Data Engineering Project

:ETL Data Integration on Covid-19 dataset based on AWS

Author: Akshit Mittal
Dated: 19 March 2023
Assigned: Self

Table of Contents

1. Glossary

1.1. ETL- Extract Transform Load

ETL stands for Extract, Transform, and Load, which is a process used in data management and data integration to gather data from various sources, transform it into a consistent format, and then load it into a target database or data warehouse. Each phase of the ETL process serves a specific purpose:

1. Extract: This phase involves extracting data from different source systems, which can be databases, spreadsheets, web services, flat files, or any other structured or unstructured data sources.

2. Transform: After extracting the data, the next step is to transform it into a standardised format that can be easily analysed and integrated with other data sources. Transformations may include data cleansing, data enrichment, aggregation, data validation, and other operations to ensure data quality and consistency.

3. Load: In the final phase, the transformed data is loaded into the target database or data warehouse, where it can be used for various purposes such as business intelligence, data analytics, reporting, or decision-making.

ETL is a fundamental process in data warehousing and data integration, enabling organisations to consolidate and analyse data from disparate sources, making it easier to derive insights and make informed business decisions. It plays a crucial role in modern data-driven environments, allowing companies to leverage data effectively and efficiently.

1.2 IAM

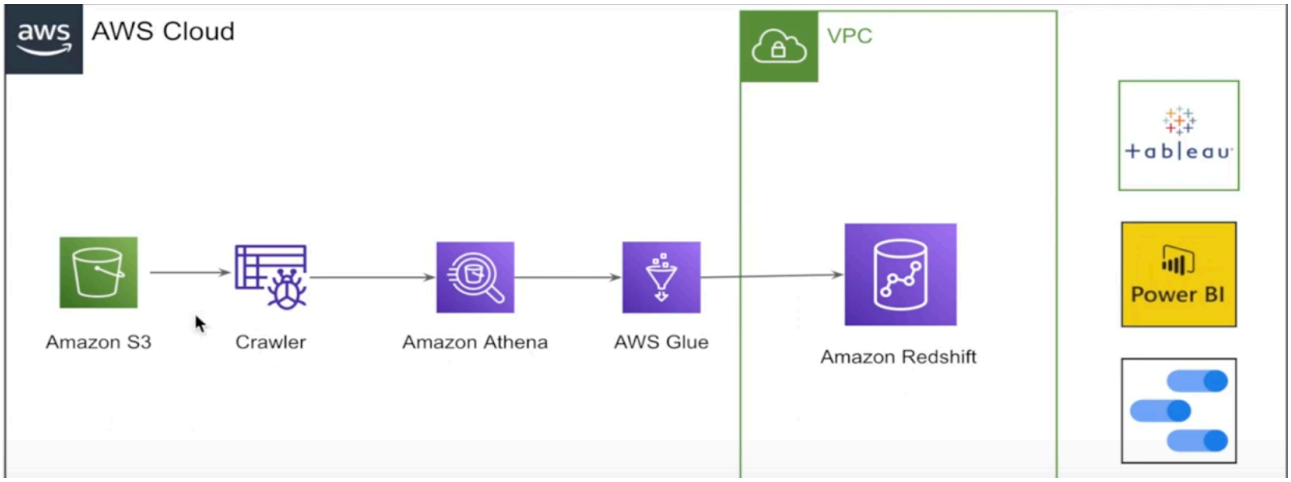
IAM stands for Identity and Access Management, and it is a crucial service provided by Amazon Web Services (AWS).
IAM enables you to securely control access to AWS resources and services for users, groups, and roles within your AWS account. With IAM, you can manage permissions and authentication for various entities, ensuring that only authorised users have the appropriate level of access to your AWS resources.

1.3 REDSHIFT

Redshift- Warehouse of Amazon
Amazon Redshift is a fully managed data warehousing service provided by Amazon Web Services (AWS). It's designed to handle large-scale data analytics and processing tasks, making it easier for organisations to analyse vast amounts of data and derive insights to support their decision-making processes.

