




Nguyễn Ngọc Tâm

Báo Cáo.docx

-  Quét trùng lặp chống đạo văn UTH 05
-  Quét trùng lặp UTH (Moodle PP)
-  Ho Chi Minh City University of Transport

Document Details

Submission ID

trn:oid::1:3361947270

Submission Date

Oct 5, 2025, 5:26 PM GMT+7

Download Date

Oct 5, 2025, 10:13 PM GMT+7

File Name

5628_Nguyễn_Ngọc_Tâm_Báo_Cáo_407562_1076302875.docx

File Size

569.8 KB

34 Pages

12,724 Words

51,049 Characters




10% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.




Filtered from the Report

- Bibliography
- Quoted Text

Top Sources

- 8%  Internet sources
 - 9%  Publications
 - 1%  Submitted works (Student Papers)
-

Top Sources

8%	 Internet sources
9%	 Publications
1%	 Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	www.ctu.edu.vn	4%
2	Publication	Phenikaa University	3%
3	Publication	Ton Duc Thang University	<1%
4	Internet	azsolutions.vn	<1%
5	Student papers	National Economics University	<1%
6	Publication	Banking Academy	<1%
7	Internet	vinateks.vn	<1%

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI THÀNH PHỐ HỒ CHÍ MINH



KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

THỰC TẬP TỐT NGHIỆP

**Đề tài: Thiết kế và Triển khai Hệ thống Data Warehouse
cho Bán hàng Đa Nền tảng**

Giảng viên hướng dẫn: Nguyễn Văn Chiến

Sinh viên thực hiện: Nguyễn Ngọc Tâm

MSSV: 2151050051

Lớp: KM21

Thành phố Hồ Chí Minh – 2025

Mục Lục

I. LỜI MỞ ĐẦU	2
1.1. Bối cảnh và lý do chọn đề tài	2
1.2. Mục tiêu nghiên cứu	2
1.3. Đối tượng và phạm vi nghiên cứu	3
1.4. Phương pháp nghiên cứu	4
II. Cơ sở lý thuyết và tổng quan công nghệ	4
2.1. Tổng quan về dữ liệu bán hàng trong thương mại điện tử	4
2.2. Các khái niệm về xử lý và trực quan hóa dữ liệu	5
2.2.1. So sánh giữa ETL và ELT	5
2.2.2. Data Pipeline và trực quan hóa dữ liệu	9
III. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG	18
3.1. Phân tích yêu cầu hệ thống	18
3.1.1. Khảo sát và xác định nguồn dữ liệu	18
3.1.2. Phân tích dữ liệu từ các sản phẩm thương mại điện tử	18
3.1.3. Phân tích dữ liệu từ các hệ thống nội bộ	18
3.1.4. Xác định các chỉ số kinh doanh (KPI) và dashboard	18
3.2. Thiết kế kiến trúc hệ thống	19
3.3. Thiết kế cơ sở dữ liệu/kho dữ liệu	21
3.3.1. Mô hình dữ liệu	21
3.3.2. Chiến lược lưu trữ dữ liệu	22
IV. Triển khai và cài đặt hệ thống	24
4.1. Môi trường phát triển và công cụ	24
4.2. Xây dựng Data Pipeline	25
4.2.1. Giai đoạn Extract	25
4.2.2. Giai đoạn Load	25
4.2.3. Giai đoạn Transform	26
4.3. Xây dựng Dashboard trực quan hóa	28
4.3.1. Kế hoạch thiết kế và phát triển dashboard	28
4.4. Triển khai API mô phỏng với FastAPI	29
4.5. Containerization và triển khai với Docker	30
4.5.1. Cấu trúc Docker	30
4.5.2. Docker Compose	30
4.5.3. Lợi ích của containerization	30
4.6. Xử lý lỗi và giám sát hệ thống	30
4.6.1. Cơ chế xử lý lỗi trong Data Pipeline	31
4.6.2. Giám sát hệ thống	31
V. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	31
5.1 Kết luận	31
5.2 Hướng phát triển	32
Tài liệu tham khảo	33

I. LỜI MỞ ĐẦU

1.1. Bối cảnh và lý do chọn đề tài

Sự phát triển vượt bậc của thương mại điện tử đã biến đổi sâu sắc cách thức các doanh nghiệp vận hành. Đối với các doanh nghiệp vừa và nhỏ kinh doanh sản phẩm công nghệ, việc này mang lại cơ hội lớn nhưng cũng đi kèm với nhiều thách thức. Dữ liệu bán hàng từ các sàn thương mại điện tử lớn như Shopee, Lazada, Tiki, cùng với các kênh bán hàng trực tuyến khác, tạo ra một dòng chảy dữ liệu khổng lồ và liên tục. Tuy nhiên, dữ liệu này thường nằm rải rác, không được tổ chức đồng nhất và thiếu một nền tảng chung để tổng hợp. Các doanh nghiệp này thường phải đối mặt với tình trạng dữ liệu bán hàng chưa được cập nhật theo thời gian thực, dẫn đến việc ra quyết định bị chậm trễ và thiếu chính xác.

Thực trạng này đặt ra một thách thức lớn: làm thế nào để tích hợp, xử lý và phân tích khối lượng dữ liệu đa dạng này một cách hiệu quả để hỗ trợ ra quyết định kinh doanh chiến lược? Việc xử lý dữ liệu thủ công không chỉ tốn kém về thời gian và nhân lực mà còn tiềm ẩn nhiều rủi ro về sai sót, ảnh hưởng trực tiếp đến khả năng kiểm soát tồn kho, tối ưu hóa chuỗi cung ứng và đánh giá hiệu quả kinh doanh tổng thể.

Để giải quyết triệt để vấn đề này, việc xây dựng một kiến trúc Data Warehouse hiện đại trở thành một giải pháp tối ưu. Kiến trúc này cho phép hợp nhất dữ liệu từ nhiều nguồn khác nhau vào một kho dữ liệu tập trung, đảm bảo tính toàn vẹn, nhất quán và khả năng mở rộng. Từ đó, doanh nghiệp có thể thực hiện các phân tích chuyên sâu, tạo ra các báo cáo và dashboard trực quan một cách dễ dàng, giúp đội ngũ quản lý có cái nhìn toàn diện và sâu sắc về hiệu suất kinh doanh, từ đó đưa ra các quyết định chiến lược kịp thời.

Với mong muốn đóng góp một giải pháp thực tiễn, có khả năng ứng dụng cao cho các doanh nghiệp thương mại điện tử, em quyết định thực hiện đề tài luận văn: "Xây dựng hệ thống xử lý và trực quan hóa dữ liệu bán hàng cho doanh nghiệp thương mại điện tử dựa trên kiến trúc Data Warehouse".

1.2. Mục tiêu nghiên cứu

Mục tiêu trọng tâm của luận văn là xây dựng một nền tảng dữ liệu (Data Platform) hoàn chỉnh và tự động, có khả năng mở rộng, nhằm giải quyết bài toán tích hợp và phân tích dữ liệu bán hàng từ nhiều nguồn khác nhau. Hệ thống này sẽ là nền tảng vững chắc giúp doanh nghiệp chuyển đổi từ việc xử lý dữ liệu thủ công, rời rạc sang một quy trình tự động, tập trung và dựa trên dữ liệu.

Các mục tiêu cụ thể bao gồm:

Phân tích và Thiết kế Kiến trúc Hệ thống:

Phân tích sâu các yêu cầu nghiệp vụ và đặc điểm dữ liệu từ các nguồn đa dạng, bao gồm API của các sàn thương mại điện tử (Shopee, Lazada, Tiki, Tiktok Shop) và cơ sở dữ liệu nội bộ (hệ thống ERP).

Thiết kế một kiến trúc Data Warehouse hiện đại theo mô hình ELT (Extract, Load, Transform), xác định rõ các tầng dữ liệu (Raw, Staging, Cleaned) để đảm bảo dữ liệu được xử lý một cách có hệ thống.

Thiết kế mô hình dữ liệu Dimensional Modeling (mô hình hóa chiều) với các bảng Fact và Dimension, tạo ra một cấu trúc tối ưu cho việc truy vấn và phân tích báo cáo.

Xây dựng chiến lược lưu trữ dữ liệu hiệu quả trên MinIO, áp dụng kỹ thuật phân vùng (partitioning) theo thời gian và kênh bán hàng để tối ưu hóa hiệu suất truy vấn.

Nghiên cứu và Ứng dụng Công nghệ Hiện đại:

Sử dụng Python và FastAPI để xây dựng các API giả lập, tạo ra nguồn dữ liệu có thể kiểm soát được, phục vụ cho quá trình phát triển và kiểm thử hệ thống.

Khai thác sức mạnh của Apache Airflow để xây dựng các quy trình dữ liệu (data pipelines) tự động, có khả năng lập lịch, giám sát, và xử lý lỗi một cách linh hoạt.

Tận dụng MinIO làm nền tảng lưu trữ chính cho Data Warehouse, sử dụng định dạng Parquet để tối ưu hóa không gian và tốc độ truy cập.

Ứng dụng DuckDB như một engine xử lý dữ liệu hiệu suất cao, cho phép thực hiện các phép biến đổi (transform) phức tạp trực tiếp trên các file Parquet mà không cần đến một hệ quản trị cơ sở dữ liệu riêng biệt.

Sử dụng Power BI để kết nối vào kho dữ liệu đã được làm sạch, xây dựng các mô hình dữ liệu và tạo ra các dashboard trực quan, tương tác.

Triển khai và Xây dựng Hệ thống Thực tế:

Xây dựng một bộ các DAG (Directed Acyclic Graph) trong Airflow để tự động hóa toàn bộ chu trình ELT: từ việc trích xuất dữ liệu định kỳ, chuyển đổi định dạng, làm sạch và chuẩn hóa dữ liệu.

Triển khai các logic kiểm tra và đảm bảo chất lượng dữ liệu (data quality) trong giai đoạn Transform để loại bỏ các sai sót và sự không nhất quán.

Xây dựng các bảng dữ liệu tổng hợp (data mart) sẵn sàng cho việc phân tích, tích hợp dữ liệu từ tất cả các kênh bán hàng.

Phát triển các dashboard trong Power BI để trực quan hóa các chỉ số kinh doanh (KPI) quan trọng, giúp ban lãnh đạo có cái nhìn tổng quan và chi tiết về hoạt động kinh doanh.

Đánh giá và Đề xuất Hướng phát triển:

Đánh giá hiệu suất của hệ thống đã xây dựng, bao gồm thời gian thực thi của các pipeline và khả năng đáp ứng của các dashboard.

Kiểm tra tính chính xác và nhất quán của dữ liệu sau khi đã được xử lý và tích hợp.

Đề xuất các hướng cải tiến và phát triển trong tương lai, chẳng hạn như tích hợp thêm các nguồn dữ liệu mới, áp dụng các mô hình phân tích nâng cao (dự báo doanh thu, phân khúc khách hàng), và tối ưu hóa hạ tầng để đáp ứng khối lượng dữ liệu lớn hơn.

1.3. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu của luận văn là các mô hình, kiến trúc và công nghệ được sử dụng để xây dựng hệ thống Data Warehouse. Luận văn cũng tập trung vào việc xử lý và phân tích dữ liệu bán hàng từ các nguồn đa dạng của doanh nghiệp thương mại điện tử vừa và nhỏ.

Luận văn sẽ tập trung vào việc thiết kế và triển khai một hệ thống Data Warehouse mẫu, xử lý dữ liệu bán hàng được thu thập từ các kênh TMĐT và website riêng của doanh nghiệp. Kết quả trực quan hóa sẽ được thể hiện thông qua các báo cáo phân tích cơ bản như doanh thu, tồn kho và hiệu quả kinh doanh.

1.4. Phương pháp nghiên cứu

Nghiên cứu tài liệu: Tổng hợp và phân tích các lý thuyết liên quan đến Data Warehouse, kiến trúc Data Lake, quy trình ELT và các công nghệ xử lý dữ liệu.

Phân tích và thiết kế hệ thống: Áp dụng phương pháp phân tích hệ thống để xác định yêu cầu, từ đó thiết kế kiến trúc và mô hình dữ liệu (Dimensional Modeling) phù hợp cho Data Warehouse.

Thực nghiệm và đánh giá: Triển khai giải pháp trên môi trường phát triển, kiểm thử tính đúng đắn và hiệu quả của hệ thống, đồng thời so sánh kết quả đạt được với các mục tiêu ban đầu.

II. Cơ sở lý thuyết và tổng quan công nghệ

2.1. Tổng quan về dữ liệu bán hàng trong thương mại điện tử

Dữ liệu bán hàng trong thương mại điện tử là tập hợp các thông tin được tạo ra và thu thập trong suốt quá trình giao dịch kinh doanh trực tuyến. Các loại dữ liệu này thường rất đa dạng và có cấu trúc khác nhau, bao gồm các bảng dữ liệu sau đây mà luận văn sẽ tập trung xử lý:

Dữ liệu đơn hàng (orders): Đây là dữ liệu cốt lõi, chứa các thông tin chi tiết về từng giao dịch mua hàng. Trong mô hình dữ liệu, bảng orders bao gồm các trường thông tin quan trọng như id, customer_id, order_date, status, total_price.

