DADS6005 MIDTERM REALTIME

Data streaming and Realtime analytics System Requirement a. Data source 3 sources

a. Data source 3 sources [1]

i. Source 1 : PageView (stream datagen) (topic1)

PageView เป็นแหล่งข้อมูลที่สร้างข้อมูลจำลองเกี่ยวกับการดูหน้าเว็บของผู้ใช้งาน โดยใช้ Kafka Connect Data Generator เพื่อสร้างข้อมูลในรูปแบบ JSON (ไม่มีการกำหนด schema) และส่งไปยัง Kafka topic 1_pageviews ข้อมูลนี้ประกอบด้วยเวลาในการดู (viewtime), รหัสผู้ใช้งาน (userid), และรหัสหน้าเว็บ (pageid) ซึ่งถูกสร้างขึ้นแบบสุ่มในช่วงเวลาที่กำหนด (random interval) จากนั้นใช้ Kafka Stream ชื่อ pageviews_stream เพื่อดึงข้อมูลเข้าสู่ระบบสำหรับการประมวลผลแบบเรียลไทม์ ข้อมูลนี้มีความสำคัญต่อการวิเคราะห์พฤติกรรมผู้ใช้ เช่น การติดตามการใช้งานเว็บไซต์และการปรับปรุงกลยุทธ์การตลาด ตัวอย่างข้อมูลที่ได้ ได้แก่ {"viewtime": 691001, "userid": "User_6", "pageid": "Page_22"} ซึ่งสามารถนำไปใช้งานในการสร้างรายงานหรือวิเคราะห์เชิงลึกได้

{
"name": "datagen-pageviews",
"connector.class":
"io.confluent.kafka.connect.datagen.DatagenConnector",
"key.converter":
"org.apache.kafka.connect.storage.StringConverter",
"value.converter":
"org.apache.kafka.connect.json.JsonConverter",
"value.converter.schemas.enable": "false",
"kafka.topic": "1_pageviews",
"max.interval": "1000",
"quickstart": "pageviews",
"interval.type": "random",
"interval.range.min": "1",
"interval.range.max": "100
	CREATE STREAM
	pageviews_stream (
	viewtime BIGINT,
	userid VARCHAR,
	pageid VARCHAR
	)
	WITH
	(KAFKA_TOPIC='1_pageviews',
	VALUE_FORMAT='JSON');
		{
		"viewtime": 691001,
		"userid": "User_6",
		"pageid": "Page_22"
		}

ii. Source 2 : Users_ (stream datagen) (topic2)

Users เป็นแหล่งข้อมูลจำลองเกี่ยวกับผู้ใช้งานที่สร้างขึ้นโดย Kafka Connect Data Generator ข้อมูลนี้ส่งไปยัง Kafka topic 2_users ในรูปแบบ JSON (ไม่มีการกำหนด schema) ประกอบด้วยเวลาลงทะเบียน (registertime), รหัสผู้ใช้งาน (userid), ภูมิภาค (regionid), และเพศ (gender) โดยมีการสร้างข้อมูลในช่วงเวลาสุ่ม (random interval) จากนั้นสร้าง Kafka Table ชื่อ users_tb ซึ่งกำหนด userid เป็น Primary Key เพื่อจัดเก็บข้อมูลดังกล่าว ข้อมูลนี้มีความสำคัญต่อการวิเคราะห์พฤติกรรมผู้ใช้งานและเชื่อมโยงข้อมูลในระบบ เช่น การติดตามกลุ่มผู้ใช้งานในแต่ละภูมิภาค ตัวอย่างข้อมูล ได้แก่ {"registertime": 1491293306724, "userid": "User_7", "regionid": "Region_8", "gender": "MALE"} ซึ่งสามารถนำไปใช้ในการวิเคราะห์ข้อมูลเชิงลึกและการตลาดแบบเฉพาะกลุ่ม.

{
"name": "datagen-users",
"connector.class":
"io.confluent.kafka.connect.datagen.DatagenConnector",
"key.converter":
"org.apache.kafka.connect.storage.StringConverter",
"value.converter":
"org.apache.kafka.connect.json.JsonConverter",
"value.converter.schemas.enable": "false",
"kafka.topic": "2_users",
"max.interval": "1000",
"quickstart": "users",
"interval.type": "random",
"interval.range.min": "1",
"interval.range.max": "100"
}
	CREATE TABLE users_tb (
	registertime BIGINT,
	userid VARCHAR PRIMARY KEY,
	regionid VARCHAR,
	gender VARCHAR
	)
	WITH (KAFKA_TOPIC='2_users',
	VALUE_FORMAT='JSON');
		{
		"registertime":
		1491293306724,
		"userid": "User_7",
		"regionid": "Region_8",
		"gender": "MALE"
		}

.iii. Source 3 : Your design (relational database) (topic3)

