

# Data Engineering COVID19 Project

Initially, we obtained the dataset from an AWS-managed data lake. The data was preprocessed and structured by AWS, featuring well-defined relationships between the entities within the dataset, along with a comprehensive data model.

**After obtaining the dataset, the data is stored in AWS S3. It is then crawled using the AWS Glue Crawler, which catalogs and organizes the data. The data is processed in AWS Athena for querying and transformation. After processing, the transformed data is loaded into Amazon Redshift, with a VPC set up around Redshift for secure and isolated access.**

The preliminary task involves storing all the datasets in specific folders within an Amazon S3 bucket. The datasets are organized by their categories, and only CSV format data is used for processing.

The screenshot shows the AWS Management Console for the 'surya-covid-19-data' bucket. The 'Objects' tab is selected, showing a list of 7 folders. The folders are: enigma-jhud/, enigma-nytimes-data-in-usa/, output/, packages/, rearc-covid-19-testing-data/, rearc-usa-hospital-beds/, and static-datasets/. The console includes a search bar, a list of actions (Copy S3 URI, Copy URL, Download, Open, Delete, Actions, Create folder, Upload), and a table of objects.

| Name                         | Type   | Last modified | Size | Storage class |
|------------------------------|--------|---------------|------|---------------|
| enigma-jhud/                 | Folder | -             | -    | -             |
| enigma-nytimes-data-in-usa/  | Folder | -             | -    | -             |
| output/                      | Folder | -             | -    | -             |
| packages/                    | Folder | -             | -    | -             |
| rearc-covid-19-testing-data/ | Folder | -             | -    | -             |
| rearc-usa-hospital-beds/     | Folder | -             | -    | -             |
| static-datasets/             | Folder | -             | -    | -             |

Currently, we are working under an IAM user created by the root user. Specific policies are assigned to this IAM user to control access. Additionally, an IAM role was created to facilitate access. The permissions for this IAM role are inherited by the root user, granting full access to Amazon S3, the AWS Glue Console, and AWS Glue services.

The screenshot displays the AWS IAM console interface for the 's3-glue-role'. The breadcrumb navigation shows 'IAM > Roles > s3-glue-role'. The role's description is 'Allows Glue to call AWS services on your behalf.' The 'Summary' section provides details: Creation date (January 22, 2025, 14:12 UTC-05:00), Last activity (2 hours ago), ARN (arn:aws:iam::741448924843:role/s3-glue-role), and Maximum session duration (1 hour). The 'Permissions' tab is active, showing 'Permissions policies (3)' which includes 'AmazonS3FullAccess', 'AWSGlueConsoleFullAccess', and 'AWSGlueServiceRole', all of which are AWS managed. The 'Permissions boundary' is noted as '(not set)'. At the bottom, there is a section to 'Generate policy based on CloudTrail events'. The footer of the console shows 'CloudShell', 'Feedback', and copyright information for Amazon Web Services, Inc.

Next, we create a crawler to scan the datasets stored in the S3 buckets. The dataset contains around 10 tables with various types of entities. When creating the crawler, we specify the S3 bucket as the data source.

Surya Deip Reddy

us-east-2.console.aws.amazon.com

AWS Glue | us-e...

Search

[Option+S]

United States (Ohio)

surya-covid19-admin @ 7414-4892-4843

AWS Glue > Crawlers > Edit crawler

Q

Step 1

Set crawler properties

Step 2

Choose data sources and classifiers

Step 3

Configure security settings

Step 4

Set output and scheduling

Step 5

Review and update

Review and update

Step 1: Set crawler properties

Edit

Set crawler properties

| Name        | Description | Tags |
|-------------|-------------|------|
| enigma-jhud | -           | -    |

Step 2: Choose data sources and classifiers

Edit

Data sources (1) Info

The list of data sources to be scanned by the crawler.

| Type | Data source                           | Parameters  |
|------|---------------------------------------|-------------|
| S3   | s3://surya-covid-19-data/enigma-jhud/ | Recrawl all |

Step 3: Configure security settings

Edit

Configure security settings

| IAM role     | Security configuration | Lake Formation configuration |
|--------------|------------------------|------------------------------|
| s3-glue-role | -                      | -                            |

Step 4: Set output and scheduling

Edit

Set output and scheduling

| Database      | Table prefix - optional | Maximum table threshold - optional | Schedule  |
|---------------|-------------------------|------------------------------------|-----------|
| surya-covid19 | -                       | -                                  | On demand |

Cancel

Previous

Update

CloudShell

Feedback

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Surya Deip Reddy

us-east-2.console.aws.amazon.com

AWS Glue | us-e...

