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

List of Tables

I. 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.

II. 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.

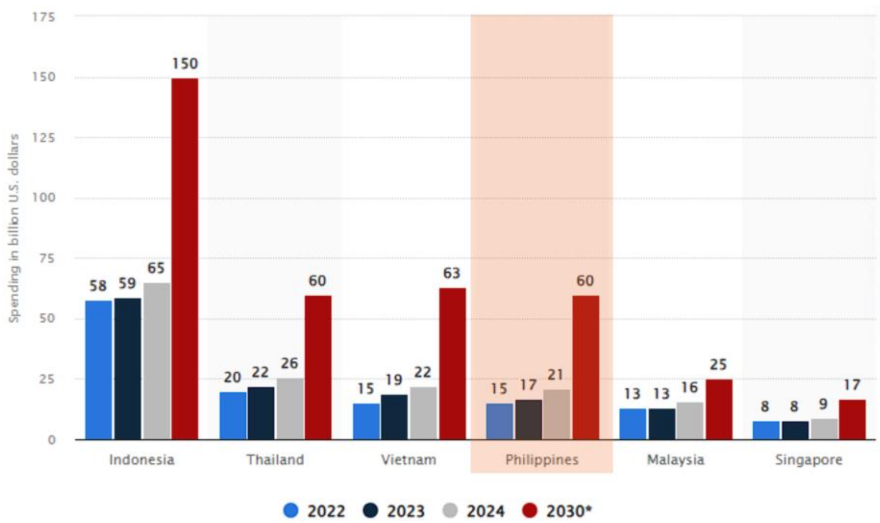


Figure 1. *E-Commerce Market Volume Outlook and Forecast (Statista, 2024)*

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.

III. 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?

IV. 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.

V. Proposed System Architecture

a. High Level System Architecture

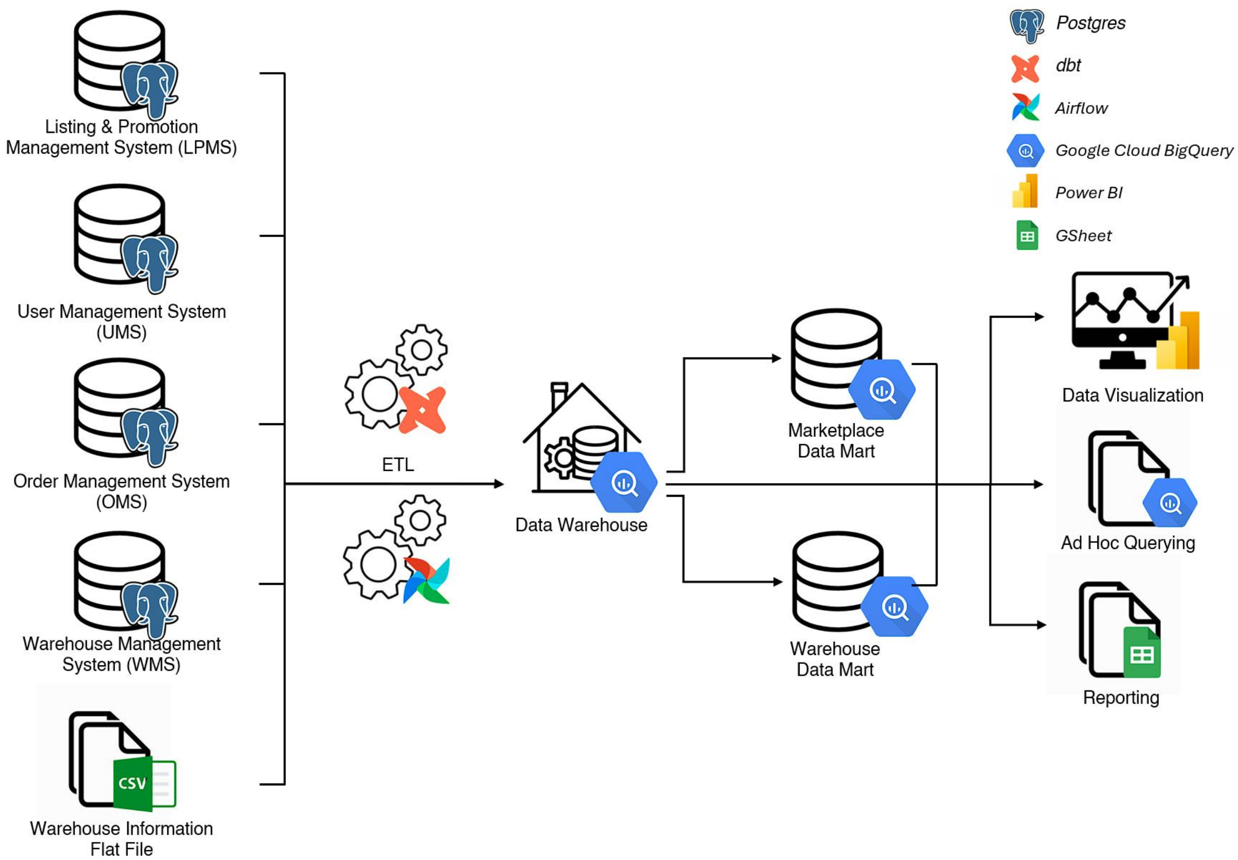


Figure 2. *System Architecture Design for the EWIP Project*

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.

b. 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.

c. 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.

d. 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

e. 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.

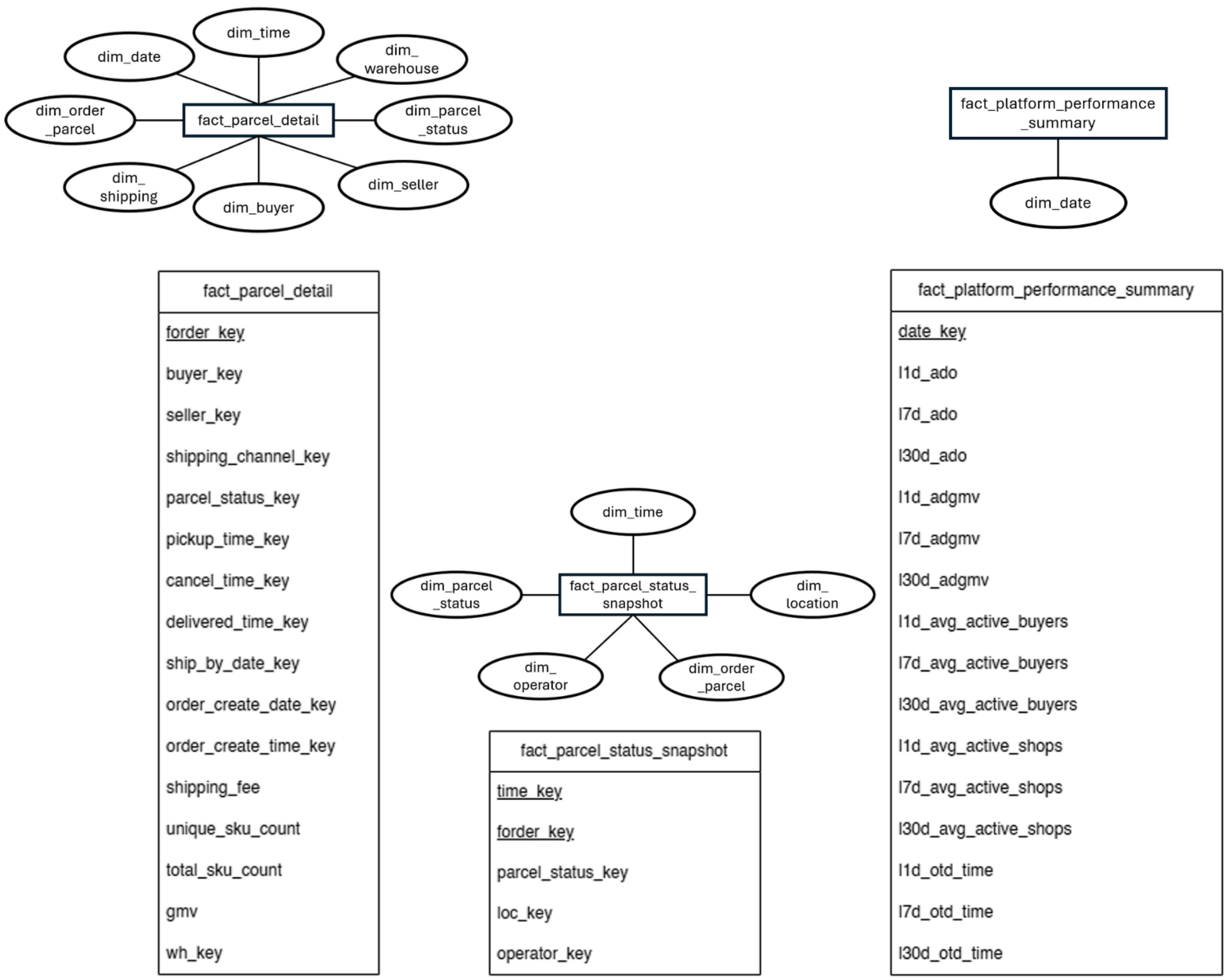


Figure 3. *Sample Data Model for Each Type of Fact Table*

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.

