Mini Motors

CONTENTS

1 BUSINESS DESCRIPTION

1.1 BUSINESS BACKGROUND

Mini Motors is an online toy company that sells transportation-themed toys: cars, planes, ships, etc. This company produces toys in factories and sells them online all over the world. Their clients are either small toy store companies (legal entities) or individuals. Selling products online and fulfilling orders for customers all over the world is a complex process involving shipping products, analyzing revenue, and assessing profitability by manipulating data according to the quantity sold and prices paid for products by customers. The company grew significantly, especially in the last two years (2023–2024), and the increasing demand has generated some problems regarding the firm's future perspective and how they will distribute their finances.

1.2 PROBLEMS BECAUSE OF POOR DATA MANAGEMENT

The company coordinators realized that, along with the high rate of sales, it's important to organize orders and sales data to gain better insights into how the company actually operates and what actions would be optimal to take in the future to eliminate confusion and misunderstanding of many important details, which are unfortunately present in the company now. Poor data management prevents Mini Motors from determining which products are sold the most, which regions are the most active, or even which customers deserve promotions based on their loyalty. All these details need to be analyzed to increase the company's reputation and, consequently, sales even further.

1.3 BENEFITS FROM IMPLEMENTING A DATA WAREHOUSE

Using data warehouse can help the company with the problems described above. Implementing a data warehouse can answer the following questions:

- Which product line is the most demanded and generates the highest revenue?
- Which region is the leader in sales?
- Is there any correlation between time periods of the year and product orders?
- Is there any significant difference between companies' and individuals' orders and

  Which types of customers are more profitable?

Further processing data would also let the company determine sales trends from many different perspectives.

1.4 DATASETS DESCRIPTION

Two OLTP systems were obtained for MiniMotors company: the first one tracks orders made by individuals, and the second one describes orders made by businesses. The purpose of DWH is to integrate these two systems into one structured schema to optimize for analysis.

The first dataset contains the following information about sales of Mini Motors company:

Orders details:

Order code: unique code of each order.

Order status: shipped, delivered, cancelled etc.

Product Information:

Product code: unique code of a product subcategory

Product group code: code of products category, we have seven categories in this dataset.

Product group: The category of products, either planes, cars, or something else.

Customer Information:

Customer name: the name of the customer companies who made the order.

Company code: unique code for the company.

Customer type: in this case “company” because this dataset tracks orders made by companies only.

Address: address of the customer(company).

Postal code: unique postal code for each address.

City: city where the customer company is located.

Country: country where the customer company is located.

Times:

Date: date of placing an order.

Year: year used to track order years.

Quarter: Quarter in which order was made.

Month: The exact month of the order.

Payment information:

Payment code: unique code of each payment record.

Payment type: PayPal, Credit card or debit Card.

Discount information:

Discount code: Unique code which was provided by customers to get the discount.

Discount rate: Percentage of how much was discounted.

The second dataset contains the following information about sales of Mini Motors company:

Orders details:

Order code: unique code of each order.

Order status: shipped, delivered, cancelled etc.

Product Information:

Product code: unique code of a product subcategory

Product group code: code of products category, we have seven categories in this dataset.

Product group: The category of products, either planes, cars, or something else.

Customer Information:

Customer first name: the first name of the customer who made the order.

Customer last name: the last name of the customer who made the order.

Customer code: unique code for the customer.

Customer type: in this case “Individual” because this dataset tracks orders made by individuals only.

Address: address of the customer.

Postal code: unique postal code for each address.

City: city where the customer lives.

Country: country where the customer lives.

Times:

Date: date of placing an order.

Year: year used to track order years.

Quarter: Quarter in which order was made.

Month: The exact month of the order.

Payment information:

Payment code: unique code of each payment record.

Payment type: PayPal, Credit card or debit Card.

Discount information:

Discount code: Unique code which was provided by customers to get the discount.

Discount rate: Percentage of how much was discounted.

The difference between two source files

As we can see, the second dataset contains almost the same entities as the first, but there is a key difference in terms of customers. In the second dataset, all customers are individuals, and only individual orders are tracked, while in the first dataset, only companies' orders are provided. Because of this, we have a difference between their attributes: in the first source file, customer company names are provided, while in the second source file, we have the customer's first and last name separately.

Description of sales and cost measures in each database:

Other than the above-mentioned entities, we have sales and cost measures in each dataset which will be displayed in the fact table of the dimensional model. For example, unit price of products in each order, quantity of products in order, cost of each product (so called COGS which is the sum of all direct costs associated with making a product), and also total sales which is basically a gross revenue (price * quantity) without subtracting the discounted amount.