1. Workflow (End to End) : Content and Context

2. The Pipeline

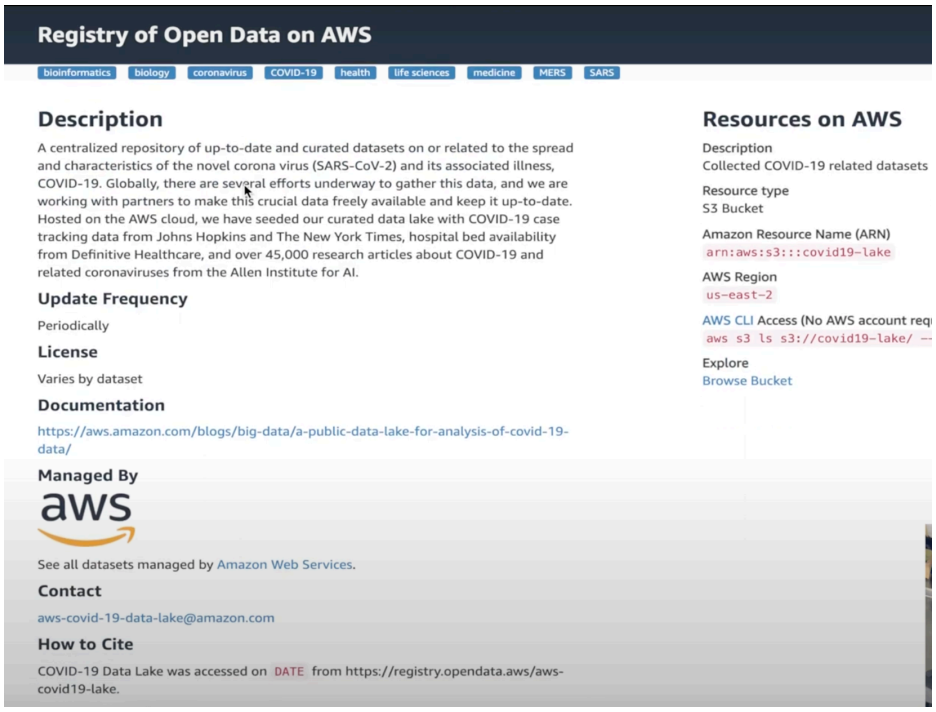


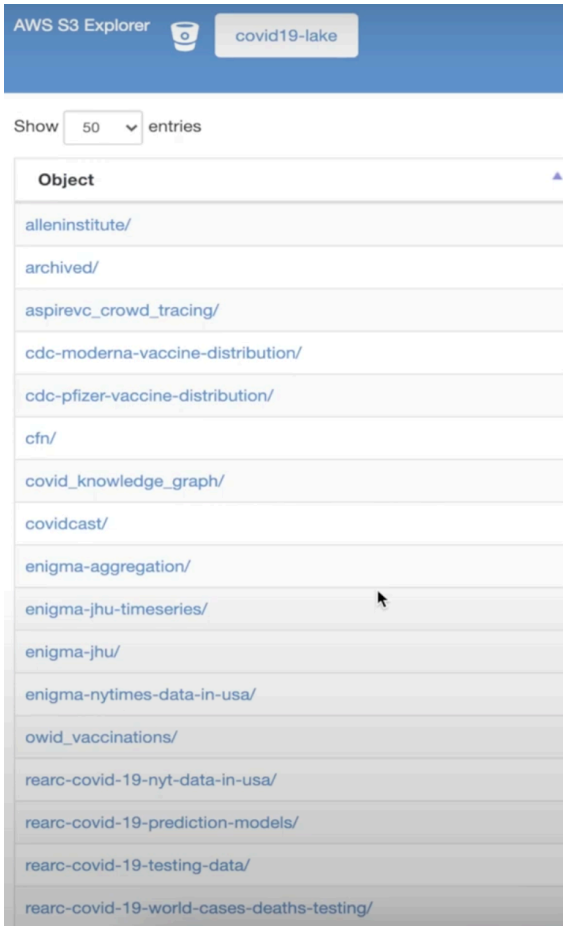
Extraction-	Dataset—> Amazon S3 — (Crawler) — (AWS Glue) —> Amazon Athena
Transform-	Transformation Job in Athena through Python (Jupiter Notebook)
Load-	Transformed data —> Amazon S3 —> Amazon Redshift warehouse
Analyses-	Tableau / Power BI. (*Not included in the project)

