Real Time Orders Payments Data Ingestion

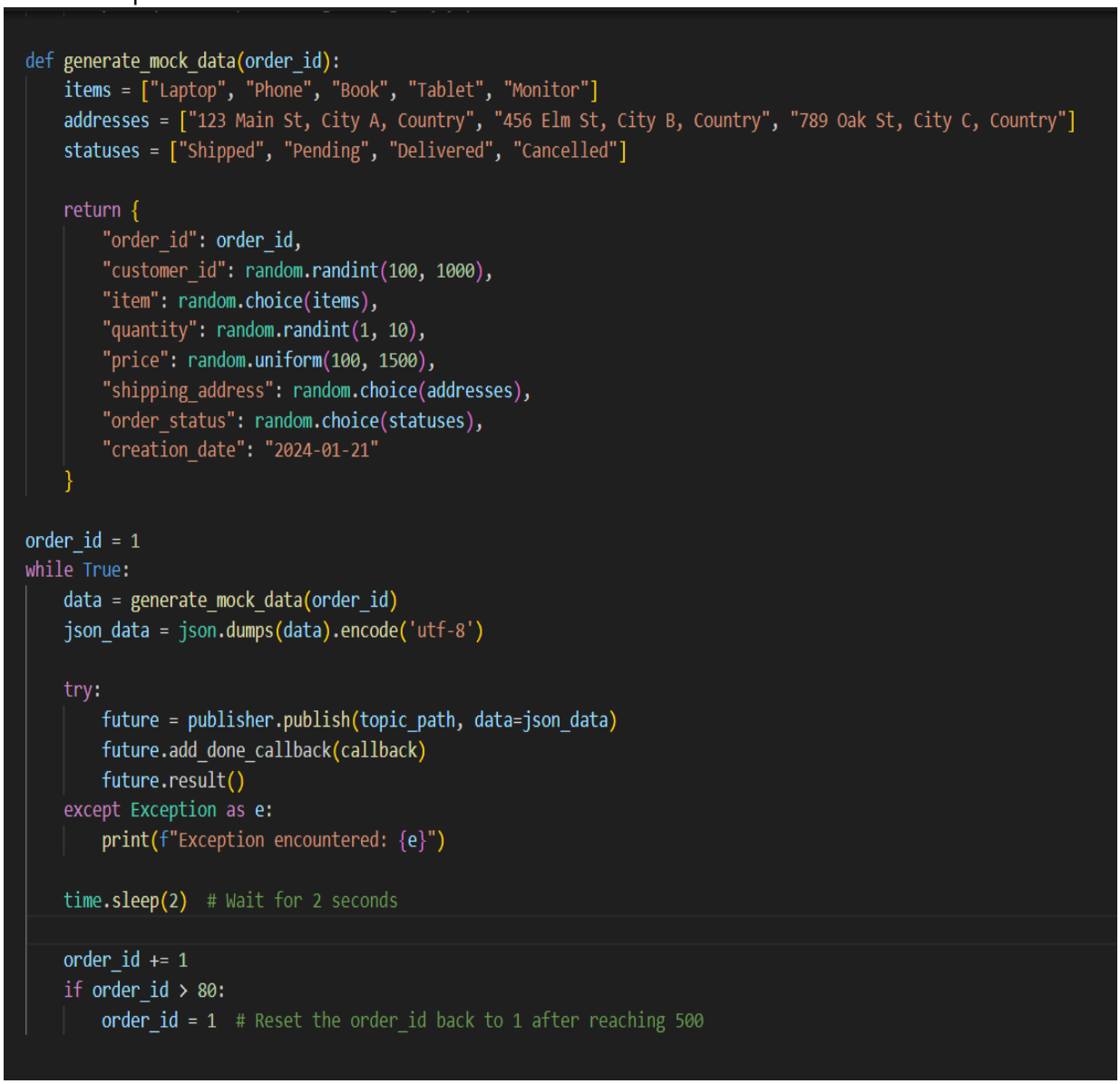
1. This project was about the ingestion of orders payments data in real time.

