รายงาน: ระบบ Data Streaming และ Real-Time Analytics

1. แหล่งข้อมูล (Data Sources)

1.1. Source 1: Pageview (Stream Datagen) - Topic1

- **What**: แหล่งข้อมูลนี้สร้างข้อมูลการเข้าชมหน้าเว็บ (Pageview) โดยใช้เครื่องมือ Datagen ที่เชื่อมต่อกับ Kafka Topic Topic1
- **Why**: ข้อมูล Pageview จำเป็นสำหรับการวิเคราะห์พฤติกรรมผู้ใช้ เช่น หน้าเว็บที่มีการเข้าชมสูงและระยะเวลาที่ผู้ใช้อยู่ในแต่ละหน้า
- **How**:
    - **Input**: ข้อมูลที่ประกอบด้วย Userid, Pageid, และ Viewtime สร้างโดย Datagen
    - **Process**: ใช้คำสั่ง Kafka-Topics เพื่อสร้าง Topic Topic1 และใช้ Curl เชื่อมต่อ Datagen-Pageviews กับ Kafka เพื่อลงข้อมูลแบบสตรีมเข้าสู่ Topic1
    - **Output**: ข้อมูล Pageview ในรูปแบบ Json จัดเก็บใน Kafka Topic Topic1

1.2. Source 2: Users_ (Stream Datagen) - Topic2

- **What**: แหล่งข้อมูลผู้ใช้ซึ่งประกอบด้วยข้อมูลส่วนตัว เช่น เพศ, พื้นที่ และเวลาลงทะเบียน โดยใช้ Datagen เชื่อมต่อกับ Kafka Topic Topic2
- **Why**: ข้อมูลผู้ใช้ช่วยให้เข้าใจลักษณะและการกระจายของกลุ่มผู้ใช้ ทำให้สามารถจัดกลุ่มตามความสนใจหรือพฤติกรรมได้
- **How**:
    - **Input**: ข้อมูล Userid, Regionid, Gender, และ Registertime จาก Datagen-Users
    - **Process**: สร้าง Topic Topic2 และใช้คำสั่ง Curl เชื่อมต่อ Datagen-Users เพื่อสร้างข้อมูลผู้ใช้และส่งไปยัง Kafka
    - **Output**: ข้อมูลผู้ใช้ใน Kafka Topic Topic2

1.3. Source 3: ข้อมูลที่ออกแบบเอง (Relational Database) - Topic3

- **What**: ข้อมูลที่ผู้ใช้กำหนดเองในฐานข้อมูลเชิงสัมพันธ์ ประกอบด้วยข้อมูลเกี่ยวกับภูมิภาค (Region) เช่น ชื่อจังหวัด , จำนวนประชากร และขนาดพื้นที่
- **Why**: ข้อมูลนี้มีบทบาทสำคัญในการให้ข้อมูลเชิงภูมิศาสตร์และประชากร เพื่อใช้ในการวิเคราะห์เชิงลึกและการจัดกลุ่ม
- **How**:
    - **Input**: ข้อมูลเชิงสัมพันธ์ เช่น Regionid, Region_Name, Country, Continent, และ Population
    - **Process**: สร้าง Kafka Topic Topic3 จากนั้นใช้ Kafka-Console-Producer เพื่อส่งข้อมูลจากไฟล์ Topic3relationaldata.Txt ลงใน Kafka
    - **Output**: ข้อมูลภูมิภาคที่ถูกจัดเก็บใน Kafka Topic Topic3

