User Story;

Complete the below project using AWS S3, EMR(Step Job) & Redshift

Collect 2 different datasets of same schema.

Load Data into two different folder in S3.

Create Table as per data schema in Redshift.

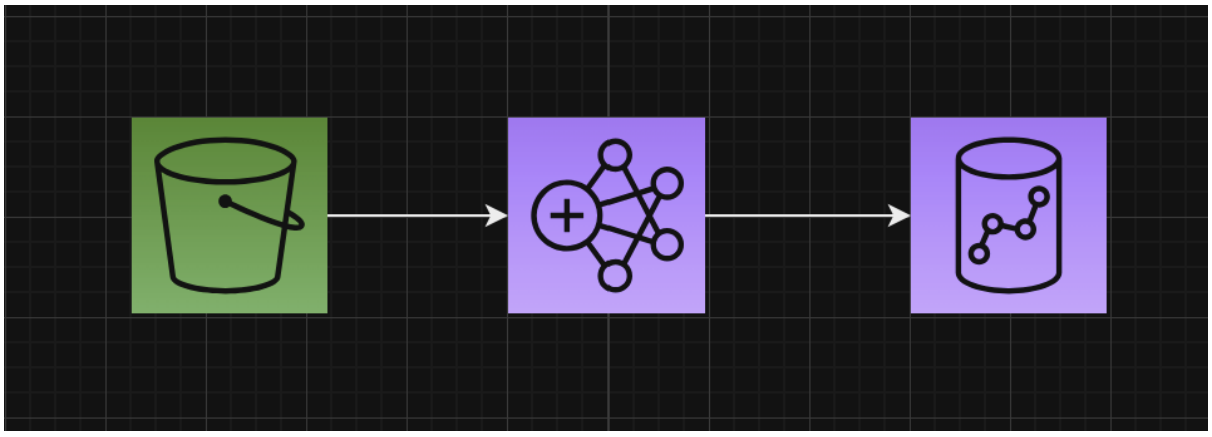
Writ a Spark Job to perform below operations:

> **Load first dataset into Redshift**
>
> **Collect 2nd Dataset and upsert (Update or Insert) into same table.**

Note: Use EMR cluster with Step job to execute your Spark script.

Submission: Submit spark script and screenshots of final table dataset. Both data is in parquet format.

Architecture:



Steps Reproduced to complete the task:

Step 1: Create a database with name was-project in redshift cluster.

Step 2: Create an EMR cluster and in next steps we can add the steps.

Step 3: Source data will present in S3 the initial data will always be there in

Location : s3://awsproject-vamsi-1/first_batch/

Every time the new data will come to the location specified.

Location: s3://awsproject-vamsi-1/next_batch/

Step 4 : Python Script.

Read the data from the parquet and make changes to the data accordingly and push the initial data to redshift using Spark JDBC.

The Code is as follows:

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job
from pyspark.sql.functions import col,lit,when
from pyspark.sql.window import Window
import pyspark.sql.functions as F
## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
df=spark.read.parquet("s3a://awsproject-vamsi-1/first_batch/")
new_flag_df = df.withColumn("flag", lit("active"))
new_df = new_flag_df[['parse_count','restaurant_name','rating_review','review_id','flag']]
new_df.write.parquet("s3a://awsproject-vamsi-1/refined_batch_first/")
refined_df = spark.read.parquet("s3a://awsproject-vamsi-1/refined_batch_first/")
refined_df.write.format("jdbc").option("url",
"jdbc:redshift://redshift-cluster-1.cphyywzefm4u.us-east-1.redshift.amazonaws.com:5439/dev").option("dbtable",
"aws_project.nyc_reviews").option("driver","com.amazon.redshift.jdbc42.Driver").option("user",
"admin").option("password", “Admin_1234”).mode("overwrite").save()
job.commit()
```

Step 5: The initial data is already from now that data will updated in next_batch. The task here is to check if we have same restaurant name with different review_id update the recent to active and make the previous reviews to inactive.

The code is as follows:

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

from pyspark.sql.functions import col,lit,when

from pyspark.sql.window import Window

import pyspark.sql.functions as F

## @params: [JOB_NAME]

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark_session

job = Job(glueContext)

job.init(args['JOB_NAME'], args)

rds_df = spark.read.format("jdbc").option("url",
"jdbc:redshift://redshift-cluster-1.cphyywzefm4u.us-east-1.redshift.amazonaws.com:5439/dev").option("dbtable",
"aws_project.nyc_reviews").option("driver","com.amazon.redshift.jdbc42.Driver").option("user",
"admin").option("password", "Admin_1234").load()

nxt_df=spark.read.parquet("s3://awsproject-vamsi-1/next_batch/")

nxt_flag_df = df.withColumn("flag", lit("active"))

next_df = nxt_flag_df[['parse_count','restaurant_name','rating_review','review_id','flag']]

next_df.write.parquet("s3a://awsproject-vamsi-1/refined_batch_next/")

refined_next_df = spark.read.parquet("s3a://awsproject-vamsi-1/refined_batch_next/")

combined_df = rds_df.unionAll(refined_next_df)

combined_df = combined_df.withColumn("review_id_numeric", F.regexp_extract("review_id", "\\d+", 0).cast("int"))

# Define a window specification to find the most recent review for each restaurant

windowSpec = Window.partitionBy("restaurant_name").orderBy(col("review_id_numeric").desc())

# Add a flag column to mark the most recent review for each restaurant as "active"

result_df = combined_df.withColumn(

    "flag",

    F.when(F.row_number().over(windowSpec) == 1, "active").otherwise("inactive")

).drop("source", "review_id_numeric")

# Show the final combined DataFrame

result_df.write.format("jdbc").option("url",
"jdbc:redshift://redshift-cluster-1.cphyywzefm4u.us-east-1.redshift.amazonaws.com:5439/dev").option("dbtable",
"aws_project.nyc_reviews").option("driver","com.amazon.redshift.jdbc42.Driver").option("user",
"admin").option("password", "Admin_1234").mode("overwrite").save()

job.commit()
```

Step 6: Create two step for the above step 4 and step 5 and the data will be uploaded into S3 whenever we run those steps.