2. We needed 3 pub sub topics for this project. Topics are orders_data, payments_data and dlq_payments_data.

3. While creating topics, a default subscription gets created for the respective topics.

4. first we run producer of orders data so that some records might be able to ingest into cassandra table. Here attributes related to payment will be null when we check cassandra table.

   orders data producer:



```python
def generate_mock_data(order_id):
    items = ["Laptop", "Phone", "Book", "Tablet", "Monitor"]
    addresses = ["123 Main St, City A, Country", "456 Elm St, City B, Country", "789 Oak St, City C, Country"]
    statuses = ["Shipped", "Pending", "Delivered", "Cancelled"]

    return {
        "order_id": order_id,
        "customer_id": random.randint(100, 1000),
        "item": random.choice(items),
        "quantity": random.randint(1, 10),
        "price": random.uniform(100, 1500),
        "shipping_address": random.choice(addresses),
        "order_status": random.choice(statuses),
        "creation_date": "2024-01-21"
    }

order_id = 1
while True:
    data = generate_mock_data(order_id)
    json_data = json.dumps(data).encode('utf-8')

    try:
        future = publisher.publish(topic_path, data=json_data)
        future.add_done_callback(callback)
        future.result()
    except Exception as e:
        print(f"Exception encountered: {e}")

    time.sleep(2)  # Wait for 2 seconds

    order_id += 1
    if order_id > 80:
        order_id = 1  # Reset the order_id back to 1 after reaching 500
```

orders data consumer:

```
cluster,session = cassandra_connection()

# Prepare the Cassandra insertion statement
insert_stmt = session.prepare("""
    INSERT INTO orders_payments_facts (order_id, customer_id, item, quantity, price, shipping_address,
    order_status, creation_date, payment_id, payment_method, card_last_four, payment_status, payment_datetime)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
""")

# Pull and process messages
def pull_messages():
    while True:
        response = subscriber.pull(request={"subscription": subscription_path, "max_messages": 10})
        ack_ids = []

        for received_message in response.received_messages:
            # Extract JSON data
            json_data = received_message.message.data.decode('utf-8')

            # Deserialize the JSON data
            deserialized_data = json.loads(json_data)

            print(deserialized_data)

            # Prepare data for Cassandra insertion
            cassandra_data = (
                deserialized_data.get("order_id"),
                deserialized_data.get("customer_id"),
                deserialized_data.get("item"),
                deserialized_data.get("quantity"),
                deserialized_data.get("price"),
                deserialized_data.get("shipping_address"),
                deserialized_data.get("order_status"),
                deserialized_data.get("creation_date"),
                None,
                None,
                None,
                None,
                None
            )

            # Insert data into Cassandra
            session.execute(insert_stmt, cassandra_data)

            print("Data inserted in cassandra !!")

            # Collect ack ID for acknowledgment
            ack_ids.append(received_message.ack_id)

        # Acknowledge the messages so they won't be sent again
        if ack_ids:
            subscriber.acknowledge(request={"subscription": subscription_path, "ack_ids": ack_ids})
```