3. Covid-19 Dataset

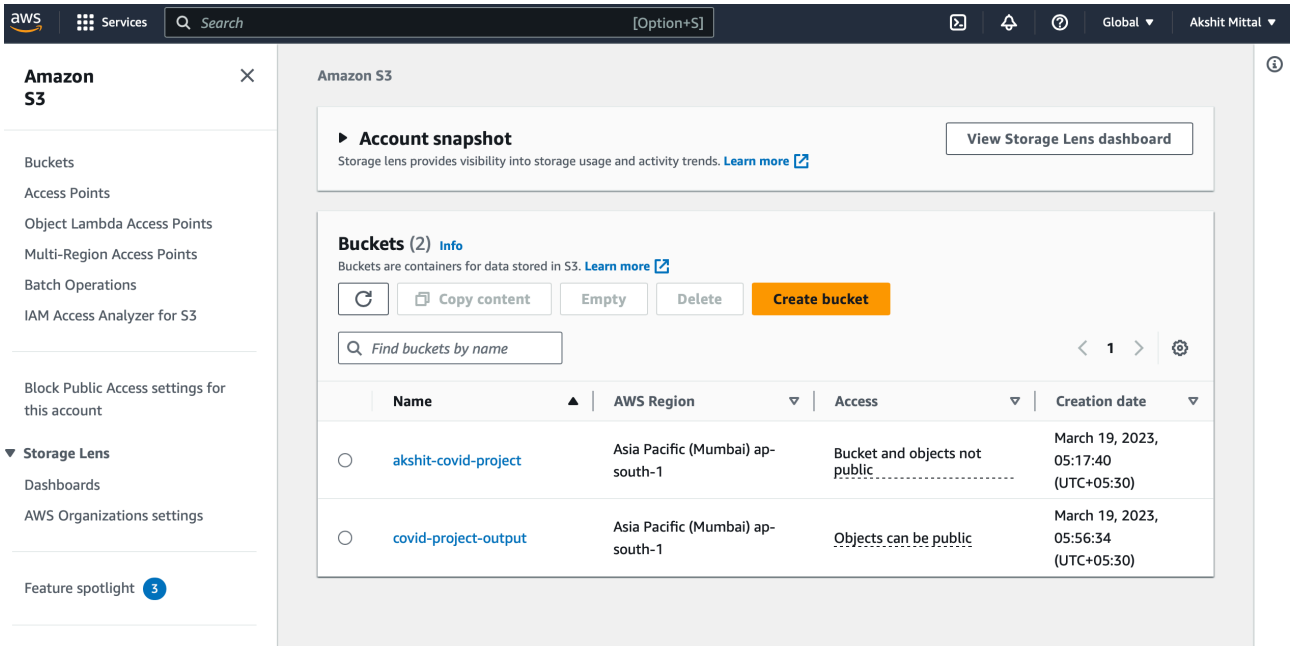
***SOU5RCE- OPEN DATA ON AWS**

- Extract data from AWS dataset

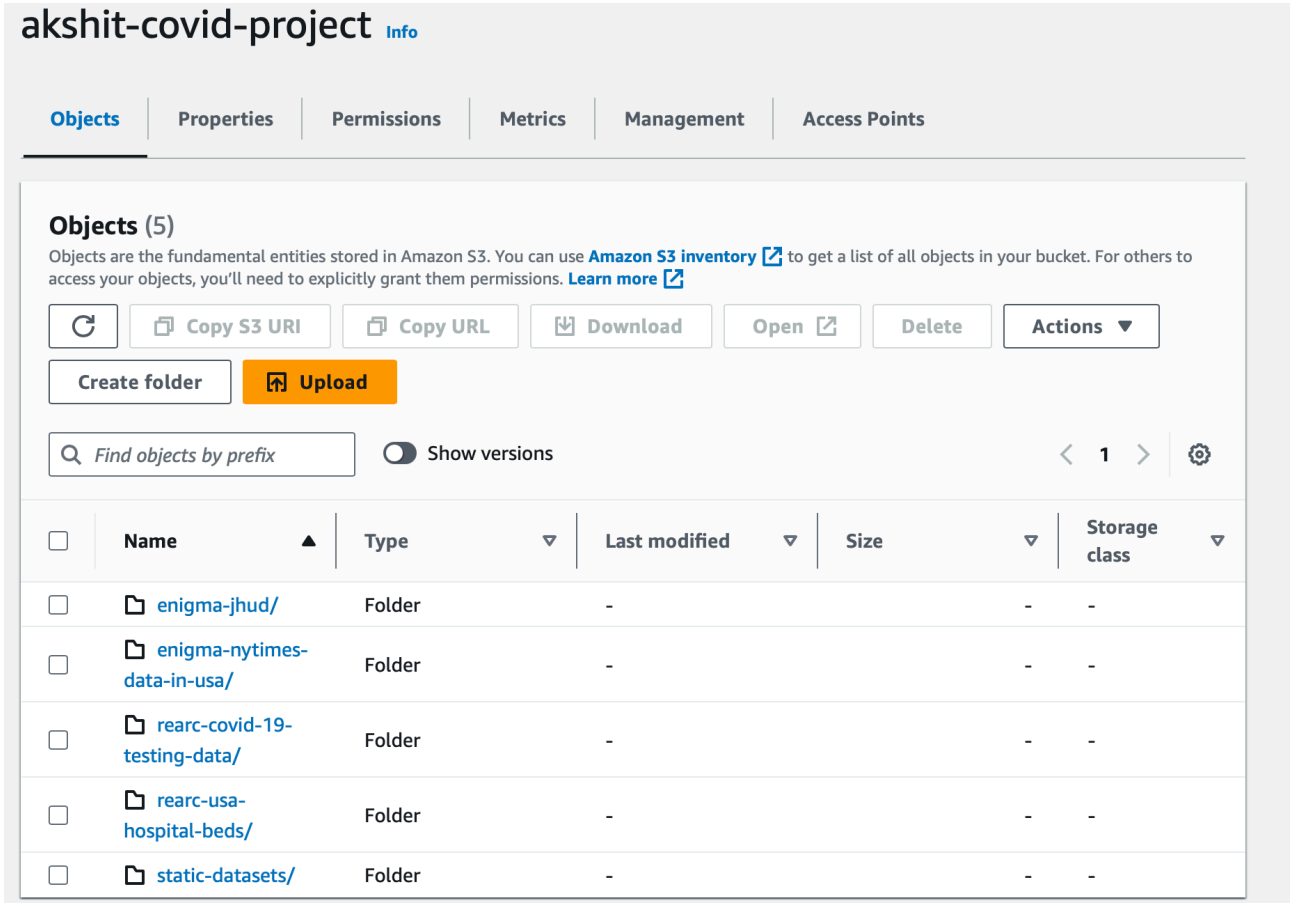




4. Amazon S3 Bucket



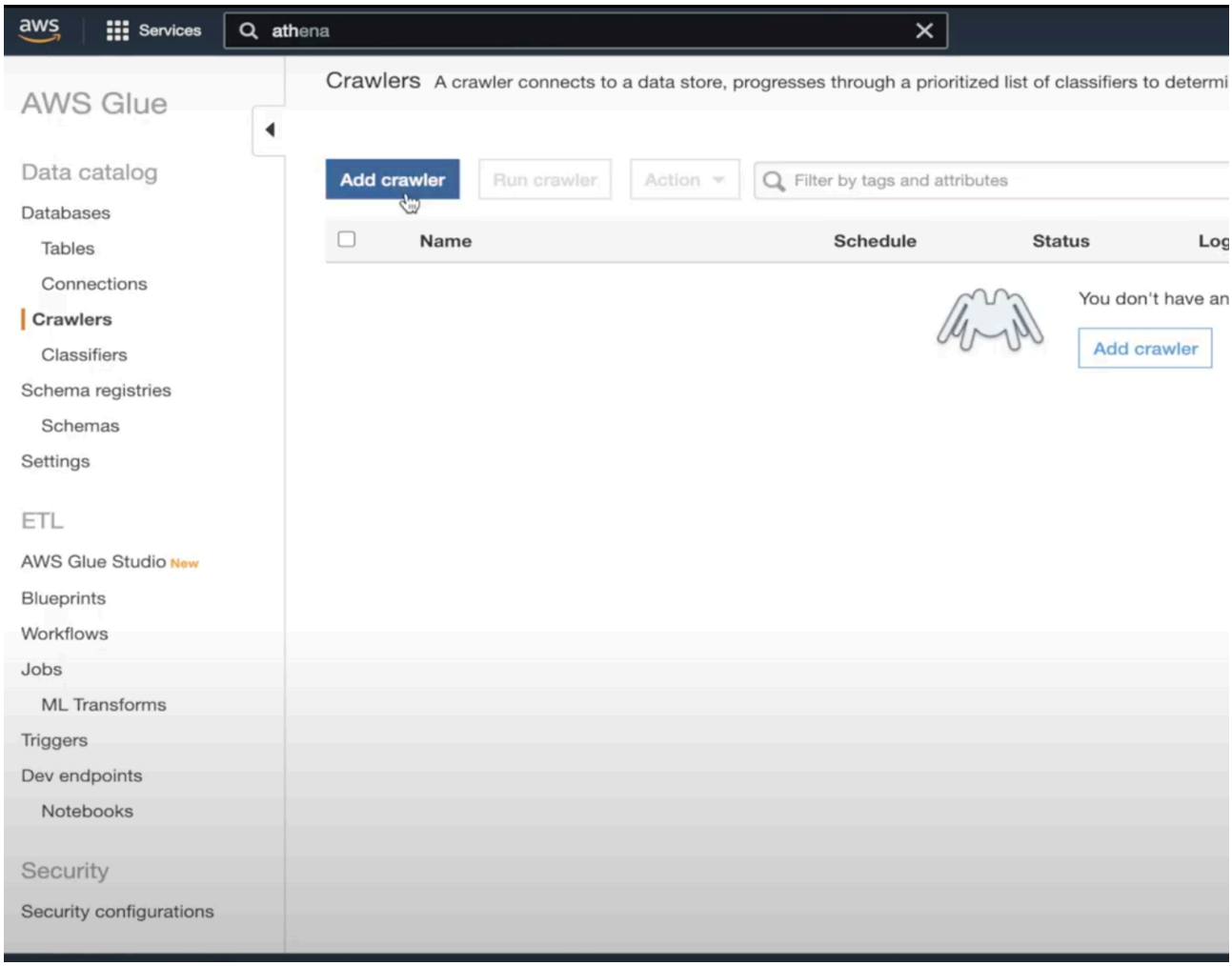
Inside the Bucket:



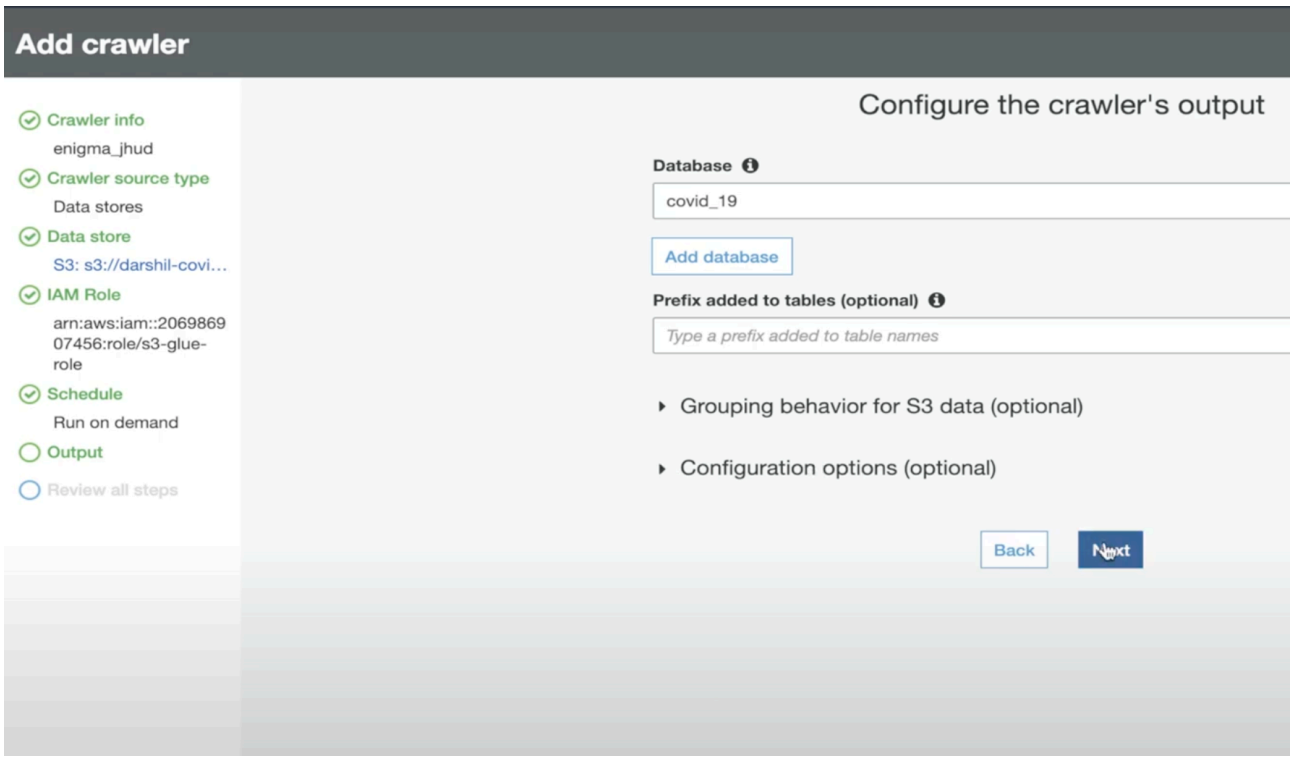
- There will be CSV & JSON Format available in the Data Set. Extract the files as per preference. We are using CSV data
- Stage the data into the S3 bucket and make the appropriate folders as shown in the image.

5. ETL Service in AWS Glue

- Search for the AWS Glue in the search bar.
- Different ETL Service tools and workflows, triggers will be available here.
- We will use the popular tool Crawler, for Crawling the dataset onto our cloud service database with can be queried, analysed and perform different functions upon in Athena
- Add the Crawler



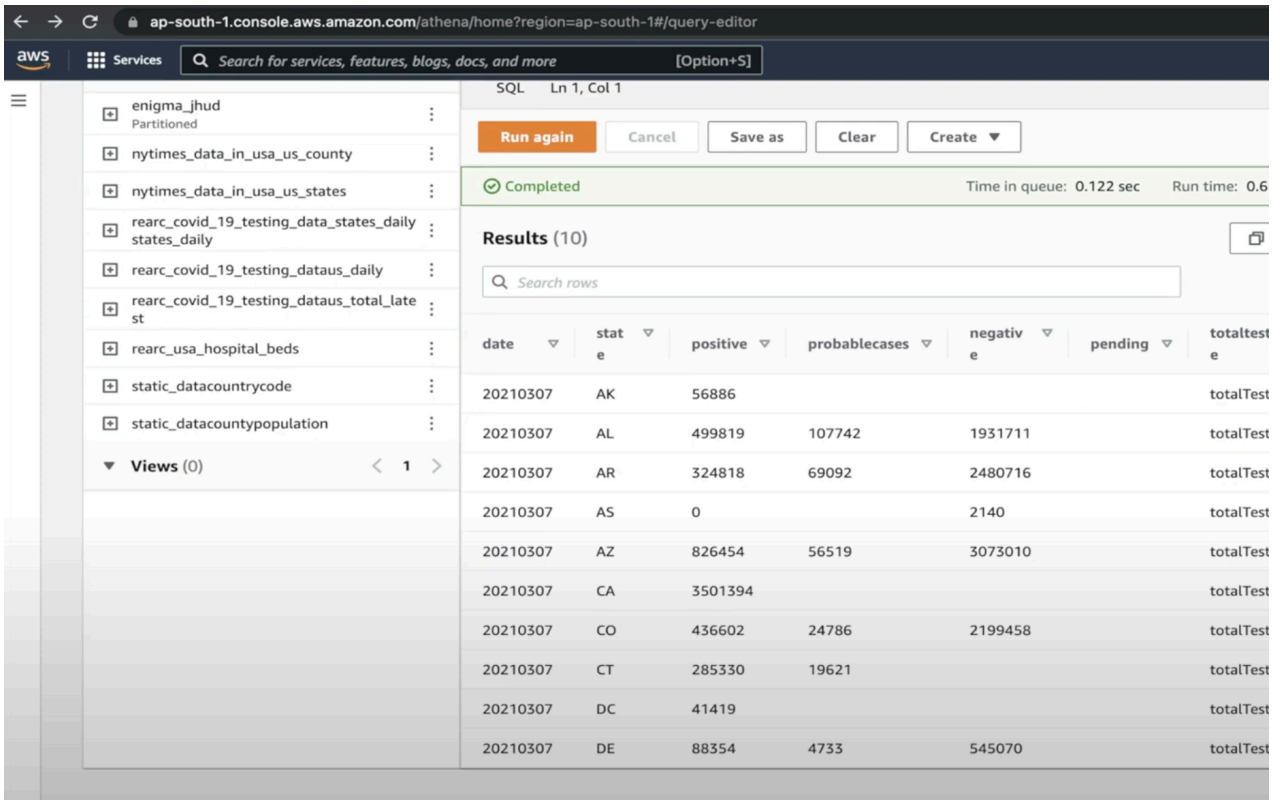
- Create a crawler by giving the path for every folder sequentially and name them according to your preferences. It will crawl all the data from the given folder of S3 bucket into the amazon Database which will be used in Athena query window, later.
- In between the process of crawling, we need to set up the different IAM roles, setting up different permissions, and giving some specific configurations for the Data.
- After IAM role, create the database name and folder, folder path.



