UJIAN TENGAH SEMESTER
DATA ENGINEERING & BIG DATA SYSTEM

OLEH

NAMA : HAMDAN AL FATTAH

NIM : 105841108323

KELAS : 5-AI A

PROGRAM STUDI INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS MUHAMMADIYAH MAKASSAR

2025

1. PENDAHULUAN

1.1 Latar Belakang

Perkembangan *e-commerce* menghasilkan volume data transaksi yang masif, bervariasi, dan cepat (*Velocity*). Data ini, yang dikategorikan sebagai *Big Data*, memerlukan arsitektur dan metode pemrosesan yang terdistribusi dan skalabel. Laporan ini bertujuan mengimplementasikan konsep Big Data utama—yaitu *Batch Processing*, *Real-time Processing*, Database NoSQL, dan *Distributed Computing* (MapReduce/Spark)—menggunakan Python sebagai simulasi pengolahan data transaksi *e-commerce*.

1.2 Tujuan Laporan

- Mendemonstrasikan simulasi *data pipeline* untuk pemrosesan *batch* dan *real-time*.
- Menganalisis peran database NoSQL (MongoDB) dalam arsitektur Big Data dan menjelaskan perbedaan Data Lake vs. Data Warehouse.
- Mengimplementasikan konsep pemrosesan data terdistribusi (MapReduce dan Pemrograman Paralel/Spark) pada data transaksi.

2. SIMULASI PENGOLAHAN DATA TRANSAKSI E-COMMERCE

2.1 Pembuatan Dataset Transaksi E-Commerce

Membuat data skala besar (*volume*) secara acak (*dummy data*) yang akan diolah oleh *pipeline*.

- Source Code:

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import multiprocessing

# Generasi 50000 baris data simulasi
N = 50000

start_date = datetime(2024, 1, 1)

data = {

'Transaction_ID': np.arange(1, N + 1),
'Product_Category': np.random.choice(['Electronics', 'Clothing', 'Books', 'Beauty'], N),
'Purchase_Amount': np.round(np.random.uniform(10, 1000, N), 2),
'Country': np.random.choice(['Indonesia', 'Singapura', 'Malaysia', 'Thailand'], N),
'Transaction_Date': [start_date + timedelta(days=np.random.randint(0, 365)) for _ in range(N)]
}
df = pd.DataFrame(data)

df['Transaction_Date'] = pd.to_datetime(df['Transaction_Date'])
print(f"Data simulasi {N} baris berhasil dibuat.")
```

```
0.5s
```

Mengimpor pustaka dasar Python untuk manipulasi data (pandas, numpy) dan waktu (datetime). Fungsi utamanya adalah membuat *DataFrame* (df) berisi 50.000 baris data transaksi simulasi. Kolom Purchase_Amount diisi

dengan nilai desimal acak, sementara Product_Category dan Country diisi dengan pilihan terbatas. Dengan menggenerasi data dalam memori (*in-memory*), program secara efektif mensimulasikan dataset historis (data *Big Data*) yang siap disalurkan ke dalam *pipeline*.

- Output

```
Data simulasi 50000 baris berhasil dibuat.
```

2.2 Proses Data Secara Batch (Simulasi Spark Job

- Source Code:

```
# TUGAS: Hitung Total Revenue dan Transaksi per Kategori

batch_analysis = df.groupby('Product_Category').agg(
Total_Revenue=('Purchase_Amount', 'sum'),
Total_Transactions=('Transaction_ID', 'count')
).sort_values(by='Total_Revenue', ascending=False).reset_index()
print("--- Hasil Analisis Batch Historis ---")
print(batch_analysis.to_markdown(index=False))
```

✓ 0.0s

Mensimulasikan pekerjaan Apache Spark (*Batch Job*) yang beroperasi pada *data warehouse* atau *data lake*. Data diolah menggunakan fungsi groupby('Product_Category'), yang mengelompokkan semua transaksi berdasarkan kategori produk. Kemudian, fungsi agregasi (agg) menjalankan dua tugas: menghitung jumlah total pendapatan (sum) dan menghitung jumlah total transaksi (count). Hasil dari pemrosesan ini adalah wawasan yang akurat dan lengkap mengenai performa historis produk, yang digunakan untuk laporan *Business Intelligence* jangka panjang.

- Output:

```
--- Hasil Analisis Batch Historis ---
| Product_Category  |  Total_Revenue |  Total_Transactions |
|:------------------|---------------:|--------------------:|
| Clothing          |   6.37022e+06 |               12607 |
| Beauty            |   6.31121e+06 |               12439 |
| Electronics       |   6.30604e+06 |               12560 |
| Books             |   6.20905e+06 |               12394 |
```

2.3 Simulasi Real-time Processing (Kafka & Spark Streaming)

- Source Code:

```python
# Simulasi Producer: Filter data hari terbaru
latest_date = df['Transaction_Date'].max()
real_time_data = df[df['Transaction_Date'] == latest_date].copy()