2. ระบบ Kafka

2.1. การตั้งค่า Kafka System

- **What**: Kafka คือแพลตฟอร์มสำหรับสตรีมข้อมูลที่สามารถจัดเก็บและส่งผ่านข้อมูลแบบเรียลไทม์ภายในระบบ
- **Why**: Kafka ช่วยให้ระบบสามารถจัดเก็บข้อมูลจากแหล่งต่างๆ ได้ในที่เดียว ทำให้การรับส่งข้อมูลมีประสิทธิภาพและสามารถขยายขนาดได้ง่าย
- **How**:
    - **Input**: ข้อมูลจากแหล่งข้อมูลทั้ง 3 แหล่ง
    - **Process**:
        1. **จัดการ Partitions**: ข้อมูลใน Kafka จะถูกแบ่งเป็น 5 Partitions เพื่อให้การประมวลผลเร็วขึ้น
        2. **การกระจาย Brokers**: ใช้ 3 Brokers เพื่อรองรับปริมาณข้อมูลสูงและเสริมความเสถียรภาพในกรณีที่ Broker ใดหยุดทำงาน
        3. **Schema Register**: กำหนด Schema เพื่อให้แน่ใจว่าข้อมูลมีโครงสร้างเดียวกัน
        4. **Kafka Connect**: ใช้เชื่อมต่อ Kafka กับ Datagen และฐานข้อมูล Relational
    - **Output**: ข้อมูลจาก Data Sources ถูกส่งเข้า Kafka และเก็บไว้ใน 8 Topics เพื่อเตรียมพร้อมสำหรับการประมวลผลต่อไปใน Ksqldb และ Apache Pinot

3. การดำเนินการของ Ksqldb

3.1. การทำความสะอาดและแปลงข้อมูล - Topic4

- **What**: Ksqldb ใช้ SQL ในการทำความสะอาดและปรับรูปแบบข้อมูล เพื่อให้ข้อมูลมีความถูกต้องและพร้อมสำหรับการใช้งาน
- **Why**: การแปลงข้อมูลช่วยปรับให้ข้อมูลอยู่ในรูปแบบที่ใช้สะดวกและพร้อมสำหรับการวิเคราะห์
- **How**:
    - **Input**: ข้อมูลจาก Topic2 เช่น Userid, Regionid, Gender, Registertime
    - **Process**: สร้างตาราง Users_Formatted ด้วยคำสั่ง Timestamptostring ใน Ksqldb เพื่อปรับ Registertime ให้อยู่ในรูปแบบวันที่ที่อ่านง่าย
    - **Output**: ข้อมูลผู้ใช้ที่ถูกจัดเก็บใน Topic4 ซึ่งพร้อมสำหรับการใช้งานในขั้นตอนถัดไป

3.2. การรวมและจัดกลุ่มข้อมูล (Aggregation) - Topic5

- **What**: Ksqldb ใช้ในการรวมข้อมูลจากหลายแหล่งและจัดกลุ่มข้อมูลตามเกณฑ์ที่กำหนด
- **Why**: การรวมข้อมูลจากแหล่งต่างๆ ช่วยให้การวิเคราะห์ข้อมูลง่ายขึ้นและได้ข้อมูลที่ครอบคลุม
- **How**:
    - **Input**: ข้อมูลจาก Topic1_Stream (Pageview), Topic2_Table (User), และ Topic3_Table (Region)
    - **Process**: สร้าง Consolidate_Stream ด้วยการ Join ข้อมูลจาก Topic1_Stream, Topic2_Table, และ Topic3_Table เพื่อรวมข้อมูลการใช้งานและข้อมูลภูมิภาค
    - **Output**: ข้อมูลที่ถูก Join และจัดกลุ่มใน Topic5

3.3. หน้าต่างข้อมูล (Windows)

3.3.1. Tumbling Window - Topic6

- **What**: สร้างหน้าต่างข้อมูลแบบ Tumbling ซึ่งแบ่งข้อมูลออกเป็นช่วงเวลาที่ไม่ทับซ้อนกัน
- **How**:
    - **Input**: ข้อมูลจาก Consolidate_Stream
    - **Process**: สร้างหน้าต่างด้วย Window Tumbling โดยเก็บข้อมูลการดูเพจในช่วง 1 นาที

- **Output**: ข้อมูลที่แบ่งตาม Tumbling Window ใน Topic6

3.3.2. **Hopping Window** - Topic7

- **What:** หน้าต่างข้อมูลแบบ Hopping ที่มีการซ้อนทับกัน
- **How:**
  - **Input:** ข้อมูลจาก Consolidate_Stream
  - **Process:** ใช้คำสั่ง Window Hopping โดยสร้างหน้าต่างขนาด 5 วินาทีและขยับทุก 2 วินาที
  - **Output:** ข้อมูลที่แบ่งตาม Hopping Window ใน Topic7

