**Project Workflow: AWS-Based Data Pipeline for COVID-19 Dataset**

**Step 1: Log in with the IAM user you created**

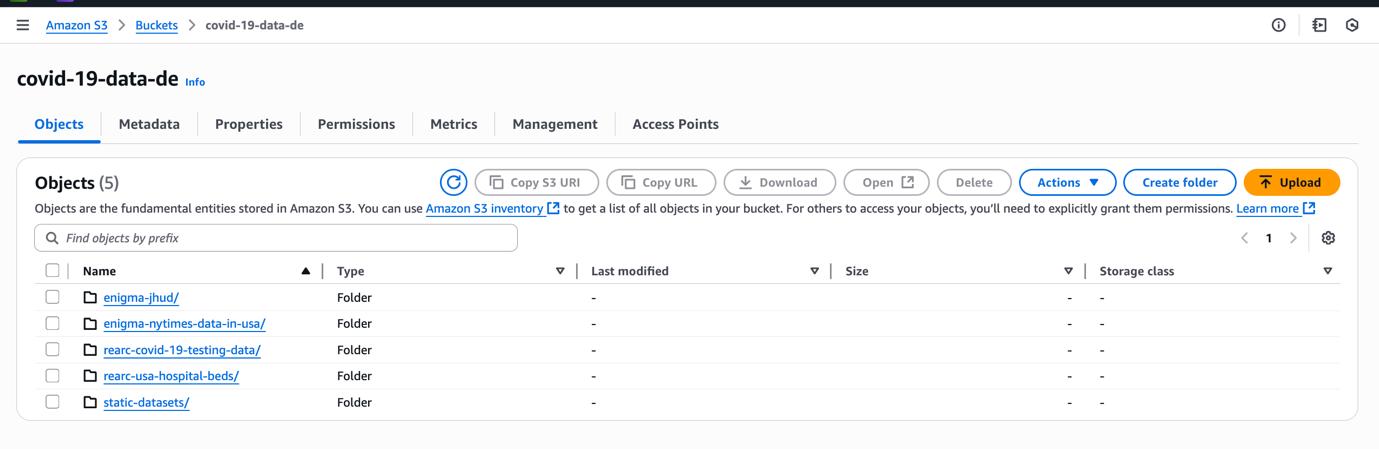
* **Goal**: Securely access AWS services using an IAM (Identity and Access Management) user with limited permissions.
* **What to Do**:
  + Go to [AWS Management Console](https://aws.amazon.com/console/).
  + Sign in using the **IAM user credentials** (not the root user).
  + Ensure the IAM user has permissions for:
    - Amazon S3
    - Amazon Athena
    - AWS Glue
    - Amazon Redshift
* **Why It Matters**:
  + IAM users ensure **principle of least privilege**, keeping AWS access secure and auditable. You should avoid using the **root user** for operational tasks.

**Step 2: Upload the dataset to your S3 bucket**

* **Goal**: Store the raw COVID-19 dataset in a centralized, durable, and scalable storage (S3) for processing.
* **What to Do**:
  + Go to **Amazon S3** service.
  + Navigate to the S3 bucket you created
  + Upload your dataset files manually or
  + Use the AWS Console **or** the AWS CLI:

aws s3 cp /path/to/local/file.csv s3://de-covid19-data-de/

* **Why It Matters**:
  + Amazon S3 is the **source layer** in your modern data architecture.
  + All downstream services (Glue crawler, Athena, Redshift) will **reference this data**.



### Step 3: Use AWS Glue Crawler to Analyze Data Structure

**Goal:**  
Automatically discover the schema and metadata of your data to build an accurate data model before processing.

**What to Do:**

**Create a Glue crawler for each dataset:**

* Name as table name for easy identification
* Specify the crawler source type:
  + Since you are starting fresh, choose **Data stores** as the data source.
  + Select **Amazon S3** as the data store type.
* Set the connection to **No connection** or keep default if not using a VPC connection.
* **Important:** Provide the path to the folder (S3 bucket + folder path), not individual files. The crawler will scan all files inside the folder automatically.
* Run the crawler to extract schema details such as number of columns, data types, and partitions.

**Create an IAM Role for Glue:**

* Go to the AWS IAM console and create a new role.
* Select **Glue** as the trusted service.
* Attach the following permissions policies:
  + AmazonS3FullAccess
  + AWSGlueServiceRole
  + AWSGlueConsoleFullAccess
* You can create this role during the crawler setup or beforehand. This role allows Glue to access your S3 data and manage resources.

