**E-Commerce & Warehouse Intelligence Platform (EWIP)**

Hernandez, Jan Cedric M.

University of the Philippines Diliman

Department of Industrial Engineering and Operations Research

IE 272: Data Engineering

December 2024

**Table of Contents**

[List of Figures 4](#_Toc185946377)

[List of Tables 5](#_Toc185946378)

[I. Introduction / Background 6](#_Toc185946379)

[II. Business Case 6](#_Toc185946380)

[III. Business Questions 7](#_Toc185946381)

[IV. Project Timeline 8](#_Toc185946382)

[V. Proposed System Architecture 9](#_Toc185946383)

[a. High Level System Architecture 9](#_Toc185946384)

[b. Data Marts 9](#_Toc185946385)

[c. Data Layers 10](#_Toc185946386)

[e. Data Model 16](#_Toc185946387)

[f. ETL Process 19](#_Toc185946388)

[g. Metadata Documentation 20](#_Toc185946389)

[VI. Physical Design of Your Data Engineering Solution 22](#_Toc185946390)

[a. Architectural Overview and Implementation Details 22](#_Toc185946391)

[b. SQL Scripts for Implementations 23](#_Toc185946392)

[*i.* *Importing libraries* 23](#_Toc185946393)

[*ii.* *Extracting from Source Table* 23](#_Toc185946394)

[*iii.* *Compare New and Changed Data with the Master Table* 24](#_Toc185946395)

[*iv.* *Insert into Extract Tables* 24](#_Toc185946396)

[*v.* *Clean X Table and Insert Into Error Table* 25](#_Toc185946397)

[*vi.* *Process Clean Data and Insert Into C Table* 26](#_Toc185946398)

[*vii.* *Update Master Table (M Table)* 26](#_Toc185946399)

[*viii.* *Initiate Transform Processes* 28](#_Toc185946400)

[*ix.* *Select Data from T Table and Insert to I and U Table* 29](#_Toc185946401)

[*x.* *Insert I Table into D Table* 30](#_Toc185946402)

[*xi.* *Insert U Table Data (Type 2) into D Table* 30](#_Toc185946403)

[*xii.* *Update Indicators to Current in D Table* 31](#_Toc185946404)

[c. Proposed Data Warehouse Design 32](#_Toc185946405)

[VII. Reports Generation 35](#_Toc185946406)

[VIII. Project Management 42](#_Toc185946407)

[IX. Conclusion 42](#_Toc185946408)

[X. References 42](#_Toc185946409)

[XI. Appendices 43](#_Toc185946410)

[a. ERD for LPMS 43](#_Toc185946411)

[b. ERD for UMS 43](#_Toc185946412)

[c. ERD for OMS 44](#_Toc185946413)

[d. ERD for WMS 45](#_Toc185946414)

[e. Relational Model for LPMS 46](#_Toc185946415)

[f. Relational Model for UMS 47](#_Toc185946416)

[g. Relational Model for OMS 48](#_Toc185946417)

[h. Relational Model for WMS 49](#_Toc185946418)

# **List of Figures**

Figure 1. E-Commerce Market Volume Outlook and Forecast……………………………………………..…7

Figure 2. System Architecture Design for the EWIP Project……………………………………………….…..9

Figure 3. Sample Data Model for Each Type of Fact Table………………………………………………….…16

Figure 4. Data Models for Data Warehouse Tables Also Included in Marketplace Data Mart……...17

Figure 5. Data Models for Data Warehouse Tables Also Included in Warehouse Data Mart…….....18

Figure 6. ETL Process for Dim Tables………………………………………………………………………………...19

Figure 7. ETL Process for Fact Tables (Detail, Summary, Snapshot)………………………………….…….20

Figure 8. Fact Tables and Their Attributes Under Marketplace Data Mart……………………………..….33

Figure 9. Dim Tables and Their Attributes Under Marketplace Data Mart…………………………….…..34

Figure 10. Fact Tables and Their Attributes Under Warehouse Data Mart…………………………….….35

Figure 11. Dim Tables and Their Attributes Under Warehouse Data Mart……………………………..…35

Figure 12. Sample Power BI Dashboard for Platform Performance Reporting……………………….…39

# **List of Tables**

Table 1. High-Level View of the Project’s Major Phases and Key Activities………………………….…….8

Table 2. Proposed Table Designs for the Data Warehouse…………………………………………………….10

Table 3. High-Level Source-to-Target (S2T) Map for the Proposed Data Warehouse Design…….….11

Table 4. Sample Detailed Source-to-Target (S2T) Map for the 3 Data Warehouse Table…………..…13

Table 5. Sample Documentation for the fact\_platform\_peformance\_summary table……………....21

Table 6. Proposed architecture and implementation of the EWIP system……………………………….22

Table 7. Complete List of Tables in the Proposed Data Warehouse and Data Marts……………….…32

Table 8. Gantt Chart for the EWIP Implementation……………………………………………………………..40

# **Introduction / Background**

In the past, the emergence of e-commerce giants like Alibaba and JD.com can be attributed to the 2003 SARS outbreak while the significant increase in shareholder value of American Express and Starbucks during the 2008−2009 global financial crisis is related to their strategy shifting towards digital operating model (Candelon et al., 2020). During the COVID-19 pandemic, there was another surge in the e-commerce industry. This caused travel restrictions and health concerns, prompting a significant shift towards online shopping and digital transactions. E-commerce is still expected to flourish in the market with the growing accessibility to the internet and mobile phones together with several digitalization initiatives reaching rural communities (Fabri & Valverde Márquez, 2020, Dragomirov, 2020).

E-commerce encompasses business transactions done electronically between or among organizations and between organizations and individuals either locally or globally. In the context of the Philippines, Shopee and Lazada are the most locally known e-commerce companies, along with other competitors such as TikTok Shop, Carousell, and E-bay. These companies provide a platform, usually through mobile apps and websites, where several consumers can buy various products from different sellers in the digital space. This approach also enables digital marketplaces to operate at any time of the day as the systems are designed to work without human support (Fabri & Valverde Márquez, 2020).

In a small to medium-scale company, operations can be initiated with four key information systems. First, the Order Management System (OMS) records all order transactions and relevant data about items, sellers, buyers, vouchers, and fulfillment. Second, the Listing & Promotion Management System (LPMS) captures details about shop listings and promotional strategies, including items and vouchers related to orders. Third, since every order transaction involves both buyers and sellers, a User Management System (UMS) is essential for storing user-related data. Finally, any e-commerce operation involving warehousing activities requires a Warehouse Management System (WMS) to manage inbound and outbound orders, staff attendance, and inventory operations.

This paper discusses the construction of a functional data warehouse to integrate data from these four information systems (IS). By applying dimensional modeling to the relational models of these systems, the data warehouse enables efficient data processing and storage.

# **Business Case**

Based on a market volume forecast by Statista (2024), the e-commerce market has shown an increasing trend over the past three years and is projected to grow threefold by 2030. This growth is expected to result in a significant surge in the number of transactions for e-commerce businesses, leading to an exponential increase in data volume, variety, and complexity. If the current databases used by e-commerce businesses are not designed to handle this scale, the data systems may struggle with querying historical data, resulting in inefficiencies.

A graph of different colored bars

Description automatically generated

**Figure 1.** *E-Commerce Market Volume Outlook and Forecast (Statista, 2024)*

Without a data warehouse, data users must query various normalized tables containing raw data from information systems and manually perform logic calculations for different metrics. This approach leads to slow query performance, higher resource utilization, and an increased risk of metric logic misalignment across functional teams. Given the high volume, variety, and complexity of data in e-commerce operations, it is essential for businesses to have a functional and efficient data warehouse. A data warehouse provides an integrated data processing system, standardizes metric definitions through single source-of-truth tables, and enables efficient and accurate data reporting to support data-driven decision-making and enhance operational efficiency.

# **Business Questions**

The data warehouse primarily aimed to output various dimensions and fact tables for streamlined data reporting. The data warehousing solution aimed to answer the following questions:

1. What are the monthly historical platform performance and month-on-month (MoM) growth in terms of average daily order (ADO), average daily gross merchandise value (ADGMV), average active buyers, average active shops, and average order-to-delivery (OTD) time?
2. What are the top item categories for monthly ADO and ADGMV? Which item categories have the highest month-on-month ADO and ADGMV growth?
3. What are the top-performing shop categories in terms of monthly ADO and ADGMV? What shop categories contribute to more than 10% of the platform ADGMV?
4. What is the monthly historical performance and MoM growth of the overall warehouse and each warehouse in terms of ADO, average daily item (ADI) count, productivity rate, and idle rate?
5. Who are the warehouse staff who have last 30-day productivity less than 90% of the average rate of the top 10 staff?

# **Project Timeline**

The project implementation is divided into 6 phases targeted to be finished after 10 weeks as outlined in Table 1. The first phase is the requirement analysis phase to check stakeholder identification and business requirements before completing the final project charter. This is targeted to be performed in 2 weeks. Next would be the data modeling phase to design and revise the proposed data models with stakeholder alignment for 1 week. Third, the ETL implementation phase will be completed in 2 weeks with corresponding testing and validation for another 2 weeks. Lastly, around 2 weeks is allotted for the user acceptance test phase where the users will be requested to provide feedback that will need any revision before the final phase of deployment and cascade for the turnover and closing of the project for the project’s final week.

**Table 1.** *High-Level View of the Project’s Major Phases and Key Activities*

|  |  |  |
| --- | --- | --- |
| **Week** | **Phase** | **Tasks** |
| 1 | Requirement Analysis | Identify key stakeholders  Gathering stakeholder and business requirements  Source system analysis (architecture, data, data type)  Completing project charter  Initial stakeholder alignment |
| 3 | Data Modeling | Designing dim tables  Designing fact tables  Designing summary and snapshot fact tables  Stakeholder alignment |
| 4-5 | ETL Implementation | ETL for the dim tables  ETL for the fact tables  ETL for the summary fact tables  Stakeholder alignment on logic |
| 6-7 | Testing and Validation | Initial testing (low-volume data)  Possible debugging  Volume testing (high-volume)  Possible debugging and optimization |
| 8-9 | UAT and Revisions | Production testing for selected users  Feedback alignment  Actions for feedback |
| 10 | Deployment and Cascade | Process and technical documentation  DW Cascade to all users  Closing the project |

A detailed Gantt chart is also presented in the project management portion of the paper.

