DATA WAREHOUSE AND BUSINESS ANALYTICS

TOPIC: Datawarehouse and Business Analytics for Electronics Store Sales in Tiki Platform

Contents

TABLE OF FIGURES

I. INTRODUCTION

1.1. Problem Overview

In an increasingly competitive e-retail market, the application of technology and data analytics to support business decisions has become a vital factor in optimizing operations, enhancing customer experience and increasing business efficiency. E-stores are currently facing a series of major challenges. First of all, stores have difficulty analyzing sales performance, including identifying high- and low-performing products, understanding seasonal sales trends and assessing the impact of promotions on sales. These limitations not only reduce the ability to increase profits but also hinder the implementation of appropriate business strategies for each stage.

In addition, inventory management is also a pressing issue. Not only does understocking or overstocking waste resources, it also reduces the customer's experience when their needs are not met in a timely manner. This is especially serious during peak seasons or promotional periods when demand spikes suddenly. In addition, ineffective forecasting of product demand leads to difficulties in optimal inventory planning, resulting in wasted resources and costs.

Furthermore, understanding current customer behavior and needs is not fully exploited. Stores have difficulty analyzing customer data to create specific segments, adjust marketing campaigns, and increase customer retention. Customer feedback, an important source of data for improving product and service quality, is not optimally used. This reduces the ability to quickly respond to changing market needs and affects customer loyalty.

Not only that, the store is also facing high operating costs, especially storage and transportation costs, which require optimization of logistics and operational processes. In addition, the lack of an integrated automation system to collect, process and analyze data has caused many limitations in making data-driven decisions. This also leads to difficulties in measuring the effectiveness of marketing campaigns and advertising investments.

These challenges highlight the urgent need to build a modern and automated data infrastructure where data from various sources can be integrated, processed and organized in a logical manner. This system not only helps to analyze data effectively but also provides reports and visual charts to support business decision making. Implementing such a solution will not only help stores improve their competitiveness in the market but also improve customer satisfaction, increase revenue and ensure sustainable development in the future.

1.2. Business Objective

a. Objectives

Primary Objective:

Develop an automated data pipeline for an electronics store to enable efficient data processing and empower data-driven decision-making.

Secondary Objectives:

- Centralize data collection from multiple sources, such as websites and APIs, into a unified storage system.
- Transform raw data into well-structured Dimensional and Fact tables for seamless analysis.
- Generate actionable insights through advanced data analysis and visualization to address critical business needs.

b. Scope of Work:

- **Data Collection**: Collect data from the TIKI e-commerce platform using both API and Selenium.
- and load it into a centralized storage system using BigQuery.
- **Data Transformation**: Use SQL to create Dimensional (Dim) and Fact tables, ensuring data is structured and ready for analysis.
- **Data Analysis and Visualization**: Use Power BI to visualize and analyze the data.

c. Purpose

The goal of this project is to create a robust data processing system that supports critical decision-making in core business functions. Through the automation of data collection, transformation, and analysis processes, the project aims to address essential business questions such as:

- Identify trends and sales performance over time on TIKI in the electronics sector.
- Optimize inventory management.
- Identify high-performing stores on TIKI.

Ultimately, the project seeks to empower stakeholders with reliable, actionable insights that improve operational efficiency, enhance customer satisfaction, and drive overall business growth.

d. Expected Outcome:

- A fully automated and scalable data pipeline for seamless data collection, transformation, and analysis.

- Enhanced decision-making capabilities by providing actionable insights through interactive dashboards in Power BI.
- Improved operational efficiency, helping the business stay competitive and responsive to market demands.

1.3. Business Questions

a. How has sales performance changed in the second half of December 2024 until now (January 5, 2025) in the electronics sector on the TIKI e-commerce platform?

- What is the overall sales performance for the period 15/12/2024 to 5/1/2025?
- Which warehouse model has the highest sales, and why?
- Which products are the best sellers in stores, and what does this reflect?
- How do customers rate products and services?
- What role do technology products and home appliances play in this period?

b. How has sales performance changed by Shop in the electronics sector on the TIKI e-commerce platform?

- How are the number of ratings and average ratings related across stores?
- How is customer satisfaction distributed across shops?

1.4. Technical Tools

1.4.1 Apache Nifi & Apache Airflow

a. Apache Nifi

An effective open-source solution for automating and controlling intricate data flows is Apache NiFi. Its user-friendly interface makes it simple to set up and monitor data operations by enabling users to configure them. NiFi can forward data to storage systems like BigQuery and SQL Server and facilitates the gathering and transfer of data from a range of sources, including the web and APIs. NiFi's features, which include data transformation, routing, and flow control, enable users to design scalable and adaptable data processes. The ability of Apache NiFi to interact with BigQuery and SQL Server, which simplifies data intake and processing, is one of its advantages. This guarantees the dependability of your data-driven solutions while simultaneously enhancing performance. Users can easily monitor and manage data flows thanks to NiFi's user-friendly and intuitive interface, which streamlines workflows and boosts productivity.

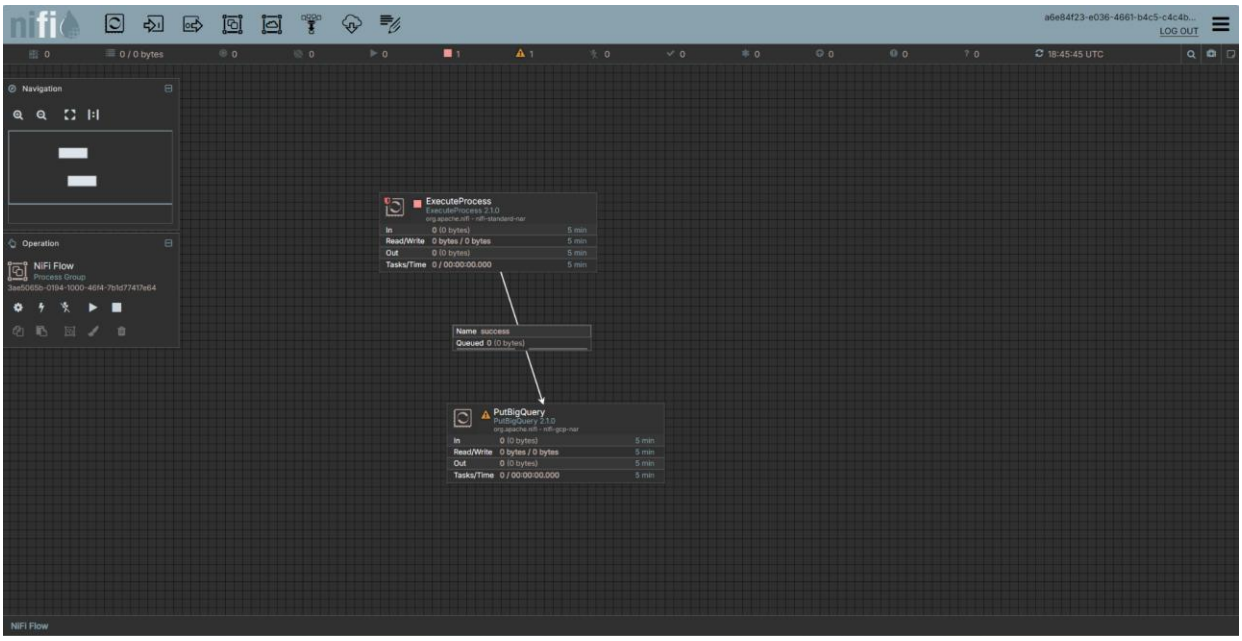

Figure 1: Nifi Menu

b. Apache Airflow

Apache Airflow is a robust, open-source workflow orchestration tool designed to automate and manage complex data pipelines. It enables users to define workflows as Directed Acyclic Graphs (DAGs) using Python, providing flexibility and scalability for various data processes. Apache Airflow can streamline the entire pipeline, from crawling web or API data, uploading it to BigQuery, transforming it into Dimensional and Fact tables in SQL, to preparing the data for visualization and analysis in Power BI. With features like task scheduling, monitoring, and error handling, Airflow ensures a seamless execution of interdependent tasks while offering a user-friendly interface to track progress. Its integration capabilities with platforms like BigQuery and SQL Server make it an ideal choice for automating workflows, improving efficiency, and maintaining reliability in your data-driven solutions.

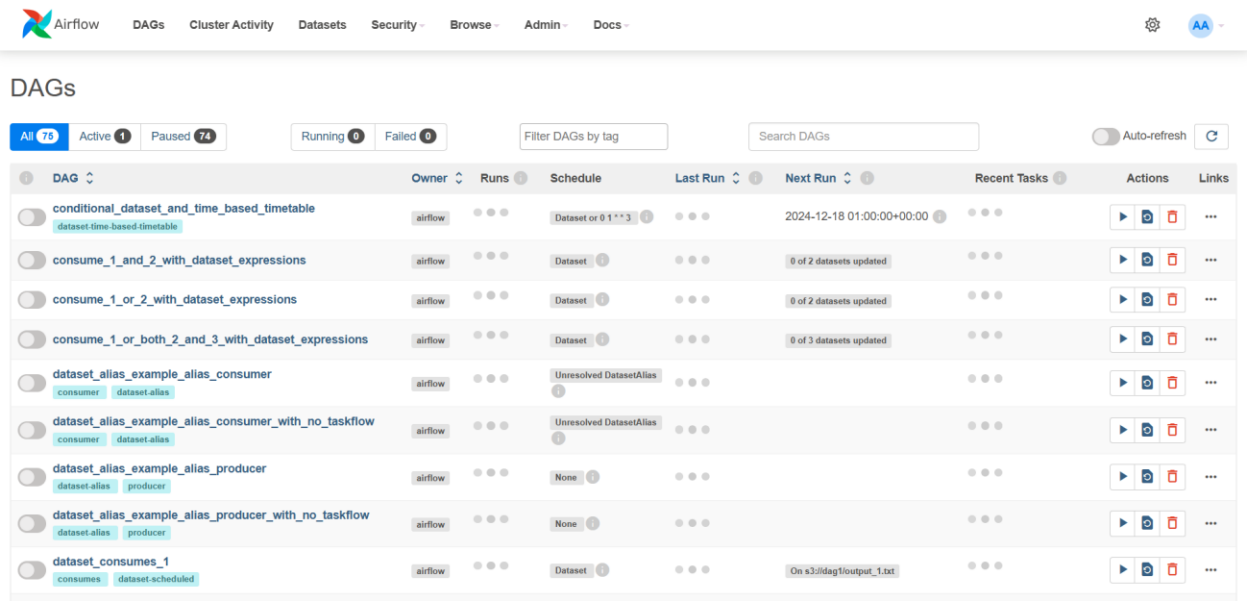

Figure 2: Airflow Menu

c. Difference Between Airflow and Nifi

Criteria	Apache Airflow	Apache NiFi
Main purpose	- Manage and automate complex workflows. - Focus on orchestration.	- Automate and manage **data flows** in real time. - Focus on data **flow**.
How to design a flow	- Based on **DAG** (Directed Acyclic Graph): Tasks are programmed in Python. - Suitable for **batch** workflows.	- **Based on drag-and-drop** interface: Connect processors to process data. - Suitable for **real-time**.
Usability	- Requires users to know **Python programming** to define workflow. - Takes time to get used to but is more flexible.	- **User-friendly** thanks to intuitive graphical interface. - No programming skills required, easy to use.
Main use cases	- Schedule recurring jobs: ETL, load data into data warehouse. - Integrate pipeline for **machine learning**. - Manage dependencies between complex tasks.	- Real-time data stream processing: ingestion, routing. - Event-driven data distribution between systems.

Scalability	- Scale by increasing the number of **worker nodes**. - Good for large **batch** workloads.	- Extend

1.4.2 Ubuntu

In this project, we use the Ubuntu operating system instead of Docker software to run Airflow. Ubuntu is the world's favorite Linux operating system. To install Ubuntu on Windows, you just need to download it from Microsoft Store and install. Ubuntu will run parallel to Windows and all tasks in the Ubuntu operating system must be performed in Command Prompt.

Figure 3: Ubuntu Command Prompt

1.4.3 GG Cloud Platform and Big Query

Google Cloud Platform (GCP) is a comprehensive collection of cloud computing services offered by Google. It provides various solutions such as computing resources, storage, databases, networking, machine learning, and more, all accessible via the internet. GCP empowers businesses to efficiently create, deploy, and scale applications and services without the need to maintain or invest in on-site infrastructure.

BigQuery is a serverless data warehouse and analytics service offered by GCP. It enables users to process and analyze vast amounts of data quickly and efficiently by leveraging Google's powerful infrastructure.