Orders เป็นแหล่งข้อมูลที่จัดเก็บคำสั่งซื้อของผู้ใช้งานในรูปแบบ relational database โดยใช้ Python script สร้างข้อมูลจำลอง เช่น รหัสคำสั่งซื้อ (orderid), รหัสผู้ใช้งาน (userid), ประเภทสินค้า (product_type), จำนวน (quantity), และสถานะ (status) จากนั้นส่งข้อมูลในรูปแบบ JSON ไปยัง Kafka topic 3_orders ข้อมูลเหล่านี้ถูกนำเข้า Kafka Stream ชื่อ orders_stream เพื่อประมวลผลแบบเรียลไทม์ ข้อมูลคำสั่งซื้อมีความสำคัญต่อการติดตามการขายสินค้า การประเมินยอดขาย และการปรับปรุงกระบวนการจัดส่ง ตัวอย่างข้อมูลที่สร้างขึ้น ได้แก่ {"ORDERID": "4e383b6b-3f38-45ad-baec-c100b17a4323", "USERID": "User_7", "PRODUCT_TYPE": "Tablet", "UNIT_PRICE": 318.12, "QUANTITY": 3, "TOTAL_PRICE": 954.36, "STATE": "Nebraska", "STATUS": "Shipped"} ซึ่งช่วยสนับสนุนการวิเคราะห์ยอดขายและการจัดการคำสั่งซื้อในระบบ.

CREATE STREAM orders_stream (
ORDERID STRING,
USERID STRING,
ORDER_TIMESTAMP STRING,
PRODUCT_TYPE STRING,
UNIT_PRICE DOUBLE,
QUANTITY INT,
TOTAL_PRICE DOUBLE,
STATE STRING,
STATUS STRING
) WITH (
KAFKA_TOPIC='3_orders',
VALUE_FORMAT='JSON'
);
	{
	"ORDERID": "4e383b6b-3f38-45ad-baec-c100b17a4323",
	"USERID": "User_7",
	"ORDER_TIMESTAMP": "2024-11-16T08:33:16.098003",
	"PRODUCT_TYPE": "Tablet",
	"UNIT_PRICE": 318.12,
	"QUANTITY": 3,
	"TOTAL_PRICE": 954.36,
	"STATE": "Nebraska",
	"STATUS": "Shipped"
	}

b. Kafka system [2]

i. 5 partitions

Kafka ใช้ partitions ในการแบ่งข้อมูลภายในแต่ละ topic เพื่อเพิ่มประสิทธิภาพในการจัดการและกระจายข้อมูลไปยัง brokers หลายตัว ในระบบนี้ topic แต่ละอันถูกแบ่งออกเป็น 5 partitions ซึ่งช่วยให้รองรับการประมวลผลข้อมูลแบบขนาน (parallel processing) ได้ดีขึ้นและเพิ่มประสิทธิภาพทั้งในด้านการเขียนและการอ่านข้อมูล นอกจากนี้ การใช้ partitions ยังช่วยเพิ่มความสามารถในการขยายระบบ (scalability) และรองรับการใช้งานที่มีข้อมูลจำนวนมาก โดยข้อมูลจะถูกกระจายอย่างสมดุลไปยัง brokers ต่างๆ เพื่อให้ระบบทำงานได้อย่างราบรื่น.

ii. 3 brokers

ในการสร้างระบบ Kafka เราจะสร้าง 3 brokers หมายถึงมี 3 servers ที่ทำหน้าที่ในการจัดการและจัดเก็บข้อมูล โดย brokers จะรับข้อมูลจาก producer และส่งข้อมูลไปยัง consumer การใช้หลาย brokers ช่วยเพิ่มความทนทาน (fault tolerance) ของระบบ เนื่องจากหาก broker ตัวใดตัวหนึ่งล้มเหลว ข้อมูลยังคงสามารถเข้าถึงได้จาก brokers อื่นๆ ที่มีข้อมูลสำรอง (replica) อยู่ ทำให้ระบบมีความเสถียรและสามารถดำเนินการต่อไปได้โดยไม่กระทบต่อการทำงานของระบบ ในที่นี้ ระบบใช้ broker:29092, broker1:29095, และ broker2:29098 เพื่อจัดการข้อมูลและรองรับการกระจายข้อมูลระหว่าง servers.

iii. 8 topics

ในการสร้าง 8 topics หมายถึงมีการตั้งค่าไว้ทั้งหมด 8 topic ที่ใช้ในการแยกประเภทข้อมูล โดยแต่ละ topic จะรับข้อมูลจาก producers และส่งข้อมูลไปยัง consumers เพื่อการประมวลผลหรือการใช้งานต่อไป การใช้หลายๆ topic ช่วยให้สามารถจัดการและแยกประเภทข้อมูลที่แตกต่างกันได้อย่างมีประสิทธิภาพ เช่น หนึ่ง topic อาจใช้สำหรับการบันทึกข้อมูลการสั่งซื้อ (orders), อีก topic อาจใช้สำหรับการบันทึกข้อมูลผู้ใช้ (users), ซึ่งทำให้สามารถควบคุมข้อมูลแต่ละประเภทได้อย่างเป็นระเบียบและสามารถเข้าถึงได้ง่ายขึ้นในระบบ

