Manual de Pyspark

O Pyspark é uma API do Python para processamento em big data, e todas as funções que você encontra no SQL estão disponíveis no Pyspark, para que você execute operações complexas de manipulação de dados.

Lembrando que também é possível usar apenas o SQL dentro do Pyspark, mas é altamente recomendado usar apenas a sintaxe do Pyspark, dado que esse tem melhor performance.

Leitura de dados

Para ler a tabela no Pyspark é necessário executar os comandos abaixo, sempre passando como parâmetro o ***nome do banco de dados, nome da tabela, e a partição selecionada.***

```
nome_database = 'nome_suatabela'
nome_base = 'nome_sua_base'
particao = 'sua partição' # exemplo: Anomesdia = 20241022

anomesdia2 = conn.get_last_partition(
    database=nome_database
    table=nome_base
)

df = glueContext.create_data_frame.from_catalog(
    database=nome_database,
    table=nome_base,
    push_down_predicate=f'{anomesdia2}', # Partição que será selecionada
  additional_options={'useCatalogSchema':   True,   'useSparkDataSource':
True}
)
```

Leitura de dados puxando a última partição sem biblioteca DataUtils

Código para puxar última partição da tabela sem o DataUtils

```
import boto3

# Configurar o cliente do Glue
glue = boto3.client('glue', region_name='us-east-1')  # Altere para a sua
região

def obter_ultima_particao(database, table):
    # Listar partições da tabela no catálogo Glue
    response = glue.get_partitions(
```

```
        DatabaseName=database,
        TableName=table,
        MaxResults=1000  # Controla o número máximo de partições retornadas
por vez
    )
    # Extrair as partições
    particoes = response['Partitions']

    # Acessar a última partição com base em ordenação lexicográfica
(alfabética)
    ultima_particao = max(particoes, key=lambda x: x['Values'])

    return ultima_particao['Values']
```

Transformando dados

Para transformar os dados usando Pyspark, é necessário usar a estrutura conforme abaixo. Nesse código todos os elementos são contemplados: ***Case when, Cast, DateDiff, Date_format, Translate, Truncate, Trim, FillNa, DropNa, orderBy, DropDuplicates entre outros***:

```
From pyspark.sql.functions import trim, col, when, translate, date_format,
date_diff, current_date

# Preenche valores nulos. Exemplo: valor se missing receberá 0
df = df.fillna(
{
 'valor': '0'
 ,'data': '2023-10-01'
}).dropna()

# Realiza o SELECT com as transformações Mais Comuns do SQL
df_select = df.select(
    col("id").cast("bigint").alias("id_bigint")
    ,trim(col("valor")).cast("decimal(10,2)").alias("valor_decimal")
    ,translate(col("codigo"), '+', '').alias("codigo_sem_mais")
    ,date_format(col("data"), 'yyyy-MM-dd').alias("data_formatada")
    , datediff(current_date(), col("data")).alias("dias_desde_data")

    ,when(col("descricao") == 'valor1', 'Descrição 1')
        .when(col("descricao") == 'valor2', 'Descrição 2')
        .otherwise('Outro').alias("descricao_transformada")
)

# Filtra o dataframe onde os valores sejam maiores que 20, e ordena de
forma ascendente pela coluna valor
df_select = df_select.filter(col("valor") > 20).orderBy(col("valor"))
```

Apagando Tabela e Pastinha do S3 pelo Glue

Para apagar a tabela do catálogo de dados é necessário usar o comando abaixo, no entanto, importante lembrar que caso reutilize o espaço do S3 é necessário apagar o caminho também

Apagando o Caminho do S3

```
import boto3

s3 = boto3.client('s3')

bucket_name = 'seu-bucket'
folder_name = 'sua-pasta/'  # Inclua a barra no final se for uma pasta

objects = s3.list_objects_v2(Bucket=bucket_name, Prefix=folder_name)['Contents']
keys = [{'Key': obj['Key']} for obj in objects]

s3.delete_objects(Bucket=bucket_name, Delete={'Objects': keys})
print("Objetos apagados com sucesso.")
```

Apagando a Tabela do Catálogo de Dados

```
import boto3

glue_client = boto3.client('glue')

try:
    # Nome do banco de dados e tabela
    database_name = 'seu-banco'
    table_name = 'sua-tabela'

    # Apaga a tabela
    glue_client.delete_table(DatabaseName=database_name, Name=table_name)
    print(f'Tabela {table_name} apagada com sucesso do banco de dados {database_name}.')

except Exception as e:
    print(f'Erro ao apagar a tabela: {e}')
```

Salvando Tabela

```
# Imports necessários
```

```
from awsglue.context import GlueContext
from awsglue.dynamicframe import DynamicFrame
from awsglue.transforms import *
from pyspark.context import SparkContext

caminho_s3 = 'Seu caminho do s3'
nome_base = 'Nome sua base'
nome_database = 'Nome do database'
df = nome_seu_dataframe

dynamic_frame = DynamicFrame.fromDF(df, GlueContext, "dynamic_frame")
sink = GlueContext.getsink(
    path=caminho_s3
    ,connection_type='s3'
    ,updateBehavior='UPDATE_IN_DATABASE'
    ,partitionKeys=[] # Caso tenha partição preencha dentro do []. Exemplo
'anomesdia'
    ,enableUpdateCatalog=True
    ,transformation_ctx=''
)
sink.setCatalogInfo(catalogDatabase=nome_database,
catalogTableName=nome_base)
sink.setFormat("glueparquet", compression='snappy')
sink.writeFrame(dynamic_frame)
```