|  |  |
| --- | --- |
| Company | pumpjackdataworks.com |
| Sender | Rais Mohammad <rais@ pumpjackdataworks.com> |
| Date, Time | 08/21/2020, 6:53 PM |
| Code Challenge | At ABC Company, each year every employee is given a percentage based increment to their salaries based on the departments they belong to. You have to write a PySpark script(s) to read from flat\_data.csv and store into employees and department tables in the schema below. Furthermore you need to read tables from the database, calculate the updated salaries and write them back.  The database contains the following schemas:   * **employee:** id :: UUID, first\_name::Text, last\_name::Text, salary::numeric, department\_id::UUID * **department:** id::UUID, name::Text, salary\_increment::numeric   The salary\_increment column in department contains the percentage value for calculating the salary. The output of the process should be the following table   * **updated\_salaries:** employee\_id, updated\_salary   All the tables must be created by script and have the necessary key relationships between them.  For doing this task you have two options.   * Use your own AWS account for the development. * Setup AWS Glue Env on your local machine <https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-etl-libraries.html> |
| Condition | Time to finish and is 24 Hours |
| Attachements |  |

**Solution and Steps**

Analyzing and discovering Data 🡪 small, demoralized dataset, containing data for 2 separated entities (Employee, Department) in CSV format

**Needed Logic to be implemented: -**

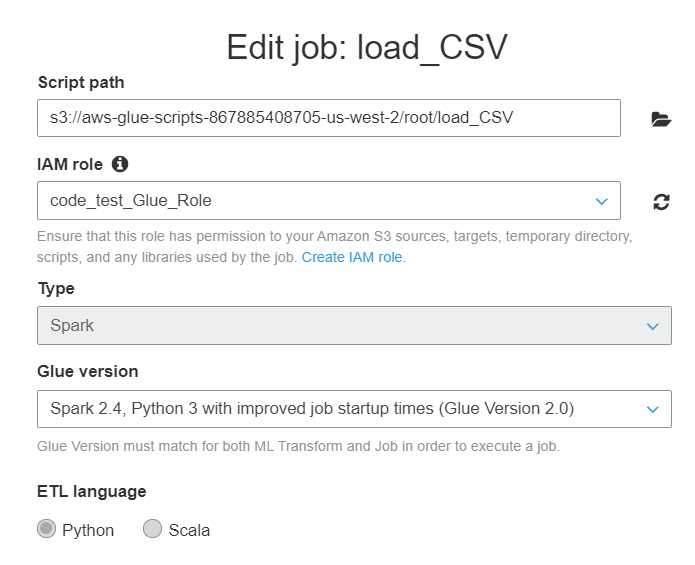
The Salary will be incremented by a percentage depending on employee’s Department and generate a result for all dataset records with same logic.

**Designing Solution brainstorm: -**

* 1. Load Raw Data to Temp Table
  2. Identify fields for each DB entities and define the integrity constrains which include primary and foreign keys for each Employee and Department entities.
  3. Define the datatypes for each field and start to build DDL to work on Normalized form tables.
  4. Load Distinct Department data into table
  5. Load Employee data with foreign keys for their departments.
  6. Join the two entities with key value (UUID of Department)
  7. Apply the logic formula ( updated\_salary = Salary \* (1+percentage/100)
  8. Write to target table.