iv. Schema Register

Schema Registry เป็นเครื่องมือที่ช่วยในการจัดการ schema ของข้อมูลที่ถูกส่งไปยัง Kafka topics เช่น Avro, JSON, หรือ Protobuf โดยทำหน้าที่ตรวจสอบและรับรองว่า ข้อมูลที่ถูกส่งไปยัง Kafka มีรูปแบบที่ถูกต้องและสอดคล้องกับ schema ที่กำหนดไว้ ช่วยให้มั่นใจได้ว่าข้อมูลมีความสมบูรณ์และสามารถตรวจสอบได้เมื่อมีการส่งข้อมูลระหว่าง producers และ consumers ผ่าน Kafka โดยการใช้ Schema Registry ยังช่วยในการป้องกันปัญหาการเข้ากันไม่ได้ระหว่างรูปแบบข้อมูลที่ต่างกัน เช่น เมื่อข้อมูลถูกส่งในรูปแบบ Avro หรือ JSON และช่วยให้ง่ายต่อการบำรุงรักษาและการเปลี่ยนแปลง schema ในระบบได้อย่างราบรื่น.

v. Kafka connect

Kafka Connect เป็นเครื่องมือที่ช่วยเชื่อมต่อ Kafka กับแหล่งข้อมูลภายนอก เช่น ฐานข้อมูล, ระบบการจัดเก็บข้อมูล, หรือแอปพลิเคชันอื่นๆ โดยใช้ connectors ที่ช่วยในการดึงข้อมูลจากแหล่งภายนอกเข้าสู่ Kafka หรือส่งข้อมูลจาก Kafka ไปยังระบบภายนอก การใช้ Kafka Connect ช่วยให้การทำงานกับข้อมูลภายนอกเป็นไปอย่างสะดวกและรวดเร็ว โดยไม่จำเป็นต้องเขียนโค้ด เนื่องจาก connectors จะทำการเชื่อมต่อและถ่ายโอนข้อมูลอย่างอัตโนมัติ โดยในกรณีนี้เราใช้ Data Generator เพื่อสร้างข้อมูลจำลองเกี่ยวกับ pageviews และ users ซึ่งช่วยให้สามารถสร้างข้อมูลทดสอบได้ง่ายและรวดเร็วโดยไม่ต้องใช้ข้อมูลจริง.

c. ksqlDB operation [3]

i. Clean or transform data (topic4)

เราได้ทำการ transforming ข้อมูลจากสองแหล่งข้อมูล (sources) คือ PAGEVIEWS_STREAM และ users_tb เราจะทำการแปลงข้อมูลที่มีอยู่ให้มีรูปแบบที่เข้าใจได้ง่ายและถูกต้องมากขึ้น:

1. PAGEVIEWS_STREAM: ข้อมูลที่เกี่ยวข้องกับ viewtime หรือเวลาในการดูหน้าเว็บจะถูกแปลงจากหน่วยมิลลิวินาทีเป็นหน่วยนาที (โดยการหารด้วย 6000) เพื่อให้สามารถเข้าใจได้ง่ายขึ้น เช่น การแปลงข้อมูล VIEWTIME = 691001 มิลลิวินาที จะกลายเป็น VIEWTIME_FORMATTED = 22 นาที ข้อมูลที่ถูกแปลงนี้จะถูกส่งไปยัง Kafka topic 4_cleaned โดยใช้คำสั่ง CREATE STREAM.
2. users_tb: ข้อมูลที่เกี่ยวข้องกับ REGISTERTIME หรือเวลาลงทะเบียนผู้ใช้งานจะถูกแปลงจากมิลลิวินาทีเป็นรูปแบบเวลาที่เข้าใจได้ง่าย (เช่น yyyy-MM-dd HH:mm

) ด้วยการใช้ฟังก์ชัน TIMESTAMPTOSTRING ซึ่งจะทำให้เวลาที่มีรูปแบบเช่น 1970-01-18 17:30:02 สามารถอ่านและตีความได้ง่ายขึ้น ข้อมูลนี้จะถูกส่งไปยัง Kafka topic 4_cleaned ด้วยคำสั่ง CREATE TABLE.

การแปลงข้อมูลเหล่านี้ช่วยให้ข้อมูลมีความถูกต้องและเข้าใจง่ายขึ้น ทำให้การประมวลผลข้อมูลในภายหลัง เช่น การวิเคราะห์หรือการใช้งานในระบบอื่น ๆ เป็นไปได้อย่างมีประสิทธิภาพและมีความแม่นยำมากขึ้น.