**Create or Select Glue Database:**

* Create a new Glue database (covid\_19) to organize your crawled tables.

**Crawler Schedule:**

* You can configure the crawler to run on-demand or schedule it (hourly, daily, weekly). For initial runs, manual triggering is recommended.

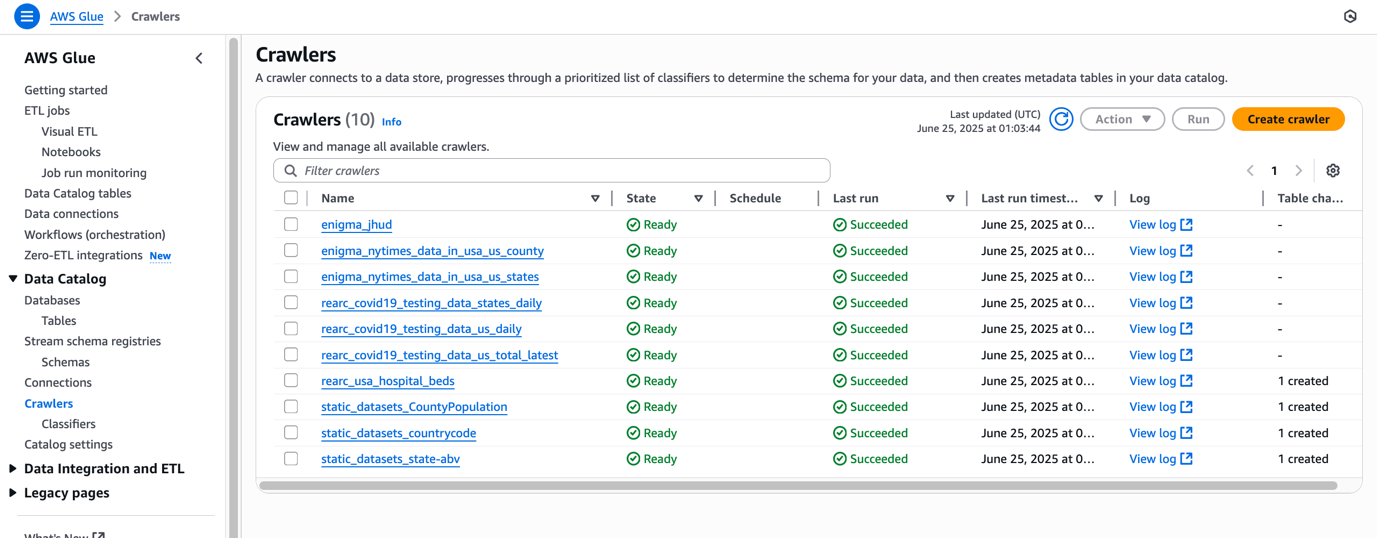
Run the crawler. It usually takes 1-2 minutes to catalog the data and create table metadata in the Glue Data Catalog. Repeat this process for each dataset you want to catalog.

**What Happens Internally:**

* When you create and run a crawler, it automatically creates a **table** in the Glue Data Catalog.
* This catalog stores all metadata related to your data files, such as column names, types, record counts, and file locations.
* The catalog acts as a centralized metadata repository, which Athena and other AWS services query to understand your data structure.
* Using Glue and Athena together means you don’t have to manually manage schemas; Glue handles it for you.

**Why It Matters:**

* The Glue crawler automatically catalogs your data, saving manual schema definition effort.
* It ensures your ETL jobs and queries work with up-to-date and accurate metadata.