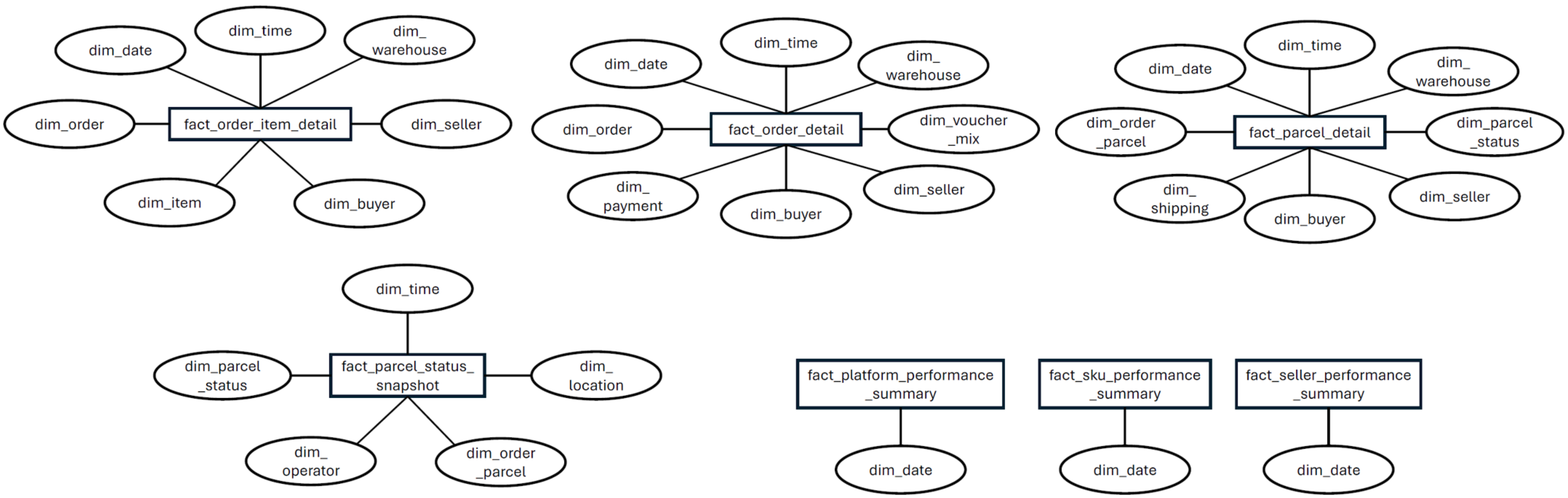


Figure 4. *Data Models for Data Warehouse Tables Also Included in Marketplace Data Mart*

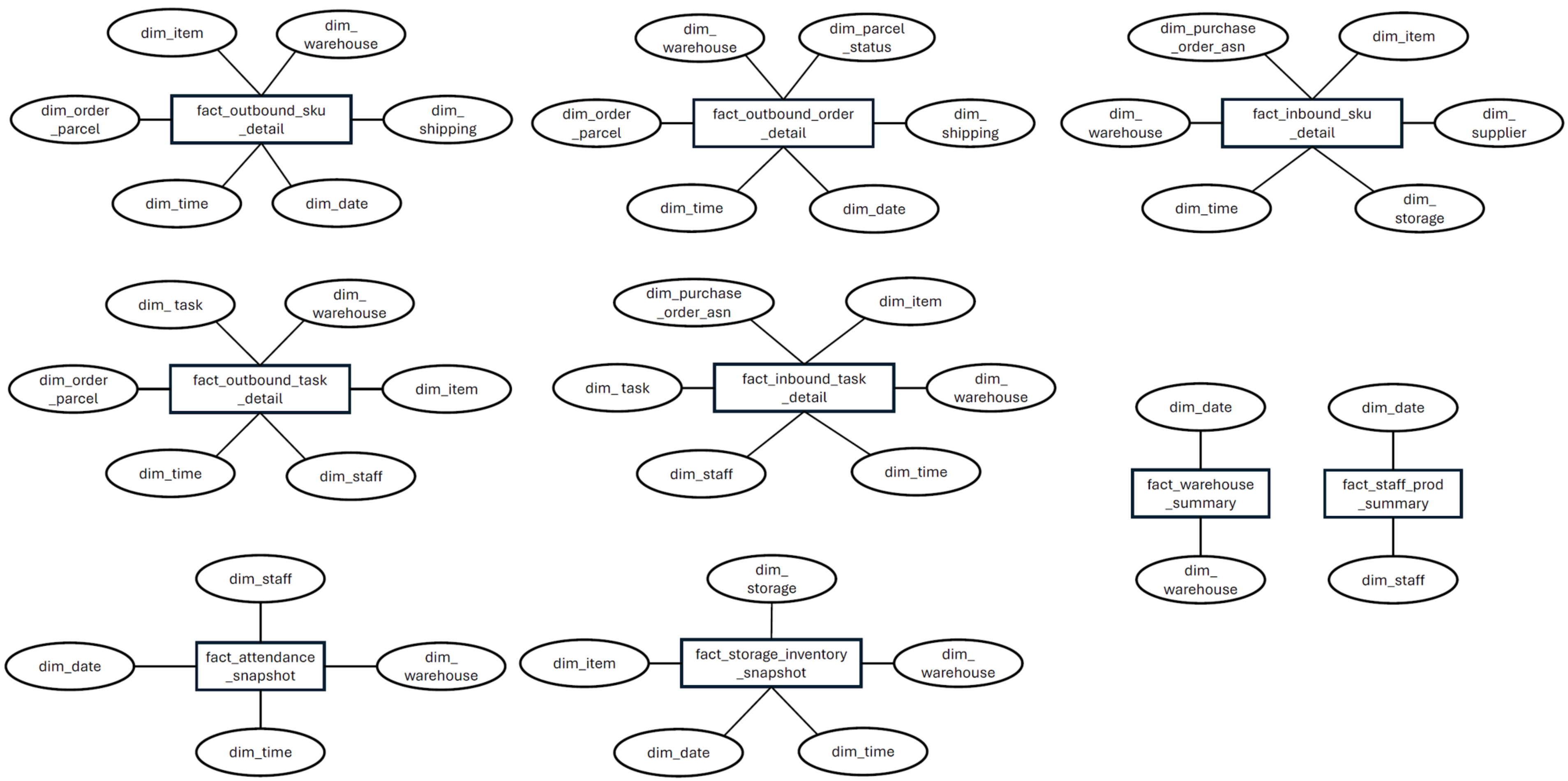


Figure 5. *Data Models for Data Warehouse Tables Also Included in Warehouse Data Mart*

f. 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.