Search

[Option+S]

United States (Ohio)

surya-covid19-admin @ 7414-4892-4843

AWS Glue > Crawlers > Edit crawler

Q

Step 1

Set crawler properties

Step 2

Choose data sources and classifiers

Step 3

Configure security settings

Step 4

Set output and scheduling

Step 5

Review and update

Review and update

Step 1: Set crawler properties

Edit

Set crawler properties

| Name  | Description | Tags |
|---|-------------|------|
| rearc-covid-19-testing-data-us-total-latest | -           | -    |

Step 2: Choose data sources and classifiers

Edit

Data sources (1) Info

The list of data sources to be scanned by the crawler.

| Type | Data source  | Parameters  |
|------|--|-------------|
| S3   | s3://surya-covid-19-data/rearc-covid-19-testing-dat... | Recrawl all |

Step 3: Configure security settings

Edit

Configure security settings

| IAM role     | Security configuration | Lake Formation configuration |
|--------------|------------------------|------------------------------|
| s3-glue-role | -                      | -                            |

Step 4: Set output and scheduling

Edit

Set output and scheduling

| Database      | Table prefix - optional      | Maximum table threshold - optional | Schedule  |
|---------------|------------------------------|------------------------------------|-----------|
| surya-covid19 | rearc_covid_19_testing_data_ | -                                  | On demand |

Cancel

Previous

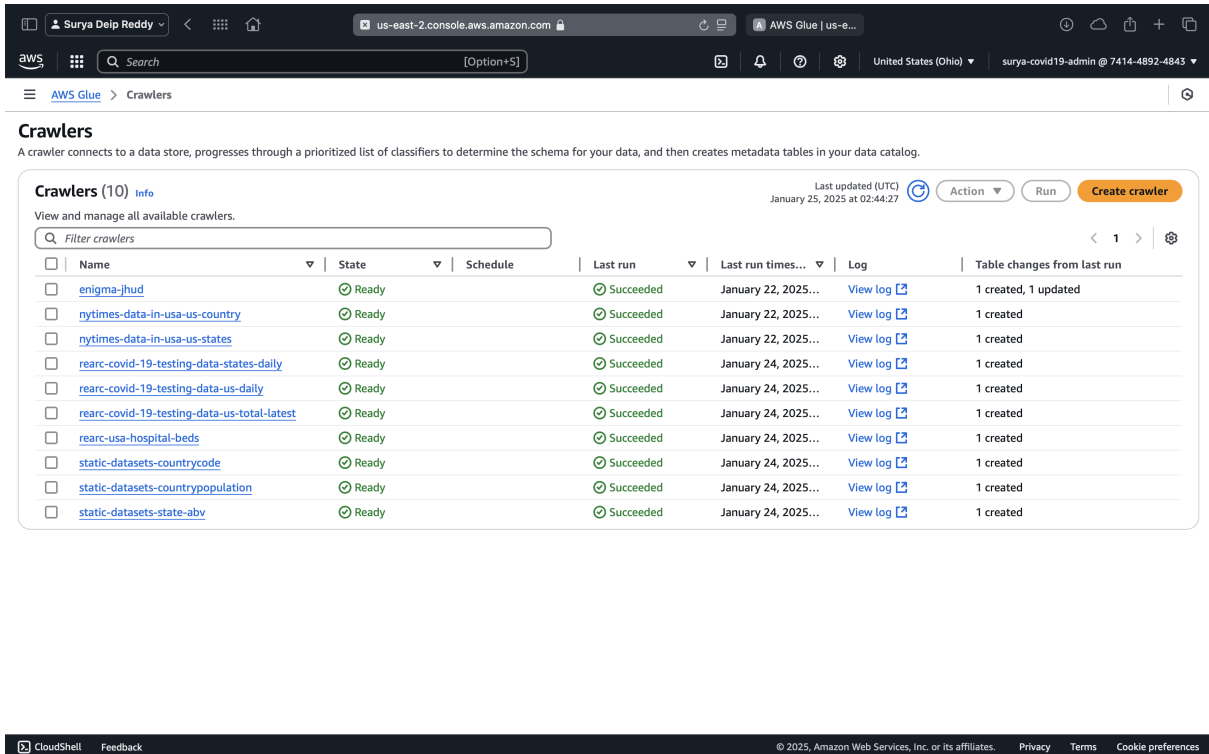
Update

CloudShell

Feedback

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Once all the crawlers are created and successfully executed, a database must be created in AWS Athena to store all the tables. After creating the database, the crawler is run to populate it with the necessary data.



