DED Assignment

Description

Your task is to create a Data Engineering pipeline using the following specifications:

1. **Database Setup**:
    - Create three tables with the following schemas in a RDBMS database of your choice (MySQL, PostgreSQL, ...):
        - `Orders`

          ```
          id: INTEGER
          total_price: FLOAT
          created_at: TIMESTAMP
          ```

        - `OrderProducts`

          ```
          id: INTEGER
          order_id: INTEGER
          product_id: INTEGER
          quantity: INTEGER
          created_at: TIMESTAMP
          ```

        - `Products`

          ```
          id: INTEGER
          title: STRING
          created_at: TIMESTAMP
          ```

2. **Data Generation**:
    - Implement a **data generator** script using **vanilla (pure) Python** to populate these tables with realistic data. Ensure:
        - The data resembles real-world scenarios.
        - Relationships between tables (`order_id`, `product_id`) are consistent and meaningful.
3. **Data Processing with PySpark**:
    - Develop a **PySpark application** that:
        - Joins the three tables into a single **DataFrame** (Please keep in mind that we are dealing with big data, so you need to read the three tables **separately** from the RDBMS database, based on traversing their **`created_at`** columns in **batches every hour**).
        - Adds a new column `products` to the `Orders` table, which contains a nested array of fields from `Products` and `OrderProducts`, in the following structure:

          ```
          products: ARRAY<STRUCT<
              title: STRING,
              quantity: INTEGER,
              product_id: INTEGER
          >>
          ```

        - Example final DataFrame schema:

          ```
          root
           |-- id: INTEGER
           |-- total_price: FLOAT
           |-- created_at: TIMESTAMP
           |-- products: ARRAY<STRUCT<
               title: STRING,
               quantity: INTEGER,
               product_id: INTEGER>>
          ```

4. **Containerization**:
    - Package the entire solution into **Docker Compose** with three services:
        - **Database Service**: Host the database.
        - **Data Generator Service**: Populate the database tables with generated data.
        - **Spark Application Service**: Process the data and display the resulting DataFrame using `df.show()`.

Deliverables

- **Python Scripts**:
    - Data generator script.
    - PySpark application script.
- **Docker Compose Configuration**:
    - Define services for the database, data generator, and PySpark app.
- **Documentation**:
    - Include clear instructions in a `README.md` on how to:
        - Build and run the containers.

- View the final DataFrame output.

Schemas

1. **Database Table Schemas**:

   - **Orders**:

     ```
     root
      |-- id: INTEGER
      |-- total_price: FLOAT
      |-- created_at: TIMESTAMP
     ```

   - **OrderProducts**:

     ```
     root
      |-- id: INTEGER
      |-- order_id: INTEGER
      |-- product_id: INTEGER
      |-- quantity: INTEGER
      |-- created_at: TIMESTAMP
     ```

   - **Products**:

     ```
     root
      |-- id: INTEGER
      |-- title: STRING
      |-- created_at: TIMESTAMP
     ```

2. **Final DataFrame Schema**:

   ```
   root
    |-- id: INTEGER
    |-- total_price: FLOAT
    |-- created_at: TIMESTAMP
    |-- products: ARRAY<STRUCT<
        title: STRING,
        quantity: INTEGER,
        product_id: INTEGER>>
   ```

Example Final DataFrame Output

Sample output of the PySpark application (`df.show(truncate=False)`):

```
+---+-----------+-------------------+-------------------------------------------------------------------------------+
|id |total_price|created_at         |products                                                                       |
+---+-----------+-------------------+-------------------------------------------------------------------------------+
|1  |250.50     |2024-12-01 10:00:00|[{'Ghormeh sabzi', 1, 101}, {'Musir Yogurt', 2, 102}, {'Cola', 1, 103}]   |
|2  |100.00     |2024-12-01 11:00:00|[{'Jujeh kebab', 3, 201}, {'Zeytoon', 1, 202}]                           |
+---+-----------+-------------------+-------------------------------------------------------------------------------+
```

Evaluation Criteria

- **Correctness**: Adherence to the specified requirements.
- **Code Quality**: Clean, readable, and modular code.
- **Realism**: Appropriateness of generated data.
- **Documentation**: Clarity and completeness of the provided `README.md`.
- **Containerization**: Functionality and reliability of the Docker Compose setup.

Good luck!