Dữ liệu mặt hàng trong đơn hàng (order_items): Dữ liệu này liên kết với đơn hàng, cung cấp chi tiết về các sản phẩm cụ thể được mua trong mỗi đơn hàng. Bảng order_items chứa order_id, product_id, quantity, và amount.

Dữ liệu sản phẩm (products): Chứa thông tin mô tả về các sản phẩm được bán. Bảng product bao gồm các trường như product_sku, brand_id, name, price, và sub_category_id.

Dữ liệu khách hàng (customers): Bao gồm thông tin về người mua hàng. Bảng customers chứa id, customer_code, và liên kết đến bảng geo_location thông qua geo_location_id.

Dữ liệu kênh bán hàng (order_channel): Đây là thông tin về nguồn gốc của đơn hàng (ví dụ: Shopee, Lazada, Tiki, Website riêng). Bảng order_channel chứa id và name của từng kênh.

Dữ liệu vận chuyển (shipment): Thông tin về quá trình giao hàng. Bảng shipment bao gồm các trường như logistics_partner_id, shipping_method_id, tracking_number, và shipping_status.

Các bảng dữ liệu bổ sung: Để tăng cường khả năng phân tích, hệ thống còn sử dụng các bảng dữ liệu bổ sung như payment (thông tin thanh toán), discount (thông tin khuyến mãi), logistics_partner (đối tác vận chuyển), brand (thương hiệu), category và sub_category (danh mục sản phẩm).

Trong bối cảnh cạnh tranh gay gắt của thị trường thương mại điện tử, việc phân tích dữ liệu bán hàng không chỉ là một lợi thế mà còn là yếu tố then chốt giúp doanh nghiệp tồn tại và phát triển. Phân tích dữ liệu bán hàng giúp doanh nghiệp:

Hiểu rõ hành vi khách hàng: Phân tích dữ liệu giúp xác định các xu hướng mua sắm, sản phẩm được ưa chuộng, và hành vi của từng phân khúc khách hàng.

Đánh giá hiệu suất bán hàng: Doanh nghiệp có thể theo dõi các chỉ số KPI quan trọng như doanh thu theo ngày, tuần, tháng, và giá trị đơn hàng trung bình.

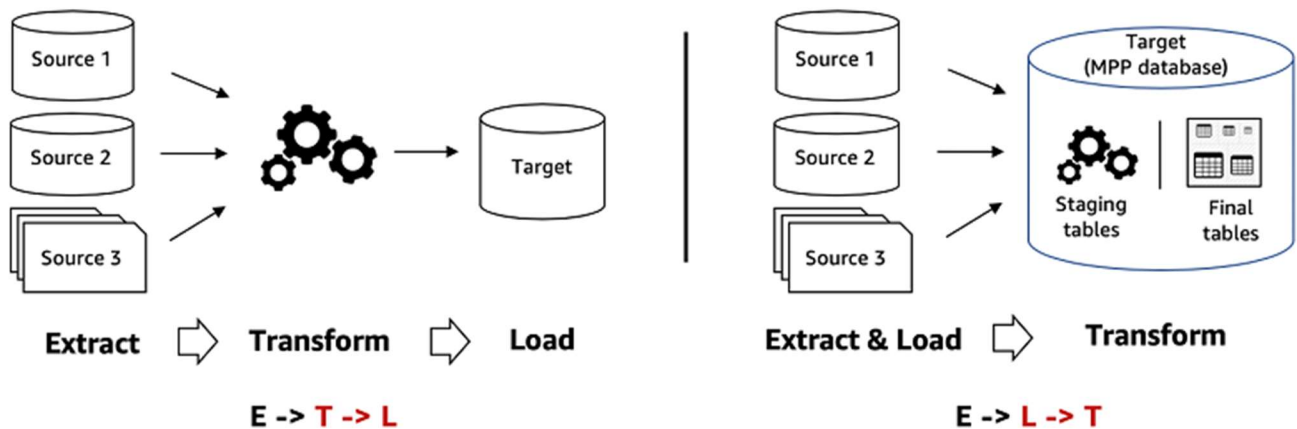
Quản lý tồn kho hiệu quả: Dựa vào dữ liệu bán hàng, doanh nghiệp có thể dự đoán nhu cầu, từ đó tối ưu hóa lượng hàng tồn kho, tránh tình trạng thiếu hàng hoặc tồn đọng.

Ra quyết định chiến lược: Phân tích dữ liệu giúp các nhà quản lý đưa ra các quyết định sáng suốt về mở rộng thị trường, phát triển sản phẩm mới, và cải thiện quy trình vận hành.

2.2. Các khái niệm về xử lý và trực quan hóa dữ liệu

2.2.1. So sánh giữa ETL và ELT

Hai mô hình xử lý dữ liệu chính trong các hệ thống kho dữ liệu hiện đại là ETL (Extract, Transform, Load) và ELT (Extract, Load, Transform). Sự khác biệt giữa hai mô hình này không chỉ là thứ tự các bước xử lý, mà còn là triết lý thiết kế và cách tiếp cận xử lý dữ liệu.



Bảng so sánh đặc điểm chính

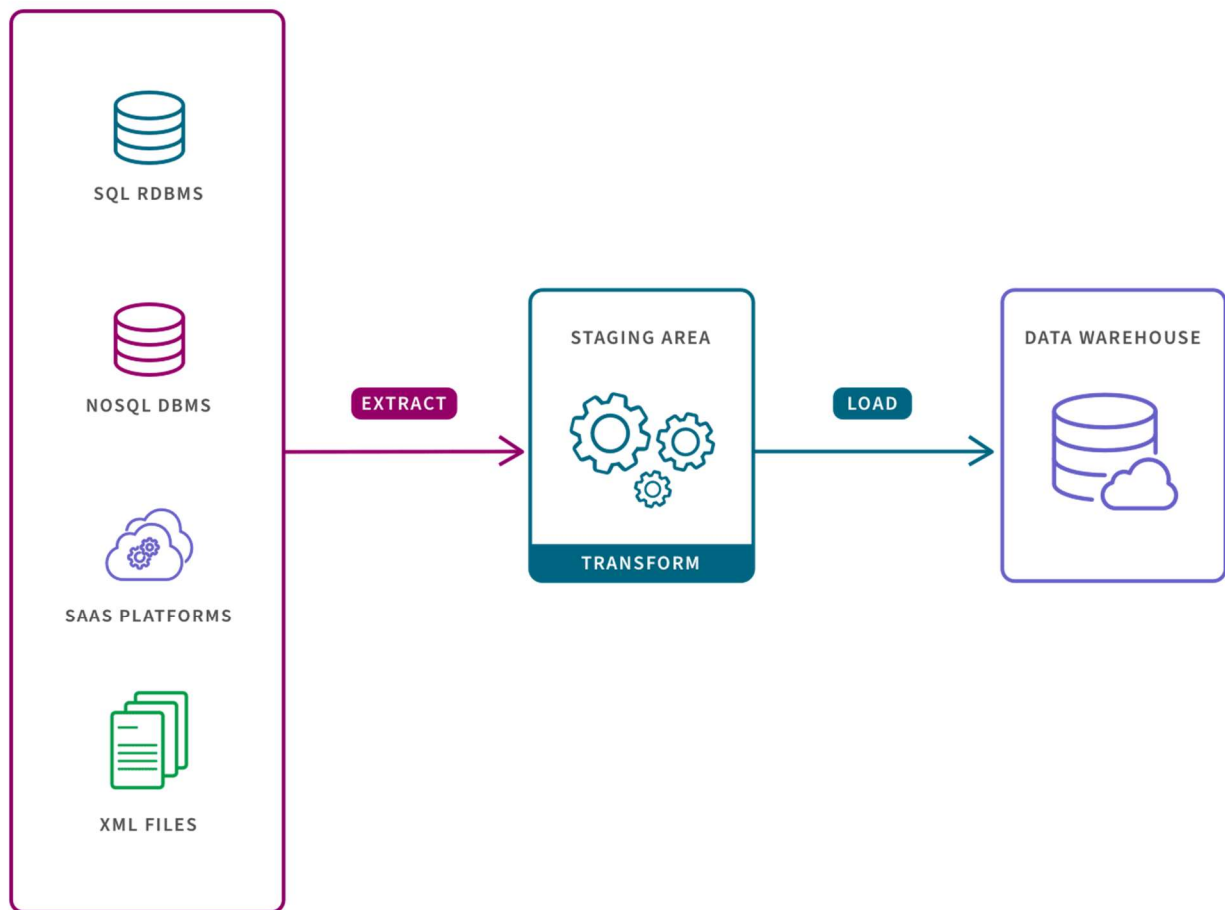
Tiêu chí	ETL (Extract, Transform, Load)	ELT (Extract, Load, Transform)
Quy trình	Dữ liệu được trích xuất, sau đó biến đổi trong một môi trường trung gian, cuối cùng mới tải vào kho dữ liệu	Dữ liệu được trích xuất và tải trực tiếp vào kho dữ liệu, sau đó mới thực hiện các biến đổi
Nơi xử lý dữ liệu	Hệ thống trung gian tách biệt (ETL server)	Trực tiếp trong kho dữ liệu đích (Data warehouse)
Loại dữ liệu phù hợp	Dữ liệu có cấu trúc, cần xử lý nghiêm ngặt trước khi tải	Dữ liệu đa dạng, bao gồm cả không cấu trúc và bán cấu trúc

Tiêu chí	ETL (Extract, Transform, Load)	ELT (Extract, Load, Transform)
Tốc độ nạp dữ liệu	Chậm hơn (do xử lý trước khi tải)	Nhanh hơn (tải trực tiếp, xử lý sau)
Khả năng mở rộng	Hạn chế bởi khả năng xử lý của máy chủ ETL	Cao hơn, tận dụng sức mạnh xử lý của kho dữ liệu
Chi phí triển khai	Cao hơn (yêu cầu hạ tầng riêng cho xử lý ETL)	Thấp hơn (tận dụng hạ tầng của kho dữ liệu)
Độ phức tạp trong phát triển	Phức tạp hơn, cần công cụ ETL chuyên dụng	Đơn giản hơn, sử dụng SQL và tính năng của kho dữ liệu
Khả năng thích ứng với thay đổi	Kém linh hoạt hơn, cần điều chỉnh quy trình ETL	Linh hoạt hơn, có thể dễ dàng thay đổi các phép biến đổi
Công nghệ phổ biến	Informatica, SSIS, Talend, IBM DataStage	Snowflake, Redshift, BigQuery, DuckDB
Tính lịch sử của dữ liệu	Thường chỉ lưu kết quả cuối, mất dữ liệu gốc	Có thể lưu cả dữ liệu thô và dữ liệu đã biến đổi
Môi trường phù hợp	Môi trường truyền thống, yêu cầu nghiêm ngặt về chất lượng dữ liệu	Kho dữ liệu hiện đại, cloud-native, dữ liệu lớn

Phân tích chi tiết

1. ETL (Extract, Transform, Load)

ETL là mô hình truyền thống trong xử lý dữ liệu, nơi dữ liệu được lấy từ các nguồn, biến đổi trong một môi trường riêng biệt, sau đó mới tải vào kho dữ liệu.



Ưu điểm:

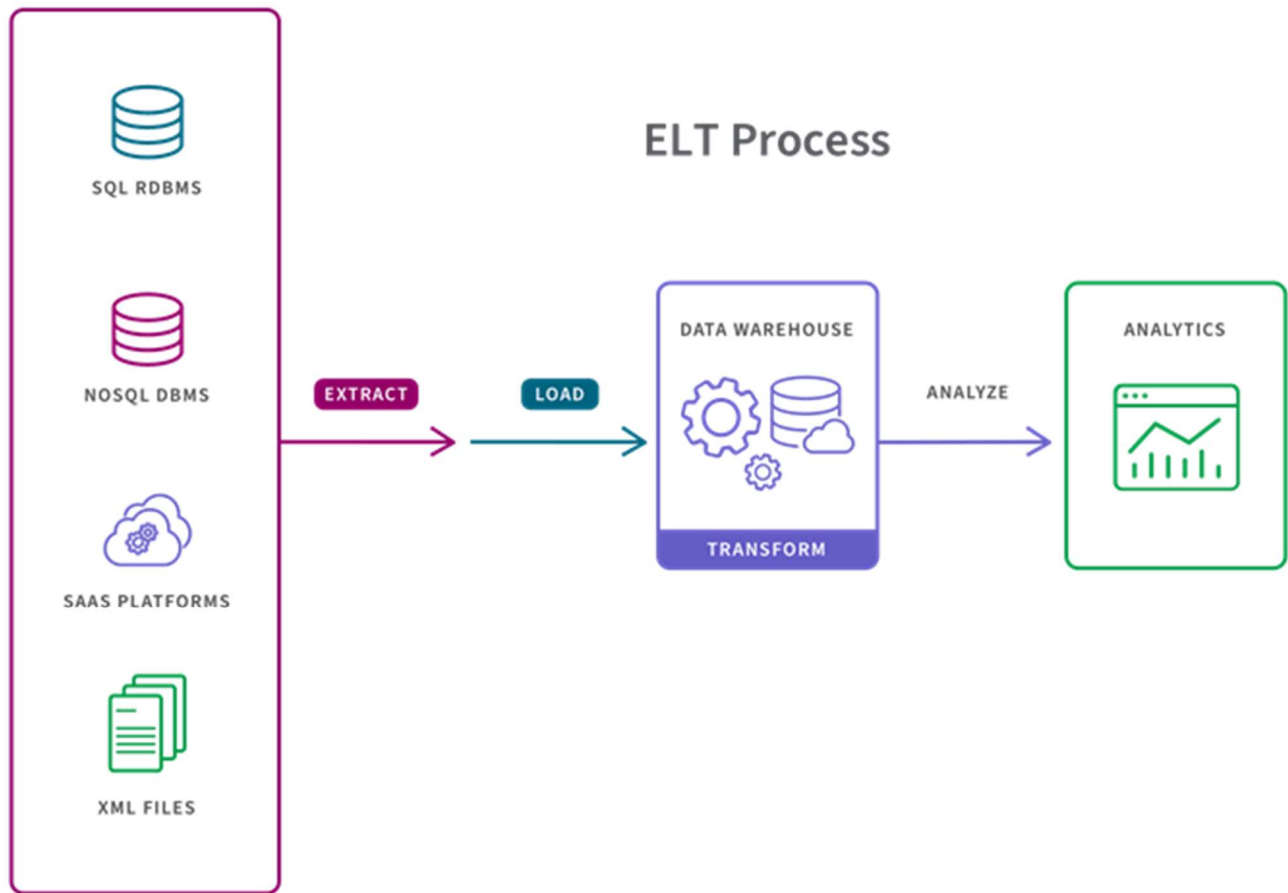
- Dữ liệu đã được làm sạch và biến đổi trước khi tải vào kho dữ liệu, đảm bảo chất lượng
- Giảm không gian lưu trữ trong kho dữ liệu (chỉ lưu dữ liệu có giá trị)
- Phù hợp với các hệ thống kho dữ liệu truyền thống không có khả năng xử lý dữ liệu mạnh
- Kiểm soát tốt quy trình biến đổi dữ liệu