### Step 4: Analyze Data with Amazon Athena and Build Data Model

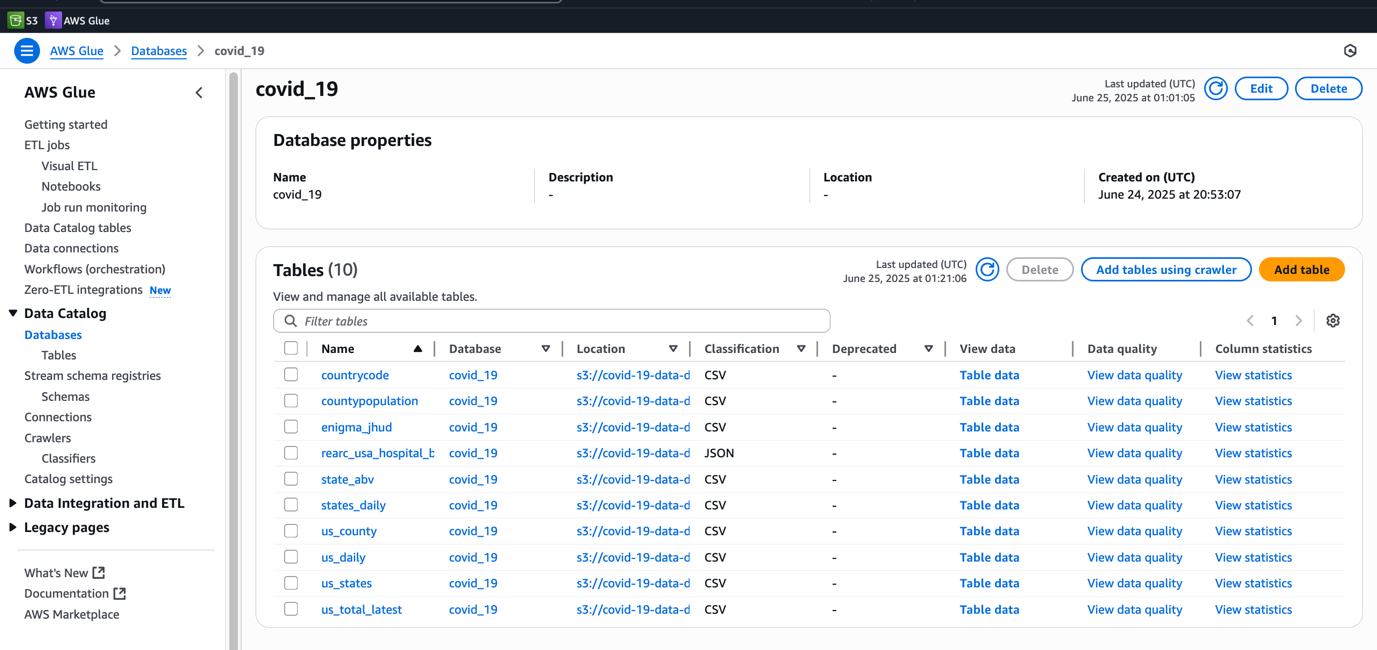
**Goal:**  
Use Amazon Athena to query and analyze your crawled data directly on S3 using standard SQL, enabling quick insights and data model building without loading data elsewhere.

