UNIVERSITY OF SANTO TOMAS
College of Information and Computing Sciences
Department of Computer Science
1st Semester of 2025-2026

ShopZada 2.0 Enterprise Data Warehouse: Technical Documentation

3CSC - Lakers Showtime
December 18, 2025

Members:

Baltazar, Jorge Kollin

Lee, Ko Han

Olmedo, Alden Alexander

Pilapil, Gbert

Santa Maria, Gerick Michael

So, Kyan Charles

Udal, Joshua Uriel

Yalung, Julian Edison

Table of Contents

1. Executive Overview

ShopZada is a rapidly growing e-commerce platform that has expanded globally, now handling over half a million orders and two million line items from diverse product categories. Despite this growth, their data remains fragmented across multiple departments namely Business, Customer Management, Enterprise, Marketing, and Operations.

The team have been brought in as ShopZada's Data Engineering Consultants to design, implement, and operationalize a complete Data Warehouse solution that integrates these datasets, delivers analytical insights, and supports data-driven decision-making. The team is also tasked with designing the full end-to-end Data Warehousing pipeline — from conceptual modeling to deployment and reporting — and demonstrate how your solution empowers ShopZada to generate business value.

The purpose of this technical documentation is to outline the business questions to be solved, methodologies, processes assumptions, the development tools, files, and the business intelligence layers all related to ShopZada's business needs. Moreover, the main goal of this project is to construct a reliable data warehouse that can support the growing needs of ShopZada and transform their data from scattered and disorganized, into a reliable, consistent, functional, and scalable data warehouse ready for easy viewing and analytics and to support their core business process: customer transactions, order informations, campaign performances, merchant performance, and staff performance.

2. Core Business Questions to Address

To address Shopzada's data needs: these business questions are listed to be addressed:

EXECUTIVE DASHBOARD

1. What is ShopZada's total revenue and order volume?
2. How many total delayed orders were there?
3. What is the average order delay (in days)?
4. What is the revenue trend per month and/or year?
5. What is the distribution of revenue per user type?
6. What are the top selling products by categories?

CAMPAIGN

7. Which campaign generates the most order volume/revenue?
8. What are the top availed campaigns?
9. What is the revenue per campaign and user type?
10. What campaigns are the most successful during the first month of creation?
11. What are the top performing campaigns based on uptake rate, revenue, and order volume?

ORDERS

12. What is the percentage of orders done during the weekend?
13. What are the top 10 and bottom 10 products with the most quantity available?
14. Which state produces the most orders?
15. Who are the top 3 users with the most orders?
16. What are the top 3 product categories that generate the most revenue?

STAFF & MERCHANTS

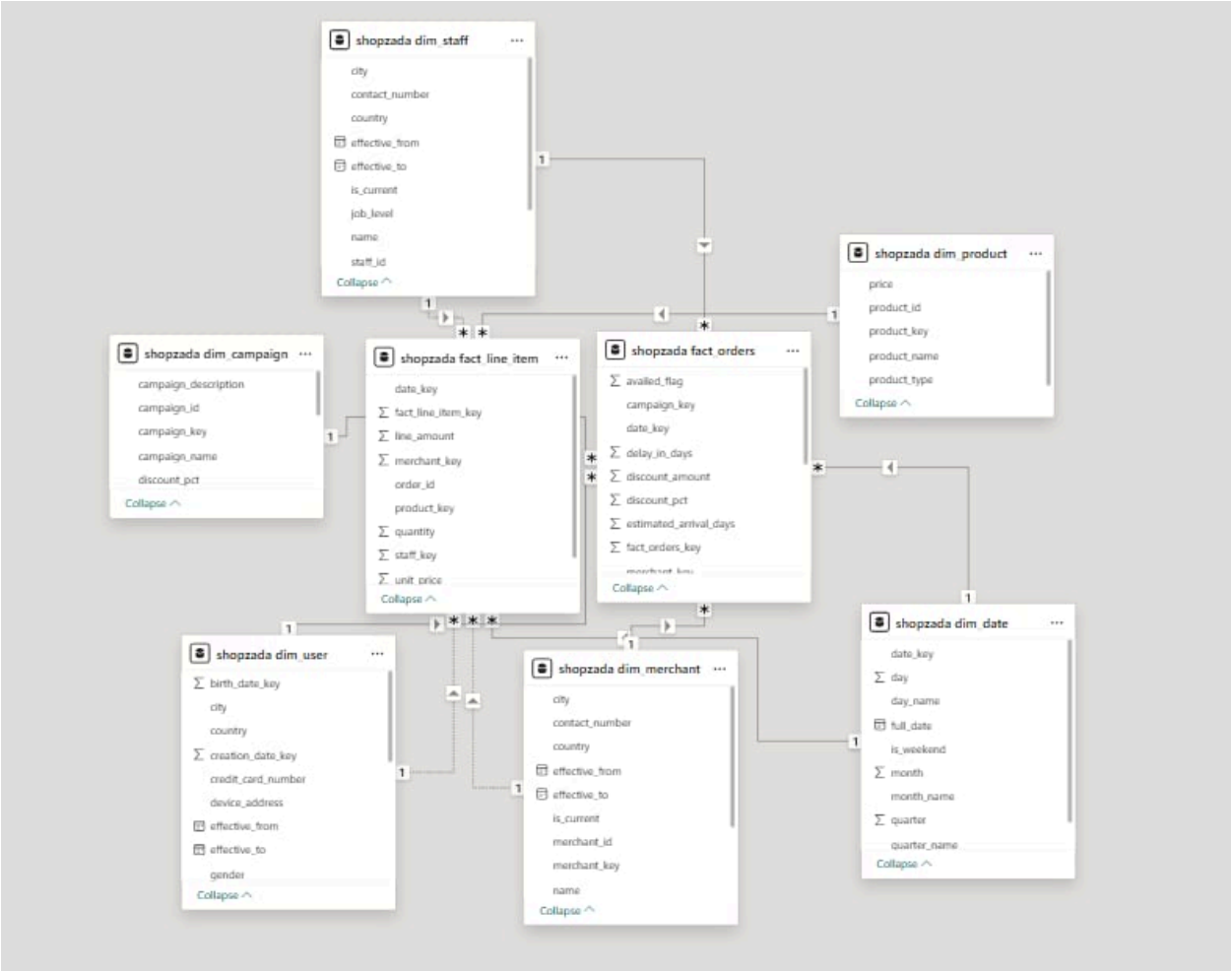
17. Who are the top performing merchants via revenue and order volumes?
18. Who are the top performing staff via job level?
19. Who are the top merchants with the most revenue per state?

USERS

20. Who is the user who spends the most?
21. Which bank issues to the most users?
22. What is the number of active and inactive users?
23. What states and cities have the most users for Shopzada?
24. Which is the number of users per state?
25. What is the average transaction value per the user's job title and each of their order volume as well?