Nhược điểm:

- Cần thêm hạ tầng và công cụ riêng cho quá trình biến đổi
- Khó mở rộng khi khối lượng dữ liệu tăng nhanh
- Mất đi dữ liệu gốc sau khi biến đổi, gây khó khăn cho việc khôi phục hoặc phân tích sau này
- Thời gian phát triển và triển khai lâu hơn

2. ELT (Extract, Load, Transform)

ELT là mô hình hiện đại, phổ biến trong các kho dữ liệu hiện đại, nơi dữ liệu được trích xuất và tải trực tiếp vào kho dữ liệu, sau đó mới thực hiện các biến đổi.



Ưu điểm:

Quá trình nạp dữ liệu nhanh hơn (tải dữ liệu thô trực tiếp)

Tận dụng được sức mạnh xử lý của các kho dữ liệu hiện đại

Lưu giữ dữ liệu gốc, tạo điều kiện cho việc phân tích và truy xuất sau này

Linh hoạt trong việc thay đổi logic biến đổi mà không cần chạy lại toàn bộ quy trình

Phù hợp với kỷ nguyên dữ liệu lớn và đa dạng

Nhược điểm:

Yêu cầu kho dữ liệu có khả năng xử lý mạnh

Chi phí lưu trữ cao hơn (lưu cả dữ liệu thô và dữ liệu đã xử lý)

Có thể gặp khó khăn trong quản lý chất lượng dữ liệu nếu không có quy trình rõ ràng

Phụ thuộc vào khả năng xử lý SQL của kho dữ liệu

Lý do lựa chọn ELT cho dự án

Dự án này đã triển khai mô hình ELT bằng Apache Airflow và DuckDB vì những lý do sau:

6 **Tính linh hoạt cao:** Cho phép lưu trữ dữ liệu thô từ nhiều nguồn khác nhau (các sản phẩm TMĐT) trong MinIO trước khi thực hiện các phép biến đổi, tạo điều kiện cho việc thay đổi logic xử lý mà không cần trích xuất lại dữ liệu.

Khả năng xử lý đa dạng nguồn dữ liệu: Mô hình ELT phù hợp với dự án này khi phải xử lý dữ liệu từ nhiều nguồn với cấu trúc khác nhau (Shopee, Lazada, Tiki, Tiktok Shop).

Tận dụng sức mạnh của DuckDB: DuckDB có khả năng xử lý dữ liệu Parquet hiệu quả, cho phép thực hiện các phép biến đổi phức tạp trực tiếp trên dữ liệu lưu trong MinIO.

Chi phí triển khai thấp: Không cần thêm hạ tầng riêng cho xử lý ETL, tất cả được thực hiện trong cùng một kiến trúc với MinIO và DuckDB.

6 **Khả năng tái sử dụng và phân tích lịch sử:** Việc lưu trữ dữ liệu thô cho phép tái xử lý khi cần thiết và thực hiện các phân tích trên dữ liệu lịch sử.

Cấu trúc đa tầng rõ ràng: Mô hình ELT triển khai trong dự án với các tầng raw, staging, cleaned giúp quản lý quy trình xử lý dữ liệu một cách có tổ chức và minh bạch.

1 7 **Tóm lại, ELT là sự lựa chọn phù hợp cho các dự án kho dữ liệu hiện đại có nhu cầu xử lý dữ liệu đa dạng, linh hoạt và có khả năng mở rộng, đặc biệt là các dự án sử dụng công nghệ lưu trữ và xử lý dữ liệu hiện đại như MinIO và DuckDB.**

2.2.2. Data Pipeline và trực quan hóa dữ liệu

3 Data Pipeline là một chuỗi các bước tự động được thiết kế để di chuyển và xử lý dữ liệu từ các nguồn khác nhau đến một điểm đích (thường là Data Warehouse) để phục vụ cho việc phân tích. Luận văn này đã triển khai mô hình ELT (Extract, Load, Transform) bằng Apache Airflow, một phương pháp xử lý dữ liệu hiện đại, trong đó:

Extract: Dữ liệu được trích xuất từ các API của sản phẩm thương mại điện tử (Shopee, Lazada, Tiki, Tiktok Shop) và website riêng của công ty thông qua các DAG Airflow chuyên biệt (daily_extract_shopee_order, daily_extract_lazada_order, v.v.), mỗi DAG sẽ gọi tới các API endpoint tương ứng và lấy về dữ liệu JSON.

Load: Dữ liệu JSON thô sau khi trích xuất được lưu trực tiếp vào MinIO theo cấu trúc thư mục được phân vùng theo kênh bán hàng, loại dữ liệu và thời gian (year/month/day). Các DAG như transform_shopee_order_to_parquet, transform_lazada_order_to_parquet... sẽ chuyển đổi dữ liệu JSON thành định dạng Parquet hiệu quả hơn và lưu vào tầng staging trong MinIO.

3 **Transform:** Dữ liệu từ tầng staging được xử lý bởi các DAG tiếp theo như clean_order_data_shopee_with_duckdb, clean_order_data_lazada_with_duckdb... để làm sạch, chuẩn hóa và chuyển đổi thành dạng phù hợp cho việc phân tích. Trong bước này, DuckDB được sử dụng để truy vấn và xử lý dữ liệu Parquet trực tiếp từ MinIO thông qua các câu lệnh SQL, giúp chuẩn hóa cấu trúc dữ liệu và thêm các trường tính toán cần thiết. Mô hình này cho phép xử lý linh hoạt và tận dụng sức mạnh của DuckDB để xử lý dữ liệu hiệu quả mà không cần di chuyển dữ liệu ra khỏi kho lưu trữ.

1 **Trực quan hóa dữ liệu:** là quá trình chuyển đổi dữ liệu thành các biểu đồ, đồ thị, và các hình ảnh khác để giúp người dùng dễ dàng hiểu và phân tích. Một số nguyên tắc quan trọng để trực quan hóa dữ liệu hiệu quả bao gồm:

Đơn giản và rõ ràng: Đồ thị và biểu đồ phải dễ hiểu, không có quá nhiều chi tiết gây xao lãng.

Lựa chọn loại biểu đồ phù hợp: Sử dụng các loại biểu đồ khác nhau (biểu đồ cột, biểu đồ đường, biểu đồ tròn) tùy thuộc vào loại dữ liệu và thông điệp muốn truyền tải.

Tập trung vào ý nghĩa dữ liệu: Trực quan hóa nên kể một câu chuyện có ý nghĩa, giúp người dùng trả lời các câu hỏi kinh doanh cụ thể.

Thiết kế tương tác: Cung cấp các chức năng tương tác (như bộ lọc, drill-down) để người dùng có thể tự khám phá dữ liệu.

2.3. So sánh các công nghệ xử lý dữ liệu phân tích

Việc lựa chọn công nghệ phù hợp cho xử lý dữ liệu phân tích là một quyết định quan trọng, ảnh hưởng trực tiếp đến hiệu suất, chi phí và khả năng mở rộng của hệ thống Data Warehouse. Dưới đây là phân tích so sánh giữa ba giải pháp xử lý dữ liệu hàng đầu hiện nay: DuckDB, Apache Spark và Google BigQuery.

Bảng so sánh đặc điểm chính

Tiêu chí	DuckDB	Apache Spark	Google BigQuery
Kiến trúc	Cơ sở dữ liệu phân tích cột nhúng trong quá trình	Framework xử lý phân tán	Dịch vụ kho dữ liệu serverless trên đám mây
Quy mô dữ liệu tối ưu	GB đến TB nhỏ	TB đến PB	TB đến PB+
Mô hình triển khai	Local/Single node	Cluster/phân tán	Dịch vụ đám mây hoàn toàn
Độ phức tạp khi triển khai	Rất thấp (thư viện nhúng)	Cao (cấu hình cluster, điều chỉnh tham số)	Trung bình (dịch vụ quản lý)
Hiệu suất với dữ liệu nhỏ-trung bình	Xuất sắc	Thấp hơn (overhead của phân tán)	Tốt nhưng phụ thuộc vào slot
Khả năng mở rộng	Hạn chế (bộ nhớ máy đơn)	Rất cao (thêm node)	Rất cao (tự động)
Chi phí	Miễn phí, mã nguồn mở	Miễn phí (mã nguồn mở), chi phí phần cứng	Trả tiền theo lượng dữ liệu quét/tính toán
Tích hợp với Parquet/formats cột	Tích hợp sâu, truy vấn trực tiếp	Hỗ trợ tốt	Hỗ trợ tốt, tối ưu cho đám mây
Cộng đồng & hỗ trợ	Đang phát triển nhanh	Rất lớn, trưởng thành	Hỗ trợ thương mại từ Google

Tiêu chí	DuckDB	Apache Spark	Google BigQuery
Môi trường phù hợp	Phân tích cục bộ, Data pipeline nhẹ	Xử lý dữ liệu lớn, ML	Phân tích dữ liệu lớn trên đám mây

Phân tích chi tiết

1. DuckDB

DuckDB là một cơ sở dữ liệu phân tích dạng cột (columnar analytical database) được thiết kế để hoạt động như một thư viện nhúng trong quá trình (in-process). Mô hình này tương tự SQLite nhưng được tối ưu hóa đặc biệt cho các tác vụ phân tích OLAP.

Điểm mạnh:

- Hiệu suất cực kỳ cao với dữ liệu vừa và nhỏ, đặc biệt khi xử lý định dạng cột như Parquet
- Đơn giản để triển khai, không yêu cầu cài đặt máy chủ riêng
- Có thể truy vấn dữ liệu trực tiếp từ nhiều nguồn (Parquet, CSV, JSON) mà không cần tải toàn bộ vào bộ nhớ
- Tiêu thụ ít tài nguyên, phù hợp với môi trường giới hạn
- Tích hợp tốt với các công cụ phân tích dữ liệu như Python, R

Điểm yếu:

- Giới hạn về khả năng xử lý dữ liệu rất lớn (petabyte)
- Không có khả năng xử lý phân tán tự nhiên
- Cộng đồng và hệ sinh thái nhỏ hơn so với các giải pháp trưởng thành

2. Apache Spark

Apache Spark là một framework xử lý dữ liệu phân tán mạnh mẽ, được thiết kế để xử lý dữ liệu lớn trên các cụm máy tính.

Điểm mạnh:

- Khả năng mở rộng gần như không giới hạn thông qua thêm node
- Hỗ trợ xử lý dữ liệu batch và streaming
- Hệ sinh thái phong phú với các module ML, Graph, SQL
- Tính năng xử lý đa dạng (SQL, dataframe, ML)
- Cộng đồng lớn và được sử dụng rộng rãi trong doanh nghiệp

Điểm yếu:

- Tốn kém về tài nguyên, ngay cả với dữ liệu nhỏ

- Phức tạp trong việc triển khai và tối ưu hóa
- Thời gian khởi động và overhead lớn cho công việc nhỏ
- Yêu cầu kiến thức chuyên sâu để vận hành hiệu quả

3. Google BigQuery

BigQuery là dịch vụ kho dữ liệu serverless hoàn toàn được quản lý trên nền tảng Google Cloud, cho phép phân tích dữ liệu lớn bằng SQL.