3.3.3. **Session Window** - Topic8

- **What: หน้าต่างข้อมูลแบบ Session Window ซึ่งปรับขนาดตามพฤติกรรมของผู้ใช้**
- **How:**
  - **Input:** ข้อมูลจาก Consolidate_Stream
  - **Process:** ใช้ Window Session เพื่อแบ่งข้อมูลตามกิจกรรมของผู้ใช้ในแต่ละ Session
  - **Output:** ข้อมูล Session ที่จัดเก็บใน Topic8

4. การทดสอบความถูกต้องของข้อมูล

- **What**: การทดสอบเพื่อยืนยันว่าการประมวลผลใน Ksqldb ให้ผลลัพธ์ที่ถูกต้อง
- **Why**: การตรวจสอบความถูกต้องช่วยให้มั่นใจได้ว่าข้อมูลมีคุณภาพ
- **How**:
  - **Input**: ข้อมูลที่จัดการใน Ksqldb เช่นข้อมูลใน Topic6, Topic7, และ Topic8
  - **Process**: ใช้คำสั่ง SQL ใน Ksqldb เพื่อทดสอบการประมวลผลและตรวจสอบข้อมูลในแต่ละขั้นตอน
  - **Output**: ข้อมูลที่ผ่านการทดสอบและมีความถูกต้องตามที่คาดหวัง

5. **Apache Pinot**

- **What:** Apache Pinot เป็นระบบฐานข้อมูลที่ออกแบบมาเพื่อรองรับการจัดเก็บและวิเคราะห์ข้อมูลขนาดใหญ่แบบเรียลไทม์ มีจุดเด่นที่สามารถตอบสนองการสืบค้นข้อมูลได้อย่างรวดเร็วและรองรับการวิเคราะห์ข้อมูลเชิงลึก โดยเฉพาะกับข้อมูลที่ไหลมาอย่างต่อเนื่อง เช่น ข้อมูลสตรีมจาก Kafka ซึ่งเหมาะสำหรับการนำมาใช้ในระบบที่ต้องการการวิเคราะห์ข้อมูลแบบทันที (real-time analytics)
- **Why:** การใช้ Apache Pinot ในระบบนี้ช่วยให้สามารถดึงข้อมูลที่เกิดขึ้นแบบเรียลไทม์จาก Kafka มาวิเคราะห์ในรูปแบบของการ query ข้อมูลได้อย่างรวดเร็ว นอกจากนี้ Apache Pinot ยังช่วยสร้างดัชนี (indexing) ที่มีประสิทธิภาพ ทำให้การเข้าถึงข้อมูลที่จัดเก็บในระบบทำได้โดยง่ายและรวดเร็ว นำมาซึ่งการแสดงผลข้อมูลที่ทันต่อเหตุการณ์ ช่วยให้ผู้ใช้สามารถตัดสินใจได้จากข้อมูลแบบเรียลไทม์บน dashboard และการวิเคราะห์เชิงลึกได้ทันที
- **How:**
    - **Input (data):** ข้อมูลที่ผ่านการประมวลผลใน ksqlDB และถูกเก็บไว้ใน Kafka topics (เช่น topic5, topic6, และ topic8) จะถูกส่งเข้าไปยัง Apache Pinot ผ่าน Kafka ซึ่งมีโครงสร้างข้อมูลในแต่ละ topic ที่กำหนด schema เพื่อให้การจัดการข้อมูลใน Pinot เป็นระบบระเบียบ
        - ตัวอย่างเช่น ข้อมูลที่ใช้ใน Consolidate_Stream (topic5) จะมีฟิลด์ เช่น USERID, PAGEID, GENDER, REGION_NAME, COUNTRY, CONTINENT, และ VIEWTIME
        - ข้อมูลที่ใช้ใน CountryViews_Tumbling (topic6) จะมีฟิลด์ CONTINENT, REGIONNAME, VIEWCOUNT, STARTWINDOW, และ ENDWINDOW
        - ข้อมูลที่ใช้ใน Continent_Session_Analysis (topic8) จะมีฟิลด์ COUNTRY, PAGEVISITCOUNT, SESSIONLENGTHSECONDS, SESSIONSTART, และ SESSIONEND
    - **Process:**
        - Create Schema: การสร้าง schema สำหรับแต่ละ topic เป็นขั้นตอนแรก โดยระบุชื่อ schema เช่น Consolidate, CountryViews_Tumbling, และ Continent_Session_Analysis พร้อมกำหนดฟิลด์ใน schema ตามประเภทข้อมูลและฟิลด์ที่จำเป็นในแต่ละ topic เช่น dimensionFieldSpecs, metricFieldSpecs, และ dateTimeFieldSpecs
            - dimensionFieldSpecs สำหรับการกำหนดฟิลด์ข้อมูลที่ใช้ระบุคุณลักษณะ เช่น USERID, REGIONNAME
            - metricFieldSpecs สำหรับการกำหนดฟิลด์ที่ใช้คำนวณ เช่น VIEWCOUNT, PAGEVISITCOUNT
            - dateTimeFieldSpecs สำหรับการกำหนดฟิลด์ที่เป็นเวลา เช่น VIEWTIME, SESSIONSTART