CREATE STREAM PAGEVIEWS_CLEAN WITH (KAFKA_TOPIC = '4_cleaned', VALUE_FORMAT = 'JSON') AS
SELECT
USERID,
PAGEID,
VIEWTIME/6000 AS VIEWTIME_FORMATTED
FROM PAGEVIEWS_STREAM;
	{
	"USERID": "User_2",
	"PAGEID": "Page_97",
	"VIEWTIME_FORMATTED": 22
	}

CREATE TABLE USERS_CLEAN WITH (KAFKA_TOPIC = '4_cleaned', VALUE_FORMAT = 'JSON') AS
SELECT
USERID,
REGIONID,
GENDER,
TIMESTAMPTOSTRING(REGISTERTIME / 1000, 'yyyy-MM-dd HH:mm:ss', 'Asia/Bangkok') AS REGISTERTIME_FORMATTED
FROM users_tb;
	{
	"USERID": "User_6",
	"REGIONID": "Region_1",
	"GENDER": "OTHER",
	"REGISTERTIME_FORMATTED": "1970-01-18 17:30:02"
	}

ii. Aggregation (join + group by) (topic5)

การ aggregation นี้ เราจะทำการรวบรวมข้อมูลคำสั่งซื้อจาก ORDERS_STREAM และข้อมูลผู้ใช้จาก USERS_CLEAN โดยการใช้ JOIN ข้อมูลทั้งสองแหล่ง และทำการ GROUP BY ตามประเภทสินค้าที่สั่งซื้อ (PRODUCT_TYPE). การ JOIN จะทำให้เราสามารถเชื่อมโยงข้อมูลคำสั่งซื้อกับข้อมูลของผู้ใช้งานผ่าน USERID เพื่อให้ได้ข้อมูลที่ครบถ้วนมากขึ้น.

หลังจากนั้น เราจะคำนวณข้อมูลต่าง ๆ ที่เกี่ยวข้อง เช่น:

- total_orders: จำนวนคำสั่งซื้อทั้งหมดในแต่ละประเภทสินค้า
- total_sales: ยอดขายรวมจากคำสั่งซื้อทั้งหมด
- avg_sales_per_order: ยอดขายเฉลี่ยต่อคำสั่งซื้อ

ข้อมูลที่ได้จากการคำนวณนี้จะถูกส่งไปยัง Kafka topic 5_aggregated ซึ่งจะช่วยให้ข้อมูลที่รวบรวมและประมวลผลแล้วสามารถนำไปใช้งานหรือวิเคราะห์ได้ต่อไป เช่น การสร้างรายงาน หรือการใช้ข้อมูลในแอปพลิเคชันที่เกี่ยวข้อง

CREATE TABLE 5_sales_by_product
WITH (KAFKA_TOPIC = '5_aggregated') AS
SELECT
o.PRODUCT_TYPE,
COUNT(o.ORDERID) AS total_orders,
SUM(o.TOTAL_PRICE) AS total_sales,
SUM(o.TOTAL_PRICE) / COUNT(o.ORDERID) AS avg_sales_per_order
FROM ORDERS_STREAM o
LEFT JOIN USERS_CLEAN u
ON o.USERID = u.USERID
GROUP BY o.PRODUCT_TYPE;
	{
	"PRODUCT_TYPE": "Keyboard",
	"TOTAL_ORDERS": 187,
	"TOTAL_SALES": 49598.950000000026,
	"AVG_SALES_PER_ORDER": 265.23502673796804
	}

iii. Windows

1. Tumbling (topic6)

การ Tumbling Window นี้ เราจะใช้ฟังก์ชัน TUMBLING (SIZE 1 HOUR) เพื่อคำนวณจำนวนการดูหน้าเว็บ (pageviews) ของผู้ใช้แต่ละคนในช่วงเวลาที่เป็น 1 ชั่วโมง ซึ่งจะไม่ทับซ้อนกัน โดยการใช้ Tumbling Window จะช่วยแบ่งข้อมูลออกเป็นช่วงเวลาที่ชัดเจน เช่น ช่วงเวลา 1 ชั่วโมง ซึ่งข้อมูลแต่ละช่วงจะถูกคำนวณแยกจากกันและไม่ทับซ้อนกัน

การคำนวณจะทำการ COUNT จำนวนการดูหน้าเว็บ (pageviews) ของผู้ใช้ (USERID) ในแต่ละช่วงเวลา (window) และผลลัพธ์ที่ได้จะประกอบด้วย:

- USERID: รหัสผู้ใช้ที่ทำการดูหน้าเว็บ
- PAGEVIEWS_COUNT: จำนวนครั้งที่ผู้ใช้งานดูหน้าเว็บในช่วงเวลา 1 ชั่วโมง
- WINDOWSTART และ WINDOWEND: เวลาเริ่มต้นและสิ้นสุดของช่วงเวลา 1 ชั่วโมงที่ใช้ในการคำนวณ

ข้อมูลที่ได้จะถูกส่งไปยัง Kafka topic 6_tumbling ซึ่งสามารถนำไปใช้งานต่อในระบบการประมวลผลข้อมูลแบบเรียลไทม์ หรือใช้ในการวิเคราะห์พฤติกรรมของผู้ใช้ในช่วงเวลาต่าง ๆ.