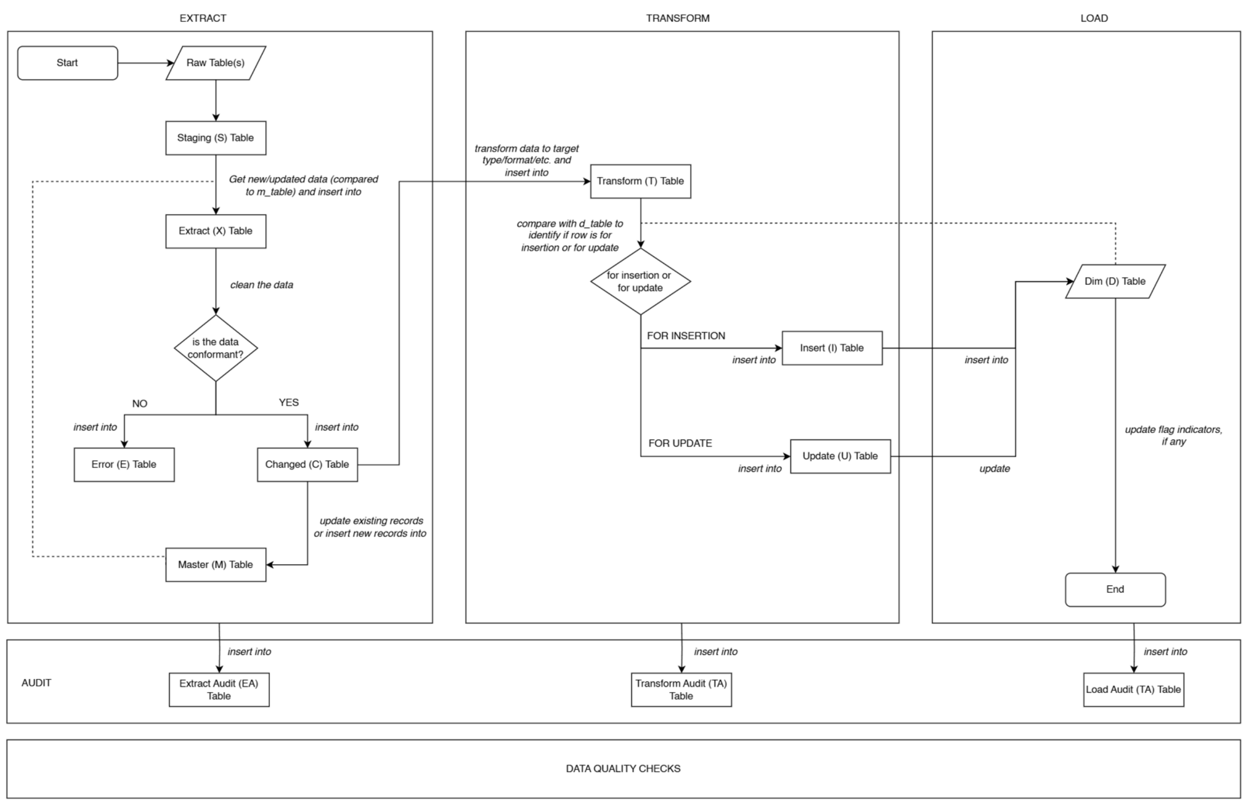


Figure 6. *ETL Process for Dim Tables*

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.

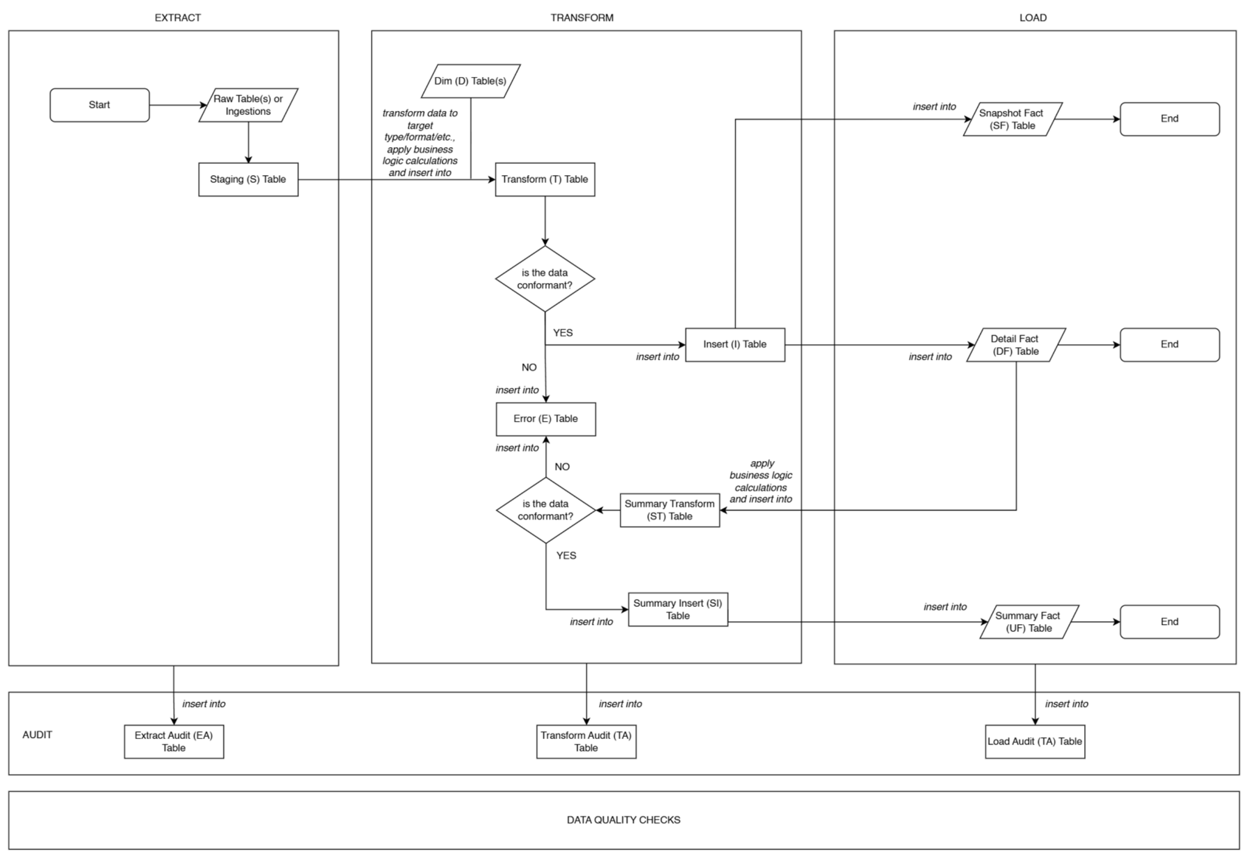


Figure 7. *ETL Process for Fact Tables (Detail, Summary, Snapshot)*

g. 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.

VI. Physical Design of Your Data Engineering Solution

a. 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

b. 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.

i. Importing libraries

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

ii. 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()
```

iii. 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()
```

iv. 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()
```

v. *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()
```

vi. 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()
```

vii. 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()
```

viii. 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()
```

ix. 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)
```

x. *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()
```

xi. *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()
```