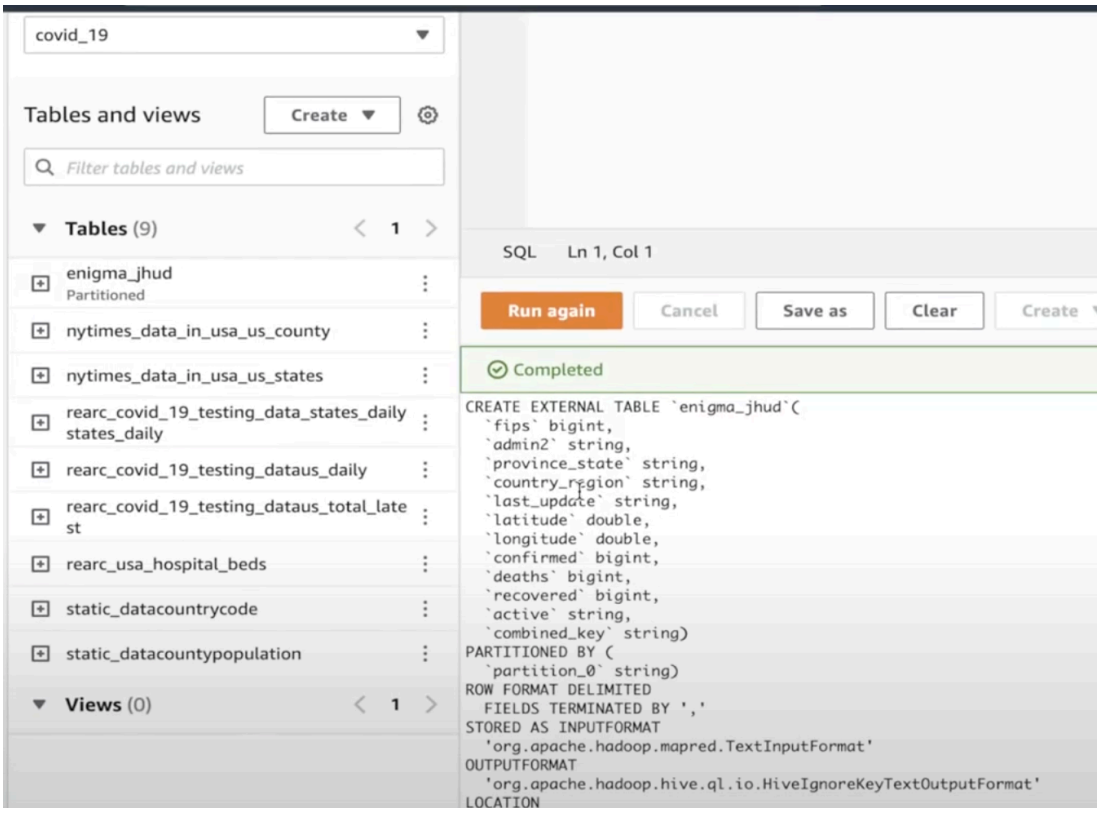
- After that , run the crawler of a particular folder we have crawled data of
- After 1-2 minutes, it will crawl the data into the database , which will be visible in the AWS Athena. Database folder you created.
- Perform this task for all the folders and files.

6. Data view and queries in Athena

- All the tables will be visible in Athena database like this

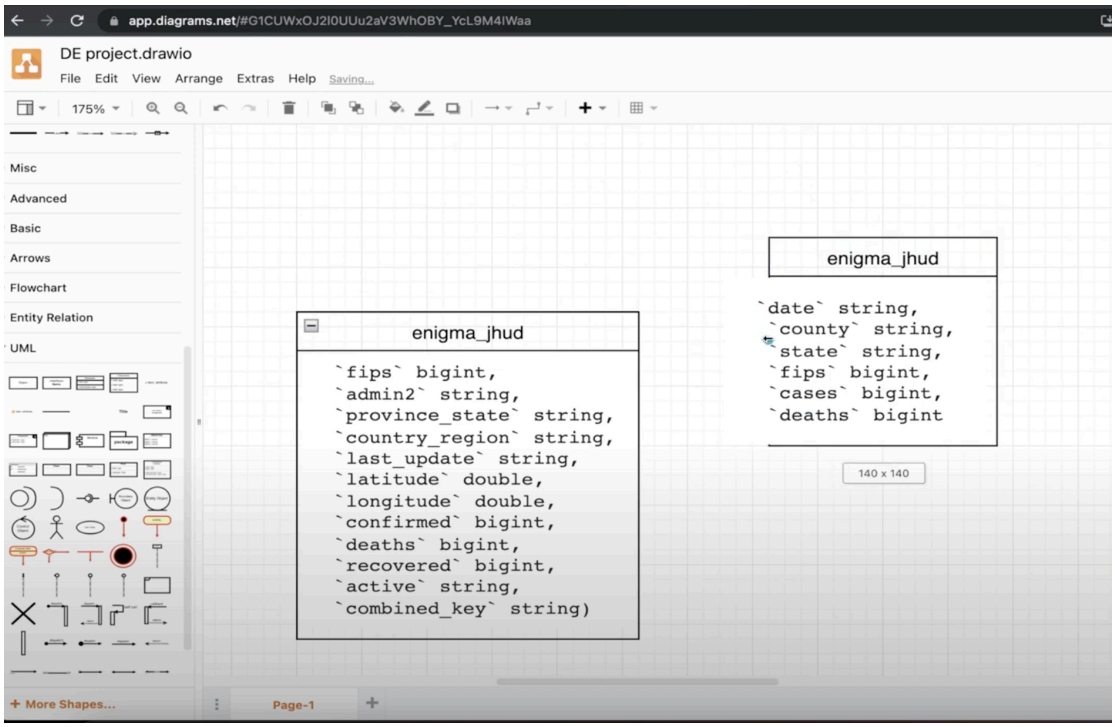


- We can run the queries in the Query field on any table to analyse the data or preview the table. Queries will be run in SQL language
- We will extract the Schema of the table to prepare our data model