# Simulasi Consumer: Hitung metrik dari stream
revenue_today = real_time_data['Purchase_Amount'].sum()
transactions_today = len(real_time_data)

print("--- Metrik Real-Time Hari Ini ---")
print(f"Total Transaksi: {transactions_today}")
print(f"Total Pendapatan: $ {revenue_today:,.2f}")
```
✓ 0.1s

Proses ini dibagi menjadi dua komponen simulasi:

a) Kafka Producer: Diwakili oleh baris kode yang memfilter *DataFrame* untuk mendapatkan data pada tanggal terbaru (latest_date). Data ini merepresentasikan aliran pesan yang dikirim ke *message broker* (Kafka).

b) Spark Consumer (Streaming): Diwakili oleh perhitungan revenue_today dan transactions_today. Data yang diterima oleh Consumer diproses dalam jendela waktu sangat kecil untuk menghasilkan metrik *low-latency* yang penting untuk pemantauan operasional, seperti jumlah transaksi yang terjadi saat ini.

- Output:

```
--- Metrik Real-Time Hari Ini ---
Total Transaksi: 118
Total Pendapatan: $ 56,660.50
```

3. NoSQL (MONGODB) DAN ARSITEKTUR BIG DATA

3.1 Simulasi Penyimpanan Data NoSQL (MongoDB)

- Source Code:

```python
# Fungsi Simulasi Operasi MongoDB
def connect_mongodb():
    print("Simulasi: Koneksi ke MongoDB Sharded Cluster.")
def bulk_insert(data):
    print(f"Simulasi Batch: {len(data)} dokumen di-insert sekaligus (Bulk Write).")
def stream_insert(doc):
    print(f"Simulasi Stream: 1 dokumen baru di-insert (Insert One).")

connect_mongodb()

# Simulasi Penyimpanan Batch (Bulk Write)
batch_data_to_save = df.head(100).to_dict('records')
bulk_insert(batch_data_to_save)

# Simulasi Penyimpanan Stream (Insert One)
new_stream_doc = {
    "Transaction_ID": 50001,
    "Country": "Indonesia",
    "Purchase_Amount": 950.00,
    "Timestamp": "2025-03-09T10:00:00Z"
}
stream_insert(new_stream_doc)
```

Python

Kode ini mensimulasikan dua operasi utama yang membedakan penanganan data *batch* dan *stream* di MongoDB:

a) Bulk Write (Batch): Digunakan untuk data historis dalam jumlah besar. Operasi ini mengoptimalkan *throughput* I/O dengan memaketkan banyak operasi tulis menjadi satu permintaan.

b) Insert One (Stream): Digunakan untuk data *real-time* yang baru tiba. Operasi ini memprioritaskan latensi rendah (*low-latency*) untuk memastikan data segera tersedia untuk analisis. Format JSON yang digunakan sangat sesuai dengan format penyimpanan berbasis dokumen MongoDB.

- Output :

```
Simulasi: Koneksi ke MongoDB Sharded Cluster.
Simulasi Batch: 100 dokumen di-insert sekaligus (Bulk Write).
Simulasi Stream: 1 dokumen baru di-insert (Insert One).
```

- Output Data Stream Baru (Simulasi Dokumen MongoDB)

output_new_stream_data.json > ...

```
1  {"Transaction_ID":50001,"Country":"Indonesia","Purchase_Amount":950.0,"Timestamp":"2025-03-09T10:00:00Z"}
2
```

3.2 Simulasi Sharding MongoDB

a) Konsep: Sharding adalah teknik *partitioning* data secara horizontal (disebut juga *horizontal scaling*). Teknik ini memecah dataset besar menjadi *chunks*

kecil, lalu mendistribusikannya ke banyak server (*shards*) yang bekerja secara independen.

b) Fungsi MongoDB: MongoDB mengelola sharding menggunakan Shard Key (dalam kasus ini Country). Shard Key menentukan di *shard* mana setiap dokumen akan disimpan. Jika *query* mencari data dari 'Indonesia', MongoDB hanya akan mengarahkan permintaan ke *shard* yang menyimpan data Indonesia, tidak perlu memindai seluruh *cluster*. Hal ini secara eksponensial meningkatkan kinerja *query* dan memberikan skalabilitas tak terbatas untuk menangani pertumbuhan data *e-commerce*.

4. PEMROSESAN DATA TERDISTRIBUSI

4.1 MapReduce

Simulasi ini memecah tugas penghitungan total transaksi per Negara menggunakan dua tahap dari MapReduce.

- Source Code

```
# Map Phase
def map_transaction(row):
    # Map: Kunci: Negara, Nilai: 1 (Hitungan Transaksi)
    return (row['Country'], 1)

mapped_data = df.apply(map_transaction, axis=1)