CREATE TABLE 6_pageviews_per_user WITH
(KAFKA_TOPIC='6_tumbling') AS
SELECT
pv.USERID,
COUNT(*) AS PAGEVIEWS_COUNT
FROM PAGEVIEWS_CLEAN pv
WINDOW TUMBLING (SIZE 1 HOUR) -- 1-hour window
GROUP BY pv.USERID EMIT CHANGES;
	{
	"USERID": "User_7",
	"WINDOWSTART": 1731740400000,
	"WINDOWEND": 1731744000000,
	"PAGEVIEWS_COUNT": 153
	}

2. Hopping (topic7)

การ Hopping Window นี้ เราจะใช้ HOPPING (SIZE 1 HOUR, ADVANCE BY 30 MINUTES) เพื่อคำนวณจำนวนคำสั่งซื้อและยอดขายรวมของผู้ใช้แต่ละคนในช่วงเวลา 1 ชั่วโมง โดยการเลื่อนช่วงเวลา (hop) ทุก ๆ 30 นาที. การใช้ Hopping Window ช่วยให้สามารถคำนวณข้อมูลได้ในหลาย ๆ ช่วงเวลา โดยที่ทุก ๆ 30 นาทีจะมีการเลื่อนช่วงเวลาใหม่เข้ามา ซึ่งทำให้สามารถคำนวณข้อมูลใหม่ในแต่ละช่วงเวลาได้อย่างต่อเนื่องและซ้ำซ้อนกัน.

ผลลัพธ์ที่ได้จากการคำนวณจะประกอบด้วย:

- USERID: รหัสผู้ใช้ที่ทำการสั่งซื้อ
- ORDER_COUNT: จำนวนคำสั่งซื้อของผู้ใช้ในช่วงเวลานั้น
- TOTAL_SALES: ยอดขายรวมจากคำสั่งซื้อในช่วงเวลานั้น
- WINDOWSTART และ WINDOWEND: เวลาเริ่มต้นและสิ้นสุดของช่วงเวลา (window) ที่ใช้ในการคำนวณ

ข้อมูลที่ได้จะถูกส่งไปยัง Kafka topic 7_hopping ซึ่งสามารถใช้ในการวิเคราะห์และติดตามพฤติกรรมการสั่งซื้อของผู้ใช้ในช่วงเวลาต่าง ๆ เช่น การวิเคราะห์ยอดขายในช่วงเวลาที่มีการเปลี่ยนแปลงอย่างรวดเร็ว.

```
CREATE TABLE 7_total_orders_hopping
  WITH (KAFKA_TOPIC='7_hopping') AS
    SELECT
        o.USERID,
        COUNT(*) AS ORDER_COUNT,
        SUM(o.TOTAL_PRICE) AS TOTAL_SALES
    FROM orders_stream o
    WINDOW HOPPING (SIZE 1 HOUR, ADVANCE BY 30 MINUTES)
-- 1-hour window, hops every 30 minutes
    GROUP BY o.USERID EMIT CHANGES;
```

```
{
  "USERID": "User_7",
  "WINDOWSTART": 1731740400000,
  "WINDOWEND": 1731744000000,
  "PAGEVIEWS_COUNT": 153
}
```

3. Session (topic8)

การ Session Window นี้ เราจะใช้ Session Window เพื่อติดตามและคำนวณจำนวนคำสั่งซื้อและยอดรวมคำสั่งซื้อของผู้ใช้แต่ละคนในช่วงเวลาที่กำหนด 10 นาที ซึ่งระบบจะพิจารณาว่าเซสชันสิ้นสุดเมื่อไม่มีการกระทำจากผู้ใช้ภายในช่วงเวลา 10 นาที. เมื่อไม่มีคำสั่งซื้อใหม่เกิดขึ้นในช่วงเวลานั้น เซสชันจะถูกปิดและเริ่มเซสชันใหม่ทันที.

ผลลัพธ์ที่ได้จะประกอบด้วย:

- ORDER_COUNT: จำนวนคำสั่งซื้อที่ผู้ใช้ทำในเซสชันนั้น
- TOTAL_ORDER_VALUE: ยอดรวมของคำสั่งซื้อที่เกิดขึ้นในเซสชัน
- WINDOWSTART และ WINDOWEND: เวลาที่เริ่มต้นและสิ้นสุดของเซสชันนั้น

ข้อมูลที่ได้จะถูกส่งไปยัง Kafka topic 8_session, ซึ่งสามารถนำไปใช้ในการวิเคราะห์พฤติกรรมการสั่งซื้อของผู้ใช้ที่มีการกระทำหรือกิจกรรมอย่างต่อเนื่องในช่วงเวลาที่กำหนด และช่วยในการติดตามความสัมพันธ์ระหว่างผู้ใช้และการสั่งซื้อในช่วงเวลานั้น ๆ.

CREATE TABLE 8_orders_session_table
WITH (KAFKA_TOPIC='8_session') AS
SELECT
USERID,
COUNT(*) AS ORDER_COUNT,
SUM(TOTAL_PRICE) AS TOTAL_ORDER_VALUE
FROM orders_stream
WINDOW SESSION (10 MINUTES)
GROUP BY USERID
EMIT CHANGES;
	{
	"USERID": "User_5",
	"WINDOWSTART": 1731745993095,
	"WINDOWEND": 1731747620802,
	"ORDER_COUNT": 180,
	"TOTAL_ORDER_VALUE": 278670.22
	}

iv. Testing the correctness [4]