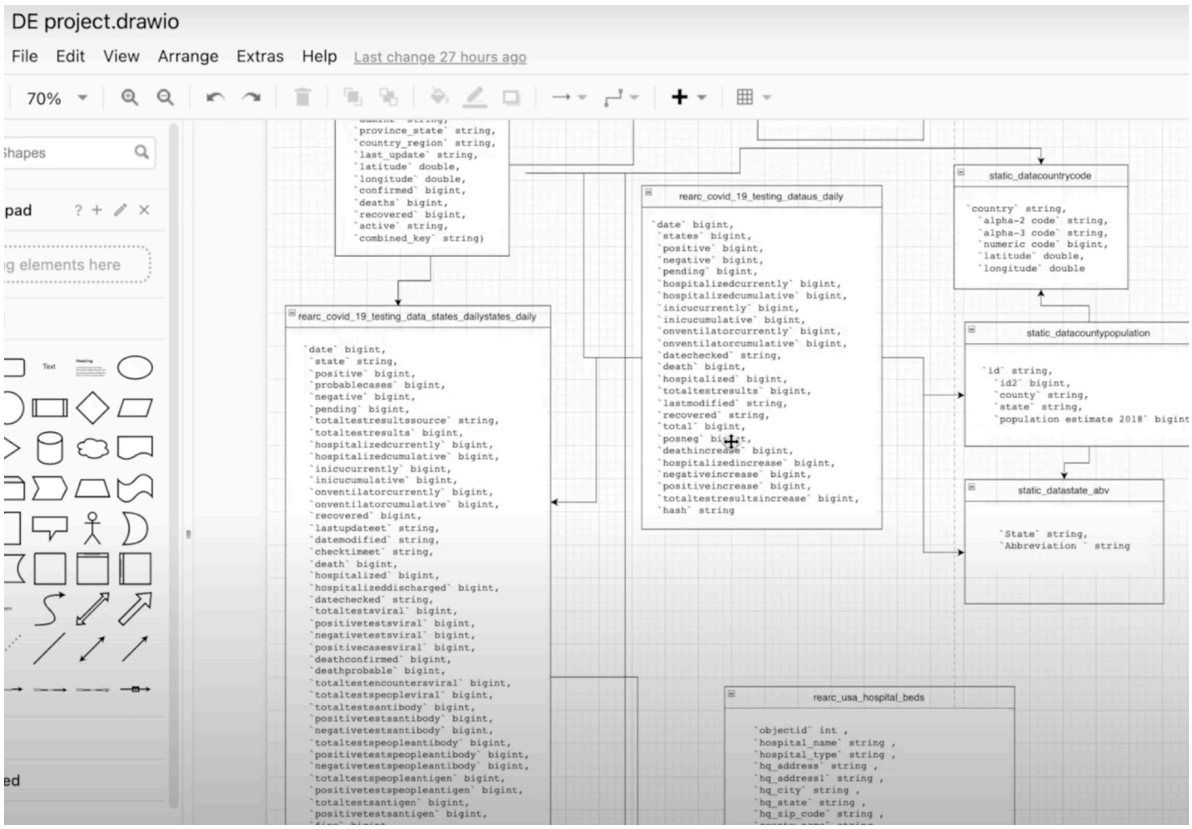


7. Preparing Data Model.

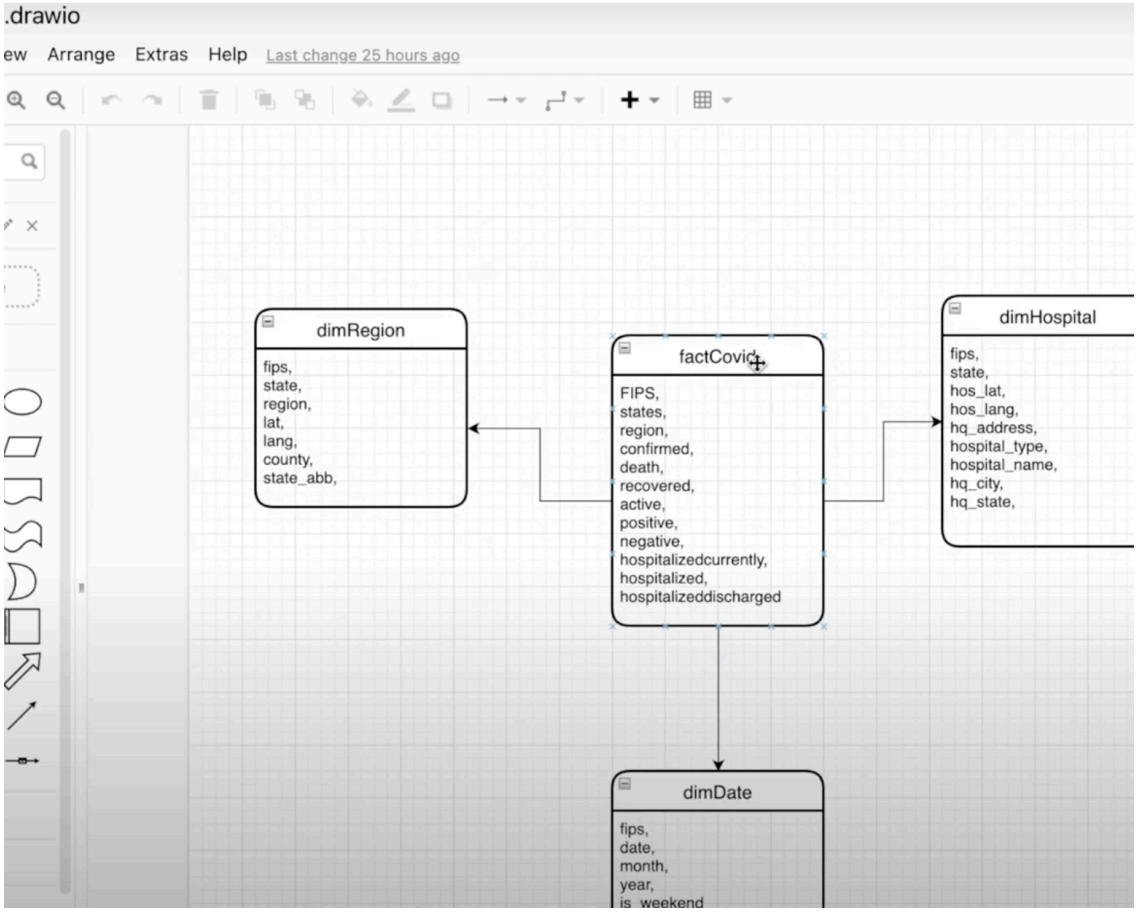
- Prepare Data Model with the Schema column names for every table



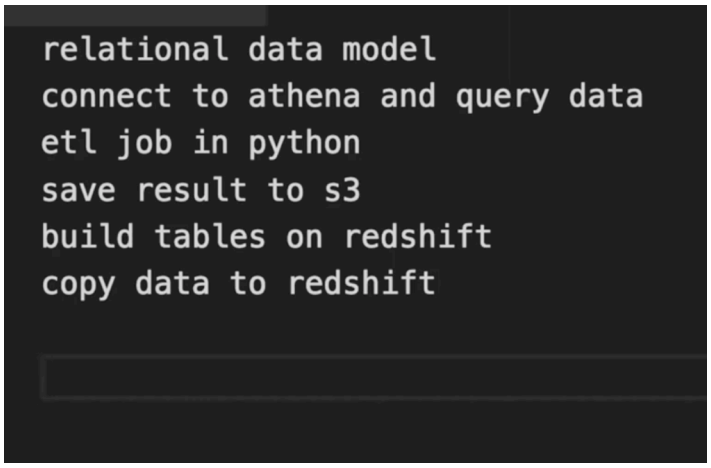
- We can use the tool draw.io for preparing data model.



- These data dimension model table can be converted to relational model afterwards
- Relational Model will look like this.



