

“SENTIMENTAL ANALYSIS OF YELP’S BUSINESS REVIEWS and USER DATA” USING-3 NF

Naveen Varma Pandeti (Pande3n)

Busireddy Sumanth Reddy(Busir1s)

Sri Swetha Tirumala Kanduri (Tirum4s)

Course: Applied Data Engineering (CPS 585)

Date: 04-26-2023

Section 1: OVERVIEW-



What is Yelp?

Yelp is a popular online directory of local businesses from various industries: bars, restaurants, spas, gas stations etc. Results of a search query are filtered by geographical location, business type, price ranges, and unique features. The idea behind Yelp is for users to leave their rating and a review of their experience with an establishment. It can be a treasure chest of guidelines on how customers view your business, and what improvements can be made to see more happy customers. So, how exactly can Yelp reviews help us explore the satisfaction of our customers.



popular online directory of local businesses from various industries



The idea behind Yelp is for users to leave their rating and a review of their experience with an establishment.



Yelp reviews can be used to gain insights into customer experiences, identify areas for improvement, and gauge customer sentiment towards the business.

We use to create data pipeline that converts the non-relational Yelp dataset, which is dispersed over **JSON** files in an **Amazon S3 bucket**, into a 3NF-normalized dataset kept on Amazon Redshift. The resulting schema serves as the authoritative source for analytical queries and business intelligence (BI) tools, guaranteeing data consistency and referential integrity across tables. The data was further enhanced with demographic and weather information from outside data sources.

About Data sets:

#1Yelp Open Dataset

The Yelp Open Dataset dataset is a subset of Yelp's businesses, reviews, and user data, available for academic use. The dataset (as of 13.08.2019) takes 9GB disk space (unzipped) and counts 6,685,900 reviews, 192,609 businesses over 10 metropolitan areas, over 1.2 million business attributes like hours, parking, availability, and ambience, 1,223,094 tips by 1,637,138 users, and aggregated check-ins over time. Each file is composed of a single object type, one JSON-object per-line. For more details on dataset structure, proceed to [Yelp Dataset JSON Documentation](#).

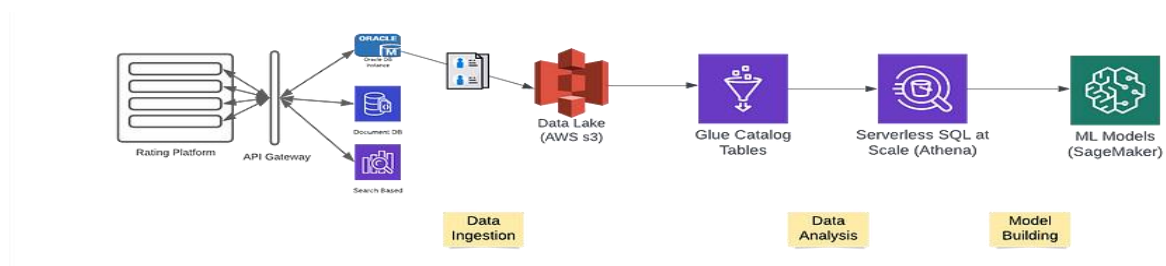
#2U.S. City Demographic Data

The U.S. City Demographic Data dataset contains information about the demographics of all US cities and census-designated places with a population greater or equal to 65,000. This data comes from the US Census Bureau's 2015 American Community Survey. Each JSON object describes the demographics of a particular city and race, and so it can be uniquely identified by the city, state and race fields.

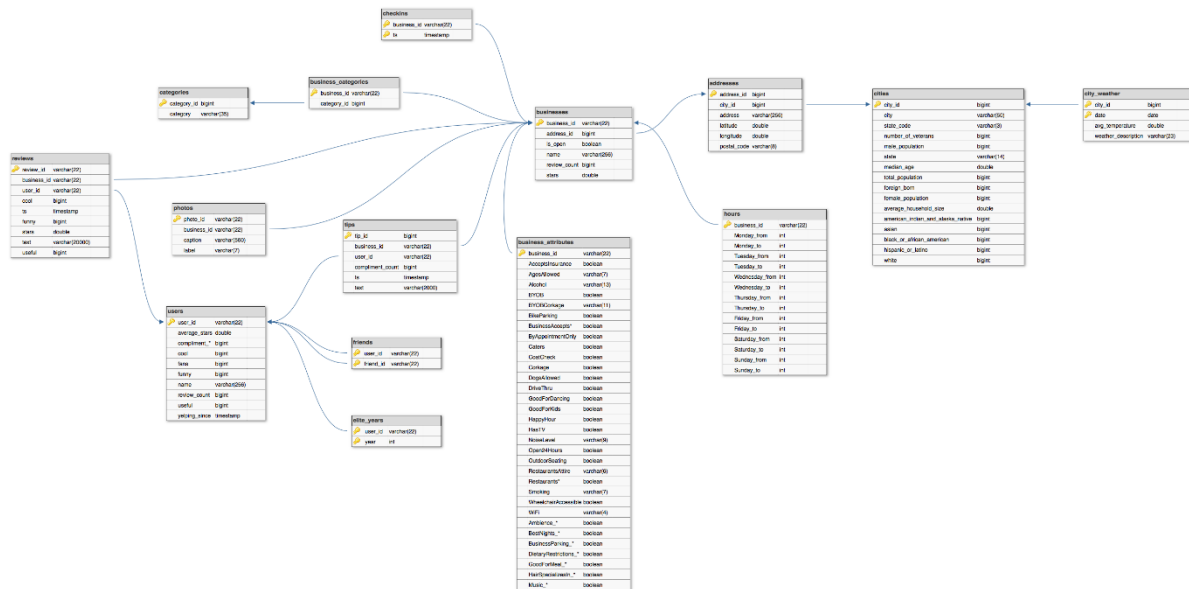
#3Historical Hourly Weather Data 2012-2017

The Historical Hourly Weather Data dataset is a dataset collected by a Kaggle competitor. The dataset contains 5 years of hourly measurements data of various weather attributes, such as temperature, humidity, and air pressure. This data is available for 27 bigger US cities, 3 cities in Canada, and 6 cities in Israel. Each attribute has it's own file and is organized such that the rows are the time axis (timestamps), and the columns are the different cities. Additionally, there is a separate file to identify which city belongs to which country.

Section 2: HIGH LEVEL DESIGN –



We have Ingested (Data Ingestion) our dataset files into AWS s3 using a Python based library to manage AWS services and their components. (By importing from AWS boto3). Later on, we have Created Glue Crawler to crawl through all the folders and files in s3 to create AWS Glue Catalog Database and tables. Run ad hoc queries using AWS Athena leveraging Glue Catalog Tables. We also talk about how Data Engineering Pipelines are typically built using AWS Analytics Services such as AWS EMR etc..