- Create Tables: การสร้าง table ใน Apache Pinot ตาม schema ที่กำหนดไว้ โดยการสร้าง table มีการกำหนด table name เช่น Consolidate_REALTIME, CountryViews_Tumbling_REALTIME, และ Continent_Session_Analysis_REALTIME รวมถึงการตั้งค่าที่สำคัญ เช่น:
  - timeColumnName เช่น VIEWTIME หรือ SESSIONSTART เพื่อระบุฟิลด์เวลาที่จะใช้ในการจัดเก็บข้อมูล
  - streamConfigs เพื่อเชื่อม Kafka topic ที่ระบุเช่น topic5, topic6, และ topic8 กับ Pinot โดยระบุ broker และตั้งค่า consumer type, offset reset, และ decoder
- Load Data: ข้อมูลจาก Kafka topics จะถูกโหลดเข้าสู่ Pinot tables ตาม schema และ table configuration ที่ตั้งค่าไว้ ทำให้ข้อมูลมีโครงสร้างที่พร้อมสำหรับการสืบค้นและการวิเคราะห์แบบเรียลไทม์

- Output (results): ผลลัพธ์จากการใช้งาน Apache Pinot คือข้อมูลที่พร้อมสำหรับการสืบค้นและการวิเคราะห์ในรูปแบบ query ซึ่งสามารถดึงข้อมูลมาใช้ใน dashboard หรือการวิเคราะห์ได้ทันที ตัวอย่างเช่น การ query จำนวนการเข้าชมหน้าเว็บโดยเฉลี่ยต่อผู้ใช้ในช่วงเวลาต่างๆ หรือการวิเคราะห์พฤติกรรมการใช้งานตามภูมิภาค ทั้งหมดนี้ช่วยให้สามารถนำเสนอข้อมูลแบบ real-time analytics ที่มีประสิทธิภาพ