- Now we will do the following steps:-



8. Write ETL Job

- We will perform the ETL job in the Python (Jupiter Notebook). We can also perform this in Spark
- Import libraries like Pandas, BOTO3 for connecting AWS to your Jupiter notebook.
- Mention Access keys, Secret keys, AWS Region and other necessary details to connect system to AWS. Scheme name, Staging directory, Result/Output directory to give the path to output data, bucket name and other necessary information.
  *The Keys has been modified now, to secure the data

```
In [1]: import boto3
        import pandas as pd
        from io import StringIO # python3; python2: BytesIO
```

```
In [3]: AWS_ACCESS_KEY = "AKIATAMLBR5AF2JM526U"
        AWS_SECRET_KEY = "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"
        AWS_REGION = "ap-south-1"
        SCHEMA_NAME = "covid_19"
        S3_STAGING_DIR = "s3://darshil-test-bucket/output/"
        S3_BUCKET_NAME = "darshil-test-bucket"
        S3_OUTPUT_DIRECTORY = "output"
```

```
In [5]: athena_client = boto3.client(
            "athena",
            aws_access_key_id=AWS_ACCESS_KEY,
            aws_secret_access_key=AWS_SECRET_KEY,
            region_name=AWS_REGION,
        )
```

- We will Connect Athena to your BOTO3 client through python. By providing these data in Python
- Then with a “download and load query results” function, it will run some queries on BOTO3 client object and Athena and will save the results in the S3 bucket.

```
In [6]: Dict = {}
        def download_and_load_query_results(
            client: boto3.client, query_response: Dict
        ) -> pd.DataFrame:
            while True:
                try:
                    # This function only loads the first 1000 rows
                    client.get_query_results(
                        QueryExecutionId=query_response["QueryExecutionId"]
                    )
                    break
                except Exception as err:
                    if "not yet finished" in str(err):
                        time.sleep(0.001)
                    else:
                        raise err
            temp_file_location: str = "athena_query_results.csv"
            s3_client = boto3.client(
                "s3",
                aws_access_key_id=AWS_ACCESS_KEY,
                aws_secret_access_key=AWS_SECRET_KEY,
                region_name=AWS_REGION,
            )
            s3_client.download_file(
                S3_BUCKET_NAME,
                f"{S3_OUTPUT_DIRECTORY}/{query_response['QueryExecutionId']}.csv",
                temp_file_location,
            )
            return pd.read_csv(temp_file_location)
```

- We will Write another query in Python for a specific table in Athena to connect that to BOTO3 client. After successful connection with the Athena, it will give the configurations in a response.

```
In [7]: response = athena_client.start_query_execution(
            QueryString="SELECT * FROM enigma_jhud",
            QueryExecutionContext={"Database": SCHEMA_NAME},
            ResultConfiguration={
                "OutputLocation": S3_STAGING_DIR,
                "EncryptionConfiguration": {"EncryptionOption": "SSE_S3"},
            },
        )

In [8]: response

Out[8]: {'QueryExecutionId': '7cc27ec6-3664-45f8-a2fd-0a6e9504402f',
         'ResponseMetadata': {'RequestId': '9adcabb0-0701-4a74-9dd7-21b70451710b',
          'HTTPStatusCode': 200,
          'HTTPHeaders': {'content-type': 'application/x-amz-json-1.1',
           'date': 'Wed, 08 Dec 2021 18:59:06 GMT',
           'x-amzn-requestid': '9adcabb0-0701-4a74-9dd7-21b70451710b',
           'content-length': '59',
           'connection': 'keep-alive'},
          'RetryAttempts': 0}}

In [9]: enigma_jhud = download_and_load_query_results(athena_client, response)
```

- We can run the function of download and load query results on this latest function query which will give us the successful extraction of database into the pandas data-frame as shown.

- In similar way, we will perform this function for every table present in Athena to load every table data into the pandas data frame.

```
In [9]: enigma_jhud = download_and_load_query_results(athena_client, response)

In [10]: enigma_jhud.head()

Out[10]:
```

	fips	admin2	province_state	country_region	last_update	latitude	longitude	confirmed	deaths	recovered	active
0	NaN	NaN	Anhui	China	2020-01-22T17:00:00	31.826	117.226	1.0	NaN	NaN	NaN
1	NaN	NaN	Beijing	China	2020-01-22T17:00:00	40.182	116.414	14.0	NaN	NaN	NaN
2	NaN	NaN	Chongqing	China	2020-01-22T17:00:00	30.057	107.874	6.0	NaN	NaN	NaN
3	NaN	NaN	Fujian	China	2020-01-22T17:00:00	26.079	117.987	1.0	NaN	NaN	NaN
4	NaN	NaN	Gansu	China	2020-01-22T17:00:00	36.061	103.834	NaN	NaN	NaN	NaN

```
In [ ]:
```

- We can also change the column name or set the imbalance syntax of the table in the python itself using inbuilt functions like iloc.
- This is called Data Transformation, Sorting out the messy real time data!

9. Write the dimension model

- Write the dimension model in Python on the fetched tables through the queries. (Use JOIN/ MERGE functions to merge 2 columns or tables into one)

```
In [30]: factCovid.shape
Out[30]: (26418, 13)

In [31]: dimRegion_1 = enigma_jhud[['fips','province_state','country_region','latitude','longitude']]
         dimRegion_2 = nytimes_data_in_usa_us_county[['fips','county','state']]
         dimRegion = pd.merge(dimRegion_1, dimRegion_2, on='fips', how='inner')

In [32]: eds[['fips','state_name','latitude','longtitude','hq_address','hospital_name','hospital_type','hq_city','hq_state']]

In [33]: dimDate = rearc_covid_19_testing_data_states_dailystates_daily[['fips','date']]

In [34]: dimDate.head()
Out[34]:
```

	fips	date
0	2	20210307
1	1	20210307
2	5	20210307
3	60	20210307
4	4	20210307

- Prepare the Relational Model from the Dimension model in the similar way.

```
In [132]: dimDatesql = pd.io.sql.get_schema(dimDate.reset_index(), 'dimDate')
          print(''.join(dimDatesql))

CREATE TABLE "dimDate" (
"index" INTEGER,
  "fips" INTEGER,
  "date" TIMESTAMP,
  "year" INTEGER,
  "month" INTEGER,
  "day_of_week" INTEGER
)

In [134]: factCovidsql = pd.io.sql.get_schema(factCovid.reset_index(), 'factCovid')
          print(''.join(factCovidsql))

CREATE TABLE "factCovid" (
"index" INTEGER,
  "fips" REAL,
  "province_state" TEXT,
  "country_region" TEXT,
  "confirmed" REAL,
  "deaths" REAL,
  "recovered" REAL,
  "active" REAL,
  "date" INTEGER,
  "positive" REAL,
  "negative" REAL,
  "hospitalizedcurrently" REAL,
  "hospitalized" REAL,
  "hospitalizeddischarged" REAL
)
```

