A black screen with white text

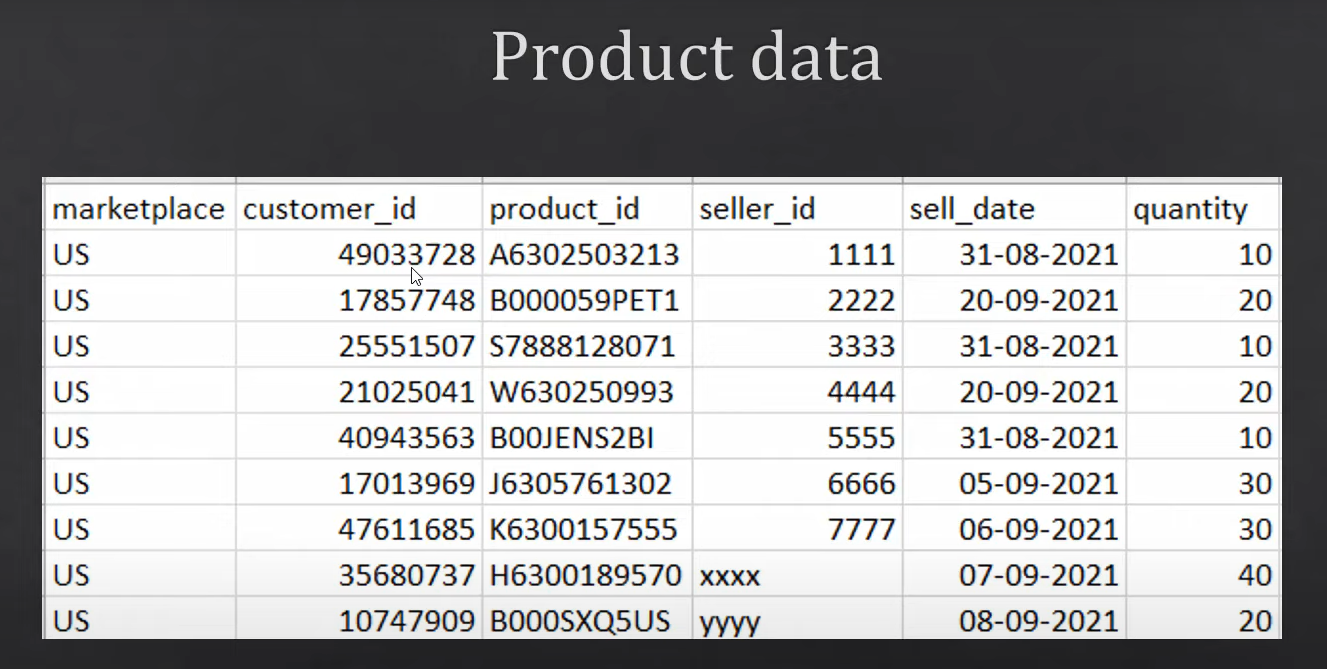
Description automatically generatedA black and white text on a black background

Description automatically generatedA black screen with white text

Description automatically generated

Project Structure:

1. Create s3 bucket named - myglue-etl-project-sit
2. Create 4 folder inside above bucket
   1. Input/product/year=2021
   2. Output
   3. Scripts
   4. Temp



1. Upload file- product\_data.csv file into year=2021 location
2. Create database named- mydatabase into glue
3. Once you create database into glue it will start reflecting into Athena
4. If we don’t create database, then it will create schema into default database
5. Create IAM role named- **myglue-crawler-role** (s3-full access, cloudwatch full access, awsglueservicerole )
6. Create first crawler named- **MyCrawlerFetchFromS3** (here we define s3 datasource, target datasource - mydabase and IAM role ).
7. When we run a crawler, it creates a schema in the Data Catalog. The Data Catalog only contains the schema, not the actual data. When we run a query in Athena, it reads the schema from the Data Catalog, finds the source location, and fetches the data from the source and displays the results in the table format defined by the schema.
8. If we upload another file named-> product\_data-2021.csv into year=2021 then it will start reflecting at Athena but if we create new folder named-> year=2022 and add new file named-> product\_data\_2022.csv into s3 then it will not reflect automatically, we need to run crawler again and when we run crawler again added the new index.