To sum up, the datasets provide a comprehensive overview of sales of Mini Motors store differentiating between individuals’ and companies’ orders, allowing for analysis and exploration of trends, sales performance, customer preferences, and more within the toy store industry.

1.5 STEPS OF DIMENSIONAL MODEL: GRAIN / DIM / FACT

1. The business process

   Designing an effective star scheme begins with identifying the correct business process. Operational activities which are performed by Mini Motors involve managing orders to provide ordered products (toys) to customers to the shipping location. However, other than managing the orders they track all necessary sales data for analytical purposes: order date, amount/quantity of products in order, cost of producing the unit of product, customer information (name, surname, their codes, address). In addition to this, company generates data about total gross revenue and one essential detail is also discounts metrics – because sometimes the company implements discounts to drive sales, they track this cost as well and use it to analyze net revenue.

2. grain

   The most suitable grain which can be declared for Mini Motor’s data is the following:

   One row = One product within a single order.

This ensures the fact table tracks detailed metrics such as price, cost, quantity, and discounts for each product in the order. Because the company's sales are based on online orders, it's essential to start measurements based on orders. This implementation enables us to create a grain which will be most applicable for the company's data. In each order there can be only **One product**, but with multiple units (e.g., Order ID=5 has Product ID=7 with Quantity = 40). It might seem a little confusing why only 1 type of products are in each order and not many different ones and here is the company's approach and strategy: "We allow only one product per order to make things easier for both customers and our business. For example, whether a person buys 5 units of a product, or a company buys 50, handling just one product per order makes packing, shipping, and stock tracking simpler. It also avoids errors while applying discounts. If a customer needs different products, they can place separate orders, which helps our company manage inventory more smoothly. This approach keeps things organized and efficient for all types of buyers."

3. dimensions

dimensional entities contain descriptive attributes which give context and meaning to business's operational activities. With the grain of the fact table in mind, there can be the following dimensions identified for Mini Motors: customers, orders, payments, products, times, discounts.

Dim_Products

This dimension contains details about the products, including their unique identifier, code, category, and the group they belong to.

Column name	Description	Data Type
PROD_ID	PK. Unique identifier of each product	Int generated always as identity
PROD_CODE	Natural key. Unique code of each product	Varchar (20), unique
PROD_CATEGORY_CODE	Unique code of each product category	Varchar (20), Unique
PROD_CATEGORY	The name of the product groups/categories	Varchar (50)

Example with filled data

PROD_ID	PROD_CODE	PROD_CATEGORY_CODE	PROD_CATEGORY
5	S24_4620	CC100	plane

Dim_Orders

This dimension stores information related to customer orders, including the order's unique ID and its status (e.g., shipped, delivered).

Column Name	Description	Data Type
ORDER_ID	PK. Unique identifier of each order	Int generated always as identity
ORDER_CODE	Natural key	INT
ORDER_STATUS	The status of the order (e.g., shipped, delivered)	Varchar (20)

Example with filled data

ORDER_ID	ORDER_CODE	ORDER_STATUS
1001	123456	Shipped

Dim_Customers

This dimension holds customer-specific details like their ID, name, type, and location (address, postal code, city, and country).

Column Name	Description	Data Type
CUST_ID	PK. Unique identifier of each customer	Int generated always as identity
CUST_CODE	Natural key- unique code for each customer	INT
CUST_FIRST_NAME	Customer’s first name	Varchar (20)
CUST_LAST_NAME	Customer’s surname	Varchar (20)
CUST_COMPANY_NAME	The name of the customer company	Varchar (50)
CUST_TYPE	Type of customer (e.g., individual, company)	Varchar (20)
ADDRESS	Address of customer or company	Varchar (100)

POSTAL_CODE	Unique Postal code for each address	INT
CITY	City for the customer address	Varchar (50)
COUNTRY	Country for the customer address	Varchar (50)

Example with filled data

CUST_ID	CUST_FIRST_NAME	CUST_LAST_NAME	CUST_COMPANY_NAME	CUST_CODE	Cust_type	ADDRESS	POSTAL_CODE	CITY	COUNTRY
50	Jayden	Murphy	NULL	89656	individual	7734 Strong St.	17257	San francisco	USA

Dim_Payment

This dimension records payment information, including the payment method (e.g., PayPal, debit, or credit).