**Crawlers (10)** [Info](#)

View and manage all available crawlers.

Filter crawlers

| Name  | State | Schedule | Last run  | Last run times...   | Log                      | Table changes from last run |
|---|-------|----------|-----------|---------------------|--------------------------|-----------------------------|
| <a href="#">enigma-jhud</a>                                 | Ready |          | Succeeded | January 22, 2025... | <a href="#">View log</a> | 1 created, 1 updated        |
| <a href="#">nytimes-data-in-usa-us-country</a>              | Ready |          | Succeeded | January 22, 2025... | <a href="#">View log</a> | 1 created                   |
| <a href="#">nytimes-data-in-usa-us-states</a>               | Ready |          | Succeeded | January 22, 2025... | <a href="#">View log</a> | 1 created                   |
| <a href="#">rearc-covid-19-testing-data-states-daily</a>    | Ready |          | Succeeded | January 24, 2025... | <a href="#">View log</a> | 1 created                   |
| <a href="#">rearc-covid-19-testing-data-us-daily</a>        | Ready |          | Succeeded | January 24, 2025... | <a href="#">View log</a> | 1 created                   |
| <a href="#">rearc-covid-19-testing-data-us-total-latest</a> | Ready |          | Succeeded | January 24, 2025... | <a href="#">View log</a> | 1 created                   |
| <a href="#">rearc-usa-hospital-beds</a>                     | Ready |          | Succeeded | January 24, 2025... | <a href="#">View log</a> | 1 created                   |
| <a href="#">static-datasets-countrycode</a>                 | Ready |          | Succeeded | January 24, 2025... | <a href="#">View log</a> | 1 created                   |
| <a href="#">static-datasets-countrypopulation</a>           | Ready |          | Succeeded | January 24, 2025... | <a href="#">View log</a> | 1 created                   |
| <a href="#">static-datasets-state-abv</a>                   | Ready |          | Succeeded | January 24, 2025... | <a href="#">View log</a> | 1 created                   |

CloudShell Feedback

© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

**Initial Data (S3) → Crawler → Athena → New S3 Bucket (to store processed data)**

Once all the crawlers are created and successfully executed, the data becomes available in AWS Athena, where it can be viewed and queried.

**Query 21**

```
1 SELECT * FROM "surya-covid19"."rearc_usa_hospital_beds" limit 10;
```

**Query results**

Completed Time in queue: 59 ms Run time: 479 ms Data scanned: 172.33 KB

**Results (10)**

| #  | objectid | hospital_name | hospital_type |
|----|----------|---------------|---------------|
| 1  |          |               |               |
| 2  |          |               |               |
| 3  |          |               |               |
| 4  |          |               |               |
| 5  |          |               |               |
| 6  |          |               |               |
| 7  |          |               |               |
| 8  |          |               |               |
| 9  |          |               |               |
| 10 |          |               |               |