A black and white screen with white text

Description automatically generated

Create 2 crawler-

1. MyCrawlerFetchFromS3
2. MyCrawlerFetchFromRedshift

Create 2 ETL Job-

1. MyGlueJobReadFromS3
2. MyGlueInsertRedshift

**Note:**

AWS Lamba - can run max 15 minutes continuously

AWS Glue- can run 48 hours continuously

Both are serverless

Now we will create our first ETL script named-> MyGlueJobReadFromS3

Use the same IAM role -> **myglue-crawler-role**

that will fetch data from catalog and convert into parquete format and store into s3 bucket.

A screenshot of a computer

Description automatically generated

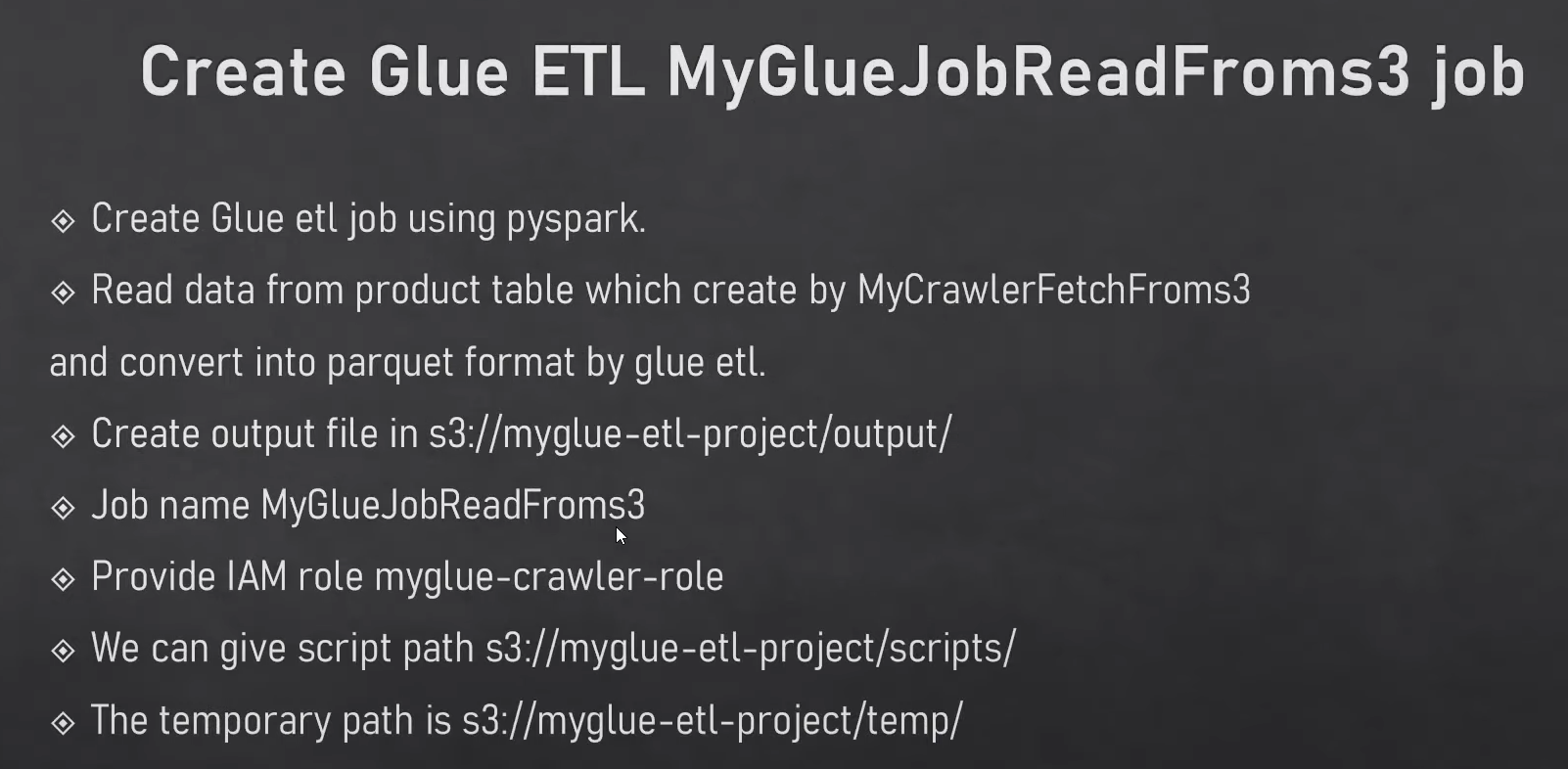
Post this we will update below job details section:-

