Design and Implementation of an ETL Pipeline and Data Warehouse for E-Commerce Business Insights

Walid K. W. Alsafadi, Ameer T. F. Alzerei, Hamza M. H. Obaid, and Hazem A. A. Muanes
Department of Computer Engineering, University College of Applied Sciences, Gaza City, Palestine

***Abstract*— This paper presents the design and implementation of an ETL (Extract, Transform, Load) pipeline and a data warehouse to enhance e-commerce analytics. The project automates data ingestion from external APIs, transforms raw data into structured formats, and loads it into a PostgreSQL data warehouse. A star schema design was selected to optimize query performance and facilitate efficient business intelligence reporting. Python was employed for data extraction, transformation, and loading, ensuring seamless integration and data management. The resulting data warehouse supports comprehensive analysis, enabling insights into customer behavior, product performance, and sales trends. Key challenges included handling nested JSON structures, ensuring data type consistency, and navigating API limitations. The final solution demonstrated effective data processing, yielding valuable business insights, with future enhancements planned through schema extensions and integration of additional data sources.**

***Index Terms*—ETL Pipeline, Data Warehouse, E-Commerce Analytics, PostgreSQL, Star Schema, Data Engineering, API Integration, Business Intelligence, Data Transformation, Data Ingestion.**

I. INTRODUCTION

Data plays a critical role in the success of modern e-commerce businesses. The ability to collect, process, and analyze vast amounts of transactional and customer data can lead to actionable insights that drive strategic decisions. However, the raw data collected from various sources is often unstructured or inconsistent, making it difficult to derive meaningful conclusions without significant processing. This challenge necessitates robust data engineering workflows that efficiently transform raw data into usable formats for analysis.

This paper presents the development of an ETL (Extract, Transform, Load) pipeline and a data warehouse designed specifically for e-commerce analytics. The primary objective is to create a seamless data engineering workflow that ingests data from the Fake Store API, transforms it into a structured format, and stores it in a PostgreSQL data warehouse designed using a star schema. This schema design aligns with data warehousing best practices, optimizing query performance and simplifying data analysis.

The ETL process consists of three key stages. First, data is extracted from various API endpoints, including products, users, categories, and carts. This raw data, initially stored in JSON format, undergoes transformation to ensure consistency in data types, normalization of nested structures, and the calculation of key metrics for fact tables. The final stage involves loading the cleaned and processed data into the data warehouse, ensuring referential integrity across dimension and fact tables.

Throughout this paper, we discuss the methodology used in data ingestion, transformation, and schema design. We also highlight challenges encountered during data transformation—such as handling nested JSON data and aligning data with the star schema—and the solutions implemented to address them. The insights generated from the data warehouse, such as identifying top-selling products, high-spending customers, and category-wise revenue distribution, are presented in the results section. Key SQL queries and code snippets are provided in Appendix A and Appendix B, respectively, for further reference.

This project not only demonstrates technical proficiency in data engineering but also illustrates the potential of data-driven decision-making in the e-commerce domain.

II. METHODOLOGY

To design and implement the ETL pipeline and data warehouse for e-commerce analytics, a systematic approach was employed, divided into three key stages: data ingestion, data transformation, and data warehouse schema design. Each stage played a critical role in ensuring the data's accuracy, consistency, and readiness for analysis.

A. Data Ingestion

The data ingestion process involves extracting data from the Fake Store API, which provides endpoints for products, categories, users, and carts. The data was retrieved in JSON format and saved locally for further processing. The ingestion.py script (see **Appendix B**) automates this process by fetching data from each endpoint and storing it in the data/raw directory.

The following API endpoints were utilized:

- **Products**: Information on titles, categories, prices, descriptions, images, and ratings.
- **Users**: Customer details including personal information and addresses.
- **Carts**: Transaction history with product IDs, quantities, and dates.

- **Categories**: Classification of products into categories.

B. Data Transformation

Once the raw data was ingested, it underwent transformation to ensure consistency and compatibility with the data warehouse schema. Key transformation steps included:

1. **Converting Nested JSON Structures:** User and product data contained nested fields that were flattened to simplify analysis.
2. **Data Type Standardization:** Ensuring numerical fields like prices and quantities were correctly formatted as floats and integers, respectively.
3. **Creating Dimension and Fact Tables:** The transformed data was organized into dimension tables (e.g., products, categories, users) and a central fact table (sales) to support analytical queries.