xii. 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()
```

c. 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

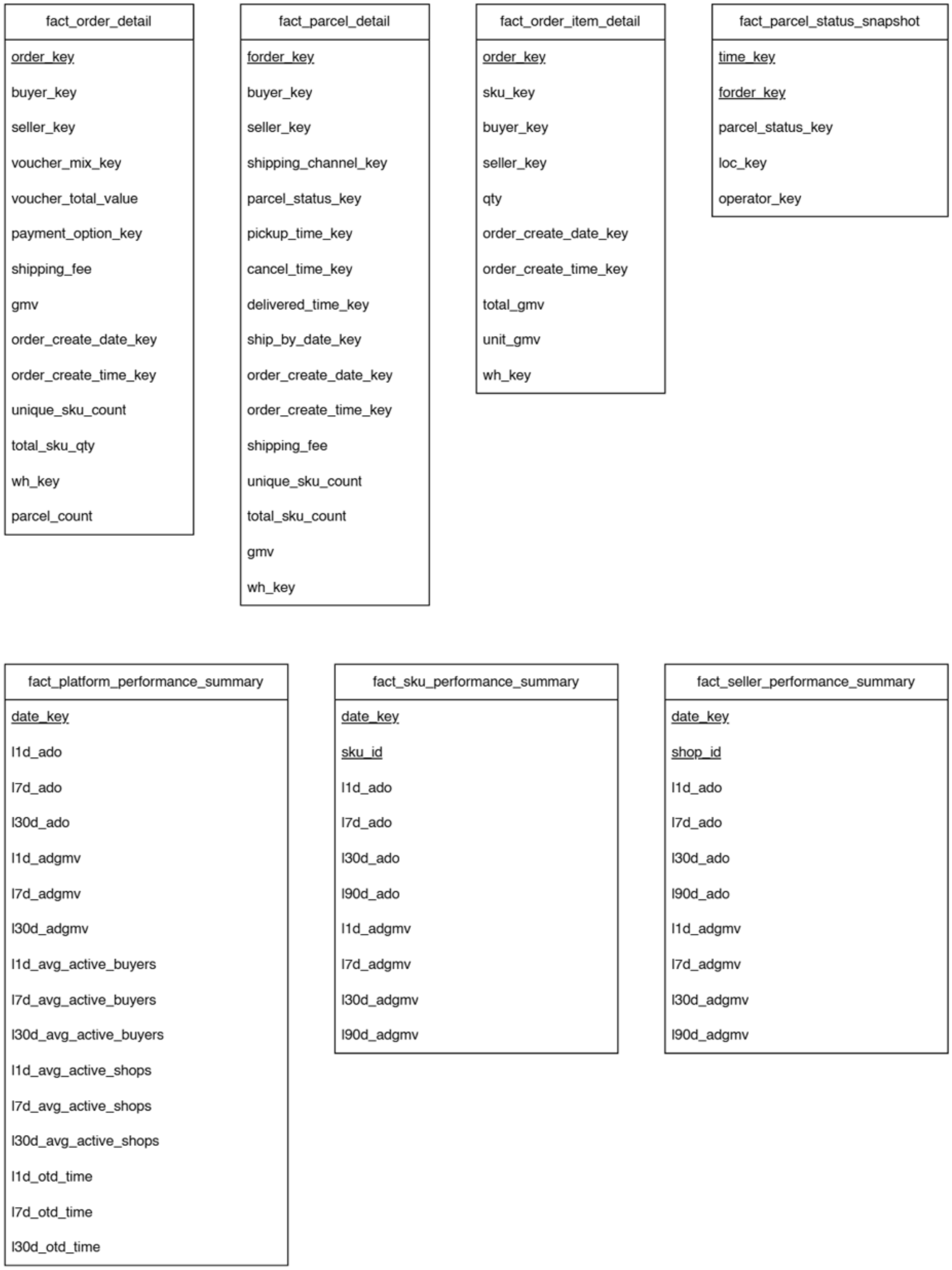


Figure 8. *Fact Tables and Their Attributes Under Marketplace Data Mart*

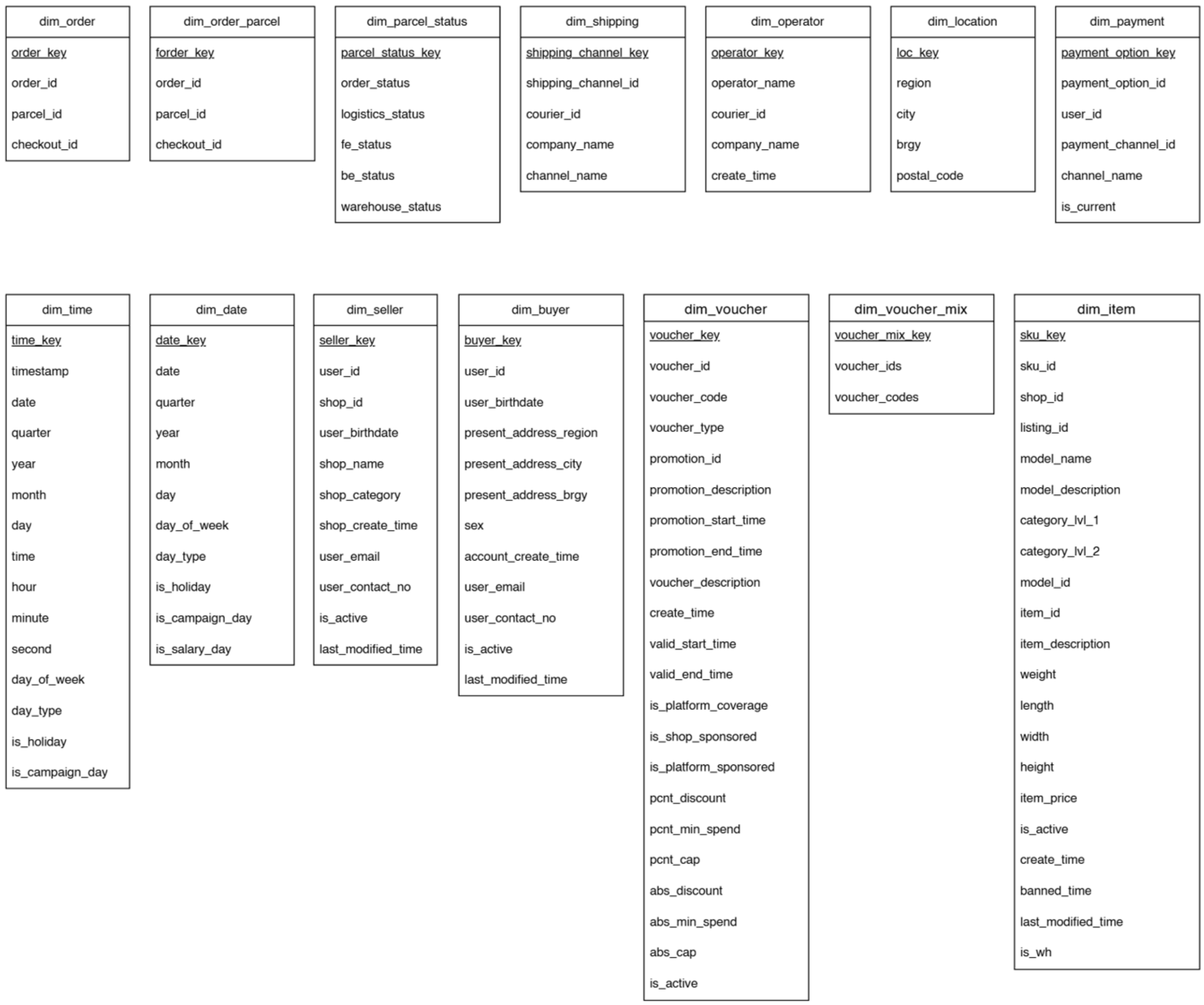


Figure 9. *Dim Tables and Their Attributes Under Marketplace Data Mart*

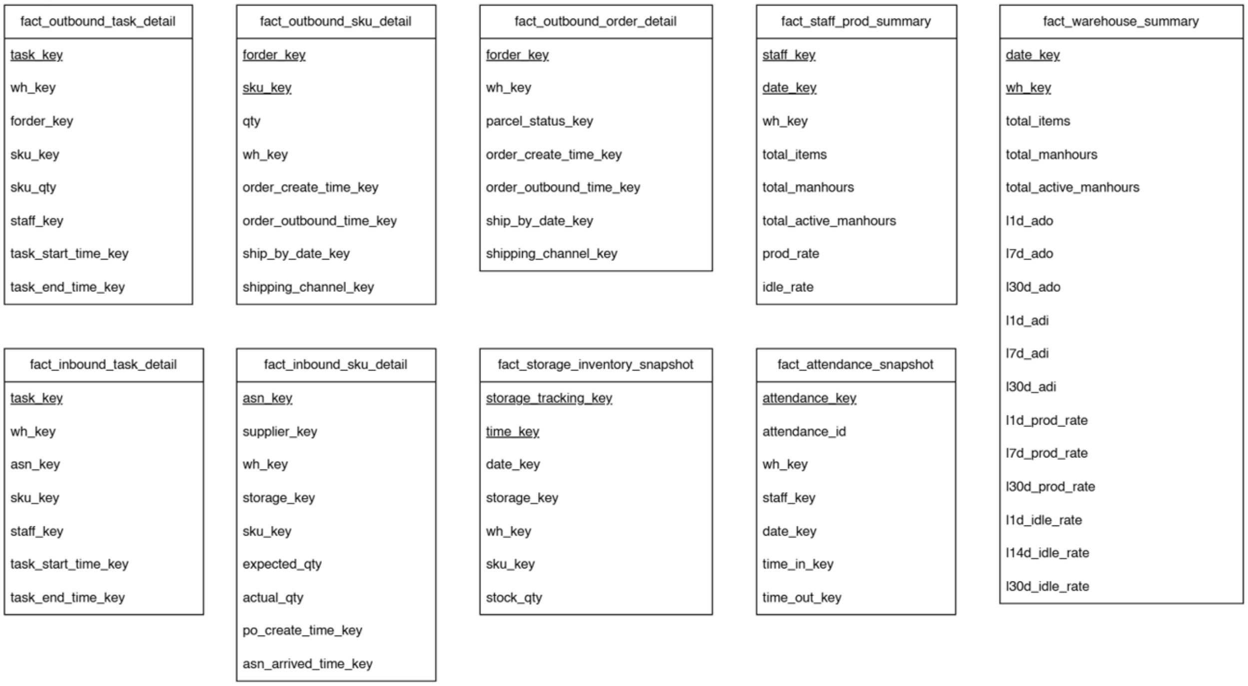


Figure 10. *Fact Tables and Their Attributes Under Warehouse Data Mart*

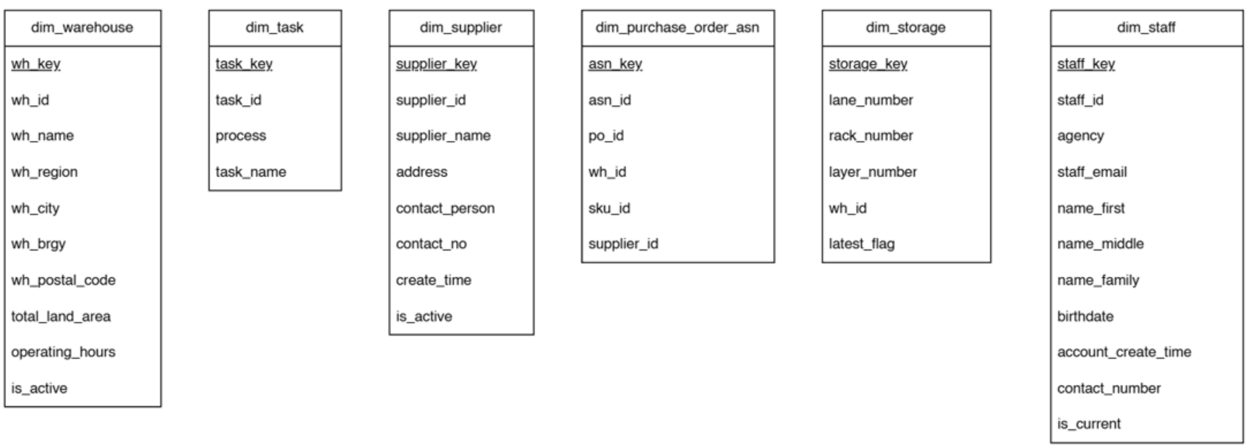


Figure 11. *Dim Tables and Their Attributes Under Warehouse Data Mart*

VII. 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?