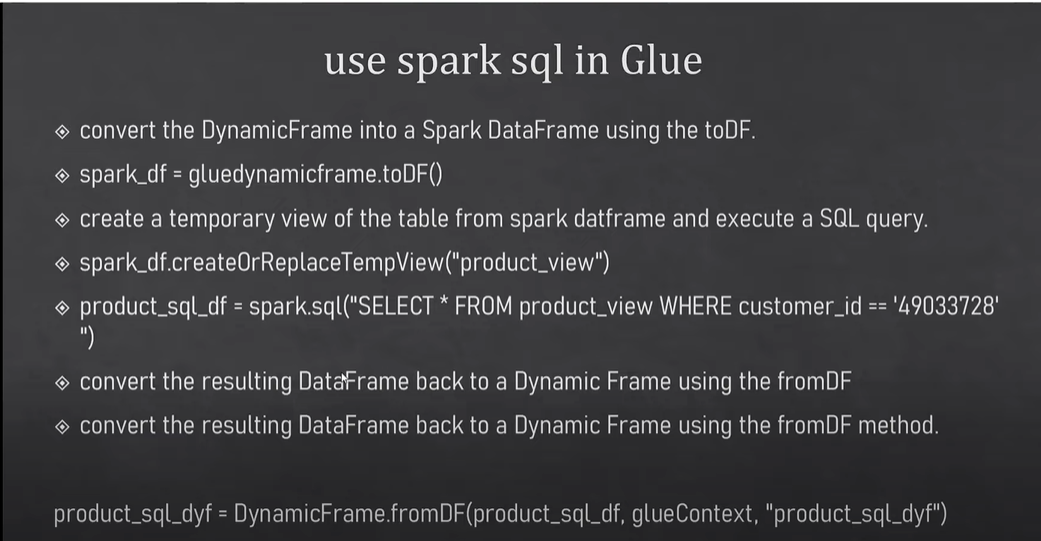
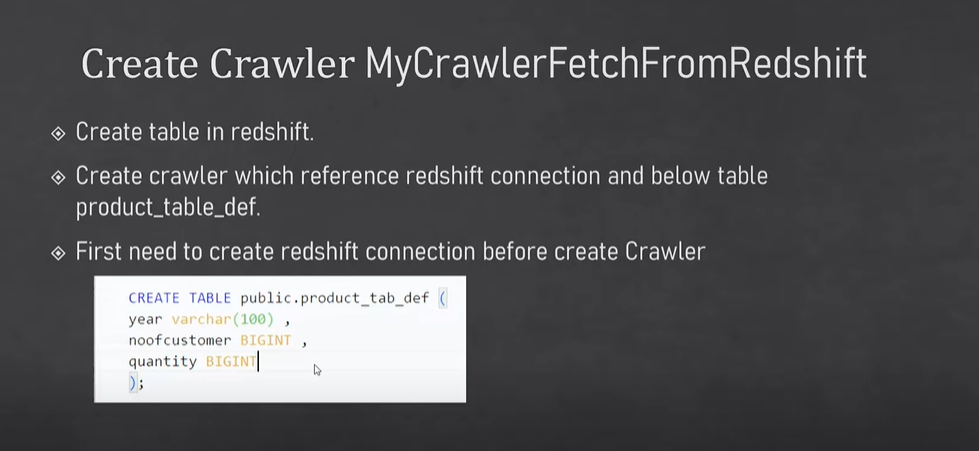
A screenshot of a computer