Column Name	Description	Data Type
PAYMENT_ID	PK. Unique identifier for each payment method	Int generated always as identity
PAYMENT_CODE	Natural key, unique code of each payment record	INT
PAYMENT_METHOD	Payment method (e.g., PayPal, debit, credit)	Varchar (20)

Example with filled data

PAYMENT_ID	PAYMENT_CODE	PAYMENT_METHOD
54000	20000508	Credit Card

Dim_Discount

This dimension tracks the discounts applied to orders, storing the discount ID and the corresponding discount rate.

Column Name	Description	Data Type
DISCOUNT_ID	PK. Unique identifier of each discount	Int generated always as identity
DISCOUNT_CODE	Natural key, code which was used to get a discount	Varchar (20)
DISCOUNT_RATE	Discount rate applied to orders	INT

Example with filled data

DISCOUNT_ID	DISCOUNT_CODE	DISCOUNT_RATE
2001	VMG197	7%

Dim_DT

This dimension stores date-related information, including the date, month, quarter, and year of the transaction.

Column Name	Description	Data Type
TIME_ID	PK. Unique identifier for each time-related record,	Date
QUARTER	Quarter number (e.g., 1, 2, 3, 4)	Int
YEAR	Year of the transaction	Int
MONTH	Month number (e.g., 1 for January, 12 for December)	Int

Example with filled data

TIME_ID	QUARTER	MONTH	YEAR
01/01/2023	1	1	2023

4. Fact

Analyzing operational activities of company is very useful to find logic on how to design star schema. The business process events capture performance metrics that are translated into facts in a facts table. The fact table representing Mini Motors' sales data is centered around order processing. The central fact table represents individual products in each unique order and captures transactional metrics like price, cost of producing unit, quantity of products in each order, discounts, and customers, products, and time keys to connect with dimensions and give context to the sales data which lacks context without extra layers.

Fact Table description:

Fact_Orders

The **Fact_Orders** table tracks transactional sales data (primary facts), including quantity, price, cost, amount, gross revenue, and is linked to relevant dimensions via keys for detailed sales analysis.

Column Name	Description	Data Type
ORDER_ID	FK. Unique identifier linking to the Orders dimension	Int
PROD_ID	FK. Unique identifier linking to the Products dimension	Int
CUST_ID	FK. Unique identifier linking to the Customers dimension	Int
DISCOUNT_ID	FK. Unique identifier linking to the Discounts dimension	Int
PAYMENT_ID	FK. Unique identifier linking to the Payments dimension	Int
TIME_ID	FK. Unique identifier linking to the Time dimension	Int
QUANTITY	Quantity of the product in the order	Int
PRICE_EACH	Price of a single unit of the product	Decimal (10,2)
COST_EACH	Cost of a single unit of the product	Decimal (10,2)
GROSS_REVENUE	Gross revenue for the product before discounts (Price * Quantity)	Decimal (10,2)

Example with filled data

ORDER_ID	PROD_ID	CUST_ID	DISCOUNT_ID
1001	1001	501	2001

PAYMENT_ID	TIME_ID	QUANTITY	PRICE_EACH
20000508	202301	5	40

COST_EACH	GROSS_REVENUE
20	200

2 BUSINESS LAYER 3NF

Slowly Changing Dimensions (SCD) are methods used to manage and track changes in data over time. It is particularly useful when we are dealing with data that will possibly evolve or change, for example, customer information, product details, or transactional data.

The most common types of SCD are Type 0, Type 1, and Type 2. Type 0 does not track changes because dimensional values remain unchanged and can’t be modified. Type 1 updates the dimension by overwriting old values with new ones, which means historical data is lost. Type 2 tracks changes and creates a new row with a new unique identifier for every version of the data, saving historical records.

SCD1 implementation

Considering the business needs and all the columns that might potentially change, SCD1 was implemented for the customer entity. The purpose is that if a person gets married and their surname changes, it will be useful to directly overwrite the old last name to ensure consistency across source systems and the data warehouse for proper analysis.

SCD2 Implementation

After evaluating the dataset and the business requirements, I decided to implement SCD2 on the Products table. This decision was made because tracking historical changes of product details is crucial to perform in depth data analysis over time. Key product attributes such as price and cost of production might change over time, and understanding when these changes happened is important for MiniMotors’ owners to perform historical sales analysis. For example, when there is the need to analyze sales performance or profitability, the company might also want to know not only what the current price or cost is but also how these attributes have changed in the past.

Each record will also have the necessary **INSERT_DT**, **UPDATE_DT, IS_ACTIVE and END_DT** columns to indicate the validity period of each version.

