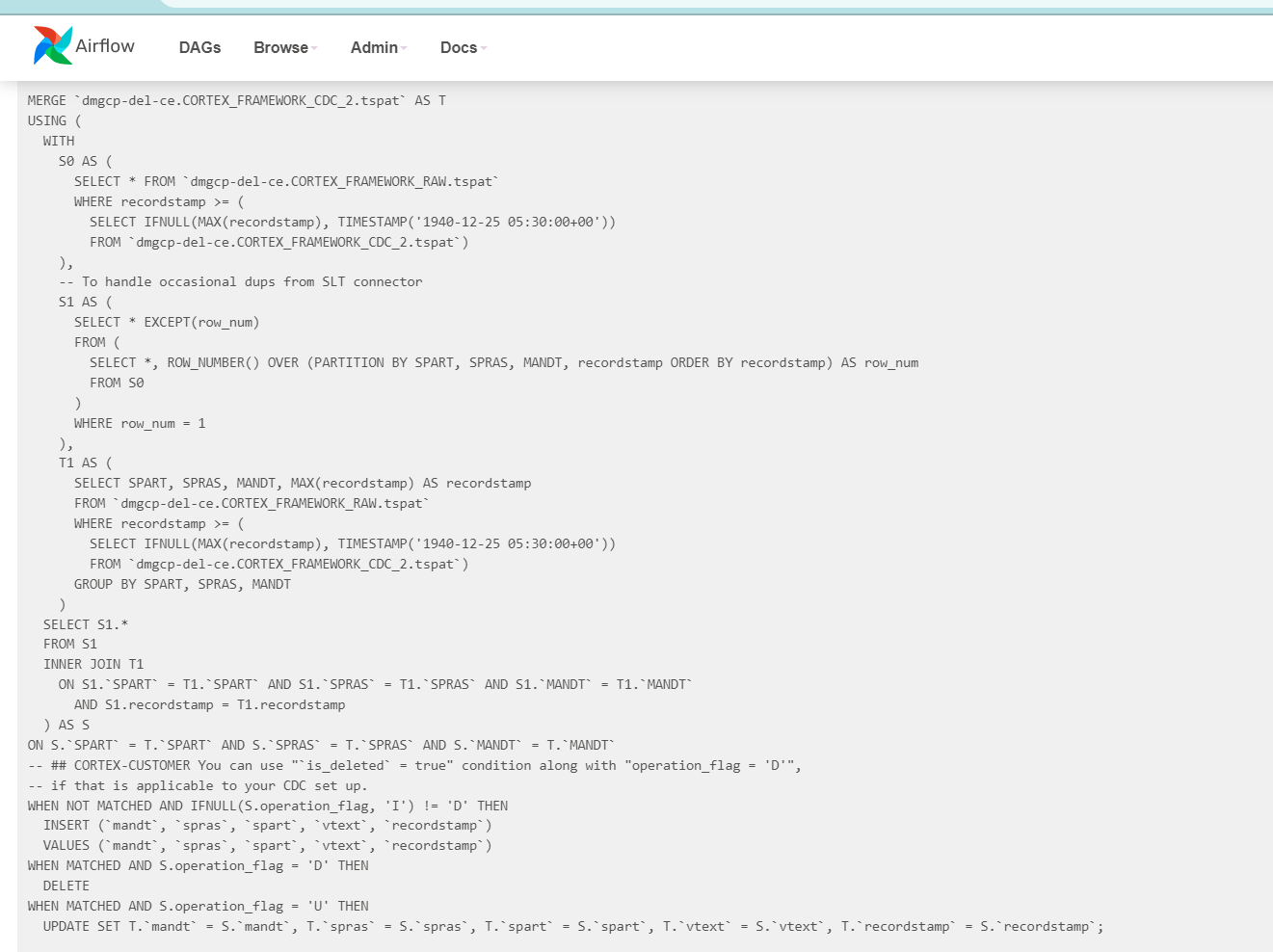
gsutil cp -r gs://<output bucket>/dags\* gs://<composer dag bucket>/dags

gsutil cp -r gs://<output bucket>/data/\* gs://<composer sql bucket>/data/

**Example - Run dag in composer and is runned success :-**



**Dag log :-**

****

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* **How to add new derived column in CDC tables :-**