The transformation process is automated using the transformation.py script, with detailed steps and code snippets available in **Appendix B**.

C. Data Warehouse Design

A **star schema** was chosen for the data warehouse due to its simplicity and efficiency in handling analytical queries. The schema consists of:

- **Dimension Tables:**
  - **Products:** Product details including title, price, category, and rating.
  - **Users:** Customer information linked to the address dimension.
  - **Categories:** Classification of products.
  - **Cart:** Cart ID and date
- **Fact Table:** sales_fact_table, which records transactional data, including product sales, quantities, and revenue.

The schema was implemented in PostgreSQL, and the SQL scripts used for schema creation are provided in **Appendix C**. An Entity-Relationship Diagram (ERD) illustrating the schema design is shown in **Figure 1** below.

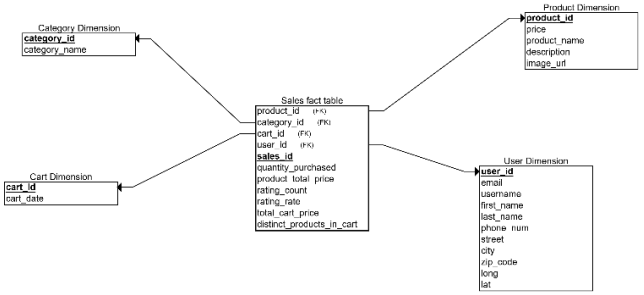

Fig. 1. Star Schema ERD for the E-Commerce Data Warehouse

III. Challenges and Solutions

Throughout the development of the ETL pipeline and data warehouse, several challenges emerged, particularly during the data ingestion and transformation phases. This section outlines the primary obstacles encountered and the solutions implemented to overcome them.

A. API Limitations

One of the initial challenges was the **limited availability of free APIs** that provided comprehensive e-commerce data suitable for this project. Many APIs either had restrictive data access policies or lacked the depth required for constructing a meaningful data warehouse. After extensive research, the **Fake Store API** was selected as it provided sufficient product, user, and transaction data to simulate a real-world e-commerce environment. Although this API offered a simplified dataset, it allowed for effective demonstration of ETL processes and data warehousing principles.

B. Data Transformation Complexities

The raw data retrieved from the API presented several **transformation challenges**:

1. **Nested JSON Structures:** The rating field in the products dataset was stored as a nested JSON object, requiring extraction and normalization.
2. **Inconsistent Data Types:** Certain fields, such as latitude and longitude, were stored as strings instead of numerical types, leading to issues during data analysis.
3. **Fact Table Calculations:** Populating the fact table required deriving additional columns, such as calculating **total sales** from **quantity** and **price**.

The data transformation phase was handled using **Python (Pandas)** for data manipulation. Key steps included:

- **Flattening Nested JSON:** The rating field was split into separate rate and count columns. The script for this transformation is provided in *Appendix B*.
- **Data Type Conversion:** String-based numerical fields were converted to appropriate data types (e.g., float for geographic coordinates).
- **Derived Columns:** New columns were created in the fact table by performing calculations on existing data (e.g., $total\_amount = quantity \times price$). Full transformation scripts can be reviewed in *Appendix B.*

C. Schema Design Alignment

Designing a schema that effectively **aligned with the retrieved data** and supported efficient analytical queries posed another challenge. The initial data did not perfectly match a traditional star schema format, requiring adjustments to both the schema design and the data. A **star schema** was selected due to its simplicity and superior performance for **OLAP (Online Analytical Processing)** queries. The schema was iteratively refined to ensure that the fact and dimension tables were properly normalized and connected via foreign keys. This alignment ensured the warehouse could address key **business questions** such as **sales trends**, **top-performing products**, and **customer behavior**. The final schema structure is provided in *Appendix C*.

IV. Results and Insights

This section presents the key insights derived from the e-commerce data warehouse. The results are based on SQL queries executed and visualized within **PostgreSQL**. The insights focus on identifying top-performing products, analyzing customer spending behavior, and understanding category-wise revenue distribution. The SQL queries used for these insights are provided in *Appendix A*.