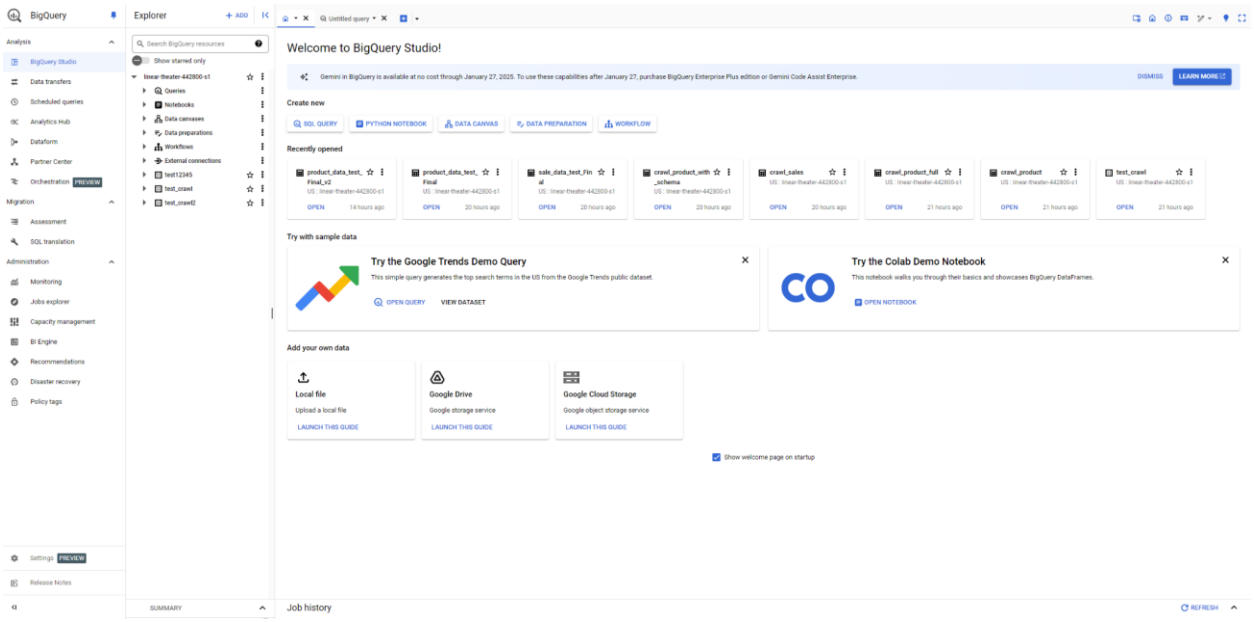

Figure 4: Google Big Query Menu

1.4.4 SSMS

Microsoft SQL Server Management Studio (SSMS) is a robust integrated solution for managing and administering Microsoft SQL Server. Writing and running SQL queries, managing databases, and configuring security are all made simple for users by SSMS's user-friendly graphical interface. In addition to managing user roles and access privileges to safeguard sensitive data, this application helps users create, edit, and remove databases and tables. Additionally, SQL Server performance monitoring and tracking functions are offered by SSMS, enabling prompt problem identification and resolution. Additionally, it makes database backup and recovery simple. In addition, SSMS can be combined with BigQuery to optimize data analysis and data warehouse management processes, helping users convert and synchronize information between SQL Server and Google's powerful analytics platform.

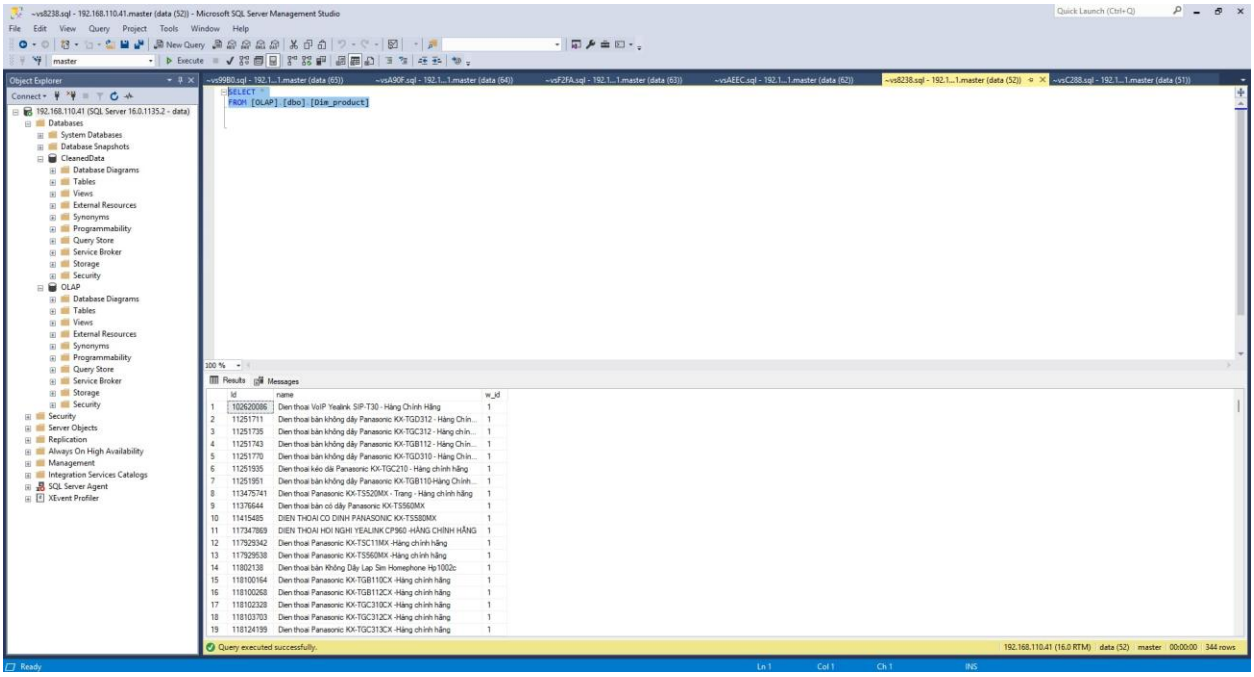

Figure 5: SSMS

1.4.5 Visual Studio Code

Visual Studio Code (VS Code) is a versatile and lightweight code editor by Microsoft, ideal for various development tasks. For your electronics store automation project, VS Code facilitates the workflow across stages like web crawling, API data extraction, data transformation in BigQuery, and visualizing insights in Power BI. Its integrated support for Python, BigQuery extensions, and SQL tools ensures seamless collaboration, efficient debugging, and streamlined development. By leveraging its features, your project gains enhanced productivity and organization, delivering valuable business insights.

1.4.6 Power BI

Power BI is a powerful business analytics tool from Microsoft that enables users to visualize and share insights from their data. It allows seamless connection to various data sources, transforming raw data into interactive reports and dashboards. With its intuitive interface, Power BI simplifies complex data analysis and helps users make informed, data-driven decisions. The tool also integrates well with other Microsoft products, enhancing its capabilities. Its cloud-based features provide real-time updates, making it ideal for businesses of all sizes. Power BI is designed to turn data into actionable insights quickly and effectively.

In my project, Power BI is used for data transformation, visualization, and analysis. It helps transform raw data into meaningful insights and interactive reports, making it easier to analyze and make data-driven decisions.

II. Database Preparation

2.1 Source Data & ERD

2.1.1 Source Data

Data is collected from the Tiki e-commerce platform with the main goal of collecting information about products in the electronics industry. Specifically, the data includes product information (name, price, reviews, description, discounts, product categories), store information (store name, URL, number of reviews), sales information (quantity sold, collection date) and review information (comment content, review score, comment posting date). To carry out the collection process, the project uses different methods such as API, Selenium and Apache NiFi. API is used to call data from Tiki in a stable manner and avoid overload, helping to collect product lists and detailed information for each product. For data that does not support API, Selenium is applied to emulate the browser, by passing security measures to collect information from data tables such as Sale, Shop and Comment. At the same time, Apache NiFi supports data flow automation, transformation, and loading of data into a central storage system on BigQuery. This system helps ensure efficient data collection, while preparing complete and accurate data sources for business reporting, inventory forecasting, and marketing strategy building.

2.1.2 ERD of Source Data

Sale Table:

Contribution	**Description**
Sale_ID	A unique code for each sales transaction, used to identify each record.
Product_ID	Product code, links to the Product table to get related product information.
Seller_Name	Name of the seller or store that provides the product.
Seller_URL	Link to the seller's page on the Tiki platform.
Quantity_Sold	The number of products sold in that transaction.
Crawl_Date	Data collection date helps track sales time.

Product Table

Contribution	**Description**
Product_ID	Unique product code, used to identify each product.
Name	Product name.
Price	Product price at the time of data collection.
URL	Link to product page on Tiki platform.
Rating	Average product rating given by customers.
Date_Date	Date the product was posted to the platform.
Created_Time	Product data collection time.
Inventory	The quantity of the product in stock at the time of collection.
Description	Detailed description of the product.
Discount	Percentage or value of product discount.
Categories	The category the product belongs to, for example "Electronics", "Home Appliances".

Shop Table:

Contribution	**Description**
Shop_ID	Unique store code, used to identify the store.
Product_ID	The product code links to the Product table, which indicates which store the product belongs to.
Shop_Name	Name of the store that supplies the product.
Shop_URL	Link to the store page on Tiki.
Review_Count	Total number of reviews the store received.

Comment Table:

Contribution	**Description**
Comment_ID	Unique comment code, used to identify each comment.
Product_ID	The product code is commented out, linking to the Product table.
User	The name or identifier of the user who wrote the comment.
Time_on_Platform	Time the user spent on the Tiki platform before leaving a review.
Title	Short title of the comment, if any.
Comment_Date	Date of comment.
Comment	Detailed content of the comment.
Rating	Product rating by customers (usually 1-5 stars).