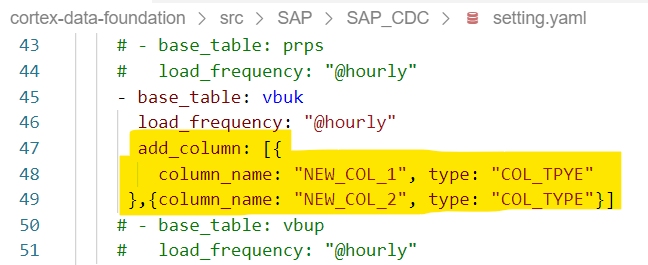
We need to change the configuration of [**setting.yaml**](https://github.com/GoogleCloudPlatform/cortex-dag-generator/blob/main/setting.yaml) file.

Code files path :- /src/SAP/SAP\_CDC/src

Yaml file path :- /src/SAP/SAP\_CDC

In [**setting.yaml**](https://github.com/GoogleCloudPlatform/cortex-dag-generator/blob/main/setting.yaml) file we add one more field add\_column and insert a new column, for better understanding refer to the below snap.

**Note :-** Remember one thing after adding new column fields ,while running the framework we will need to do some changes in **generate\_query.py** and **config\_reader.py** files.So accordingly do these changes.



Once columns are added and a table is created then how to populate these columns.Columns can be populated by using other column references (i.e. Derived Column).

Means for example in the table we have two columns first\_name and last\_name so what we can do is add a new column full\_name and **change the merge sql** for data loading from RAW to CDC tables.And add there this column condition (concat this column ex.:- full\_name = concat(first\_name,’ ’,last\_name))

Merge sql will be found in the bucket (in the data folder) that is provided while running the framework.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* **How to add new column in CDC tables :-**

If a new column is added in **raw** datasets tables then we will need to do some changes in **generate\_query.py** and **config\_reader.py** files and need to run the cortex framework again. No changes in [**setting.yaml**](https://github.com/GoogleCloudPlatform/cortex-dag-generator/blob/main/setting.yaml) file .These will only alter the table i.e. if **CDC** table already has data, column is added in last.So data is not truncated.(Alter table)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Below are reporting tables in cortex , but in cortex no any dags for these tables. That's why we created a view in the reporting layer .**

MaterialTypesMD

MaterialMovementTypesMD

PurchaseDocumentTypesMD

ReasonForMovementTypesMD

SpecialStocksMD

StockInHand

BillingBlockingReasonsMD

DeliveryBlockingReasonsMD

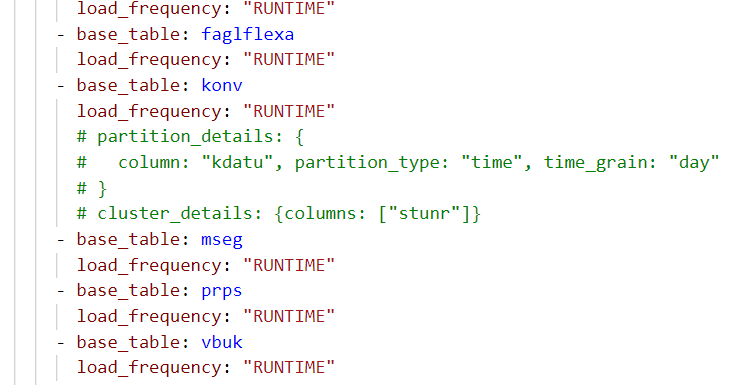
InvoiceDocuments\_Flow

* **Optional :-**

**If we create view for real time changes in CDC :-**

In [**setting.yaml**](https://github.com/GoogleCloudPlatform/cortex-dag-generator/blob/main/setting.yaml) file , we will need to change frequency as **RUNTIME** and make sure partition and clustering aren't used. If used then views will fail to create .