3. Architecture Overview

The data warehouse applies a Kimball methodology with the following architecture displayed below via PowerBI:



Power BI Star Schema

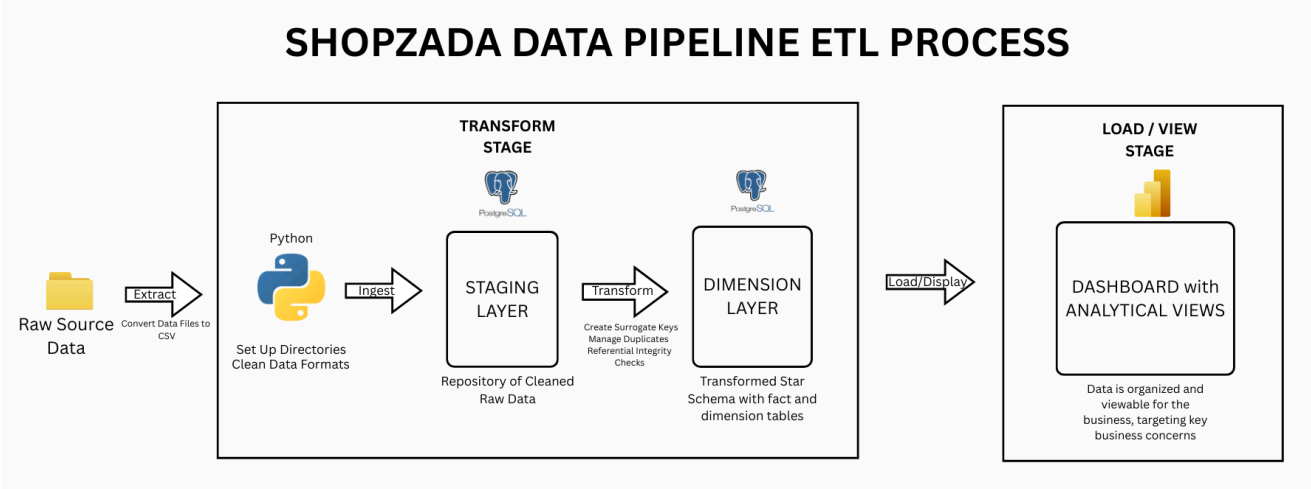
Kimball architecture was applied because it is simpler and faster to set up, provides ShopZada with the needed analytics based on its large amount of data, is easy to scale for future their future needs, and is intended to make it convenient for ShopZada employees to view, query, understand, and use their data in an organized manner.

Moreover, the Kimball architecture presented consists of five dimensions based on the given raw datasets and the team's data exploration, cleaning, and transformation results. Furthermore, the team also added a date dimension, as it will ensure consistent time and date logic across the entire data warehouse.

Additionally, three fact tables are present in the Kimball architecture presented: "FactLineItem" and "FactOrders" Firstly, the fact line item represents a transaction, connecting

all dimensions into a single entity. Since transactions are the primary driving process in ShopZada, this ensures that all transactions are detailed and can be used for analytics, prediction, and any other necessary information. Lastly, the FactOrders table simply stores the specific order details of each transaction and the entities connected to it.

4. Brief Methodology



ShopZada Data Pipeline ETL Process

The data pipeline process above displays a simple overview of the team's ETL process methodology for the ShopZada data warehouse. Specifically, the ShopZada data warehouse applies a hybrid ETL pipeline that consists of three main stages: Python-based cleaning, PostgreSQL staging and dimensional modeling through Airflow orchestration, and a dashboard layer that queries analytical views through Power BI.

The pipeline will begin with the raw data collected from various departments which will be standardized into CSV files. Python will then be used to extract, explore, and clean these files by setting up directory structures, enforcing consistent data types and formats which will prepare them for loading.

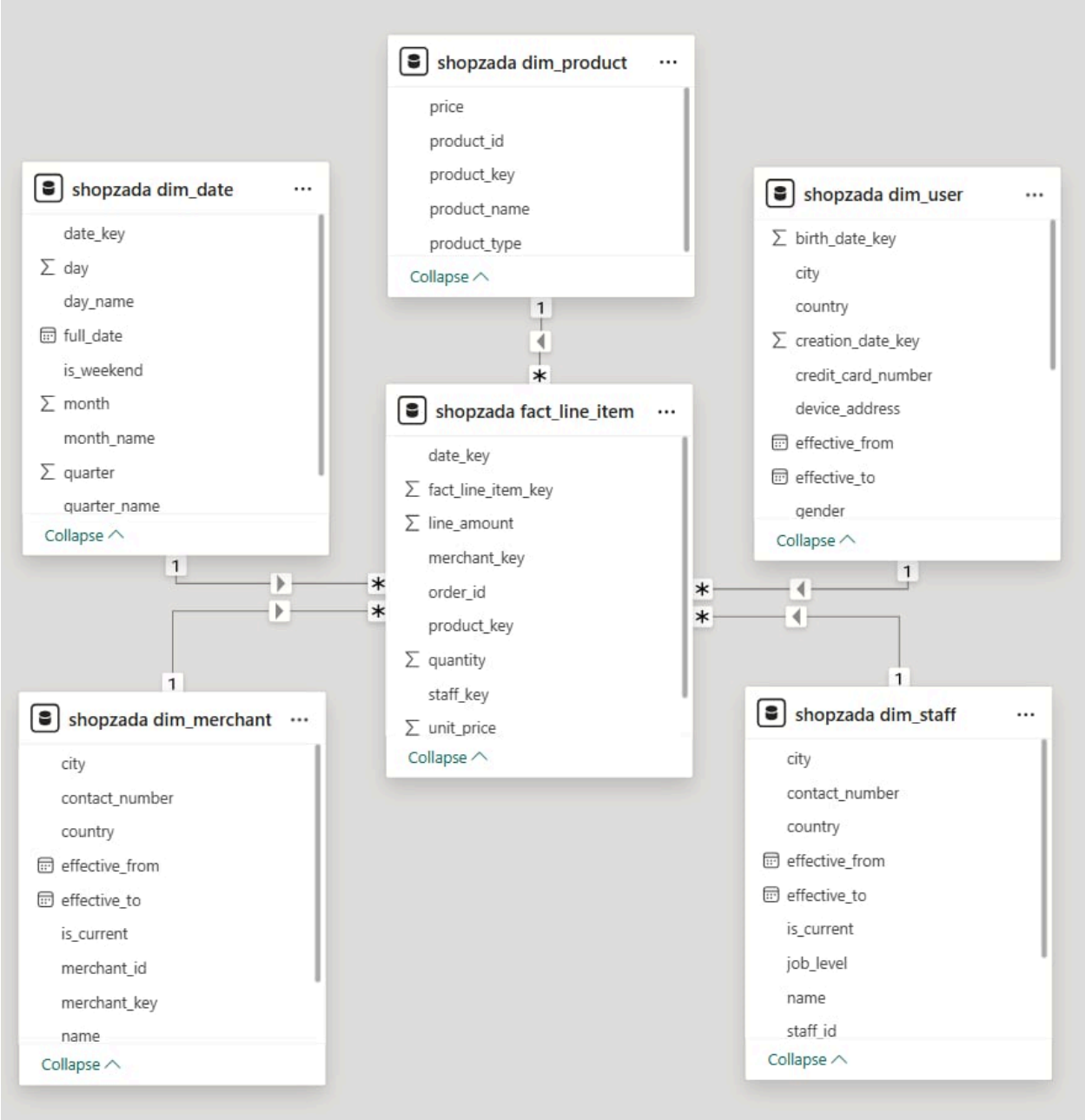
After this stage of the pipeline, orchestration will begin via Airflow because it allows these workflows to be triggered and monitored. This will ensure reliable data pipelines and processes. It will be responsible for loading and cleaning the data into the staging layer in PostgreSQL. The cleaning process will include standardizing inconsistent formats, finalizing data types, joining separate tables, handling duplicates, and completing other necessary transformations to gain and ensure a smooth loading process into the Postgres database and Kimball data architecture. This will provide a repository of cleaned but source-oriented tables.