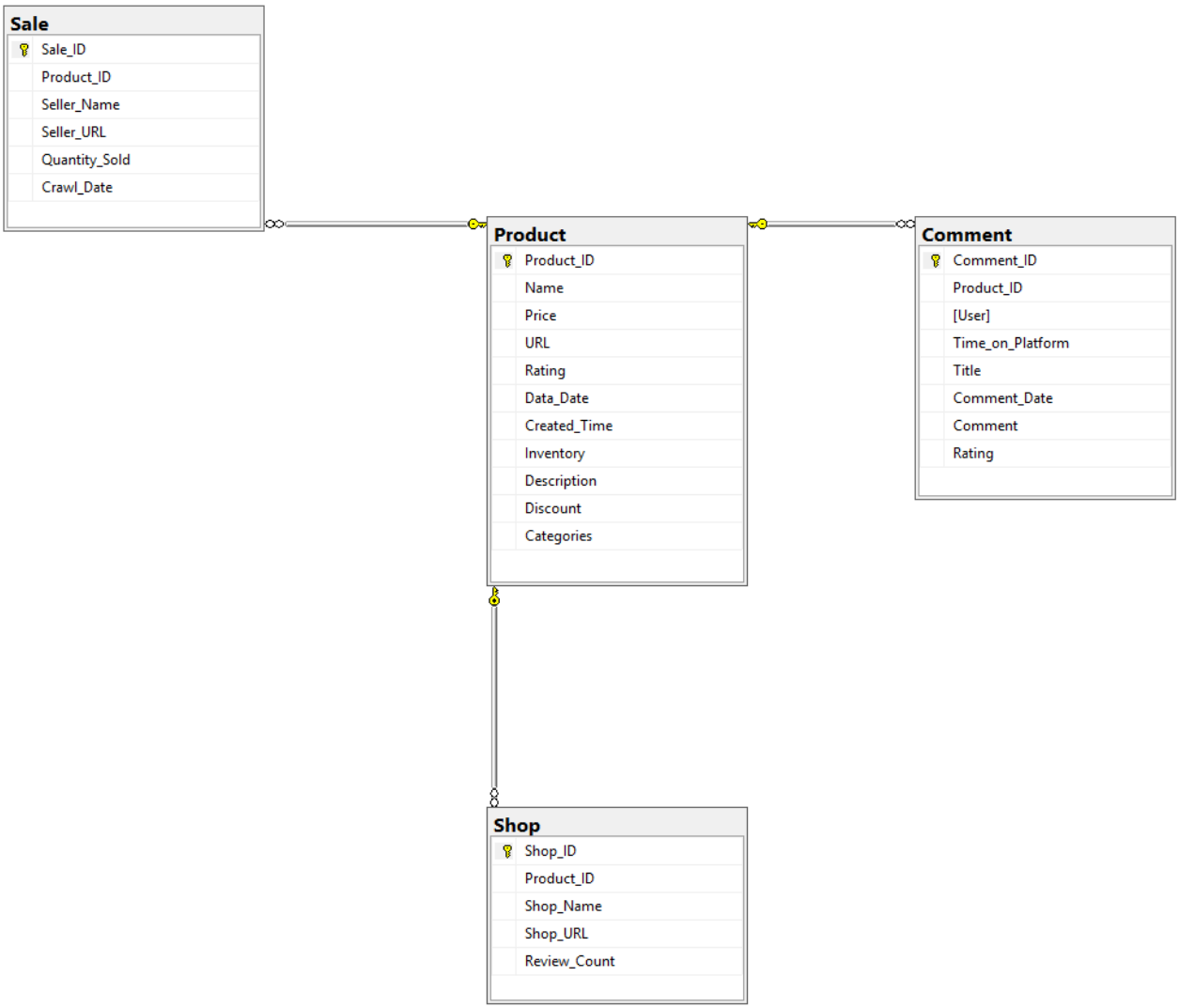

Figure 6: ERD of Source Data

2.2 ELT (Extract - Load - Transform) Design

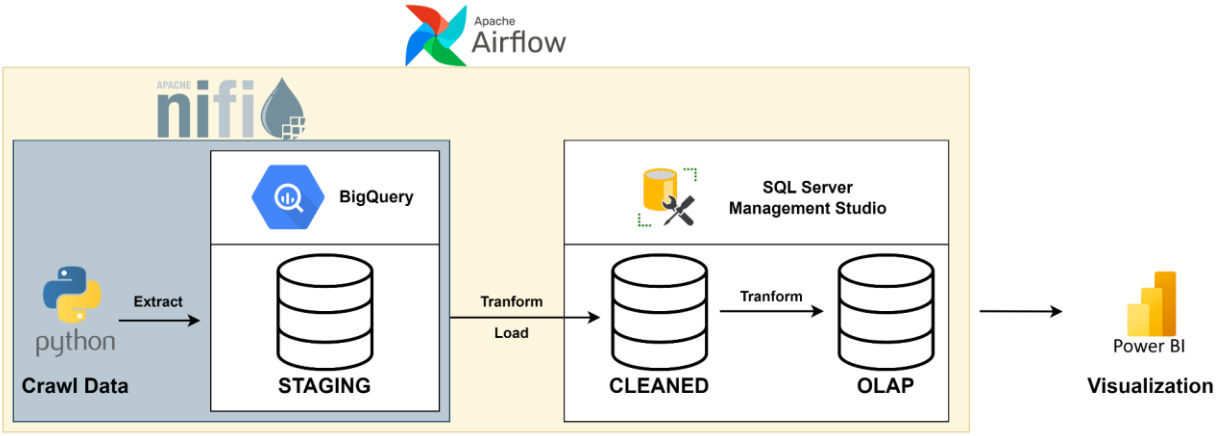

Figure 7: Data Pipeline

2.3 Configuration

2.3.1 Crawl and Extract data to BigQuery with NIFI

a. Crawl Product table with API

We will collect data for the Product table through three main functions utilizing TIKI's API. These functions will include ensuring stability when calling the API to avoid overload or connection failures, simulating a real browser to bypass API security measures, and implementing multiple retries to maximize the chances of successful data retrieval.

```python
def get_data_and_upload():
    # URL cơ bản cho API của Tiki để lấy danh sách sản phẩm
    base_url = "https://tiki.vn/api/personalish/v1/blocks/listings?limit=40&include=advertisement&aggregations=2&trackity_id=fb703a37-2f3d-2956-96ac-bd90536bb792&category={}&page="

    # Headers để giả lập yêu cầu từ trình duyệt
    headers = {
        "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36"
    }

    # Tạo một session với cơ chế retry
    session = requests.Session()
    retry_strategy = Retry(
        total=5,
        status_forcelist=[429, 500, 502, 503, 504],
        allowed_methods=["HEAD", "GET", "OPTIONS"],
        backoff_factor=1
    )
    adapter = HTTPAdapter(max_retries=retry_strategy)
    session.mount("https://", adapter)
    session.mount("http://", adapter)
```

Figure 8: Get_data_and_upload Function

- Crawl tiki data function

This function is designed to fetch a list of products from Tiki's API based on a specific category of ID. Key features include: automatically stopping when there is no more data or when the maximum number of products is reached, and error handling to prevent program interruptions.

```python
def crawl_tiki_data(category_id, max_products=1000):
    product_data = []
    page_num = 1
    while len(product_data) < max_products:
        url = base_url.format(category_id) + str(page_num)
        try:
            response = session.get(url, headers=headers)
            if response.status_code != 200:
                break
            data = response.json().get('data', [])
            if not data:
                break
            for item in data:
                product_id = item.get('id')
                product_url = f"https://tiki.vn/{item.get('url_path')}"

                product_data.append({
                    "Id": product_id,
                    "Tên": item.get('name'),
                    "Giá": item.get('price'),
                    "URL": product_url,
                    "Sao đánh giá": item.get('rating_average'),
                    "Ngày lấy dữ liệu": datetime.now().strftime('%Y-%m-%d'),
                })
                if len(product_data) >= max_products:
                    break
            page_num += 1
        except requests.exceptions.RequestException:
            break
    return product_data
```

Figure 9: Crawl_tiki_data Funtion

- Get product details function

The `get_product_details` function takes a `product_id` as input, constructs the corresponding API URL, and sends a GET request to retrieve detailed product information from Tiki. In case of a connection error or an unsuccessful response, the function will return an empty dictionary.

```python
def get_product_details(product_id):
    details_url = f"https://tiki.vn/api/v2/products/{product_id}"
    try:
        response = session.get(details_url, headers=headers)
        if response.status_code == 200:
            product_details = response.json()
            return {
                "Thời gian tạo sản phẩm": product_details.get('created_at'),
                "Kho hàng": product_details.get('inventory', {}).get('fulfillment_type', None),
                "Mô tả ngắn": product_details.get('short_description', None),
                "Giảm giá": product_details.get('discount', None),
                "Danh mục": json.dumps(product_details.get('categories', []))
            }
        else:
            return {}
    except requests.exceptions.RequestException:
        return {}

# Chọn ID danh mục và số lượng sản phẩm cần lấy
category_id = '1789'  # Thay thế bằng ID danh mục mong muốn
max_products = 10  # Số lượng sản phẩm để lấy (có thể điều chỉnh)

# Lấy dữ liệu danh sách sản phẩm
products = crawl_tiki_data(category_id, max_products)

# Lấy thông tin chi tiết sản phẩm và bổ sung vào danh sách
for product in products:
    details = get_product_details(product['Id'])
    product.update(details)

# Tạo DataFrame từ danh sách sản phẩm
df_products = pd.DataFrame(products)
```

Figure 10: Get_product_deltails Funtion

b. Crawl Sale, Shop, Comment table with Selenium

To crawl the Sale, Shop, and Comment tables, we will first use the function `run_bigquery_query_to_dataframe` to connect to Google BigQuery. This function will execute a query to select all data (`SELECT *`) with a limit of 1,000 rows and then convert the results into a pandas dataframe for further processing.

In addition, we will configure the Chrome Driver to operate with Selenium in headless mode. This configuration allows Selenium to run the browser without a graphical interface, enabling automated web interactions and data collection tasks efficiently.

```python
def run_bigquery_query_to_dataframe(json_key_path, project_id, dataset_id, table_id, query):
    """
    Executes a BigQuery SQL query and returns the result as a pandas DataFrame.
    """
    try:
        # Initialize BigQuery client
        client = bigquery.Client.from_service_account_json(json_key_path)

        # Construct full table reference if needed
        full_table_ref = f"{project_id}.{dataset_id}.{table_id}"

        # If no specific query provided, default to SELECT all from the table
        if not query:
            query = f"SELECT * FROM `{full_table_ref}` LIMIT 1000"

        # Execute query
        query_job = client.query(query)
        results = query_job.result()

        # Convert to pandas DataFrame
        dataframe = results.to_dataframe()
        return dataframe

    except Exception as e:
        print(f"Error while executing BigQuery query: {e}")
        return pd.DataFrame()

json_key_path = "/opt/nifi/scripts/linear-theater-442800-s1-26e4d70ab28a.json"
project_id = "linear-theater-442800-s1"
dataset_id = "test_crawl"
table_id = "test_product_table"
query = ''' SELECT DISTINCT * FROM `linear-theater-442800-s1.test_crawl.crawl_product_full` '''

df_products = run_bigquery_query_to_dataframe(json_key_path=json_key_path, project_id=project_id, dataset_id=dataset_id, table_id=table_id, query=query)

# Cài đặt và khởi chạy trình duyệt Chrome
options = webdriver.ChromeOptions()
options.binary_location = "/usr/bin/google-chrome"
options.add_argument("--headless")
options.add_argument("--no-sandbox")
options.add_argument("--disable-dev-shm-usage")
options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36")
driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)
```

Figure 11: Run_bigquery_query_to_dataframe Function

The next step is to collect data for each table by creating an empty list to store the data, and then iterating through each product from `df_product` using Selenium. Specifically, the data for the Sale, Comment, and Shop tables will be collected based on the products in the Product table that was previously gathered.

For the web interface, we will open the product URL, scroll down the page 10 times to load all detailed content, and refresh the page if necessary to ensure all content is fully displayed before starting the data collection process.

Most importantly, to collect data through Selenium, we must repeatedly check and extract elements in the HTML using the `find_element` method. Finally, the collected data will be stored in the corresponding columns of the initially created empty lists.

Below are the code snippets for collecting data for the three tables: Sale, Shop, and Comment

```python
df_sales_list = []

# Duyệt qua từng sản phẩm
for index, row in df_products.head(3).iterrows():
    product_id = row['Id']
    product_url = row['URL']

    # Mở trang web sản phẩm
    driver.get(product_url)
    driver.set_window_size(1920, 1080)
    time.sleep(5)

    # Cuộn và tải thêm nội dung
    for _ in range(10):
        driver.execute_script("window.scrollBy(0, 1000);")
        time.sleep(2)

    # Nhấn vào phần tử nếu cần
    try:
        clickable_element = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.CLASS_NAME, "class-to-trigger-content"))
        )
        clickable_element.click()
        time.sleep(5)
    except Exception:
        pass

    # Làm mới trang để tải nội dung đầy đủ
    driver.refresh()
    time.sleep(5)

    # Thu thập dữ liệu bán hàng
    try:
        seller_element = driver.find_element(By.CLASS_NAME, "seller-name")
        seller_link = seller_element.find_element(By.XPATH, '//span[@class="seller-name"]//a')
        seller_name = seller_element.text
        seller_url = seller_link.get_attribute('href')
    except Exception:
        seller_name, seller_url = "N/A", "N/A"

    try:
        quantity_sold_element = driver.find_element(By.XPATH, "//div[contains(text(), 'Đã bán')]")
        quantity_sold_text = quantity_sold_element.text.strip()
        quantity_sold = int(quantity_sold_text.split(" ")[-1])
    except Exception:
        quantity_sold = 0

    # Lưu dữ liệu vào danh sách
    df_sales_list.append({
        'Product_ID': product_id,
        'Seller_Name': seller_name,
        'Seller_URL': seller_url,
        'Quantity_Sold': quantity_sold,
        'Crawl_Date': datetime.now().strftime('%Y-%m-%d')
    })

# Chuyển danh sách thành DataFrame
df_sales = pd.DataFrame(df_sales_list)
```

Figure 12: The code for collecting data for Sale Table

```
df_shops_list = []

for index, row in df_products.head(3).iterrows():
    product_id = row['Id']
    product_url = row['URL']

    driver.get(product_url)
    driver.set_window_size(1920, 1080)
    time.sleep(5)

    for _ in range(10):
        driver.execute_script("window.scrollBy(0, 1000);")
        time.sleep(2)

    try:
        clickable_element = WebDriverWait(driver, 10).until(
            EC.element_to_be_clickable((By.CLASS_NAME, "class-to-trigger-content"))
        )
        clickable_element.click()
        time.sleep(5)
    except Exception:
        pass

    driver.refresh()
    time.sleep(5)

    try:
        sellers = driver.find_element(By.CLASS_NAME, "seller-name")
        seller_links = driver.find_element(By.XPATH, '//span[@class="seller-name"]//a')
        seller_name = sellers.text
        seller_url = seller_links.get_attribute('href')
    except Exception:
        seller_name, seller_url = "N/A", "N/A"

    try:
        review_count_element = driver.find_element(By.CLASS_NAME, "sub-title")
        review_count = review_count_element.text
    except Exception:
        review_count = "N/A"

    df_shops_list.append({
        'Product_ID': product_id,
        'Shop_Name': seller_name,
        'Shop_URL': seller_url,
        'Review_Count': review_count
    })

df_shops = pd.DataFrame(df_shops_list)
if not df_shops .empty:
    df_shops.to_csv(sys.stdout, index=False, encoding="utf-8-sig")
else:
    print("No products fetched.", file=sys.stderr)
```

Figure 13: The code for collecting data for Shop Table

```python
data_list = []

for index, row in df_products.head(3).iterrows():
    product_id = row['Id']
    product_url = row['URL']

    driver.get(product_url)
    time.sleep(5)

    for i in range(5):
        driver.execute_script("window.scrollBy(0, 1000);")
        time.sleep(2)

    try:
        comments = driver.find_elements(By.CLASS_NAME, "review-comment__content")
        users = driver.find_elements(By.CLASS_NAME, "review-comment__user-name")
        dates_on_platform = driver.find_elements(By.CLASS_NAME, "review-comment__user-date")
        titles = driver.find_elements(By.CLASS_NAME, "review-comment__title")
        review_dates = driver.find_elements(By.CLASS_NAME, "review-comment__created-date")
        ratings = driver.find_elements(By.CLASS_NAME, "review-comment__rating-title")

        stars = []
        for rating in ratings:
            try:
                width_div = rating.find_element(By.CSS_SELECTOR, "div[style*='width']")
                style = width_div.get_attribute("style")
                if "width" in style:
                    width_percent = int(style.split("width:")[1].strip().replace("%;", "").replace(";", ""))
                    star_count = width_percent // 20
                    stars.append(star_count)
            except Exception:
                stars.append(0)

        for i in range(min(len(comments), 10)):
            comment = comments[i].text if i < len(comments) else "N/A"
            user = users[i].text if i < len(users) else "N/A"
            date_on_platform = dates_on_platform[i].text if i < len(dates_on_platform) else "N/A"
            title = titles[i].text if i < len(titles) else "N/A"
            created_date = review_dates[i].text if i < len(review_dates) else "N/A"
            star = stars[i] if i < len(stars) else 0

            data_list.append({
                'Product_ID': product_id,
                'User': user,
                'Time_on_Platform': date_on_platform,
                'Title': title,
                'Comment_Date': created_date,
                'Comment': comment,
                'Rating': star
            })
    except Exception:
        pass

driver.quit()

df_comments = pd.DataFrame(data_list)
if not df_comments.empty:
    df_comments.to_csv(sys.stdout, index=False, encoding="utf-8-sig")
else:
    print("No comments found.", file=sys.stderr)
```

Figure 14: The code for collecting data for Comment Table

c. Combined with Nifi Apache

To integrate the Python files used for crawling with Apache NiFi, for each table and each Python file, we will use a separate flow branch as follows:

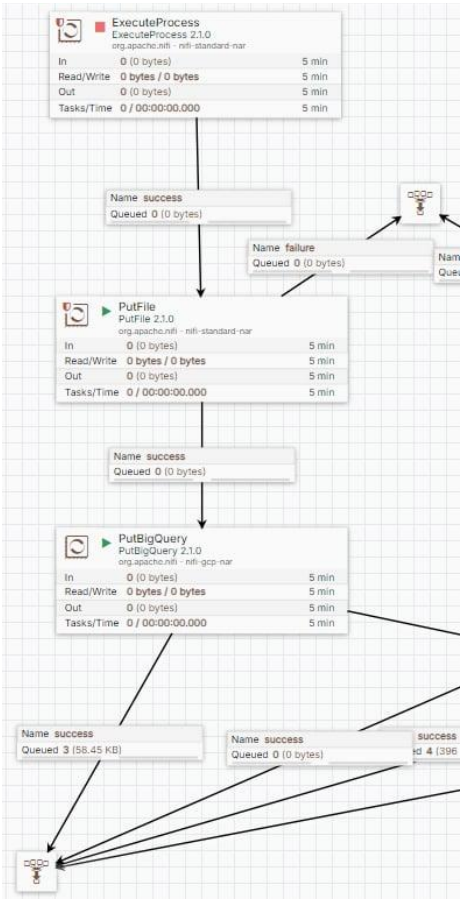

Figure 15: NiFi Flow for scraping and pushing Product table data to BigQuery.