Description automatically generated

Now replace the script with -> MyGlueJobReadFromS3.py file

When we run this etl job it will format and store into s3 bucket

 A black sign with white text

Description automatically generated  

CREATE TABLE public.product\_tab\_def(

    year VARCHAR(100),

    noofcustomer BIGINT,

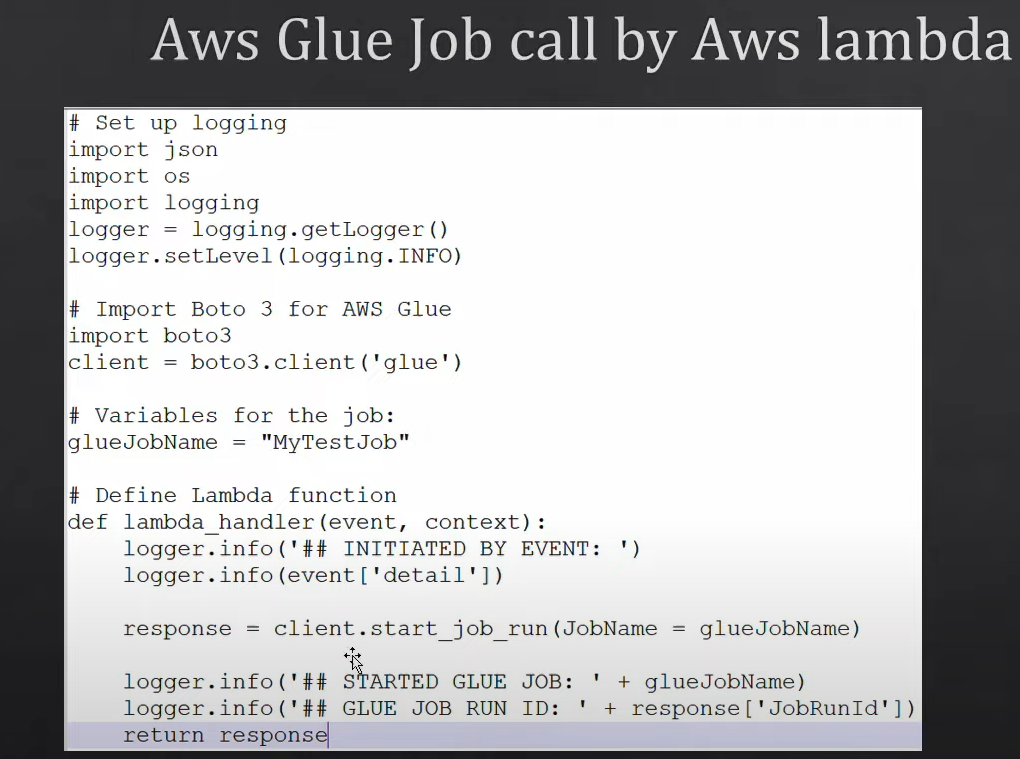
    quantity BIGINT

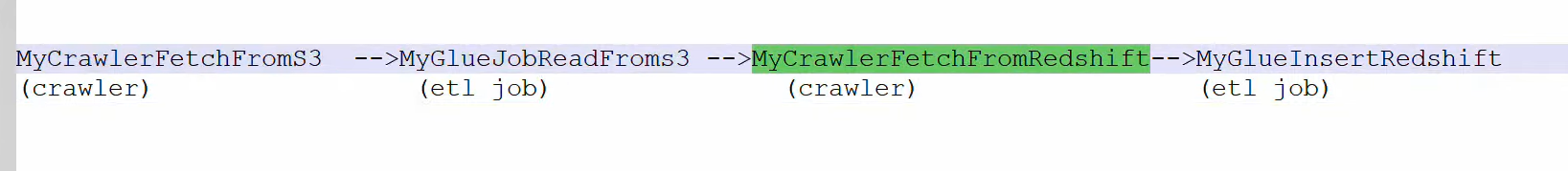
);

SELECT \* from dev.public.product\_tab\_def;

1. Create serverless redshift cluster named(namespace) – myredshiftcluster and create above table inside the redshift cluster.
2. Create username and password- awsuser /Admin123
3. Create connection in glue before creating crawler for redshift and test this connection
4. Redshift is a jdbc type of database- so while creating crawler we need to select jdbc type of data source.
5. Create another role named- **GlueRedshiftCrawlerRole** Which have s3,glue,redshift full access. It will attach with redshift.
6. Create crawler named – MyCrawlerFetchFromRedshift (source- redshift and ouput-mydatabase)
7. After running this crawler table is created in glue but not visible in athena
8. Create etl job named- MyGlueInsertRedshift (source s3 output location -> target redshift)
9. Post this run parquet file data inserted into redshift.

**We can run this etl job using aws lambda.**



* We can create [Workflows (orchestration)](https://ap-south-1.console.aws.amazon.com/glue/home?region=ap-south-1#/v2/etl-configuration/workflows) named-myworkflow in glue that will run etl job in sequence
* 
  1. First create new trigger named – Myeventtrigger that will run ondemand

Myeventtrigger-for-MyCrawlerFetchFromS3

Myeventtrigger-for-MyGlueJobReadFromS3

Myeventtrigger-for-MyCrawlerFetchFromRedshift

Myeventtrigger-for-MyGlueInsertRedshift

A screenshot of a computer screen

Description automatically generated