**Query 22**

```
1 SELECT * FROM "surya-covid19"."enigma_jhud" limit 10;
```

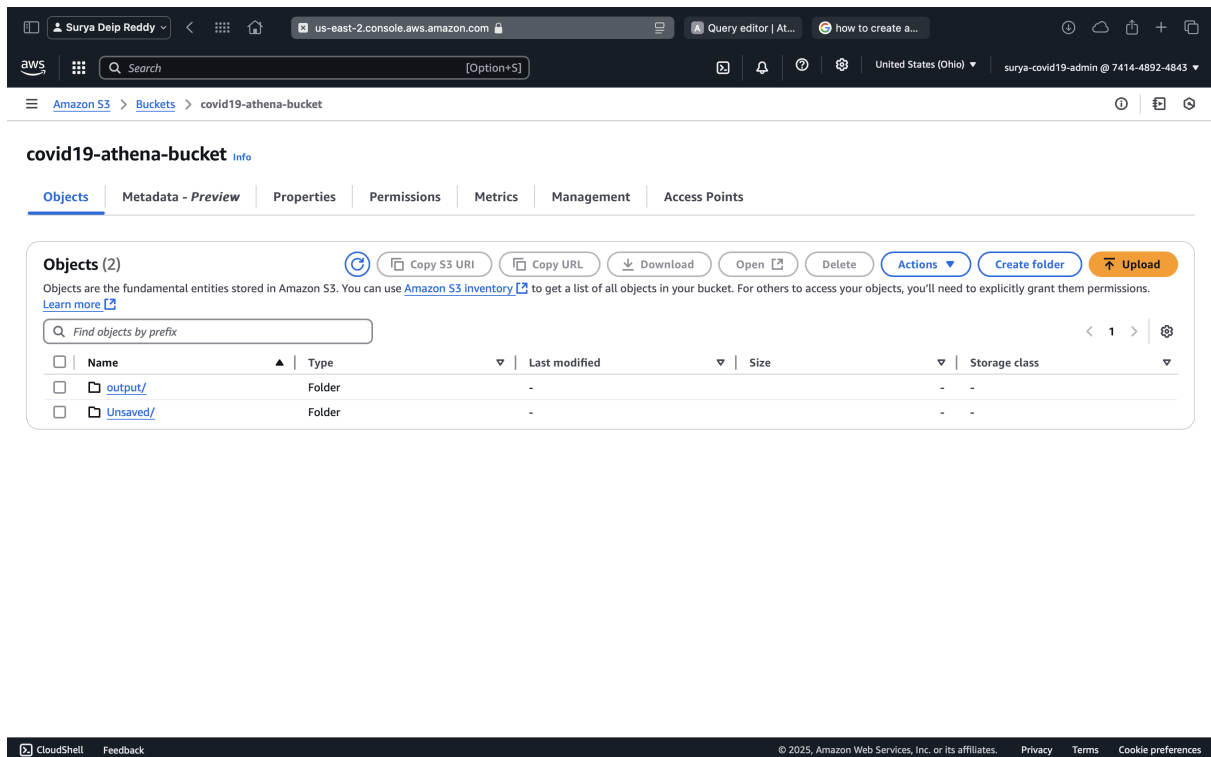
**Query results**

Completed Time in queue: 129 ms Run time: 1.152 sec Data scanned: 144.00 KB

**Results (10)**

| #  | fips | admin2 | province_state | country_region | last_update         | latitude | longitude | confirmed |
|----|------|--------|----------------|----------------|---------------------|----------|-----------|-----------|
| 1  |      |        | Anhui          | China          | 2020-01-22T17:00:00 | 31.826   | 117.226   | 1         |
| 2  |      |        | Beijing        | China          | 2020-01-22T17:00:00 | 40.182   | 116.414   | 14        |
| 3  |      |        |                |                |                     |          |           |           |
| 4  |      |        |                |                |                     |          |           |           |
| 5  |      |        |                |                |                     |          |           |           |
| 6  |      |        |                |                |                     |          |           |           |
| 7  |      |        |                |                |                     |          |           |           |
| 8  |      |        |                |                |                     |          |           |           |
| 9  |      |        |                |                |                     |          |           |           |
| 10 |      |        |                |                |                     |          |           |           |

In the Athena settings, a new S3 bucket is created to store the metadata and any performed actions on the CSV data. This bucket holds the processed data and associated metadata for future reference.



- A relational data model is created from the dataset to understand how data is interlinked within the entities.
- The relational data model is then connected to Athena to query the data.
- The ETL job is written in Python.
- The results of the ETL job are saved to an S3 bucket.
- Glue deployment is carried out to manage and execute the ETL tasks.
- Tables are built in Amazon Redshift.
- The next tasks involve copying the data to Redshift for further analysis and reporting.

[COVID19-DE-Project.ipynb](#)

## 1. Setting Up AWS Services

Importing Required Libraries

- boto3: AWS SDK for interacting with S3 and Athena
- pandas Library for data manipulation and analysis of tabular data

```
- StringIO: Handles in-memory string operations for CSV files
- time: Adds delays for AWS Athena query completion
- Dict: Type annotation from typing module for dictionary structures
```

## 2. Initialization of AWS Credentials and Client Setup

At the start of the script, several AWS credentials are defined. These credentials allow the script to access AWS services like Athena, S3, and Redshift. The configuration specifies the region (us-east-2) and various keys for accessing AWS resources.