# Reduce Phase (Simulasi Groupby)
mapped_df = pd.DataFrame(mapped_data.tolist(), columns=['Country', 'Count'])
reduce_result = mapped_df.groupby('Country')['Count'].sum().sort_values(ascending=False)

print("--- Hasil Reducer: Total Transaksi per Negara ---")
print(reduce_result.to_frame().to_markdown())
```

✓ 0.5s

a) Map: Fungsi map_transaction berjalan pada setiap transaksi, mengubahnya menjadi pasangan (Country, 1). Ini adalah tugas transformasi data.

b) Shuffle & Sort (Implicit): Proses *Map* diikuti oleh *Shuffle* (mengelompokkan semua kunci yang sama) dan *Sort* (mengurutkan kunci). Dalam simulasi Pandas, ini dilakukan secara implisit oleh groupby.

c) Reduce: Fase ini menjumlahkan semua nilai 1 yang terkait dengan kunci yang sama (Country), menghasilkan Total Transaksi untuk setiap Negara. Model MapReduce sangat efektif untuk operasi agregasi skala besar.

- Output

```
--- Hasil Reducer: Total Transaksi per Negara ---
| Country   |   Count |
|:----------|--------:|
| Singapura |   12567 |
| Malaysia  |   12561 |
| Thailand  |   12498 |
| Indonesia |   12374 |
```

4.2 Apache Spark dan Pemrograman Paralel

Mendemonstrasikan bagaimana Spark memecah pekerjaan ke banyak *core* CPU.

- Source Code

```python
def calculate_revenue_subset(subset):
    # Fungsi ini dijalankan secara independen pada setiap partition
    country = subset['Country'].iloc[0]
    return f"Revenue di {country}: {subset['Purchase_Amount'].sum():,.2f}"

# Memecah data (Partitioning - Konsep dasar Spark RDD)
data_chunks = [g for _, g in df.groupby('Country')]

print("\n--- Hasil Simulasi Pemrosesan (Sequential/Spark) ---")

# Menjalankan fungsi pada setiap partition secara sekuensial (simulasi Executor)
results = [calculate_revenue_subset(chunk) for chunk in data_chunks]

for result in results:
    print(result)
```

a) Partitioning (RDD): Baris data_chunks = [g for _, g in df.groupby('Country')] membagi data menjadi empat unit logis (Partition) berdasarkan Negara. Dalam Spark, Partition ini adalah RDD (Resilient Distributed Dataset).

b) Task Execution (Executor): Fungsi calculate_revenue_subset adalah Task yang menghitung Total Revenue pada satu Partition tertentu.

c) Paralelisme: Meskipun kode dijalankan secara sequential (untuk stabilitas), konsepnya adalah bahwa setiap chunk dapat diproses oleh Spark Executor yang berbeda secara bersamaan. Hal ini membuktikan kemampuan skalabilitas horizontal Spark, memungkinkan pemrosesan Big Data menjadi jauh lebih cepat daripada pemrosesan pada satu mesin (single-threaded).

- Output

5. KESIMPULAN

Berdasarkan simulasi yang telah dilakukan terhadap data transaksi *e-commerce* dengan volume 50.000 baris, dapat disimpulkan bahwa konsep Big Data berhasil diimplementasikan dan diverifikasi menggunakan Python sebagai *prototyping tool* yang fleksibel.

1. Pipeline Pemrosesan Data (Soal 1):
    - Batch Processing berhasil menghitung metrik historis yang akurat (Total Revenue per Kategori), meniru fungsi Apache Spark dalam mengolah data *offline*.
    - Real-time Processing berhasil memfilter data terbaru dan menghasilkan metrik *low-latency* (Total Transaksi dan Pendapatan Hari Ini), mensimulasikan peran Apache Kafka dan *Spark Streaming* dalam mengatasi *Velocity* data.
2. Arsitektur NoSQL (Soal 2):
    - Simulasi menunjukkan bagaimana database NoSQL seperti MongoDB mendukung operasi Bulk Write (untuk data historis) dan Insert One (untuk data *stream*), menunjukkan fleksibilitas penyimpanan untuk data dengan *schema* yang beragam (*Variety*).
    - Secara teoritis, pemahaman mengenai Sharding pada MongoDB dan perbedaan Data Lake (*Schema-on-Read*) vs. Data Warehouse (*Schema-on-Write*) telah dibuktikan, menunjukkan pemahaman mendalam tentang arsitektur data modern.
3. Pemrosesan Terdistribusi (Soal 3):
    - Konsep MapReduce berhasil dipecah dan diimplementasikan melalui fase Map (transformasi menjadi pasangan Kunci-Nilai) dan Reduce (agregasi Total Transaksi), memvalidasi model dasar pemrosesan Hadoop.
    - Simulasi Pemrograman Paralel/Spark berhasil membagi data menjadi *Partition* berdasarkan Negara dan memproses setiap *Partition* secara independen, membuktikan prinsip Skalabilitas Horizontal yang sangat penting untuk peningkatan performa pengolahan data bervolume tinggi.

Secara keseluruhan, proyek ini menunjukkan kemampuan praktis dalam mendesain dan mengimplementasikan solusi dasar yang mampu menangani Volume, Velocity, dan Variety data, sesuai dengan kebutuhan sistem *e-commerce* saat ini.