6. Streamlit Dashboard

- **What:** Streamlit เป็นแพลตฟอร์มที่ช่วยในการสร้าง Dashboard แบบ interactive ซึ่งช่วยให้ผู้ใช้สามารถวิเคราะห์ข้อมูลแบบเรียลไทม์ได้อย่างมีประสิทธิภาพและสะดวกสบาย โดยใช้การเชื่อมต่อข้อมูลจาก Apache Pinot ผ่าน SQL query และนำเสนอข้อมูลในรูปแบบที่เป็นภาพ เช่น กราฟแท่ง (Bar Chart) และกราฟวงกลม (Pie Chart) เพื่อให้เข้าใจได้ง่ายและสามารถโต้ตอบกับข้อมูลได้
- **Why:** การใช้ Streamlit ช่วยให้การวิเคราะห์และแสดงผลข้อมูลแบบเรียลไทม์จาก Apache Pinot ทำได้ง่ายและสะดวก Streamlit Dashboard มีความสำคัญในการนำเสนอข้อมูลจากหลายแหล่งในรูปแบบกราฟิกที่ผู้ใช้สามารถโต้ตอบได้ ซึ่งเป็นประโยชน์ในการวิเคราะห์ข้อมูลอย่างรวดเร็ว เพื่อช่วยในการตัดสินใจแบบเรียลไทม์
- **How**
  - **Input (data)**: ข้อมูลที่ใช้ใน Streamlit Dashboard มาจาก Apache Pinot โดยมีการ query จาก Kafka topics หลายหัวข้อซึ่งเก็บอยู่ในตาราง Consolidate, CountryViews_Tumbling, และ Continent_Session_Analysis
    - ตัวอย่างข้อมูลประกอบด้วย:
      - ข้อมูลการเข้าชมเพจ (Page Visits) จาก Consolidate
      - ข้อมูลการดูเพจตามภูมิภาค (Regional Page Views) จาก CountryViews_Tumbling

- ข้อมูลความยาว session เฉลี่ย (Average Session Length) จาก Continent_Session_Analysis

- **Process**:
  - **เชื่อมต่อกับ Apache Pinot**: ใช้ pinotdb ในการเชื่อมต่อกับ Apache Pinot เพื่อดึงข้อมูลและส่ง query ไปยังแต่ละตารางผ่านคำสั่ง SQL
  - **Query ข้อมูลสำหรับกราฟแต่ละประเภท**:
    - **Visualization 1**: แสดงจำนวนการเข้าชมเพจตามประเทศและเพศโดยใช้คำสั่ง SQL เพื่อจัดกลุ่มข้อมูลจาก Consolidate ตาม COUNTRY และ GENDER
    - **Visualization 2**: คำนวณค่าเฉลี่ย VIEWTIME ต่อประชากรในแต่ละประเทศโดยใช้ SUM ของ VIEWTIME หารด้วย POPULATION จากตาราง Consolidate
    - **Visualization 3**: แสดงความนิยมของภูมิภาคต่างๆ ผ่านการรวม VIEWCOUNT จาก CountryViews_Tumbling จัดกลุ่มตาม REGIONNAME และแสดงผลเป็นกราฟวงกลม
    - **Visualization 4**: แสดงจำนวนการเข้าชมต่อ session โดยเฉลี่ยในแต่ละประเทศจากตาราง Continent_Session_Analysis ผ่านการคำนวณ AVG(PAGEVISITCOUNT)
    - **Visualization 5**: แสดงจำนวนการเข้าชมเพจแบบเรียลไทม์ตามภูมิภาค และความยาว session เฉลี่ยในแต่ละประเทศ ซึ่งเป็นการรวมข้อมูลแบบทันทีเพื่อแสดงศักยภาพการติดตามข้อมูลเรียลไทม์ของระบบ
  - **ตั้งค่าการรีเฟรชอัตโนมัติ**: ใช้ JavaScript เพื่อรีเฟรชหน้า Dashboard ทุกๆ 2 วินาที ทำให้ข้อมูลแสดงผลใหม่ได้อย่างต่อเนื่องและอัปเดตตามข้อมูลจริงที่ไหลเข้ามาในระบบ
  - **การตั้งค่าโซนเวลา**: ตั้งค่าให้ Dashboard แสดงเวลาตามเขตเวลา GMT+7 (กรุงเทพฯ) เพื่อความสะดวกในการติดตามข้อมูลแบบเรียลไทม์ในเวลาท้องถิ่น
- **Output (results)**: Streamlit Dashboard ที่ประกอบด้วยกราฟและแผนภูมิ 5 ส่วนที่แสดงข้อมูลแบบ interactive
  - ผู้ใช้สามารถเห็นการเปลี่ยนแปลงของข้อมูลแบบเรียลไทม์ ทั้งการเข้าชมเพจในภูมิภาคต่างๆ ความนิยมของแต่ละประเทศ และพฤติกรรมการใช้งานของผู้ใช้ตาม session โดยข้อมูลจะถูกอัปเดตทุก 2 วินาที ช่วยให้การตัดสินใจแบบทันทีทำได้สะดวกและแม่นยำ