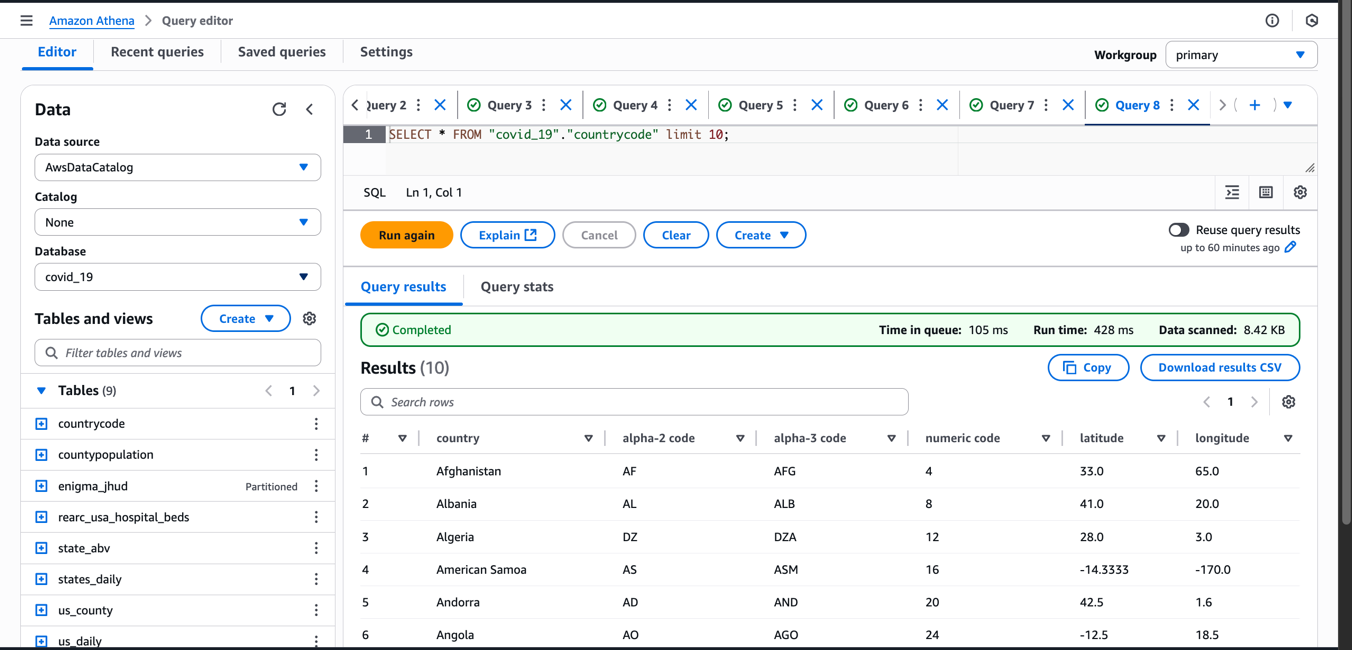
**What to Do:**

* Open Amazon Athena in the AWS Console.
* Set the **Data source** to AwsDataCatalog and select the Glue database you created (e.g., covid\_19).
* Before running queries, **configure a query result location** in S3:
  + Create a dedicated S3 bucket for Athena query outputs, e.g., de-covid19-athena-output.
  + Set this bucket as the output location in Athena settings.
* Athena automatically recognizes tables created by Glue Crawlers. For example, the table name matches the last folder name in S3 (e.g., enigma\_jhud).
* Use the Athena Query Editor to run SQL queries on your data.
* Preview tables to inspect columns and sample data (e.g., use SELECT \* FROM enigma\_jhud LIMIT 10).
* Athena supports **partitioning** based on S3 folder structure, which helps optimize queries when data is organized in subfolders like CSV1, CSV2, etc.
* While you can use Python (with Pandas or other libraries) to read S3 data, Athena is more efficient and scalable for large datasets because it queries directly on S3 without data movement.

**Why It Matters:**

* Athena enables **serverless, scalable SQL querying** on your data without ETL or data movement.
* Using Glue catalog tables with Athena provides a seamless way to query semi-structured data stored on S3.
* Partitioning improves query performance and reduces cost by limiting data scanned.
* Athena is ideal for large datasets where local tools like Python would be inefficient.