Once the team finishes cleaning the data, they then stage it in Postgres. Staging is performed before loading to ensure the cleaned data is processed and transformed properly while also providing a buffer between the cleaning and loading phases. Furthermore, the team

creates the required tables in advance to again ensure smooth loading into the Postgres database. Afterwards, the team then loads all staged data into PostgreSQL (Dimension Layer), specifically the shopzada schema in the data warehouse, following the simplified Kimball

architecture shown below:

Star Schema

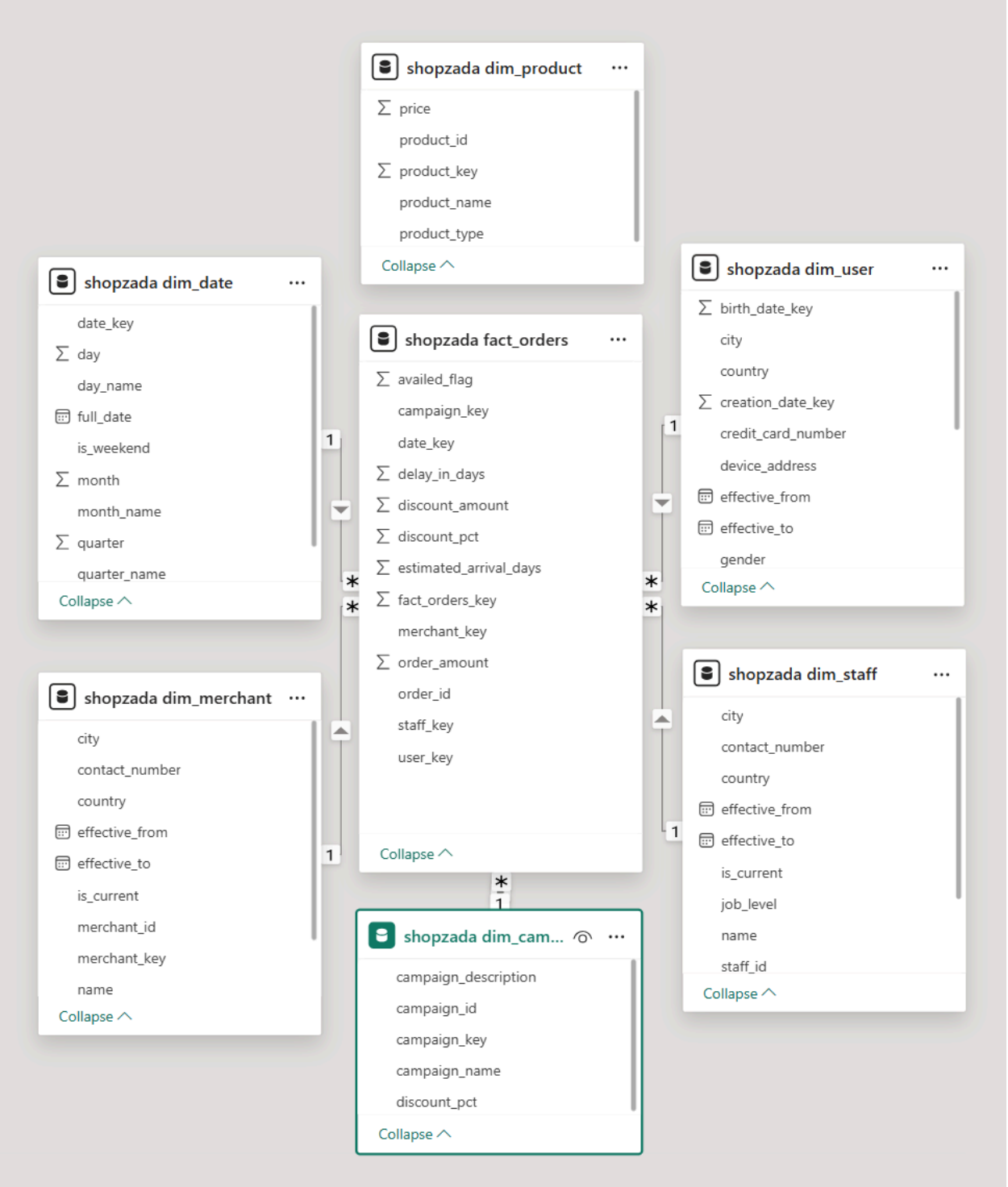


Star Schema with fact_line_item at the center

The grain for each schema of this Star Schema is as follows:

- dim_user
    - One row per user version per effective time period.

- dim_staff
    - One row per staff member per effective time period.
- dim_merchant
    - One row per merchant per effective time period
- dim_date
    - One row per calendar day
- dim_product
    - One row per unique product.
- fact_line_item
    - One row per product line item within a single order.



Star Schema with fact_orders at the center

The grain for each schema of this Star Schema is as follows:

- Dim_user
    - One row per user version per effective time period.
- dim_staff

    - One row per staff member per effective time period.
- dim_merchant
    - One row per merchant per effective time period
- dim_date
    - One row per calendar day
- fact_orders
    - One row per complete order placed

After successfully loading the data into the PostgreSQL database and shopzada schema, the team transforms the dimension and fact tables into analytics views. Finally, they expose these analytics views to the business intelligence layer (PowerBI dashboard), where ShopZada employees can view and analyze the data.

Additionally, the team utilized Docker, Power BI, Metabase, and GitHub. Docker was used to containerize the entire data warehouse and development files, bundling all dependencies and configurations into a single container. This ensured consistent behavior across development machines, simplified environment setup, and made the system more reliable and production-ready by eliminating "works on my machine" issues.

Next, Power BI was utilized as the analytical layer of the analytics views to accurately address Shopzada's business needs and provide a clear overview across the entire Shopzada data warehouse. The team also utilized Metabase for quick prototyping of the business intelligence dashboards. And finally, the team utilized GitHub throughout the entire development process of the data warehouse, ensuring proper version control and facilitating enhanced development collaboration.

5. Assumptions and Evidences

Listed below are the assumptions the group made in constructing the data warehouse for ShopZada:

1. All countries are USA instead of random ones (e.g. Iraq, China).
    - **Evidence:** Since all cities are located in a state, and all states are based on the United States, the group decided to standardize all countries to “United States” while keeping the current city and street.
    -

2. Duplicated users may entail that a previous user with a user ID may have deleted their account, letting a new user be given his/her ID.
    - **Evidence:** The user_id column in the customer data contained duplicates wherein the same name and address appeared with different account creation dates. The group assumed that these are re-registered accounts rather than actual duplicates.

3. If a certain transaction date does not match with a user’s creation date. It might mean that the system was only updated recently and data quality issues exist.
    - **Evidence:** From data of the operations department, numerous orders contained transaction dates that are earlier than the recorded creation date of its respective user which should not be the case. Therefore, the team interpreted this issue of such mismatches as effects of a delayed system update rather than intentional.

4. Most analytics regarding user information have been based on active users. Inactive users or users with deleted accounts are not counted in our analytics for certain material views (besides revenue analytics and certain product analytics)
    - **Evidence:** Shopzada’s management primarily interests itself in understanding the current active customer base. Therefore, our analytics did not include users who are inactive or deleted.

5. User, Staff, and Merchant creation dates are the base dates of each
    - **Evidence:** Source files, user_data.csv, staff_data.csv, and merchant_data.csv, the only time related fields for these are the creation_date. Because of this, we treat creation_date as the date when each user, staff member, or merchant first becomes active in shopzada, and we use that as their starting date when linking them to fact_orders through user_key, staff_key, merchant_key, and date_key. Any orders with transaction dates not matching the respective user’s creation date will be changed to having no user (null users have a 0 user id).

6. Every user has a credit card
    - According to the credit card data from the Customer Management Department, every user has a credit card linked to a specific bank. The credit_card_number

column has no null values, and the users listed in user_data.csv matches the users in user_credit_card.csv.

7. ShopZada is similar to Grab, combining a Mart feature and also allows for ordering of food.
    - **Evidence:** The data from the business department shows that the product_list.csv contains product_type with categories for ready made breakfast, lunch, and dinner. Furthermore, it has other product categories that are not ready-to-eat food, such as groceries, as well as products like furniture and accessories.

6. Data Extraction and Cleaning Process

In this section, the team's detailed steps in extracting and cleaning the data will be described considering these steps are the foundations of the whole data warehouse architecture.

6.1 Table of Converted Raw CSV Files:

The table below displays all of the raw data converted to CSV format using pandas. CSV is easy to read from VSCode and integrates well with Airflow, Python, and database bulk-loading techniques, making it ideal for dependable, repeatable data input in a Kimball-style data pipeline.