The main steps include using **ExecuteProcess** to execute the Python file, which is used for crawling data. After that, we use **PutBigQuery** to upload the collected data to the specified dataset and table.

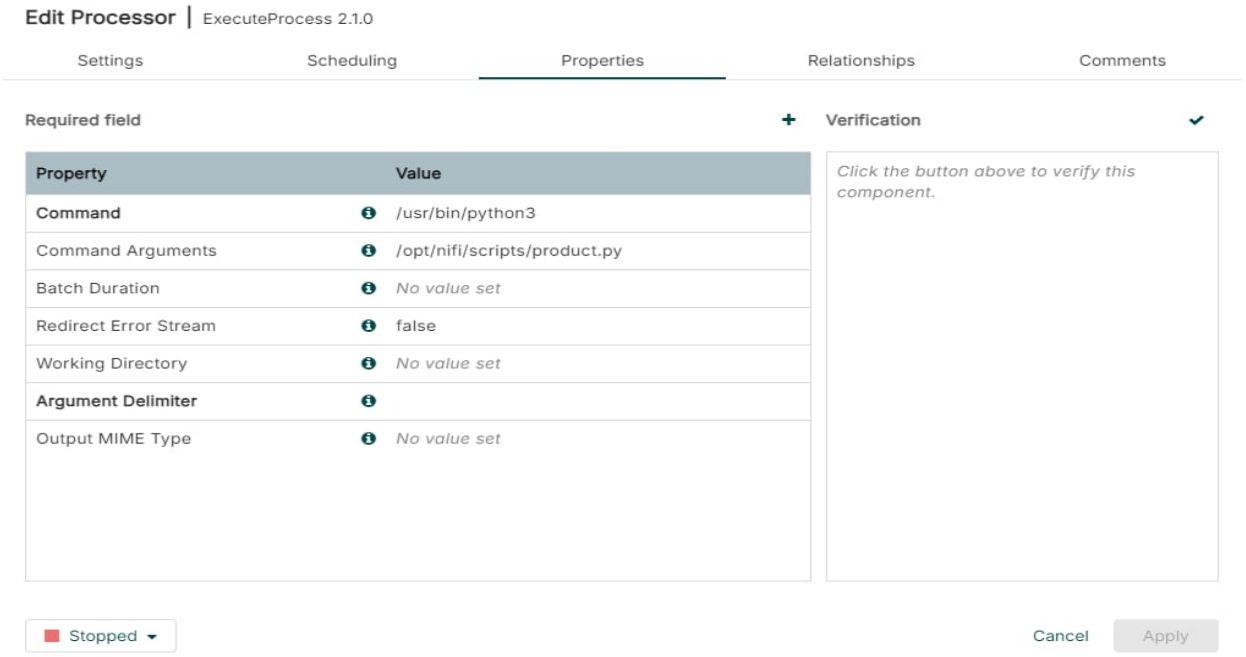

Figure 16: Execute Process

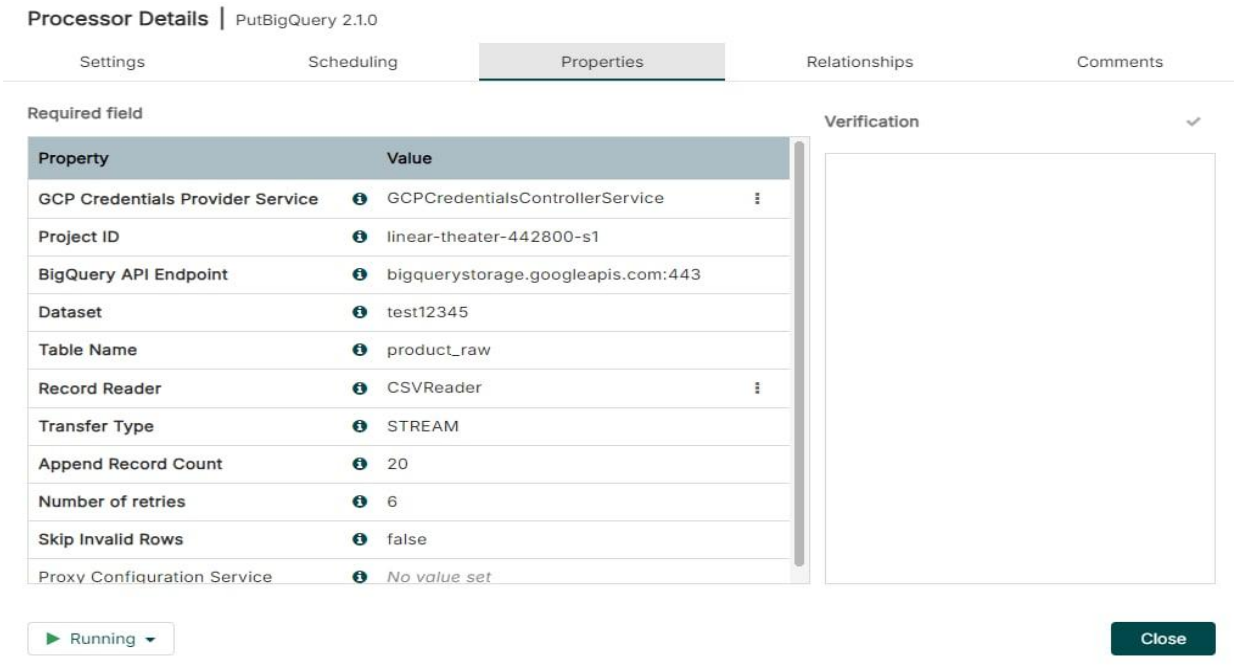

Figure 17: PutBigQuery Process

2.3.2 Running NIFI

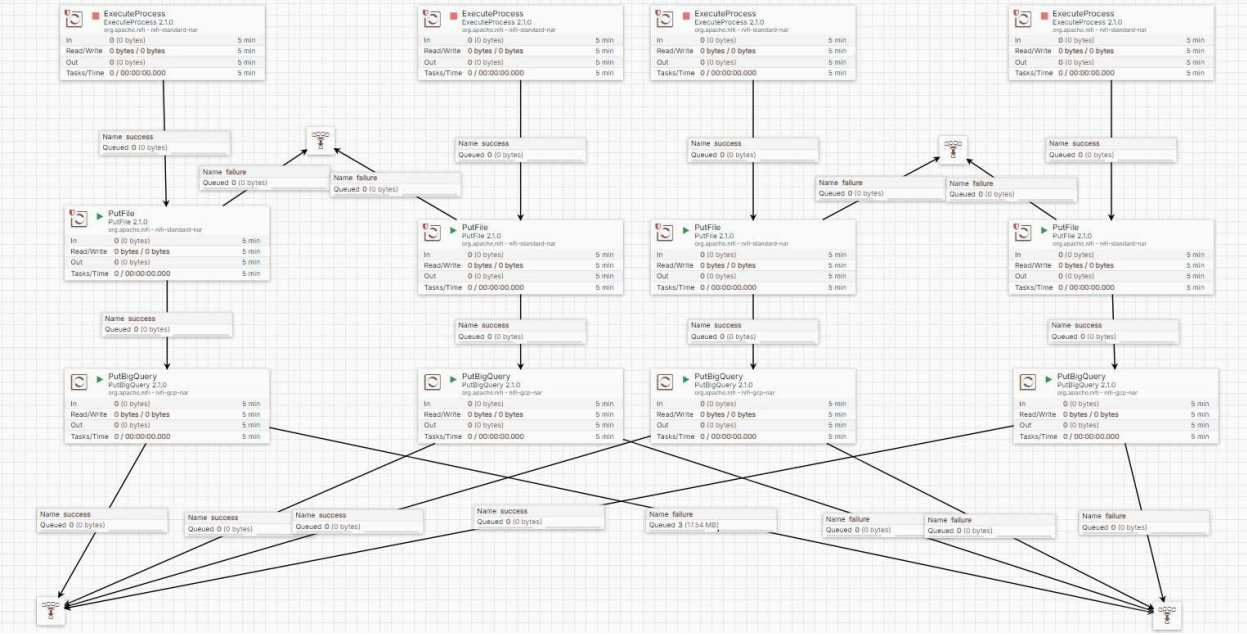

Figure 18: Nifi flow

2.3.3 STAGING to CLEANED with Airflow and Ubuntu

In this step, we use Airflow and Ubuntu to automate the data cleaning process by selecting only unique values (SELECT DISTINCT) from the STAGING dataset on Google BigQuery. The cleaned tables are then uploaded to the CLEANED dataset in SQL Server. This process includes setting up connections, executing data cleaning queries, defining data loading functions, and transferring the data.

Below is the syntax for the cleaning functions for each table: Product, Comment, Shop, and Sale.

```python
def task_product_clean():
    json_key_path = '/opt/airflow/dags/linear-theater-442800-s1-26e4d70ab28a.json'
    query = ''' SELECT distinct Id, `Tên`as name
    ,`Giá` as price
    ,URL
    ,`Sao đánh giá` as rating
    ,`Ngày lấy dữ liệu` as crawl_date
    ,`Thời gian tạo sản phẩm` as created_at
    ,`Kho hàng` as warehouse
    ,`Mô tả ngắn` as discribe
    ,`Giảm giá` as discount
    , `Danh mục` as danhmuc
    FROM `linear-theater-442800-s1.test_crawl.crawl_product_full` '''
    client = bigquery.Client.from_service_account_json(json_key_path)
    df = client.query(query).to_dataframe()
    server_name = '192.168.110.41'  # Thay bằng tên server của bạn
    database_name = 'CleanedData'            # Thay bằng tên database của bạn
    table_name = 'Product_clean'      # Tên bảng
    username = "data"
    password = "data123"
    def push_to_sqlserver(dataframe, table_name, server_name, database_name):
        try:
            # Đảm bảo mọi dữ liệu đều là chuỗi trước khi đẩy vào SQL Server
            dataframe = dataframe.applymap(lambda x: str(x) if pd.notnull(x) else None)

            # Chuỗi kết nối với Windows Authentication
            connection_string = (
                f"mssql+pyodbc://{username}:{password}@{server_name}/{database_name}"
                f"?driver=ODBC+Driver+17+for+SQL+Server"
            )
            engine = create_engine(connection_string, fast_executemany=True)

            # Đẩy dữ liệu vào SQL Server
            dataframe.to_sql(
                name=table_name,
                con=engine,
                if_exists='replace',
                index=False,
                dtype={
                    'Id': NVARCHAR(None),
                    'Name': NVARCHAR(None),
                    'warehouse': NVARCHAR(None),
                    'discribe' :NVARCHAR(None),
                    'danhmuc' :NVARCHAR(None),
                    'crawl_date': Date,
                'created_at': Date,
                'rating': Float,
                'discount': Float,
                'price': Float
                }
            )
            print(f"Đã đẩy dữ liệu thành công vào bảng {table_name} trong database {database_name}.")
        except Exception as e:
            print(f"Lỗi khi đẩy dữ liệu vào SQL Server: {e}")
    push_to_sqlserver(df, table_name, server_name, database_name)
```

Figure 19: Clean and Push to SQL Function for Product Table

```python
def task_comment_clean():
    json_key_path = '/opt/airflow/dags/linear-theater-442800-s1-26e4d70ab28a.json'
    query = ''' SELECT distinct
    Product_id,
    User,
    Time_on_Platform,
    Title,
    Comment_Date,
    Comment,
    Rating
    FROM `linear-theater-442800-s1.test_crawl.crawl_comment`
    '''
    client = bigquery.Client.from_service_account_json(json_key_path)
    df = client.query(query).to_dataframe()
    server_name = '192.168.110.41'  # Thay bằng tên server của bạn
    database_name = 'CleanedData'            # Thay bằng tên database của bạn
    table_name = 'Comment_clean'      # Tên bảng
    username = "data"
    password = "data123"
    def push_to_sqlserver(dataframe, table_name, server_name, database_name):
        try:
            # Đảm bảo mọi dữ liệu đều là chuỗi trước khi đẩy vào SQL Server
            dataframe = dataframe.applymap(lambda x: str(x) if pd.notnull(x) else None)

            # Chuỗi kết nối với Windows Authentication
            connection_string = (
                f"mssql+pyodbc://{username}:{password}@{server_name}/{database_name}"
                f"?driver=ODBC+Driver+17+for+SQL+Server"
            )
            engine = create_engine(connection_string, fast_executemany=True)

            # Đẩy dữ liệu vào SQL Server
            dataframe.to_sql(
                name=table_name,
                con=engine,
                if_exists='replace',
                index=False,
                dtype={
                    'Product_id': NVARCHAR(None),
                    'User': NVARCHAR(None),
                    'Time_on_Platform': NVARCHAR(None),
                    'Title' :NVARCHAR(None),
                    'Comment_Date' :NVARCHAR(None),
                    'Comment':NVARCHAR(None),
                    'Rating': Float

                }
            )
            print(f"Đã đẩy dữ liệu thành công vào bảng {table_name} trong database {database_name}.")
        except Exception as e:
            print(f"Lỗi khi đẩy dữ liệu vào SQL Server: {e}")
    push_to_sqlserver(df, table_name, server_name, database_name)
```