Điểm mạnh:

- Khả năng mở rộng tự động, không cần quản lý cơ sở hạ tầng
- Xử lý hiệu quả dữ liệu cực lớn (petabyte+)
- Tích hợp sâu với hệ sinh thái Google Cloud
- Mô hình serverless giảm chi phí quản lý vận hành
- Tính năng chia sẻ dữ liệu và bảo mật doanh nghiệp mạnh mẽ

Điểm yếu:

- Chi phí có thể tăng nhanh với khối lượng truy vấn lớn
- Phụ thuộc vào Google Cloud, khó di chuyển sang nền tảng khác
- Không phù hợp với môi trường local hoặc hybrid
- Độ trễ khởi động truy vấn cao hơn với công việc nhỏ

Lý do lựa chọn DuckDB cho dự án

Dự án này đã lựa chọn DuckDB làm công nghệ xử lý dữ liệu chính dựa trên các yếu tố sau:

1. **Phù hợp với quy mô dữ liệu:** Khối lượng dữ liệu bán hàng từ các kênh TMĐT của doanh nghiệp vừa và nhỏ nằm trong phạm vi xử lý hiệu quả của DuckDB (gigabyte đến terabyte nhỏ).
1. **Đơn giản hóa kiến trúc:** DuckDB cho phép xử lý dữ liệu trực tiếp từ MinIO mà không cần một hệ thống cơ sở dữ liệu riêng biệt, giảm thiểu số lượng thành phần trong hệ thống.
1. **Hiệu suất với file Parquet:** DuckDB có hiệu năng vượt trội khi xử lý dữ liệu định dạng cột như Parquet, đặc biệt là các phép lọc, tổng hợp và phân tích thường dùng trong trường hợp này.
1. **Chi phí triển khai thấp:** Là giải pháp mã nguồn mở, không tốn chi phí license và yêu cầu tài nguyên hệ thống thấp, phù hợp với ngân sách của doanh nghiệp vừa và nhỏ.
2. **Tích hợp dễ dàng:** DuckDB tích hợp liền mạch với Python trong các script Airflow, cho phép xử lý dữ liệu hiệu quả ngay trong luồng ELT mà không cần chuyển dữ liệu giữa các hệ thống.
1. **Thời gian phát triển nhanh:** Với cú pháp SQL quen thuộc và cấu hình đơn giản, việc phát triển và bảo trì các pipeline dữ liệu trở nên dễ dàng hơn đáng kể so với các giải pháp phức tạp như Spark.

Trong khi Apache Spark và Google BigQuery sẽ trở thành lựa chọn tốt hơn khi doanh nghiệp mở rộng với khối lượng dữ liệu lớn hơn nhiều hoặc yêu cầu phân tích thời gian thực, DuckDB hiện là giải pháp cân bằng tối ưu giữa hiệu suất, chi phí và độ phức tạp cho nhu cầu hiện tại của dự án.

2.4. So sánh các công nghệ lưu trữ dữ liệu phân tích: MinIO, Hadoop HDFS, và Microsoft OneLake

Việc lựa chọn nền tảng lưu trữ dữ liệu phù hợp là yếu tố then chốt quyết định hiệu quả, khả năng mở rộng và chi phí của hệ thống Data Warehouse hiện đại. Dưới đây là phần giới thiệu và so sánh ba giải pháp lưu trữ phổ biến: MinIO, Hadoop HDFS và Microsoft OneLake.

Bảng so sánh đặc điểm chính

Tiêu chí	MinIO	Hadoop HDFS	Microsoft OneLake
Kiến trúc	Lưu trữ đối tượng (object storage), tương thích S3	Hệ thống file phân tán (distributed file system)	Lakehouse-as-a-Service, tích hợp sâu với Fabric
Triển khai	On-premises, cloud, hybrid	On-premises, cloud (qua Hadoop distros)	Cloud (Azure), tích hợp Power BI, Synapse
Khả năng mở rộng	Rất cao, scale-out dễ dàng	Rất cao, scale-out theo cluster	Rất cao, tự động mở rộng trên Azure
Tương thích API	S3 API, đa nền tảng	API riêng (HDFS), tích hợp Hadoop stack	S3 API, ADLS Gen2 API, Fabric API
Quản lý dữ liệu	Đơn giản, giao diện web, CLI	Quản lý phức tạp, cần cluster manager	Quản lý tập trung qua Fabric Portal
Bảo mật	TLS, IAM, bucket policy	Kerberos, ACL, encryption	Azure AD, RBAC, Data Governance
Tích hợp phân tích	DuckDB, Spark, Presto, Trino, Power BI	Spark, Hive, Impala, Presto	Power BI, Synapse, Spark, Data Factory
Chi phí	Miễn phí (mã nguồn mở), trả phí cho support	Miễn phí (mã nguồn mở), chi phí phần cứng	Trả phí theo dung lượng và dịch vụ Azure
Tính sẵn sàng	Hỗ trợ replication, distributed erasure code	Hỗ trợ replication, HA	SLA cao, geo-redundancy trên Azure
Môi trường phù hợp	Doanh nghiệp vừa/nhỏ, hybrid, cloud-native	Doanh nghiệp lớn, Big Data truyền thống	Doanh nghiệp sử dụng hệ sinh thái Microsoft

Phân tích chi tiết

1. MinIO

MinIO là giải pháp lưu trữ đối tượng mã nguồn mở, tương thích hoàn toàn với giao diện S3 của Amazon. MinIO có thể triển khai trên hạ tầng vật lý, đám mây hoặc hybrid, rất phù hợp cho các hệ thống Data Lake/Data Warehouse hiện đại nhờ khả năng tích hợp tốt với các công cụ phân tích như DuckDB, Spark, Presto, Power BI.

Ưu điểm:

- Dễ triển khai, cấu hình đơn giản, giao diện quản trị trực quan
- Tương thích S3, dễ tích hợp với nhiều hệ sinh thái Big Data/Cloud
- Hiệu suất cao, hỗ trợ phân tán, replication, erasure code
- Phù hợp cho cả môi trường on-premises và cloud-native

Nhược điểm:

- Một số tính năng nâng cao (multi-site replication, object locking) cần trả phí
- Không có hệ sinh thái phân tích dữ liệu tích hợp sẵn như các giải pháp cloud lớn

2. Hadoop HDFS

Hadoop Distributed File System (HDFS) là hệ thống file phân tán truyền thống, nền tảng của các hệ thống Big Data như Hadoop, Spark, Hive. HDFS được thiết kế để lưu trữ và xử lý dữ liệu lớn trên các cluster vật lý.

Ưu điểm:

- Khả năng mở rộng rất lớn, phù hợp với Big Data truyền thống
- Tích hợp sâu với các công cụ phân tích như Spark, Hive, Impala
- Hỗ trợ replication, đảm bảo tính sẵn sàng và an toàn dữ liệu

Nhược điểm:

- Quản trị phức tạp, cần đội ngũ vận hành chuyên sâu
- Không tương thích S3, khó tích hợp với các dịch vụ cloud hiện đại
- Không phù hợp với các workload cloud-native hoặc hybrid

3. Microsoft OneLake

OneLake là nền tảng lưu trữ dữ liệu lakehouse thế hệ mới của Microsoft, tích hợp sâu với Microsoft Fabric, Power BI, Synapse Analytics. OneLake cung cấp khả năng lưu trữ tập trung, quản lý dữ liệu cho toàn bộ hệ sinh thái Microsoft trên Azure.

Ưu điểm:

- Tích hợp liền mạch với Power BI, Synapse, Data Factory, AI/ML
- Quản lý tập trung, bảo mật mạnh mẽ với Azure AD, RBAC
- Hỗ trợ đa định dạng (Parquet, Delta, CSV), truy cập qua S3 API, ADLS Gen2 API

- Tự động mở rộng, tối ưu chi phí theo nhu cầu sử dụng

Nhược điểm:

- Phụ thuộc vào Azure, khó triển khai on-premises hoặc hybrid
- Chi phí có thể tăng nhanh với khối lượng dữ liệu lớn
- Một số tính năng nâng cao cần đăng ký dịch vụ bổ sung

Lý do lựa chọn MinIO cho dự án

Dự án lựa chọn MinIO vì các lý do sau:

- **Tương thích S3:** Dễ dàng tích hợp với các công cụ phân tích hiện đại như DuckDB, Spark, Power BI.
- **Triển khai linh hoạt:** Có thể triển khai trên máy chủ vật lý, cloud hoặc hybrid, phù hợp với doanh nghiệp vừa và nhỏ.
- **Hiệu suất cao, chi phí thấp:** Miễn phí, mã nguồn mở, không phụ thuộc vào nền tảng cloud lớn.
- **Đơn giản hóa quản trị:** Giao diện quản trị trực quan, dễ vận hành, không cần đội ngũ chuyên sâu như Hadoop.

MinIO là lựa chọn tối ưu cho các hệ thống Data Warehouse hiện đại quy mô vừa và nhỏ, cần sự linh hoạt, hiệu suất và khả năng tích hợp tốt với các công cụ phân tích dữ liệu mới nhất.

2.5. So sánh các công cụ trực quan hóa dữ liệu: Power BI và Google Data Studio

Việc lựa chọn công cụ trực quan hóa dữ liệu phù hợp đóng vai trò quan trọng trong việc khai thác giá trị từ kho dữ liệu, giúp doanh nghiệp dễ dàng theo dõi, phân tích và ra quyết định dựa trên dữ liệu. Dưới đây là phần giới thiệu và so sánh hai công cụ phổ biến: Power BI và Google Data Studio.

Bảng so sánh đặc điểm chính

Tiêu chí	Power BI	Google Data Studio
Nhà phát triển	Microsoft	Google
Triển khai	Desktop, Cloud (Power BI Service), Mobile	Cloud (Web-based), miễn phí
Tích hợp dữ liệu	Rất mạnh, hỗ trợ nhiều nguồn (SQL, Excel, Parquet, API, Azure, AWS, Google Cloud, v.v.)	Tốt, tập trung vào Google ecosystem (BigQuery, Sheets, Analytics, v.v.), hỗ trợ API và CSV
Khả năng xử lý dữ liệu	Mạnh mẽ, hỗ trợ DAX, Power Query, mô hình hóa dữ liệu phức tạp	Chủ yếu trực quan hóa, xử lý dữ liệu đơn giản, ít tính năng ETL

Tiêu chí	Power BI	Google Data Studio
Tính năng trực quan hóa	Đa dạng, nhiều loại biểu đồ, custom visuals, drill-down, drill-through, bookmarks, dashboard tương tác cao	Đầy đủ biểu đồ cơ bản, hỗ trợ custom charts qua cộng đồng, dashboard tương tác tốt
Khả năng chia sẻ	Chia sẻ qua Power BI Service, workspace, publish to web, embed, xuất PDF/PPT	Chia sẻ qua link, embed, xuất PDF, kiểm soát quyền truy cập đơn giản
Bảo mật & quản trị	Hỗ trợ Azure AD, RLS (Row Level Security), workspace, quản lý người dùng chi tiết	Quản lý qua Google Account, phân quyền đơn giản, không có RLS nâng cao
Chi phí	Miễn phí (Desktop), trả phí cho Power BI Pro/Service	Miễn phí (đa số tính năng), trả phí cho BigQuery hoặc nguồn dữ liệu doanh nghiệp
Cộng đồng & hỗ trợ	Rất lớn, tài liệu phong phú, nhiều mẫu dashboard	Lớn, tài liệu đầy đủ, cộng đồng Google mạnh
Môi trường phù hợp	Doanh nghiệp vừa/lớn, nhu cầu phân tích phức tạp, tích hợp nhiều nguồn	Doanh nghiệp nhỏ/vừa, startup, nhu cầu báo cáo nhanh, ưu tiên Google Cloud

Phân tích chi tiết

1. Power BI

Power BI là bộ công cụ BI mạnh mẽ của Microsoft, hỗ trợ từ việc kết nối, xử lý, mô hình hóa đến trực quan hóa dữ liệu. Power BI nổi bật với khả năng tích hợp nhiều nguồn dữ liệu, xử lý dữ liệu lớn, hỗ trợ các phép tính phức tạp với DAX, và khả năng xây dựng dashboard tương tác chuyên sâu. Ngoài ra, Power BI còn hỗ trợ bảo mật dữ liệu ở mức dòng (Row Level Security), chia sẻ báo cáo linh hoạt qua Power BI Service, và tích hợp tốt với hệ sinh thái Microsoft (Azure, Office 365).

Ưu điểm:

- Xử lý và mô hình hóa dữ liệu mạnh mẽ, phù hợp cho phân tích chuyên sâu
- Hỗ trợ nhiều nguồn dữ liệu, kể cả file Parquet, cơ sở dữ liệu lớn, API
- Dashboard tương tác cao, nhiều loại biểu đồ, hỗ trợ custom visuals
- Bảo mật và quản trị người dùng chi tiết, phù hợp doanh nghiệp lớn
- Có thể xuất báo cáo ra PDF, PowerPoint, embed vào website hoặc ứng dụng

Nhược điểm:

- Một số tính năng nâng cao yêu cầu trả phí (Power BI Pro/Service)
- Giao diện và thao tác có thể phức tạp với người mới
- Tối ưu nhất khi dùng trong hệ sinh thái Microsoft

2. Google Data Studio (Looker Studio)

Google Data Studio (nay là Looker Studio) là công cụ trực quan hóa dữ liệu miễn phí, chạy hoàn toàn trên nền web, tập trung vào sự đơn giản, dễ chia sẻ và tích hợp sâu với các dịch vụ Google như BigQuery, Google Sheets, Google Analytics.