A. Top 5 Products by Total Sales Revenue

To identify the top-performing products, sales data was aggregated to calculate the total revenue per product. This required joining the **products** dimension table with the **sales** fact table to compute the revenue contributions. The SQL query used for this analysis can be found in *Appendix A, Query 1*. The top five products with the highest revenue are presented in **Table I** and visualized in **Fig. 2**.

TABLE I
TOP 5 PRODUCTS BY TOTAL SALES REVENUE

Product Name	Description	Total Revenue ($)
Fjallraven - Foldsack No. 1 Backpack	Your perfect pack for everyday use and walks in the forest.	2199.00
John Hardy Women's Legends Naga Gold & Silver Bracelet	From our Legends Collection, the Naga was inspired by the mythical water dragon.	1390.00
WD 4TB Gaming Drive Works with Playstation 4 Portable	Expand your PS4 gaming experience, Play anywhere.	342.00
Men's Cotton Jacket	Great outerwear jackets for Spring/Autumn/Winter, suitable for many occasions.	335.94
SanDisk SSD PLUS 1TB Internal SSD	Easy upgrade for faster boot up, shutdown, application load, and response.	218.00

The **Fjallraven backpack** emerged as the highest-grossing product with a total revenue of **$2,199.00**. High-value fashion items like the **John Hardy bracelet** also showed significant sales, indicating a strong interest in premium products.

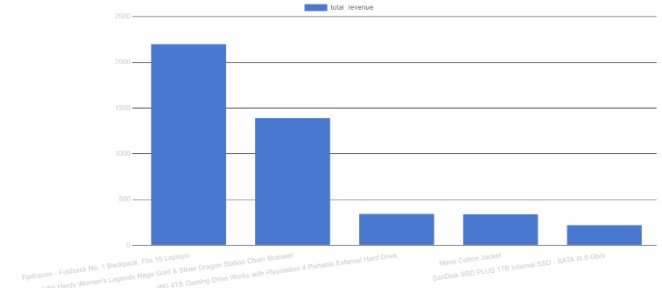

Fig. 2. Top 5 Products by Total Sales Revenue.

B. Top Customers by Total Spending

To identify the most valuable customers, the total amount spent by each customer was calculated. This analysis involved joining the **users** dimension table with the **carts** fact table and aggregating purchase data. The corresponding SQL query is available in *Appendix A, Query 2*. The top customers by total spending are presented in **Table II** and visualized in **Fig. 3**.

TABLE II
TOP CUSTOMERS BY TOTAL SPENDING

Customer Name	Email	Total Spent ($)
John Doe	john@gmail.com	10130.22
Don Romer	don@gmail.com	1120.00
David Morrison	morrison@gmail.com	567.80
Kevin Ryan	kevin@gmail.com	481.76
William Hopkins	william@gmail.com	9.85

John Doe stands out as the top customer, spending over **$10,000**, which is significantly higher than other customers. This suggests that personalized marketing efforts targeting high-spending individuals could yield further revenue growth.

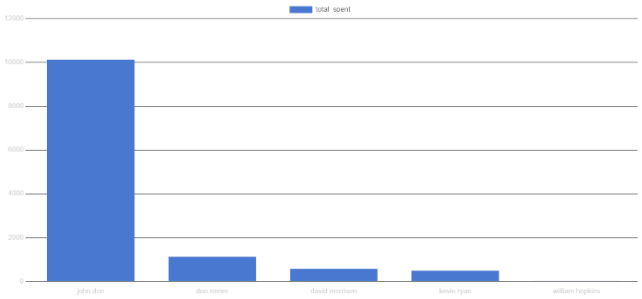

Fig. 3. Top Customers by Total Spending.

C. Total Sales Revenue by Product Category

To understand which product categories contributed the most to overall revenue, total sales were aggregated by category. This was achieved by joining the **categories** and **products** dimension tables with the **sales** fact table. The SQL query used for this analysis can be found in *Appendix A, Query 3*. The revenue by category is summarized in **Table III** and visualized in **Fig. 4**.