See the below snap for your reference.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# **Cortex Framework - ML Models**

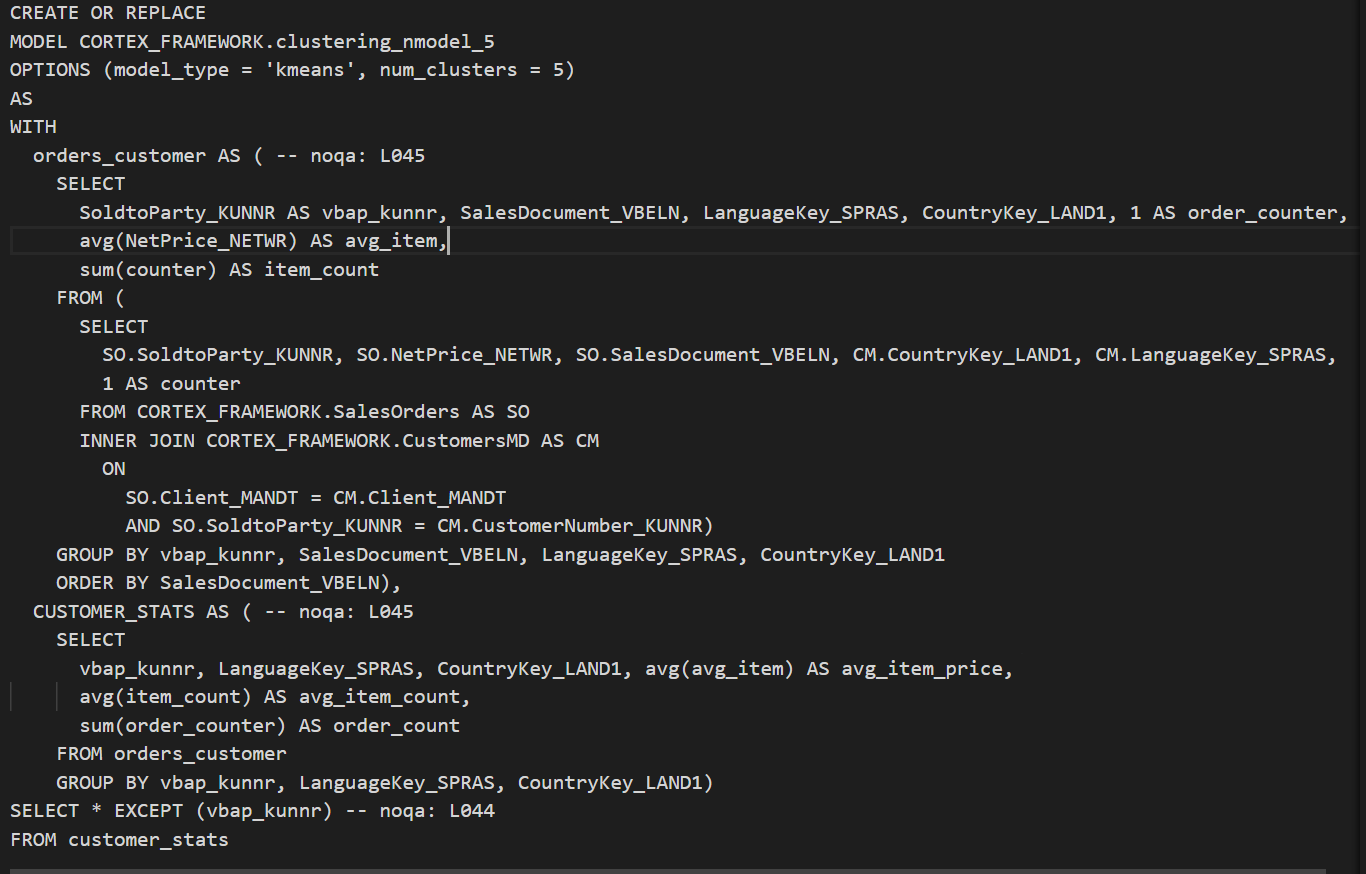
# 

There are two preexisting ML Models in the Cortex Framework:

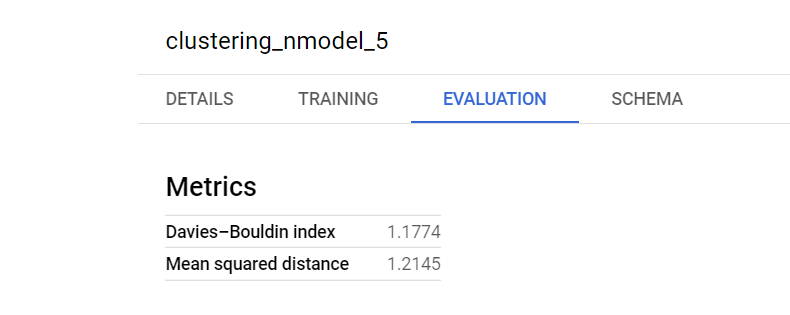
* K-Means Clustering
  + K-means clustering is a method of vector quantization that aims to partition **n** observations into **k** clusters, in which each observation belongs to the cluster with the nearest mean (cluster centroid), serving as a prototype of the cluster.
  + It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs to only one group that has similar properties.

**Input tables**: SalesOrders and CustomersMD

**Step 1:-** Run the below code in the editor to generate a clustering model, with desired number of clusters (Here, k=5).

****

Below is the evaluation for the model with 5 clusters:

****

As per the evaluation parameters, the lower the Davies-Bouldin Index value, the higher accuracy the clustering model has.

**Step 2:-** Create clusters with different values of k(2,3,4,6,7,etc) to find the best model with the least Davies-Bouldin Index.

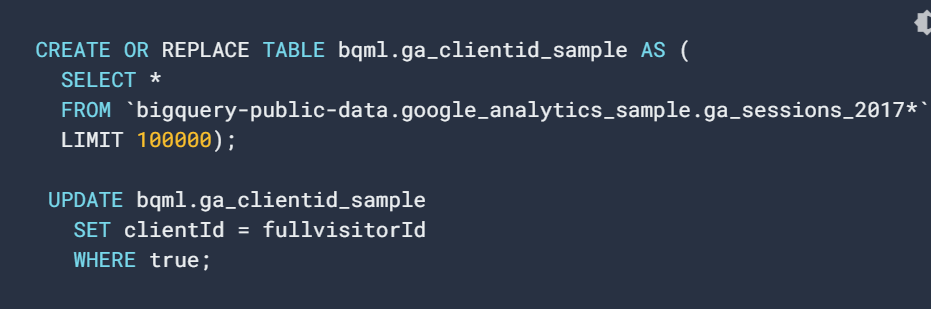
| **Index** | **No of cluster** | **Model** | **Model priority** | **Davies-Bouldin Index(DBI)** | **Mean Squared DIstance(MSD)** |
| --- | --- | --- | --- | --- | --- |
| 1 | 3 | clustering\_nmodel | 3 | 0.9142 | 1.9698 |
|  |  |  |  |  |  |
| 2 | 2 | clustering\_nmodel\_1 | 5 | 1.0018 | 2.581 |
|  |  |  |  |  |  |
| 3 | 4 | clustering\_nmodel\_4 | 1 | 0.6878 | 1.2358 |
|  |  |  |  |  |  |
| 4 | 5 | clustering\_nmodel\_5 | 6 | 1.0674 | 1.033 |
|  |  |  |  |  |  |
| 5 | 6 | clustering\_nmodel\_6 | 2 | 0.911 | 1.0646 |
|  |  |  |  |  |  |
| 6 | 7 | clustering\_nmodel\_7 | 4 | 0.9722 | 0.9322 |
|  |  |  |  |  |  |
| 7 | 8 | clustering\_nmodel\_8 | 6 | 1.042 | 0.6722 |
|  |  |  |  |  |  |
| 8 | 9 | clustering\_nmodel\_9 | 7 | 1.0603 | 0.5958 |
|  |  |  |  |  |  |
| 9 | 10 | clustering\_nmodel\_10 | 8 | 1.1064 | 0.5705 |

Based on the above details, the clustering model with **k=4**, has the least DBI, thereby indicating higher accuracy.