สำหรับการทำ testing correctness ในระบบที่ใช้ ksqlDB, การทดสอบจะมุ่งเน้นที่การตรวจสอบความถูกต้องของการประมวลผลข้อมูลและการทำงานของ streams และ tables ที่ถูกสร้างขึ้นใน ksqlDB เพื่อให้แน่ใจว่าข้อมูลที่ได้จากการคิวรีเป็นไปตามที่คาดหวังและสามารถทำงานได้ตามความต้องการ. โดยในที่นี้เราจะทำการสร้าง input.json, output.json และ statement.sql ในการทดสอบ

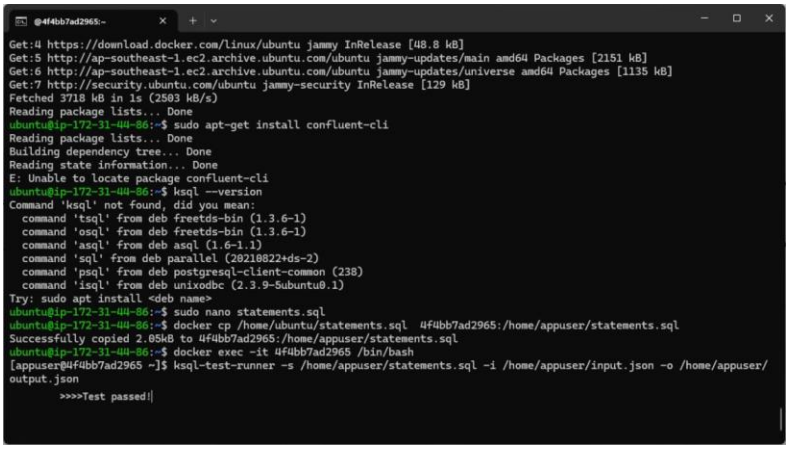
การทดสอบความถูกต้องใน ksqlDB มุ่งเน้นที่การตรวจสอบการประมวลผลข้อมูลจาก Kafka stream ว่าถูกต้องหรือไม่ โดยใช้ input.json, output.json, และ statement.sql ดังนี้:

1. input.json: ข้อมูลตัวอย่างที่ส่งไปยัง Kafka topic เช่น USER_ID, PAGE_URL, และ TIMESTAMP.
2. statement.sql: คำสั่ง SQL สำหรับสร้าง stream และคิวรีข้อมูลจาก Kafka topic.
3. output.json: ผลลัพธ์ที่คาดหวังจากการคิวรีข้อมูลใน ksqlDB.

โดยข้อมูลใน input.json จะถูกส่งไป Kafka, จากนั้นจะใช้คำสั่งใน statement.sql เพื่อดึงข้อมูลและเปรียบเทียบกับ output.json เพื่อทดสอบความถูกต้องในการประมวลผลข้อมูล.

```
CREATE STREAM pageviews (
  USER_ID STRING,
  PAGE_URL STRING,
  TIMESTAMP TIMESTAMP
) WITH (
  KAFKA_TOPIC='datagen-pageviews',
  VALUE_FORMAT='JSON'
);

SELECT USER_ID, PAGE_URL, TIMESTAMP
FROM pageviews
LIMIT 10;  -- LIMIT to check a small
batch of records
```



d. Apache Pinot [5,6]

Apache Pinot เป็นระบบจัดเก็บข้อมูลแบบคิวรีเรียลไทม์ที่ออกแบบมาเพื่อให้สามารถประมวลผลข้อมูลจากแหล่งข้อมูลภายนอก เช่น Kafka และทำการคิวรีข้อมูลแบบเรียลไทม์ได้อย่างมีประสิทธิภาพ เราได้ใช้ Apache Pinot ในการกำหนด schema และสร้าง real-time tables เพื่อจัดการกับข้อมูลที่ถูกนำเข้ามาจาก Kafka ซึ่งจะช่วยให้เราสามารถประมวลผลข้อมูลในเวลาจริงและทำการคิวรีข้อมูลได้ทันที เราได้สร้าง 3 ตารางใน Apache Pinot ซึ่งมาจาก 3 แหล่งข้อมูลหลัก ได้แก่ pageviews, users, และ orders การตั้งค่าระบบนี้ช่วยให้เราสามารถดึงข้อมูลจาก Kafka มาประมวลผลในระบบของ Apache Pinot และใช้ข้อมูลเหล่านั้นเพื่อการวิเคราะห์ที่รวดเร็วและมีประสิทธิภาพในเวลาจริง.

[i]. At least 3 queries

1. คิวรีข้อมูลเฉลี่ยราคาต่อรัฐ : ในการคิวรีข้อมูลเฉลี่ยราคาต่อรัฐ เราจะคำนวณค่าเฉลี่ยของราคาสินค้า (AVG(TOTAL_PRICE)) แยกตามรัฐ (STATE) จากข้อมูลที่จัดเก็บในตาราง 3_orders ซึ่งบันทึกการสั่งซื้อทั้งหมด โดยจะใช้การจัดกลุ่มข้อมูลตามรัฐ (GROUP BY STATE) เพื่อคำนวณค่าเฉลี่ยราคาสินค้าของแต่ละรัฐ. ผลลัพธ์ที่ได้จะประกอบด้วย:

- STATE: รหัสหรือชื่อของรัฐที่มีการสั่งซื้อ
- avg_price_per_state: ค่าเฉลี่ยของราคาสินค้าต่อรัฐค่าเฉลี่ยของราคาสินค้าต่อรัฐ

ข้อมูลที่ได้จะถูกใช้ในการวิเคราะห์และตัดสินใจทางธุรกิจ เช่น การกำหนดราคาและกลยุทธ์การตลาดที่เหมาะสมตามลักษณะของแต่ละรัฐ โดยผลลัพธ์จะถูกส่งไปยัง dashboard เพื่อแสดงผลต่อไป

```
SELECT
    STATE,
    AVG(TOTAL_PRICE) AS
avg_price_per_state
FROM 3_orders
GROUP BY STATE;
```

QUERY RESULT

Search...