Figure 20: Clean and Push to SQL Function for Comment Table

```python
def task_shop_clean():
    json_key_path = '/opt/airflow/dags/linear-theater-442800-s1-26e4d70ab28a.json'
    query = ''' SELECT distinct
    Product_id,
    Shop_Name,
    Shop_URL,
    Review_Count
    FROM `linear-theater-442800-s1.test_crawl.crawl_product`
    where Shop_Name <> "N/A"
    '''
    client = bigquery.Client.from_service_account_json(json_key_path)
    df = client.query(query).to_dataframe()
    server_name = '192.168.110.41'  # Thay bằng tên server của bạn
    database_name = 'CleanedData'            # Thay bằng tên database của bạn
    table_name = 'Shop_clean'      # Tên bảng
    username = "data"
    password = "data123"
    def push_to_sqlserver(dataframe, table_name, server_name, database_name):
        try:
            # Đảm bảo mọi dữ liệu đều là chuỗi trước khi đẩy vào SQL Server
            dataframe = dataframe.applymap(lambda x: str(x) if pd.notnull(x) else None)

            # Chuỗi kết nối với Windows Authentication
            connection_string = (
                f"mssql+pyodbc://{username}:{password}@{server_name}/{database_name}"
                f"?driver=ODBC+Driver+17+for+SQL+Server"
            )
            engine = create_engine(connection_string, fast_executemany=True)

            # Đẩy dữ liệu vào SQL Server
            dataframe.to_sql(
                name=table_name,
                con=engine,
                if_exists='replace',
                index=False,
                dtype={
                    'Product_id': NVARCHAR(None),
                    'Shop_Name': NVARCHAR(None),
                    'Shop_URL': NVARCHAR(None),
                    'Review_Count' :NVARCHAR(None)
                }
            )
            print(f"Đã đẩy dữ liệu thành công vào bảng {table_name} trong database {database_name}.")
        except Exception as e:
            print(f"Lỗi khi đẩy dữ liệu vào SQL Server: {e}")
    push_to_sqlserver(df, table_name, server_name, database_name)
```

Figure 21: Clean and Push to SQL Function for Shop Table

```python
def task_sale_clean():
    json_key_path = '/opt/airflow/dags/linear-theater-442800-s1-26e4d70ab28a.json'
    query = ''' SELECT distinct
    Product_id,
    seller_name,
    seller_url,
    Quantity_Sold,
    Crawl_Date
    FROM `linear-theater-442800-s1.test_crawl.crawl_sales`
    where seller_name <> "N/A"
    '''
    client = bigquery.Client.from_service_account_json(json_key_path)
    df = client.query(query).to_dataframe()
    server_name = '192.168.110.41'  # Thay bằng tên server của bạn
    database_name = 'CleanedData'            # Thay bằng tên database của bạn
    table_name = 'Sale_clean'      # Tên bảng
    username = "data"
    password = "data123"
    def push_to_sqlserver(dataframe, table_name, server_name, database_name):
        try:
            # Đảm bảo mọi dữ liệu đều là chuỗi trước khi đẩy vào SQL Server
            dataframe = dataframe.applymap(lambda x: str(x) if pd.notnull(x) else None)

            # Chuỗi kết nối với Windows Authentication
            connection_string = (
                f"mssql+pyodbc://{username}:{password}@{server_name}/{database_name}"
                f"?driver=ODBC+Driver+17+for+SQL+Server"
            )
            engine = create_engine(connection_string, fast_executemany=True)

            # Đẩy dữ liệu vào SQL Server
            dataframe.to_sql(
                name=table_name,
                con=engine,
                if_exists='replace',
                index=False,
                dtype={
                    'Product_id': NVARCHAR(None),
                    'seller_name': NVARCHAR(None),
                    'seller_url': NVARCHAR(None),
                    'Quantity_Sold' :Float,
                    'Crawl_Date':Date
                }
            )
            print(f"Đã đẩy dữ liệu thành công vào bảng {table_name} trong database {database_name}.")
        except Exception as e:
            print(f"Lỗi khi đẩy dữ liệu vào SQL Server: {e}")
    push_to_sqlserver(df, table_name, server_name, database_name)
```

Figure 22: Clean and Push to SQL Function for Sale Table

2.3.4 CLEANED to OLAP with Airflow and Ubuntu

After successfully transferring the CLEANED dataset to SQL Server, we will use this data to transform it into dimension tables and fact tables based on the dimension model, and then load them into the OLAP dataset. Below is the syntax for each individual dimension and fact table:

```
def  task_dim_warehouse():
    query = '''select distinct
      [warehouse]
    FROM [CleanedData].[dbo].[Product_clean]
    '''
    df = read_from_sqlserver(query)
    df['w_id'] = range(1, len(df) + 1)

    def push_to_sqlserver(dataframe, table_name, if_exists='replace'):
        server_name = '192.168.110.41'  # Thay bằng tên server của bạn
        database_name = 'OLAP'            # Thay bằng tên database của bạn
        username = "data"
        password = "data123"
        try:
            # Chuỗi kết nối SQL Server
            connection_string = (
                f"mssql+pyodbc://{username}:{password}@{server_name}/{database_name}"
                f"?driver=ODBC+Driver+17+for+SQL+Server"
            )

            # Tạo engine kết nối
            engine = create_engine(connection_string, fast_executemany=True)

            # Đẩy dữ liệu vào SQL Server
            dataframe.to_sql(
                name=table_name,
                con=engine,
                if_exists=if_exists,
                index=False
            )
            print(f"Đã đẩy dữ liệu thành công vào bảng {table_name} trong database {database_name}.")
        except Exception as e:
            print(f"Lỗi khi đẩy dữ liệu vào SQL Server: {e}")
    push_to_sqlserver(df, table_name = 'Dim_warehouse', if_exists='replace')
```

Figure 23: Cleaned to OLAP dataset Function for Warehouse Dimension Table

```python
def  task_dim_product():
    query = '''SELECT a.[Id]
      ,a.[name]
      ,b.w_id
    FROM [CleanedData].[dbo].[Product_clean] a
    left join OLAP.dbo.Dim_warehouse b on a.warehouse = b.warehouse
    '''
    def push_to_sqlserver(dataframe, table_name, if_exists='replace'):
        server_name = '192.168.110.41'  # Thay bằng tên server của bạn
        database_name = 'OLAP'            # Thay bằng tên database của bạn
        username = "data"
        password = "data123"
        try:
            # Chuỗi kết nối SQL Server
            connection_string = (
                f"mssql+pyodbc://{username}:{password}@{server_name}/{database_name}"
                f"?driver=ODBC+Driver+17+for+SQL+Server"
            )

            # Tạo engine kết nối
            engine = create_engine(connection_string, fast_executemany=True)

            # Đẩy dữ liệu vào SQL Server
            dataframe.to_sql(
                name=table_name,
                con=engine,
                if_exists=if_exists,
                index=False,
                dtype={
                    'name': NVARCHAR(None)
                }
            )
            print(f"Đã đẩy dữ liệu thành công vào bảng {table_name} trong database {database_name}.")
        except Exception as e:
            print(f"Lỗi khi đẩy dữ liệu vào SQL Server: {e}")
    df = read_from_sqlserver(query)
    push_to_sqlserver(df, table_name = 'Dim_product', if_exists='replace')
```

Figure 24: Cleaned to OLAP dataset Function for Product Dimension Table

```python
def  task_dim_user():
    query = '''SELECT distinct
      [User]
      ,[Time_on_Platform]
    FROM [CleanedData].[dbo].[Comment_clean]
'''
    df = read_from_sqlserver(query)
    df['user_id'] = range(1, len(df) + 1)
    def push_to_sqlserver(dataframe, table_name, if_exists='replace'):
        server_name = '192.168.110.41'  # Thay bằng tên server của bạn
        database_name = 'OLAP'            # Thay bằng tên database của bạn
        username = "data"
        password = "data123"
        try:
            # Chuỗi kết nối SQL Server
            connection_string = (
                f"mssql+pyodbc://{username}:{password}@{server_name}/{database_name}"
                f"?driver=ODBC+Driver+17+for+SQL+Server"
            )

            # Tạo engine kết nối
            engine = create_engine(connection_string, fast_executemany=True)

            # Đẩy dữ liệu vào SQL Server
            dataframe.to_sql(
                name=table_name,
                con=engine,
                if_exists=if_exists,
                index=False,
                dtype={
                    'Time_on_Platform': NVARCHAR(None),
                    'User': NVARCHAR(None)
                }
            )
            print(f"Đã đẩy dữ liệu thành công vào bảng {table_name} trong database {database_name}.")
        except Exception as e:
            print(f"Lỗi khi đẩy dữ liệu vào SQL Server: {e}")
    push_to_sqlserver(df, table_name = 'Dim_user', if_exists='replace')
```

Figure 25: Cleaned to OLAP dataset Function for User Dimension Table

```python
def  task_dim_shop():
    query = '''SELECT distinct
      [Shop_Name]
      ,[Shop_URL]
      ,[Review_Count]
    FROM [CleanedData].[dbo].[Shop_clean]
    '''
    df = read_from_sqlserver(query)
    df['shop_id'] = range(1, len(df) + 1)
    def push_to_sqlserver(dataframe, table_name, if_exists='replace'):
        server_name = '192.168.110.41'  # Thay bằng tên server của bạn
        database_name = 'OLAP'            # Thay bằng tên database của bạn
        username = "data"
        password = "data123"
        try:
            # Chuỗi kết nối SQL Server
            connection_string = (
                f"mssql+pyodbc://{username}:{password}@{server_name}/{database_name}"
                f"?driver=ODBC+Driver+17+for+SQL+Server"
            )

            # Tạo engine kết nối
            engine = create_engine(connection_string, fast_executemany=True)

            # Đẩy dữ liệu vào SQL Server
            dataframe.to_sql(
                name=table_name,
                con=engine,
                if_exists=if_exists,
                index=False,
                dtype={
                    'Shop_Name': NVARCHAR(None),
                    'Review_Count': NVARCHAR(None)
                }
            )
            print(f"Đã đẩy dữ liệu thành công vào bảng {table_name} trong database {database_name}.")
        except Exception as e:
            print(f"Lỗi khi đẩy dữ liệu vào SQL Server: {e}")
    push_to_sqlserver(df, table_name = 'Dim_shop', if_exists='replace')
```

Figure 26: Cleaned to OLAP dataset Function for Shop Dimension Table

```python
def task_dim_crawl_date():
    query = '''
    SELECT DISTINCT
        CAST(Crawl_Date AS DATE) AS DateKey,
        YEAR(CAST(Crawl_Date AS DATE)) AS Year,
        MONTH(CAST(Crawl_Date AS DATE)) AS Month,
        DAY(CAST(Crawl_Date AS DATE)) AS Day,
        DATENAME(WEEKDAY, CAST(Crawl_Date AS DATE)) AS DayName,
        DATEPART(QUARTER, CAST(Crawl_Date AS DATE)) AS Quarter,
        DATEPART(WEEK, CAST(Crawl_Date AS DATE)) AS Week,
        CASE
            WHEN DATEPART(WEEKDAY, CAST(Crawl_Date AS DATE)) IN (1, 7) THEN 'Weekend'
            ELSE 'Weekday'
        END AS IsWeekend
    FROM CleanedData.dbo.Sale_clean
    ORDER BY DateKey;
    '''
    df = read_from_sqlserver(query)
    def push_to_sqlserver(dataframe, table_name, if_exists='replace'):
        server_name = '192.168.110.41'  # Thay bằng tên server của bạn
        database_name = 'OLAP'            # Thay bằng tên database của bạn
        username = "data"
        password = "data123"
        try:
            # Chuỗi kết nối SQL Server
            connection_string = (
                f"mssql+pyodbc://{username}:{password}@{server_name}/{database_name}"
                f"?driver=ODBC+Driver+17+for+SQL+Server"
            )

            # Tạo engine kết nối
            engine = create_engine(connection_string, fast_executemany=True)

            # Đẩy dữ liệu vào SQL Server
            dataframe.to_sql(
                name=table_name,
                con=engine,
                if_exists=if_exists,
                index=False
            )
            print(f"Đã đẩy dữ liệu thành công vào bảng {table_name} trong database {database_name}.")
        except Exception as e:
            print(f"Lỗi khi đẩy dữ liệu vào SQL Server: {e}")
    push_to_sqlserver(df, table_name = 'Dim_CrawlDate', if_exists='replace')
```