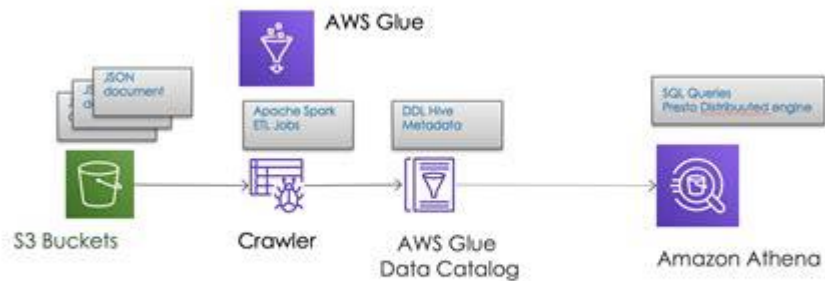
Our goal data model is a 3NF-normalized relational model that was designed to be neutral to various types of analytical queries. The data should be dependent on the key [1NF], the entire key [2NF], and nothing but the key [3NF].

Our The model has 15 tables as a consequence of normalizing and merging 6 Yelp tables, 1 table with demographic information, and 2 tables with meteorological information. Because there are two fact tables - reviews and tips - and numerous dimensional tables with multiple levels of hierarchy and many-to-many links, the design is more similar to a Snowflake model. Some tables retain their native keys, whereas others have ids that increase monotonically. Use generated keys for entities and composite keys for relationships as a general rule. Furthermore, timestamps and dates were translated into Spark's native data types so that they could be imported correctly into **Amazon Redshift** or **Amazon Glue**.

Overview of Glue:

AWS Glue is fully managed service from AWS which have several Data Engineering Components.

It is used to crawl the data and quickly create tables which can be used by other AWS Analytics services such as Athena, EMR, Glue Workflows, etc.

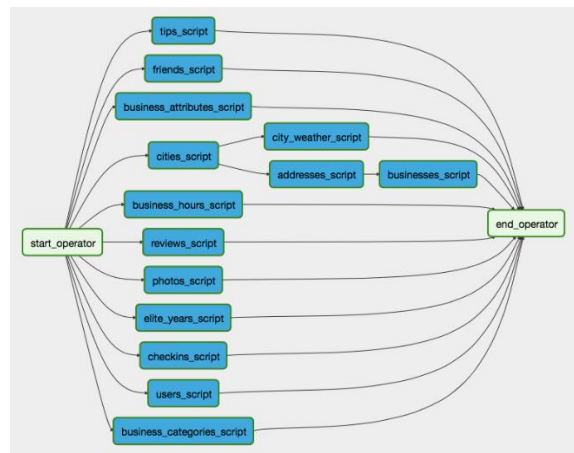


Process with Spark:



The data needs to be transformed into a relational form to be loaded into Amazon Redshift, which does not support nested data. Apache Spark is used to perform this transformation and execute the data processing pipeline in an Amazon EMR cluster. Spark also allows for data quality checks to be performed at this stage.

Assumption load:



Section 3: IMPLEMENTATION

Main Pre-requisites:

#1 Need an Valid AWS account .

#2 Required a terminal like AWS CLI or Gitbash to configure with Aws instances.

Load – Loading data in to S3 bucket



Amazon S3

Create an S3 bucket.

- Ensure that the bucket is in the same region as your Amazon EMR and Redshift clusters.

We have Download Yelp Open Dataset and directly Uploaded that data sets to our S3 bucket (Cps585 folder).

The screenshot shows the AWS S3 'Upload' page. The breadcrumb trail is 'Amazon S3 > Buckets > cps585 > Upload'. The page title is 'Upload' with an 'info' link. Below the title, there is a message: 'Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. [Learn more](#)'. A dashed box contains the instruction: 'Drag and drop files and folders you want to upload here, or choose **Add files**, or **Add folders**.' Below this, a section titled 'Files and folders (5 Total, 8.7 GB)' contains a table of files to be uploaded. The table has columns for 'Name', 'Folder', 'Type', and 'Size'. There are five files listed, all of type 'application/json' and located in the 'Project json/' folder. A search bar with the placeholder 'Find by name' and a pagination control showing '1' are also present.

Name	Folder	Type	Size
yelp_academic_dataset_business.json	Project json/	application/json	113.4 MB
yelp_academic_dataset_checkin.json	Project json/	application/json	273.7 MB
yelp_academic_dataset_review.json	Project json/	application/json	5.0 GB
yelp_academic_dataset_tip.json	Project json/	application/json	172.2 MB
yelp_academic_dataset_user.json	Project json/	application/json	3.1 GB

We have successfully loaded our data into S3 bucket :

The screenshot shows the 'Upload succeeded' confirmation page in AWS S3. At the top, a green banner says 'Upload succeeded' with a link to 'View details below.'. Below this is a 'Summary' section with three columns: 'Destination' (s3://cps585), 'Succeeded' (5 files, 8.7 GB (100.00%)), and 'Failed' (0 files, 0 B (0%)). Below the summary are two tabs: 'Files and folders' (selected) and 'Configuration'. The 'Files and folders' tab shows a table with columns for 'Name', 'Folder', 'Type', 'Size', 'Status', and 'Error'. All five files are listed with a status of 'Succeeded' and no errors.

Name	Folder	Type	Size	Status	Error
yelp_academic_dataset_business.json	Project json/	application/json	113.4 MB	✓ Succeeded	-
yelp_academic_dataset_checkin.json	Project json/	application/json	273.7 MB	✓ Succeeded	-
yelp_academic_dataset_review.json	Project json/	application/json	5.0 GB	✓ Succeeded	-
yelp_academic_dataset_tip.json	Project json/	application/json	172.2 MB	✓ Succeeded	-
yelp_academic_dataset_user.json	Project json/	application/json	3.1 GB	✓ Succeeded	-

After successfully loading our data in to s3 bucket we have to create Ec2 instance and Launch an EC2 instance with at minimum storage of 20GB SSD .

The screenshot displays the AWS Management Console interface. On the left, the 'Instances' section is selected in the navigation menu. The main area shows a list of instances with one instance, 'yelpInstance', in a 'Running' state. Below the list, the details for 'Instance: i-0eadad2c56c47c439d (yelpInstance)' are shown. The details include the Instance ID, IP address, Hostname type, Public IPv4 address (18.188.18.148), Instance state (Running), Private IP DNS name, and Private IP addresses (172.31.34.155). The console also shows the instance's Public IPv4 DNS (ec2-18-188-18-148.us-east-2.compute.amazonaws.com) and Networking details (c2-18-188-18-148).