Departments				
Business	**Customer Management**	**Enterprise Department**	**Marketing Department**	**Operations Department**
product_list.csv	user_credit_card.csv	merchant_data.csv	campaign_data.csv	line_item_data_prices1.csv
	user_data.csv	order_with_merchant_data1.csv	transactional_campaign_data.csv	line_item_data_prices2.csv
	user_job.csv	order_with_merchant_data2.csv		line_item_data_prices3.csv
		order_with_merchant_data3.csv		line_item_data_products1.csv
		staff_data.csv		line_item_data_products2.csv
				line_item_data_products3.csv

				order_data_20200101-20200701.csv
				order_data_20200701-20211001.csv
				order_data_20211001-20220101.csv
				order_data_20220101-20221201.csv
				order_data_20221201-20230601.csv
				order_data_20230601-20240101.csv
				order_delays.csv

6.2 Cleaning Details Per Dimension

Moving on to the cleaning layer, detailed steps will be listed below per dimension:

Campaign Dimension Cleaning Details	
Source Files	campaign_data.csv
Cleaning Steps	1. Parsed and reconstructed columns 2. Removed excessive quotation marks 3. Normalized discount values into integer percentages 4. Duplicate campaign IDs resolved using Type 1 overwrite semantics 5. Invalid records (if any) written to campaign_data_issues.csv
Output file	campaign_data.csv campaign_data.parquet campaign_data_issues.csv

Merchant Dimension Cleaning Details	
Source Files	merchant_data.csv
Cleaning Steps	1. Removed auto-generated index column. 2. Ensure no unnecessary quotation marks in the merchant name. 3. Parsed creation date. 4. Normalized country column where it only contains “United States”. 5. Normalized contact number format for consistency.
Output file	merchant_data.csv merchant_data.parquet merchant_data_issues.csv

Product Dimension Cleaning Details	
Source Files	product_list.csv
Cleaning Steps	1. Removed auto-generated index columns 2. Standardized product categories (e.g., toolss → tools, cosmetic → cosmetics) 3. Filled missing product categories deterministically (e.g., “bottle of paint” → stationary and school supplies) 4. Validated numeric prices 5. Flagged unresolved issues into an issues file
Output file	product_list_clean.csv, product_list_clean.parquet product_list_issues.csv

Staff Dimension Cleaning Details	
Source Files	staff_data.csv
Cleaning Steps	1. Removed auto-generated index columns. 2. Normalized contact number column to follow a consistent format. 3. Normalized country column where it only contains “United States”. 4. Normalized job levels for consistent spacing and capitalization. 5. Invalid or incomplete records will be written to staff_data_issues.csv.
Output file	staff_data.csv staff_data.parquet staff_data_issues.csv

User Dimension Cleaning Details	
Source Files	user_data.csv, user_job.csv, user_credit_card.csv
Cleaning Steps	1. Merged the three files row-by-row based on their aligned structure 2. Removed technical artifacts (e.g., Unnamed: 0,fixed duplicate ID columns) 3. Standardized data types (dates, strings) 4. Normalized business rules (e.g., student job handling)

	5. Applied validation checks: a. name consistency across tables b. missing or invalid values 6. Invalid or inconsistent rows are written to *_issues.csv
Output file	user_data_all.csv user_data_all.parquet user_data_all_issues.csv

6.3 Fact table cleaning details

Next, the detailed cleaning steps for the three fact tables will be listed below:

Fact line item Cleaning Details	
Source Files	line_item_data_prices1.csv, line_item_data_prices2.csv, line_item_data_prices3.csv, line_item_data_products1.csv, line_item_data_products2.csv, line_item_data_products3.csv , order_data_20200101-20200701.csv, order_data_20200701-20211001.csv, order_data_20211001-20220101.csv, order_data_20220101-20221201.csv, order_data_20221201-20230601.csv, order_data_20230601-20240101.csv , order_with_merchant_data1.csv, order_with_merchant_data2.csv, order_with_merchant_data3.csv,
Cleaning Steps (General)	1. Concatenated all related source files 2. Removed technical artifacts
Cleaning Steps (Line item prices data)	1. Extracted numeric quantity from messy strings (e.g., 6pcs, 4PCs) 2. Standardized price to numeric 3. Removed invalid or non-numeric quantities

	4. Preserved row-level alignment with products
Cleaning Steps (Order data)	1. Standardized transaction_date to proper datetime 2. Derived date_key in YYYYMMDD format for dimensional joins 3. Normalized estimated arrival into integer days 4. Preserved order_id as a degenerate identifier (not a dimension)
Cleaning Steps (Order with merchant data)	1. Normalized merchant_id and staff_id formats
Output file	orders_clean.csv, line_item_products_clean.csv, line_item_prices_clean.csv, order_with_merchant_clean.csv,

Fact campaign availed Cleaning Details	
Source Files	transactional_campaign_data.csv orders_data*.csv order_with_merchant_data*.csv campaign_data.csv
Cleaning Steps (All Files)	1. Concatenated all related source files into a single dataset. 2. Removed technical artifacts (Unnamed: 0). 3. Sort for deterministic joins.
Cleaning Steps (Transactional Campaign)	1. Parsed day values (e.g., 13days, 8days, 15days) into a numeric value. 2. Converted datetime to YYYYMDD integer key. 3. Normalized availed flag.

Cleaning Steps (Orders)	1. Standardized transaction_date to proper datetime 2. Derived date_key in YYYYMMDD format for dimensional joins 3. Normalized estimated arrival into integer days 4. Preserved order_id as a degenerate identifier (not a dimension)
Cleaning Steps (Order With Merchant)	1. Normalized IDs into Strings.
Cleaning Steps (Campaign)	1. Parsed and reconstructed columns 2. Removed excessive quotation marks 3. Normalized discount values into integer percentages 4. Duplicate campaign IDs resolved using Type 1 overwrite semantics 5. Invalid records (if any) written to campaign_data_issues.csv
Output file	transactional_campaign_clean.csv orders_clean.csv order_with_merchant_clean.csv campaign_data.csv

Fact orders Cleaning Details	
Source Files	line_item_data_prices*.csv order_data_*.csv order_with_merchant_data*.csv order_delays.csv campaign_data.csv
Cleaning Steps (All Files)	1. Concatenated all related source files into a single dataset. 2. Removed technical artifacts (Unnamed: 0). 3. Preserved one row per order (order-level grain).

Cleaning Steps (Line Item Prices)	1. Extracted numeric quantity from messy strings (e.g., 6pcs, 4PCs) 2. Standardized price to numeric 3. Removed invalid or non-numeric quantities 4. Preserved row-level alignment with products
Cleaning Steps (Order)	1. Standardized transaction_date to proper datetime 2. Derived date_key in YYYYMMDD format for dimensional joins 3. Normalized estimated arrival into integer days 4. Preserved order_id as a degenerate identifier (not a dimension)
Cleaning Steps (Order with Merchant)	1. Normalized IDs into Strings.
Cleaning Steps (Order Delays)	1. Parse day values (e.g., 13days, 8days, 15days) into a numeric value. 2. Convert datetime to YYYYMDD integer key. 3. Derived transaction_date_key (YYYYMMDD) for downstream dimensional joins.
Cleaning Steps (Campaign)	1. Parsed and reconstructed columns 2. Removed excessive quotation marks 3. Normalized discount values into integer percentages 4. Duplicate campaign IDs resolved using Type 1 overwrite semantics 5. Invalid records (if any) written to campaign_data_issues.csv
Output file	orders_clean.csv order_with_merchant_clean.csv line_item_prices_clean.csv campaign_data.csv

7. Data Staging Process

The staging layer has been expanded to support additional dimensional data and fact table data while strictly adhering to Kimball modeling principles. This enhancement introduces a set of well-defined staging tables that serve as a buffer between cleaned data and loading scripts. Moreover, the implemented staging tables include staging.user_data_all, staging.product_list, staging.staff_data, staging.merchant_data, and staging.campaign_data.