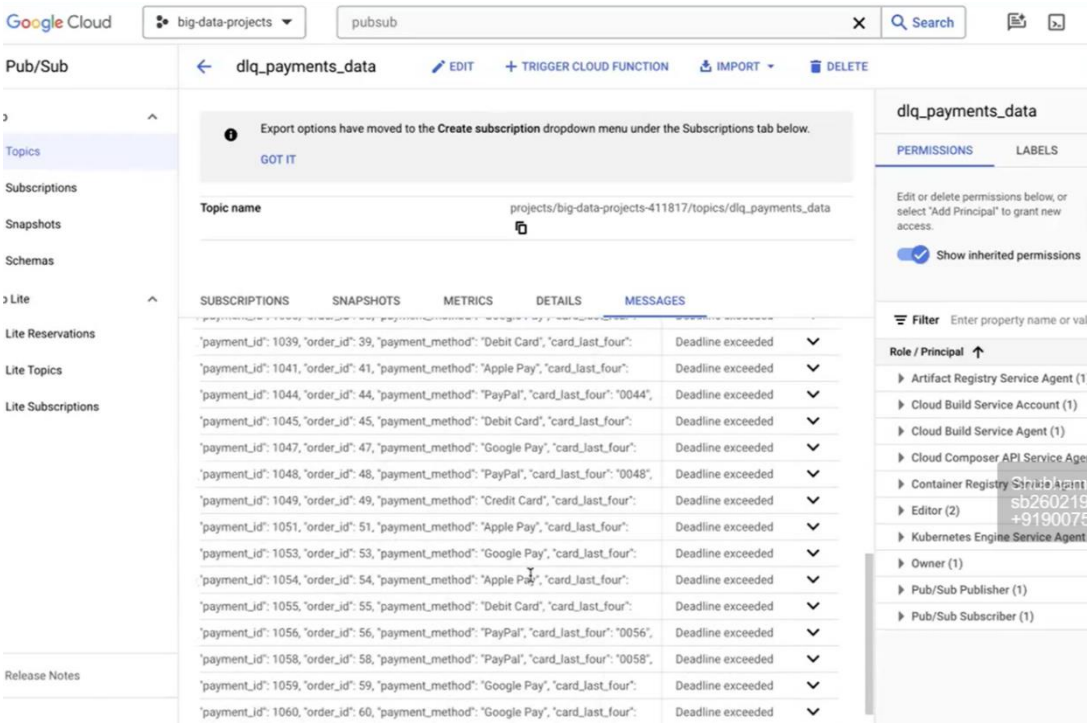
5. Then we start the payments producer so that it can push the data related to the respective order ids.

```
# Initialize the Pub/Sub publisher client
publisher = pubsub_v1.PublisherClient()

# Project and Topic details
project_id = "big-data-projects-411817"
topic_name = "payments_data"
topic_path = publisher.topic_path(project_id, topic_name)

# Payment methods mock data
payment_methods = ["Credit Card", "Debit Card", "PayPal", "Google Pay", "Apple Pay"]

def generate_mock_payment(order_id):
    return {
        # Starting from 1001 as in the example
        "payment_id": order_id + 1000,
        "order_id": order_id,
        "payment_method": choice(payment_methods),
        # Just using order_id to make last 4 digits
        "card_last_four": str(order_id).zfill(4)[-4:],
        "payment_status": "Completed",
        # Using order_id to vary the hour for variety
        "payment_datetime": f"2024-01-21T{str(order_id).zfill(2)}:01:30Z"
    }

def callback(future):
    try:
        # Get the message_id after publishing.
        message_id = future.result()
        print(f"Published message with ID: {message_id}")
    except Exception as e:
        print(f"Error publishing message: {e}")

# Produce mock payment data for each order_id and publish to the topic
for order_id in range(1, 501):  # 500 order_ids
    mock_payment = generate_mock_payment(order_id)
    json_data = json.dumps(mock_payment).encode('utf-8')
    try:
        future = publisher.publish(topic_path, data=json_data)
        future.add_done_callback(callback)
        time.sleep(1)  # Sleep for 2 seconds before producing the next message
    except Exception as e:
        print(f"Exception encountered: {e}")
```

6. Next executed ingestion in fact table using payments consumer which starts reading data from payments topic so that it can check order id existence in cassandra table and update the payment data for that order id there at table in real time.

```python
# Setup Cassandra connection

cluster,session = cassandra_connection()

# Pull and process messages
def pull_messages():
    while True:
        response = subscriber.pull(request={"subscription": subscription_path, "max_messages": 10})
        ack_ids = []

        for received_message in response.received_messages:
            # Extract JSON data
            json_data = received_message.message.data.decode('utf-8')

            # Deserialize the JSON data
            deserialized_data = json.loads(json_data)

            # Check if order_id exists in the Cassandra table
            query = f"SELECT order_id FROM orders_payments_facts WHERE order_id = {deserialized_data.get('order_id')}"
            rows = session.execute(query)
            if rows.one():  # if order_id found
                # Assuming you have a `payments` column, you can update it like this:
                update_query = """
                    UPDATE orders_payments_facts
                    SET payment_id = %s,
                        payment_method = %s,
                        card_last_four = %s,
                        payment_status = %s,
                        payment_datetime = %s
                    WHERE order_id = %s
                """
                values = (
                    deserialized_data.get('payment_id'),
                    deserialized_data.get('payment_method'),
                    deserialized_data.get('card_last_four'),
                    deserialized_data.get('payment_status'),
                    deserialized_data.get('payment_datetime'),
                    deserialized_data.get('order_id')
                )
                session.execute(update_query, values)
                print("data updated in Cassandra -> ", deserialized_data)
            else:  # if order_id not found
                publisher.publish(dlq_topic_path, data=json_data.encode('utf-8'))
                print("Data thrown in dlq because order_id not found -> ", deserialized_data)

            # Collect ack ID for acknowledgment
            ack_ids.append(received_message.ack_id)

        # Acknowledge the messages so they won't be sent again
        if ack_ids:
            subscriber.acknowledge(request={"subscription": subscription_path, "ack_ids": ack_ids})
```

7. If order id not found, those data thrown to dlq payments topic.



8. We wrote another streaming process on dlq topic to reprocess dlq data in cassandra. That's how we can handle failure data which was somehow didn't ingest into table.