Ưu điểm:

- Miễn phí, dễ sử dụng, giao diện trực quan, thao tác kéo thả
- Tích hợp mạnh với Google Cloud, BigQuery, Sheets, Analytics
- Chia sẻ báo cáo dễ dàng qua link, embed, xuất PDF
- Phù hợp cho báo cáo nhanh, dashboard marketing, startup, doanh nghiệp nhỏ

Nhược điểm:

- 3 • Khả năng xử lý và mô hình hóa dữ liệu hạn chế, không phù hợp cho phân tích phức tạp
- Hỗ trợ nguồn dữ liệu ngoài Google ecosystem còn hạn chế
- 4 • Không có tính năng bảo mật nâng cao như RLS
- Ít tính năng custom visuals hơn Power BI

Lý do lựa chọn Power BI cho dự án

Dự án lựa chọn Power BI vì các lý do sau:

- **Tích hợp đa nguồn:** Kết nối tốt với file Parquet, DuckDB, MinIO, các cơ sở dữ liệu lớn và API.
- **Khả năng phân tích mạnh:** Hỗ trợ DAX, Power Query, phù hợp cho các dashboard phân tích KPI, drill-down, phân tích đa chiều.
- 4 • **Bảo mật và chia sẻ:** Hỗ trợ chia sẻ nội bộ, bảo mật dữ liệu, phù hợp với doanh nghiệp vừa và nhỏ có nhu cầu mở rộng.
- 2 • **Khả năng mở rộng:** Khi doanh nghiệp phát triển, Power BI vẫn đáp ứng tốt nhu cầu phân tích dữ liệu lớn, tích hợp với Azure và các dịch vụ đám mây khác.

Google Data Studio là lựa chọn phù hợp cho các doanh nghiệp nhỏ, startup hoặc các nhóm marketing cần báo cáo nhanh, chi phí thấp, ưu tiên tích hợp với Google Cloud. Tuy nhiên, với nhu cầu phân tích phức tạp, đa nguồn và bảo mật cao, Power BI là lựa chọn tối ưu cho hệ thống Data Warehouse của dự án này.

III. PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.1. Phân tích yêu cầu hệ thống

Dựa trên bối cảnh doanh nghiệp và mục tiêu đã đặt ra, việc phân tích yêu cầu hệ thống tập trung vào việc xác định rõ các nhu cầu về dữ liệu và phân tích kinh doanh. Mục tiêu chính là xây dựng một hệ thống tự động, hiệu quả để xử lý dữ liệu bán hàng và tạo ra các dashboard trực quan, hỗ trợ ra quyết định.

3.1.1. Khảo sát và xác định nguồn dữ liệu

Các nguồn dữ liệu được xác định thông qua việc giả lập API của cửa hàng thương mại điện tử lớn (Shopee, Tiktok Shop, Tiki, ...) và các hệ thống nội bộ của doanh nghiệp (ERP). Quá trình này giúp nắm bắt được cấu trúc dữ liệu, phương thức truy cập và cách thức cập nhật dữ liệu từ mỗi nguồn, từ đó làm cơ sở cho việc thiết kế hệ thống.

3.1.2. Phân tích dữ liệu từ các sàn thương mại điện tử

Phương thức lấy dữ liệu:

Chủ yếu thông qua API. Các API cho phép lấy dữ liệu theo khoảng thời gian nhất định (ví dụ: danh sách đơn hàng, danh sách sản phẩm)

Định dạng dữ liệu:

Dữ liệu trả về từ các API của ở định dạng JSON.

Các loại dữ liệu chính:

Dữ liệu đơn hàng, dữ liệu sản phẩm, thông tin vận chuyển, thông tin khách hàng, và các chỉ số liên quan khác.

3.1.3. Phân tích dữ liệu từ các hệ thống nội bộ

Phương thức lấy dữ liệu:

Sử dụng các API do hệ thống nội bộ (website, ERP, CRM) cung cấp.

Định dạng dữ liệu:

Các API này trả về dữ liệu dưới dạng file CSV.

Các loại dữ liệu chính:

Dữ liệu khách hàng, thông tin tồn kho, chi tiết sản phẩm, và các chương trình khuyến mãi.

Mô phỏng dữ liệu:

Để phục vụ cho việc phát triển và kiểm thử hệ thống, FastAPI sẽ được sử dụng để xây dựng các API giả lập, mô phỏng hoạt động của các hệ thống nội bộ này. Các API giả lập sẽ cung cấp dữ liệu bán hàng có kiểm soát.

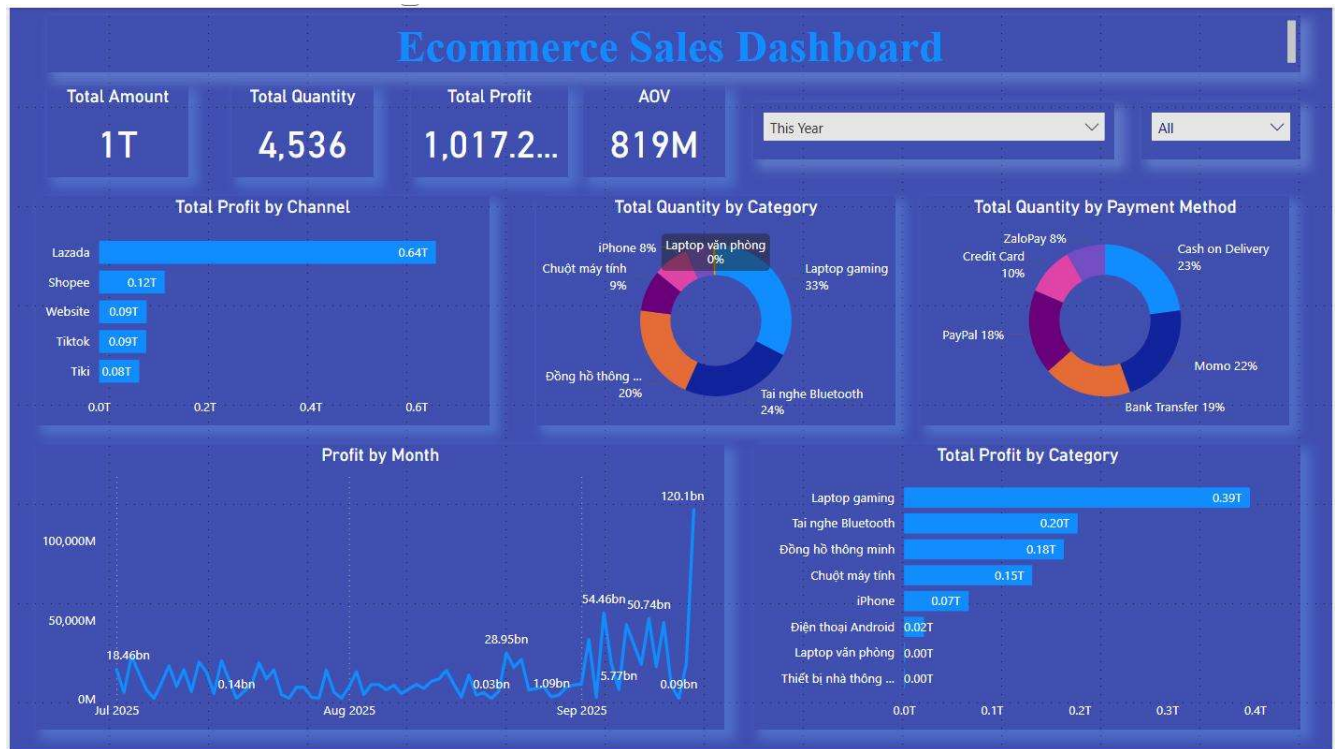
3.1.4. Xác định các chỉ số kinh doanh (KPI) và dashboard

Dựa trên các yêu cầu phân tích từ doanh nghiệp, các chỉ số kinh doanh (KPI) và dashboard cần xây dựng bao gồm:

Các chỉ số kinh doanh (KPI): Tổng doanh thu, doanh thu theo ngày/tháng/năm, doanh thu theo từng kênh bán hàng, doanh thu theo sản phẩm/danh mục sản phẩm, hiệu quả khuyến mãi, tình trạng tồn kho, và lợi nhuận.

Dashboard:

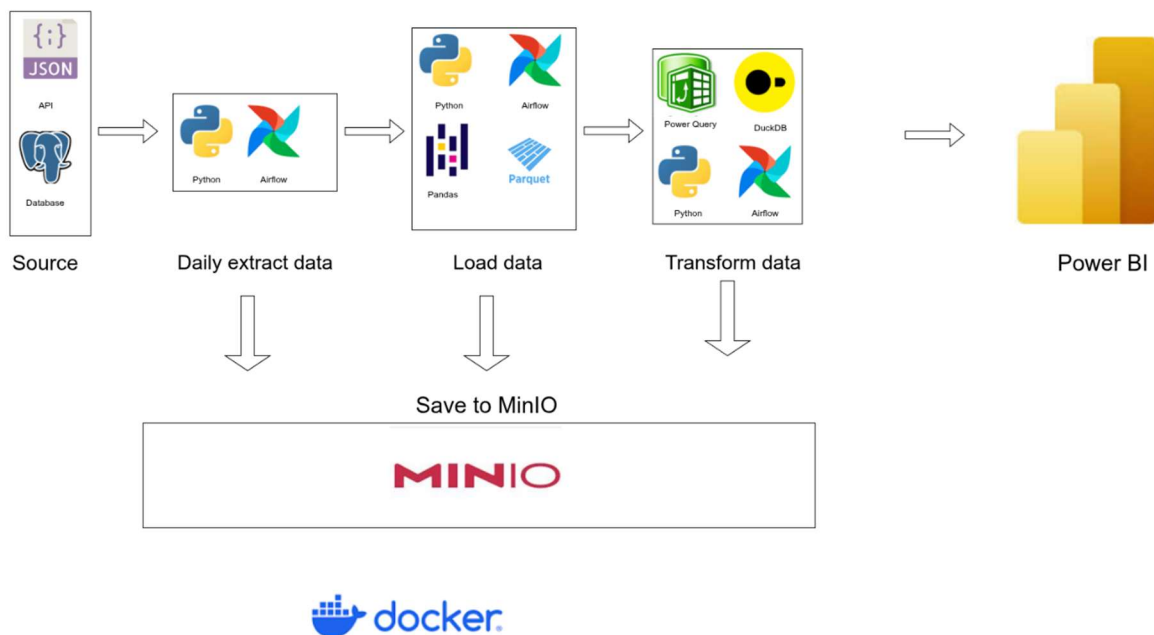
Sales Performance Dashboard: Tổng quan về doanh số, xu hướng doanh thu, lợi nhuận.



3.2. Thiết kế kiến trúc hệ thống

Kiến trúc hệ thống đã được triển khai theo mô hình ELT (Extract - Load - Transform), tận dụng các công cụ mã nguồn mở để xây dựng một giải pháp tối ưu cho doanh nghiệp vừa và nhỏ.

Kiến trúc tổng thể: Kiến trúc hệ thống được xây dựng với mục tiêu tự động hóa toàn bộ luồng dữ liệu, từ nguồn đến dashboard, với khả năng mở rộng và tính ổn định cao.



Thành phần nguồn dữ liệu:

Dữ liệu được thu thập từ các API giả lập được phát triển bằng FastAPI, trả về định dạng JSON mô phỏng dữ liệu từ các sàn thương mại điện tử (Shopee, Lazada, Tiki, Tiktok Shop) và website riêng.

API giả lập bao gồm các endpoint như /extract-order, /extract-user, /extract-order-items để cung cấp dữ liệu từng loại theo kênh bán hàng.

Hệ thống đã được thiết kế để mở rộng dễ dàng, có thể thay thế API giả lập bằng các API thật của các sàn TMĐT khi triển khai thực tế.

Thành phần xử lý dữ liệu (ELT Pipeline):

Hệ thống sử dụng Apache Airflow làm công cụ điều phối (orchestrator) các pipeline dữ liệu. Airflow là một nền tảng mã nguồn mở mạnh mẽ, cho phép lập lịch, quản lý và giám sát các quy trình xử lý dữ liệu phức tạp thông qua các DAG (Directed Acyclic Graph). Nhờ khả năng mở rộng, linh hoạt và giao diện quản trị trực quan, Airflow giúp tự động hóa toàn bộ luồng xử lý dữ liệu từ trích xuất, chuyển đổi đến tải dữ liệu, đảm bảo tính nhất quán và khả năng kiểm soát cao cho hệ thống.

Extract: Các DAG Airflow (daily_extract_shopee_order, daily_extract_lazada_order, daily_extract_tiki_order, daily_extract_tiktok_order, daily_extract_website_order) được lên lịch chạy hàng ngày để gọi API và lấy dữ liệu JSON từ các nguồn.

Load: Dữ liệu JSON thô được lưu trữ trực tiếp vào MinIO theo cấu trúc thư mục được tổ chức rõ ràng theo kênh, loại dữ liệu, và thời gian (year/month/day). Các DAG transform (transform_shopee_order_to_parquet, transform_lazada_order_to_parquet, v.v.) sẽ chuyển đổi JSON thành Parquet và lưu vào tầng staging.