To ensure consistency, the following principles are enforced across all staging tables:

- No joins are performed at the staging level
- No surrogate keys are introduced
- No historical or slowly changing logic is applied
- Strict one-to-one alignment is maintained with cleaned CSV structures

Data ingestion into the staging layer is orchestrated through a single dedicated Airflow DAG which is responsible for the whole domain of the company:

- dag_stage_all_sources

This staging DAG follows an execution pattern to ensure repeatability and reliability:

1. Executes the corresponding data cleaning script
2. Creates the staging table if it does not already exist
3. Loads the cleaned data while enforcing strict column ordering

This approach ensures that the staging layer remains fully compliant with Kimball best practices, while providing a stable foundation for the next steps in the team's ETL process.

8. Data Loading Process

The data loading process is very straightforward where the staging layer is an essential pre-requisite. After all the necessary data has been successfully staged, python scripts containing logic to combine various staging tables will be executed to correctly validate and transform them into the proper Kimball Dimensions outlined in the brief methodology of this documentation. Additionally, these loaded data will be stored in the Postgres image of the

docker-compose container and PgAdmin will be utilized to further validate the result of the data loading process.

9. Data Dictionary

Listed below are the tables and each of its attributes used in the Kimball architecture, and starting with the Dimension tables first:

Name: dim_Campaign **Table type:** Dimension Table **SCD Strategy:** Type 1 **Grain:** One row per unique campaign			
Attribute	**Data Type**	**Constraint**	**Description**
campaign_key	Int	Primary Key, Surrogate Key	Unique Surrogate Key generated by the DW. Used for joining to Fact tables
campaign_id	Varchar(30)	Natural Key Unique, Not Null	The Natural Key (Business Key) from the source system
campaign_name	Varchar(255)		The official name of the campaign. Changes overwrite the current value.
campaign_description	TEXT		A detailed description of the campaign's purpose or goal. Changes overwrite the current value.
discount_pct	Int		The standard discount percentage offered by this campaign. Changes overwrite the current value.

Name: dim_Date **Table type:** Dimension Table **SCD Strategy:** N/A **Grain:** One row per calendar day			
Attribute	**Data Type**	**Constraint**	**Description**

date_key	Int	Primary Key, Surrogate Key	Unique Surrogate Key generated by the DW. Used for joining to Fact tables
full_date	Date	Not null	The full date value (e.g., 2025-12-15).
day	Int		Day of the month (1-31).
day_name	Varchar(10)		Full name of the day (e.g., Monday).
month	Int		Month number (1-12).
month_name	Varchar(10)		Full name of the month (e.g., December).
quarter	Int		Calendar quarter number (1-4).
quarter_name	Varchar(6)		Descriptive quarter name (e.g., Q4-2025).
year	Int		Four-digit calendar year.
is_weekend	Boolean		Flag (1/0) indicating if the day is a Saturday or Sunday.

Name: dim_Merchant
Table type: Dimension Table
SCD Strategy: Type 2
Grain: One row per merchant per effective time period

Attribute	Data Type	Constraint	Description
merchant_key	Int	Primary Key, Surrogate Key	Unique Surrogate Key generated by the DW. Used for joining to Fact tables
merchant_id	Varchar(30)	Natural Key Unique, Not Null	The Natural Key (Business Key) from the source system
name	Varchar(255)		The name of the merchant
street	Varchar(255)		Street address of the merchant
city	Varchar(100)		City address of the merchant
country	Varchar(100)		Country of the merchant

contact_number	Varchar(50)		Contact number of the merchant
effective_from	Date	Not null	The start date/time this record version became valid.
effective_to	Date		The end date/time this record version was valid. NULL indicates current.
is_current	Boolean	Not Null	Flag (1/0) indicating the active record for this Merchant ID.

Name: dim_Product
Table type: Dimension Table
SCD Strategy: Type 1
Grain: One row per unique product.

Attribute	Data Type	Constraint	Description
product_key	Int	Primary Key, Surrogate Key	Unique Surrogate Key generated by the DW. Used for joining to Fact tables
product_id	Varchar(30)	Natural Key Unique, Not Null	The Natural Key (Business Key) from the source system
product_name	Varchar(255)	Unique	The official name of the product. Updates overwrite the old value.
product_type	Varchar(100)		The category or type of the product. Updates overwrite the old value.
price	Numeric(10,2)		The most recent list price of the product. Updates overwrite the old value.

Name: dim_Staff
Table type: Dimension Table
SCD Strategy: Type 2
Grain: One row per staff member per effective time period.

Attribute	Data Type	Constraint	Description

staff_key	Int	Primary Key, Surrogate Key	Unique Surrogate Key generated by the DW. Used for joining to Fact tables
staff_id	Int	Natural Key Unique, Not Null	The Natural Key (Business Key) from the source system
name	varchar(255)		The staff member's full name. Changes trigger a new record version.
job_level	varchar(50)		The staff member's assigned job seniority or level. Promotion/demotion triggers a new record.
street	varchar(255)		The staff member's street address. Changes trigger a new record version.
state	varchar(100)		The staff member's state or region of residence/assignment. Changes trigger a new record.
city	varchar(100)		The staff member's city of residence/assignment. Changes trigger a new record.
country	varchar(100)		The staff member's country of residence/assignment. Changes trigger a new record.
contact_number	varchar(50)		The primary contact phone number. Changes trigger a new record version.
effective_from	Date	Not null	The start date/time this record version became valid.
effective_to	Date		The end date/time this record version was valid. NULL indicates current.
is_current	Boolean	Not Null	Flag (1/0) indicating the active record for this Staff ID.

Name: dim_User

Table type: Dimension Table

SCD Strategy: Type 2

Grain: One row per user version per effective time period.

Attribute	Data Type	Constraint	Description
user_key	Int	Primary Key, Surrogate Key	Unique Surrogate Key generated by the DW. Used for joining to Fact tables
user_id	Varchar(30)	Natural Key Unique, Not Null	The Natural Key (Business Key) from the source system
name	Varchar(200)		The user's full name. Changes should trigger a new record
gender	Varchar(20)		The user's gender. Changes trigger a new record
birth_date_key	Int	Foreign Key	Link to dim_Date for the user's birth date.
creation_date_key	Int	Foreign Key	Link to dim_Date for the date the user account was created.
street	Varchar(200)		The user's street address. Changes should trigger a new record.
city	Varchar(100)		The user's city. Changes should trigger a new record.
state	Varchar(100)		The user's state or region. Changes should trigger a new record.
country	Varchar(100)		The user's country. Changes should trigger a new record.
user_type	Varchar(50)		The user's classification. Changes should trigger a new record.
device_address	Varchar(50)		The last known IP or physical device address. Changes should trigger a new record.
job_title	Varchar(200)		The user's current job title. Changes should trigger a new record.
job_level	Varchar(200)		The user's job seniority or level. Changes should trigger a new record.
credit_card_number	Varchar(40)		The primary credit card number (should be tokenized or masked)
issuing_bank	Varchar(50)		The bank that issued the credit card. Changes trigger a new record

effective_from	Date	Not null	The start date/time this record version became valid.
effective_to	Date		The end date/time this record version was valid. NULL indicates current.
is_current	Boolean	Not null	Flag (1/0) indicating if this is the currently active version of the user.

Additionally, listed below are the data dictionaries of the team's current fact tables, the column SCD Type was changed to Additive Type since fact tables are for defining how measures can be mathematically summarized:

Name: fact_line_item

Table type: Transactional Fact Table

Grain: One row per product line item within a single order.