# **Proposed System Architecture**

## **High Level System Architecture**

A diagram of a data flow

Description automatically generated

**Figure 2.** *System Architecture Design for the EWIP Project*

Figure 2 shows the proposed system architecture showing the flow of data from the source systems up to the output. Source data will be fetched from Postgres databases and flat file ingestion. The ETL process will be performed via dbt tasks orchestrated by Airflow before storing in the Google Cloud-based data warehouse. The two (2) data marts will be populated from the data warehouse by selecting only the relevant data per business domain. The data can be consumed by Power BI for visualization, Google BigQuery for ad hoc queries, and GSheet for tabular reporting. Further details are provided in the next sections.

## **Data Marts**

In the proposed system, there will be 2 data marts that serve as specialized subsets of the data warehouse for each business domain. The first one is the marketplace data mart (MDM) which contains almost all data that can be browsed in the e-commerce application. It includes data regarding orders, users, items, parcels, vouchers, and fulfillment. The other is the warehouse data mart (WDM) which involves data in the warehouse fulfillment operations such as data related to purchase orders, outbound orders, staff performance, warehouse tasks, and storage.

Data marts are part of the architecture due to their several advantages. First, it allows - intuitive distinction of group of tables to allow self-service analytics and ease of use. Since each domain has its own set of metrics and reporting needs, the division of data per domain offers clarity and relevance for the stakeholders. It also contributes to query performance optimization since the data relevant to the business domain are the only ones included which reduces the amount of data to be processed and speeds up queries. Third, it also helps in the data governance side to allow easier table-level or mart-level access control. Lastly, it supports scalability as new data marts can easily be built for other specific business domains or any business operation expansion.

## **Data Layers**

In the proposed design of the data warehouse and data marts, there are 4 types of tables according to the stakeholder’s needs: dim, detail fact, summary fact, and snapshot fact tables as described in Table 2.

**Table 2.** *Proposed Table Designs for the Data Warehouse*

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Type** | **Frequency** | **Data Mart** | **Sample Tables** |
| Dim Tables | 20 | MDM | *buyer, seller, item, date, time, location, shipping* |
| WDM | *warehouse, staff, supplier* |
| Detail Fact Tables | 8 | MDM | *order, order item, parcel* |
| WDM | *outbound task, outbound order* |
| Summary Fact Tables | 5 | MDM | *performance of platform, seller, and SKU* |
| WDM | *staff productivity, warehouse performance* |
| Snapshot Fact Tables | 3 | MDM | *historical parcel status* |
| WDM | *staff attendance, storage* |

In the data warehouse, there are several dim tables per entity such as order, item, warehouse, time, date, buyer, and seller. For the fact tables, there are 3 types of tables generated. First are detailed fact tables which contain information on a transactional level. Available granularities include levels of order, parcel, order item, outbound task, outbound order, inbound task, and inbound order stock-keeping unit(SKU). Based on the data of the detail fact tables, summary fact tables were made readily for management-level report data to have standardized logic calculation and single sources of truth for the business’ key metrics (SSOT). These tables contain the historical performance of the platform, sellers, items, warehouse, and staff. Lastly, there are also snapshot fact tables. These tables are designed for historical tracking including historical parcel status, staff attendance, and storage history.

* 1. **Source Systems**

For the case study, there are four information systems (IS) catering to the whole operation of the e-commerce platform. First, there is the order management system (OMS) which contains order transaction and order-related data such as order ID, order items, and fulfillment status. Second, there is a user management system (UMS) that stores all user-related data whether they act as buyer, seller, or admin. This also includes the preferred pickup locations for sellers, and payment options, and delivery details for the buyers which are all configured in the e-commerce application. There is also a listings & promotion management system (LPMS) where sellers register their listings and promotional vouchers. Lastly, data will also be sourced from the warehouse management system (WMS) for all warehouse data including inbound, outbound, and fulfillment operation data. For this case study, the current OLTP databases of the business are hosted in the PostgreSQL Cloud. In addition, some data will also be sourced from flat-file ingestion for warehouse-specific data necessary for profiling the dim\_warehouse table.

The entity relationship diagram and relational model designs of the 4 databases can be browsed in the appendix. These were transformed into dimensional models wherein the high-level source-to-target (S2T) mapping can be browsed in Table 3. For the current design, most data warehouse tables get data from a single table from the OLTP database while others get data from 2-5 source tables. The data warehouse the dim\_location table gets from 5 source tables to serve as a complete mapping of all location-related data from all available data sources. Sample detailed S2T maps for fact\_order\_detail, dim\_warehouse, and fact\_platform\_peformance\_summary were also provided in Table 4.

**Table 3.** *High-Level Source-to-Target (S2T) Map for the Proposed Data Warehouse Design*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Row** | **System Source** | **Source Table** | **Target Data Mart** | **Target Table** |
| 1 | OMS | order | MDM | dim\_order |
| 2 | OMS | order | MDM | dim\_order\_parcel |
| 3 | OMS | fulfillment | MDM | dim\_parcel\_status |
| 4 | WMS | outbound\_order | MDM | dim\_parcel\_status |
| 5 |  |  | MDM | dim\_time |
| 6 |  |  | MDM | dim\_date |
| 7 | OMS | courier | MDM | dim\_shipping |
| 8 | OMS | fulfillment | MDM | dim\_operator |
| 9 | OMS | courier | MDM | dim\_operator |
| 10 | OMS | fulfillment | MDM | dim\_location |
| 11 | UMS | delivery\_detail | MDM | dim\_location |
| 12 | UMS | pickup\_location | MDM | dim\_location |
| 13 | UMS | user | MDM | dim\_location |
| 14 | Flat File Ingestion | wh\_information\_ingestion | MDM | dim\_location |
| 15 | UMS | payment\_option | MDM | dim\_payment |
| 16 | LPMS | item | MDM | dim\_item |
| 17 | LPMS | listing | MDM | dim\_item |
| 18 | UMS | seller | MDM | dim\_item |
| 19 | UMS | seller | MDM | dim\_seller |
| 20 | UMS | user | MDM | dim\_seller |
| 21 | UMS | user | MDM | dim\_buyer |
| 22 | LPMS | voucher | MDM | dim\_voucher |
| 23 | LPMS | voucher\_redemption | MDM | dim\_voucher\_mix |
| 24 | LPMS | voucher | MDM | dim\_voucher\_mix |
| 25 | OMS | order | MDM | fact\_order\_detail |
| 26 | OMS | item | MDM | fact\_order\_detail |
| 27 | WMS | outbound\_order | MDM | fact\_order\_detail |
| 28 | LPMS | voucher\_redemption | MDM | fact\_order\_detail |
| 29 | OMS | order | MDM | fact\_parcel\_detail |
| 30 | OMS | item | MDM | fact\_parcel\_detail |
| 31 | WMS | outbound\_order | MDM | fact\_parcel\_detail |
| 32 | OMS | fulfillment | MDM | fact\_parcel\_detail |
| 33 | OMS | order | MDM | fact\_order\_item\_detail |
| 34 | OMS | item | MDM | fact\_order\_item\_detail |
| 35 | WMS | outbound\_task | MDM | fact\_order\_item\_detail |
| 36 | OMS | fulfillment | MDM | fact\_parcel\_status\_snapshot |
| 37 | MDM | fact\_order\_detail | MDM | fact\_platform\_performance\_summary |
| 38 | MDM | fact\_order\_item\_detail | MDM | fact\_sku\_performance\_summary |
| 39 | MDM | fact\_order\_detail | MDM | fact\_seller\_performance\_summary |
| 40 | Flat File Ingestion | wh\_information\_ingestion | WDM | dim\_warehouse |
| 41 | WMS | outbound\_task | WDM | dim\_task |
| 42 | WMS | inbound\_task | WDM | dim\_task |
| 43 | WMS | staff | WDM | dim\_staff |
| 44 | WMS | supplier | WDM | dim\_supplier |
| 45 | WMS | storage | WDM | dim\_storage |
| 46 | WMS | inbound\_order | WDM | dim\_purchase\_order\_asn |
| 47 | WMS | purchase\_order | WDM | dim\_purchase\_order\_asn |
| 48 | WMS | outbound\_task | WDM | fact\_outbound\_task\_detail |
| 49 | WMS | outbound\_order | WDM | fact\_outbound\_sku\_detail |
| 50 | WMS | outbound\_task | WDM | fact\_outbound\_sku\_detail |
| 51 | WMS | outbound\_order | WDM | fact\_outbound\_order\_detail |
| 52 | WMS | inbound\_task | WDM | fact\_inbound\_task\_detail |
| 53 | WMS | inbound\_order | WDM | fact\_inbound\_sku\_detail |
| 54 | WMS | purchase\_order | WDM | fact\_inbound\_sku\_detail |
| 55 | WMS | storage | WDM | fact\_storage\_inventory\_snapshot |
| 56 | WMS | attendance | WDM | fact\_attendance\_snapshot |
| 57 | WDM | fact\_attendance\_snapshot | WDM | fact\_staff\_prod\_summary |
| 58 | WDM | outbound\_task | WDM | fact\_staff\_prod\_summary |
| 59 | WDM | inbound\_task | WDM | fact\_staff\_prod\_summary |
| 60 | WDM | outbound\_order | WDM | fact\_warehouse\_summary |
| 61 | WDM | outbound\_task | WDM | fact\_warehouse\_summary |
| 62 | WDM | fact\_staff\_prod\_summary | WDM | fact\_warehouse\_summary |

*Note: OMS – order management system, WMS – warehouse management system, UMS – user management system, LPMS – listings and promotions management system, MDM – marketplace data mart, WDM – warehouse data mar*