TABLE III
TOTAL SALES REVENUE BY PRODUCT CATEGORY

Category Name	Total Revenue ($)
Men's Clothing	2646.44
Jewelery	1410.98
Electronics	624.00
Women's Clothing	9.85

Men's clothing generated the highest revenue among all categories, followed by **jewelery** and **electronics**. Surprisingly, **women's clothing** accounted for the least revenue, indicating a potential area for business growth through targeted marketing or expanding the product line.

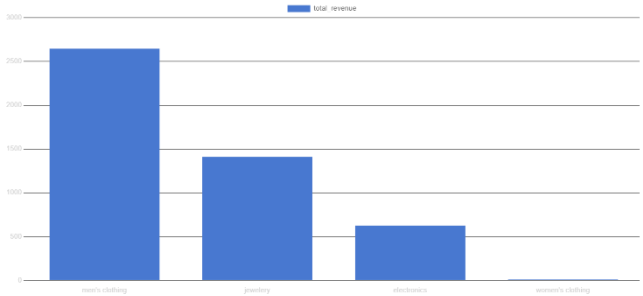

Fig. 4. Total Sales Revenue by Product Category.

V. Future Work and Improvements

While this project successfully designed and implemented an ETL pipeline and data warehouse for e-commerce analytics, several opportunities for future enhancements can further improve its scalability, performance, and analytical capabilities.

1. **Expand the Data Warehouse Schema**
   - **Incorporate Additional Fact and Dimension Tables:** Future iterations can extend the current star schema by adding more fact and dimension tables to capture a wider range of business metrics:
     - **Fact Tables**: Introduce tables to track user engagement metrics, returns, or refund histories to analyze customer satisfaction and product performance.
     - **Dimension Tables**: Add dimensions for suppliers, shipping details, or payment methods to enhance the granularity of insights.
   - **Enhanced Relationships**: Refining relationships between these additional tables would support more complex business queries, enabling deeper insights into customer behavior, supply chain efficiency, and sales performance.
2. **Automate the ETL Pipeline**
   - **Scheduling and Automation Tools**: Implement tools like **Apache Airflow** or **Cron Jobs** to automate data ingestion, transformation, and loading processes. This ensures **real-time** or **scheduled updates** without manual intervention.
   - **Error Handling and Logging**: Introduce robust **logging mechanisms** and **error-handling** to monitor the ETL process and quickly address failures.
3. **Integrate More Data Sources**
   - **APIs and External Datasets**: To gain a more comprehensive view of the business, integrate with **additional APIs** such as **Google Analytics** for website data or **payment gateway APIs** for transaction details.
   - **Real-Time Data Streams**: Incorporate **real-time data streams** using tools like **Apache Kafka** to enable **live dashboards** and **real-time analytics**.
4. **Advanced Analytics and Machine Learning Integration**
   - **Predictive Analytics**: Apply **machine learning algorithms** for **sales forecasting**, **customer segmentation**, and **recommendation systems**.
   - **Data Fusion Techniques**: Explore **data fusion** methods to combine e-commerce data with **social media insights** or **market trends** for a broader business perspective.
5. **Improve Data Quality and Validation**
   - **Data Cleansing Pipelines**: Implement advanced data validation and cleansing routines to ensure **data consistency** and **accuracy**.
   - **Schema Evolution Handling**: Design mechanisms to handle **schema changes** in source APIs without disrupting the ETL process.
6. **Scalability and Performance Optimization**
   - **Database Optimization**: Optimize **PostgreSQL performance** through indexing, partitioning, and query optimization.
   - **Cloud Migration**: Consider migrating the data warehouse to a **cloud-based solution** (e.g., **Amazon Redshift**, **Google BigQuery**) to improve **scalability** and **cost-efficiency**.
7. **Enhanced Reporting and Visualization**
   - **Interactive Dashboards**: Integrate with **Business Intelligent (BI)** tools like **Tableau** or **Power BI** for interactive, user-friendly dashboards.
   - **Custom Reports**: Allow users to generate **customizable reports** based on their specific needs.

VI. Conclusion

This project successfully demonstrated the design and implementation of an ETL pipeline and a data warehouse tailored for e-commerce analytics. Utilizing data from the **Fake Store API**, we executed a comprehensive workflow that included data ingestion, transformation, and the creation of a **star schema** within a **PostgreSQL** data warehouse. This schema was specifically designed to optimize querying efficiency and support the generation of actionable business insights.