- And then we have connected to this instance via SSH (secure shell protocol) by using gitbash .

```

MINGW64/C:/Users/Varma06/Downloads
C:\Users\Varma06\Downloads>ssh -i C:\Users\Varma06\Downloads\yelpInstance.pem root@18.188.18.148
Last login: Fri Apr 21 22:53:52 2023 from 138.229.56.88
[root@ip-172-31-34-155 ~]# sudo su
[ec2-user@ip-172-31-34-155 ~]$ cd /tmp
[ec2-user@ip-172-31-34-155 ~]$ wget -O yelp_dataset.tar.gz "https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset?select=yelp_academic_dataset_business.json"
--2023-04-21 22:24:32-- https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset?select=yelp_academic_dataset_business.json
Resolving www.kaggle.com (www.kaggle.com)... 35.244.233.98
Connecting to www.kaggle.com (www.kaggle.com)|35.244.233.98|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'yelp_dataset.tar.gz'

yelp_dataset.tar.gz  [ <> ] 13.43K --.-KB/s in 0.003s
2023-04-21 23:24:38 (4.04 MB/s) - 'yelp_dataset.tar.gz' saved [13750]

gzip: stdin: not in gzip format
tar: Child returned status 1
tar: Error is not recoverable: exiting now
[ec2-user@ip-172-31-34-155 ~]$ cd /tmp
[ec2-user@ip-172-31-34-155 ~]$ wget -O yelp_dataset.tar.gz "https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset?select=yelp_academic_dataset_business.json"
--2023-04-21 23:25:14-- https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset?select=yelp_academic_dataset_business.json
Resolving www.kaggle.com (www.kaggle.com)... 35.244.233.98
Connecting to www.kaggle.com (www.kaggle.com)|35.244.233.98|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'yelp_dataset.tar.gz'

yelp_dataset.tar.gz  [ <> ] 13.43K --.-KB/s in 0.003s
2023-04-21 23:25:14 (4.34 MB/s) - 'yelp_dataset.tar.gz' saved [13750]

[ec2-user@ip-172-31-34-155 ~]$ aws configure
AWS Access key ID [None]: AKIA2H2AM63PD7WU0
AWS Secret Access Key [None]: Ww0ilbyh0xvvgj2Ny5sMuhz7E9NRL33Gvc2Yw7WU
Default region name [None]:
Default output format [None]:
[ec2-user@ip-172-31-34-155 ~]$ aws --version
aws-cli/2.9.19 Python/2.9.16 Linux/x86_64/amzn2023.x86_64 source/x86_64/amzn.2023 prompt/off
[ec2-user@ip-172-31-34-155 ~]$ client_loop: send disconnect: Connection reset by peer
varma@DESKTOP-ABQDQCF MINGW64 ~/Downloads
$

```

Next we have created an IAM user and in that we have created a group with s3 full access permissions so that we can easily configure the data from S3 bucket using EC2 instance .

```

[root@ip-172-31-34-155 ec2-user]# wget -O yelp_dataset.tar.gz "https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset?select=yelp_academic_dataset_business.json"
--2023-04-21 23:25:14-- https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset?select=yelp_academic_dataset_business.json
Resolving www.kaggle.com (www.kaggle.com)... 35.244.233.98
Connecting to www.kaggle.com (www.kaggle.com)|35.244.233.98|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'yelp_dataset.tar.gz'

```

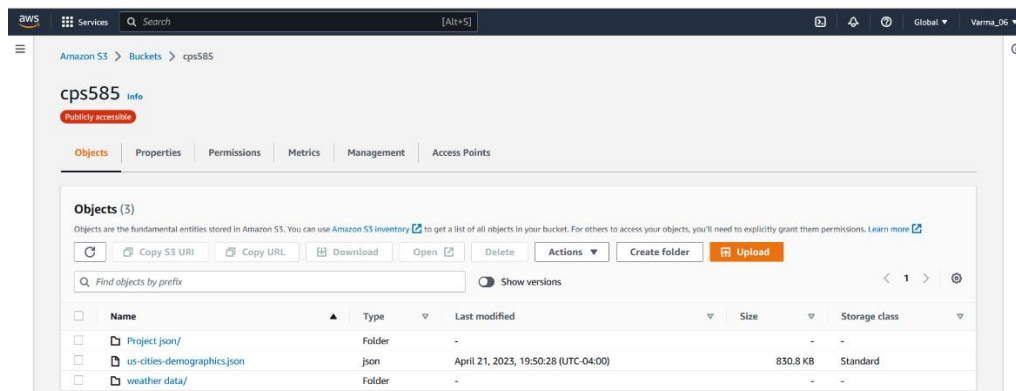
To read our file's we have used following commands :

```
wget -O yelp_dataset.tar.gz "[our_download_link]" (kaggle's path)
```

```
tar -xvzf yelp_dataset.tar.gz
```

tar command is generally used to unzip the file and read it to configure data but we have initially unzipped the datasets file so we skipped using tar command .

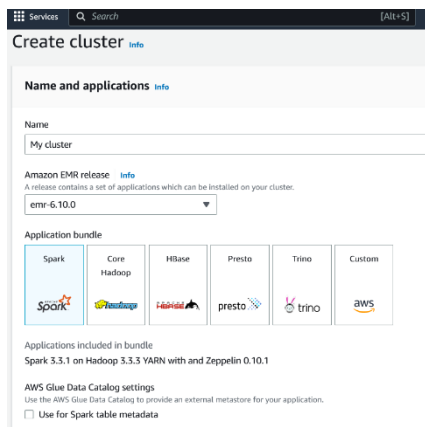
NXT, we have also Downloaded the JSON file from U.S. City Demographic Data and Uploaded it to a separate folder in our S3 bucket. Along with that we have also Download the whole dataset from Historical Hourly Weather Data 2012-2017 Unzipped and uploaded city_attributes.csv, temperature.csv, and weather_description.csv files to a separate folder (weather_dataset) in our S3 bucket.



Amazon EMR

To Configure and create our EMR cluster. We have managed to advanced options, and enabled Apache Spark, Livy and AWS Glue Data Catalog for Spark.

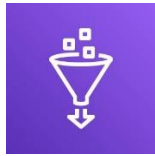
To make python default here we have to follow this JSON configuration:



```
[
  {
    "Classification": "spark-env",
    "Configurations": [
      {
        "Classification": "export",
        "Properties": {
          "PYSPARK_PYTHON":
            "/usr/bin/python3"
        }
      }
    ]
  }
]
```


To run these we have to Go to EC2 Security Groups, and select master node and enable inbound connections to 8998.

Extract – Amazon glue extracting data from S3 bucket



Here in this Extract before getting data path from s3 bucket we have imported some python sdk like boto3 and executed code whether the data in s3 bucket is reverting or not here is the snippet.

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [0]: def write_to_s3(file, df, s3_client):
        dir_path = file.split('/')[0:-2]
        file_name_suffix = str(uuid.uuid1())
        df.to_json(
            f's3://itvypwps/yelp-dataset-json/{dir_path}/part-{file_name_suffix}.gz',
            orient='records',
            lines=True
        )

In [10]: import glob
import boto3
s3_client = boto3.client('s3')
dfs = []
files = glob.glob('./data/yelp-dataset-json/*/*.json', recursive=True)
s3_client = boto3.client('s3')
for file in files:
    json_reader = pd.read_json(file, lines=True, chunksize=100000)
    for chunk_id, df in enumerate(json_reader):
        print(f'Processing chunk {chunk_id} in {file}')
        write_to_s3(file, df, s3_client)