**Table 4.** *Sample Detailed Source-to-Target (S2T) Map for the 3 Data Warehouse Table*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TARGET** | | | | | | **SOURCE** | | | | | | | **HISTORY** | | | | |
| **Warehouse Table** | **Attribute Name** | **Definition** | **Sample Values** | **Target Data Type** | **Target Length** | **Nullable** | **Source System** | **Source File/Table** | **Source Field/Column** | **Source Data Type** | **Source Length** | **Transformation Rule** | **Analytical or Detail** | **Run Frequency** | **Insert Into/Overwrite** | **Change Frequency** | **History Strategy Type** |
| fact\_order\_detail | order\_key | Surrogate key to identify the order and also be used as a foreign key to identify order details | 1,2,3 | bigint | 19 | N | OMS, MDM | oms.order, mdm.dim\_order | order\_id, order\_key | bigint, bigint | 19, 19 | order\_key from oms.order join on mdm.dim\_order using (order\_id) | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | buyer\_key | Foreign key linking to the dim\_buyer table to identify order's buyer | 1,2,3 | int | 10 | N | OMS, MDM | oms.order, mdm.dim\_buyer | buyer\_id, buyer\_key | bigint, int | 19, 10 | buyer\_key from oms.order join on mdm.dim\_buyer using (buyer\_id) | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | seller\_key | Foreign key linking to the dim\_seller table to identify the order's seller | 1,2,3 | int | 10 | N | OMS, MDM | oms.order, mdm.dim\_seller | seller\_id, seller\_key | bigint, int | 19, 10 | seller\_key from oms.order join on mdm.dim\_seller using (seller\_id) | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | voucher\_mix\_key | Foreign key linking to the dim\_voucher\_mix table to identify any voucher mix used in the order | 1,2,3 | bigint | 19 | Y | OMS,  LPMS, MDM | oms.order,  lpms.voucher\_redemption,  mdm.dim\_voucher\_mix | order\_id,  voucher\_id,  voucher\_mix\_key | bigint, bigint, bigint | 19, 19, 19 | voucher\_mix\_key from oms.order  join on lpms.voucher\_redemption using (order\_id)  join on mdm.dim\_voucher\_ms using (voucher\_id) | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | voucher\_total\_value | Total value of all used vouchers for the order | 1.0000, 100.0000, 100000.0000 | decimal | (10, 4) | Y | OMS, LPMS | oms.order,  lpms.voucher\_redemption | order\_id,  redeemed\_amount | bigint, decimal | 19, (10, 4) | sum(redeemed\_amount) from oms.order  join on lpms.voucher\_redemption using (order\_id) | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | payment\_option\_key | Foreign key linking to the dim\_payment\_option tableto identify payment method | 1,2,3 | bigint | 19 | N | OMS, MDM | oms.order, mdm.dim\_seller | payment\_option\_id,  payment\_option\_key | bigint, bigint | 19, 19 | payment\_option\_key from oms.order join on mdm.dim\_payment using (payment\_option\_id) | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | shipping\_fee | Amount of total shipping fee cost paid by buyer | 1.0000, 100.0000, 100000.0000 | decimal | (10, 4) | N | OMS | oms.order | shipping\_fee | decimal | (10, 4) | order from oms.order | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | gmv | Gross merchandise value which is the total value of the goods | 1.0000, 100.0000, 100000.0000 | decimal | (10, 4) | N | OMS | oms.order | gmv | decimal | (10, 4) | gmv from oms.order | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | order\_create\_date\_key | Foreign key to the dim\_date table to identify order's create date | 1,2,3 | bigint | 19 | N | OMS, MDM | oms.order, mdm.dim\_date | order\_create\_time, date\_key | timestamp, bigint | \_, 19 | date\_key from oms.order a join on mdm.dim\_date b on date(a.order\_create\_time) = b.date | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | order\_create\_time\_key | Foreign key to the dim\_time table to identify order's create timestamp | 1,2,3 | bigint | 19 | N | OMS, MDM | oms.order, mdm.dim\_time | order\_create\_time, time\_key | timestamp, bigint | \_, 19 | time\_key from oms.order a join on mdm.dim\_time b on a.order\_create\_time = b.timestamp | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | unique\_sku\_count | Number of distinct SKUs in the order | 1,2,3 | smallint | 5 | N | OMS | oms.item | sku\_id | varchar | 20 | count(distint sku\_id) from oms.item group by order\_id | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | total\_sky\_qty | Total quantity of items in the order | 1,2,3 | smallint | 5 | N | OMS | oms.item | qty | smallint | 5 | sum(qty) from oms.item group by order\_id | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | wh\_key | Foreign key linking to the dim\_warehouse table to identify the assigned warehouse, if any | 1,2,3 | smallint | 3 | Y | OMS,  WMS,  WDM | oms.order, wms.outbound\_order,  wdm.dim\_warehouse | wh\_id, wh\_key | smallint, smallint | 3, 3 | wh\_key from oms.order  left join wms.oubound\_order using (order\_id)  left join wdm.dim\_warehouse using (wh\_id) | Detail | Daily | Insert Into | N/A | N/A |
| fact\_order\_detail | parcel\_count | Number of parcels for the order | 1,2,3 | smallint | 3 | N | OMS | oms.order | parcel\_id | bigint | 19 | count(parcel\_id) from oms.order | Detail | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | date\_key | Foreign key to the dim\_date table, representing the relevant date for time-series metrics | 1,2,3 | bigint | 19 | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | date\_key | bigint | 19 | distinct date\_key\_ from mdm.fact\_order\_detail | Detail | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l1d\_ado | Average daily orders for the last 1 day | 1000000.1234, 10000000.1234, 10000000.1234 | decimal | (12, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | order\_key, date\_key | bigint, bigint | 19, 19 | count(order\_key)/1.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date = current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l7d\_ado | Average daily orders for the last 7 days | 1000000.1234, 10000000.1234, 10000000.1234 | decimal | (12, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | order\_key, date\_key | bigint, bigint | 19, 19 | count(order\_key)/7.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '7 days' and current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l30d\_ado | Average daily orders for the last 30 days | 1000000.1234, 10000000.1234, 10000000.1234 | decimal | (12, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | order\_key, date\_key | bigint, bigint | 19, 19 | count(order\_key)/30.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '30 days' and current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l1d\_adgmv | Average daily GMV for the last 1 day | 10000000.1234, 100000000.1234, 100000000.1234 | decimal | (15, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | gmv, date\_key | decimal, bigint | (10, 4), 19 | sum(gmv)/1.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date = current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l7d\_adgmv | Average daily GMV for the last 7 days | 10000000.1234, 100000000.1234, 100000000.1234 | decimal | (15, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | gmv, date\_key | decimal, bigint | (10, 4), 19 | sum(gmv)/7.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '7 days' and current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l30d\_adgmv | Average daily GMV for the last 30 days | 10000000.1234, 100000000.1234, 100000000.1234 | decimal | (15, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | gmv, date\_key | decimal, bigint | (10, 4), 19 | sum(gmv)/30.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '30 days' and current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l1d\_avg\_active\_buyers | Average number of active buyers per day in the last 1 day | 1000000.1234, 10000000.1234, 10000000.1234 | decimal | (12, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | buyer\_key, date\_key | int, bigint | 10, 19 | count(distinct buyer\_key)/1.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date = current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l7d\_avg\_active\_buyers | Average number of active buyers per day in the last 7 days | 1000000.1234, 10000000.1234, 10000000.1234 | decimal | (12, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | buyer\_key, date\_key | int, bigint | 10, 19 | count(distinct buyer\_key)/7.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '7 days' and current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l30d\_avg\_active\_buyers | Average number of active buyers per day in the last 30 days | 1000000.1234, 10000000.1234, 10000000.1234 | decimal | (12, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | buyer\_key, date\_key | int, bigint | 10, 19 | count(distinct buyer\_key)/7.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '30 days' and current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l1d\_avg\_active\_shops | Average number of active shops per day in the last 1 day | 1000000.1234, 10000000.1234, 10000000.1234 | decimal | (12, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | seller\_key, date\_key | int, bigint | 10, 19 | count(distinct seller\_key)/1.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date = current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l7d\_avg\_active\_shops | Average number of active shops per day in the last 7 days | 1000000.1234, 10000000.1234, 10000000.1234 | decimal | (12, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | seller\_key, date\_key | int, bigint | 10, 19 | count(distinct seller\_key)/7.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '7 days' and current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l30d\_avg\_active\_shops | Average number of active shops per day in the last 30 days | 1000000.1234, 10000000.1234, 10000000.1234 | decimal | (12, 4) | N | MDM | mdm.fact\_order\_detail, mdm.dim\_date | seller\_key, date\_key | int, bigint | 10, 19 | count(distinct seller\_key)/7.0000 from mdm.fact\_order\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '30 days' and current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l1d\_otd\_time | Average on-time delivery time (in hours) for the last 1 day | 1.0000, 10.1234, 99.9999 | decimal | (6, 4) | N | MDM | mdm.fact\_parcel\_detail, mdm.dim\_date | order\_create\_time\_key, delivered\_time\_key, date\_key | bigint, bigint, bigint | 19, 19, 19 | (delivered\_time\_key - order\_create\_time\_key)/(count(parcel\_id) \* 3600.0000)  from mdm.fact\_parcel\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where a.delivered\_time\_key is not null   and b.date = current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l7d\_otd\_time | Average on-time delivery time (in hours) for the last 7 days | 1.0000, 10.1234, 99.9999 | decimal | (6, 4) | N | MDM | mdm.fact\_parcel\_detail, mdm.dim\_date | order\_create\_time\_key, delivered\_time\_key, date\_key | bigint, bigint, bigint | 19, 19, 19 | (delivered\_time\_key - order\_create\_time\_key)/(count(parcel\_id) \* 3600.0000)  from mdm.fact\_parcel\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where a.delivered\_time\_key is not null   and b.date between current\_date - interval '7 days' and current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| fact\_platform\_peformance\_summary | l30d\_otd\_time | Average on-time delivery time (in hours) for the last 30 days | 1.0000, 10.1234, 99.9999 | decimal | (6, 4) | N | MDM | mdm.fact\_parcel\_detail, mdm.dim\_date | order\_create\_time\_key, delivered\_time\_key, date\_key | bigint, bigint, bigint | 19, 19, 19 | (delivered\_time\_key - order\_create\_time\_key)/(count(parcel\_id) \* 3600.0000)  from mdm.fact\_parcel\_detail a  join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where a.delivered\_time\_key is not null   and b.date between current\_date - interval '30 days' and current\_date - interval '1 day' | Analytical | Daily | Insert Into | N/A | N/A |
| dim\_warehouse | wh\_key | Foreign key linking to the dim\_warehouse table to identify the assigned warehouse, if any | 1,2,3 | serial | 3 | N | N/A | N/A | N/A | N/A | N/A | system generated | N/A | N/A | N/A | N/A | N/A |
| dim\_warehouse | wh\_id | Identification for warehouse (natural key) | 1,2,3 | smallint | 3 | N | WMS | wms.outbound\_order | wh\_id | smallint | 3 | distinct(wh\_id) from wms.outbound\_order | Detail | Monthly | Insert Into | No | N/A |
| dim\_warehouse | wh\_name | Designated name of the warehouse | LUZ01, VIZ02, MIN03 | varchar | 5 | N | Flat File Ingestion | wh\_information\_ingestion | wh\_name | varchar | 5 | wh\_name from wh\_information\_ingestion | Detail | Monthly | Insert Into | No | N/A |
| dim\_warehouse | wh\_region | Warehouse's geographic region address | Luzon, Visayas, Mindanao | varchar | 30 | N | Flat File Ingestion | wh\_information\_ingestion | wh\_region | varchar | 30 | wh\_region from wh\_information\_ingestion | Detail | Monthly | Insert Into | No | N/A |
| dim\_warehouse | wh\_city | Warehouse's city address | Mandaluyong, Bansud, Manila | varchar | 30 | N | Flat File Ingestion | wh\_information\_ingestion | wh\_city | varchar | 30 | wh\_city from wh\_information\_ingestion | Detail | Monthly | Insert Into | No | N/A |
| dim\_warehouse | wh\_brgy | Warehouse's barangay address | Wack-Wack, Alcadesma, Poblacion | varchar | 30 | N | Flat File Ingestion | wh\_information\_ingestion | wh\_brgy | varchar | 30 | wh\_brgy from wh\_information\_ingestion | Detail | Monthly | Insert Into | No | N/A |
| dim\_warehouse | wh\_postal\_code | Postal code of the warehouse's location | 123,456,782,468 | smallint | 4 | N | Flat File Ingestion | wh\_information\_ingestion | wh\_postal\_code | smallint | 4 | wh\_postal\_code from wh\_information\_ingestion | Detail | Monthly | Insert Into | No | N/A |
| dim\_warehouse | total\_land\_area | Total land area of the warehouse in sq meters | 1000000.1234, 10000000.1234, 10000000.1234 | decimal | (12, 4) | N | Flat File Ingestion | wh\_information\_ingestion | total\_land\_area | decimal | (12, 4) | total\_land\_area from wh\_information\_ingestion | Detail | Monthly | Insert Into | Rarely | SCD Type 2 |
| dim\_warehouse | operating\_hours | Operational warehouse of the warehouse | 10:00 - 20:00, 12:00 - 22:30, 8:30 - 80:00 | varchar | 14 | N | Flat File Ingestion | wh\_information\_ingestion | operating\_hours | varchar | 14 | operating\_hours from wh\_information\_ingestion | Detail | Monthly | Insert Into | Rarely | SCD Type 2 |
| dim\_warehouse | is\_active | 1 if active, 0 if not operational | 1, 0 | boolean |  | N | Flat File Ingestion | wh\_information\_ingestion | is\_active | boolean |  | is\_active from wh\_information\_ingestion | Detail | Monthly | Insert Into | Rarely | SCD Type 2 |