```sql
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?

```sql
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?

```sql
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?

```sql
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.

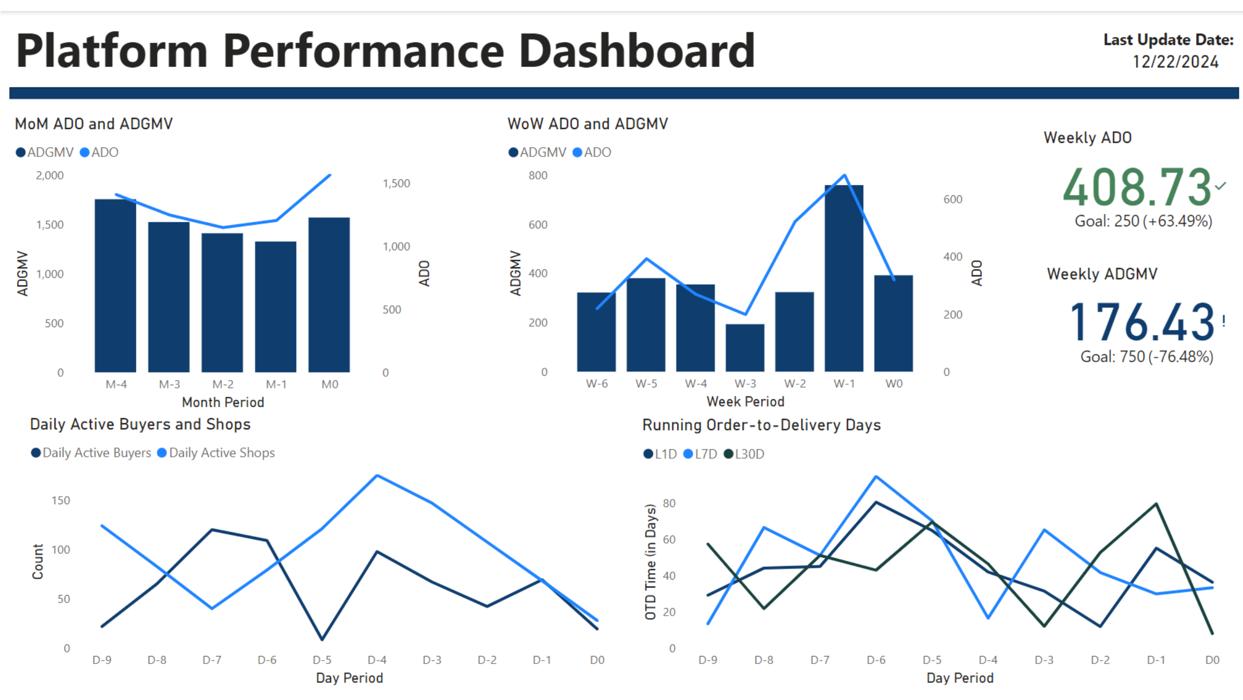


Figure 12. *Sample Power BI Dashboard for Platform Performance Reporting*

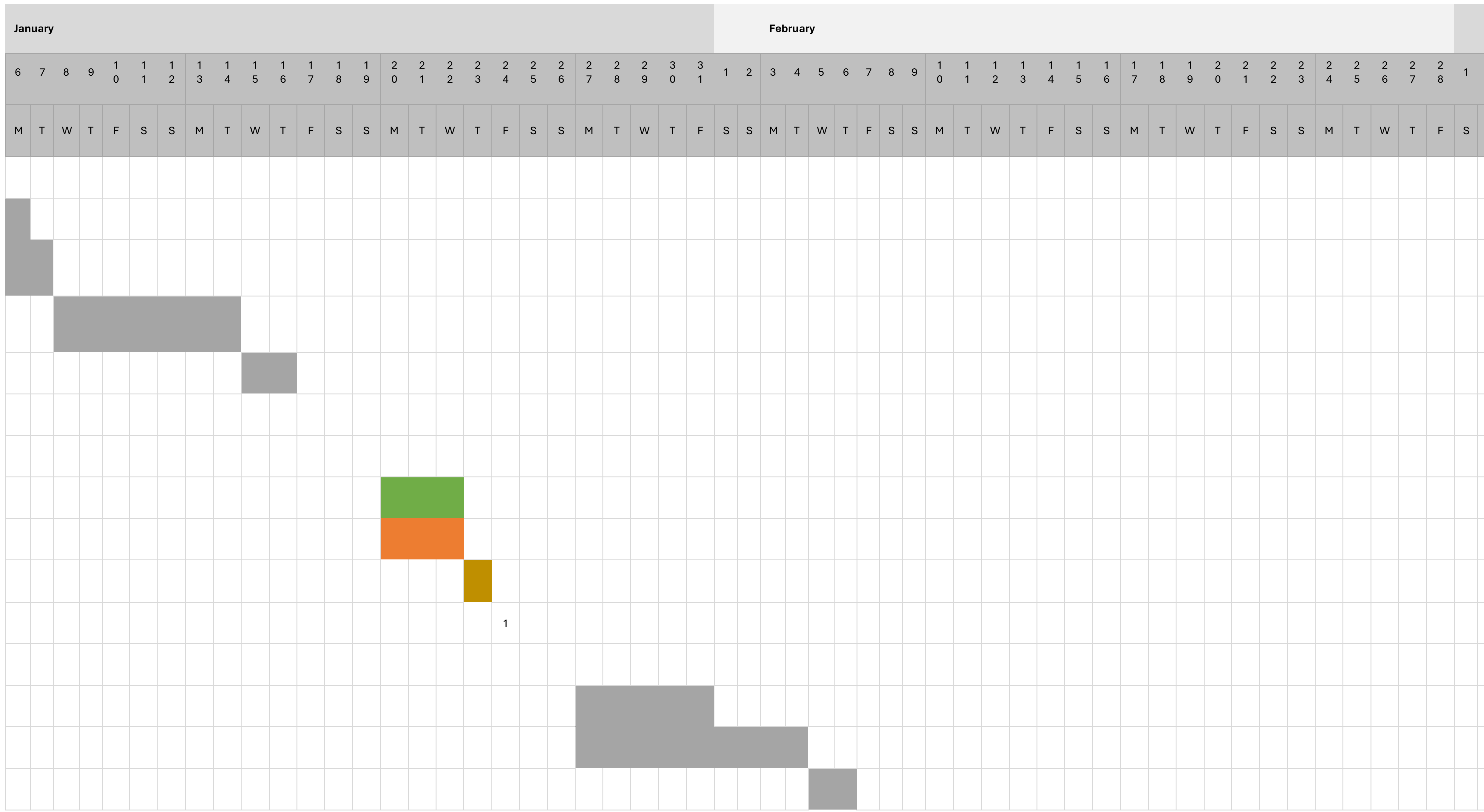
Table 8. *Gantt Chart for the EWIP Implementation*

Legend: On track | Low risk | Med risk | High risk | Unassigned

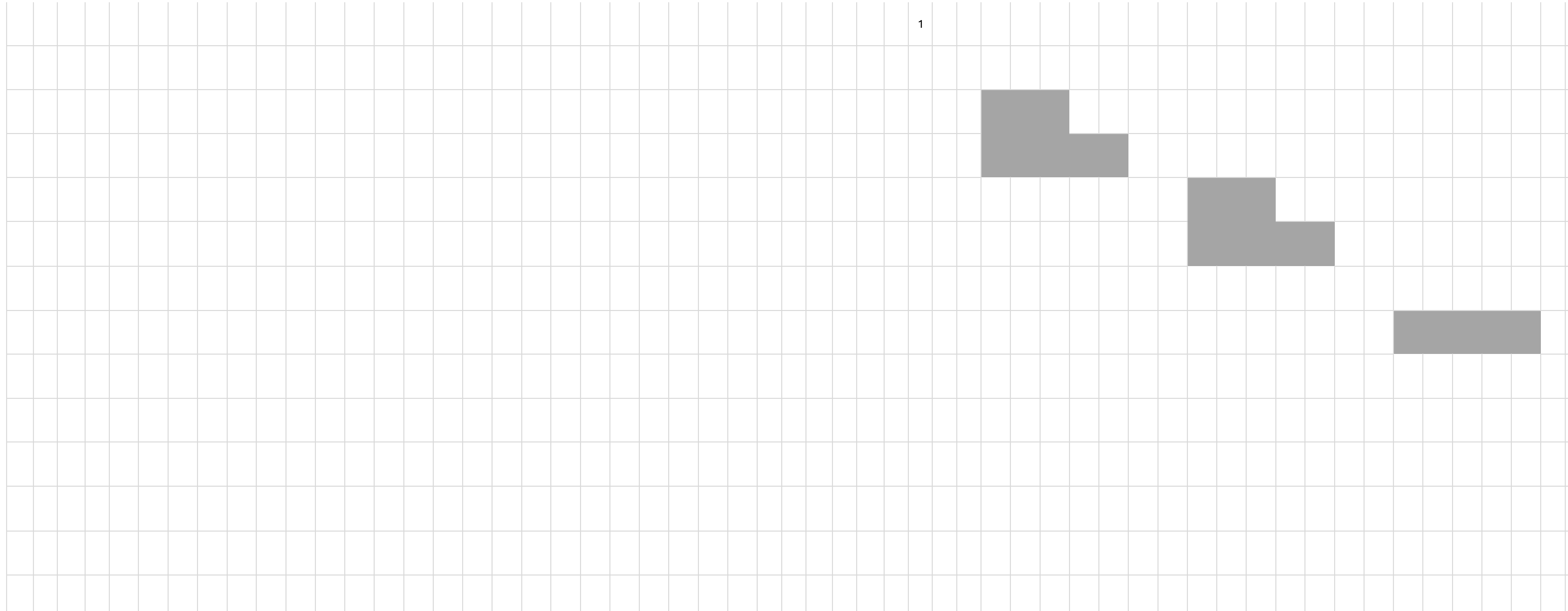
Project start date: 06/01/2025

Scrolling increment: 0

Milestone description	Category	Progress	Start	Days
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
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
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



