Schema Overview and Design Choices

Following the exploratory analysis of the source CSV files, including Customers, Orders, OrderItems, Products, and Payments, we developed an OLTP schema focused on the primary business workflows of an e-commerce platform. The resulting schema encompasses the following key entities:

- **CUSTOMER(customer_id, customer_zip_code_prefix, customer_city, customer_state)**
- **ORDER(order_id, customer_id, order_status, order_purchase_timestamp, order_approved_at, order_delivered_timestamp, order_estimated_delivery_date)**
- **PRODUCT(product_id, product_category_name, product_weight_g, product_length_cm, product_height_cm, product_width_cm)**
- **ORDER_ITEM(order_id, product_id, seller_id, price, shipping_charges)**
- **PAYMENT(order_id, payment_sequential, payment_type, payment_installments, payment_value)**
- **SELLER(seller_id)**

The **primary keys (PKs)** and **foreign keys (FKs)** are:

- CUSTOMER:
    - PK: `customer_id`
- ORDER:
    - PK: `order_id`
    - FK: `customer_id` → CUSTOMER.customer_id
- PRODUCT:
    - PK: `product_id`
- ORDER_ITEM:
    - PK: (`order_id`, `product_id`)

    - FK: `order_id` → ORDER.order_id
    - FK: `product_id` → PRODUCT.product_id
    - FK: `seller_id` → SELLER.seller_id
- PAYMENT:
    - PK: (`order_id`, `payment_sequential`)
    - FK: `order_id` → ORDER.order_id
- SELLER:
    - PK: `seller_id`

This design organizes **customers, orders, products, order line items, payments,** and **sellers** into distinct tables. Each table is assigned a specific purpose, reflecting a corresponding real-world concept—identifying who is buying, what they purchase, how payments are made, and who the sellers are. This structured separation serves as the basis for meeting the criteria of Third Normal Form (3NF).

Normalization to Third Normal Form (3NF)

We checked the schema systematically through 1NF, 2NF, and 3NF.

First Normal Form (1NF)

A relation achieves 1NF when all its attributes are atomic and there are no repeating groups present.

- Each column must contain atomic values, such as a single city for each customer, one payment type per payment, and one product per order item.

- Multi-valued or repeating data, such as items included in an order or payments associated with an order, are organized as separate rows in tables like ORDER_ITEM and PAYMENT, instead of being represented as repeated columns within the ORDER table.

Therefore, all tables satisfy 1NF.

Second Normal Form (2NF)

2NF necessitates that a table is already in First Normal Form (1NF) and eliminates any partial dependency, ensuring that non-key attributes are not dependent on only a part of a composite primary key

Only two tables have composite primary keys:

- **ORDER_ITEM(order_id, product_id)**
    - Non-key attributes: `seller_id`, `price`, `shipping_charges`
    - These attributes describe a specific product within a specific order (e.g., the negotiated price for that product in that order), so they depend on the **whole** key (`order_id`, `product_id`) and not on just `order_id` or just `product_id`.
- **PAYMENT(order_id, payment_sequential)**
    - Non-key attributes: `payment_type`, `payment_installments`, `payment_value`
    - These attributes describe a specific payment event for a given order (e.g., the first or second installment), so they depend on the **combination** of (`order_id`, `payment_sequential`).

All other tables have simple (non-composite) primary keys, so partial dependency does not arise there. Hence, the schema satisfies 2NF.

Third Normal Form (3NF)

3NF additionally requires that there be **no transitive dependencies**: non-key attributes must not depend on other non-key attributes.

- **CUSTOMER**
    - Key: `customer_id`
    - Attributes: `customer_zip_code_prefix`, `customer_city`, `customer_state`
    - All attributes represent the characteristics associated with the customer identified by `customer_id`. Within the operational schema, these attributes are considered directly dependent on `customer_id`. This approach prevents the need to store customer demographic details, such as city or state, in other tables

like ORDER, thereby avoiding potential anomalies.

- **ORDER**
    - Key: `order_id`
    - Attributes: `customer_id` (FK), `order_status`, purchase/approval/delivery timestamps, `order_estimated_delivery_date`
    - All non-key attributes represent specific details about the given order. None of these attributes rely on another non-key attribute for their value (for instance, order_status does not define timestamps).
- **PRODUCT**
    - Key: `product_id`
    - Attributes: `product_category_name`, `product_weight_g`, `product_length_cm`, `product_height_cm`, `product_width_cm`
    - These are inherent characteristics of a product, each solely reliant on the product_id. None of these attributes functionally determines any other attribute (for instance, category does not uniquely determine weight or dimensions), which eliminates the possibility of transitive dependency.
- **ORDER_ITEM**
    - Key: (`order_id`, `product_id`)
    - Attributes: `seller_id`, `price`, `shipping_charges`
    - This section outlines the details of the line item, including the seller who fulfilled it and the commercial terms associated with the order. These attributes are independent of any other non-key features. For instance, `seller_id` does not universally define the `price`, as the same seller and product can be listed at varying prices across different orders. The price is determined specifically by the composite key.
- **PAYMENT**
    - Key: (`order_id`, `payment_sequential`)
    - Attributes: `payment_type`, `payment_installments`, `payment_value`

    - This outlines a specific payment record. All attributes are directly tied to the key combination (order + sequential payment number) and are not derived from any other non-key attribute.

- **SELLER**
    - Key: `seller_id`
    - Attributes, whether current or future (like name, city, or state), are exclusively dependent on the seller identifier and are not influenced by other non-key attributes.

In these functional dependencies, all tables comply with the rules of 3NF: every non-key attribute is dependent **on the key, the entire key, and only the key**..

How the Schema Avoids Data Anomalies

By designing the schema in 3NF with clear entity boundaries and foreign key relationships, we reduce **update**, **insert**, and **delete** anomalies.

- **Update anomalies**
    - To maintain data integrity and efficiency, customer details like city and state are centralized in the `CUSTOMER` table, deliberately excluding them from the `ORDERS` table. This design choice means that changes to a customer's location require only a single update to the `CUSTOMER` table, preventing the need to locate and modify multiple rows across all their associated orders..
    - Product attributes such as category and dimensions are exclusively maintained within the `PRODUCT` table. This centralization ensures efficiency: if a category label needs modification, only a single update is required. Subsequent queries involving `ORDER_ITEM` will join with `PRODUCT` to access this most current information.
- **Insert anomalies**
    - It is possible to add a new product to the Product table even if it hasn’t been included in any orders yet. There is no need to create a placeholder or dummy order to register a product in the system.
    - In a similar manner, it is possible to register a new CUSTOMER or SELLER separately, without requiring an order or payment simultaneously.

- **Delete anomalies**
    - Deleting the last ORDER for a CUSTOMER does not remove the CUSTOMER record itself; customer data remains in CUSTOMER. We do not risk “forgetting” a customer’s existence just because they have no active orders.
    - Deleting an ORDER_ITEM row does not delete the underlying PRODUCT or SELLER; those entities are managed in their respective tables.

Overall, the separation of concerns, addressing entities like customers, orders, items, products, payments, and sellers, along with the application of 3NF, ensures that each piece of information is stored in a single, definitive location. This approach reduces redundancy, maintains data consistency, and strengthens the OLTP database against typical issues associated with denormalized schemas.