- **AWS\_ACCESS\_KEY:** Your AWS access key used for authentication with AWS services.
- **AWS\_SECRET\_KEY:** Your AWS secret key, paired with the access key for secure authentication.
- **AWS\_REGION:** Specifies the AWS region where resources (e.g., Athena, S3) are hosted. In this case, it's us-east-2.
- **SCHEMA\_NAME:** Name of the Athena database schema being queried (surya-covid19).
- **S3\_STAGING\_DIR:** Path to the S3 directory where Athena query results are stored (s3://covid19-athena-bucket/output/).
- **S3\_BUCKET\_NAME:** Name of the S3 bucket used for storing query outputs and data (covid19-athena-bucket).
- **S3\_OUTPUT\_DIRECTORY:** Subdirectory inside the S3 bucket for storing results (output).

## 3. Initializing Athena Client

- **athena\_client:** Creates a connection to AWS Athena using the boto3 library.
- **"athena":** Specifies that the client will interact with the Athena service.
- **aws\_access\_key\_id:** The AWS access key for authentication.

- `aws_secret_access_key`: The corresponding AWS secret key for secure authentication.
- `region_name`: Specifies the AWS region where Athena is hosted (in this case, `us-east-2`)
- This client will be used to execute queries and interact with Athena.

#### 4. Download and Load Athena Query Results

This function checks if an Athena query is complete, downloads the results from S3, and loads them into a Pandas DataFrame.

##### Steps:

1. Check query status.
2. Download results from S3.
3. Load data into a DataFrame.

**Returns:** A Pandas DataFrame with the query results.

#### 5. Starting Athena Query Execution for Multiple Tables

The following steps are repeated for each table (e.g., `enigma_jhud`, `nytimes_data_in_usa_us_county`, etc.):

**Query Execution:** Executes a SQL query ( `SELECT * FROM <table_name> LIMIT 10000` ) on each table.

##### Query Configuration:

- `QueryExecutionContext` : Specifies the database ( `SCHEMA_NAME` ).
- `ResultConfiguration` : Defines where the results will be stored in S3 ( `S3_STAGING_DIR` ) and uses encryption ( `SSE_S3` ).



```
response = athena_client.start_query_execution(
    QueryString="SELECT * FROM <table_name> limit 10000",
    QueryExecutionContext={"Database": SCHEMA_NAME},
    ResultConfiguration={
        "OutputLocation": S3_STAGING_DIR,
        "EncryptionConfiguration": {"EncryptionOption": "SSE_S3"},
    },
)
```

## 6. Downloading and Displaying Athena Query Results for `enigma_jhud`

- **Downloading:** The results of the query are downloaded into the `enigma_jhud` DataFrame using the `download_and_load_query_results` function.
- **Displaying:** The `.head()` method is used to display the first few rows of the data for inspection.

```
enigma_jhud = download_and_load_query_results(athena_client, response)
enigma_jhud.head()
```

## 7. Creating Fact and Dimension Tables

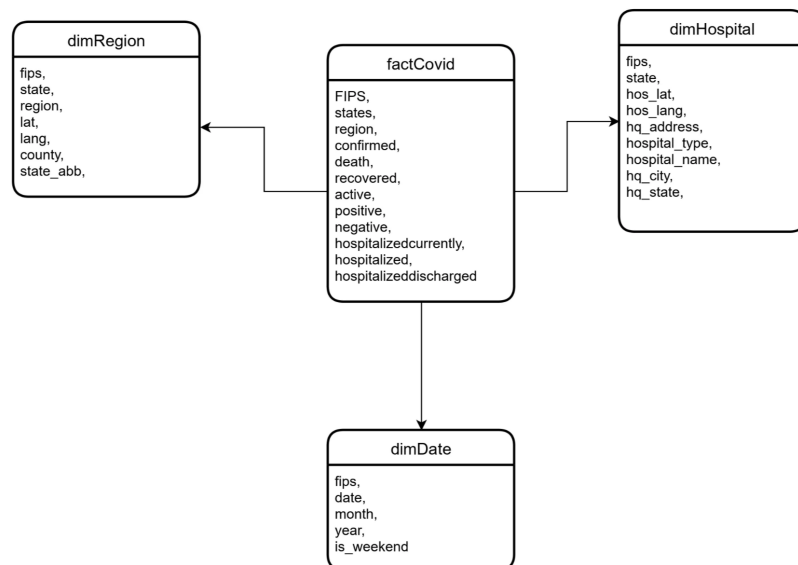
### Fact Table:

We created a **fact table** (`factCovid`) by selecting specific columns from the `enigma_jhud` and `rearc_covid_19_testing_data_states_daily` datasets:

- `factCovid_1`: Columns selected from `enigma_jhud` include `fips`, `province_state`, `country_region`, `confirmed`, `deaths`, `recovered`, `active`.
- `factCovid_2`: Columns selected from `rearc_covid_19_testing_data_states_daily` include `fips`, `date`, `positive`, `negative`, `hospitalizedcurrently`, `hospitalized`, `hospitalizeddischarged`.
- These are merged on the `fips` column using an **inner join**.