STATE	avg_price_per_state
Missouri	1529.3299882766712
Indiana	1545.0907540173052
Illinois	1651.6371727748694
Michigan	1569.0954289215688
Minnesota	1626.2678299776283
Kansas	1615.9134561626431
Ohio	1497.04423030303
Iowa	1617.5715301204816

2. คิวรีจำนวนการดูหน้าเว็บต่อผู้ใช้และหน้าเว็บ : คำนวณจำนวนการดูหน้าเว็บ (PAGEVIEWS_COUNT) สำหรับแต่ละผู้ใช้ (USERID) และแต่ละหน้าเว็บ (PAGEID) จากข้อมูลในตาราง 1_pageviews ซึ่งบันทึกข้อมูลการดูหน้าเว็บทั้งหมด โดยใช้การจัดกลุ่มข้อมูลตาม USERID และ PAGEID. ผลลัพธ์ที่ได้จะประกอบด้วย:

- USERID: รหัสผู้ใช้ที่ดูหน้าเว็บ
- PAGEID: รหัสหรือชื่อของหน้าเว็บที่ถูกดู
- PAGEVIEWS_COUNT: จำนวนการดูหน้าเว็บของผู้ใช้ในแต่ละหน้าเว็บ

ข้อมูลที่ได้จากคิวรีนี้จะถูกนำไปใช้ในการวิเคราะห์พฤติกรรมการเข้าชมของผู้ใช้ โดยสามารถดูได้ว่าผู้ใช้แต่ละคนดูหน้าเว็บใดบ่อยที่สุด หรือหน้าเว็บไหนที่ได้รับความนิยมมากที่สุด ซึ่งข้อมูลนี้สามารถช่วยในการปรับปรุงประสบการณ์ผู้ใช้บนเว็บไซต์และสนับสนุนการตัดสินใจทางการตลาด.

```
SELECT
    USERID,
    PAGEID,
    COUNT(*) AS PAGEVIEWS_COUNT
FROM 1_pageviews
GROUP BY USERID, PAGEID;
```

QUERY RESULT

Search...

userid	pageid	PAGEVIEWS_COUNT
User_8	Page_70	318
User_7	Page_70	329
User_9	Page_70	354
User_2	Page_70	322
User_1	Page_70	327
User_4	Page_70	332
User_3	Page_70	332
User_9	Page_54	314

3. คำนวณยอดขายรวม (SUM(TOTAL_PRICE)) ตามประเภทของสินค้า (PRODUCT_TYPE) จาก table 3_orders ซึ่งบันทึกข้อมูลการสั่งซื้อ โดยจะแสดงผลลัพธ์ยอดขายรวมของแต่ละประเภทสินค้า ผลลัพธ์ที่ได้:
   - PRODUCT_TYPE: ประเภทของสินค้าที่ขาย
   - TOTAL_SALES: ยอดขายรวมของแต่ละประเภทสินค้า

ข้อมูลที่ได้จากคิวรีนี้จะช่วยในการวิเคราะห์ยอดขายตามประเภทสินค้า เช่น เพื่อตรวจสอบสินค้าที่มียอดขายสูงที่สุด หรือสินค้าที่มีแนวโน้มในการขายดีในช่วงเวลาหนึ่ง ๆ ซึ่งสามารถนำไปใช้ในการวางแผนการจัดสรรสินค้าและการตัดสินใจทางการตลาด.

```
SELECT
    PRODUCT_TYPE,
    SUM(TOTAL_PRICE) AS total_sales
FROM 3_orders
GROUP BY PRODUCT_TYPE;
```

QUERY RESULT

Search...

userid	pageid	PAGEVIEWS_COUNT
User_8	Page_70	318
User_7	Page_70	329
User_9	Page_70	354
User_2	Page_70	322
User_1	Page_70	327
User_4	Page_70	332
User_3	Page_70	332
User_9	Page_54	314

e. Dashboard [7] i. At least 4 panels ii. Bonus: Interactive dashboard

	USERID	ORDER_COUNT	TOTAL_PRICE	AVG_PRICE
0	User_1	910	1,547,068.08	1,700.0748
1	User_2	882	1,269,021.28	1,438.7996
2	User_3	920	1,453,114.67	1,579.4725
3	User_4	903	1,394,896.33	1,544.7357
4	User_5	971	1,549,295.26	1,595.5667
5	User_6	952	1,513,995.21	1,590.3311
6	User_7	901	1,446,263.34	1,605.1757
7	User_8	858	1,378,756.84	1,606.9427
8	User_9	906	1,413,669.83	1,560.342