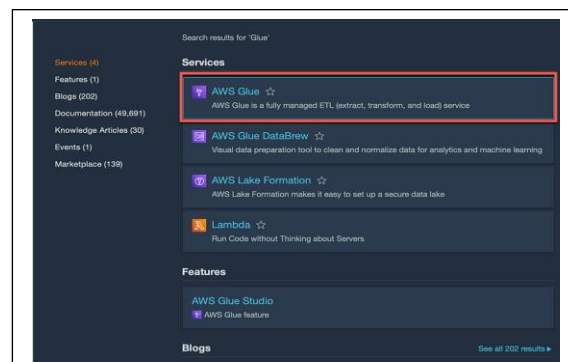
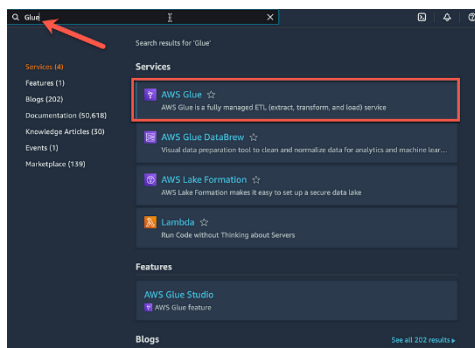
In [17]: import os
import glob
os.makedirs('./data/yelp-dataset-json', exist_ok=True)
base_dir = 'Users\\Varma06\\Downloads\\archive'
for file in glob.glob(f'{base_dir}/*.json'):
    file_name = file.split('/')[-1]
    print(f'Moving {file} from downloads to ../data/yelp-dataset-json/{file_name.split(".")[0]}')
    data_dir = f'../data/yelp-dataset-json/{file_name.split(".")[0]}'
    os.makedirs(data_dir, exist_ok=True)
    os.rename(file, f'{data_dir}/{file_name}')

In [66]: import pandas as pd
file_path = '../Data/yelp_academic_dataset_review/yelp_academic_dataset_review.json'
df = pd.read_json(file_path, lines=True, nrows=100)
```

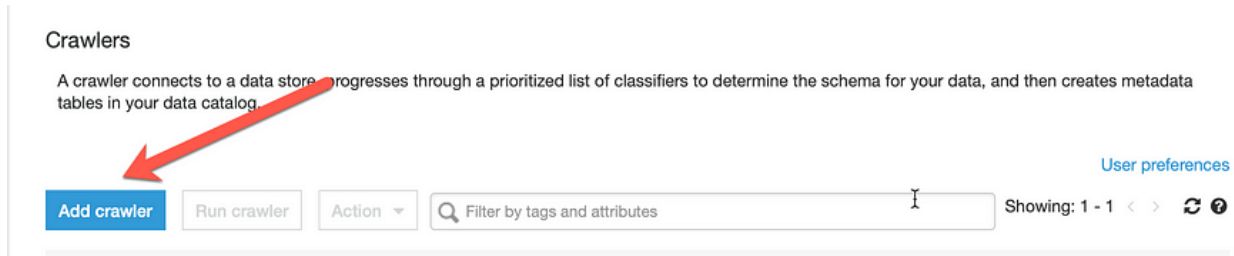
To validate this we have to run below command in AWS CLI or git bash or Command prompt.