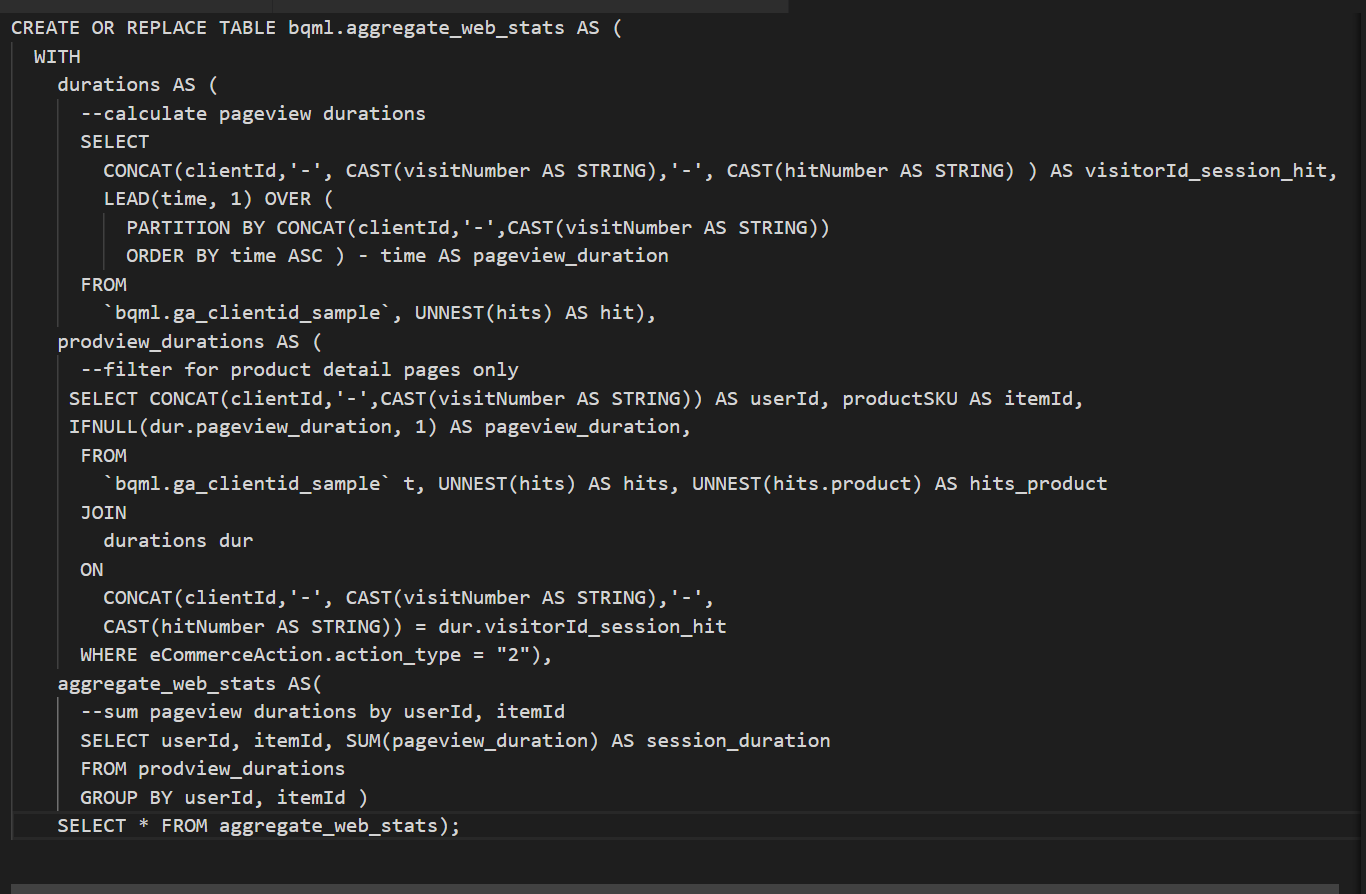
* Matrix Factorization
* Matrix factorization is a way to generate latent features when multiplying two different kinds of entities. It is used to identify the relationship between items’ and users’ entities.
* Assuming users to be a group of **m**, and items to be a group of **n**, matrix factorization maps **m** to **n**.