## **Data Model**

For the data warehouse design, dimensional modelling was used utilizing star schema. This was chosen for its simplicity since query would only require 1-level of table join avoiding multi-level joins and promotes efficient ETL process due to clear separation of processing of the dim and fact tables. In addition, this will also result in relatively faster query performance due to less complex join. Sample data models with the corresponding table design are shown in Figure 3.

A diagram of a database

Description automatically generated with medium confidence

**Figure 3.** *Sample Data Model for Each Type of Fact Table*

The complete list of data models for all fact tables of the proposed data warehouse design is also shown in Figures 4-5. Illustrated here are the dim tables that can be joined per fact table depending on the use case.

**A diagram of a diagram

Description automatically generated**

**Figure 4.** *Data Models for Data Warehouse Tables Also Included in Marketplace Data Mart*

*A diagram of a company

Description automatically generated*

**Figure 5.** *Data Models for Data Warehouse Tables Also Included in Warehouse Data Mart*

## **ETL Process**

For the ETL process of the data warehouse tables, the two processes outlined in figures 6-7 for dim and fact tables, respectively, will be followed as adapted from Schmitz’s (2014) methods.

For the extract part of the ETL process for dim tables, the data will be inserted into the staging table from the raw source tables. Each row will be then compared with the master table to choose rows for insertion or update (i.e., which are new data) to be inserted in the extract table. Depending if the data is clean or conformant to standards, they are either inserted into the error or changed table. Then, the master table will be updated to reflect the changes done for the specific data. All necessary transformation operations will be performed in the contents of the changed table according to each attribute before inserting it into the transform table. The content will then be compared to the final dim table to check if each row is for insertion or update and will be inserted into either the insert or update table. The content of these 2 tables will now be loaded into the final dim table by either inserting into or updating operations. Lastly, flag indicators can be updated, if any. There will be an audit table for each phase of the ETL process for tracking of operations done by each task. Data quality checks will also be incorporated into the ETL pipeline to monitor any unexpected data output and format.

A diagram of a diagram

Description automatically generated

**Figure 6.** *ETL Process for Dim Tables*

The ETL process for the 3 types of fact tables follows almost the same process. Data from the raw table and or ingestion are inserted into the staging table. Transformation operations are then performed if needed, and data are converted into the corresponding surrogate keys of the dim tables. The processed data will be inserted into the transform table. Depending on whether the data is conformant or not with the business rules, they can either be inserted into the error table or insert table. All clean data will then be loaded into the snapshot fact or detail fact tables. For the summary fact tables which contain aggregated data, note that data sources are the detail fact tables wherein there will also be checkpoints on whether the summary transform data are conformant or not to the business standards. The same practice for audit tables and data quality checks will also be reinforced for the fact tables.

A diagram of a flowchart

Description automatically generated

**Figure 7.** *ETL Process for Fact Tables (Detail, Summary, Snapshot)*

## **Metadata Documentation**

Before the end of the project, it is expected that a comprehensive table documentation will be provided such as the sample document shown in Table 5 containing the data type, description, and logic per column of all tables in the data warehouse and data marts. This is to help the end-users find the table they need easily and to avoid data misinterpretation.

**Table 5.** *Sample Documentation for the fact\_platform\_peformance\_summary table*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Mart** | **Table Name** | **Column** | **Data Type** | **Description** | **Logic** |
| Marketplace | fact\_platform\_peformance\_summary | date\_key | bigint | Foreign key to the dim\_date table, representing the relevant date for time-series metrics | distinct date\_key\_ from mdm.fact\_order\_detail |
| Marketplace | fact\_platform\_peformance\_summary | l1d\_ado | decimal(12,4) | Average daily orders for the last 1 day | count(order\_key)/1.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date = current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l7d\_ado | decimal(12,4) | Average daily orders for the last 7 days | count(order\_key)/7.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '7 days' and current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l30d\_ado | decimal(12,4) | Average daily orders for the last 30 days | count(order\_key)/30.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '30 days' and current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l1d\_adgmv | decimal(12,4) | Average daily GMV for the last 1 day | sum(gmv)/1.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date = current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l7d\_adgmv | decimal(12,4) | Average daily GMV for the last 7 days | sum(gmv)/7.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '7 days' and current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l30d\_adgmv | decimal(12,4) | Average daily GMV for the last 30 days | sum(gmv)/30.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '30 days' and current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l1d\_avg\_active\_buyers | decimal(12,4) | Average number of active buyers per day in the last 1 day | count(distinct buyer\_key)/1.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date = current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l7d\_avg\_active\_buyers | decimal(12,4) | Average number of active buyers per day in the last 7 days | count(distinct buyer\_key)/7.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '7 days' and current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l30d\_avg\_active\_buyers | decimal(12,4) | Average number of active buyers per day in the last 30 days | count(distinct buyer\_key)/7.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '30 days' and current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l1d\_avg\_active\_shops | decimal(12,4) | Average number of active shops per day in the last 1 day | count(distinct seller\_key)/1.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date = current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l7d\_avg\_active\_shops | decimal(12,4) | Average number of active shops per day in the last 7 days | count(distinct seller\_key)/7.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '7 days' and current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l30d\_avg\_active\_shops | decimal(12,4) | Average number of active shops per day in the last 30 days | count(distinct seller\_key)/7.0000 from mdm.fact\_order\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where b.date between current\_date - interval '30 days' and current\_date - interval '1 day' |
| Marketplace | fact\_platform\_peformance\_summary | l1d\_otd\_time | decimal(12,4) | Average on-time delivery time (in hours) for the last 1 day | (delivered\_time\_key - order\_create\_time\_key)/(count(parcel\_id) \* 3600.0000) from mdm.fact\_parcel\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where a.delivered\_time\_key is not null      and b.date = current\_date - interval '1 day' |
| Warehouse | fact\_platform\_peformance\_summary | l7d\_otd\_time | decimal(12,4) | Average on-time delivery time (in hours) for the last 7 days | (delivered\_time\_key - order\_create\_time\_key)/(count(parcel\_id) \* 3600.0000) from mdm.fact\_parcel\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where a.delivered\_time\_key is not null      and b.date  between current\_date - interval '7 days' and current\_date - interval '1 day' |
| Warehouse | fact\_platform\_peformance\_summary | l30d\_otd\_time | decimal(12,4) | Average on-time delivery time (in hours) for the last 30 days | (delivered\_time\_key - order\_create\_time\_key)/(count(parcel\_id) \* 3600.0000) from mdm.fact\_parcel\_detail a join mdm.dim\_date b on a.order\_create\_date\_key = b.date\_key where a.delivered\_time\_key is not null      and b.date   between current\_date - interval '30 days' and current\_date - interval '1 day' |

There is also the option to have a metadata data mart. One sample table can be around the historical workflow performance wherein the table records each task’s start time, end time, peak memory consumption, total memory consumption, and failure status. Another table can be created to track table usage such as storage consumption per table, a list of users who queried from the table and have access to a table, and the last 7-day query count for each table. This can help in data governance in making sure that only allowed users can access sensitive data and can help point tables for optimization activities.

# **Physical Design of Your Data Engineering Solution**

## **Architectural Overview and Implementation Details**

Table 6 presents the overview of the proposed architecture of the EWIP system. The platform will be mainly hosted in Google Cloud environments to serve as the centralized platform. This will be connected to PostgreSQL Cloud data sources via the Cloud Dataflow tool to perform daily batch ingestions. Apache Airflow will be used for the orchestration of dbt-based ETL tasks before the data warehouse tables are stored in Google BigQuery. Airflow was chosen as the orchestrator as an open-sourced and industry standard tool of choice also known for its comprehensive and high integrability features, while the dbt tool was preferred due to its SQL-based syntax and version control and built-in data quality checks capabilities. All tables will have partition date wherein daily added data will be stored in separate partitions to help in query optimization. End-user analytics can be performed in Power BI for visualizations, in Google BigQuery for ad hoc tasks, and in GSheet for reporting and basic operations.

**Table 6.** Proposed architecture and implementation of the EWIP system

|  |  |
| --- | --- |
| **Aspect** | **Remarks** |
| Infrastructure Details | - Platform hosted in Google Cloud environment as a centralized platform  - Allows parallel processing (for scalability) and integrability with other tools for simplified architecture |
| Data Sources | - 4 OLTP DB, 1 Flat File Ingestion  - Daily batch ingestion from PostgreSQL Cloud to Google Cloud via Cloud Dataflow tool |
| Storage Design | - Star schema design for DW  - Storage at Google BigQuery for the intermediate and final tables for the DW |
| ETL Process and Pipelines | - Deploying dbt in Google Cloud for the data transformation  - Apache Airflow orchestration via Google Cloud Composer for job scheduling |
| Optimization Techniques | - Implementation of partition columns by date in all tables  - Separation of tables into data marts |
| Security and Governance | - Built-in data governance for user- and row-level access control and data masking in the Google Cloud environment  - Data retention protocol for raw tables  - Utilize cost control features such as resource caps |
| End-User Analytics | - Visualizations via Power BI for its intuitive UI and simplicity  - Ad hoc querying via Google BigQuery  - GSheet for reporting and minimal analysis |

## **SQL Scripts for Implementations**

In the following subsections, the sample ETL code for the dim\_item table is shown. Currently, it is coded using Python and Pandas for demo purposes. Moving forward, dbt-based code will be used for the ETL tasks.