Designing the Data Model in 3NF

The data model was designed to follow the Third Normal Form (3NF) which ensures that the schema eliminates data redundancy and is more consistent. This process was implemented carefully to ensure that each table only contains information about a single entity and subject.

- **Normalization Process**: All entities were analyzed to ensure that non-key attributes depend only on the primary key. Primary keys will be generated as a surrogate key and will uniquely identify records. Additionally, data was carefully analyzed to prevent transitive dependencies because attributes must be fully dependent on the primary key.
- **Source Triplet and Surrogate Keys**: source triplet columns: **SOURCE_SYSTEM**, **SOURCE_ENTITY**, and **SOURCE_ID** were added in each table to track the origin of each record. These fields ensure that the system can differentiate between different data sources and handle any inconsistencies across datasets. The majority of tables had a natural key from the source files, which was used as a source id, however, the case of several tables requires further clarification. For example, in case of cities and countries tables natural keys from source systems were not available, but because city and country names are unique, those attributes were used as a source id. For instance, CITIES_SOURCE_ID in the cities table refers to the names of cities from original source datasets, same happens for countries table as well. Also, for table addresses, postal codes from source files are used as a source_id because it is unique for every address, therefore, ADDRESSES_SOURCE_ID refers to the column POSTAL_CODE from source datasets.

- **INSERT_DT, UPDATE_DT COLUMNS**: each table has insert and update date columns. Additionally, IS_ACTIVE and END_DT columns are added for SCD2 table to probably have update and validity dates of records.

3nf model entities:

- **CE_PRODUCTS_SCD**:

  Columns: PROD_ID (PK), PRODUCTS_SRC_ID, PRICE_EACH, COST_EACH, PROD_CATEGORY_ID (FK), SOURCE_SYSTEM, SOURCE_ENTITY, IS_ACTIVE, INSERT_DT, UPDATE_DT, END_DT

- **CE_PRODUCT_CATEGORIES**:

  Columns: PROD_CATEGORY_ID (PK), PROD_CATEGORY_SRC_ID , PROD_CATEGORY, SOURCE_SYSTEM, SOURCE_ENTITY, INSERT_DT, UPDATE_DT

- **CE_ORDERS**:

  Columns: ORDER_ID (PK), ORDERS_SRC_ID, PRODUCT_ID (FK), QUANTITY, ORDER_DT, ORDER_STATUS, SOURCE_SYSTEM, SOURCE_ENTITY, INSERT_DT, UPDATE_DT

- **CE_PAYMENTS**:

Columns: PAYMENT_ID (PK), PAYMENTS_SRC_ID, ORDER_ID (FK), CUSTOMER_ID (FK), GROSS_AMOUNT, PAYMENT_TYPE, DISCOUNT_ID (FK), SOURCE_SYSTEM, SOURCE_ENTITY, INSERT_DT, UPDATE_DT

- **CE_DISCOUNTS**:

  Columns: DISCOUNT_ID (PK), DISCOUNT_SRC_ID, DISCOUNT_RATE, SOURCE_SYSTEM, SOURCE_ENTITY, INSERT_DT, UPDATE_DT

- **CE_CUSTOMERS**:

  Columns: CUST_ID (PK), CUST_SRC_ID, CUST_FIRST_NAME, CUST_LAST_NAME, CUST_COMPANY_NAME, CUST_TYPE, SOURCE_SYSTEM, SOURCE_ENTITY, INSERT_DT, UPDATE_DT

- **CE_ADDRESSES**:

  Columns: ADDRESS_ID (PK), ADDRESS, POSTAL_CODE, CITY_ID (FK), SOURCE_SYSTEM, SOURCE_ENTITY, SOURCE_ID, INSERT_DT, UPDATE_DT

