import os

from datetime import datetime

from airflow.decorators import dag, task

from airflow.operators.python import ShortCircuitOperator

from airflow.providers.postgres.operators.postgres import PostgresOperator

from airflow.utils.state import State

default\_args = {

  'owner': 'DWH',

  'depends\_on\_past': False,

  'retries': 0,

  'pg\_schema': 'project',

  'pg\_connection\_id': 'dwh\_pg'

}

INPUT\_FOLDER = os.path.join(os.getenv('DATA\_FOLDER'), 'input')

SQL\_FOLDER = os.path.join(os.getenv('DATA\_FOLDER'), 'sql')

NEO4J\_FOLDER = os.path.join(os.getenv('DATA\_FOLDER'), 'neo4j')

@task()

def generate\_csv\_files():

    # generate CSV files from the data warehouse

    #print("CSV files generated")

@task()

def load\_into\_neo4j\_task():

    uri = os.getenv('NEO4J\_URI', 'bolt://neo4j:7687')

    user = os.getenv('NEO4J\_USER', 'neo4j')

    password = os.getenv('NEO4J\_PASSWORD', 'test')

    # the logic to load CSV files into Neo4j: 'LOAD CSV' Cypher commands for incremental loads.

    #print("Data loaded into Neo4j")

@task()

def run\_cypher\_queries():

    # Cypher queries against the Neo4j database

    #print("Cypher queries executed")

@dag(

  default\_args=default\_args,

  description='ETL pipeline for Kaggle data',

  schedule\_interval='\* \* \* \* \*',  # execute every minute

  start\_date=datetime(2023, 10, 1, 0, 0, 0),

  catchup=False,

  concurrency=1,

  max\_active\_runs=1,

  template\_searchpath=[os.getenv('DATA\_FOLDER'), SQL\_FOLDER],

)

def etl\_submissions():

  @task(

    trigger\_rule='all\_done'

  )

  def complete\_chunk(\*\*kwargs):

    tasks = kwargs["ti"].get\_dagrun().get\_task\_instances()

    failed\_tasks = [ti for ti in tasks if ti.state == State.FAILED]

    chunk\_id = kwargs["ti"].xcom\_pull(task\_ids='extract\_chunk', key='return\_value')[0][0]

    chunk\_status = 'failed' if len(failed\_tasks) > 0 else 'processed'

    print(f"Marking chunk {chunk\_id} as {chunk\_status}")

    pg\_operator = PostgresOperator(

      task\_id='complete\_chunk',

      postgres\_conn\_id=default\_args["pg\_connection\_id"],

      sql="update project.chunk\_queue set status = %(chunk\_status)s where id = %(chunk\_id)s",

      parameters={

        'chunk\_id': chunk\_id,

        'chunk\_status': chunk\_status

      }

    )

    return pg\_operator.execute({})

  def chunk\_in\_xcom(\*\*kwargs):

    chunk = kwargs["ti"].xcom\_pull(task\_ids='extract\_chunk', key='return\_value')

    return len(chunk) > 0

  extract\_chunk\_task = PostgresOperator(

    task\_id='extract\_chunk',

    postgres\_conn\_id=default\_args["pg\_connection\_id"],

    sql="select \* from project.chunk\_queue where status = 'pending' or status = 'failed' order by id limit 1"

  )

  check\_for\_chunks\_to\_process = ShortCircuitOperator(

    task\_id="check\_for\_chunks\_to\_process",

    python\_callable=chunk\_in\_xcom,

  )

  # Task for generating CSV files

  generate\_csv\_task = generate\_csv\_files()

  # Task for loading CSV data into Neo4j

  load\_neo4j\_task = load\_into\_neo4j\_task()

  # Task for running Cypher queries

  run\_cypher\_queries\_task = run\_cypher\_queries()

  # Final task to mark the chunk as complete

  complete\_chunk\_task = complete\_chunk()

  # Setting up the task dependencies

  extract\_chunk\_task >> check\_for\_chunks\_to\_process >> generate\_csv\_task >> load\_neo4j\_task >> run\_cypher\_queries\_task >> complete\_chunk\_task

etl\_submissions\_dag = etl\_submissions()