Figure 27: Cleaned to OLAP dataset Function for Crawl Date Dimension Table

```python
def  task_fact_sale():
    query = '''SELECT distinct [Product_id],
      b.shop_id
      ,[Quantity_Sold]
      ,[Crawl_Date]
  FROM [CleanedData].[dbo].[Sale_clean] a
  left join OLAP.dbo.Dim_shop b on a.seller_url = b.shop_url
    '''
    df = read_from_sqlserver(query)
    df['shop_id'] = range(1, len(df) + 1)
    def push_to_sqlserver(dataframe, table_name, if_exists='replace'):
        server_name = '192.168.110.41'  # Thay bằng tên server của bạn
        database_name = 'OLAP'            # Thay bằng tên database của bạn
        username = "data"
        password = "data123"
        try:
            # Chuỗi kết nối SQL Server
            connection_string = (
                f"mssql+pyodbc://{username}:{password}@{server_name}/{database_name}"
                f"?driver=ODBC+Driver+17+for+SQL+Server"
            )

            # Tạo engine kết nối
            engine = create_engine(connection_string, fast_executemany=True)

            # Đẩy dữ liệu vào SQL Server
            dataframe.to_sql(
                name=table_name,
                con=engine,
                if_exists=if_exists,
                index=False
            )
            print(f"Đã đẩy dữ liệu thành công vào bảng {table_name} trong database {database_name}.")
        except Exception as e:
            print(f"Lỗi khi đẩy dữ liệu vào SQL Server: {e}")
    push_to_sqlserver(df, table_name = 'Fact_sale', if_exists='replace')
```

Figure 28: Cleaned to OLAP dataset Function for Fact Sale Table

```python
def  task_fact_comment():
    query = '''SELECT distinct [Product_id]
      ,b.user_id
      ,[Title]
      ,[Comment_Date]
      ,[Comment]
      ,[Rating]
  FROM [CleanedData].[dbo].[Comment_clean] a
  left join OLAP.dbo.Dim_user b on a.[user] =  b.[user]
'''
    df = read_from_sqlserver(query)
    df['shop_id'] = range(1, len(df) + 1)
    def push_to_sqlserver(dataframe, table_name, if_exists='replace'):
        server_name = '192.168.110.41'  # Thay bằng tên server của bạn
        database_name = 'OLAP'            # Thay bằng tên database của bạn
        username = "data"
        password = "data123"
        try:
            # Chuỗi kết nối SQL Server
            connection_string = (
                f"mssql+pyodbc://{username}:{password}@{server_name}/{database_name}"
                f"?driver=ODBC+Driver+17+for+SQL+Server"
            )

            # Tạo engine kết nối
            engine = create_engine(connection_string, fast_executemany=True)

            # Đẩy dữ liệu vào SQL Server
            dataframe.to_sql(
                name=table_name,
                con=engine,
                if_exists=if_exists,
                index=False,
                dtype={
                    'Title': NVARCHAR(None),
                    'Comment_Date': NVARCHAR(None),
                    'Comment' :NVARCHAR(None),
                }
            )
            print(f"Đã đẩy dữ liệu thành công vào bảng {table_name} trong database {database_name}.")
        except Exception as e:
            print(f"Lỗi khi đẩy dữ liệu vào SQL Server: {e}")
    push_to_sqlserver(df, table_name = 'Fact_comment', if_exists='replace')
```

Figure 29: Cleaned to OLAP dataset Function for Fact Comment Table

2.3.5 Combined with Airflow Apache & Running DAGs file

a. Combined with Airflow Apache

We integrate all the above code by defining a DAG and a PythonOperator to execute tasks. Then, we arrange the order of tasks to run.

```
dag = DAG(
    'test_bigquery_dag_full',
    default_args={
        'retries': 2,
        'retry_delay': timedelta(minutes=0.5),
        'start_date': days_ago(1)
    },
    schedule_interval=None,
    catchup=False
)
```

Figure 30: Defining the DAG

```
Shop_clean_and_push_to_SQL = PythonOperator(
    task_id='Shop_clean_and_push_to_SQL',
    python_callable=task_shop_clean,
    dag=dag
)

Sale_clean_and_push_to_SQL = PythonOperator(
    task_id='Sale_clean_and_push_to_SQL',
    python_callable=task_sale_clean,
    dag=dag
)


Comment_clean_and_push_to_SQL = PythonOperator(
    task_id='Comment_clean_and_push_to_SQL',
    python_callable=task_comment_clean,
    dag=dag
)


get_product_clean = PythonOperator(
    task_id='Product_clean_and_push_to_SQL' ,
    python_callable=task_product_clean,
    dag=dag
)


Dim_warehouse = PythonOperator(
    task_id='Dim_warehouse',
    python_callable=task_dim_warehouse,
    dag=dag
)
```

```
Dim_crawldate = PythonOperator(
    task_id='Dim_CrawlDate',
    python_callable=task_dim_crawl_date,
    dag=dag
)

Dim_Products = PythonOperator(
    task_id='Dim_Products',
    python_callable=task_dim_product,
    dag=dag
)

Dim_Shops  = PythonOperator(
    task_id='Dim_Shops',
    python_callable=task_dim_shop,
    dag=dag
)

Dim_Users = PythonOperator(
    task_id='Dim_Users',
    python_callable=task_dim_user,
    dag=dag
)

Fact_Comments = PythonOperator(
    task_id='Fact_Comments',
    python_callable=task_fact_comment,
    dag=dag
)


Fact_Sales = PythonOperator(
    task_id='Fact_Sales',
    python_callable=task_fact_sale,
    dag=dag
)
```

Figure 31: Defining Python Operator

```
Shop_clean_and_push_to_SQL >> Dim_Shops
get_product_clean >> Dim_warehouse >> Dim_Products
Comment_clean_and_push_to_SQL >> Dim_Users
Sale_clean_and_push_to_SQL >> Dim_crawldate


dim_tasks_sale = [Dim_Shops, Dim_Products, Dim_Users, Dim_crawldate]
dim_tasks_comment = [Dim_Shops, Dim_Products, Dim_Users,Sale_clean_and_push_to_SQL]

dim_tasks_sale >> Fact_Sales
dim_tasks_comment >>Fact_Comments
```

Figure 32: The order of the tasks

a. Running DAGs file

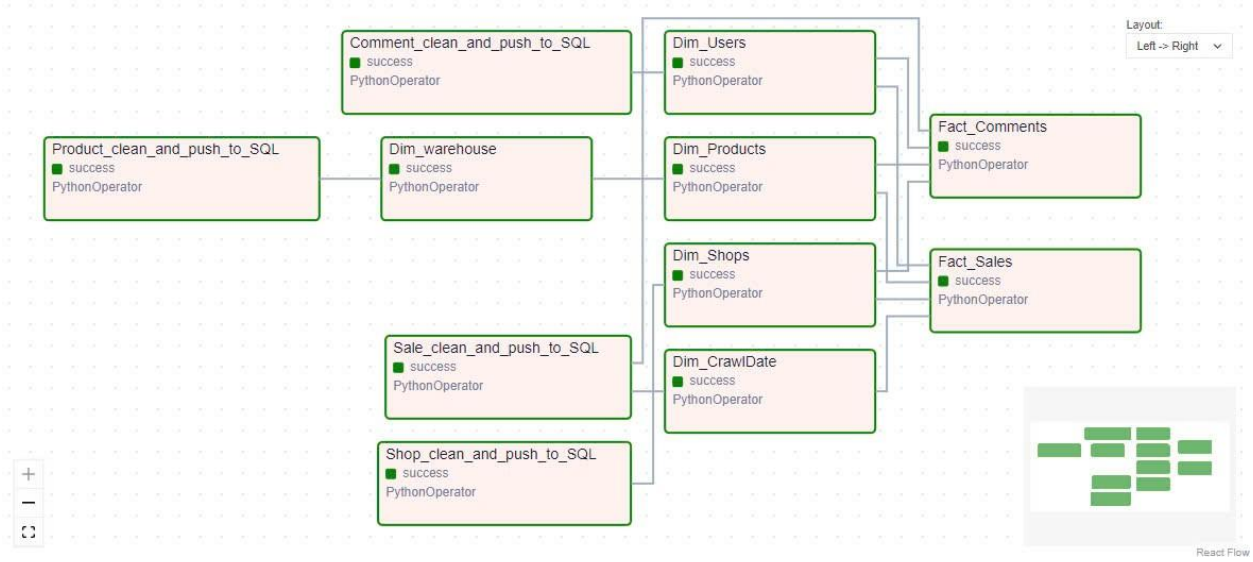

Figure 33: DAGs file flow

III. Building Data Warehouse

3.1 Business Process

In today's fast-paced and highly competitive electronics market, a deep understanding of sales performance is the cornerstone of sustained success. At the outset of this data warehouse project, our primary focus was directed toward achieving the key objective: Sales Performance Analysis Process. The Sales Performance Analysis Process for the electronic store seeks to uncover critical insights by thoroughly evaluating sales trends, product performance, and revenue patterns across categories and time periods. By leveraging transactional data, this process identifies high-demand products, seasonal peaks, and underperforming inventory, while also assessing key factors such as pricing strategies, promotional campaigns, and customer buying behavior. These valuable insights will enable the store to optimize inventory management, craft data-driven marketing strategies, and fine-tune pricing models to boost profitability. Ultimately, this process will empower the store to make informed, strategic decisions that drive long-term growth and operational excellence.

3.1.1 Sale Performance Analysis Process

This section will focus on answering 2 main questions:

- How has sales performance changed in the second half of December 2024 until now (January 5, 2025) in the electronics sector on the TIKI e-commerce platform?
- How has sales performance changed by Shop in the electronics sector on the TIKI e-commerce platform?

We examined sales performance data from the TIKI e-commerce platform during this timeframe, particularly focusing on the electronics sector. The analysis includes identifying key trends, highlighting best-performing products, and uncovering factors contributing to changes in sales performance.

Our findings offer a comprehensive view of the sales dynamics within the electronics sector, shedding light on opportunities to optimize inventory, refine marketing strategies, and enhance customer engagement. Additionally, the study evaluates the impact of seasonal events and promotional campaigns on consumer behavior, uncovering patterns that influenced buying decisions.

Based on these insights, we propose actionable recommendations to capitalize on growth opportunities, address existing challenges, and drive sustained improvements in sales performance for the electronics sector on the TIKI platform.

3.2 Dimension Modeling

In this project, we adopt dimension modeling using the Star Schema to effectively organize and analyze sales and performance data in the electronics sector on the TIKI e-commerce platform. This method structures the data into fact and dimension tables, making high-performance analytics more intuitive. By classifying data into facts (quantitative measures) and dimensions (contextual details), the model streamlines querying and reporting, while accommodating both detailed and aggregated analysis. The chosen transaction grain reflects transactional data, enabling in-depth insights and trend analysis. This strategy allows us to assess sales patterns, evaluate shop performance, and understand customer behaviors, offering the flexibility to identify opportunities, enhance strategies, and tackle challenges in the fast-paced e-commerce landscape.

3.2.1 Definition of dimension

Warehouse Dimension

- **Warehouse:** Warehouse's name
- **w_id:** ID number of warehouse - Unique identifier for the warehouse

```
query = '''select distinct
  [warehouse]
FROM [CleanedData].[dbo].[Product_clean]
'''
df = read_from_sqlserver(query)
df['w_id'] = range(1, len(df) + 1)
```

Figure 34: Create Dim_Warehouse

Product Dimension

- **Name:** Product's name
- **w_id: ID** number of warehouse - Unique identifier for the warehouse

```
query = '''SELECT a.[Id]
  ,a.[name]
  ,b.w_id
FROM [CleanedData].[dbo].[Product_clean] a
left join OLAP.dbo.Dim_warehouse b on a.warehouse = b.warehouse
'''
```

Figure 35: Create Dim_Product

User Dimension

- **User:** User's name
- **Time_on_Platform:** Time using platform

```
query = '''SELECT distinct
   [User]
   ,[Time_on_Platform]
FROM [CleanedData].[dbo].[Comment_clean]
'''
```

Figure 36: Create Dim_User

Shop Dimension

- **Shop_Name:** Shop's name
- **Shop_URL:** Link to shop
- **Review_Count:** Number of reviews of customers

```
query = '''SELECT distinct
   [Shop_Name]
   ,[Shop_URL]
   ,[Review_Count]
FROM [CleanedData].[dbo].[Shop_clean]
'''
```

Figure 37: Create Dim_Shop

Crawl date Dimension

- **DateKey**: The crawl date in the format `mm/dd/yyyy`
- **Year**: The year of the crawl
- **Month**: The month of the crawl

- **Day**: The day of the crawl
- **DayName**: The name of the day (e.g., Monday, Tuesday)
- **Quarter**: The quarter of the year in which the crawl occurred
- **Week**: The week of the year in which the crawl occurred

```
query = '''
SELECT DISTINCT
    CAST(Crawl_Date AS DATE) AS DateKey,
    YEAR(CAST(Crawl_Date AS DATE)) AS Year,
    MONTH(CAST(Crawl_Date AS DATE)) AS Month,
    DAY(CAST(Crawl_Date AS DATE)) AS Day,
    DATENAME(WEEKDAY, CAST(Crawl_Date AS DATE)) AS DayName,
    DATEPART(QUARTER, CAST(Crawl_Date AS DATE)) AS Quarter,
    DATEPART(WEEK, CAST(Crawl_Date AS DATE)) AS Week,
    CASE
        WHEN DATEPART(WEEKDAY, CAST(Crawl_Date AS DATE)) IN (1, 7) THEN 'Weekend'
        ELSE 'Weekday'
    END AS IsWeekend
FROM CleanedData.dbo.Sale_clean
ORDER BY DateKey;
'''
```