VIII. 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.

IX. 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.

X. 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

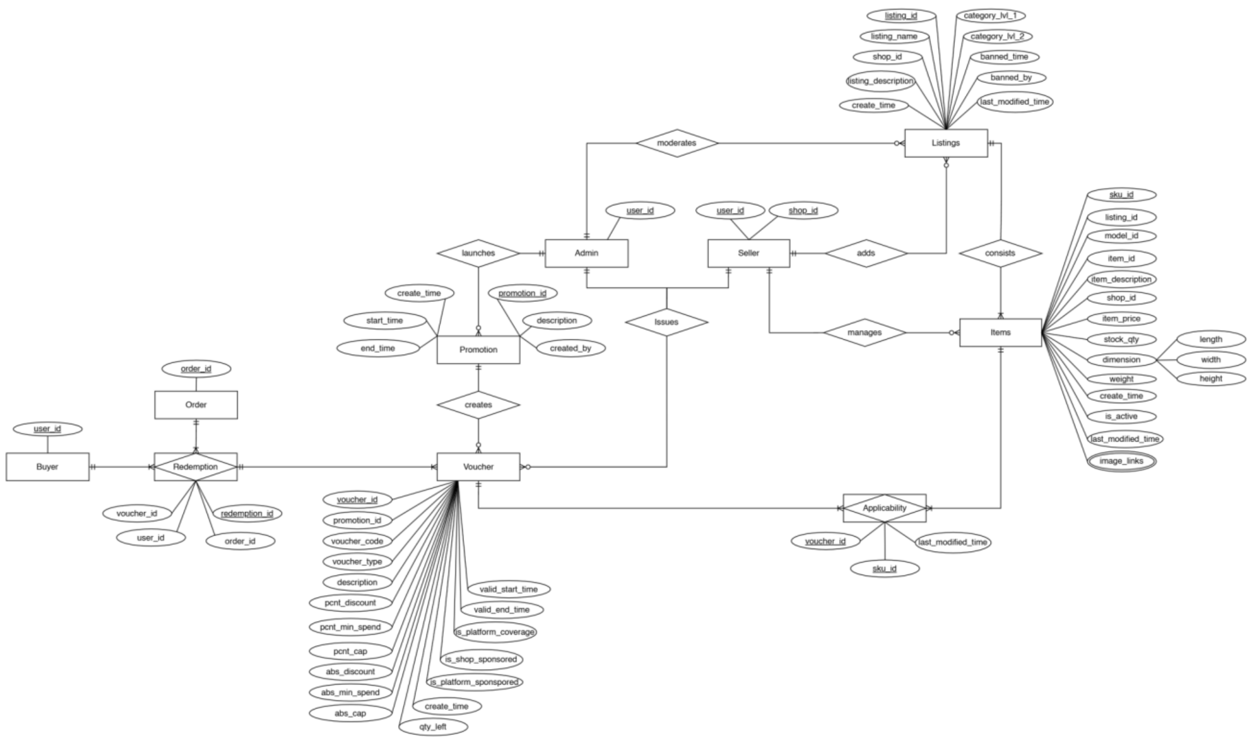
Fabri, L., & Valverde Márquez, I. (2020). Will e-commerce dominate physical store? *International Journal of Technology for Business*, *2*(1), 23–20. https://doi.org/10.5281/zenodo.3894442

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

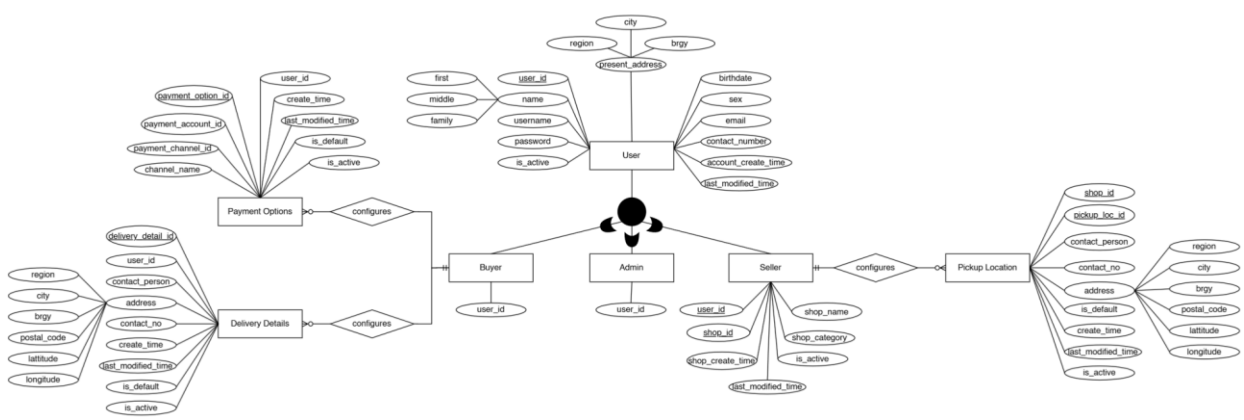
Schmitz, M. (2014). *High Performance Data Warehousing*. UCI Irvine Data Warehousing Notes.

XI. Appendices

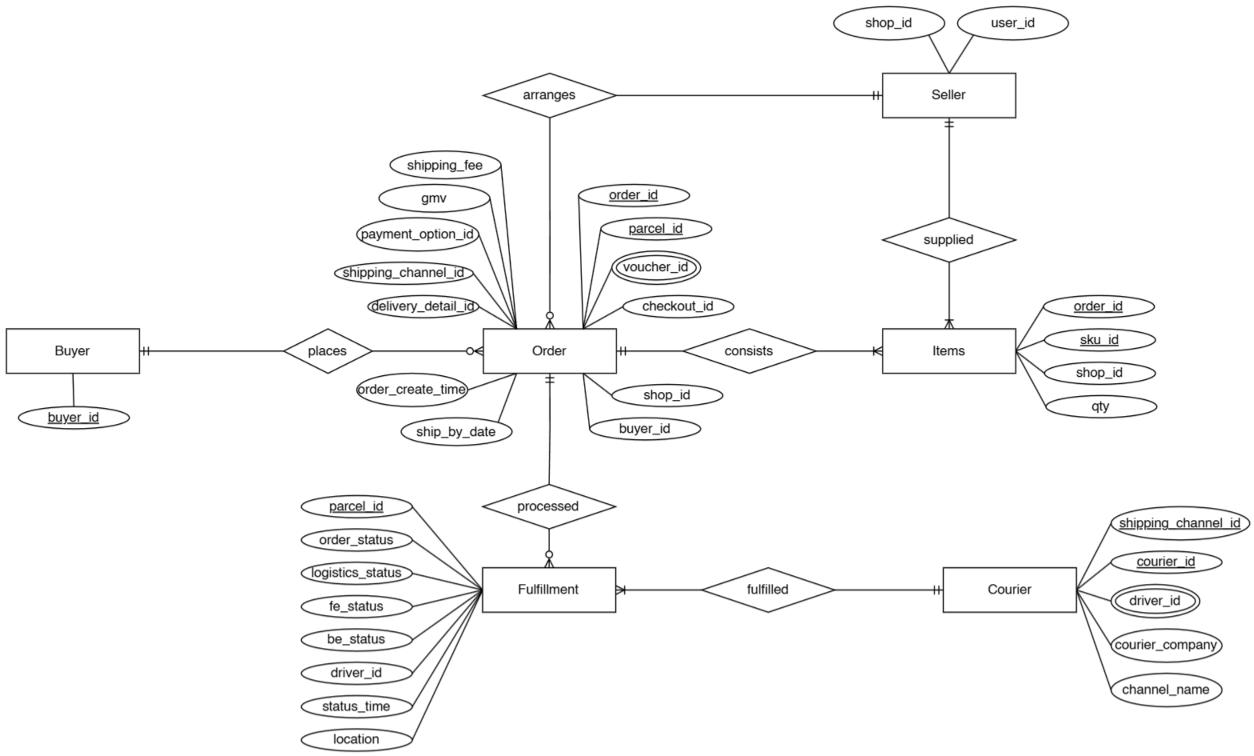
a. ERD for LPMS



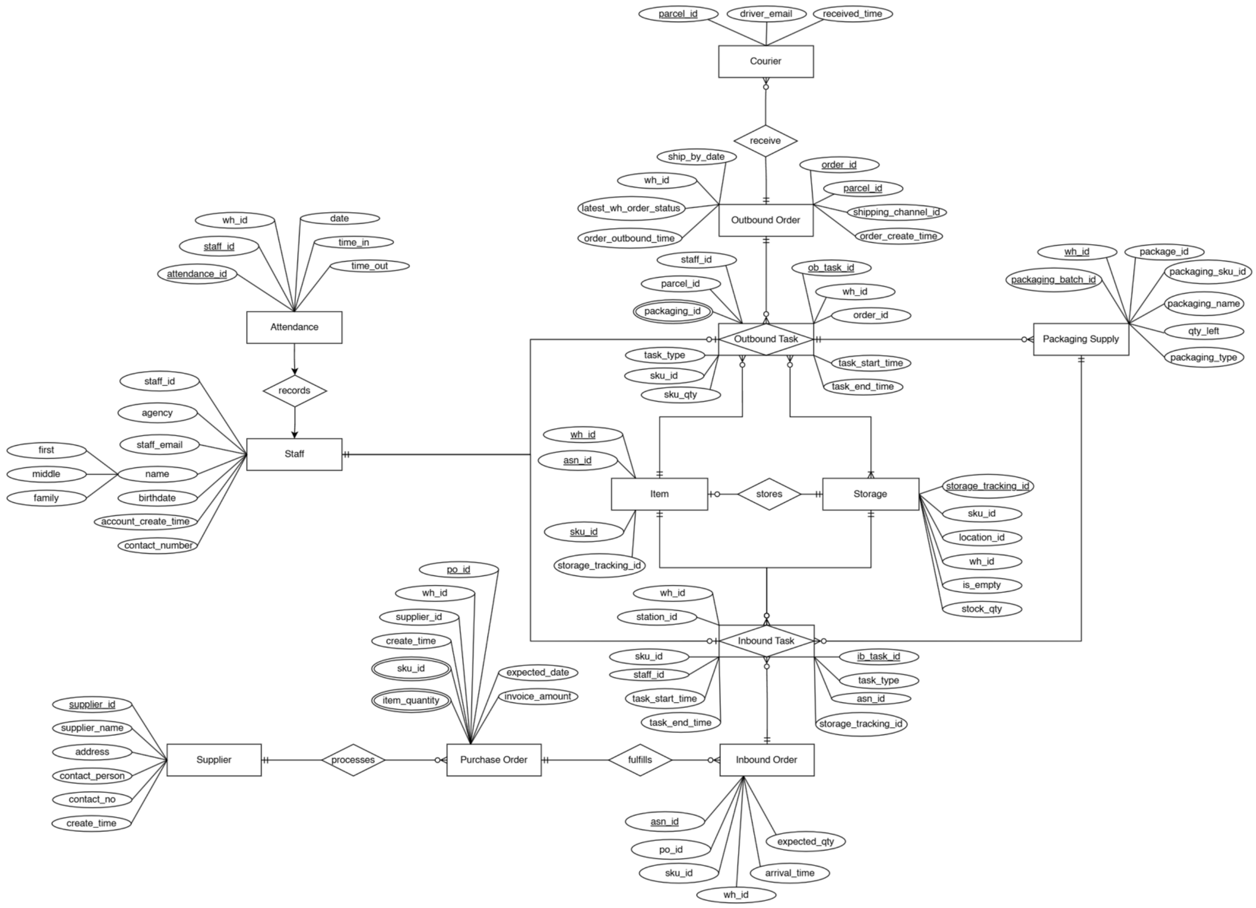
b. ERD for UMS



c. ERD for OMS

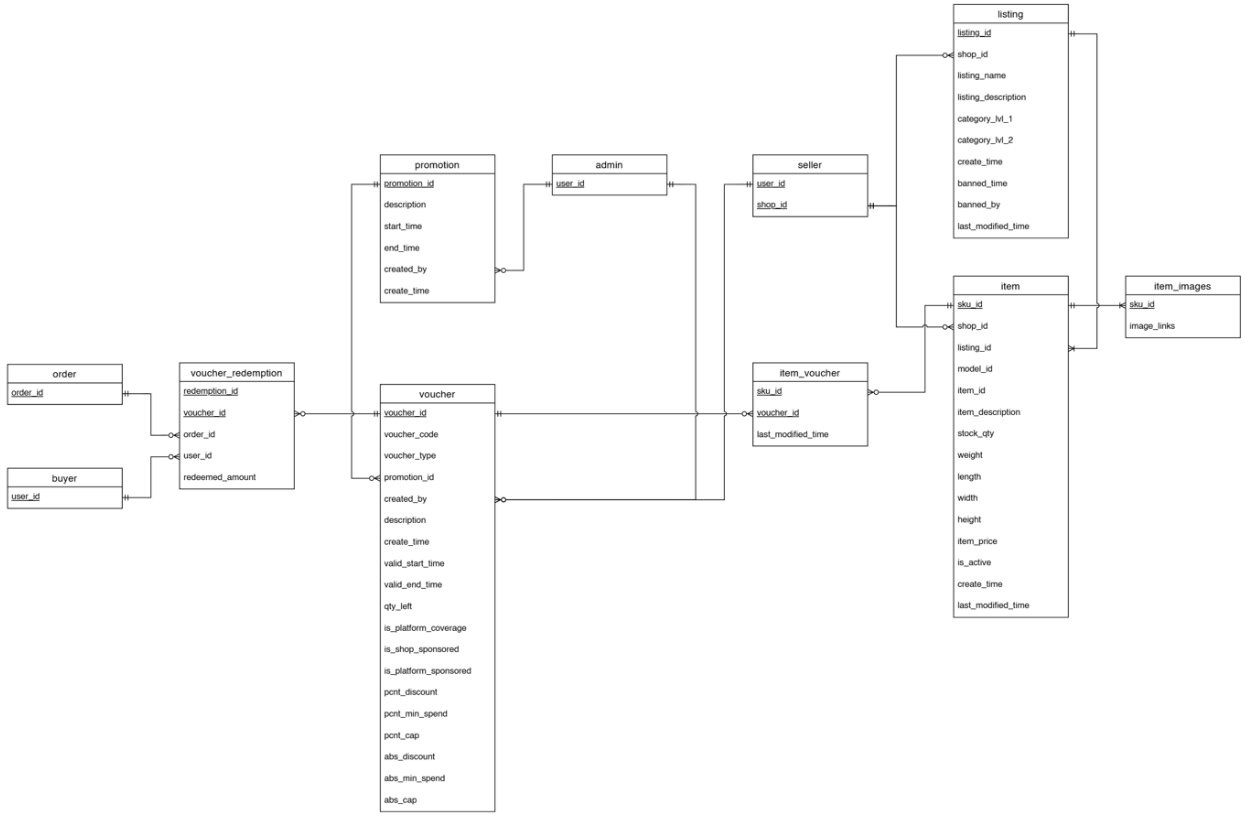

shop_id
user_id
Seller
arranges
shipping_fee
gmv
payment_option_id
shipping_channel_id