```
aws s3 ls
s3:// s3://cps585/archive/
https://cps585.s3.us-east-2.amazonaws.com/archive/
```

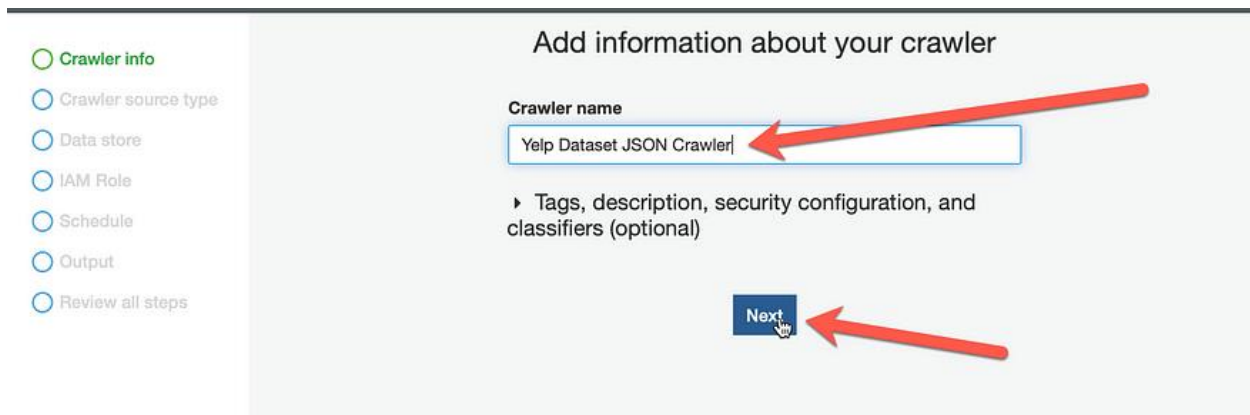
Step by Step process of amazon glue involvement:



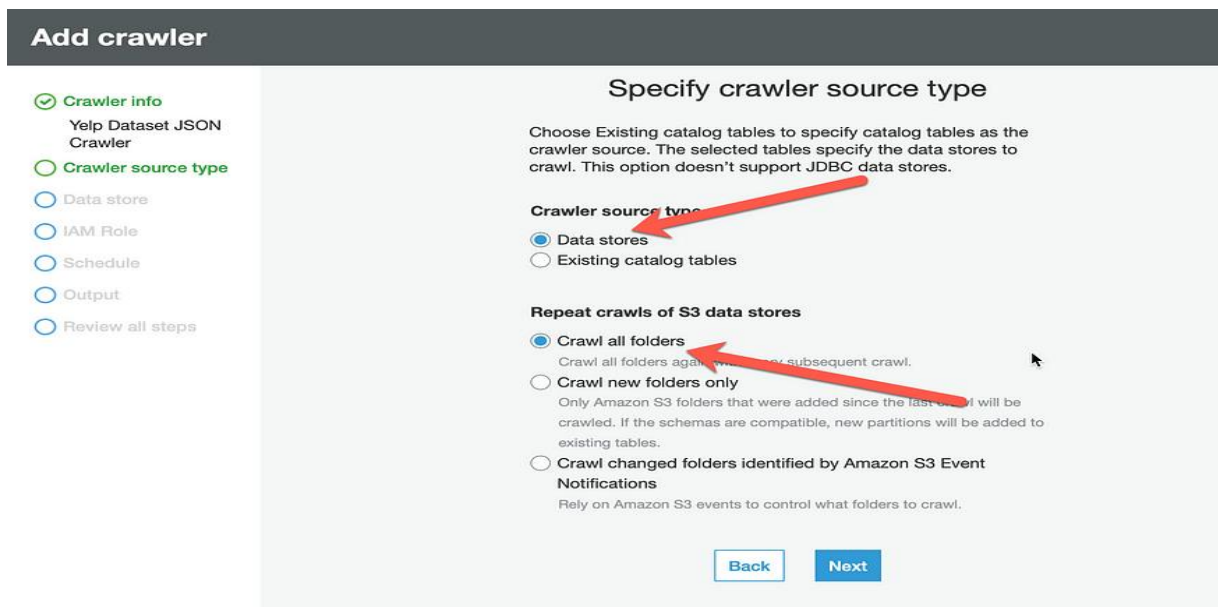
Re-open your AWS account and search for Amazon glue in your search box present at the top left of aws Page. Click on Aws glue and create an amazon Crawler with all the specifications requesting while creating a crawler.



Next;



Next;



Add crawler

Add a data store

Choose a data store: **S3**

Connection: **Select a connection**

Include path: **s3://yelp/yelp-dataset.json**

Sample size (optional): **Enter a number between 1 and 250**

Exclude patterns (optional):

Back **Next**

Crawlers

A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Crawler **Yelp Dataset JSON Crawler** was created to run on demand. **Run it now?**

Add crawler **Run crawler** **Action** Showing: 0 - 0

Add crawler

Crawler info
Yelp Dataset JSON Crawler

Crawler source type
Data stores

Data stores
S3: s3://yelp/yelp-dataset.json

IAM Role
arn:aws:iam::582945781536:role/service-role/AWSGlueServiceRole-yelp

Schedule
Run on demand

Output
Database: yelp_json_db
Prefix added to tables (optional):
Create a single schema for each S3 path (optional):
Table level (optional):
Configuration options

Back **Finish**

AWS Glue **Tables**

A table is the metadata definition that represents your data, including its schema. A table can be used as a source or target in a job definition.

Tables (7)

View and manage all available tables.

Name	Database	Location	Classification	Deprecated	View data
weather_data	data	-	-	-	-
weather_data	yelp_json_db	s3://s3s585/weather_data/	csv	-	Table data
yelp_academic_dataset_business	yelp_json_db	s3://s3s585/archive/Data/yelp_	json	-	Table data
yelp_academic_dataset_checkins	yelp_json_db	s3://s3s585/archive/Data/yelp_	json	-	Table data
yelp_academic_dataset_review	yelp_json_db	s3://s3s585/archive/Data/yelp_	json	-	Table data
yelp_academic_dataset_tip	yelp_json_db	s3://s3s585/archive/Data/yelp_	json	-	Table data
yelp_academic_dataset_user	yelp_json_db	s3://s3s585/archive/Data/yelp_	json	-	Table data

AWS Glue Crawler is a service provided by Amazon Web Services (AWS) that automatically discovers and extracts metadata from various data sources such as Amazon S3, JDBC databases, and other cloud-based or on-premises data stores. The extracted metadata includes information such as the schema of the data, the format, and the location.

To create a crawler, you need to provide a name for the crawler, specify the data source type and location, and configure any necessary credentials for accessing the data source. You can also set

up a schedule for the crawler to run automatically, and specify any custom classifiers or other options.

Once you have created a crawler, you can use it to generate tables in the AWS Glue Data Catalog, which is a central metadata repository for all your data assets in AWS. The tables contain the schema and metadata of the data that the crawler has discovered, and can be used by other AWS services such as Amazon Athena, Amazon Redshift, and Amazon EMR for querying and analysis.

To generate tables using a crawler, you need to run the crawler and specify the target database and table name in the AWS Glue Data Catalog. The crawler will then automatically create the table and populate it with the metadata it has extracted from the data source. You can also customize the table schema and metadata if needed.

Transform – via Amazon Athena ← Amazon Glue



Amazon Athena is used for querying data stored in Amazon S3 using SQL statements without having to set up any infrastructure or manage any servers. It allows users to analyze and extract insights from large amounts of data stored in S3 without the need for complex ETL processes or data warehousing solutions.

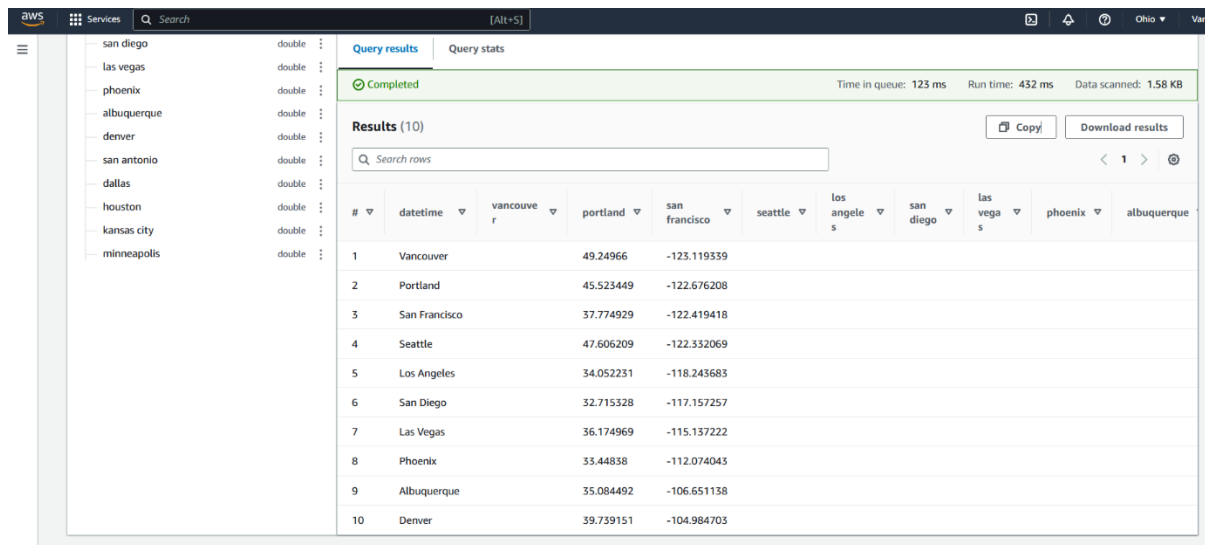
Some of the key benefits of using Amazon Athena include:

1. **Serverless:** Amazon Athena is a serverless service, which means that users do not have to provision, configure, or manage any infrastructure. This makes it easy to get started with querying data in S3 without any upfront costs or complex setup.
2. **Scalability:** Amazon Athena is highly scalable and can handle large volumes of data with ease. It uses a distributed query engine that can parallelize and optimize queries across a cluster of nodes, which enables it to process large datasets quickly.
3. **Cost-effective:** Amazon Athena is a pay-per-query service, which means that users only pay for the queries they run. This makes it a cost-effective solution for ad hoc analysis or occasional queries.
4. **Familiar interface:** Amazon Athena uses standard SQL syntax, which makes it easy for users who are familiar with SQL to get started with querying data in S3.
5. **Integration with AWS Glue:** Amazon Athena integrates seamlessly with AWS Glue, which provides metadata management and ETL capabilities for data stored in S3. This makes it easy to discover and access data stored in S3 and to transform it into the desired format for analysis.

Overall, Amazon Athena is a powerful and flexible solution for analyzing data stored in Amazon S3, and it can help organizations extract valuable insights from their data quickly and easily.

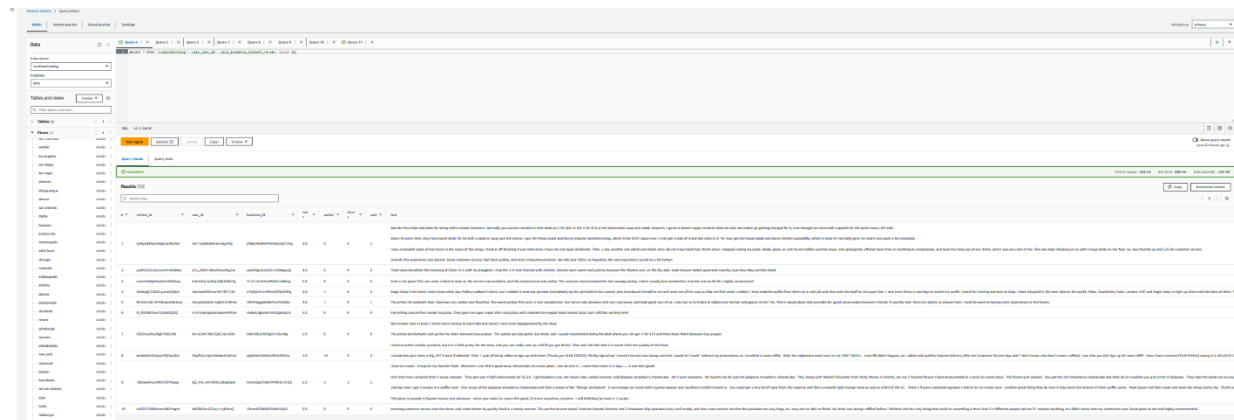
Interacting with DB queries to extract data from amazon glue crawler :