10. Store the output in S3 Bucket

- Deploy the output bucket
- After giving the bucket name of output bucket of S3 onto the python notebook, store the data into S3 through the following code.

```
In [40]: csv_buffer = StringIO()

In [41]: csv_buffer
Out[41]: <_io.StringIO at 0x11e246710>

In [42]: factCovid.to_csv(csv_buffer)

In [43]: s3_resource = boto3.resource('s3')
         s3_resource.Object(bucket, 'output/factCovid.csv').put(Body=csv_buffer.getvalue())

Out[43]: {'ResponseMetadata': {'RequestId': 'X5XEEM7K77YXS4Y9',
           'HostId': 'OkiHRr3pTrcOjfdbAcYhc6i5vgLFp3uSDLbOOILLM3ZFjiwIjkHNRc9qwtgdyd1Lb7Bh2fiV+xo=',
           'HTTPStatusCode': 200,
           'HTTPHeaders': {'x-amz-id-2': 'OkiHRr3pTrcOjfdbAcYhc6i5vgLFp3uSDLbOOILLM3ZFjiwIjkHNRc9qwtgdyd1Lb7Bh2fiV+xo=',
            'x-amz-request-id': 'X5XEEM7K77YXS4Y9',
            'date': 'Wed, 08 Dec 2021 19:19:54 GMT',
            'etag': '"2b9776182b96c2fda29a5fdbbce21dd6"',
            'server': 'AmazonS3',
            'content-length': '0'},
           'RetryAttempts': 0},
          'ETag': '"2b9776182b96c2fda29a5fdbbce21dd6"'}

In [44]: csv_buffer.getvalue()

Out[44]: ',fips,province_state,country_region,confirmed,deaths,recovered,active,date,positive,negative,hospitalizedcurrently
         ,hospitalized,hospitalizeddischarged\n0,72.0,Puerto Rico,US,3.0,0.0,0.0,,20210307,101327.0,305972.0,147.0,,\n1,72.0
         ,Puerto Rico,US,3.0,0.0,0.0,,20210306,101327.0,305972.0,147.0,,\n2,72.0,Puerto Rico,US,3.0,0.0,0.0,,20210305,101066
```

- Covert the data into the Csv format with the .io class defined at the start of ETL job
- Put the bucket name into the S3 , and you will get the response as output after it gets successful
- We have successfully stored the data into he S3 bucket

11. Importing data into the Amazon RedShift

- Now we need to build the schema of table into the redshift based on schema of our output of our relational data model folder in S3
- First step is, write python query to extract schema of the 1st output table. Similarly, perform this task for all other output tables.

```
In [135]: dimRegionsql = pd.io.sql.get_schema(dimRegion.reset_index(), 'dimRegion')
          print(''.join(dimRegionsql))

          CREATE TABLE "dimRegion" (
          "index" INTEGER,
            "fips" REAL,
            "province_state" TEXT,
            "country_region" TEXT,
            "latitude" REAL,
            "longitude" REAL,
            "county" TEXT,
            "state" TEXT
          )

In [136]: dimHospitalsql = pd.io.sql.get_schema(dimHospital.reset_index(), 'dimHospital')
          print(''.join(dimHospitalsql))

          CREATE TABLE "dimHospital" (
          "index" INTEGER,
            "fips" REAL,
            "state_name" TEXT,
            "latitude" REAL,
            "longtitude" REAL,
            "hq_address" TEXT,
            "hospital_name" TEXT,
            "hospital_type" TEXT,
            "hq_city" TEXT,
            "hq_state" TEXT
          )
```

-

- Import library - redshift connector.
- Connect redshift with help of username, key and database and password from AWS
- Import the schema using cursor automation technique in python

```
In [137]: import redshift_connector

In [138]: conn = redshift_connector.connect(
              host='redshift-cluster-2.ctlwvzbuur6m.ap-south-1.redshift.amazonaws.com',
              database='dev',
              user="awsuser",
              password='Passw0rd123'
           )

In [139]: conn.autocommit = True

In [140]: cursor= redshift_connector.Cursor = conn.cursor()

In [141]: cursor.execute("""
          CREATE TABLE "dimDate" (
          "index" INTEGER,
            "fips" INTEGER,
            "date" TIMESTAMP,
            "year" INTEGER,
            "month" INTEGER,
            "day_of_week" INTEGER
          )
          """)

Out[141]: <redshift_connector.cursor.Cursor at 0x12fe311e0>
```

```
  "hospitalizeddischarged" REAL
)
""")
```

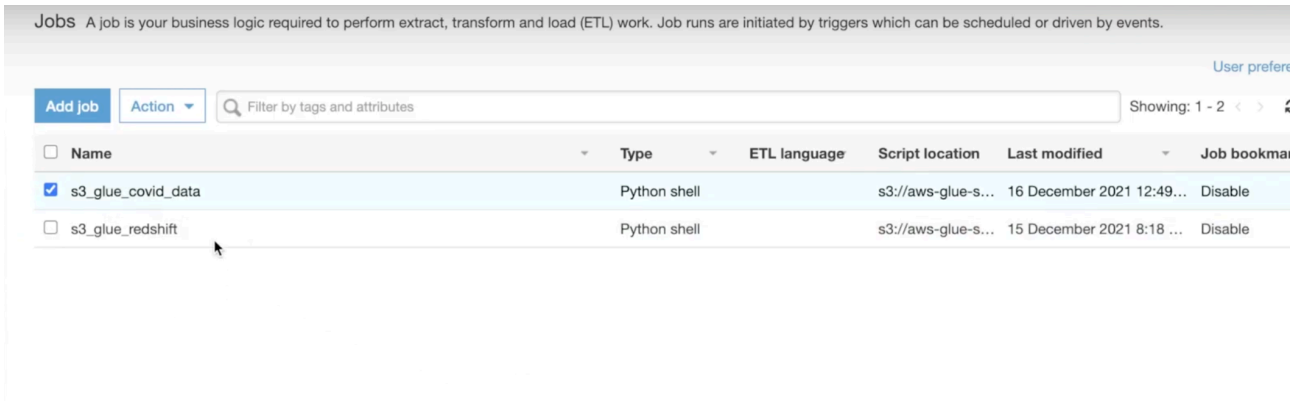
```
Out[143]: <redshift_connector.cursor.Cursor at 0x12fe311e0>
```

```
In [144]: cursor.execute("""CREATE TABLE "dimRegion" (
"index" INTEGER,
  "fips" REAL,
  "province_state" TEXT,
  "country_region" TEXT,
  "latitude" REAL,
  "longitude" REAL,
  "county" TEXT,
  "state" TEXT
)""")
```

```
Out[144]: <redshift_connector.cursor.Cursor at 0x12fe311e0>
```

```
In [ ]: cursor.execute("""
copy dimDate from 's3://darshil-covid-de-project/output/dimDate.csv'
credentials 'aws_iam_role=arn:aws:iam::206986907456:role/redshift-s3-access'
delimiter ','
region 'ap-south-1'
IGNOREHEADER 1
""")
```

- Perform importing schema and table on the redshift for all the tables.
- Importing to redshift is successful



- All the relational data tables will be visible on redshift.
- ETL Job is completed