**Data Loading Workflow:**

1. **Data Storage**: The actual data is stored in sources such as Amazon S3, DynamoDB, RDS, or other databases.
2. **Crawler Operation**:
   * AWS Glue Crawlers scan these data sources, infer the schema, and create or update metadata tables in the Glue Data Catalog.
   * These metadata tables describe the structure and location of the data but do not contain the data itself.

**How ETL Fetches Data:**

1. **Metadata Retrieval**:
   * The Glue job uses the metadata in the Glue Data Catalog to understand the structure and location of the data it needs to process.
   * This is done using the create\_dynamic\_frame.from\_catalog method, which leverages the metadata to access the actual data stored in the data source.
2. **Loading Data into DynamicFrame**:
   * The create\_dynamic\_frame.from\_catalog method reads the metadata and loads the actual data from the data source into a DynamicFrame in the Glue job.
   * Here’s the relevant code snippet from your example:

python

Copy code

S3bucket\_node1 = glueContext.create\_dynamic\_frame.from\_catalog(

database="mydatabase",

table\_name="product",

transformation\_ctx="S3bucket\_node1"

)

* + This code fetches data from the product table in the mydatabase database as defined in the Glue Data Catalog and loads it into the S3bucket\_node1 DynamicFrame.

**Detailed Steps in ETL:**

1. **Define Data Source**: In the Glue Data Catalog, define the source location and schema (using a crawler).
2. **Fetch Metadata**: The ETL job uses this metadata to understand how to access and interpret the data.
3. **Load Data**: The ETL job reads the actual data from the defined source location (e.g., an S3 bucket) using the from\_catalog method.
4. **Transform Data**: Perform any required transformations using DynamicFrames or Spark DataFrames.
5. **Store Transformed Data**: Write the transformed data back to a data sink (e.g., another S3 bucket, database, etc.).