## ***Importing libraries***

|  |
| --- |
| # Importing libraries  import sqlite3  import pandas as pd  import numpy as np  import shutil  import os  from datetime import datetime |

## ***Extracting from Source Table***

|  |
| --- |
| # Establish connection to the source database  s1 = lpms\_conn.cursor()  lpms\_conn.commit()  s2 = ums\_conn.cursor()  ums\_conn.commit()  # Establish connection to the DW database  c = ewip\_dw\_conn.cursor()  ewip\_dw\_conn.commit()  # Extract the data from the source database  # listing  s1.execute('SELECT \* FROM listings')  listing = s1.fetchall()  # items  s1.execute('SELECT \* FROM items')  items = s1.fetchall()  # sellers  s2.execute('SELECT \* FROM sellers')  sellers = s2.fetchall()  # Convert each fetched data to pandas DataFrame  listing\_df = pd.DataFrame(listing, columns=['listing\_listing\_id', 'listing\_shop\_id', 'model\_id', 'model\_name', 'model\_description', 'category\_lvl\_1', 'category\_lvl\_2', 'create\_time\_listing', 'banned\_time', 'banned\_by','last\_modified\_time'])  items\_df = pd.DataFrame(items, columns=['sku\_id', 'items\_shop\_id','listing\_id', 'model\_id', 'item\_id','item\_description', 'stock\_qty', 'weight', 'length', 'width', 'height', 'item\_price', 'is\_active', 'create\_time', 'last\_modified\_time'])  sellers\_df = pd.DataFrame(sellers, columns=['user\_id', 'shop\_id', 'shop\_name', 'shop\_category', 'shop\_create\_time', 'is\_active\_shop', 'last\_modified\_time', 'is\_wh'])  # Merging the 3 DataFrames  staging = pd.merge(listing\_df, items\_df, on='model\_id', how='inner')  staging = pd.merge(staging, sellers\_df, left\_on='listing\_shop\_id', right\_on='shop\_id', how='inner')  # Insert data into S\_dim\_item  for index, row in staging.iterrows():      c.execute('''          INSERT INTO S\_dim\_item (sku\_id, shop\_id, listing\_id, model\_name, model\_description, category\_lvl\_1, category\_lvl\_2, model\_id, item\_id, item\_description, weight, length, width, height, item\_price, is\_active, create\_time, banned\_time, last\_modified\_time, is\_wh)          VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)          ''',          tuple(row[['sku\_id', 'shop\_id', 'listing\_id', 'model\_name', 'model\_description', 'category\_lvl\_1', 'category\_lvl\_2', 'model\_id', 'item\_id', 'item\_description', 'weight', 'length', 'width', 'height', 'item\_price', 'is\_active', 'create\_time', 'banned\_time', 'last\_modified\_time', 'is\_wh']]))  # Select all data from S\_dim\_item to verify insertion  c.execute('SELECT \* FROM S\_dim\_item')  # Fetch all data from the cursor  rows = c.fetchall()  ewip\_dw\_conn.commit()  df\_s\_dim\_item = pd.DataFrame(rows, columns=[desc[0] for desc in c.description])  df\_s\_dim\_item.head() |

## ***Compare New and Changed Data with the Master Table***

|  |
| --- |
| # New data from S\_dim\_item not present in M\_dim\_item  s\_table\_new\_data\_df = pd.read\_sql("""      SELECT \*      FROM S\_dim\_item      WHERE sku\_id NOT IN (SELECT sku\_id FROM M\_dim\_item)      """, ewip\_dw\_conn)  # Changed data in S\_dim\_item compared to M\_dim\_item  s\_table\_changed\_data\_df = pd.read\_sql("""      SELECT s.\*      FROM S\_dim\_item s      INNER JOIN M\_dim\_item m      ON s.sku\_id = m.sku\_id      WHERE s.model\_name != m.model\_name          OR s.model\_description != m.model\_description          OR s.category\_lvl\_1 != m.category\_lvl\_1          OR s.category\_lvl\_2 != m.category\_lvl\_2          OR s.item\_description != m.item\_description          OR s.weight != m.weight          OR s.length != m.length          OR s.width != m.width          OR s.height != m.height          OR s.item\_price != m.item\_price          OR s.is\_active != m.is\_active          OR s.banned\_time != m.banned\_time          OR s.is\_wh != m.is\_wh      """, ewip\_dw\_conn)  # Combine new and changed data  s\_table\_extract\_df = pd.concat([s\_table\_new\_data\_df, s\_table\_changed\_data\_df], ignore\_index=True)  # Display the extracted data  s\_table\_extract\_df.head() |

## ***Insert into Extract Tables***

|  |
| --- |
| # Delete data inside the X table first, if any  delete\_x\_data = c.execute('DELETE FROM X\_dim\_item')  ewip\_dw\_conn.commit()  c.execute('SELECT \* FROM X\_dim\_item')  c.fetchall()  # INSERT INTO X\_Items from the Staging table (S\_Table)  #Creating column list for insertion  cols = '","'.join([str(i) for i in s\_table\_extract\_df.columns.tolist()])  #Insert records one by one INTO X\_dim\_item  for i, row in s\_table\_extract\_df.iterrows():      sql = f'INSERT INTO X\_dim\_item ("{cols}") VALUES ({",".join(["?"] \* len(row))})'      c.execute(sql, tuple(row))  ewip\_dw\_conn.commit()  # Check if inserted  c.execute("SELECT \* FROM X\_dim\_item")  rows = c.fetchall()  df\_x\_dim\_item = pd.DataFrame(rows, columns=[desc[0] for desc in c.description])  df\_x\_dim\_item.head() |

## ***Clean X Table and Insert Into Error Table***

|  |
| --- |
| # Select rows with null values for non-nullable columns  x\_table\_null\_violations = pd.read\_sql("""  SELECT \* FROM X\_dim\_item  WHERE sku\_id IS NULL      OR shop\_id IS NULL      OR listing\_id IS NULL      OR model\_name IS NULL      OR model\_description IS NULL      OR category\_lvl\_1 IS NULL      OR model\_id IS NULL      OR item\_id IS NULL      OR item\_description IS NULL      OR item\_price IS NULL      OR is\_active IS NULL      OR create\_time IS NULL      OR last\_modified\_time IS NULL  """, ewip\_dw\_conn)  x\_table\_null\_violations['ErrorType'] = 'Null values in non-nullable columns'  # Select duplicated rows  x\_table\_duplicate\_sku\_df = pd.read\_sql("""  SELECT \* FROM X\_dim\_item  WHERE sku\_id IN (      SELECT sku\_id  FROM X\_dim\_item      GROUP BY sku\_id      HAVING COUNT(sku\_id ) > 1  )""", ewip\_dw\_conn)  x\_table\_duplicate\_sku\_df['ErrorType'] = 'Duplicate Company Name'  # Combine errors into one dataframe  x\_table\_errors\_df = pd.concat([x\_table\_null\_violations, x\_table\_duplicate\_sku\_df])  # Cleaning  # Set Unknown blank is\_wh to 0  update\_xitems = c.execute("UPDATE X\_dim\_item SET is\_wh = 0 WHERE is\_wh IS NULL")  c.execute("SELECT \* FROM X\_dim\_item")  c.fetchall()  # Other cleaning operations depending on the actual data  # Delete data inside E\_dim\_item first  delete\_eitems = c.execute('DELETE FROM E\_dim\_item')  c.execute("SELECT \* FROM E\_dim\_item")  c.fetchall()  # Creating column list for insertion  cols = '","'.join([str(i) for i in x\_table\_errors\_df.columns.tolist()])  # Insert records one by one INTO E\_dim\_item  for i, row in x\_table\_errors\_df.iterrows():      sql = "INSERT INTO E\_dim\_item (sku\_id, CompanyName, Phone, ErrorType) VALUES (" + ','.join(['?'] \* len(row)) + ")"      c.execute(sql, tuple(row))  ewip\_dw\_conn.commit()  # Check if inserted  c.execute("SELECT \* FROM E\_dim\_item")  c.fetchall() |

## ***Process Clean Data and Insert Into C Table***

|  |
| --- |
| #Select Clean Data  x\_table\_clean\_data\_df = pd.read\_sql("""  SELECT \*  FROM X\_dim\_item  WHERE sku\_id NOT IN (SELECT sku\_id FROM E\_dim\_item)  """, ewip\_dw\_conn)  #DELETE existing data in C table  delete\_citems = c.execute('DELETE FROM C\_dim\_item')  c.execute("SELECT \* FROM C\_dim\_item")  c.fetchall()  # Actual INSERT INTO C Table  # Creating column list for insertion  cols = '","'.join([str(i) for i in x\_table\_clean\_data\_df.columns.tolist()])  # Insert records one by one INTO C\_dim\_item  for i, row in x\_table\_clean\_data\_df.iterrows():      sql = f'INSERT INTO C\_dim\_item ("{cols}") VALUES ({",".join(["?"] \* len(row))})'      c.execute(sql, tuple(row))  ewip\_dw\_conn.commit()  # Check if inserted  c.execute("SELECT \* FROM C\_dim\_item")  rows = c.fetchall()  df\_c\_dim\_item = pd.DataFrame(rows, columns=[desc[0] for desc in c.description])  df\_c\_dim\_item.head() |