delivery_detail_id
order_id
parcel_id
voucher_id
checkout_id
supplied
order_id
sku_id
shop_id
qty
Buyer
places
Order
consists
Items
buyer_id
order_create_time
ship_by_date
shop_id
buyer_id
processed
parcel_id
order_status
logistics_status
fe_status
be_status
driver_id
status_time
location
Fulfillment
fulfilled
Courier
shipping_channel_id
courier_id
driver_id
courier_company
channel_name

d. ERD for WMS

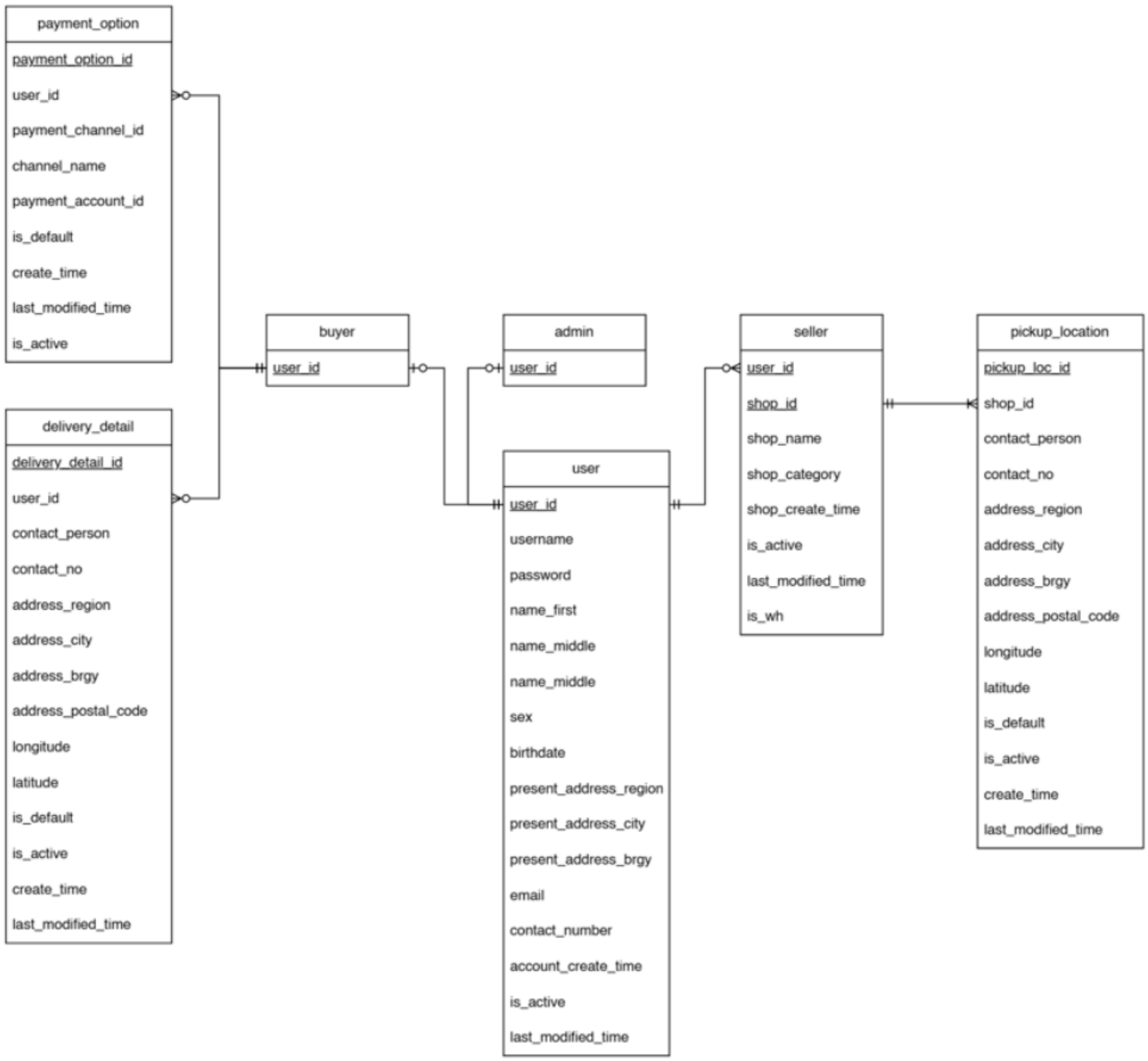


Courier
parcel_id
driver_email
received_time
receive
Outbound Order
ship_by_date
wh_id
latest_wh_order_status
order_outbound_time
order_id
parcel_id
shipping_channel_id
order_create_time
Outbound Task
staff_id
parcel_id
packaging_id
task_type
sku_id
sku_qty
ob_task_id
wh_id
order_id
task_start_time
task_end_time
Packaging Supply
wh_id
package_id
packaging_batch_id
packaging_sku_id
packaging_name
qty_left