Source Code

Upload the Docker Compose configuration file to the remote server

scp -i Hafiz_Keypair_1.pem docker-compose.yml ubuntu@ec2-122-248-218-143.ap-southeast-1.compute.amazonaws.com:/home/ubuntu

Access the remote server using SSH

ssh -i "Hafiz_Keypair_1.pem" ubuntu@ec2-122-248-218-143.ap-southeast-1.compute.amazonaws.com

Access the ksqlDB CLI within Docker to run SQL commands for streaming data

sudo docker exec -it ksqldb-cli ksql http://ksqldb-server:8088

Access the Kafka broker within Docker to create topics and manage data

sudo docker exec -it broker bash

Create 8 Kafka topics with 5 partitions and 3 replicas for data redundancy and distribution

kafka-topics --bootstrap-server broker:29092 --create --topic topic1 --partitions 5 --replication-factor 3

kafka-topics --bootstrap-server broker:29092 --create --topic topic2 --partitions 5 --replication-factor 3

kafka-topics --bootstrap-server broker:29092 --create --topic topic3 --partitions 5 --replication-factor 3

kafka-topics --bootstrap-server broker:29092 --create --topic topic4 --partitions 5 --replication-factor 3

kafka-topics --bootstrap-server broker:29092 --create --topic topic5 --partitions 5 --replication-factor 3

kafka-topics --bootstrap-server broker:29092 --create --topic topic6 --partitions 5 --replication-factor 3

kafka-topics --bootstrap-server broker:29092 --create --topic topic7 --partitions 5 --replication-factor 3

kafka-topics --bootstrap-server broker:29092 --create --topic topic8 --partitions 5 --replication-factor 3

Kafka Connectors for Data Generation

Configure a datagen connector to generate page view events for topic1

curl -X POST -H "Content-Type: application/json" \

-d '{

```
  "name": "datagen-pageviews",
  "config": {
    "connector.class": "io.confluent.kafka.connect.datagen.DatagenConnector",
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "kafka.topic": "topic1",
    "quickstart": "pageviews",
    "interval.type": "random",
    "interval.range.min": "1",
    "interval.range.max": "100"
  }
}' http://localhost:8083/connectors


# Configure a datagen connector to generate user data for topic2
curl -X POST -H "Content-Type: application/json" \
-d '{
  "name": "datagen-users",
  "config": {
    "connector.class": "io.confluent.kafka.connect.datagen.DatagenConnector",
    "kafka.topic": "topic2",
    "quickstart": "users"
  }
}' http://localhost:8083/connectors
```

Kafka Console Producer

```
# Load relational data from topic3relationaldata.txt into topic3 using Kafka console producer

docker exec -i broker kafka-console-producer \

--broker-list broker:29092 \

--topic topic3 \

--property "parse.key=true" \

--property "key.separator=:" \

--property "value.format=JSON" \

< topic3relationaldata.txt
```

ksqlDB Commands

```
# Create a stream for topic1 to capture real-time page view data

CREATE STREAM topic1_stream (

    userid VARCHAR,

    pageid VARCHAR,

    viewtime BIGINT

) WITH (

    KAFKA_TOPIC='topic1',

    VALUE_FORMAT='JSON'

);



# Create a table for user data from topic2 with a primary key of USERID

CREATE TABLE topic2_table (
```

```
    USERID STRING PRIMARY KEY,
    REGIONID STRING,
    GENDER STRING,
    REGISTERTIME BIGINT
) WITH (
    KAFKA_TOPIC = 'topic2',
    VALUE_FORMAT = 'JSON'
);

# Create a table for relational data from topic3, which includes geographical information
CREATE TABLE topic3_table (
    regionid VARCHAR PRIMARY KEY,
    region_name VARCHAR,
    country VARCHAR,
    continent VARCHAR,
    population INTEGER
) WITH (
    KAFKA_TOPIC='topic3',
    VALUE_FORMAT='JSON'
);

# Create a table that formats user registration time to a readable format and stores in topic4
CREATE TABLE users_formatted WITH (
```

```
    KAFKA_TOPIC='topic4',
    VALUE_FORMAT='JSON'
) AS
SELECT
    USERID,
    REGIONID,
    GENDER,
    TIMESTAMPTOSTRING(REGISTERTIME, 'yyyy-MM-dd HH:mm:ss') AS REGISTERTIME_FORMATTED
FROM topic2_table;
```