Attribute	Data Type	Key Type / Constraint	Additive Type	Description
fact_line_item_key	Int	Primary Key, Surrogate Key	N/A	The unique primary key for the fact record.
order_id	Varchar(100)	Degenerate Dimension	N/A	The source system ID of the order. Used to group line items belonging to the same transaction.
user_key	Int	Foreign Key	N/A	Foreign key to dim_user. Links the user's state (SCD Type 2) at the time of the order.
merchant_key	Int	Foreign Key	N/A	Foreign key to dim_Merchant. Links the merchant's state (SCD Type 2) at the time of the order.
staff_key	Int	Foreign Key	N/A	Foreign key to dim_Staff. Links the staff member's state (SCD Type 2) who processed the order.
product_key	Int	Foreign Key	N/A	Foreign key to dim_Product. Links the product's attributes (SCD Type 1) at the time of the order.
date_key	Int	Foreign Key	N/A	Foreign key to dim_Date (SCD

				not applicable). Links the calendar day of the order.
unit_price	Numeric (12, 2)		Non-Additive	The price of a single unit at the time of sale. Cannot be summed (Non-Additive).
quantity	Int		Fully Additive	The number of units sold in this line item. Can be summed across all dimensions.
line_amount	Numeric (14,2)		Fully Additive	The total revenue for this line item (unit_price * quantity). Can be summed across all dimensions.

Name: fact_orders

Table type: Transactional Fact Table

Grain: One row per complete order placed

Attribute	Data Type	Key Type / Constraint	Additive Type	Description
fact_orders_key	BIGSERIAL	Primary Key, Surrogate Key	N/A	The unique primary key for the fact record.
order_id	Varchar(100)	Degenerate Dimension, Not null	N/A	The source system ID of the order. Used to track the business transaction key.
user_key	Int	Foreign Key, Not Null	N/A	Foreign key to dim_user. Links the user's state (SCD Type 2) at the time of the order.
merchant_key	Int	Foreign Key, Not Null, Default 0	N/A	Foreign key to dim_Merchant. Links the merchant's state (SCD Type 2) at the time of the order.
staff_key	Int	Foreign Key, Not Null, Default 0	N/A	Foreign key to dim_Staff. Links the staff member's state (SCD Type 2) who processed the order.
date_key	Int	Foreign Key, Not Null	N/A	Foreign key to dim_Date (SCD not applicable). Links the calendar day of the order.
campaign_key	Int	Foreign Key,	N/A	Foreign key to dim_Campaign

		Not Null, Default -1, – -1		(SCD Type 1).
order_amount	Numeric (14, 2)	Not Null	Fully Additive	The total revenue/cost of the entire order (excluding tax/shipping). Can be summed.
estimated_arrival_in_days	Int		Semi-Additive	The delivery estimate provided to the customer in days. Can be averaged, but not summed.
delay_in_days	Int		Semi-Additive	The actual number of days the order was late. Can be averaged, but not summed.
availed	Int	Not null	Fully Additive	A flag measure representing the event (always 1). Used to count the number of times a campaign was availed.
discount_amount	Numeric(14, 2)		Non-Additive	The amount deducted to the order_amount.
discount_pct	Int		Non-Additive	The discount percentage (e.g., 15) recorded directly on the fact row for immediate reporting. This is also a degenerate dimension.
availed_flag	Int	Not Null, Default 0,0	N/A	Indicates if an order availed (1) an order or not (0).

10. Docker File and Docker-Compose File

10.1 Docker File

```
FROM apache/airflow:2.9.0-python3.10

# Set working directory inside container
ENV AIRFLOW_HOME=/opt/airflow

# Switch to airflow user (recommended best practice)
USER airflow

# Copy dependency list
COPY requirements.txt /requirements.txt

# Install Python packages needed by Airflow + your ETL scripts
RUN pip install --no-cache-dir -r /requirements.txt
```

The team's Dockerfile creates a custom Apache Airflow image with the required Python dependencies for the ETL pipelines. It starts from the official Apache Airflow 2.9.0 image with Python 3.10. The AIRFLOW_HOME environment variable is set to define Airflow's working directory inside the container. The container then switches to the airflow user, which is a best practice for security and compatibility with the Airflow image.

Next, the requirements.txt file is copied into the container, and all necessary Python dependencies are installed using 'pip'. This ensures that Airflow and the custom ETL scripts have all required libraries available when the container runs.

10.2 Docker-Compose File

```
version: "3.9"

services:
  postgres:
    image: postgres:13
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  pgadmin:
    image: dpage/pgadmin4
    environment:
      # These are the credentials for the pgAdmin web UI login, NOT the DB
      PGADMIN_DEFAULT_EMAIL: admin@admin.com
      PGADMIN_DEFAULT_PASSWORD: admin
    ports:
      # Access pgAdmin web UI at http://localhost:5050
      - "5050:80"
    depends_on:
      - postgres
      # Connect postgres by using:
      # name: ShopZada
      # hostname: postgres
      # port: 5432
```

```
      # database: airflow
      # username: airflow
      # password: airflow

  airflow-init:
    build:
      context: .
    depends_on:
      - postgres
    environment:
      AIRFLOW__CORE__EXECUTOR: LocalExecutor
      AIRFLOW__CORE__SQL_ALCHEMY_CONN:
postgresql+psycopg2://airflow:airflow@postgres:5432/airflow
      # AIRFLOW_CONN_POSTGRES_DEFAULT:
postgresql+psycopg2://airflow:airflow@postgres:5432/airflow
      AIRFLOW__WEBSERVER__SECRET_KEY: "supersecretkey123"
      AIRFLOW__CORE__TEMPLATE_SEARCHPATH: "/opt/airflow/sql"

      AIRFLOW__SCHEDULER__DAG_DIR_LIST_INTERVAL: 10
      AIRFLOW__SCHEDULER__MIN_FILE_PROCESS_INTERVAL: 10
      AIRFLOW__SCHEDULER__SCHEDULER_HEARTBEAT_SEC: 5
      AIRFLOW__SCHEDULER__PARSING_PROCESSES: 4

    entrypoint: >
      bash -c "airflow db init &&
      airflow users create --username admin --firstname Air --lastname
Flow --role Admin --email admin@example.com --password admin"

    volumes:
      - ./dags:/opt/airflow/dags
      - ./data_files:/data_files
      - ./scripts:/scripts
      - ./sql:/opt/airflow/sql
      - ./workflows:/workflows
      - ./clean_data:/clean_data

  airflow-webserver:
    build:
      context: .
    depends_on:
```

```
      - postgres
    environment:
      AIRFLOW__CORE__EXECUTOR: LocalExecutor
      AIRFLOW__CORE__SQL_ALCHEMY_CONN:
postgresql+psycopg2://airflow:airflow@postgres:5432/airflow
      # AIRFLOW_CONN_POSTGRES_DEFAULT:
postgresql+psycopg2://airflow:airflow@postgres:5432/airflow
      AIRFLOW__CORE__TEMPLATE_SEARCHPATH: "/opt/airflow/sql"
      AIRFLOW__WEBSERVER__EXPOSE_CONFIG: "True"
      AIRFLOW__WEBSERVER__SECRET_KEY: "supersecretkey123"
      AIRFLOW__CORE__DEFAULT_TIMEZONE: utc
      AIRFLOW__CORE__ENABLE_XCOM_PICKLING: "True"
      AIRFLOW__WEBSERVER__SESSION_BACKEND: securecookie
      AIRFLOW__SCHEDULER__DAG_DIR_LIST_INTERVAL: 10
      AIRFLOW__SCHEDULER__MIN_FILE_PROCESS_INTERVAL: 10
      AIRFLOW__SCHEDULER__SCHEDULER_HEARTBEAT_SEC: 5
      AIRFLOW__SCHEDULER__PARSING_PROCESSES: 4

    ports:
      - "8999:8080"
    command: webserver
    volumes:
      - ./dags:/opt/airflow/dags
      - ./data_files:/data_files
      - ./scripts:/scripts
      - ./sql:/opt/airflow/sql
      - ./workflows:/workflows
      - ./clean_data:/clean_data

  airflow-scheduler:
    build:
      context: .
    depends_on:
      - airflow-webserver
      - postgres
    environment:
      AIRFLOW__CORE__EXECUTOR: LocalExecutor
      AIRFLOW__CORE__SQL_ALCHEMY_CONN:
postgresql+psycopg2://airflow:airflow@postgres:5432/airflow
```

```
      # AIRFLOW_CONN_POSTGRES_DEFAULT:
postgresql+psycopg2://airflow:airflow@postgres:5432/airflow
      AIRFLOW__WEBSERVER__SECRET_KEY: "supersecretkey123"
      AIRFLOW__CORE__TEMPLATE_SEARCHPATH: "/opt/airflow/sql"

      AIRFLOW__CORE__DEFAULT_TIMEZONE: utc
      AIRFLOW__CORE__ENABLE_XCOM_PICKLING: "True"
      AIRFLOW__WEBSERVER__SESSION_BACKEND: securecookie

      AIRFLOW__SCHEDULER__DAG_DIR_LIST_INTERVAL: 10
      AIRFLOW__SCHEDULER__MIN_FILE_PROCESS_INTERVAL: 10
      AIRFLOW__SCHEDULER__SCHEDULER_HEARTBEAT_SEC: 5
      AIRFLOW__SCHEDULER__PARSING_PROCESSES: 4

    command: scheduler
    volumes:
      - ./dags:/opt/airflow/dags
      - ./data_files:/data_files
      - ./scripts:/scripts
      - ./sql:/opt/airflow/sql
      - ./workflows:/workflows
      - ./clean_data:/clean_data

volumes:
  postgres-db-volume:
```