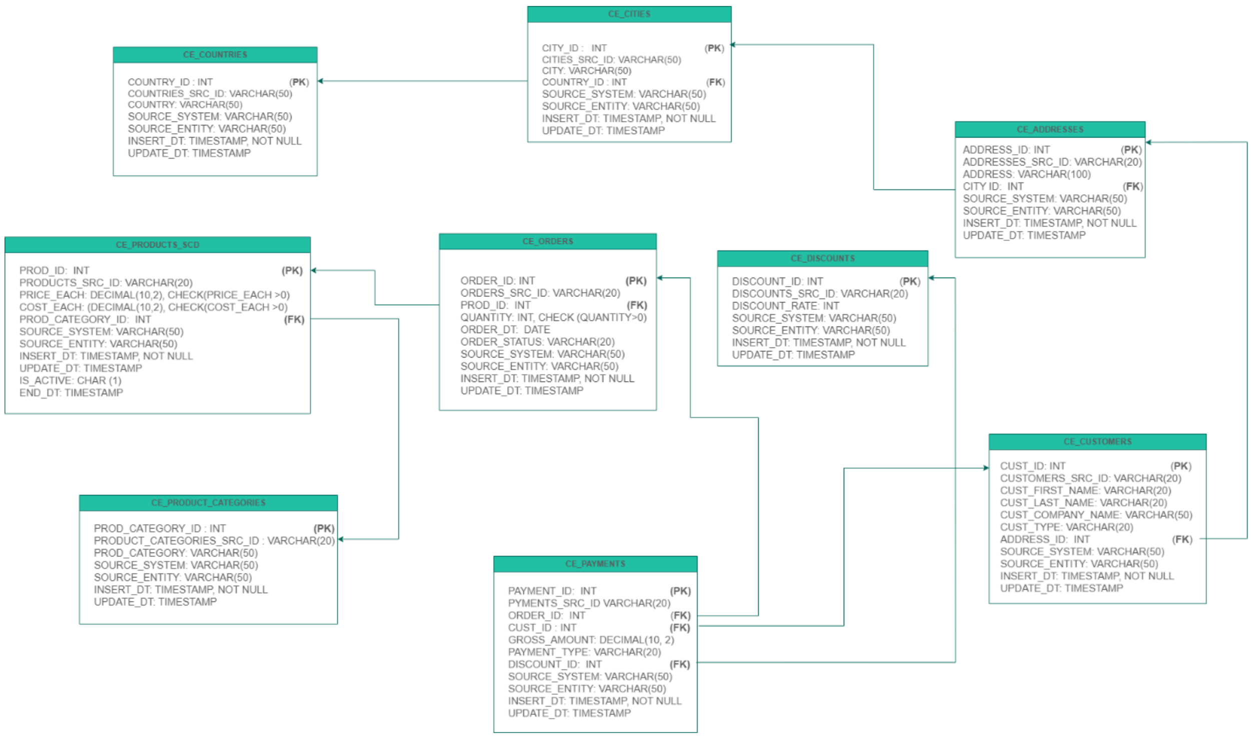
- **CE_COUNTRIES**:

  Columns: COUNTRY_ID (PK), COUNTRIES_SRC_ID, COUNTRY, SOURCE_SYSTEM, SOURCE_ENTITY, INSERT_DT, UPDATE_DT

- **CE_CITIES**:

  Columns: CITY_ID (PK), CITIES_SRC_ID, CITY, COUNTRY_ID (FK), SOURCE_SYSTEM, SOURCE_ENTITY, INSERT_DT, UPDATE_DT

3NF SCHEME


CE_COUNTRIES
COUNTRY_ID : INT (PK)
COUNTRIES_SRC_ID: VARCHAR(50)
COUNTRY: VARCHAR(50)
SOURCE_SYSTEM: VARCHAR(50)
SOURCE_ENTITY: VARCHAR(50)
INSERT_DT: TIMESTAMP, NOT NULL
UPDATE_DT: TIMESTAMP
CE_CITIES
CITY_ID : INT (PK)
CITIES_SRC_ID: VARCHAR(50)
CITY: VARCHAR(50)
COUNTRY_ID : INT (FK)
SOURCE_SYSTEM: VARCHAR(50)
SOURCE_ENTITY: VARCHAR(50)
INSERT_DT: TIMESTAMP, NOT NULL
UPDATE_DT: TIMESTAMP
CE_ADDRESSES
ADDRESS_ID: INT (PK)
ADDRESSES_SRC_ID: VARCHAR(20)
ADDRESS: VARCHAR(100)
CITY ID: INT (FK)
SOURCE_SYSTEM: VARCHAR(50)
SOURCE_ENTITY: VARCHAR(50)
INSERT_DT: TIMESTAMP, NOT NULL
UPDATE_DT: TIMESTAMP
CE_PRODUCTS_SCD
PROD_ID: INT (PK)
PRODUCTS_SRC_ID: VARCHAR(20)
PRICE_EACH: DECIMAL(10,2), CHECK(PRICE_EACH >0)
COST_EACH: (DECIMAL(10,2), CHECK(COST_EACH >0)
PROD_CATEGORY_ID: INT (FK)
SOURCE_SYSTEM: VARCHAR(50)
SOURCE_ENTITY: VARCHAR(50)
INSERT_DT: TIMESTAMP, NOT NULL
UPDATE_DT: TIMESTAMP
IS_ACTIVE: CHAR (1)
END_DT: TIMESTAMP
CE_ORDERS
ORDER_ID: INT (PK)
ORDERS_SRC_ID: VARCHAR(20)
PROD_ID: INT (FK)
QUANTITY: INT, CHECK (QUANTITY>0)
ORDER_DT: DATE
ORDER_STATUS: VARCHAR(20)
SOURCE_SYSTEM: VARCHAR(50)
SOURCE_ENTITY: VARCHAR(50)
INSERT_DT: TIMESTAMP, NOT NULL
UPDATE_DT: TIMESTAMP
CE_DISCOUNTS
DISCOUNT_ID: INT (PK)
DISCOUNTS_SRC_ID: VARCHAR(20)
DISCOUNT_RATE: INT
SOURCE_SYSTEM: VARCHAR(50)
SOURCE_ENTITY: VARCHAR(50)
INSERT_DT: TIMESTAMP, NOT NULL
UPDATE_DT: TIMESTAMP
CE_CUSTOMERS
CUST_ID: INT (PK)
CUSTOMERS_SRC_ID: VARCHAR(20)
CUST_FIRST_NAME: VARCHAR(20)
CUST_LAST_NAME: VARCHAR(20)
CUST_COMPANY_NAME: VARCHAR(50)
CUST_TYPE: VARCHAR(20)
ADDRESS_ID: INT (FK)
SOURCE_SYSTEM: VARCHAR(50)
SOURCE_ENTITY: VARCHAR(50)
INSERT_DT: TIMESTAMP, NOT NULL
UPDATE_DT: TIMESTAMP
CE_PRODUCT_CATEGORIES
PROD_CATEGORY_ID : INT (PK)
PRODUCT_CATEGORIES_SRC_ID : VARCHAR(20)
PROD_CATEGORY: VARCHAR(50)
SOURCE_SYSTEM: VARCHAR(50)
SOURCE_ENTITY: VARCHAR(50)
INSERT_DT: TIMESTAMP, NOT NULL
UPDATE_DT: TIMESTAMP
CE_PAYMENTS
PAYMENT_ID: INT (PK)
PYMENTS_SRC_ID VARCHAR(20)
ORDER_ID: INT (FK)
CUST_ID : INT (FK)
GROSS_AMOUNT: DECIMAL(10, 2)
PAYMENT_TYPE: VARCHAR(20)
DISCOUNT_ID: INT (FK)
SOURCE_SYSTEM: VARCHAR(50)
SOURCE_ENTITY: VARCHAR(50)
INSERT_DT: TIMESTAMP, NOT NULL
UPDATE_DT: TIMESTAMP

3 BUSINESS LAYER DIMENSIONAL MODEL

After modelling 3nf schema the next step is dimensional model, more specifically star schema, which is more redundant than 3nf model, but it makes querying and analysis process quicker and more convenient. In the Star Scheme of MiniMotors' business case, we have the following dimensions and facts:

Dimensions:

- **DIM_ORDER_DETAILS**: Contains order-level details like order ID and status information.
- **DIM_PRODUCTS_SCD**: Stores product details, including pricing and category information.
- **DIM_DATES**: Represents the date dimension with attributes like day, month, quarter, and year.
- **DIM_PAYMENTS**: Holds payment-related details, such as payment type.
- **DIM_CUSTOMERS**: Stores customer information, including names and address details.
- **DIM_DISCOUNTS**: Contains discount-related information, such as discount rate and discount code.

Fact:

- **FCT_ORDERS**: Fact table that captures transactional data like order ID, product ID, customer ID, quantity, price, cost, gross amount, net amount.

There are 2 calculated metrics in the fact table:

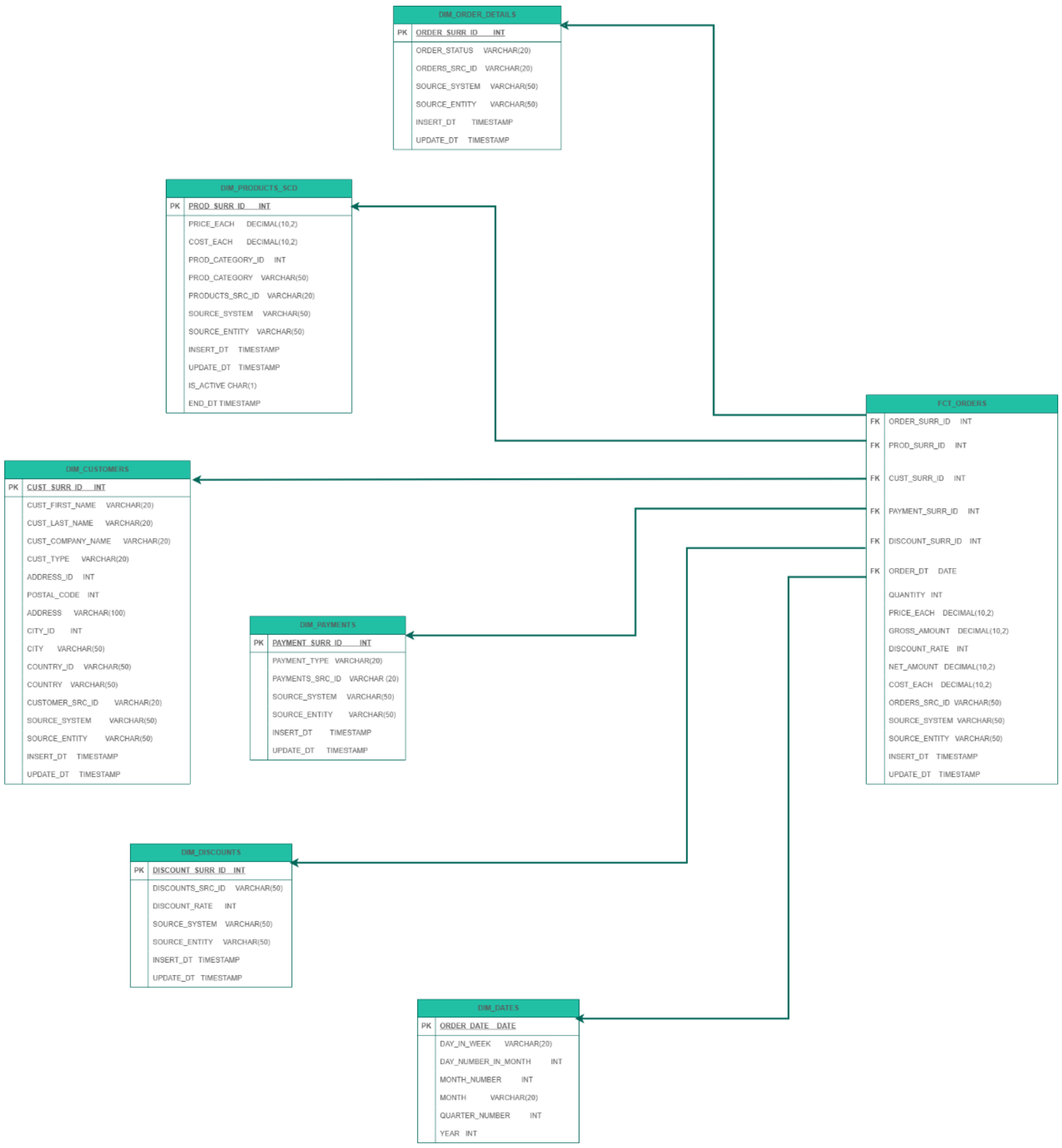
1. GROSS_AMOUNT which is calculated by **PRICE_EACH * QUANTITY**
2. NET_AMOUNT which is calculated by **GROSS_AMOUNT** – (**GROSS_AMOUNT * DISCOUNT_RATE)**

   And shows the amount of net revenue earned by firm from each order after substracting the amount which was discounted for customers.

Schema is designed in a way that descriptive data (dimensions) are separated from transactional data (fact), improving query efficiency and providing structured data for analysis.