```
SELECT * FROM "AwsDataCatalog"."yelp_json_db"."weather_data" limit 10;
```



#	datetime	vancouver	portland	san francisco	seattle	los angeles	san diego	las vegas	phoenix	albuquerque
1	Vancouver	49.24966	-123.119339							
2	Portland	45.523449	-122.676208							
3	San Francisco	37.774929	-122.419418							
4	Seattle	47.606209	-122.332069							
5	Los Angeles	34.052231	-118.243683							
6	San Diego	32.715328	-117.157257							
7	Las Vegas	36.174969	-115.137222							
8	Phoenix	33.44838	-112.074043							
9	Albuquerque	35.084492	-106.651138							
10	Denver	39.739151	-104.984703							

```
SELECT * FROM "AwsDataCatalog"."yelp_json_db"."yelp_academic_dataset_review" limit 10;
```



#	datetime	vancouver	portland	san francisco	seattle	los angeles	san diego	las vegas	phoenix	albuquerque
1	Vancouver	49.24966	-123.119339							
2	Portland	45.523449	-122.676208							
3	San Francisco	37.774929	-122.419418							
4	Seattle	47.606209	-122.332069							
5	Los Angeles	34.052231	-118.243683							
6	San Diego	32.715328	-117.157257							
7	Las Vegas	36.174969	-115.137222							
8	Phoenix	33.44838	-112.074043							
9	Albuquerque	35.084492	-106.651138							
10	Denver	39.739151	-104.984703							

```
SELECT * FROM "AwsDataCatalog"."yelp_json_db"."yelp_academic_dataset_business" limit 10;
```

```
SELECT * FROM "AwsDataCatalog"."yelp_json_db"."yelp_academic_dataset_checkin" limit 10;
```

```
SELECT * FROM "AwsDataCatalog"."yelp_json_db"."yelp_academic_dataset_tip" limit 10;
```

```
SELECT * FROM "AwsDataCatalog"."yelp_json_db"."yelp_academic_dataset_user" limit 10;
```

Like wise all tables have been Successfully converted in to tabular format extracting all json files and converted to csv and downloaded successfully.

Some code snippets we worked on during data ingestion for amazon glue and redshift:

Extension for Amazon Redshift:



To load Parquet data into Redshift, we can use spark-redshift or an AWS Glue job. However, it's better to define tables manually for better control over data quality, consistency, and performance. This involves issuing SQL statements to first create tables and then copy data. AWS Glue's data catalog can help derive the correct data types to make this process easier and more transparent.

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4
5
6
7 from pyspark.sql import functions as F
8 from pyspark.sql import types as T
9 from pyspark.sql import Window, Row
10
11
12
13 # In[1]:
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

68 business_attribute_path = staging_dir + "business_attributes"
69 active_path = staging_dir + "active"
70 address_path = staging_dir + "address"
71 category_path = staging_dir + "category"
72 business_category_path = staging_dir + "business_category"
73 business_hour_path = staging_dir + "business_hour"
74 business_name_path = staging_dir + "business_name"
75 active_path = staging_dir + "active"
76 user_path = staging_dir + "user"
77 show_page_path = staging_dir + "show_page"
78 from_page_path = staging_dir + "from_page"
79 meeting_path = staging_dir + "meeting"
80 tips_path = staging_dir + "tips"
81 phone_path = staging_dir + "phone"
82 city_meeting_path = staging_dir + "city_meeting"
83
84 # test()
85
86
87 def main(self):
88     # the total number of records
89     print("_____")
90     print("count")
91     print(df.count())
92
93     # the number of null values for each column
94     print("_____")
95     print("nulls")
96     for col in df.select(df.count().limit(1), df.alias('c')).collect():
97         print(col)
98
99     # the maximum number of chars for each string column needed for value definition (is feasible)
100     print("_____")
101     print("max length")
102     print([(k, df.select(df.alias('maxf_length(b)')).filter(df.k == v, v in df.columns if v.startswith('string'))
103             .first().as_string())
104            for k, v in df.columns if v.startswith('string')])])
105
106     # Print schema
107     print("_____")
108     print("schema")
109     df.printSchema()
110
111     # Print first record
112     print("_____")
113     print("first row")
114     df.first().printSchema()
115
116
117