Figure 38: Create Dim_CrawlDated

Comment date Dimension

- **Comment_Date:** The time the customer reviewed product
- **Time_ago:** The time is extracted from Comment_Date
- **Unit_time_ago:** Units of time
- **CommentDATE:** The time the customer reviewed product has been modified (mm/dd/yyyy)
- **Day:** The day the customer comment
- **DayName:** Name of the day
- **Month:** The month the customer comment
- **MonthName:** Name of the month
- **Quarter:** The quarter
- **Year:** The year the customer comment

To create this dimension table, we have decided not to generate it automatically through queries using DAGs. The reason is that the crawled data is of poor quality and the processing is quite complex. Below is the data from the ‘Comment_Date’ column:

Comment	Comment_Date	Product_id	Rating	Title	user_id	shop_id
Bring form no pay spend change talk wind.	Đánh giá vào 3 tháng trướcĐã dùng 0 ngày	2166	5	Rất hài lòng	225	10
Head remain major so occur true history.	Đánh giá vào 3 tháng trướcĐã dùng 0 ngày	2555	5	Rất hài lòng	325	10
House sound wind official year rock.	Đánh giá vào 4 năm trướcĐã dùng 2 tháng	2964	5	Rất hài lòng	30	10
Become sister raise long fall.	Đánh giá vào 4 năm trướcĐã dùng 10 tháng	2767	5	Rất hài lòng	219	10
Probably stop young dark expert society good.	Đánh giá vào 1 năm trướcĐã dùng 10 tháng	831	5	Rất hài lòng	388	10
Deal report across reason.	Đánh giá vào 2 năm trướcĐã dùng 0 tháng	2024	5	Rất hài lòng	2	10
Computer way rich theory south article make.	Đánh giá vào 1 năm trướcĐã dùng 9 tháng	2623	5	Rất hài lòng	109	10
Foreign provide how entire.	Đánh giá vào 4 năm trướcĐã dùng 3 tháng	1257	5	Rất hài lòng	349	10
Coach success conference.	Đánh giá vào 3 năm trướcĐã dùng 9 tháng	1567	5	Rất hài lòng	233	10
Very officer hard loss family include major different.	Đánh giá vào 4 năm trướcĐã dùng 0 tháng	2842	5	Rất hài lòng	38	10
Market follow civil.	Đánh giá vào 2 năm trướcĐã dùng 11 tháng	1093	5	Rất hài lòng	227	10
Strong others east no present support room central.	Đánh giá vào 1 năm trướcĐã dùng 0 tháng	1070	5	Rất hài lòng	191	10

Figure 39: Data type of Comment_Date column

We have decided to use Power Query in Power BI to process and create the 'Dim_CommentDate' table with the following steps:

- Use the ‘Comment_Date’ column as the foreign key.
- Split the data

For example, transforming ‘Reviewed 3 months ago Used 0 days ago’ into two columns: ‘Time_ago’ with the value of 3 and ‘unit_time_ago’ with the value of "month".

Comment_Date	Time_ago	unit_time_ago
Valid 100% Error 0% Empty 0%	Valid 100% Error 0% Empty 0%	Valid 100% Error 0% Empty 0%
54 distinct, 54 unique	6 distinct, 2 unique	2 distinct, 0 unique
Đánh giá vào 3 năm trướcĐã dùng 0 tháng	3	Năm
Đánh giá vào 3 tháng trướcĐã dùng 0 ngày	3	Tháng
Đánh giá vào 3 năm trướcĐã dùng 5 tháng	3	Năm
Đánh giá vào 2 năm trướcĐã dùng 11 tháng	2	Năm
Đánh giá vào 4 năm trướcĐã dùng 2 tháng	4	Năm
Đánh giá vào 2 năm trướcĐã dùng 8 tháng	2	Năm

Figure 40: Create Time_ago and unit_time_ago columns

Next, create the 'CommentDATE' column by using the current time minus the value in the 'Time_ago' column, with the unit from the 'unit_time_ago' column.

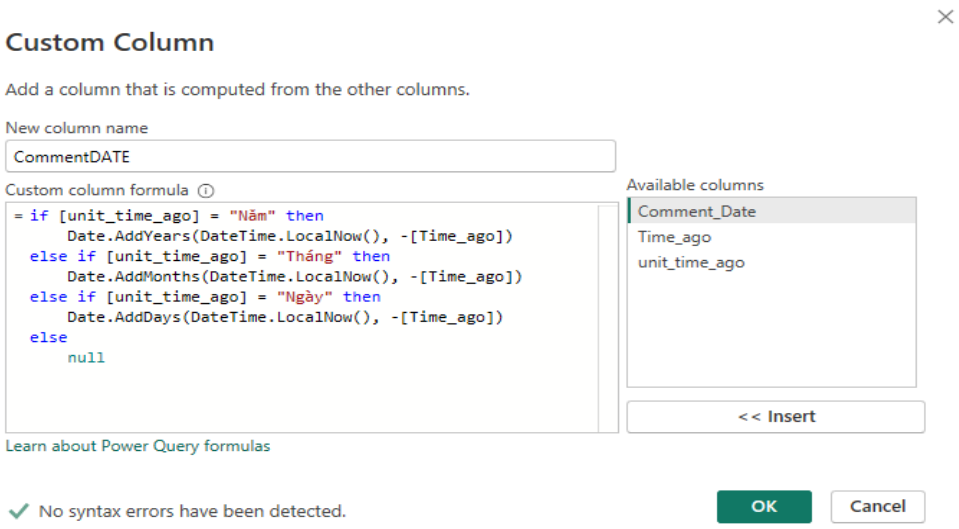

Figure 41: Create CommentDate colums

Finally, create columns such as Day, DayName, ….

CommentDATE	Month	Day	DayName	MonthName	Year	Quarter
Valid 100% Error 0% Empty 0% 10 distinct, 6 unique	Valid 100% Error 0% Empty 0%	Valid 100% Error 0% Empty 0%	Valid 100% Error 0% Empty 0%	Valid 100% Error 0% Empty 0%	Valid 100% Error 0% Empty 0%	Valid 100% Error 0% Empty 0%
1/8/2022	1	8	Sat	Jan	2022	Q1
10/8/2024	10	8	Tue	Oct	2024	Q4
1/8/2022	1	8	Sat	Jan	2022	Q1
1/8/2023	1	8	Sun	Jan	2023	Q1
1/8/2021	1	8	Fri	Jan	2021	Q1
1/8/2023	1	8	Sun	Jan	2023	Q1
1/8/2024	1	8	Mon	Jan	2024	Q1
1/8/2023	1	8	Sun	Jan	2023	Q1
1/8/2024	1	8	Mon	Jan	2024	Q1
1/8/2022	1	8	Sat	Jan	2022	Q1
1/8/2023	1	8	Sun	Jan	2023	Q1
9/8/2024	9	8	Sun	Sep	2024	Q3
1/8/2022	1	8	Sat	Jan	2022	Q1
12/8/2024	12	8	Sun	Dec	2024	Q4
11/8/2024	11	8	Fri	Nov	2024	Q4
1/8/2022	1	8	Sat	Jan	2022	Q1

Figure 42: Full Dim_CommentDate

3.2.2 Definition of fact

Fact Sale

- **Product_id:** Unique identifier for the product
- **Shop_id:** Unique identifier for the shop
- **Quantity_Sold:** Quantity of goods sold
- **Crawl_date:** Crawl data

```
query = '''SELECT distinct [Product_id],
  b.shop_id
  ,[Quantity_Sold]
  ,[Crawl_Date]
FROM [sale].[dbo].[sales_clean] a
left join mart.dbo.dim_shop b on a.seller_url = b.shop_url
'''
```

Figure 43: Create Fact_Sale

Fact Comment

- **Product_id:** Unique identifier for the product
- **User_id:** Unique identifier for the user
- **Comment_date:** Customer comment date
- **Comment:** Customer feedback content
- **Rating:** Customer rating

```
query = '''SELECT distinct [Product_id]
  ,b.user_id
  ,[Title]
  ,[Comment_Date]
  ,[Comment]
  ,[Rating]
FROM [sale].[dbo].[commet_clean] a
left join mart.dbo.dim_user b on a.[user] =  b.[user]
'''
```

Figure 44: Create Fact_Comment

3.2.3 Star Schema

When connecting to PowerBI, we decided to add an additional simple data transformation step to create new columns based on existing ones in each dimension table to make analysis easier.

Below is the final Star Schema after all the data transformation steps. The Fact_Comment and Fact_Sale tables are at the center. Dimension tables such as Dim_User, Dim_Shop, Dim_Product, Dim_CommentDate, Dim_CrawlDate, etc., are directly connected to the Fact tables.

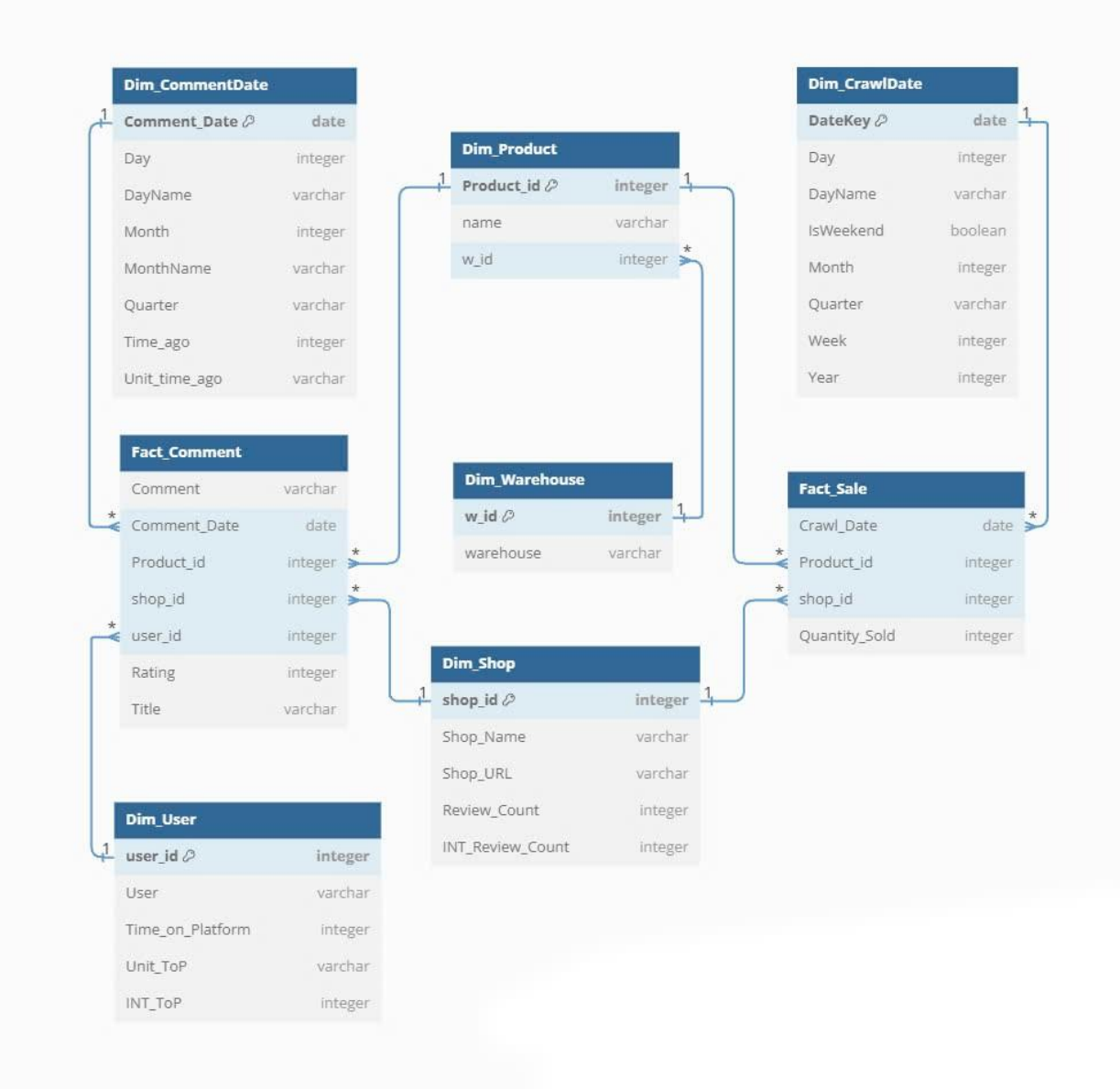

Figure 45: Final Star Schema

IV. Business Analytic & Results

Problem Background

During the period from late December 2024 to early January 2025, the electronics industry on the TIKI e-commerce platform witnessed remarkable growth, mainly due to consumer demand during the holiday season (Christmas and New Year). However, this growth raises many questions about how to optimize business performance, service quality, and customer satisfaction. Factors such as

sales, supply chain management (warehouse), and customer feedback are aspects that need to be carefully analyzed to better understand operational efficiency and growth potential.

a. Target Audience

- TIKI Management Board and Executives

Objective: Understand sales performance over time, warehouse efficiency, and customer satisfaction. They need these analytics to plan business strategies and operations during peak periods.