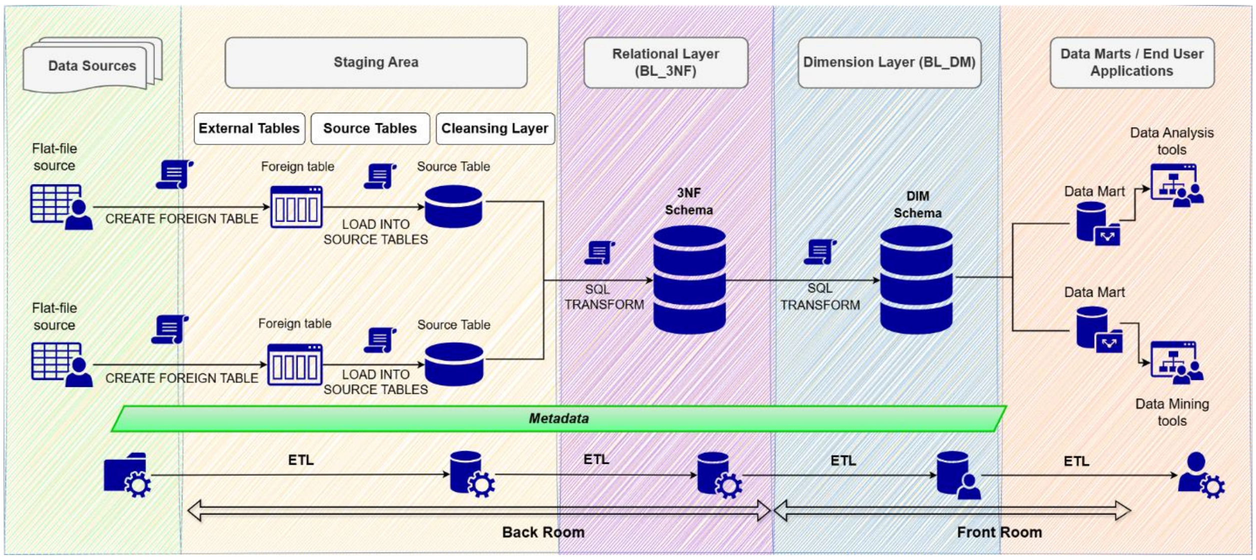
Here is the result:

STAR SCHEME



4 LOGICAL SCHEME

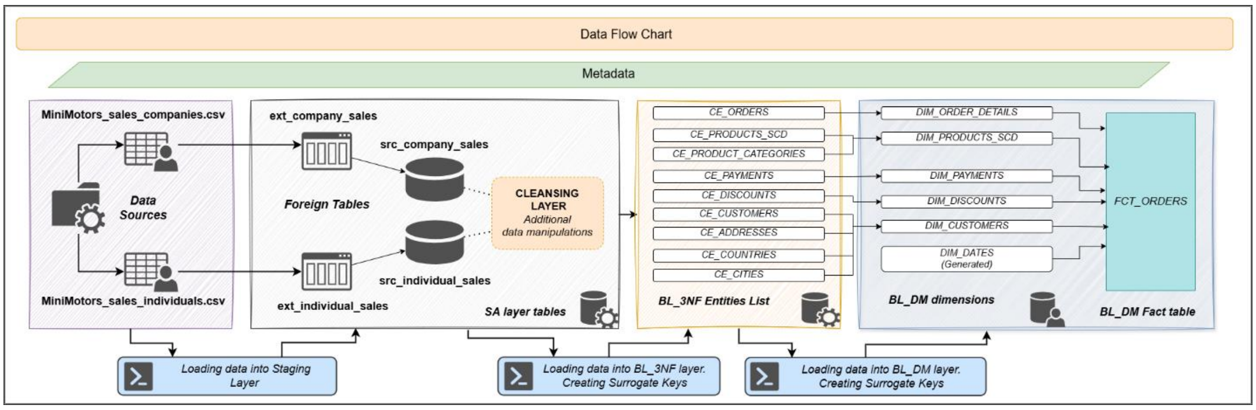
A Logical Schema in a Data Warehouse defines the conceptual structure of data. Initially, data flows from source systems into a staging area and is then transformed into 3NF (Third Normal Form) for integration and consistency. Later, data gets loaded into dimensional model which can either be star or snowflake schema, but most of the times it's star, because it optimizes query performance and analytics more. Finally, after data is stored in **Data Marts**, it is processed using **Data Mining** and **Data Analysis** tools for gaining insights and supporting decision-making process.



5 DATA FLOW

The data flow diagram illustrates the transformation of data from a 3NF model into a dimensional model with more details and specifications. Initially, raw data from source systems are loaded into a staging area for cleaning, next it goes through normalization process in 3NF tables, which is essential to ensure data consistency. These normalized tables are then divided into dimensional tables that include descriptive features such as customer, payment, or product details. Meanwhile, transactional data is converted into a fact table that contains numerical measurements and is connected to dimensions via keys. This denormalized structure improves query performance and enables efficient reporting and analytics.

data flow diagram



6 FACT TABLE PARTITIONING STRATEGY

The FCT_ORDERS table uses a partitioning strategy based on the ORDER_DT column, dividing data into 3-month intervals. This method improves performance by organizing large volumes of data into smaller, more manageable chunks. The partitions are created dynamically within the procedure, ensuring that new partitions are automatically added as new data arrives. This strategy optimizes query performance by limiting the amount of data scanned during filtering and joining operations. It also simplifies data maintenance, as historical data can be archived or removed by partition without affecting the entire table. Partitioning helps balance system performance while ensuring that both active and historical data are efficiently stored and accessed.