ในส่วนของการแสดงผลข้อมูลบน Streamlit: BoBoShop, เราได้พัฒนา dashboard เพื่อให้สามารถดูข้อมูลต่างๆ ที่มีการอัปเดตในแบบเรียลไทม์ โดยการเชื่อมต่อกับ Apache Pinot ผ่านพอร์ต 8099 เพื่อดึงข้อมูลจากฐานข้อมูลและแสดงผลได้ดังนี้:

1. ค่าเฉลี่ยราคาต่อประเภทสินค้า (Product Type)
   ใช้ st.metric() เพื่อแสดงค่าเฉลี่ยราคาของสินค้าแต่ละประเภท โดยคำนวณจากคำสั่ง SQL ที่ดึงข้อมูลจากตาราง 3_orders. ผลลัพธ์จะถูกแสดงในรูปแบบของ metric สำหรับแต่ละประเภทสินค้า พร้อมทั้งแสดงความแตกต่างระหว่างข้อมูลล่าสุดกับข้อมูลก่อนหน้า เพื่อให้ผู้ใช้สามารถติดตามการเปลี่ยนแปลงของราคาสินค้าได้อย่างชัดเจนและทันที.

2. พฤติกรรมการสั่งซื้อของลูกค้า (Behavior User Data Frame)
   ใช้ st.dataframe() เพื่อแสดงข้อมูลพฤติกรรมการสั่งซื้อของลูกค้า ซึ่งจะแสดงข้อมูลเช่น จำนวนการสั่งซื้อ, ยอดรวม, และ ค่าเฉลี่ยของการสั่งซื้อ. ฟังก์ชันนี้ช่วยให้เราสามารถดูข้อมูลเชิงลึกในรูปแบบที่มีระเบียบและเข้าใจง่าย โดยการแสดงผลในรูปแบบ dataframe ที่อัปเดตแบบเรียลไทม์.

3. ยอดขายตามประเภทสินค้า
   ใช้ plotly.express.pie เพื่อแสดง ยอดขายทั้งหมด แยกตามประเภทสินค้าในรูปแบบ Pie Chart. กราฟนี้จะช่วยให้ผู้ใช้เห็นภาพรวมของยอดขายในแต่ละประเภทสินค้าได้อย่างชัดเจนและเข้าใจง่าย โดยการใช้แผนภาพที่เข้าใจได้ง่ายและมีสีสันที่โดดเด่นเพื่อแสดงสัดส่วนยอดขายแต่ละประเภท.

4. จำนวนการดูหน้าเว็บตามผู้ใช้
   ข้อมูลจำนวนการดูหน้าเว็บ (view_count) จะถูกแสดงตาม PAGEID ของแต่ละหน้าเว็บที่ผู้ใช้เข้าชม โดยสามารถกรองข้อมูลตามหน้าเว็บที่เลือกได้จาก multiselect. ฟังก์ชันนี้ช่วยให้ผู้ใช้สามารถเลือกหน้าเว็บที่ต้องการดูข้อมูลและแสดงกราฟที่อัปเดตตามการเลือก, โดยการติดตามข้อมูลการเข้าชมแต่ละหน้าเว็บจะทำได้ง่ายและสะดวก.

5. ยอดการซื้อเฉลี่ยตามรัฐ
   ใช้ plotly.express.choropleth เพื่อแสดง ยอดการซื้อเฉลี่ย (avg_purchase_amount) ตามรัฐต่างๆ ของสหรัฐอเมริกาในรูปแบบ Choropleth map. แผนที่จะแบ่งสีตามค่าของยอดการซื้อที่แตกต่างกันในแต่ละรัฐ โดยการใช้แผนที่แบบสีที่มีการเน้นให้เห็นความแตกต่างของยอดการซื้อระหว่างรัฐต่างๆ ช่วยให้ผู้ใช้เห็นข้อมูลการซื้อที่แตกต่างกันในแต่ละพื้นที่ได้อย่างชัดเจน.

ฟีเจอร์เสริม:

- Auto-refresh: หน้า dashboard มีการแสดงผลแบบอัปเดตอัตโนมัติทุก 5 วินาทีผ่าน checkbox เพื่อให้ข้อมูลที่แสดงผลมีความทันสมัยและถูกต้องตลอดเวลา.
- การแสดงภาพสินค้า: ด้านท้ายของ dashboard ยังมีการแสดง รูปภาพสินค้า เพื่อเสริมการนำเสนอข้อมูลและทำให้การนำเสนอข้อมูลมีความหลากหลายและน่าสนใจมากยิ่งขึ้น.

การใช้งาน Streamlit ร่วมกับ Apache Pinot ทำให้สามารถแสดงข้อมูลจาก Kafka แบบเรียลไทม์ได้อย่างมีประสิทธิภาพ โดยสามารถติดตามการเปลี่ยนแปลงและแสดงผลข้อมูลเชิงลึกได้สะดวกผ่าน dashboard ที่ใช้งานง่าย.