## ***Update Master Table (M Table)***

|  |
| --- |
| # Select All NEW From C Tables  c\_table\_new\_date\_df = pd.read\_sql("""  SELECT \* FROM C\_dim\_item c  WHERE c.sku\_id NOT IN  (SELECT m.sku\_id FROM M\_dim\_item m)  """, ewip\_dw\_conn)  # Creating column list for insertion  cols = '","'.join([str(i) for i in c\_table\_new\_date\_df.columns.tolist()])  # Insert records one by one INTO M\_dim\_item  for i, row in c\_table\_new\_date\_df.iterrows():      sql = f'INSERT INTO M\_dim\_item ("{cols}") VALUES ({",".join(["?"] \* len(row))})'      c.execute(sql, tuple(row))  ewip\_dw\_conn.commit()  # Check if inserted  c.execute("SELECT \* FROM M\_dim\_item")  rows = c.fetchall()  df\_m\_dim\_item = pd.DataFrame(rows, columns=[desc[0] for desc in c.description])  df\_m\_dim\_item.head()  # Processing Changed Data and Update the master Data.  #Select ALL Changed from C Table  c\_table\_changed\_data\_df = pd.read\_sql("""  SELECT c.\*  FROM C\_dim\_item AS c  JOIN M\_dim\_item AS m ON c.sku\_id = m.sku\_id  WHERE c.model\_name != m.model\_name          OR c.model\_description != m.model\_description          OR c.category\_lvl\_1 != m.category\_lvl\_1          OR c.category\_lvl\_2 != m.category\_lvl\_2          OR c.item\_description != m.item\_description          OR c.weight != m.weight          OR c.length != m.length          OR c.width != m.width          OR c.height != m.height          OR c.item\_price != m.item\_price          OR c.is\_active != m.is\_active          OR c.banned\_time != m.banned\_time          OR c.is\_wh != m.is\_wh  """, ewip\_dw\_conn)  # Delete from M\_dim\_item\_Test where data has changed  delete\_updated\_data = c.execute("""  DELETE FROM M\_dim\_item  WHERE sku\_id IN (          SELECT c.sku\_id          FROM C\_dim\_item AS c          JOIN M\_dim\_item AS m ON c.sku\_id = m.sku\_id          WHERE c.model\_name != m.model\_name                  OR c.model\_description != m.model\_description                  OR c.category\_lvl\_1 != m.category\_lvl\_1                  OR c.category\_lvl\_2 != m.category\_lvl\_2                  OR c.item\_description != m.item\_description                  OR c.weight != m.weight                  OR c.length != m.length                  OR c.width != m.width                  OR c.height != m.height                  OR c.item\_price != m.item\_price                  OR c.is\_active != m.is\_active                  OR c.banned\_time != m.banned\_time                  OR c.is\_wh != m.is\_wh  )  """)  print("Deleted from M\_dim\_item which are updated:")  print(c\_table\_changed\_data\_df)  print(f"\n")  # Verify deletion  result = c.execute("""          SELECT \* FROM M\_dim\_item WHERE sku\_id IN (                  SELECT c.sku\_id                  FROM C\_dim\_item AS c                  JOIN M\_dim\_item AS m ON c.sku\_id = m.sku\_id                  WHERE c.model\_name != m.model\_name                          OR c.model\_description != m.model\_description                          OR c.category\_lvl\_1 != m.category\_lvl\_1                          OR c.category\_lvl\_2 != m.category\_lvl\_2                          OR c.item\_description != m.item\_description                          OR c.weight != m.weight                          OR c.length != m.length                          OR c.width != m.width                          OR c.height != m.height                          OR c.item\_price != m.item\_price                          OR c.is\_active != m.is\_active                          OR c.banned\_time != m.banned\_time                          OR c.is\_wh != m.is\_wh          )  """).fetchall()  ewip\_dw\_conn.commit()  df\_m\_dim\_item = pd.DataFrame(result, columns=[desc[0] for desc in c.description])  df\_m\_dim\_item.head()  # INSERT Clean Data INTO M Table with changed data  # Creating column list for insertion  cols = '","'.join([str(i) for i in c\_table\_changed\_data\_df.columns.tolist()])  # Insert records one by one INTO M\_dim\_item  for i, row in c\_table\_changed\_data\_df.iterrows():      sql = 'INSERT INTO M\_dim\_item ("' + cols + '") VALUES (' + ','.join(['?'] \* len(row)) + ')'      c.execute(sql, tuple(row))  ewip\_dw\_conn.commit()  # Check if inserted  c.execute("SELECT \* FROM M\_dim\_item")  rows = c.fetchall()  df\_m\_dim\_item = pd.DataFrame(rows, columns=[desc[0] for desc in c.description])  df\_m\_dim\_item.head() |

## ***Initiate Transform Processes***

|  |
| --- |
| # Select data from C and Transform to DW Format  # For this case, no needed transformation  c\_table\_data\_df = pd.read\_sql("""      SELECT \* FROM C\_dim\_item  """, ewip\_dw\_conn)  # INSERT INTO T Table  # DELETE existing data in T table  delete\_titems = c.execute('DELETE FROM T\_dim\_item')  c.execute("SELECT \* FROM T\_dim\_item")  c.fetchall()  # Actual INSERT C Table data into I Table  # Creating column list for insertion  cols = '","'.join([str(i) for i in c\_table\_data\_df.columns.tolist()])  # Insert records one by one INTO T\_dim\_item  for i, row in c\_table\_data\_df.iterrows():      sql = f'INSERT INTO T\_dim\_item ("{cols}") VALUES ({",".join(["?"] \* len(row))})'      c.execute(sql, tuple(row))  ewip\_dw\_conn.commit()  # Check if inserted  pd.read\_sql("SELECT \* FROM T\_dim\_item", ewip\_dw\_conn).head() |

## ***Select Data from T Table and Insert to I and U Table***

|  |
| --- |
| # SELECT New data from the T Table  t\_table\_new\_data\_df = pd.read\_sql("""  SELECT t.\*  FROM t\_dim\_item t  LEFT JOIN dim\_item d ON t.sku\_id = d.sku\_id  WHERE d.sku\_id IS NULL  """, ewip\_dw\_conn)  t\_table\_new\_data\_df['is\_latest\_record'] = 1  # INSERT New Data INTO I Table  # DELETE existing data in I table  delete\_i\_dim\_item= c.execute('DELETE FROM I\_dim\_item')  c.execute("SELECT \* FROM I\_dim\_item")  c.fetchall()  # Creating column list for insertion  cols = '","'.join([str(i) for i in t\_table\_new\_data\_df.columns.tolist()])  # Insert records one by one INTO I\_dim\_item  for i, row in t\_table\_new\_data\_df.iterrows():      sql = f'INSERT INTO I\_dim\_item ("{cols}") VALUES ({",".join(["?"] \* len(row))})'      c.execute(sql, tuple(row))  ewip\_dw\_conn.commit()  # Check if inserted  pd.read\_sql("SELECT \* FROM I\_dim\_item", ewip\_dw\_conn).head()  # INSERT Changed Data INTO U Table  #Select Changed from T  t\_table\_changed\_data\_df = pd.read\_sql('''  SELECT t.\*  FROM t\_dim\_item t  INNER JOIN dim\_item d  ON t.sku\_id = d.sku\_id  WHERE (t.model\_name != d.model\_name          OR t.model\_description != d.model\_description          OR t.category\_lvl\_1 != d.category\_lvl\_1          OR t.category\_lvl\_2 != d.category\_lvl\_2          OR t.item\_description != d.item\_description          OR t.weight != d.weight          OR t.length != d.length          OR t.width != d.width          OR t.height != d.height          OR t.item\_price != d.item\_price          OR t.is\_active != d.is\_active          OR t.banned\_time != d.banned\_time          OR t.is\_wh != d.is\_wh)      AND is\_latest\_record = 1  ''', ewip\_dw\_conn)  t\_table\_changed\_data\_df['is\_latest\_record'] = 1  #Delete existing data from the U Table first  delete\_uitems = c.execute('DELETE FROM U\_dim\_item')  c.execute("SELECT \* FROM U\_dim\_item")  c.fetchall()  # Actual INSERT of Changed data into U table  #INSERT Changed Data INTO U  #Creating column list for insertion  cols = "','".join([str(i) for i in t\_table\_changed\_data\_df.columns.tolist()])  #Insert records one by one INTO U\_dim\_item  for i, row in t\_table\_changed\_data\_df.iterrows():      sql = "INSERT INTO U\_dim\_item ('" + cols + "') VALUES (" + ','.join(['?'] \* len(row)) + ")"      c.execute(sql, tuple(row))  ewip\_dw\_conn.commit()  # Check if inserted  pd.read\_sql("SELECT \* FROM U\_dim\_item", ewip\_dw\_conn) |

## ***Insert I Table into D Table***

|  |
| --- |
| # INSERT I INTO D Table  # Get Max Warehouse Key  maxkey = pd.read\_sql('''SELECT COALESCE(MAX(sku\_key), 0) as MAX FROM dim\_item''', ewip\_dw\_conn)  # Select Data to be INSERTED from I Table  i\_table\_data\_df = pd.read\_sql("SELECT \* FROM I\_dim\_item", ewip\_dw\_conn)  # Identify the next set of ItemKey’s to be assigned to the New Data from I Table  if not i\_table\_data\_df.empty and not maxkey.empty:      start\_value = pd.to\_numeric(maxkey.iloc[0]).values + 1      i\_table\_data\_df['sku\_key'] = np.arange(start\_value, start\_value + len(i\_table\_data\_df))  else:      print("Either the data table or maxkey is empty. No operation performed.")  # Rearrange according to the D table format of columns  i\_table\_data\_df = i\_table\_data\_df[['sku\_key', 'sku\_id','shop\_id', 'listing\_id', 'model\_name', 'model\_description', 'category\_lvl\_1', 'category\_lvl\_2','model\_id', 'item\_id', 'item\_description', 'weight', 'length', 'width', 'height','item\_price', 'is\_active', 'create\_time', 'banned\_time', 'last\_modified\_time','is\_wh', 'is\_latest\_record']]  # Changing last\_modified\_time to current time  i\_table\_data\_df[['last\_modified\_time']] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')  i\_table\_data\_df.head()  # Now INSERT into D Table  # Creating column list for insertion  cols = '","'.join([str(i) for i in i\_table\_data\_df.columns.tolist()])  # Insert records one by one INTO D\_dim\_item  for i, row in i\_table\_data\_df.iterrows():      sql = f'INSERT INTO dim\_item ("{cols}") VALUES ({",".join(["?"] \* len(row))})'      c.execute(sql, tuple(row))  ewip\_dw\_conn.commit()  # Check if inserted  pd.read\_sql("SELECT \* FROM dim\_item", ewip\_dw\_conn).head() |

