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 awsgluedq.transforms import EvaluateDataQuality

from awsglue.dynamicframe import DynamicFrame

from pyspark.sql import functions as F

def sparkSqlQuery(glueContext, query, mapping, transformation\_ctx) -> DynamicFrame:

for alias, frame in mapping.items():

frame.toDF().createOrReplaceTempView(alias)

result = spark.sql(query)

return DynamicFrame.fromDF(result, glueContext, transformation\_ctx)

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

# Data quality rules

DEFAULT\_DATA\_QUALITY\_RULESET = """

Rules = [

ColumnCount > 0

]

"""

# Read raw data

inputS3\_node = glueContext.create\_dynamic\_frame.from\_catalog(

database="customer\_db",

table\_name="customer\_data\_input\_raw",

transformation\_ctx="inputS3\_node"

)

# Convert to DataFrame for advanced transformations

df = inputS3\_node.toDF()

# Standardize and clean data

cleaned\_df = df.select(

F.col("customerid"),

F.when(F.col("name").isNull() | (F.trim(F.col("name")) == ""), "Unknown")

.otherwise(F.col("name")).alias("name"),

F.when(F.col("email").isNull() | (F.trim(F.col("email")) == "") | ~F.col("email").contains("@"),

"noemail@example.com")

.otherwise(F.lower(F.trim(F.col("email")))).alias("email"),

F.when(F.col("phone").isNull() | (F.trim(F.col("phone")) == ""), "0000000000")

.otherwise(

F.lpad(

F.expr("right(regexp\_replace(phone, '[^0-9]', ''), 10)"),

10,

"0"

)

).alias("phone"),

F.when(F.col("address").isNull() | (F.trim(F.col("address")) == ""), "Not Provided")

.otherwise(F.col("address")).alias("address")

)

# Remove duplicates

deduped\_df = cleaned\_df.dropDuplicates()

# Optional: Write as single file

coalesced\_df = deduped\_df.coalesce(1)

# Convert back to DynamicFrame

final\_output\_dynamicframe = DynamicFrame.fromDF(coalesced\_df, glueContext, "final\_output\_dynamicframe")

# Data quality evaluation

EvaluateDataQuality().process\_rows(

frame=final\_output\_dynamicframe,

ruleset=DEFAULT\_DATA\_QUALITY\_RULESET,

publishing\_options={

"dataQualityEvaluationContext": "dq\_node",

"enableDataQualityResultsPublishing": True

},

additional\_options={

"dataQualityResultsPublishing.strategy": "BEST\_EFFORT",

"observations.scope": "ALL"

}

)

# Write to S3

glueContext.write\_dynamic\_frame.from\_options(

frame=final\_output\_dynamicframe,

connection\_type="s3",

format="csv",

connection\_options={

"path": "s3://customer-data-output-processed/single\_output/",

"partitionKeys": []

},

transformation\_ctx="outputS3\_node"

)

job.commit()