Neo4j Handler Class

```python
#Author : Vithiya Saraswathi a/p Sockalingam

from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType
from neo4j import GraphDatabase
from neo4j.exceptions import ServiceUnavailable, AuthError

class Neo4jHandler:
    def __init__(self, uri, user, password):
        self.uri = uri
        self.user = user
        self.password = password
        self.driver = None
        self.connect()

    def connect(self):
        try:
            self.driver = GraphDatabase.driver(self.uri, auth=(self.user, self.password))
            self.driver.verify_connectivity()
            print("Successfully connected to Neo4j!")
        except ServiceUnavailable:
            print("Failed to connect to Neo4j. Service is unavailable.")
        except AuthError:
            print("Failed to connect to Neo4j. Authentication error. Please check your credentials.")
        except Exception as e:
            print(f"Failed to connect to Neo4j. Error: {str(e)}")
            self.driver = None
    def close(self):
        if self.driver:
            self.driver.close()
            print("Connection to Neo4j closed.")

    def clear_database(self):
        def clear(tx):
            tx.run("MATCH (n) DETACH DELETE n")

        try:
            with self.driver.session() as session:
                session.execute_write(clear)
                print("Database cleared successfully.")
        except ServiceUnavailable:
            print("Failed to connect to Neo4j. Service is unavailable.")
        except AuthError:
            print("Failed to connect to Neo4j. Authentication error. Please check your credentials.")
        except Exception as e:
            print(f"An error occurred: {e}")
    def create_product_nodes_and_relationships(self, data, batch_size=1000):
        cypher_query = """
        UNWIND $data as row
        MERGE (p:Product {skuInfo: row.SkuInfo})
        FOREACH(ignoreMe IN CASE WHEN row.Review IS NOT NULL THEN [1] ELSE [] END |
            CREATE (r:Review {
            name: row.Name,
            review: row.Review,
            starcount: row.StarCount,
            date: row.Date
            })
            MERGE (r)-[:REVIEWS]->(p)
        )
        """
        count_queries = {
            "Product": "MATCH (p:Product) RETURN COUNT(p) AS count",
            "Review": "MATCH (r:Review) RETURN COUNT(r) AS count"
        }
        try:
            with self.driver.session() as session:
                # Process data in batches
                for i in range(0, len(data), batch_size):
                    batch = data[i:i + batch_size]
                    session.run(cypher_query, data=batch)
                    print(f"Batch {i//batch_size + 1} processed successfully.")

                # Count and print the nodes
                for label, query in count_queries.items():
                    result = session.run(query)
                    count = result.single()["count"]
                    print(f"Total {label} nodes: {count}")
        except Exception as e:
            print(f"An error occurred: {e}")
    def load_reviews_to_dataframe(self, spark):
        query = """
        MATCH (r:Review)-[:REVIEWS]->(p:Product)
        RETURN r.name AS Name,
               r.review AS Review,
               r.starcount AS StarCount,
               r.date AS Date,
               p.skuInfo AS SkuInfo
        """

        def execute_query(driver, query):
            with driver.session() as session:
                result = session.run(query)
                records = [record.data() for record in result]
            return records

        def neo4j_date_to_str(neo4j_date):
            return str(neo4j_date) if neo4j_date else None

        try:
            records = execute_query(self.driver, query)
            records = [{'Name': r['Name'],
                        'Review': r['Review'],
                        'StarCount': r['StarCount'],
                        'Date': neo4j_date_to_str(r['Date']),
                        'SkuInfo': r['SkuInfo']} for r in records]

            schema = StructType([
                StructField("Name", StringType(), True),
                StructField("Review", StringType(), True),
                StructField("SkuInfo", StringType(), True),
                StructField("Date", StringType(), True),
                StructField("StarCount", IntegerType(), True)
            ])

            df_spark = spark.createDataFrame(records, schema=schema)
            return df_spark
        except Exception as e:
            print(f"An error occurred: {e}")
            return spark.createDataFrame([], schema=schema)
```