## ***Insert U Table Data (Type 2) into D Table***

|  |
| --- |
| # Get Max Warehouse Key  maxkey = pd.read\_sql('''SELECT MAX(sku\_key) as MAX FROM dim\_item''', ewip\_dw\_conn)  # Select Data to be INSERTED from U Table  u\_table\_type2\_data\_df = pd.read\_sql('''      SELECT u.\*      FROM u\_dim\_item u      INNER JOIN dim\_item d      ON u.sku\_id = d.sku\_id      WHERE (u.model\_name != d.model\_name              OR u.model\_description != d.model\_description              OR u.category\_lvl\_1 != d.category\_lvl\_1              OR u.category\_lvl\_2 != d.category\_lvl\_2              OR u.item\_description != d.item\_description              OR u.weight != d.weight              OR u.length != d.length              OR u.width != d.width              OR u.height != d.height              OR u.item\_price != d.item\_price              OR u.is\_active != d.is\_active              OR u.banned\_time != d.banned\_time              OR u.is\_wh != d.is\_wh)          AND d.is\_latest\_record = 1  ''', ewip\_dw\_conn)  # Identify the next set of Item\_Key's to be assigned to the New Data from U Table  u\_table\_type2\_data\_df['sku\_key'] = np.arange(pd.to\_numeric(maxkey.iloc[0].values) + 1,                                                   pd.to\_numeric(maxkey.iloc[0].values) + 1 + len(u\_table\_type2\_data\_df))  # Rearrange according to the D table format of columns  u\_table\_type2\_data\_df = u\_table\_type2\_data\_df[['sku\_key', 'sku\_id','shop\_id', 'listing\_id', 'model\_name', 'model\_description', 'category\_lvl\_1', 'category\_lvl\_2','model\_id', 'item\_id', 'item\_description', 'weight', 'length', 'width', 'height','item\_price', 'is\_active', 'create\_time', 'banned\_time', 'last\_modified\_time','is\_wh', 'is\_latest\_record']]  # Changing last\_modified\_time to current time  u\_table\_type2\_data\_df[['last\_modified\_time']] = datetime.now().strftime('%Y-%m-%d %H:%M:%S')  u\_table\_type2\_data\_df.head()  # Now INSERT CHANGED data (Type 2) from U Table into D Table  # Creating column list for insertion  cols = '","'.join([str(i) for i in u\_table\_type2\_data\_df.columns.tolist()])  # Insert records from the U Table one by one INTO D\_dim\_item  for i, row in u\_table\_type2\_data\_df.iterrows():      sql = f'INSERT INTO dim\_item ("{cols}") VALUES ({",".join(["?"] \* len(row))})'      c.execute(sql, tuple(row))  ewip\_dw\_conn.commit()  # Check if inserted  pd.read\_sql("SELECT \* FROM dim\_item", ewip\_dw\_conn).head() |

## ***Update Indicators to Current in D Table***

|  |
| --- |
| # Update is\_latest\_record indicator in D Table  c.execute('''  UPDATE dim\_item  SET is\_latest\_record = 0  WHERE (sku\_key, last\_modified\_time) NOT IN (      SELECT sku\_key, MAX(last\_modified\_time)      FROM dim\_item      GROUP BY sku\_key  )  ''')  ewip\_dw\_conn.commit()  pd.read\_sql("SELECT \* FROM dim\_item", ewip\_dw\_conn).head() |

## **Proposed Data Warehouse Design**

The planned tables in the proposed data marts are listed in Table 7. Note that all tables in the data marts are also present in the data warehouse. The complete list of tables with their corresponding attributes can also be browsed in Figures 8-11.

**Table 7.** *Complete List of Tables in the Proposed Data Warehouse and Data Marts*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No. | Data Mart | Table Name | No. | Data Mart | Table Name |
| 1 | MDM | dim\_buyer | 19 | MDM | fact\_platform\_performance\_summary |
| 2 | MDM | dim\_date | 20 | MDM | fact\_seller\_performance\_summary |
| 3 | MDM | dim\_item | 21 | MDM | fact\_sku\_performance\_summary |
| 4 | MDM | dim\_location | 22 | WDM | dim\_purchase\_order\_asn |
| 5 | MDM | dim\_operator | 23 | WDM | dim\_staff |
| 6 | MDM | dim\_order | 24 | WDM | dim\_storage |
| 7 | MDM | dim\_order\_parcel | 25 | WDM | dim\_supplier |
| 8 | MDM | dim\_parcel\_status | 26 | WDM | dim\_task |
| 9 | MDM | dim\_payment | 27 | WDM | dim\_warehouse |
| 10 | MDM | dim\_seller | 28 | WDM | fact\_attendance\_snapshot |
| 11 | MDM | dim\_shipping | 29 | WDM | fact\_inbound\_sku\_detail |
| 12 | MDM | dim\_time | 30 | WDM | fact\_inbound\_task\_detail |
| 13 | MDM | dim\_voucher | 31 | WDM | fact\_outbound\_order\_detail |
| 14 | MDM | dim\_voucher\_mix | 32 | WDM | fact\_outbound\_sku\_detail |
| 15 | MDM | fact\_order\_detail | 33 | WDM | fact\_outbound\_task\_detail |
| 16 | MDM | fact\_order\_item\_detail | 34 | WDM | fact\_staff\_prod\_summary |
| 17 | MDM | fact\_parcel\_detail | 35 | WDM | fact\_storage\_inventory\_snapshot |
| 18 | MDM | fact\_parcel\_status\_snapshot | 36 | WDM | fact\_warehouse\_summary |

A screenshot of a computer program

Description automatically generated

**Figure** **8**. *Fact Tables and Their Attributes Under Marketplace Data Mart*

A screenshot of a computer screen

Description automatically generated

**Figure** **9**. *Dim Tables and Their Attributes Under Marketplace Data Mart*

A group of white squares with black text

Description automatically generated

**Figure 10.** *Fact Tables and Their Attributes Under Warehouse Data Mart*

A screenshot of a computer

Description automatically generated

**Figure 11.** *Dim Tables and Their Attributes Under Warehouse Data Mart*

# **Reports Generation**

The data warehouse was designed to accommodate the crucial questions needed by the senior management, operations, and business development team. For report generation, several fact tables with varying granularities depending on the need are available for ad hoc analysis and regular reports. The 5 priority business questions can be queried from the available summary fact tables as follows:

|  |
| --- |
| 1**) PLATFORM PERFORMANCE.** What are the monthly historical platform performance and month-on-month (MoM) growth in terms of average daily order (ADO), average daily gross merchandise value (ADGMV), average active buyers, average active shops, and average order-to-delivery (OTD) time? |
| WITH monthly\_platform\_metrics AS (      SELECT          strftime('%Y-%m', d.date) AS month,          SUM(f."1d\_ado") AS total\_ado,          SUM(f."1d\_adgmv") AS total\_adgmv,          SUM(f."1d\_avg\_active\_buyers") AS total\_active\_buyers,          SUM(f."1d\_avg\_active\_shops") AS total\_active\_shops,          SUM(f."1d\_otd\_time") AS total\_otd\_time,          strftime('%d', MAX(d.date)) AS days\_in\_month      FROM fact\_platform\_performance\_summary f      JOIN dim\_date d ON f.date\_key = d.date\_key      GROUP BY strftime('%Y-%m', d.date)  )  SELECT      month,      total\_ado / days\_in\_month AS avg\_daily\_ado,      total\_adgmv / days\_in\_month AS avg\_daily\_adgmv,      total\_active\_buyers / days\_in\_month AS avg\_daily\_active\_buyers,      total\_active\_shops / days\_in\_month AS avg\_daily\_active\_shops,      total\_otd\_time / days\_in\_month AS avg\_daily\_otd\_time,      (total\_ado / days\_in\_month) - LAG(total\_ado / days\_in\_month) OVER (ORDER BY month) AS ado\_growth,      (total\_adgmv / days\_in\_month) - LAG(total\_adgmv / days\_in\_month) OVER (ORDER BY month) AS adgmv\_growth,      (total\_active\_buyers / days\_in\_month) - LAG(total\_active\_buyers / days\_in\_month) OVER (ORDER BY month) AS active\_buyers\_growth,      (total\_active\_shops / days\_in\_month) - LAG(total\_active\_shops / days\_in\_month) OVER (ORDER BY month) AS active\_shops\_growth,      (total\_otd\_time / days\_in\_month) - LAG(total\_otd\_time / days\_in\_month) OVER (ORDER BY month) AS otd\_growth  FROM monthly\_platform\_metrics  ORDER BY month; |
| 2) **ITEMS PERFORMANCE.** What are the top item categories in terms of monthly ADO and ADGMV? Which item categories have the highest month-on-month ADO and ADGMV growth? |
| WITH monthly\_category\_metrics AS (      SELECT          strftime('%Y-%m', d.date) AS month,          i.category\_lvl\_1 AS category,          SUM(f."1d\_ado") AS total\_ado,          SUM(f."1d\_adgmv") AS total\_adgmv,          strftime('%d', MAX(d.date)) AS days\_in\_month      FROM fact\_sku\_performance\_summary f      JOIN dim\_date d ON f.date\_key = d.date\_key      JOIN dim\_item i ON f.sku\_id = i.sku\_id      GROUP BY strftime('%Y-%m', d.date), i.category\_lvl\_1  ),  category\_rankings AS (      SELECT          month,          category,          total\_ado / days\_in\_month AS avg\_daily\_ado,          total\_adgmv / days\_in\_month AS avg\_daily\_adgmv,          RANK() OVER (PARTITION BY month ORDER BY total\_ado / days\_in\_month DESC) AS ado\_rank,          RANK() OVER (PARTITION BY month ORDER BY total\_adgmv / days\_in\_month DESC) AS adgmv\_rank,          (total\_ado / days\_in\_month) - LAG(total\_ado / days\_in\_month) OVER monthly\_window AS ado\_growth,          (total\_adgmv / days\_in\_month) - LAG(total\_adgmv / days\_in\_month) OVER monthly\_window AS adgmv\_growth      FROM monthly\_category\_metrics      WINDOW monthly\_window AS (PARTITION BY category ORDER BY month)  )  SELECT \*,      RANK() OVER (PARTITION BY month ORDER BY ado\_growth DESC) AS ado\_growth\_rank,      RANK() OVER (PARTITION BY month ORDER BY adgmv\_growth DESC) AS adgmv\_growth\_rank  FROM category\_rankings  WHERE ado\_rank <= 10 OR adgmv\_rank <= 10  ORDER BY month, ado\_rank, adgmv\_rank; |
| 3) **SHOP PERFORMANCE.** What are the top-performing shop categories in terms of monthly ADO and ADGMV? What shop categories contribute to more than 10% of the platform ADGMV? |
| WITH monthly\_shop\_category\_metrics AS (      SELECT          strftime('%Y-%m', d.date) AS month,          s.shop\_category,          SUM(f."1d\_ado") AS total\_ado,          SUM(f."1d\_adgmv") AS total\_adgmv,          strftime('%d', MAX(d.date)) AS days\_in\_month      FROM fact\_seller\_performance\_summary f      JOIN dim\_date d ON f.date\_key = d.date\_key      JOIN dim\_seller s ON f.shop\_id = s.shop\_id      GROUP BY strftime('%Y-%m', d.date) , s.shop\_category  ),  shop\_category\_contributions AS (      SELECT          month,          shop\_category,          total\_ado / days\_in\_month AS avg\_daily\_ado,          total\_adgmv / days\_in\_month AS avg\_daily\_adgmv,          SUM(total\_adgmv) OVER (PARTITION BY month) / days\_in\_month AS total\_platform\_adgmv,          RANK() OVER (PARTITION BY month ORDER BY total\_adgmv / days\_in\_month DESC) AS adgmv\_rank      FROM monthly\_shop\_category\_metrics  )  SELECT      month,      shop\_category,      avg\_daily\_ado,      avg\_daily\_adgmv,      (avg\_daily\_adgmv / total\_platform\_adgmv) \* 100 AS adgmv\_percentage  FROM shop\_category\_contributions  WHERE adgmv\_rank <= 10 OR (avg\_daily\_adgmv / total\_platform\_adgmv) > 0.1  ORDER BY month, adgmv\_rank; |
| 4) **WAREHOUSE PERFORMANCE.** What is the monthly historical performance and MoM growth of the overall warehouse and each warehouse in terms of ADO, average daily item (ADI) count, productivity rate, and idle rate? |
| WITH monthly\_warehouse\_metrics AS (      SELECT          strftime('%Y-%m', d.date) AS month,          w.wh\_key,          SUM(w."1d\_ado") AS total\_ado,          SUM(w."1d\_adi") AS total\_adi,          SUM(w."1d\_prod\_rate") AS total\_prod\_rate,          SUM(w."1d\_idle\_rate") AS total\_idle\_rate,          strftime('%d', MAX(d.date)) AS days\_in\_month      FROM fact\_warehouse\_summary w      JOIN dim\_date d ON w.date\_key = d.date\_key      GROUP BY strftime('%Y-%m', d.date), w.wh\_key  ),  overall\_warehouse\_metrics AS (      SELECT          month,          'Overall' AS wh\_key,          SUM(total\_ado) AS total\_ado,          SUM(total\_adi) AS total\_adi,          SUM(total\_prod\_rate) AS total\_prod\_rate,          SUM(total\_idle\_rate) AS total\_idle\_rate,          MAX(days\_in\_month) AS days\_in\_month      FROM monthly\_warehouse\_metrics      GROUP BY month  )  SELECT      wh\_key,      month,      total\_ado / days\_in\_month AS avg\_daily\_ado,      total\_adi / days\_in\_month AS avg\_daily\_adi,      total\_prod\_rate / days\_in\_month AS avg\_daily\_prod\_rate,      total\_idle\_rate / days\_in\_month AS avg\_daily\_idle\_rate,      (total\_ado / days\_in\_month) - LAG(total\_ado / days\_in\_month) OVER (PARTITION BY wh\_key ORDER BY month) AS ado\_growth,      (total\_adi / days\_in\_month) - LAG(total\_adi / days\_in\_month) OVER (PARTITION BY wh\_key ORDER BY month) AS adi\_growth,      (total\_prod\_rate / days\_in\_month) - LAG(total\_prod\_rate / days\_in\_month) OVER (PARTITION BY wh\_key ORDER BY month) AS prod\_rate\_growth,      (total\_idle\_rate / days\_in\_month) - LAG(total\_idle\_rate / days\_in\_month) OVER (PARTITION BY wh\_key ORDER BY month) AS idle\_rate\_growth  FROM (      SELECT \* FROM monthly\_warehouse\_metrics      UNION ALL      SELECT \* FROM overall\_warehouse\_metrics  ) combined\_metrics  ORDER BY month, wh\_key; |
| 5) **STAFF PERFORMANCE.** Who are the warehouse staff who have last 30-day productivity less than 90% of the average rate of the top 10 staff? |
| WITH staff\_productivity AS (      SELECT          s.staff\_key,          s.staff\_email,          SUM(f.prod\_rate) AS total\_prod\_rate,          30.000 AS days\_in\_month,          SUM(f.prod\_rate) / 30.000 AS avg\_prod\_rate      FROM fact\_staff\_prod\_summary f      JOIN dim\_date d ON f.date\_key = d.date\_key      JOIN dim\_staff s ON f.staff\_key = s.staff\_key      WHERE d.date >= DATE('now', '-30 days')      GROUP BY s.staff\_key  ),  top\_10\_avg\_productivity AS (      SELECT          AVG(avg\_prod\_rate) AS avg\_top\_10\_productivity      FROM (          SELECT              avg\_prod\_rate          FROM staff\_productivity          ORDER BY avg\_prod\_rate DESC          LIMIT 10      ) top\_10  )  SELECT      sp.staff\_key,      sp.staff\_email,      sp.avg\_prod\_rate,      t10.avg\_top\_10\_productivity  FROM staff\_productivity sp, top\_10\_avg\_productivity t10  WHERE sp.avg\_prod\_rate < 0.9 \* t10.avg\_top\_10\_productivity  ORDER BY sp.avg\_prod\_rate; |