This Docker Compose setup defines a complete local Apache Airflow environment, with each service having a clearly scoped responsibility. Together, these components provide database persistence, workflow orchestration, scheduling, and operational visibility in a reproducible and developer-friendly setup.

postgres (postgres:13)

- Acts as the metadata database for Airflow
- Stores Airflow state, DAG runs, and task metadata
- Persists data using a named Docker volume
- Exposes port 5432 for local access

pgadmin

- Web-based UI for managing and inspecting the Postgres database
- Accessible at http://localhost:5050
- Used for debugging, validation, and manual SQL queries

airflow-init

- Initializes the Airflow metadata database
- Creates the default Airflow admin user
- Runs once during setup
- Mounts DAGs, scripts, SQL, and data directories

airflow-webserver

- Hosts the Airflow Web UI
- Accessible at http://localhost:8999
- Loads DAGs and configuration from mounted volumes

airflow-scheduler

- Responsible for scheduling and executing DAG tasks
- Monitors DAG files and triggers task runs

Volumes

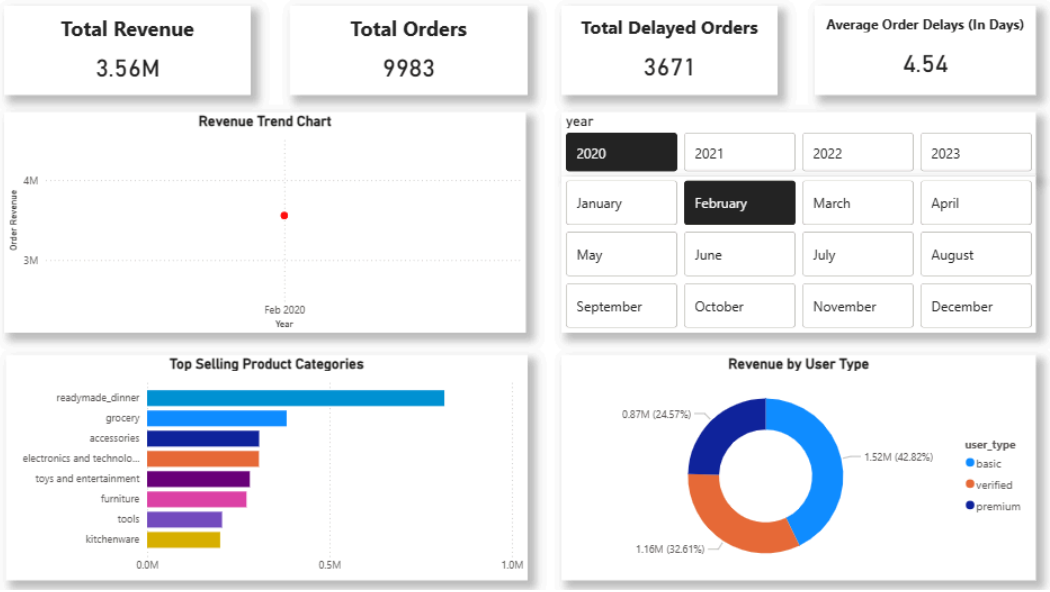
- Contains postgres-db-volumes that ensure Postgres data persists across container restarts

11. Analytical Layer and Dashboard UI

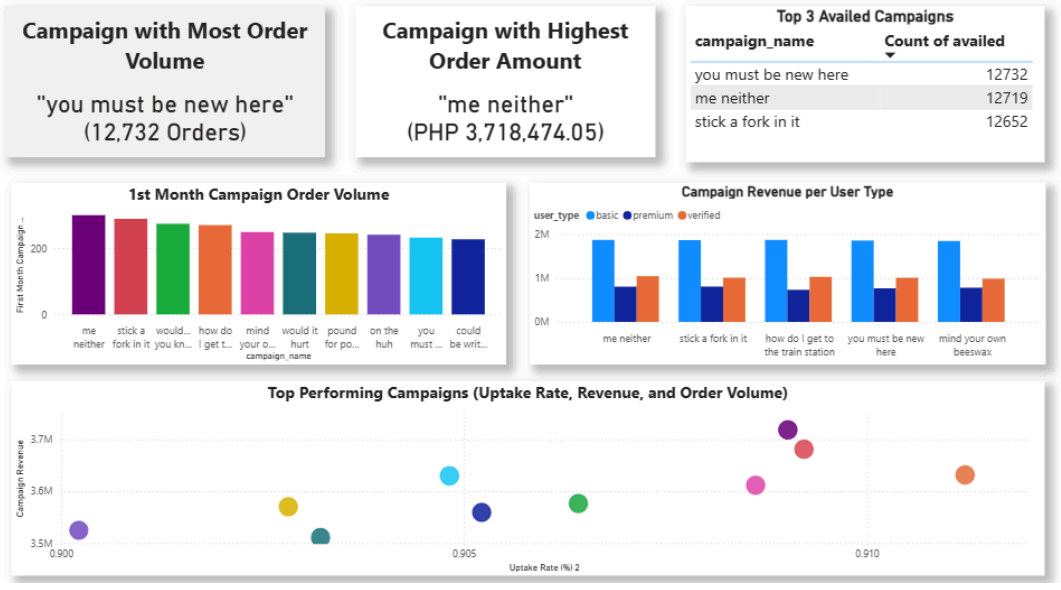
Below is the dashboard user interface made in Power BI that satisfies each business question listed on “2. Core Business Questions to address”. Additionally, the layout and design of the Analytical layer is intended to provide convenience and allow instant understanding of the data presented to further accelerate their decision making activities. Finally, filtering by year and

month on data is synced across the entire dashboard to also support date filter and aggregations:

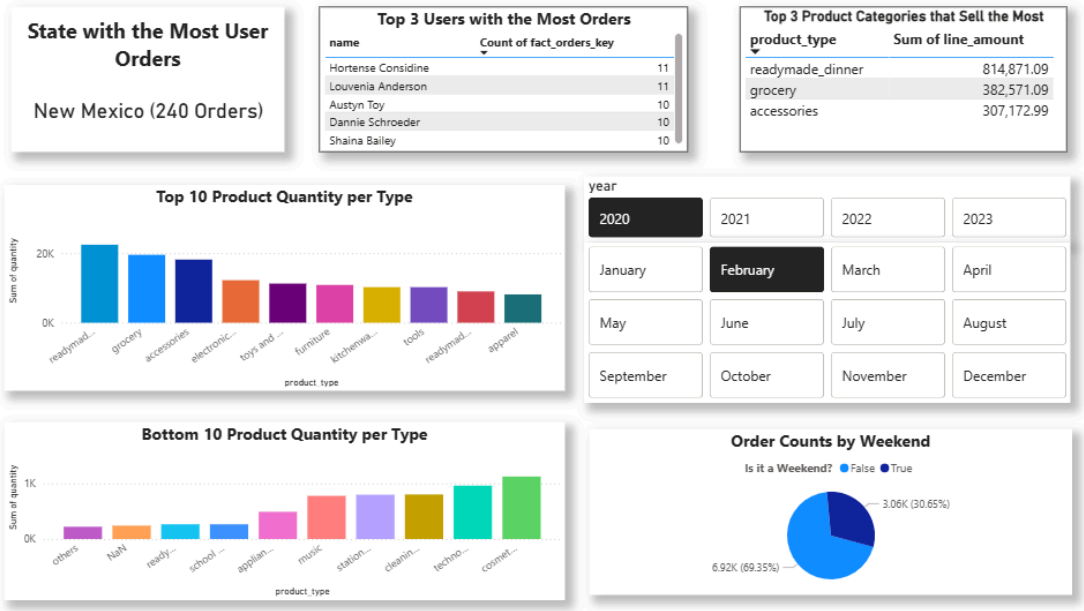
a. Executive Dashboard Tab



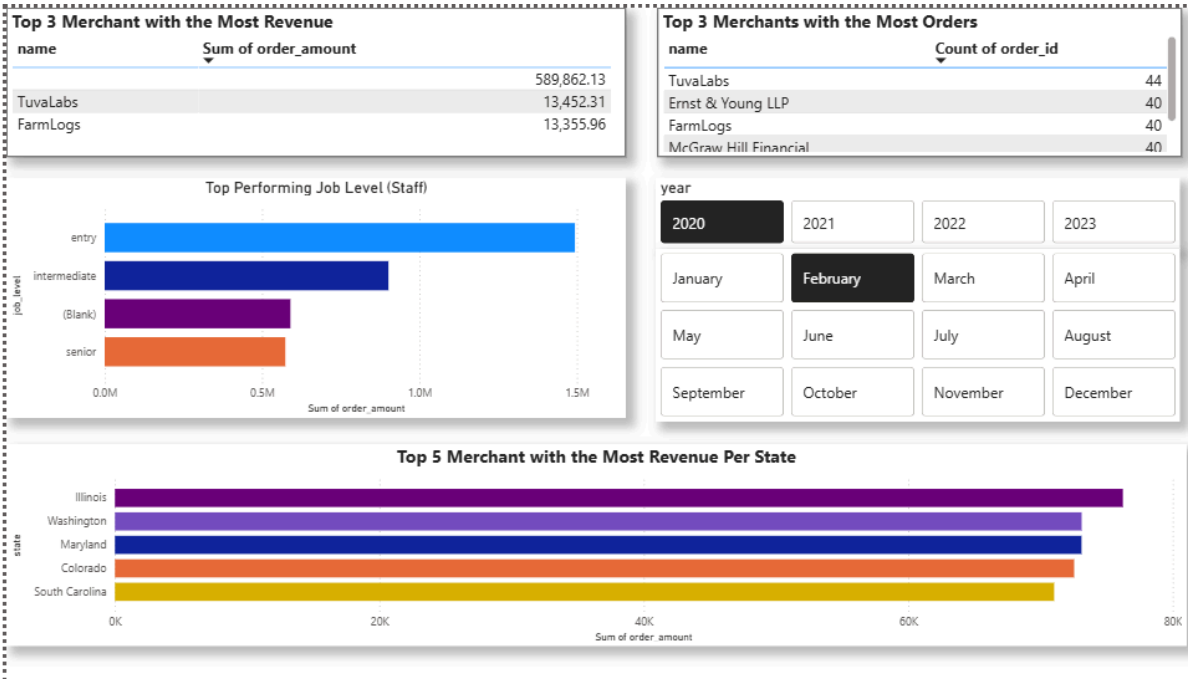
b. Campaign Tab



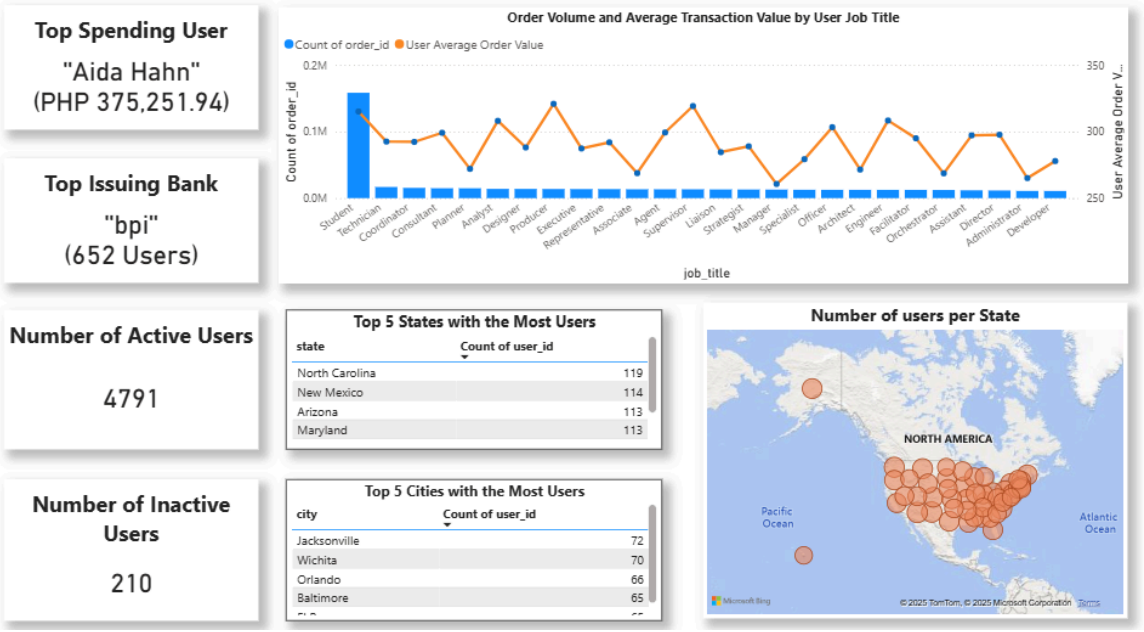
c. Orders Tab



d. Merchant and Staff Tab



e. Users Tab



12. Conclusion

The proposed Data Warehouse of the team for ShopZada 2.0 aims to unify fragmented data from the Business, Customer Management, Enterprise, Marketing, and Operations department into a single and consistent architecture. Furthermore, the chosen architecture for this project is a Kimball-style architecture for its straightforward implementation, optimized queries, and aligned for subject-area reporting. Through a hybrid ETL process using Python, Apache Airflow, PostgreSQL, and Docker, the group delivered a repeatable pipeline from raw CSV files, cleaned tables, staging, dimensional models, and analytical views.

Throughout the duration of developing the data warehouse, the team relied on GitHub as a tool in sharing our progress and tracking errors through commits, for push and pull requests, and creating separate branches to streamline development and prevent version conflicts.

The resulting star schema that is anchored by fact tables for orders, line items, and campaign availments and supported by conformed dimensions for users, products, merchants, staff, and dates will directly answer ShopZada's core business questions which focus on campaign performance, customer behavior, employee productivity, and other concerns(?) about the business.

Finally, the team will present these models through Power BI and Metabase to enable ShopZada's business stakeholders to further understand the data through our dashboards while still allowing flexible analysis. This foundation will position ShopZada to scale their analytics,

improve decision making, and further extend the developed warehouse with additional facts, dimensions, use cases, and other components in the future.