Transform: Các DAG clean (clean_order_data_shopee_with_duckdb, clean_order_data_lazada_with_duckdb, v.v.) sử dụng DuckDB để truy vấn và xử lý dữ liệu Parquet từ tầng staging, thực hiện việc làm sạch và chuẩn hóa, sau đó lưu kết quả vào tầng cleaned. Các truy vấn SQL được thiết kế để xử lý các vấn đề như định dạng ngày tháng, loại bỏ dữ liệu trùng lặp, và tính toán các trường phái sinh.

Thành phần lưu trữ dữ liệu:

MinIO: Đóng vai trò là kho lưu trữ đối tượng chính, với cấu trúc thư mục phân cấp:

raw/year={year}/month={month}/day={day}/{channel}/{data_model}/: Chứa dữ liệu JSON thô từ các nguồn.

staging/year={year}/month={month}/day={day}/{channel}/{data_model}/: Chứa dữ liệu đã được chuyển đổi sang Parquet nhưng chưa được làm sạch.

cleaned/year={year}/month={month}/day={day}/{channel}/{data_model}/: Chứa dữ liệu đã được làm sạch và chuẩn hóa, sẵn sàng cho phân tích.

DuckDB: Được sử dụng để truy vấn và xử lý dữ liệu trực tiếp từ MinIO thông qua extension httpfs, cho phép thực hiện các phép biến đổi phức tạp và hiệu quả trên dữ liệu Parquet mà không cần tải xuống toàn bộ dữ liệu. Các tác vụ như chuẩn hóa dữ liệu, tính toán các trường phái sinh như cost_price, và kết hợp dữ liệu từ nhiều nguồn được thực hiện thông qua các truy vấn SQL.

Thành phần trực quan hóa dữ liệu:

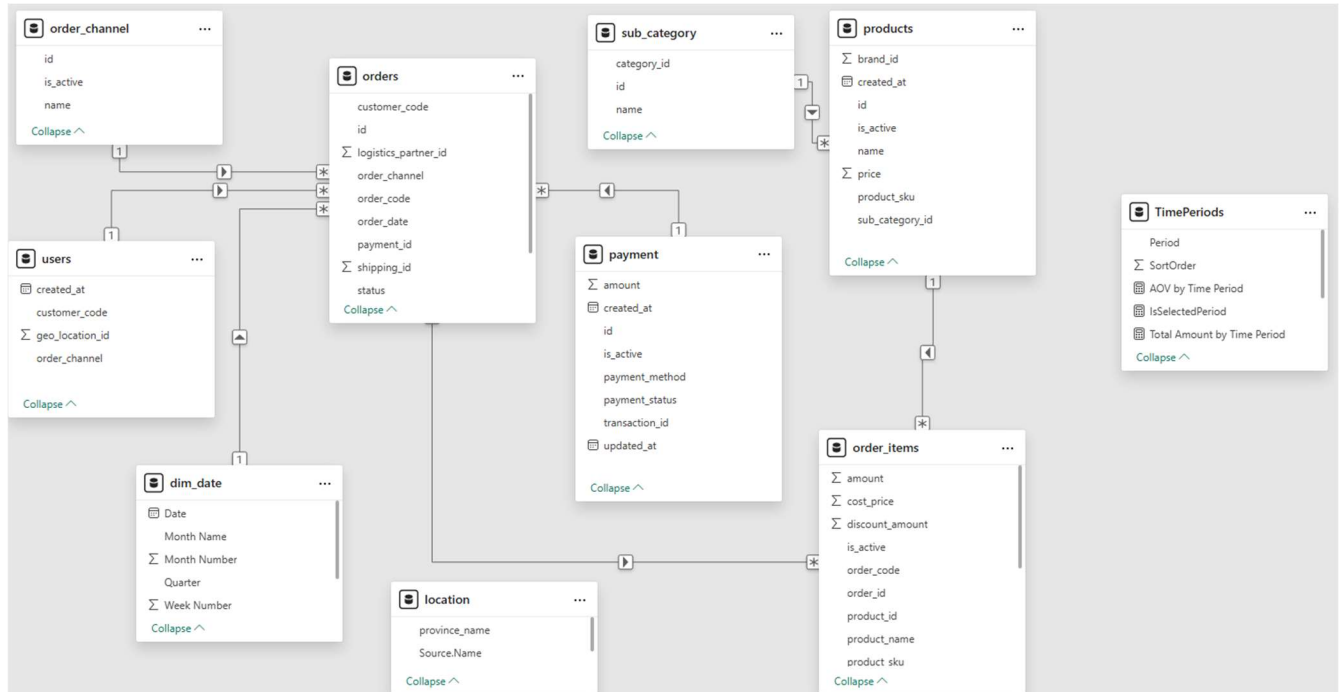
Power BI: Kết nối với các file Parquet đã được xử lý và lưu trong thư mục output của DuckDB để xây dựng các báo cáo và dashboard trực quan. Các dashboard bao gồm phân tích doanh thu theo kênh, theo thời gian, và theo danh mục sản phẩm.

3.3. Thiết kế cơ sở dữ liệu/kho dữ liệu

Dựa trên yêu cầu phân tích, mô hình dữ liệu Dimensional Modeling sẽ được áp dụng để thiết kế các bảng dữ liệu. Mô hình này phù hợp cho các tác vụ báo cáo và phân tích, với các bảng Fact (chứa các chỉ số đo lường) và các bảng Dimension (chứa thông tin mô tả).

3.3.1. Mô hình dữ liệu

Mô hình dữ liệu quan hệ:



Bảng Fact chính:

fact_orders:

Các trường: order_id, order_date, customer_id, order_channel_id, shipping_id, payment_id, logistics_partner_id, status, total_price, shipping_cost, profit

Mô tả: Bảng này chứa các chỉ số bán hàng chính từ các đơn hàng, được tích hợp từ tất cả các kênh bán hàng (Shopee, Lazada, Tiki, Tiktok, Website).

fact_order_items:

Các trường: order_item_id, order_id, product_id, quantity, unit_price, discount_amount, cost_price, amount, created_at

Mô tả: Bảng này chứa thông tin chi tiết về từng mặt hàng trong mỗi đơn hàng, cung cấp dữ liệu chi tiết cho phân tích doanh số theo sản phẩm.

Các bảng Dimension:

dim_customers:

Các trường: customer_id, customer_code, first_name, last_name, email, phone, geo_location_id, created_at

Mô tả: Chứa thông tin về khách hàng từ tất cả các kênh, đã được hợp nhất và làm sạch.

dim_products:

Các trường: product_id, product_sku, name, description, brand_id, category_id, sub_category_id, price, cost_price

Mô tả: Chứa thông tin về sản phẩm, với liên kết đến thương hiệu và danh mục.

dim_order_channels:

Các trường: order_channel_id, name, is_active

Mô tả: Chứa thông tin về các kênh bán hàng (Shopee, Lazada, Tiki, Tiktok, Website).

dim_dates:

Các trường: date_id, date, year, quarter, month, week, day, day_of_week, is_weekend, is_holiday

Mô tả: Bảng dimension chuẩn cho phân tích theo thời gian, hỗ trợ các truy vấn phân tích theo các khoảng thời gian khác nhau.

dim_geo_locations:

Các trường: geo_location_id, ward_name, district_name, city_name, region

Mô tả: Chứa thông tin về vị trí địa lý để phân tích phân bố khách hàng và hiệu suất bán hàng theo khu vực.

dim_payment_methods:

Các trường: payment_id, name, is_online, description

Mô tả: Chứa thông tin về phương thức thanh toán được sử dụng.

dim_logistics_partners:

Các trường: logistics_partner_id, name, geo_location_id, contact_info

Mô tả: Chứa thông tin về đối tác vận chuyển.

3.3.2. Chiến lược lưu trữ dữ liệu

Chiến lược lưu trữ dữ liệu là một trong những nền tảng quan trọng nhất của hệ thống Data Warehouse, ảnh hưởng trực tiếp đến hiệu suất truy vấn, chi phí lưu trữ và khả năng quản lý. Trong dự án này, một chiến lược đa lớp được thiết kế cẩn thận, kết hợp giữa định dạng file tối ưu và kỹ thuật phân vùng thông minh.

1. Lựa chọn Định dạng File: Parquet

Thay vì sử dụng các định dạng truyền thống như CSV hay JSON, dự án đã lựa chọn Parquet làm định dạng lưu trữ chính cho dữ liệu ở các tầng Staging và Cleaned. Đây là một quyết định có chủ đích dựa trên các ưu điểm vượt trội của Parquet trong các hệ thống phân tích dữ liệu lớn:

Lưu trữ theo cột (Columnar Storage): Đây là đặc tính quan trọng nhất. Không giống như CSV (lưu theo hàng), Parquet tổ chức dữ liệu theo từng cột. Khi thực hiện một truy vấn phân tích (ví dụ: tính tổng doanh thu), hệ thống chỉ cần đọc cột doanh_thu thay vì phải quét toàn bộ file. Điều này giúp giảm thiểu đáng kể I/O và tăng tốc độ truy vấn lên nhiều lần.

Nén dữ liệu hiệu quả: Parquet hỗ trợ nhiều thuật toán nén (như Snappy, Gzip, Brotli) và áp dụng chúng trên từng cột. Vì dữ liệu trong cùng một cột thường có tính tương đồng cao, tỷ lệ nén đạt được rất tốt, giúp tiết kiệm từ 60-80% không gian lưu trữ so với file không nén.

4
1

Hỗ trợ Schema Evolution: Cấu trúc (schema) của dữ liệu được lưu trữ ngay trong metadata của file Parquet. Điều này cho phép schema có thể thay đổi theo thời gian (ví dụ: thêm một cột mới) mà không làm hỏng các file dữ liệu cũ, đảm bảo tính linh hoạt cho hệ thống.

2

Tích hợp sâu với hệ sinh thái Big Data: Parquet là định dạng tiêu chuẩn và được hỗ trợ bởi hầu hết các công cụ xử lý dữ liệu hiện đại như DuckDB, Spark, Presto, và Power BI, đảm bảo khả năng tương tác và mở rộng trong tương lai.

2. Kỹ thuật Phân vùng Dữ liệu (Data Partitioning)

Để tránh tình trạng "quét toàn bộ" (full scan) trên một kho dữ liệu ngày càng phình to, dữ liệu được phân vùng một cách có chiến lược thành các thư mục logic. Các công cụ truy vấn như DuckDB có thể tận dụng cấu trúc này để "cắt tỉa" (partition pruning), tức là bỏ qua hoàn toàn các thư mục không chứa dữ liệu liên quan đến câu lệnh WHERE.

Dữ liệu trong MinIO được phân vùng theo các cấp sau:

Theo Mô hình dữ liệu (data_model): data_model=users, data_model=orders, data_model=products,... Đây là cấp phân vùng cao nhất, giúp tách biệt hoàn toàn các thực thể dữ liệu khác nhau.

Theo Kênh bán hàng (channel): channel=shopee, channel=lazada,... Giúp dễ dàng phân tích hiệu suất của từng kênh một cách độc lập.

Theo Thời gian (year, month, day): year=2025/month=09/day=20. Đây là cách phân vùng phổ biến và hiệu quả nhất, vì hầu hết các truy vấn phân tích kinh doanh đều có yếu tố thời gian (ví dụ: "doanh thu theo ngày", "so sánh tháng này với tháng trước").

Ví dụ về một đường dẫn đầy đủ trong MinIO:

```
datawarehouse/cleaned/year=2025/month=09/day=20/channel=shopee/data_model=orders/data.parquet
```

Sự kết hợp này cho phép các truy vấn được tối ưu hóa ở mức tối đa. Ví dụ, một truy vấn lấy doanh thu của Shopee trong 3 ngày đầu tháng 9 sẽ chỉ cần quét dữ liệu trong 3 thư mục con cụ thể, thay vì hàng trăm hoặc hàng nghìn thư mục khác.

3. Cấu trúc các Tầng lưu trữ trong MinIO

Hệ thống áp dụng kiến trúc đa tầng (multi-layered architecture) trong Data Warehouse, mỗi tầng có một vai trò và mục đích riêng:

Tầng raw:

Mục đích: Lưu trữ dữ liệu thô, nguyên bản nhất được trích xuất từ các hệ thống nguồn (API, Database).

Định dạng: Thường là JSON hoặc định dạng gốc của nguồn.

Vai trò: Đóng vai trò như một bản sao lưu (backup) vĩnh viễn. Nếu có lỗi trong các bước xử lý sau, ta luôn có thể chạy lại pipeline từ dữ liệu gốc ở tầng này mà không cần phải gọi lại API.

Tầng staging:

Mục đích: Chuyển đổi dữ liệu từ định dạng gốc sang định dạng Parquet.

Xử lý: Ở tầng này, chỉ thực hiện các chuyển đổi cơ bản nhất như ép kiểu dữ liệu (ví dụ: chuỗi sang số, chuỗi sang ngày tháng) và chuẩn hóa tên cột. Không áp dụng các logic nghiệp vụ phức tạp.

2

Vai trò: Là một tầng trung gian, chuẩn bị dữ liệu cho bước làm sạch sâu hơn, giúp tách biệt quá trình "load" và "transform".