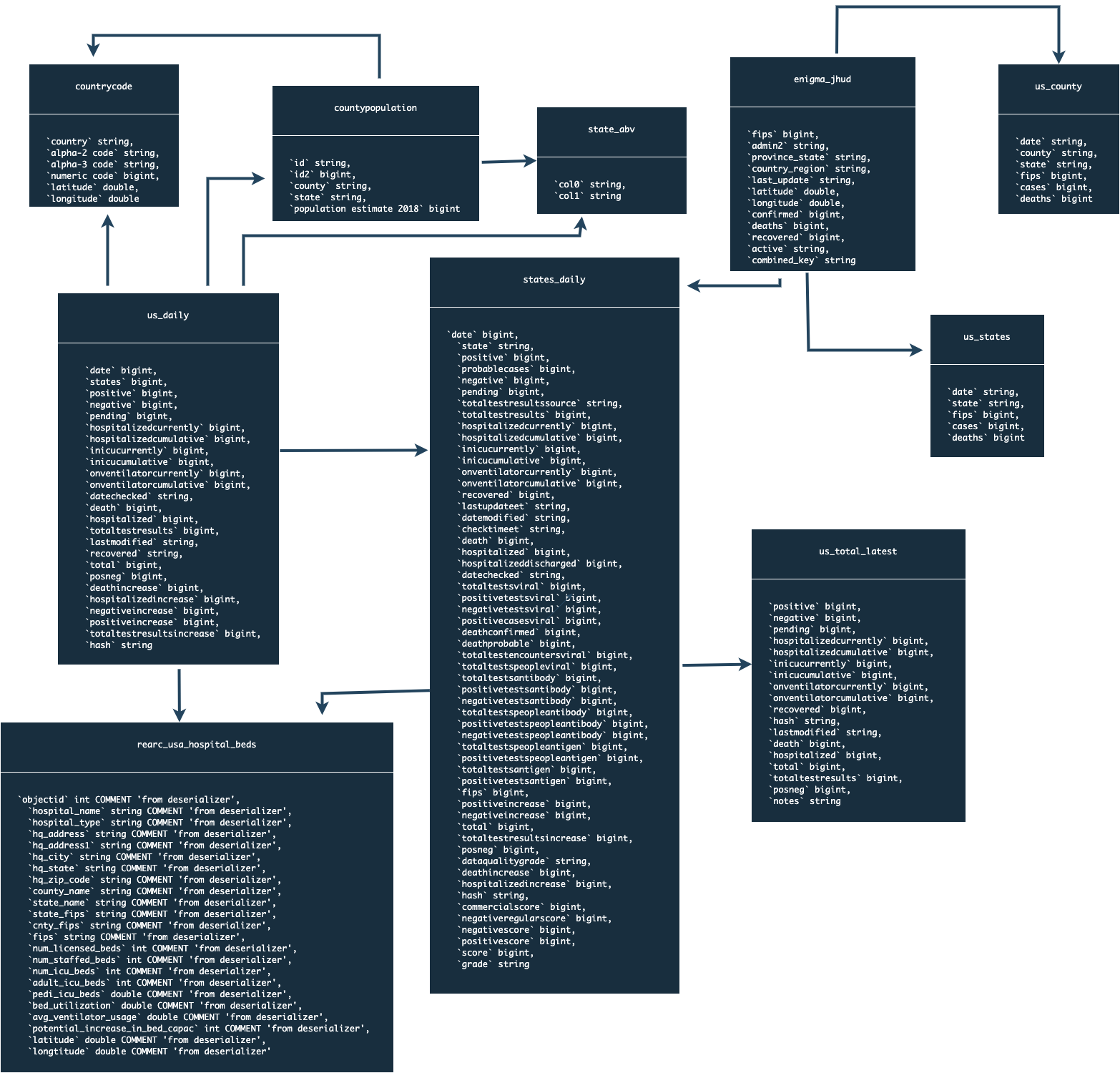


### Step 5: Build a Data Model Visually and Prepare for Data Warehouse

**Goal:**  
Create a clear visual representation of your data to understand relationships and structure, and prepare a dimension model for the data warehouse.

**What to Do:**

* Use a diagram tool like [draw.io](https://app.diagrams.net/) to build your data model visually.
* Export the table definitions (DDL) from Athena by clicking **Generate DDL**
* Design a **relational data model** showing how tables relate, focusing on key columns such as fips (a primary key in most tables), state, and county.
* Use the relationships (like fips key) to join tables logically
* Convert this relational model into a **dimension model** suitable for a data warehouse (star schema with fact and dimension tables).
* Plan your ETL and transformation scripts accordingly, which will create and populate these dimension tables in Amazon Redshift.
* Choose your ETL tool — for this project, Python is used (instead of Spark**Why It Matters:**
* Visual data models improve clarity on how your data is structured and related.
* Dimension modeling optimizes the schema for analytic queries in Redshift, improving performance and usability.
* Clear models guide your ETL and transformation logic to accurately populate your data warehouse.

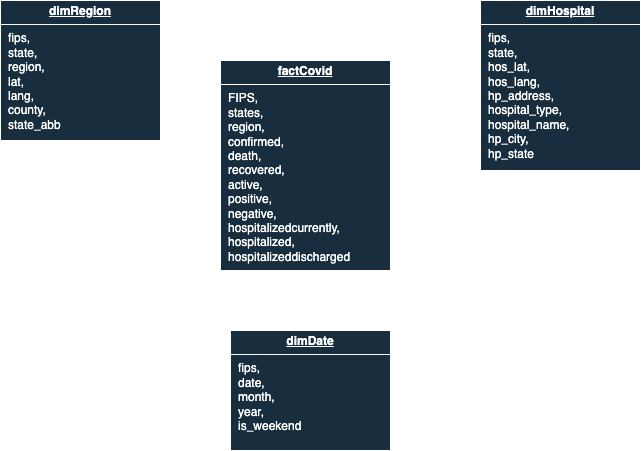


### Step 6: Convert the Relational Data Model into a Dimension Model (Star Schema) - Python

**Goal:**  
Transform the relational model into a star schema dimension model optimized for a data warehouse.

**What to Do:**

* Understand that the **star schema** consists of one central **fact table** surrounded by multiple **dimension tables**.
* The **fact table** (factCovid) stores the core measurable data — COVID-related facts like confirmed cases, deaths, recoveries, active cases, positives, negatives, hospitalized counts, and discharges. This table has a primary key to uniquely identify each record.
* Surrounding the fact table are **dimension tables** that provide descriptive context:
  + **Region dimension:** Contains geographic data such as state, province, latitude, longitude, county, and state abbreviation — useful for spatial analyses and mapping.
  + **Hospital dimension:** Includes hospital-specific details like hospital name, type, address, city, state, latitude, and longitude — enabling tracking of hospitals per region.
  + **Date dimension:** Stores date-related attributes such as the full date, year, month, day of week, and whether the day is a weekend — useful for time-based analysis and reporting.
* Assign primary keys in dimension tables for efficient joins to the fact table.
* Use this star schema as the blueprint for creating your Redshift tables and building ETL processes that load and transform data accordingly.