In the proposed intelligence platform, there are 3 recommended ways to consume the data. It is highly recommended to utilize data visualization tools such as Power BI to show the data in easy-to-comprehend visualizations such as the sample dashboard presented in Figure 12. For other reporting use cases, outputs can also be generated in Google Sheets wherein other simple operations and modifications can be performed. Lastly, ad hoc queries can be performed in the Google BigQuery environment for special analysis and data pulls.

A graph and diagram of different types of graphs

Description automatically generated with medium confidence

**Figure 12.** *Sample Power BI Dashboard for Platform Performance Reporting*

***Table 8.*** *Gantt Chart for the EWIP Implementation*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | Legend: |  | **On track** | | | |  | **Low risk** | | | |  | **Med risk** | | | |  | **High risk** | | | |  | **Unassigned** | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Project start date: | 06/01/2025 |  |  |  |  | **January** | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **February** | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Scrolling increment: | 0 |  |  |  |  | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 1 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Milestone description** | **Category** | **Progress** | **Start** | **Days** |  | M | T | W | T | F | S | S | M | T | W | T | F | S | S | M | T | W | T | F | S | S | M | T | W | T | F | S | S | M | T | W | T | F | S | S | M | T | W | T | F | S | S | M | T | W | T | F | S | S | M | T | W | T | F | S | S |
| **Requirement Analysis** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Identify key stakeholders | On Track | **100%** | 06/01/2025 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Gathering stakeholder and business requirements | On Track | **100%** | 06/01/2025 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Source system analysis (architecture, data, data type) | On Track | **100%** | 08/01/2025 | 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Completing project charter | On Track | **100%** | 15/01/2025 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Initial stakeholder alignment | Milestone | **100%** | 17/01/2025 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Data Modelling** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Designing dim tables | Low Risk | **60%** | 20/01/2025 | 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Designing fact tables | Med Risk | **50%** | 20/01/2025 | 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Designing summary and snapshot fact tables | High Risk | **33%** | 23/01/2025 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Stakeholder alignment | Milestone |  | 24/01/2025 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **ETL Implementation** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ETL for the dim tables | On Track |  | 27/01/2025 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ETL for the fact tables | On Track |  | 27/01/2025 | 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ETL for the summary fact tables | On Track |  | 05/02/2025 | 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Stakeholder alignment on logic | Milestone |  | 07/02/2025 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Testing and Validation** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Initial testing (low-volume data) | On Track |  | 10/02/2025 | 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Possible debugging | On Track |  | 10/02/2025 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Volume testing (high-volume) | On Track |  | 17/02/2025 | 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Possible debugging and optimization | On Track |  | 17/02/2025 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **User Acceptance Test** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Production testing for selected users | On Track |  | 24/02/2025 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Feedback alignment | On Track |  | 03/03/2025 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Actions for feedback | On Track |  | 03/03/2025 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Deployment and Cascade** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Process and technical documentation | On Track |  | 10/03/2025 | 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| DW Cascade to all users | On Track |  | 14/03/2025 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Closing the project | Milestone |  | 14/03/2025 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

# **Project Management**

Figure 8 shows Gannt chart for the project detailing each specific steps per major phase of the project. In this chart, each activity can be marked as ‘On Track’, ‘Low Risk’, ‘Med Risk’, or ‘High Risk’ to mark the status of each task. Note that the corresponding Excel file is highly configurable in cases where there are needed deadline push backs for any phase of the project.

# **Conclusion**

This case study proposes a data warehouse and analytical platform as a solution to enhance the performance of analytical data queries. This is particularly essential for e-commerce businesses that lack online analytical processing (OLAP) databases and rely on transactional databases for analytics and reporting. Data from four PostgreSQL-based information systems were transformed into dimensional models following a star schema. This resulted in 20 dimension tables, 8 detailed fact tables, 5 summary fact tables, and 3 snapshot fact tables which can be stored in the data warehouse, marketplace data mart, and warehouse data mart.

The proposed data warehouse solution offers centralized data storage and a querying platform via the Google Cloud environment, enabling standardized data reporting across various key metrics. Tools such as dbt for data transformations, Airflow for workflow orchestration, and Power BI and Google Sheets for reporting can be integrated into the system to support end-to-end data engineering and reporting needs. This solution is specifically designed and optimized for analytical requirements, including historical analysis and business intelligence, providing a competitive advantage through data-driven decision-making.

# **References**

Candelon, F., Reichert, T., Duranton, S., Charme, R., Carlo, D., & De Bondt, M. (2020). *The Rise of The AI-Powered Company in the Postcrisis World*.

Dragomirov, N. (2020). E-Commerce Platforms and Supply Chain Management – Functionalities Study. *Economic Alternatives*, *26*(2), 250–261. https://doi.org/10.37075/EA.2020.2.04

Fabri, L., & Valverde Márquez, I. (2020). Will e-commerce dominate physical store? *International Journal of Technology for Business*, *2*(1), 23–20. https://doi.org/10.5281/zenodo.3894442

Statista. (2024). *E-commerce market volume in Southeast Asia from 2022 to 2024, with forecasts to 2030, by country.*

Schmitz, M. (2014). *High Performance Data Warehousing*. UCI Irvine Data Warehousing Notes.

# **Appendices**

# **ERD for LPMS**

A diagram of a flowchart

Description automatically generated

# **ERD for UMS**

A diagram of a diagram

Description automatically generated

# **ERD for OMS**

A diagram of a flowchart

Description automatically generated

# **ERD for WMS**

A diagram of a flowchart

Description automatically generated

# **Relational Model for LPMS**

A diagram of a computer

Description automatically generated

# **Relational Model for UMS**

A diagram of a server

Description automatically generated

# **Relational Model for OMS**

A diagram of a server

Description automatically generated

# **Relational Model for WMS**

A diagram of a computer

Description automatically generated