```

```
def parse_boolean(x):
    # Convert boolean strings to native boolean format
    if x is None or x == 'None':
        return None
    if x == 'True':
        return True
    if x == 'False':
        return False

parse_boolean_udf = F.udf(parse_boolean, T.BooleanType())

bool_attr = {
    "acceptInsurance",
    "sm",
    "likeFaking",
    "businessAcceptBitonic",
    "businessAcceptBitonicCuts",
    "byAppointmentOnly",
    "caters",
    "checkCheck",
    "corkage",
    "dogsAllowed",
    "driveThru",
    "goodForDancing",
    "goodForKids",
    "happyHour",
    "hasTV",
    "open24Hours",
    "outdoorSeating",
    "restaurantCounterService",
    "restaurantsDelivery",
    "restaurantsGoodForGroups",
    "restaurantsHaveAlcohol",
    "restaurantsTableService",
    "restaurantsTakeOut",
    "wheelchairAccessible"
}

for attr in bool_attr:
    business_attributes_df = business_attributes_df.withColumn(attr, parse_boolean_udf(attr
    # In[17]:

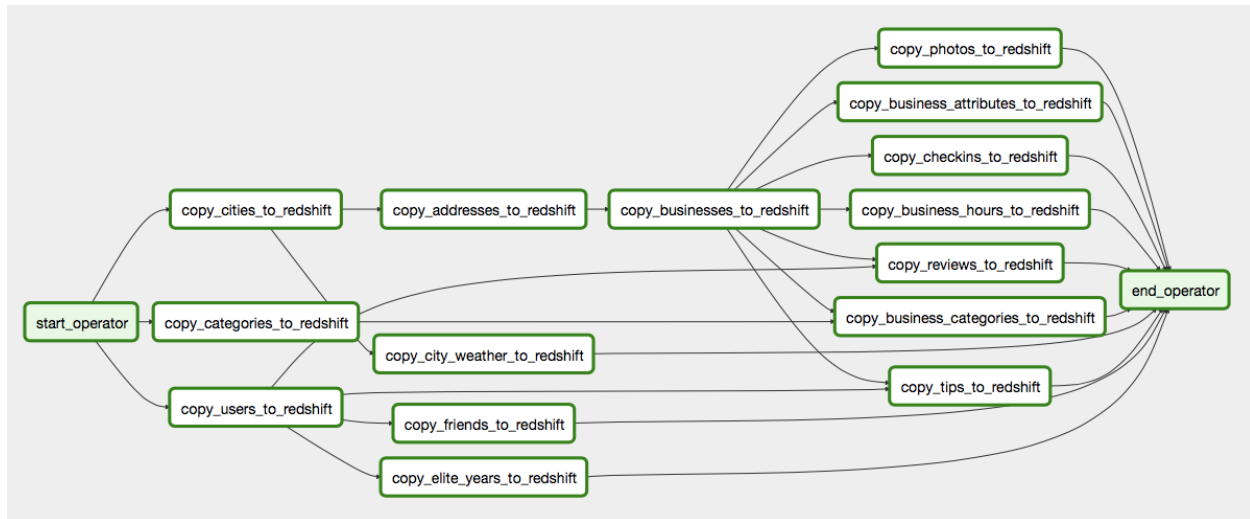
business_attributes_df.select("acceptInsurance").distinct().collect()
```

```

2   "cell": {
3       {
4           "cell_type": "code",
5           "execution_count": 1,
6           "metadata": {},
7           "outputs": [
8               {
9                   "name": "stdout",
10                  "output_type": "stream",
11                  "text": [
12                      "Starting spark application\n"
13                  ]
14              },
15              {
16                  "data": {
17                      "text/html": [
18                          "SparkUI Application ID=<th>Kind</th><th>State</th><th>Spark UI</th><th>Driver log</th><th>Current session?</th><th>tz</th><th>cd</th><th>cd:application_165067330247_0001</th><th>cd:pd
19                          \n"
20                      ]
21                  },
22                  "text/plain": [
23                      "If Python code displays HTML object"
24                  ]
25              },
26              "metadata": {},
27              "output_type": "display_data"
28          ],
29          {
30              "name": "stdout",
31              "output_type": "stream",
32              "text": [
33                  "SparkSession available as 'spark'.\n"
34              ]
35          },
36          "source": [
37              "from pyspark.sql import functions as F\n",
38              "from pyspark.sql import types as T\n",
39              "from pyspark.sql import Window, Row"
40          ],
41          },
42          {
43              "cell_type": "code",
44              "execution_count": 2,
45              "metadata": {},
46              "outputs": [

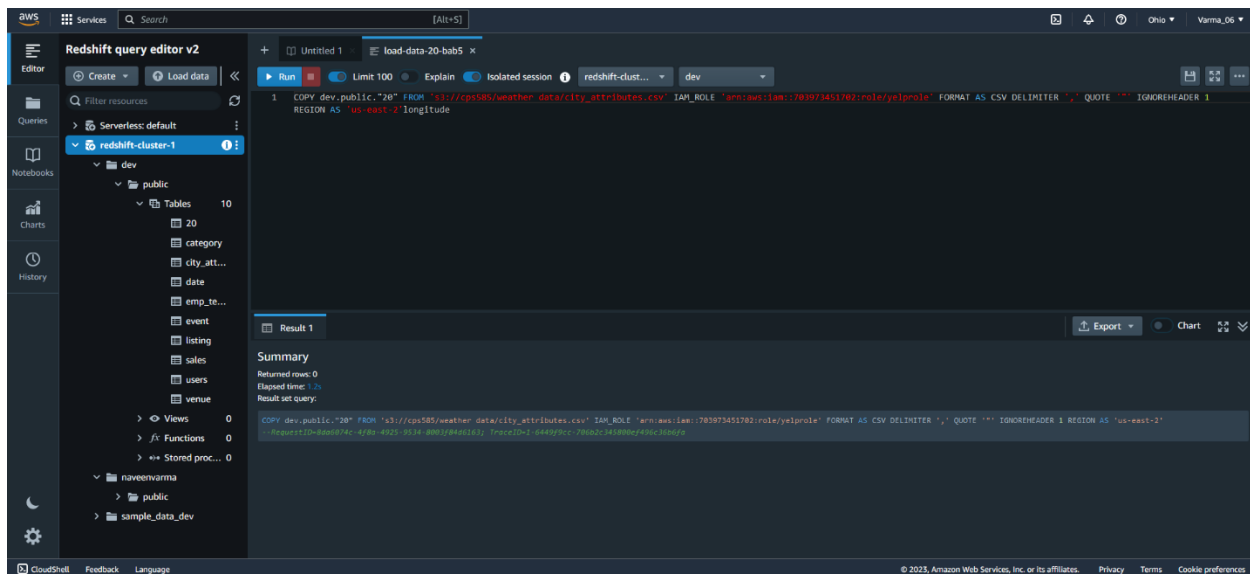
```

copy_to_redshift



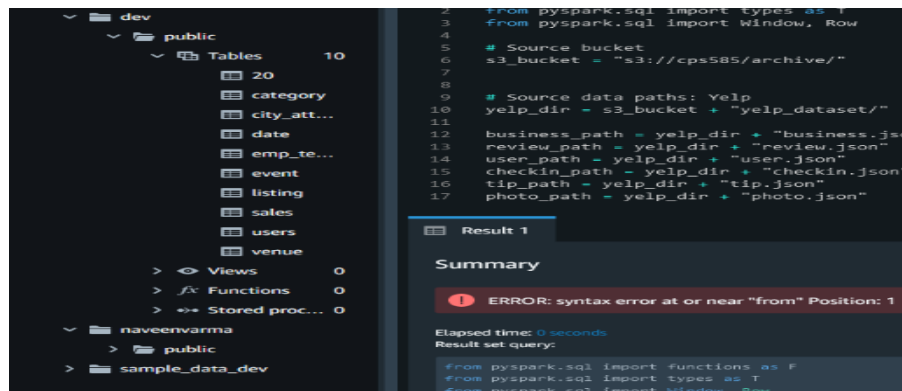
To load data from an S3 bucket into Amazon Redshift,

1. We have Created an IAM role that has permissions to read from the S3 bucket and write to the Redshift cluster. This role will be used by the Redshift cluster to access the S3 bucket.
2. We also Launched a Redshift cluster with the appropriate configuration settings. You can specify the number of nodes, node type, and other configuration options based on your requirements.
3. Created a table in Redshift that matches the structure of the data you want to load from the S3 bucket. This table will be used to store the data after it is loaded into Redshift.