Throughout the ETL process, we addressed several challenges, including:

- **Data Type Inconsistencies:** Managing conversions of string-based numerical data to appropriate types for accurate analysis.
- **Nested JSON Transformations:** Flattening complex data structures, such as product ratings, for seamless integration into relational tables.
- **Schema Alignment:** Ensuring that the transformed data aligned with the star schema design to support efficient analytical queries.

These challenges underscored the complexities inherent in real-world data engineering projects and provided valuable learning opportunities in data modeling, schema design, and ETL automation.

The resulting data warehouse facilitated the extraction of meaningful insights, such as:

- Identifying top-performing products based on sales revenue.
- Highlighting high-spending customers to inform targeted marketing strategies.
- Analyzing category-wise revenue distribution to uncover potential growth opportunities.

These insights underscore the importance of a well-structured data warehouse in driving data-informed business decisions.

Looking ahead, there are numerous avenues for future enhancements:

- **ETL Automation:** Implementing tools like Apache Airflow to automate data workflows.
- **Integration of Additional Data Sources:** Incorporating APIs for payment gateways, web analytics, and real-time data streams to enrich the dataset.
- **Advanced Analytics Techniques:** Applying predictive modeling and machine learning to uncover deeper insights and forecast future trends.

By expanding the data warehouse schema and incorporating real-time analytics capabilities, the project can evolve to address increasingly complex business questions, supporting more nuanced and strategic decision-making.

In summary, this project not only met its objectives but also established a scalable, robust framework that can be extended to more complex and larger-scale e-commerce environments. It provides a solid foundation for future work in the field of data engineering and analytics.

Appendix

Appendix A

SQL Queries for Insights

1. Top 5 Products by Total Sales Revenue:

```
SELECT
    title AS product_name,
    description,
    SUM(price * quantity) AS
total_revenue
FROM
    products
JOIN
    carts ON products.ProductsId =
carts.productId
GROUP BY
    product_name, description
ORDER BY
    total_revenue DESC
LIMIT 5;
```

2. Top 5 Customers by Total Spending:

```
SELECT
    CONCAT(firstname, ' ', lastname)
AS customer_name,
    email,
    SUM(price * quantity) AS
total_spent
FROM
    users
JOIN
    carts ON users.UsersId =
carts.userId
JOIN
    products ON carts.productId =
products.ProductsId
GROUP BY
    customer_name, email
ORDER BY
    total_spent DESC
LIMIT 5;
```

3. Total Sales Revenue by Product Category:

```
SELECT
    categories AS category_name,
    SUM(price * quantity) AS
total_revenue
FROM
    categories
JOIN
    products ON categories.CategoryId
= products.CategoryId
JOIN
    carts ON products.ProductsId =
carts.productId
GROUP BY
    category_name
ORDER BY
    total_revenue DESC;
```

Appendix B

ETL Code Snippets

1. Data Ingestion

```
import requests
import json
import os

# Create directories for raw data
```

```
os.makedirs("data/raw",
exist_ok=True)

# Function to fetch data from API and
save as JSON
def fetch_and_save(endpoint,
filename):
    url =
f"https://fakestoreapi.com/{endpoint}
"
    response = requests.get(url)

    if response.status_code == 200:
        with
open(f"data/raw/{filename}.json",
"w") as file:

json.dump(response.json(), file,
indent=4)
        print(f"Data saved to
data/raw/{filename}.json")
    else:
        print(f"Failed to fetch data
from {url}")

# Fetching data
fetch_and_save("products",
"products")
fetch_and_save("users", "users")
```

2. Data Transformation

```
import pandas as pd

# Processing nested JSON for users
def process_user_dimension(users):
    user_dimension =
pd.DataFrame(users)
    user_dimension["first_name"] =
user_dimension["name"].apply(lambda
x: x["firstname"])
    user_dimension["last_name"] =
user_dimension["name"].apply(lambda
x: x["lastname"])
    user_dimension["city"] =
user_dimension["address"].apply(lambd
a x: x["city"])
    return user_dimension[["id",
"first_name", "last_name", "city"]]

# Fact table creation with calculated
columns
def process_sales_fact_table(carts,
products):
    fact_table = []
    for cart in carts:
        for item in cart["products"]:
            product_price =
products[item["productId"] -
1]["price"]
            total_price =
product_price * item["quantity"]
            fact_table.append({
                "cart_id":
cart["id"],
                "product_id":
item["productId"],
                "quantity":
item["quantity"],
                "total_price":
total_price
            })
    return pd.DataFrame(fact_table)
```