- Shop Owners/Managers

Objective: Understand best-selling products, service strengths/weaknesses, and how other stores on TIKI are performing. They need to use this information to adjust product portfolio, improve customer service, and increase business performance.

b. Analytic & Results

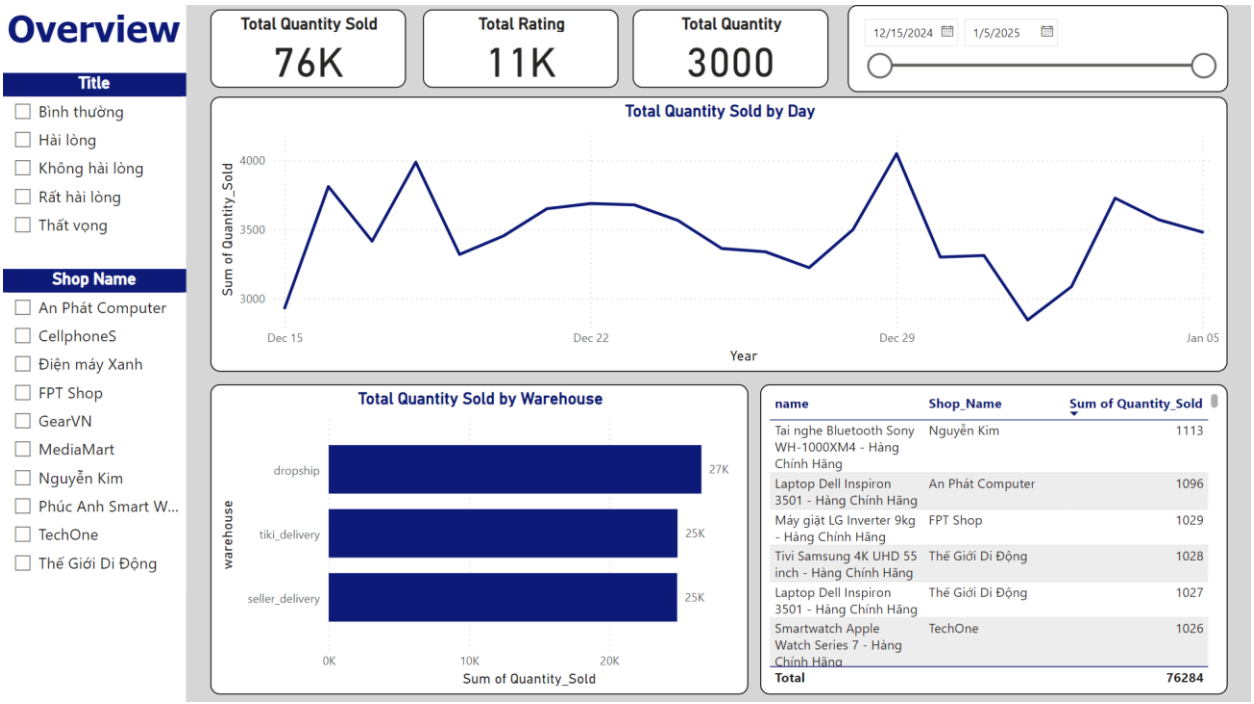

Figure 46: Overview Business Analytics

Question 1:

How has sales performance changed in the second half of December 2024 until now (January 5, 2025) in the electronics sector on the TIKI e-commerce platform?

A total of 76,284 units of electronics-related products were sold on the TIKI platform between December 15, 2024, and January 5, 2025. Sales tend to increase sharply from mid-December, peaking around December 28-29, with more than 4,000 products sold in a day. After January 1, 2025, sales decreased slightly, stabilizing at around 3,000 products per day. The increase in sales at the end of December coincided with the year-end shopping season (Christmas and New Year), reflecting high consumer demand during the holiday season. The decrease in sales after January 1 may be due to shopping demand settling down after the peak period, which is in line with the general market trend.

Dropship warehouse has the highest sales with 27,000 products, followed by Tiki_delivery (25,000 products) and Seller_delivery (24,000 products). Dropship warehouses dominate thanks to their flexible processing capabilities, minimizing delivery times. Stores using the dropship model are taking advantage of logistics optimization, reduced storage costs and faster delivery. The balanced sales between the three types of warehouses show that TIKI is implementing an efficient operating system that suits the needs of both retailers and customers.

Sony WH-1000XM4 Bluetooth headphones at Nguyen Kim led with 1,113 products sold. Dell Inspiron 3501 laptops at An Phat Computer reached sales of 1,096 products. LG Inverter 9kg washing machines at FPT Shop and Samsung 4K UHD TVs at The Gioi Di Dong both reached more than 1,000 products. High-tech products (headphones, laptops) and home appliances (washing machines, TVs) were both popular during this period, reflecting the diverse needs of customers. High sales at large stores such as Nguyen Kim and An Phat Computer show that they are optimizing the selection of best-selling products to meet market demand.

With 11,000 evaluations overall, there is a high degree of satisfied consumer interaction. A favorable experience is indicated by the preponderance of "Very satisfied" and "Satisfied" evaluations. Best-selling items frequently get a lot of evaluations. For instance, Sony headphones at Nguyen Kim sell well and get a lot of positive feedback. Delivery and after-sales services are just as important to customer satisfaction as the goods themselves. This highlights how crucial it is to uphold service quality in order to increase client confidence.

The busiest shopping season occurs in December, and strong sales necessitate effective warehouse operations. Dropship warehouses are crucial to supplying this spike in demand because of their quick delivery capabilities. Technological items like computers and headphones frequently sell well and get good ratings. This demonstrates that consumers are happy with the high caliber of goods and prompt delivery from warehouses. Businesses like Nguyen Kim, which sells the most Sony headphones, have been successful in streamlining their product line. Product variety has also helped other retailers, such as FPT Shop and The Gioi Di Dong, keep their competitive edge.

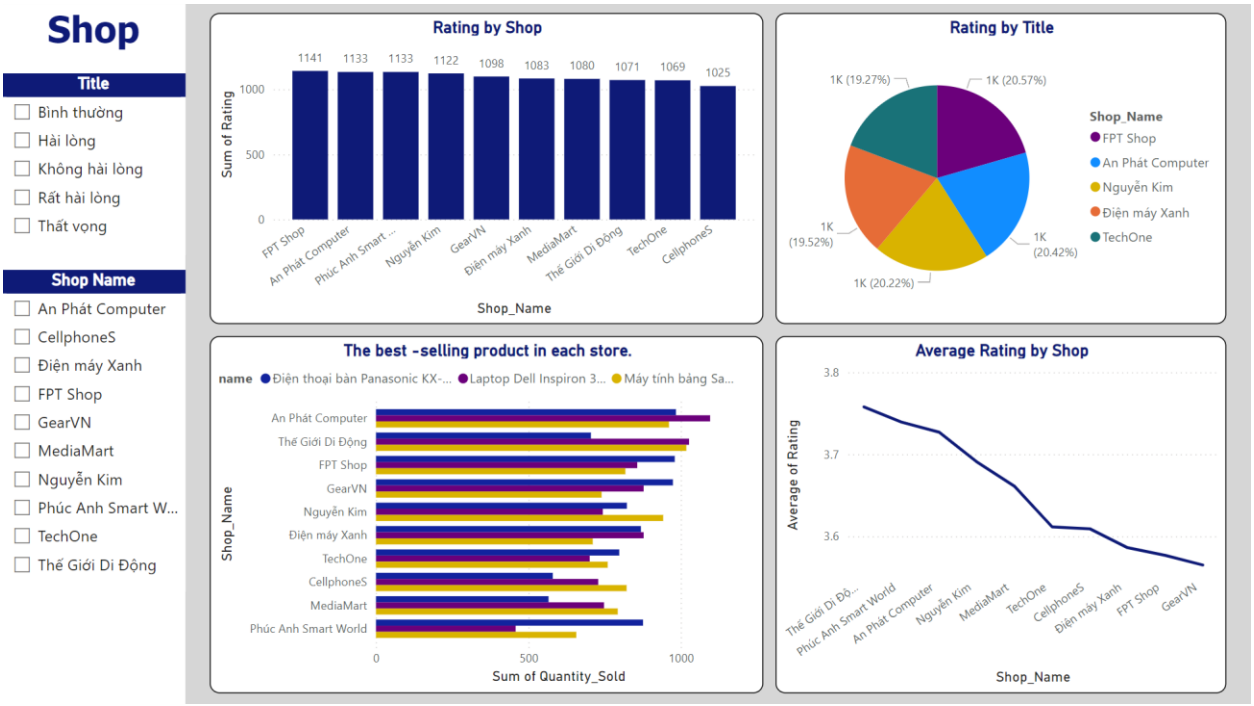

Figure 47: Shop Analytics

Question 2:

How has sales performance changed by Shop in the electronics sector on the TIKI e-commerce platform?

Although FPT Shop has the most reviews overall (1,141), its average rating is in the bottom range (3.6). This demonstrates that despite the store's high transaction volume and numerous customer reviews, the level of service or shopping experience may fall short of what customers anticipate. However, The Gioi Di Dong gets the highest average rating (3.8) while having the fewest reviews (1,080). This demonstrates how much their clients value their emphasis on providing high-quality services and products. A large quantity of reviews does not necessarily indicate a high degree of satisfaction. Instead of raising the quantity of reviews, retailers such as FPT Shop should concentrate on enhancing their customer service in order to raise the average score.

Best-selling Dell Inspiron 3501 laptops are sold at An Phat Computer and Phuc Anh Smart World, which have a high average rating (3.7). Although the Samsung Galaxy Tab A tablet is well-liked at Nguyen Kim, the store's average rating is only 3.65. This indicates that even when the product sells well, problems with service or after-sales care may still exist and impact the rating. Rating performance is significantly impacted by best-selling products. While service quality is a determining element for customer happiness, stores that provide products that are easy to suit customer needs typically have higher review scores.

At 20.57% and 20.42%, respectively, FPT Shop and An Phat Computer had high "Very satisfied" ratings. However, compared to the larger stores, TechOne and CellphoneS had higher unfavorable ratings while having low ratings (1,069 and 1,025). Despite not having the most reviews, Dien May Xanh maintains a steady level of customer happiness, as evidenced by its "Very Satisfied" (19.52%) and "Satisfied" (20.22%) ratings. While smaller stores might concentrate on particular client categories with more extreme replies, stores with a large number of reviews typically have a wide range of satisfaction levels (both positive and negative).

Even if they sell a number of items, such as the Samsung Galaxy Tab A Tablet, stores like FPT Shop have a poor average score. This can be because there are more problems with service or the shopping experience as a result of the high volume of transactions. The Gioi Di Dong, on the other hand, maintains a high average score by optimizing the quality of its products and services, despite having fewer reviews and concentrating on items like the Panasonic KX Desk Phone. Both product and service quality have an impact on sales performance. To guarantee customer happiness, a top-selling product must be backed by excellent customer service.

VI. Conclusion & Recommendation

1. Conclusion

The project has successfully built an automated data processing system, from data collection, cleaning, to data analysis and visualization, to effectively address important business questions. Data analysis has highlighted important trends, including the strong growth in electronic product sales during the year-end holiday season, the superiority of the Dropship warehouse model thanks to its flexibility and delivery speed, as well as the outstanding performance of large stores such as Nguyen Kim and An Phat thanks to product category optimization. At the same time, the analysis results also pointed out some limitations, especially in stores with a large volume of reviews but low average scores, due to issues related to the quality of after-sales service. The implemented data system not only provides deep insights but also provides a solid foundation for businesses to make strategic decisions, contributing to improving operational efficiency, increasing customer satisfaction and promoting sustainable development.

2. Recommendation

a. Business

Improve customer service quality:

- Analyze negative customer feedback in detail to identify common causes and then propose specific solutions.

- Train staff in customer care skills, especially focusing on stores with low average scores like FPT Shop.
- Enhance after-sales policies, including improving response times, return policies and technical support.

Optimize product portfolio:

- Focus on developing best-selling product lines such as Bluetooth headsets, laptops and home appliances.
- Research the needs of other customer segments to add new products to diversify the portfolio and increase competitiveness.
- Maintain flexible pricing strategies and promotions appropriate to each high demand period.

Improve warehouse management efficiency:

- Invest more in the Dropship model to maximize fast delivery and reduce inventory costs.
- Analyze historical data to forecast demand more accurately, helping to avoid shortages or surpluses, especially during peak seasons.

Building a data-driven marketing strategy:

- Segment customers in more detail to create personalized marketing campaigns to increase conversion rates and retain loyal customers.
- Leverage data from customer reviews to improve ad copy and increase brand trust.

b. Systemically

Improve data collection and processing systems:

- Improved integration between tools like Apache NiFi, Apache Airflow, and BigQuery to optimize the speed and accuracy of data collection.
- Automate more steps in the data processing workflow, especially data cleaning and normalization, to minimize manual errors.

Expand your data analysis capabilities:

- Develop more detailed reports on Power BI, including sales forecasts and market trend analysis.
- Integrate machine learning algorithms to make detailed predictions about customer behavior, marketing effectiveness, and warehouse operations optimization.

Improve demand forecasting capabilities:

- Apply predictive models based on historical data to optimize inventory planning and allocation.
- Analyze seasonality and market trends to prepare in time for peak periods.

Enhanced data security:

- Ensure security policies are enforced across the entire system, especially customer and transaction data.
- Regularly update security systems to avoid potential external threats.

Optimize infrastructure system:

- Upgrade your BigQuery and SQL Server configurations to meet growing analytics needs.
- Use performance monitoring tools to ensure smooth system operation, especially during peak seasons.

VII. Reference

[1] airflow.apache.org. (n.d.). *Running Airflow in Docker — Airflow Documentation*. [online] Available at: https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html.

[2] Doan, N. (2024). *Crawl data đầy đủ hơn với thư viện selenium*. [online] Viblo. Available at: https://viblo.asia/p/crawl-data-day-du-hon-voi-thu-vien-selenium-EbNVQ5GmVvR [Accessed 7 Jan. 2025].

[3] nifi.apache.org. (n.d.). *Apache NiFi Overview*. [online] Available at: https://nifi.apache.org/docs/nifi-docs/html/overview.html.