### Dimension Tables:

- **dimRegion:** Merged data from `enigma_jhud` and `nytimes_data_in_usa_us_county`. Contains columns like `fips`, `province_state`, `country_region`, `latitude`, `longitude`, `county`, and `state`.
- **dimHospital:** Merged data from `rearc_usa_hospital_beds`. Contains columns like `fips`, `state_name`, `latitude`, `longitude`, `hospital_name`, and `hospital_type`.
- **dimDate:** Extracted date-related data from `rearc_covid_19_testing_data_states_daily`. Includes columns like `fips`, `date`, `year`, `month`, and `day_of_week`.



## 8. Uploading Data to S3

The code saves the fact and dimension tables as CSV files and uploads them to an **S3 bucket** (`surya-covid-19-data`).

1. **Set S3 Bucket Name:** The bucket name is defined as `'surya-covid-19-data'`.

2. **Create In-Memory CSV Buffer:** A StringIO buffer is created to hold the CSV data temporarily.

3. **Convert DataFrames to CSV:** The factCovid, dimRegion, dimHospital, and dimDate DataFrames are converted to CSV format and saved in the in-memory buffer using to\_csv().

4. **Upload Data to S3:**

- A connection to S3 is established using boto3.resource().
- The data is uploaded to specific paths in the S3 bucket ('output/factCovid.csv', 'output/dimRegion.csv', etc.) using the put() method.

5. **Display CSV Content:**

- The content of the in-memory CSV buffer is printed with csv\_buffer.getvalue().

Data is saved in memory as CSV and uploaded to the S3 bucket for storage and further processing.

## 9. Generating SQL Schema for Tables

This section generates SQL `CREATE TABLE` statements for each of the tables (`dimDate`, `dimRegion`, `dimHospital`, `factCovid`):

- `dimDate`: SQL schema for the `dimDate` DataFrame.
- `dimRegion`: SQL schema for the `dimRegion` DataFrame.
- `dimHospital`: SQL schema for the `dimHospital` table (from `rearc_usa_hospital_beds` DataFrame).
- `factCovid`: SQL schema for the `factCovid` DataFrame.

These schemas can be used to create tables in a database like Redshift.

## 10. Connecting to Amazon Redshift

This part of the code connects to **Amazon Redshift** using the `redshift_connector` library:

1. **Install Connector:** The `redshift_connector` package is installed with `pip3`.
2. **Import:** The connector is imported to interact with Redshift.
3. **Connection:** A connection is established to the Redshift cluster by specifying:
  - **Host:** Redshift cluster endpoint.
  - **Database:** The Redshift database ( `dev` ).
  - **User and Password:** Credentials for authentication.
  - **SSL:** Ensures a secure connection.
4. **Autocommit:** Enables autocommit mode to automatically commit queries.
5. **Cursor:** Creates a cursor object to run SQL queries in the Redshift database.

## 11. Creating Tables in Redshift

This section creates the required **tables** in **Amazon Redshift** to store the fact and dimension data.