**Step 1:-** Create a table from the public BQ dataset **ga\_sessions\_2017\***

named **ga\_clientid\_sample.**

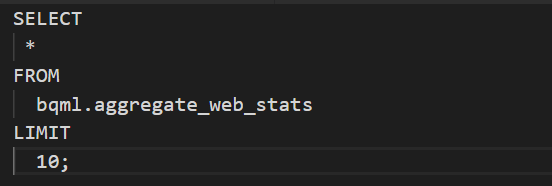
****

**Step 2:-** To create the training dataset from **ga\_clientid\_sample table,** run the below query in the editor window.

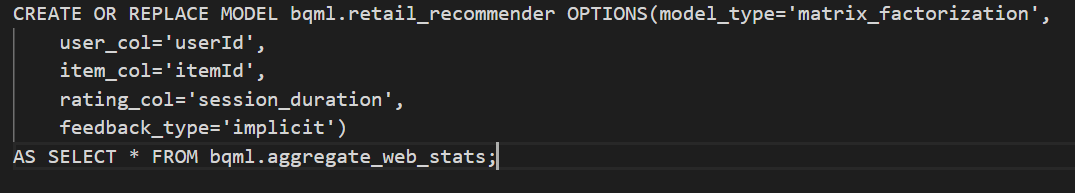


The result of the above query is a new table named ‘**aggregate\_web\_stats**’.

**Step 3:-** To review the data in the ‘**aggregate\_web\_stats**’ table, run the below query in the editor window.

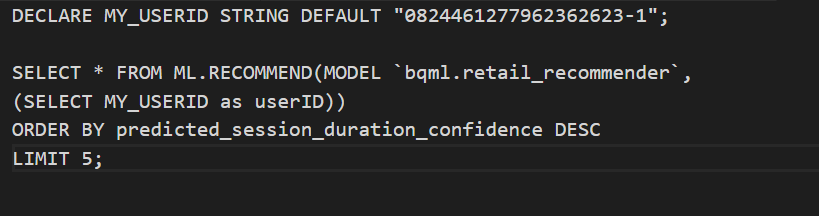


**Step 4:-** The next step is to create the matrix factorization model using fields from **aggregate\_web\_stats**.

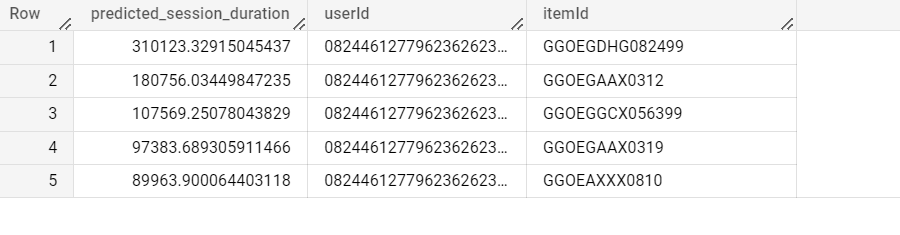


A new matrix factorization model called **retail\_recommender** is created.

**Step 4:-** Execute the following SQL statement to get predictions that represent the top 5 recommendations for a specified **userId.**

****

Below are the results of the query:

****

**Step 5:-**Create output table of top 5 predictions from ML Model (**retail\_recommender\_1**) using ML.RECOMMEND.