packaging_type
Attendance
wh_id
staff_id
attendance_id
date
time_in
time_out
records
Staff
staff_id
agency
staff_email
name
first
middle
family
birthdate
account_create_time
contact_number
Item
wh_id
asn_id
sku_id
storage_tracking_id
stores
Storage
storage_tracking_id
sku_id
location_id
wh_id
is_empty
stock_qty
Inbound Task
wh_id
station_id
sku_id
staff_id
task_start_time
task_end_time
ib_task_id
task_type
asn_id
storage_tracking_id
Purchase Order
po_id
wh_id
supplier_id
create_time
sku_id
item_quantity
expected_date
invoice_amount
Supplier
supplier_id
supplier_name
address
contact_person
contact_no
create_time
processes
fulfills
Inbound Order
asn_id
po_id
sku_id
wh_id
expected_qty
arrival_time

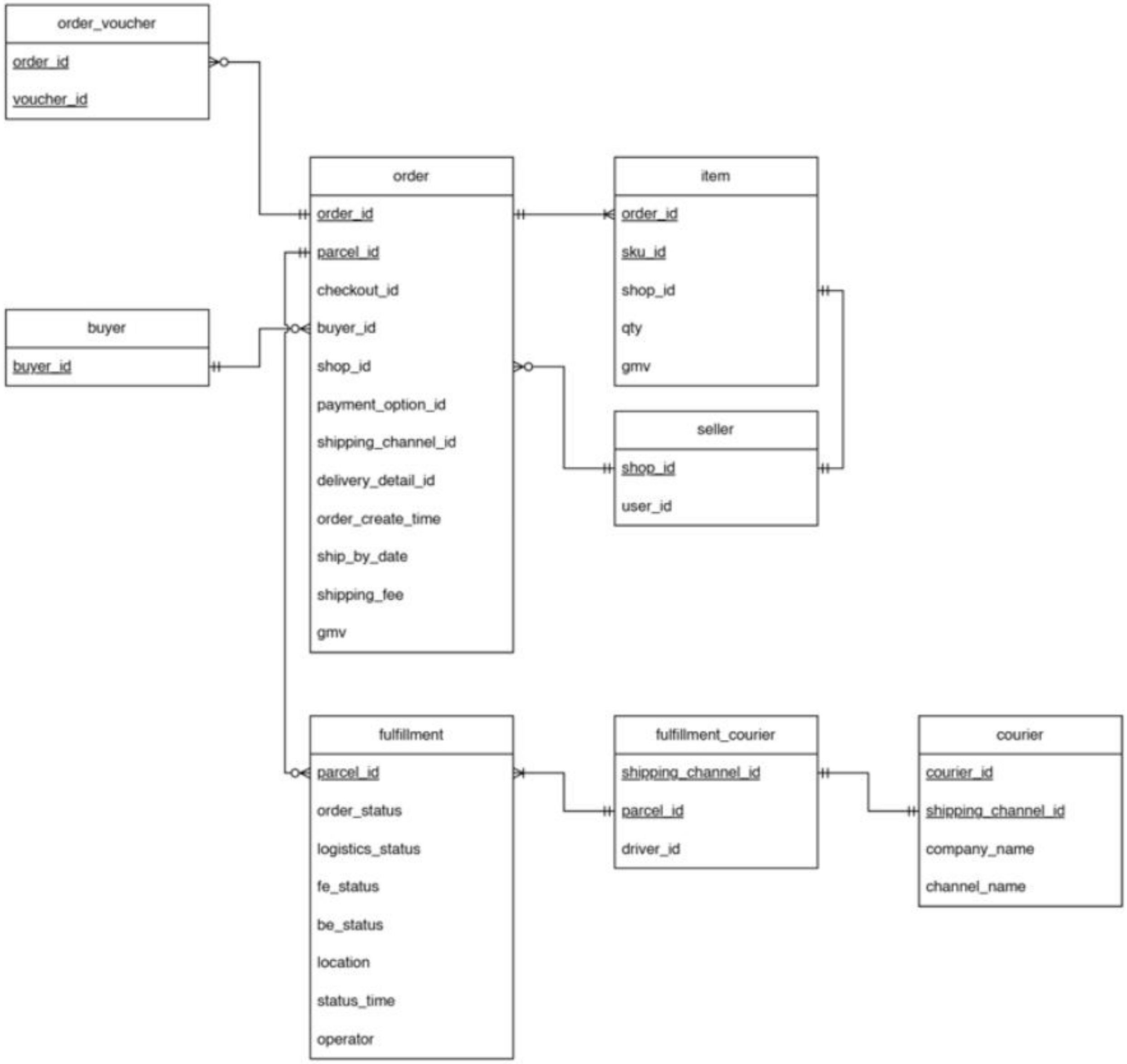
e. Relational Model for LPMS



f. Relational Model for UMS



g. Relational Model for OMS



h. Relational Model for WMS