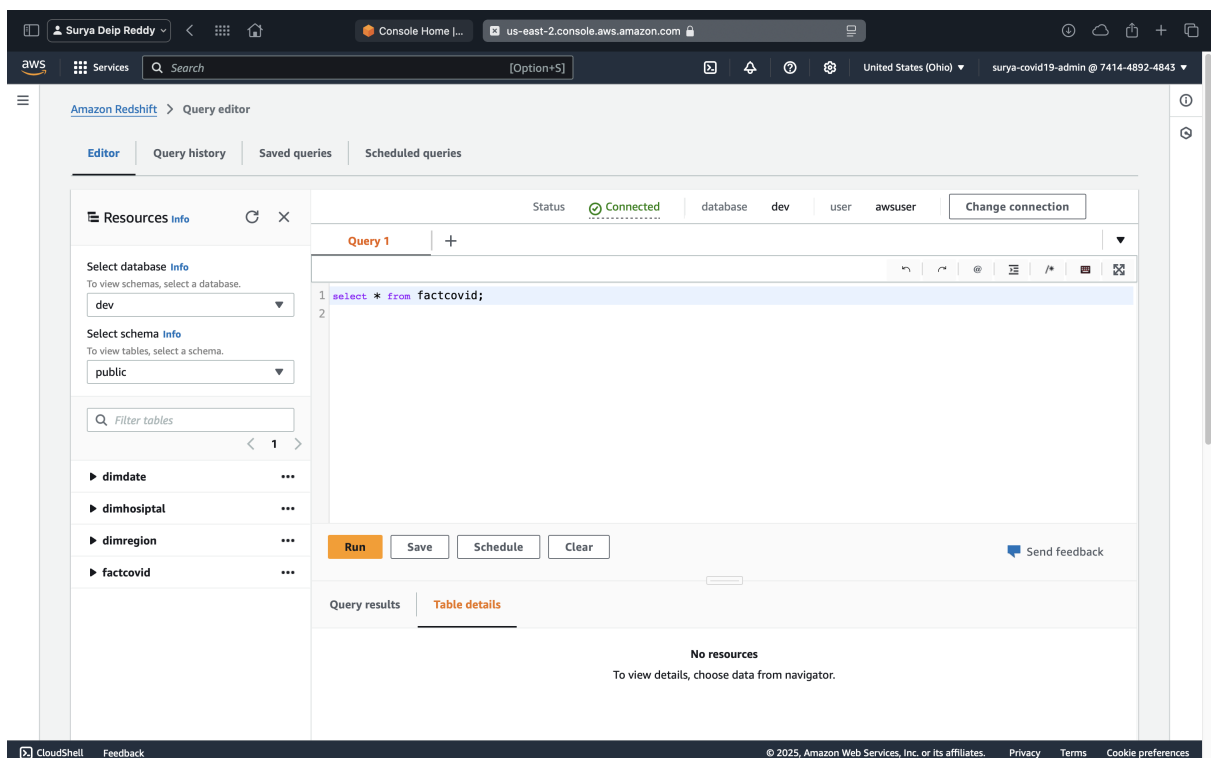
1. **factcovid Table:**
  - Stores COVID-19 related data such as `fips` , `province_state` , `confirmed` , `deaths` , etc.
2. **dimDate Table:**
  - Stores date-related information like `fips` , `date` , `year` , `month` , and `day_of_week` .

### 3. **dimRegion** Table:

- Contains regional information, including `fips`, `province_state`, `latitude`, and `county`.

### 4. **dimHospital** Table:

- Contains hospital-related data, such as `fips`, `hospital_name`, `latitude`, and `hospital_type`.



## 12. Loading Data into Redshift from S3

The `COPY` command is used to load data from **Amazon S3** into the corresponding tables in Redshift:

- factcovid** Table: Loads data from `factCovid.csv` in S3 into the `factcovid` table.
- dimDate** Table: Loads data from `dimDate.csv` in S3 into the `dimDate` table.
- dimRegion** Table: Loads data from `dimRegion.csv` in S3 into the `dimRegion` table.

4. **dimHospital Table:** Loads data from **dimHospital.csv** in S3 into the **dimHospital** table.

The screenshot displays the AWS Glue console interface. On the left, a navigation pane shows the hierarchy: **dimhospital** > **dimregion** > **factcovid**. The main panel shows the execution of a query (30304). The query is completed, started on January 25, 2025 at 21:15:17, with an elapsed time of 00 m 03 s. The results table shows 2180 rows returned. The table has columns: **index**, **fips**, **province\_state**, **country\_region**, **confirmed**, **deaths**, **recovered**, and **active**. The first 9 rows of data are visible.

| index | fips | province_state | country_region | confirmed | deaths | recovered | active |
|-------|------|----------------|----------------|-----------|--------|-----------|--------|
| 0     | 2    |                |                |           |        |           |        |
| 1     | 1    |                |                |           |        |           |        |
| 2     | 5    |                |                |           |        |           |        |
| 3     | 60   |                |                |           |        |           |        |
| 4     | 4    |                |                |           |        |           |        |
| 5     | 6    |                |                |           |        |           |        |
| 6     | 8    |                |                |           |        |           |        |
| 7     | 9    |                |                |           |        |           |        |
| 8     | 11   |                |                |           |        |           |        |