Tầng cleaned:

Mục đích: Chứa dữ liệu đã được làm sạch, chuẩn hóa, và làm giàu (enriched), sẵn sàng cho việc phân tích.

Xử lý: Đây là nơi diễn ra các logic nghiệp vụ quan trọng: xử lý giá trị NULL, loại bỏ dữ liệu trùng lặp, kết hợp dữ liệu từ nhiều nguồn (join), tính toán các trường mới (ví dụ: order_total).

Vai trò: Là "nguồn chân lý duy nhất" (Single Source of Truth) cho tất cả các báo cáo và dashboard. Power BI và các công cụ phân tích khác sẽ kết nối trực tiếp vào tầng này.

Việc tổ chức dữ liệu theo cấu trúc nhiều tầng và phân vùng chặt chẽ không chỉ giúp tối ưu hóa hiệu suất mà còn tăng cường khả năng quản trị, bảo mật và đảm bảo chất lượng dữ liệu, tạo nền tảng vững chắc cho hệ thống phát triển trong tương lai.

IV. Triển khai và cài đặt hệ thống

4.1. Môi trường phát triển và công cụ

Hệ thống được triển khai trong một môi trường container hóa (containerized) hoàn toàn bằng Docker và Docker Compose. Cách tiếp cận này đảm bảo tính nhất quán giữa môi trường phát triển và sản xuất, đồng thời cho phép cô lập các thành phần và dễ dàng quản lý, mở rộng.

Các công nghệ và công cụ chính được sử dụng:

Python 3.10: Ngôn ngữ lập trình chính cho toàn bộ hệ thống, từ việc xây dựng API giả lập, viết các script xử lý dữ liệu cho đến định nghĩa các DAG trong Airflow.

Apache Airflow 2.7.1: Đóng vai trò là "bộ não" của hệ thống, chịu trách nhiệm điều phối và tự động hóa toàn bộ quy trình ELT. Airflow được triển khai với CeleryExecutor để có khả năng xử lý đồng thời nhiều tác vụ, tăng hiệu suất cho các pipeline.

MinIO: Một hệ thống lưu trữ đối tượng (object storage) hiệu suất cao và tương thích hoàn toàn với API của Amazon S3. MinIO được sử dụng làm kho dữ liệu trung tâm, lưu trữ dữ liệu ở tất cả các tầng (raw, staging, cleaned).

DuckDB: Một công cụ phân tích dữ liệu cột tại chỗ (in-process columnar database) mạnh mẽ. DuckDB được sử dụng để thực hiện các phép biến đổi (transform) phức tạp trực tiếp trên các file Parquet lưu trữ trong MinIO mà không cần một máy chủ cơ sở dữ liệu riêng biệt, giúp tối ưu hóa hiệu suất và đơn giản hóa kiến trúc.

FastAPI: Một framework Python hiện đại, hiệu suất cao để xây dựng API. Trong dự án này, FastAPI được dùng để tạo ra một hệ thống API giả lập, mô phỏng các endpoint từ các sản phẩm thương mại điện tử, cung cấp nguồn dữ liệu đầu vào cho pipeline.

Power BI: Công cụ trực quan hóa dữ liệu hàng đầu của Microsoft, được sử dụng để kết nối vào dữ liệu đã được làm sạch, xây dựng các mô hình dữ liệu và tạo ra các dashboard phân tích kinh doanh tương tác.

Các dịch vụ (services) được triển khai qua Docker Compose:

Toàn bộ hệ thống được định nghĩa và quản lý thông qua file docker-compose.yml, bao gồm các dịch vụ chính sau:

Postgres: Container chạy cơ sở dữ liệu PostgreSQL, được sử dụng làm backend để lưu trữ metadata (siêu dữ liệu) cho Airflow, chẳng hạn như thông tin về các DAG, lịch sử các lần chạy, và trạng thái của các task.

Redis: Container chạy Redis, đóng vai trò là message broker cho CeleryExecutor của Airflow, giúp điều phối và gửi các tác vụ đến các worker.

Airflow-webserver: Chạy giao diện người dùng web của Airflow, cho phép người dùng theo dõi, quản lý và tương tác với các DAG.

Airflow-scheduler: Thành phần cốt lõi của Airflow, chịu trách nhiệm lập lịch và đưa các DAG vào hàng đợi để thực thi khi đến thời điểm được định sẵn.

Airflow-worker: Các tiến trình thực thi tác vụ (task) của Airflow. Hệ thống có thể được mở rộng bằng cách tăng số lượng worker để xử lý nhiều tác vụ song song.

MinIO: Chạy dịch vụ lưu trữ MinIO, cung cấp các "bucket" để lưu trữ dữ liệu của pipeline.

Fastapi-app: Chạy ứng dụng FastAPI, cung cấp các API endpoint giả lập dữ liệu.

Flower: Một công cụ giám sát cho Celery, cung cấp giao diện web để theo dõi trạng thái của các worker và các tác vụ đang được xử lý.

4.2. Xây dựng Data Pipeline

4.2.1. Giai đoạn Extract

Hệ thống sử dụng các DAG Airflow chuyên biệt cho từng kênh bán hàng và loại dữ liệu:

daily_extract_shopee_order: Trích xuất dữ liệu đơn hàng từ Shopee.

daily_extract_lazada_order: Trích xuất dữ liệu đơn hàng từ Lazada.

daily_extract_tiki_order: Trích xuất dữ liệu đơn hàng từ Tiki.

daily_extract_tiktok_order: Trích xuất dữ liệu đơn hàng từ Tiktok Shop.

daily_extract_website_order: Trích xuất dữ liệu đơn hàng từ website riêng.

Mỗi DAG chứa hai task chính:

Task **fetch_json_from_api**: Kết nối đến API endpoint tương ứng và lấy dữ liệu JSON về.

Task **save_raw_json_to_minio**: Lưu trữ dữ liệu JSON vào MinIO theo cấu trúc thư mục được định nghĩa.

Mã ví dụ từ DAG extract_order.py:

```
fetch_json_task = PythonOperator(
    task_id='fetch_json_from_api',
    python_callable=fetch_json_from_api,
)

save_to_minio_task = PythonOperator(
    task_id='save_raw_json_to_minio',
    python_callable=save_raw_json_to_minio,
)
```

4.2.2. Giai đoạn Load

Sau khi dữ liệu JSON thô được lưu vào MinIO, các DAG transform được kích hoạt tự động để chuyển đổi dữ liệu từ JSON sang Parquet:

transform_shopee_order_to_parquet

transform_lazada_order_to_parquet

transform_tiki_order_to_parquet

```
transform_tiktok_order_to_parquet
transform_website_order_to_parquet
```

Mỗi DAG transform bao gồm các task:

1. **get_yesterday_file_paths**: Xác định các file JSON cần xử lý.
2. **download_all_json_files**: Tải các file JSON từ MinIO.
3. **parse_json_to_table**: Phân tích cấu trúc JSON và chuyển đổi thành dạng bảng.
4. **convert_to_parquet**: Chuyển đổi dữ liệu sang định dạng Parquet và lưu vào tầng staging của MinIO.

Ví dụ mã từ DAG staging_order.py:

```
parse_json = PythonOperator(
    task_id='parse_json_to_table',
    python_callable=parse_json_to_table,
)

convert_to_parquet = PythonOperator(
    task_id='convert_to_parquet',
    python_callable=convert_to_parquet_and_save
)
```

4.2.3. Giai đoạn Transform

Dữ liệu Parquet từ tầng staging được xử lý tiếp bởi các DAG clean để làm sạch và chuẩn hóa:

```
clean_order_data_shopee_with_duckdb
clean_order_data_lazada_with_duckdb
clean_order_data_tiki_with_duckdb
clean_order_data_tiktok_with_duckdb
clean_order_data_website_with_duckdb
```

Mỗi DAG clean thực hiện:

1. **Đọc dữ liệu từ tầng staging bằng DuckDB.**
2. **Thực hiện các phép biến đổi SQL để:**
 - Chuẩn hóa định dạng ngày tháng (created_at, order_date).
 - Lọc bỏ các bản ghi trùng lặp.
 - Tính toán các trường phái sinh (như cost_price dựa trên unit_price và discount_amount).
3. **Lưu dữ liệu đã làm sạch vào tầng cleaned của MinIO.**

Ví dụ mã SQL từ DuckDB để xử lý dữ liệu đơn hàng:

```
SELECT
    id,
    status,
    CASE
        WHEN TRY_CAST(created_at AS DOUBLE) IS NOT NULL
            AND CAST(created_at AS DOUBLE) BETWEEN 0 AND 32503680000 THEN
```

```

        CAST(to_timestamp(CAST(created_at AS DOUBLE)) AT TIME ZONE 'UTC' AS VARCHAR)
    ELSE CAST(created_at AS VARCHAR)
END AS created_at,
order_code,
order_channel,
payment_id,
customer_code,
shipping_id,
total_price,
logistics_partner_id
FROM read_parquet('{path}', union_by_name=True)
ORDER BY created_at

```

Sau giai đoạn clean, một script Python được thực thi để đọc dữ liệu đã làm sạch từ MinIO, kết hợp dữ liệu từ nhiều nguồn và tạo các bảng tổng hợp cuối cùng. Các truy vấn SQL trong DuckDB được sử dụng để thực hiện các tính toán phức tạp và tạo ra các chỉ số kinh doanh cần thiết.

4.2.4. Đảm bảo chất lượng dữ liệu và kiểm tra tính hợp lệ

Để đảm bảo chất lượng dữ liệu trong hệ thống Data Warehouse, một quy trình kiểm tra và xác thực dữ liệu đã được triển khai trong các giai đoạn xử lý:

Kiểm tra tính hợp lệ của dữ liệu thô:

Các kiểm tra ban đầu được thực hiện khi dữ liệu JSON được trích xuất từ API

Kiểm tra cơ bản bao gồm xác minh cấu trúc JSON đúng định dạng và kiểm tra số lượng bản ghi

Validation trong Power Query:

Sử dụng Power Query Editor trong Power BI để thực hiện các kiểm tra và biến đổi dữ liệu

Các quy tắc xác thực được áp dụng thông qua các bước chuyển đổi trong Power Query:

Kiểm tra kiểu dữ liệu: tự động phát hiện và chuyển đổi kiểu dữ liệu cho các cột

Loại bỏ giá trị null: xác định và xử lý các giá trị null theo chiến lược phù hợp

Kiểm tra phạm vi giá trị: lọc các bản ghi với giá trị nằm ngoài phạm vi hợp lý

Xác thực mối quan hệ: kiểm tra tính toàn vẹn tham chiếu giữa các bảng

Xử lý dữ liệu không hợp lệ:

Tạo các bước chuyển đổi tùy chỉnh trong Power Query để xử lý các trường hợp ngoại lệ

Các chiến lược xử lý dữ liệu không hợp lệ bao gồm:

Thay thế giá trị không hợp lệ bằng giá trị mặc định

Đánh dấu bản ghi có dữ liệu không hợp lệ để kiểm tra thêm

Loại bỏ các bản ghi không đáp ứng các ràng buộc nghiệp vụ quan trọng

Ví dụ mã M trong Power Query để kiểm tra tính hợp lệ của đơn hàng:

```

let
    Source = ... , // nguồn dữ liệu
    #"Kiểm tra tính hợp lệ" = Table.AddColumn(Source, "Valid", each if [total_price] >= 0
and [order_date] <> null then true else false),

```

```
#"Lọc dữ liệu không hợp lệ" = Table.SelectRows(#"Kiểm tra tính hợp lệ", each [Valid] = true)
in
#"Lọc dữ liệu không hợp lệ"
```

4.3. Xây dựng Dashboard trực quan hóa

4.3.1. Kế hoạch thiết kế và phát triển dashboard

Việc xây dựng các dashboard trong Power BI được thực hiện theo một quy trình có kế hoạch, bắt đầu từ việc xác định nhu cầu phân tích, thu thập và chuẩn bị dữ liệu, đến thiết kế và triển khai các dashboard tương tác. Dưới đây là kế hoạch chi tiết cho từng dashboard:

a) Quy trình chung cho tất cả dashboard:

Giai đoạn chuẩn bị dữ liệu:

Kết nối Power BI với dữ liệu Parquet từ thư mục output của DuckDB

Thiết lập quan hệ (relationships) giữa các bảng dữ liệu

Xác định các chỉ số (measures) và cột được tính toán (calculated columns) cần thiết

Thiết lập phân cấp dữ liệu (hierarchies) để hỗ trợ drill-down (ví dụ: Year > Quarter > Month > Day)

Thiết kế trực quan:

Lựa chọn bảng màu nhất quán cho toàn bộ dashboard

Sử dụng nguyên tắc thiết kế trực quan hiệu quả (data-ink ratio, nổi bật thông tin quan trọng)

Tạo các bộ lọc (slicers) chung

Kiểm thử và hoàn thiện:

Kiểm tra tính chính xác của dữ liệu trên dashboard

Tối ưu hóa hiệu suất truy vấn

b) Kế hoạch chi tiết cho từng dashboard:

Sales Performance Dashboard:

Mục tiêu: Cung cấp **cái nhìn tổng quan và chi tiết về hiệu suất bán hàng** trên tất cả các kênh.