Consolidate Stream

```
# Create a consolidated stream that joins data from multiple topics for aggregated analysis
CREATE STREAM Consolidate_Stream WITH (
   KAFKA_TOPIC = 'topic5',
   VALUE_FORMAT = 'JSON'
) AS
  SELECT
        topic1_stream.userid AS UserId,
        topic1_stream.pageid,
        topic1_stream.viewtime,
        topic2_table.regionid,
        topic2_table.gender,
        topic3_table.region_name,
        topic3_table.country,
```

```
    topic3_table.continent,
    topic3_table.population
  FROM topic1_stream
  LEFT JOIN topic2_table ON topic1_stream.userid = topic2_table.userid
  LEFT JOIN topic3_table ON topic2_table.regionid = topic3_table.regionid;
```

Windowed Tables for Aggregated Analysis

```
# Create a tumbling window table for analyzing page views per country in 1-minute intervals, stored in topic6
CREATE TABLE CountryViews_Tumbling WITH (
   KAFKA_TOPIC = 'topic6',
   VALUE_FORMAT = 'JSON',
   PARTITIONS = 5,
   REPLICAS = 3
) AS
SELECT
   country,
   LATEST_BY_OFFSET(continent) AS Continent,
   LATEST_BY_OFFSET(region_name) AS RegionName,
   COUNT(*) AS ViewCount,
   WINDOWSTART AS StartWindow,
   WINDOWEND AS EndWindow
FROM Consolidate_Stream
WINDOW TUMBLING (SIZE 1 MINUTE)
GROUP BY country;
```

Apache Pinot Schemas and Tables

```
# Define the schema for Consolidate_Stream in Apache Pinot, specifying fields and data types
SCHEMA_JSON='{
  "schemaName": "Consolidate",
  "dimensionFieldSpecs": [
    { "name": "USERID", "dataType": "STRING" },
    { "name": "PAGEID", "dataType": "STRING" },
    { "name": "COUNTRY", "dataType": "STRING" }
  ],
  "metricFieldSpecs": [
    { "name": "POPULATION", "dataType": "INT" }
  ],
  "dateTimeFieldSpecs": [
    { "name": "VIEWTIME", "dataType": "LONG", "format": "1:MILLISECONDS:EPOCH" }
  ]
}'

# Upload schema to Apache Pinot
curl -X POST -H "Content-Type: application/json" -d "$SCHEMA_JSON" http://localhost:9000/schemas

# Define and create the real-time table in Apache Pinot based on the Consolidate schema
TABLE_JSON='{
```

```
  "tableName": "Consolidate_REALTIME",
  "tableType": "REALTIME",
  "segmentsConfig": {
    "schemaName": "Consolidate",
    "replication": "1"
  },
  "streamConfigs": {
      "streamType": "kafka",
      "stream.kafka.topic.name": "topic5"
  }
}'
```

curl -X POST -H "Content-Type: application/json" -d "$TABLE_JSON" http://localhost:9000/tables

Streamlit Setup

Transfer the Streamlit application to the remote server

scp -i Hafiz_Keypair_1.pem streamlit.py ubuntu@ec2-122-248-218-143.ap-southeast-1.compute.amazonaws.com:/home/ubuntu

Install Python and necessary packages on the remote server

sudo apt update

sudo apt install python3 python3-pip

Install Streamlit and other required libraries for running the dashboard

pip3 install streamlit pandas plotly pinotdb

Run the Streamlit application on the remote server

streamlit run streamlit.py --server.port 8501 --server.enableCORS false

Access the Streamlit dashboard by navigating to the server's IP address with port 8501

http://ec2-122-248-218-143.ap-southeast-1.compute.amazonaws.com:8501/