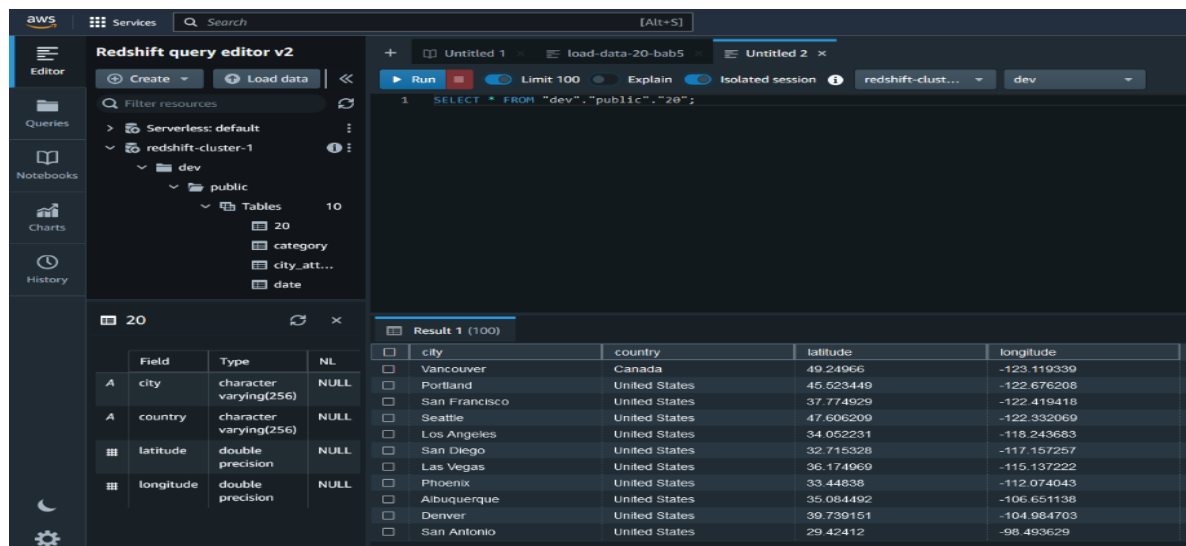


Successfully connected to redshift via creating redshift cluster-1 and allowing instances.

After the data is loaded into Redshift, you can use SQL queries to analyze and transform the data as needed.



Overcome:



➤ Datawarehouse is Successfully created on Amazon Redshift

Section 4: CONCLUSION

Yelp is a popular online platform that allows users to search and discover local businesses, read and write reviews, and find deals. To support its data-intensive operations, Yelp uses a modern data infrastructure that leverages various AWS services such as S3, Redshift, Glue, and Athena.

The data in Yelp's system is stored in S3, which provides a highly scalable and durable object storage service. To enable fast and efficient querying of this data, Yelp uses Redshift, a fully managed data warehouse that is optimized for analytics workloads. Redshift provides a fast, scalable, and cost-effective solution for querying and analysing large volumes of data.

To ensure that its data is properly structured and organized, Yelp uses a normalized data model that adheres to third normal form (3NF) principles. This means that data is broken down into smaller, atomic units and organized into separate tables to minimize redundancy and improve data integrity. Glue, an ETL service, is used to extract, transform, and load data into Redshift from S3 in a way that maintains the 3NF structure.

Athena, a serverless query service, is also used by Yelp to provide ad hoc querying and analysis of data in S3. Athena enables users to query data using standard SQL syntax, without the need for any infrastructure or setup.

Overall, Yelp's data infrastructure using S3, Redshift, Glue, and Athena provides a highly scalable, flexible, and cost-effective solution for managing and analysing large volumes of data. By using a 3NF data model, Yelp ensures that its data is organized and structured in a way that maximizes data integrity and minimizes redundancy. This enables Yelp to extract valuable insights from its data, which it can use to improve its services and enhance the user experience.

APPENDIX A: WHAT I DID & LEARNED

Naveen Varma Pandeti:

As a team member my part of contribution to the project is that I have collected all data sets from different sources that I've mentioned in project proposal and extracted those JSON files and loaded in to Amazon S3 bucket and created an ec2 instance and emr cluster to connect with command line terminal (i.e git bash) and successfully configured data from S3 bucket. Created IAM user(naveen) with Amazon S3FullAccess. Generated Access key and secret key to login to Aws through AWS CLI. Created the IAM Role with AmazonS3FullAccess and attached that role to the EC2 instance. So that the users who login to that instance have also permissions to do anything with S3 buckets as well. Later on I've implemented performing all Extract - Load - Transform with Amazon glue, Athena. Worked on Python scripts for establishing Athena and S3 clients, designing a star schema data model, and configuring AWS services such as S3 buckets, IAM roles, Data base connections and security groups were all part of the job. Almost all I've involved in whole project pin to pin. Finally I've also attained all the results from the loaded data through created Crawler in Amazon glue and executing them in Amazon Athena.

There are various things that I've learnt from Aws S3, Apache spark, Amazon Glue, Amazon Athena, Redshift, Ec2 instances, cluster creation and connection etc. Yelp is a large database with a significant amount of data. We can make sure that the database can scale quickly to handle the expanding amount of data by utilizing S3, Spark, and Redshift. The main information I would like

to share is that in every stage of our project I was continuously facing errors one on one like schema not loaded in database (redshift), Fatal – password authentication error, although without fail I've kept working on multiple times and finally overcame all the errors that have encountered.

Overall, the combination of normalizing the Yelp database and using AWS services like S3, Spark, Glue, and Redshift can provide valuable insights and help businesses make informed decisions based on the data stored in the database.

Sumanth Reddy Busireddy:

Working on this project was a kind of a new experience for me. I had zero knowledge on the aws environment. Now I am good at using AWS Glue, AWS Athena, and AWS Redshift, which can provide valuable learning opportunities in a variety of areas. Firstly, I have learnt how efficiently we can ingest large amounts of data from various sources into your data lake using AWS Glue. Next, I have learned how to transform and prepare your data for analysis using AWS Glue's powerful data processing capabilities. Later I have also learned how effectively manage metadata for your data using AWS Glue's Data Catalog.

In addition, I have also learned how to run SQL queries on your data lake using AWS Athena, which can help you to analyse and visualize your data and gain insights. Furthermore, I have learned how to build a highly scalable data infrastructure that can handle large amounts of data and scale as your data grows. I have also learned how to automate your data processing workflows using AWS Glue's job scheduling and automation features, which can save you time and increase efficiency in your data engineering projects.

Overall, working on an AWS data engineering project using AWS Glue, AWS Athena, and AWS Redshift has taught me valuable skills in data ingestion, preparation, querying, warehousing, scalability, and automation. These skills can be applied to a wide range of data engineering projects and can help you to build more effective and efficient data infrastructure for your organization.

Sri Swetha Kanduri:

Working on this project helped me to gain more knowledge technically. Initially, all three of us in our team put in great effort in coming up with the project. This was the time when we explored different tools that we are not familiar with. I collaborated with my teammate in collecting data from different sources and loading it into Amazon S3. I have worked on creating some clusters and connecting their path in representing fields to perform ETL operations mainly.

Worked on issues and errors in Python scripts for establishing Athena and S3 clients. Creating a separate database to perform amazon redshift operations and S3 data from Athena tables was the main part of my work. I also contributed to the presentation part and report creation. We were struck at a point in data transformation while converting JSON files into data frames which I took care of. Overall working on this helped me to increase my knowledge of AWS tools and Amazon S3. Clearly understood phases of data engineering like ingestion, transformation, extraction, and storing.

APPENDIX B: References:

<https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset>

<https://www.yelp.com>

<https://aws.amazon.com/getting-started/>

https://docs.aws.amazon.com/ec2/index.html?nc2=h_q1_doc_ec2

<https://www.cloudmantra.net/blog/aws-glue-simple-flexible-and-cost-effective-etl/>

<https://youtu.be/ppDx5Fmvgsk>

<https://youtu.be/EetkEf359QE>

<https://youtu.be/JyQ9EFFR3n8>

<https://youtu.be/kn2GdPcU-dU>

<https://youtu.be/92CgVMi7OTw>