Nguồn dữ liệu:

fact_orders, fact_order_items từ tầng cleaned

dim_products, dim_order_channels, dim_dates

Các chỉ số chính cần tính toán:

Tổng doanh thu = SUM(fact_orders[total_price])

Lợi nhuận = SUM(fact_orders[profit])

Tỷ suất lợi nhuận = [Lợi nhuận] / [Tổng doanh thu]

Giá trị đơn hàng trung bình = [Tổng doanh thu] / COUNT(fact_orders[order_id])

Các visual chính:

Card hiển thị tổng doanh thu, lợi nhuận, số đơn hàng

Line chart hiển thị xu hướng doanh thu theo thời gian với drill-down từ năm đến ngày

Bar chart so sánh doanh thu giữa các kênh bán hàng

Donut chart thể hiện tỷ lệ doanh thu theo danh mục sản phẩm

Bộ lọc (slicers): Khoảng thời gian, kênh bán hàng

4.4. Triển khai API mô phỏng với FastAPI

Để phục vụ cho việc phát triển và thử nghiệm hệ thống mà không phụ thuộc vào API thực tế của các sàn thương mại điện tử, một hệ thống API mô phỏng đã được xây dựng bằng FastAPI. Hệ thống này cung cấp dữ liệu giả lập với cấu trúc tương tự như API thực từ Shopee, Lazada, Tiki, và các nguồn dữ liệu khác.

4.4.1. Tính năng chính của API mô phỏng

Tạo dữ liệu giả lập có tính thực tế:

Sử dụng thư viện Faker để tạo dữ liệu mô phỏng thông tin khách hàng, sản phẩm

Áp dụng các quy tắc kinh doanh như giá không âm, trạng thái đơn hàng hợp lệ

Tùy chỉnh dữ liệu theo tham số:

Hỗ trợ lọc theo khoảng thời gian

Phân trang dữ liệu

Tùy chỉnh số lượng bản ghi trả về

Mô phỏng lỗi và trường hợp ngoại lệ:

Endpoint đặc biệt để mô phỏng lỗi máy chủ, timeout

Tùy chọn trả về dữ liệu không hợp lệ hoặc thiếu thông tin để kiểm tra khả năng xử lý ngoại lệ của hệ thống

Tài liệu tương tác:

Tài liệu OpenAPI/Swagger được tự động tạo và có thể truy cập tại /docs

Mô tả chi tiết từng endpoint, tham số, và cấu trúc dữ liệu trả về

API mô phỏng này đã chứng minh giá trị trong quá trình phát triển, cho phép đội phát triển thử nghiệm các kịch bản khác nhau, bao gồm cả xử lý ngoại lệ và các trường hợp biên, mà không cần phụ thuộc vào các API thực tế từ các sàn TMĐT.

4.5. Containerization và triển khai với Docker

Hệ thống Data Warehouse được đóng gói và triển khai bằng Docker, mang lại nhiều lợi ích về tính nhất quán, dễ mở rộng và di động. Chi tiết về cấu hình và triển khai như sau:

4.5.1. Cấu trúc Docker

Hệ thống được tổ chức thành nhiều container riêng biệt, mỗi container chịu trách nhiệm cho một phần của hệ thống:

airflow-webserver và airflow-scheduler: Chạy Apache Airflow, điều phối và tự động hóa các quy trình ETL.

airflow-worker: Thực thi các task Airflow phân tán.

postgres: Cơ sở dữ liệu lưu trữ metadata và lịch sử chạy của Airflow.

redis: Broker message queue cho Airflow CeleryExecutor.

minio: Dịch vụ lưu trữ đối tượng tương thích S3.

fastapi: Chạy API mô phỏng.

flower: Giám sát Celery workers.

4.5.2. Docker Compose

File `docker-compose.yml` được sử dụng để định nghĩa và quản lý toàn bộ các dịch vụ của hệ thống. Nó thiết lập các container, định nghĩa các biến môi trường cần thiết (như thông tin kết nối MinIO, Airflow), và quản lý mạng lưới (networking) để các container có thể giao tiếp với nhau một cách an toàn. Với một lệnh duy nhất (`docker-compose up -d`), toàn bộ hệ thống có thể được khởi chạy.

4.5.3. Lợi ích của containerization

Tính nhất quán: Môi trường phát triển và sản xuất giống nhau, giảm thiểu các vấn đề "works on my machine".

Dễ mở rộng: Có thể dễ dàng mở rộng bằng cách thêm worker nodes.

Cô lập tài nguyên: Mỗi thành phần hoạt động trong môi trường cô lập riêng.

Quản lý phụ thuộc: Các phụ thuộc và cấu hình được quản lý tập trung.

Dễ dàng triển khai: Một lệnh duy nhất (`docker-compose up -d`) để khởi động toàn bộ hệ thống.

Việc containerization với Docker không chỉ giúp đơn giản hóa quá trình phát triển và triển khai mà còn tạo nền tảng vững chắc cho việc mở rộng hệ thống trong tương lai, khi khối lượng dữ liệu và yêu cầu xử lý tăng lên.

4.6. Xử lý lỗi và giám sát hệ thống

Để đảm bảo tính ổn định của pipeline, hệ thống tận dụng các cơ chế xử lý lỗi và giám sát được tích hợp sẵn trong Apache Airflow.

4.6.1. Cơ chế xử lý lỗi trong Data Pipeline

Chiến lược Thử lại (Retry):

Mỗi tác vụ (task) trong các DAG đều được cấu hình với cơ chế tự động thử lại khi gặp lỗi. Các tham số như retries (số lần thử lại) và retry_delay (khoảng thời gian giữa các lần thử) được thiết lập để xử lý các lỗi tạm thời, chẳng hạn như mất kết nối mạng hoặc API bị quá tải.

Điều này giúp tăng khả năng phục hồi của pipeline mà không cần sự can thiệp thủ công.

Xử lý ngoại lệ (Exception Handling):

Bên trong mã nguồn của các tác vụ Python, các khối try...except được sử dụng để bắt và xử lý các lỗi cụ thể có thể lường trước.

Khi một lỗi xảy ra, nó sẽ được ghi lại (log) một cách chi tiết, giúp cho việc gỡ lỗi sau này trở nên dễ dàng hơn. Airflow sẽ đánh dấu tác vụ là failed và dừng các tác vụ phụ thuộc, ngăn ngừa lỗi lan truyền.

4.6.2. Giám sát hệ thống

Việc giám sát chủ yếu được thực hiện thông qua giao diện người dùng web của Airflow, cung cấp một cái nhìn tổng quan và chi tiết về sức khỏe của toàn bộ hệ thống:

Theo dõi trạng thái DAG và Task: Giao diện Airflow cho phép theo dõi trạng thái (running, success, failed, skipped) của từng DAG và từng tác vụ trong đó một cách trực quan.

Xem lại lịch sử: Cung cấp lịch sử chi tiết về các lần chạy của DAG, thời gian bắt đầu, thời gian kết thúc và thời gian thực thi.

Truy cập Logs: Đối với mỗi lần chạy tác vụ, người dùng có thể truy cập trực tiếp vào file log chi tiết để kiểm tra kết quả đầu ra hoặc điều tra nguyên nhân gây lỗi.

Cách tiếp cận này giúp đơn giản hóa việc quản lý và bảo trì hệ thống, tập trung vào các công cụ có sẵn mà không cần triển khai thêm các hệ thống giám sát phức tạp từ bên ngoài.

V. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết luận

Tóm tắt kết quả đạt được:

Đã thiết kế và triển khai thành công một hệ thống kho dữ liệu hiện đại cho việc phân tích dữ liệu bán hàng đa nền tảng, tích hợp dữ liệu từ nhiều sàn thương mại điện tử (Shopee, Lazada, Tiki, Tiktok Shop) và website riêng của doanh nghiệp.

Xây dựng thành công kiến trúc ELT hoàn chỉnh với các tầng dữ liệu raw, staging và cleaned, giúp duy trì tính toàn vẹn và truy xuất nguồn gốc dữ liệu.

Triển khai các pipeline tự động bằng Apache Airflow, xử lý dữ liệu với DuckDB, và lưu trữ hiệu quả trong MinIO với định dạng Parquet.

7 Phát triển các dashboard trực quan trong Power BI, cung cấp cái nhìn tổng quan về hiệu suất kinh doanh của doanh nghiệp.

Lợi ích của giải pháp:

1 **Tự động hóa:** Giảm thiểu thời gian và công sức cho việc tổng hợp dữ liệu từ các nguồn khác nhau, thay thế quy trình thủ công bằng pipeline tự động.

2 **Tính nhất quán:** Đảm bảo dữ liệu được xử lý và trình bày một cách nhất quán, độ tin cậy cao thông qua các quy trình kiểm soát chất lượng dữ liệu.

Chi phí hiệu quả: Sử dụng công nghệ mã nguồn mở (Airflow, DuckDB, MinIO) giúp tiết kiệm chi phí đáng kể so với các giải pháp thương mại.

2 **Khả năng mở rộng:** Thiết kế theo kiến trúc module, cho phép dễ dàng tích hợp thêm nguồn dữ liệu mới hoặc thay đổi công nghệ trong từng thành phần.

Thách thức và giải pháp:

Đa dạng nguồn dữ liệu: Mỗi sản phẩm có cấu trúc dữ liệu khác nhau, đã giải quyết thông qua các bước chuẩn hóa dữ liệu và thiết kế mô hình chiều.

Đảm bảo chất lượng dữ liệu: Triển khai các cơ chế kiểm tra và xác thực tại nhiều tầng, từ raw data đến cleaned data và trong Power Query.

Hiệu suất xử lý: Tối ưu hóa quy trình ELT với việc sử dụng định dạng Parquet, kỹ thuật phân vùng dữ liệu, và tận dụng sức mạnh xử lý của DuckDB.

Triển khai đơn giản: Sử dụng Docker giúp đơn giản hóa quá trình triển khai và bảo trì hệ thống.

5.2 Hướng phát triển

Mở rộng nguồn dữ liệu:

2 **Tích hợp thêm dữ liệu từ các sàn thương mại điện tử khác và các kênh bán hàng truyền thống.**

4 Bổ sung dữ liệu từ các hệ thống CRM, ERP, và các ứng dụng nội bộ khác để có cái nhìn toàn diện hơn về hoạt động kinh doanh.

4 **Tích hợp dữ liệu mạng xã hội và dữ liệu marketing để phân tích hiệu quả của các chiến dịch quảng cáo.**

Nâng cao phân tích dữ liệu:

Nghiên cứu sâu hơn về các nghiệp vụ kinh doanh để xác định nhu cầu phân tích thực tế của từng phòng ban.

Xây dựng thêm nhiều dashboard chuyên biệt phục vụ các mục tiêu quản trị, vận hành, marketing, tài chính, v.v.

1 **Tối ưu hóa giao diện và chức năng dashboard để đáp ứng tốt hơn nhu cầu khai thác dữ liệu của người dùng cuối.**

Tự động hóa và tích hợp thông minh:

1 **Phát triển hệ thống cảnh báo tự động dựa trên ngưỡng KPI định trước.**

Tích hợp các công cụ quản lý metadata và data lineage để theo dõi nguồn gốc và sự biến đổi của dữ liệu.

Xây dựng các quy trình tự động cho việc kiểm tra chất lượng dữ liệu, bao gồm giám sát độ trễ, tính đầy đủ và tính chính xác.

Nâng cấp hệ thống lên kiến trúc data lakehouse:

Chuyển đổi hệ thống từ mô hình data warehouse truyền thống sang kiến trúc data lakehouse hiện đại, kết hợp ưu điểm của data lake và data warehouse.

Triển khai các giải pháp lưu trữ và quản lý dữ liệu như Apache Iceberg, Delta Lake để hỗ trợ versioning, ACID và quản lý metadata tốt hơn.

1 Tích hợp các công cụ như dbt (data build tool) để quản lý, kiểm soát và tài liệu hóa các phép biến đổi dữ liệu một cách hiệu quả trên nền tảng lakehouse.

Mở rộng khả năng xử lý dữ liệu thời gian thực (real-time processing) và batch trên cùng một nền tảng, phục vụ đa dạng nhu cầu phân tích và khai thác dữ liệu.

Cải thiện trải nghiệm người dùng:

Phát triển giao diện self-service để người dùng có thể tự tạo báo cáo và truy vấn dữ liệu mà không cần kiến thức kỹ thuật chuyên sâu.

Thêm các tính năng cảnh báo tự động khi có biến động bất thường trong các chỉ số kinh doanh.

Xây dựng các dashboard chuyên sâu hơn cho các phòng ban khác nhau (marketing, vận hành, tài chính, v.v.).

2 Với khả năng mở rộng và tính linh hoạt của kiến trúc hiện tại, giải pháp này có tiềm năng áp dụng rộng rãi không chỉ cho các doanh nghiệp thương mại điện tử mà còn cho nhiều lĩnh vực kinh doanh khác có nhu cầu tích hợp và phân tích dữ liệu từ nhiều nguồn khác nhau.

Tài liệu tham khảo

Liệt kê đầy đủ các tài liệu, sách, báo khoa học, website đã tham khảo theo định dạng chuẩn.

<https://aws.amazon.com/vi/compare/the-difference-between-etl-and-elt/>

Phụ lục (nếu có)

Mã nguồn: <https://github.com/ngoctam033/DWH.git>