Appendix C

Data Warehouse Schema

```
-- Product Dimension
CREATE TABLE product_dimension (
    product_id SERIAL PRIMARY KEY,
    product_name VARCHAR(255),
    price DECIMAL(10, 2)
);

-- User Dimension
CREATE TABLE user_dimension (
    user_id SERIAL PRIMARY KEY,
    email VARCHAR(255),
    first_name VARCHAR(50),
    city VARCHAR(100)
);

-- Sales Fact Table
CREATE TABLE sales_fact_table (
    sales_id SERIAL PRIMARY KEY,
    product_id INT,
    user_id INT,
    quantity_purchased INT,
    total_price DECIMAL(10, 2)
);
```

For the complete source code, refer to the project repository: ECommerce Sales Data Warehouse.

Acknowledgements

The authors would like to thank Eng. Basem Khalaf for their invaluable guidance and feedback throughout the course of this project. We also extend our appreciation to the developers of the Fake Store API for providing the dataset used in this study. Additionally, we acknowledge the collaborative efforts of our team members, whose contributions were critical to the successful design and implementation of the ETL pipeline and data warehouse.

References

[1] R. Kimball and M. Ross, *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*, 3rd ed. Wiley, 2013.

[2] W. Inmon, *Building the Data Warehouse*, 4th ed. Wiley, 2005.

[3] PostgreSQL Global Development Group, "PostgreSQL Documentation," [Online]. Available: https://www.postgresql.org/docs/. [Accessed: Feb. 5, 2025].

[4] “Fake Store API Documentation," [Online]. Available: https://fakestoreapi.com/docs. [Accessed: Feb. 5, 2025].

[5] P. Dubois, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*, 2nd ed. O'Reilly Media, 2017.

[6] M. Stonebraker, "The Case for Shared Nothing," *IEEE Database Engineering Bulletin*, vol. 9, no. 1, pp. 4-9, 1986.

Walid K. W. Alsafadi was born in Ras Al Khaimah, UAE. He is currently pursuing a Bachelor of Science in Data Science and Artificial Intelligence at the University College of Applied Science, Gaza, Palestine. His major field of study focuses on data science and artificial intelligence, with expertise in machine learning, deep learning, and natural language processing.

Walid has previously studied at the American University of Ras Al Khaimah, UAE, where he completed two semesters before transferring to UCAS. His academic achievements include multiple scholarships and recognition on the Dean’s List for academic excellence. He has worked on diverse projects, including sales forecasting, sentiment analysis, and diabetes prediction, utilizing machine learning techniques and frameworks.

Mr. Alsafadi is proficient in multiple programming languages, including Python and SQL, and is a native Arabic speaker with full professional proficiency in English.

Ameer T. F. Alzerei was born in Deir Al-Balah, Gaza, graduated from high school scientific route with average 99.1%, enrolled in Data Science and Artificial Intelligence undergraduate program at UCAS, Gaza. Ameer obtained some professional certificates, with self-learning. Had an internship in Switzerland as an intern student in the first cohort through a program called Bridges to Growth.

Ameer has worked on various projects, including buildwize expert system, employees attrition analysis, and grasp app CV tools. He is proficient in multiple programming languages, including Python, SQL, TypeScript, and C++, and is a native Arabic speaker with full professional proficient in English.

Hamza M. H. Obaid was born in Gaza in 2003. He graduated from the scientific branch of high school in 2021. He joined the University College of Applied Sciences, specializing in data science and artificial intelligence, in the same year. He is currently a third-year student.

Hazem A. A. Muanes was born in Deir Al-Balah, Gaza, and enrolled in Data Science and Artificial Intelligence program at UCAS, Gaza. He has worked on several projects, including UFO sightings prediction and IMDB movies prediction. Hazem is proficient in multiple programming languages, including Python and SQL. And is a native Arabic speaker with limited work proficiency.