**Implementation**

Start with AWS Glue (personal account) and define an ETL job named load\_CSV as



The code defines the details implementation

|  |
| --- |
| 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  import time  import pyspark  from pyspark.sql import SparkSession  from pyspark.sql.types import \*  from pyspark.sql.functions import col,array\_contains,row\_number  from pyspark.sql import functions as f  from uuid import uuid4  from pyspark.sql.window import Window  ## INITIALIZE Glue  glueContext = GlueContext(SparkContext.getOrCreate())  glueJob = Job(glueContext)  args = getResolvedOptions(sys.argv, ['JOB\_NAME'])  glueJob.init(args['JOB\_NAME'] , args )  #spark\_session = SparkSession.builder.appName('pumpjackdataworks.com').getOrCreate()  ## INITIALIZE Spark Session  sparkSession = glueContext.sparkSession  ##ETL read $RAW DATA from S3 file flat\_data.csv$  source\_schema = StructType() \  .add("first\_name",StringType(),True) \  .add("last\_name",StringType(),True) \  .add("salary",IntegerType(),True) \  .add("dept\_name",StringType(),True) \  .add("salary\_increment",IntegerType(),True)  df\_with\_source\_schema = sparkSession.read.format("csv") \  .option("header", True) \  .schema(source\_schema) \  .load("s3://pumpjackdataworks/flat\_data.csv")    #df\_with\_source\_schema.printSchema()  #df\_with\_source\_schema.show()  ##SELECT DEPARTMENT DISTINCAT and filling TABLE  dept\_schema = StructType() \  .add("UUID",IntegerType(),True) \  .add("dept\_name",StringType(),True) \  .add("salary\_increment",IntegerType(),True)  #sqlDF\_dept\_dist = sparkSession.createDataFrame(sparkSession.emptyRDD(), dept\_schema)  ## create department table  df\_with\_source\_schema.createOrReplaceTempView("department")  sqlDF\_dept\_dist = sparkSession.sql("select distinct dept\_name, salary\_increment from department")  ## Temp table to add UUID  sqlDF\_dept\_dist = sqlDF\_dept\_dist.alias("\_tmp")  sqlDF\_dept\_dist.registerTempTable("\_tmp")  df2 = sparkSession.sql("select uuid() as UUID , \* from \_tmp")  ## test results  #df2.show()  #sqlDF\_dept\_dist.show()  #sqlDF\_dept\_dist.write.option("header","true").csv("s3://pumpjackdataworks/departments.csv")  ## write a single file of $department$ into S3 departments.csv  df2.repartition(1).write.mode("overwrite").option("header", "true").csv("s3://pumpjackdataworks/departments")  ## EMPLOYEE DISTINCAT TABLE  source\_schema\_emp = StructType() \  .add("UUID",IntegerType(),True) \  .add("first\_name",StringType(),True) \  .add("last\_name",StringType(),True) \  .add("salary",IntegerType(),True) \  .add("dept\_name",StringType(),True) \  .add("salary\_increment",IntegerType(),True)  df\_with\_source\_schema.createOrReplaceTempView("employee")  sqlDF\_emp\_dist = sparkSession.sql("select distinct first\_name, last\_name, salary, dept\_name, salary\_increment from employee")  #sqlDF\_emp\_dist.show()  sqlDF\_emp\_dist = sqlDF\_emp\_dist.alias("\_tmp1")  sqlDF\_emp\_dist.registerTempTable("\_tmp1")  df3 = sparkSession.sql("select uuid() as UUID , \* from \_tmp1")  ## Test Employee Data after adding UUID  #df3.show()  ## write a single file of $department$ into S3 employee  df3.repartition(1).write.mode("overwrite").option("header", "true").csv("s3://pumpjackdataworks/employee")  #### Updated Salaries  df3 = df3.alias("updated\_salaries")  df3.registerTempTable("updated\_salaries")  #df\_with\_source\_schema.createOrReplaceTempView("updated\_salaries")  sqlDF\_US = sparkSession.sql("select distinct UUID as employee\_id, salary\*(1+salary\_increment/100) as updated\_salary from updated\_salaries")  ## Test Updated Salaries  #sqlDF\_US.show()  sqlDF\_US.repartition(1).write.mode("overwrite").option("header", "true").csv("s3://pumpjackdataworks/updatedsalaries")  #df\_with\_source\_schema.write.format('jdbc').options(  # url='jdbc:mysql://database-2.cakb1ijxchmw.us-west-2.rds.amazonaws.com:3306/test',  # driver='com.mysql.jdbc.Driver',  # dbtable='departments',  # user='admin',  # password='adminadmin').mode('append').save()  glueJob.commit() |

**Test Case**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Actual Data | |  |  |  |  |  | | --- | --- | --- | --- | --- | | first\_name | last\_name | salary | dept\_name | salary\_increment | | Porter | Cain | **9067** | Sales | 17 | |
| Expected Data | **Salary\*(1+DepartmentPercentageIncrease/100) =** 9067\*(1+17/100)  =  10608.39 |
| Test Case | **Passed** |
|  |  |

**Statistics on the run**

|  |  |
| --- | --- |
| **Avg Run Time** | 7.5 Sec using 10 workloads as maximum capacity for spark |
|  |  |

**Deliverables**: -

|  |  |
| --- | --- |
| ETL PySpark job | s3://aws-glue-scripts-867885408705-us-west-2/root/load\_CSV |
| Department Output | https://pumpjackdataworks.s3-us-west-2.amazonaws.com/departments/part-00000-7faf6ed7-4041-41c2-92d6-435aa9709b31-c000.csv |
| Employee Output | https://pumpjackdataworks.s3-us-west-2.amazonaws.com/employee/part-00000-fe1ffb9c-deef-4167-86e3-a3c4c0911b8b-c000.csv |
| UpdatedSalaries Output | https://pumpjackdataworks.s3-us-west-2.amazonaws.com/updatedsalaries/part-00000-43d5a565-eb24-413a-94e9-c9a34c2917e1-c000.csv |

**As attachments files: -**

|  |  |
| --- | --- |
| ETL PySpark job |  |
| Department Output |  |
| Employee Output |  |
| UpdatedSalaries Output |  |

**GitHub URL**

**Other Trials**:

I tried Arura MySQL, RDS services and it had issues to be seen inside AWS Glue

I tried IAM, VPC and subnet mask of MySql Instance to be in the same discoverable VPC area of AWS Glue to be able to define connection to MySQL, but it give error which currently I am googleing a solution for it.

Defined roles and policies to execute AWS services in two different VPC’s and grant proper permission to S3 and RDS-Instance-MySQL and AWS Glue.

**Known issues**: -

Due to issue of no RDB access from AWS Glue, so I decided to go with DataFrame entities instead of tables as a workaround

Not linking tables using UUID, instead I applied the logic into raw data itself, and organize the output to match requirements format.