**Why It Matters:**

* The star schema design improves query performance and simplifies analytics in your data warehouse.
* Dimension tables enrich the fact data with detailed context, enabling powerful slicing and dicing of COVID data by time, location, and hospital.
* This structure is a common best practice in building scalable, maintainable data warehouses.

### Step 7: Write and Deploy the ETL Job in Python to Build the Data Warehouse

**Goal:**  
After creating the relational and dimension data models, the next step is to build the ETL pipeline that transforms and loads data to your data warehouse.

**What to Do:**

* **Connect to Athena and Query Data:**  
  Use Athena to query the raw data stored in S3 and perform necessary transformations. This leverages Athena’s serverless SQL querying on your S3 data.
* **Transform Data:**  
  Write Python scripts that run ETL transformations on the queried data, cleaning and shaping it to fit the dimension model (star schema).
* **Store Transformed Data in S3:**  
  After transformation, save the cleaned and structured data back to a designated S3 bucket.
* **Deploy Python ETL Script to AWS Glue:**  
  Upload and configure the Python ETL job as an AWS Glue job to run your transformations at scale and on schedule.

**Tools and Files:**

* The ETL Python script (covid19\_etl.py) is provided in the scripts/ folder of the project repository.
* Use Visual Studio (or your preferred IDE) to edit and manage your ETL scripts before deploying to Glue.

**Why This Matters:**

* This step automates the data transformation and loading process, making the pipeline scalable, repeatable, and maintainable.
* Using AWS Glue and Redshift together creates a powerful data warehousing solution optimized for analytics.

## **Step 8: Copy Data to Redshift Using AWS Glue Job**

### Goal:

This step covers loading the transformed data from S3 into Amazon Redshift by using an AWS Glue Python job. The goal is to build and populate Redshift tables based on the dimension model created earlier.

### Summary of the Process:

1. **Recap of Previous Steps:**
   * Built the relational data model and converted it to a dimension (star schema) model.
   * Queried and transformed data using Athena and Python scripts.
   * Stored transformed data back to S3.
2. **Create Redshift Cluster:**
   * Set up a Redshift cluster where the data warehouse tables will be created.
3. **Create Tables in Redshift:**
   * Use the dimension model schema to create tables such as DimDate, DimHospital, DimRegion, and FactCovid in Redshift.
4. **Write COPY Commands:**
   * Use Redshift’s COPY command to load data efficiently from S3 into these tables.
5. **Set Up AWS Glue Job:**
   * Create a Glue job (e.g., s3\_glue\_covid\_data) using **Python Shell** and Python 3.
   * Glue has limited support for external libraries, so you must:
     + Download the external redshift-connector library (e.g., version 2.0.889) from PyPI.
     + Upload this package to your S3 bucket (e.g., under covid-19-data-de/package/).
     + Reference this package in the Glue job’s Python library path.
6. **Glue Job Script:**
   * The job script (provided as aws\_glue\_job.py in the scripts/ folder) includes:
     + Connecting to Redshift using redshift\_connector.
     + Creating necessary tables in Redshift.
     + Executing COPY commands to load data from S3 to Redshift.
     + Using autocommit to ensure SQL statements are committed immediately.
7. **Run the Glue Job:**
   * Deploy and save the Glue job.
   * Run the job from the Glue console.
   * Monitor the execution logs for success or errors.
8. **Verify Data Load:**
   * Once the Glue job finishes, verify the tables and data in Redshift using SQL queries.
   * The data from transformed CSV files should now be loaded into Redshift, ready for further analysis or visualization.

### Additional Notes:

* You may need to drop existing tables in Redshift before running the Glue job again to avoid conflicts.
* The Glue Python Shell job does not support all Python libraries, so proper packaging and uploading of dependencies are crucial.
* This step completes the data pipeline from raw data in S3 to a quarriable data warehouse in Redshift.