Data Model:

1.Given Source Files:

Summary:

Tables	Records	Columns	Keys	File Format
Customer Table	250 records	5 Columns	Customer_ID is Primary Key	XlS
Order Table	250 records	4 Columns	Order_ID (Primary Key)	csv
Shipping Table	250 records	3 Columns	Shipping_ID (Primary Key)	JSON

Snapshots of the given data Files

Customer_ID	First	Last	Age	Country
1	Joseph	Rice	43	USA
2	Gary	Moore	71	USA
3	John	Walker	44	UK
4	Eric	Carter	38	UK
5	William	Jackson	58	UAE
6	N!cole	Jones	33	USA
7	David	Davis	59	USA
8	Jason	Montgomery	58	UK
9	Kent	Weaver	61	UK
10	Darrell	Dillon	50	UAE
11	Jacqueline	Wang	22	USA
12	Jodi	Gonzalez	69	USA

Customer_ID	Order_ID	Item	Amount
4	118	Mousepad	$200
5	19	DDR RAM	$1,500
8	109	DDR RAM	$1,500
8	25	Mousepad	$250
8	158	Mousepad	$200
8	117	Webcam	$350
10	144	Keyboard	$400
12	166	Harddisk	$5,000
13	130	Headset	$900
13	234	Keyboard	$400
13	173	Monitor	$12,000
15	17	Webcam	$350

Shipping_ID	Status	Customer_ID
197	Pending	1
25	Delivered	2
91	Pending	3
144	Pending	3
20	Delivered	6
200	Delivered	8
210	Delivered	8
49	Pending	9
104	Pending	9
168	Delivered	9
154	Pending	10

2. Define Table Structures and Relationships

Customer Table

Purpose: Holds personal information about each customer.

Columns:

Customer_ID (Primary Key): Unique identifier for each customer.
First Name: First name of the customer.
Last Name: Last name of the customer.
Age: Age of the customer.
Country: Country where the customer resides.
Customer Name: Full name or formatted name of the customer.

Customer_ID (Primary Key):	Unique identifier for each customer	NotNull Unique		
First Name	First name of the customer			
Last Name	Last name of the customer			
Customer Name	Full Name	Concatinate both Last Name and First Name	`CustomerName = Customer[First]&""&Customer[Last]`	Derived
Country	where the customer resides			
Age	Age of the customer			
AgeBucket	<30 =30 and >30	Using Switch create this AgeBucket	`Agebucket = SWITCH(TRUE,Customer[Age]<30,"<30", Customer[Age]=30,"=30",Customer[Age]>30,">30")`	Derived

Order Table:

Purpose: Stores detailed information about each order placed by customers.

Columns:

Order_ID (Primary Key): Unique identifier for each order.
Item: The name of the product or service ordered.
Amount: The total amount for the order.
Customer_ID (Foreign Key): References Customer_ID from the Customer Table, linking the order to the customer.

Order_ID (Primary Key)	Unique identifier for each order	NotNull Unique
Item	The name of the product or service ordered	
Amount	The total amount for the order	
Customer_ID (Foreign Key)	Foreign Key	References Customer_ID from the Customer Table, linking the order to the customer

Shipping Table

Purpose: Tracks the shipping status of orders.
Columns:

Shipping_ID (Primary Key): Unique identifier for each shipping record.
Status: The current shipping status (e.g., Pending, Shipped, Delivered).
Customer_ID (Foreign Key): References Customer_ID from the Customer Table, inking the shipping status to the customer.

Shipping_ID (Primary Key)	Unique identifier for each order	NotNull Unique
Status	The current shipping status (e.g., Pending, Shipped, Delivered).	
Amount	The total amount for the order	
Customer_ID (Foreign Key)	Foreign Key	References Customer_ID from the Customer Table, linking the shipping status to the customer.

3. Establish Relationships

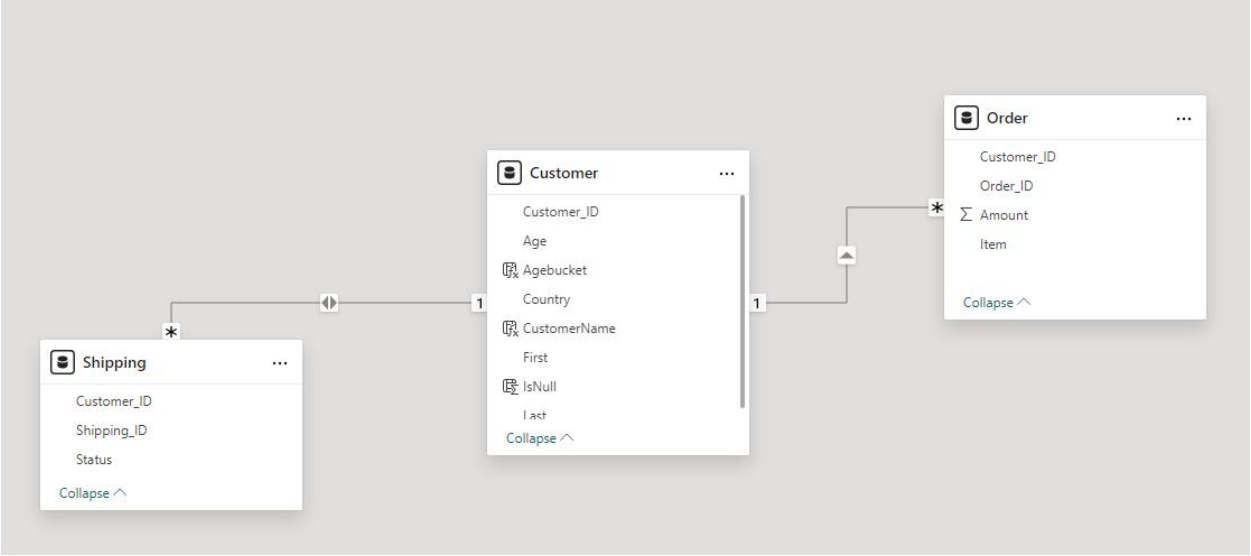
One-to-Many Relationship between Customer Table and Order Table: A single customer can have multiple orders. The Customer_ID field in both tables facilitates this relationship.

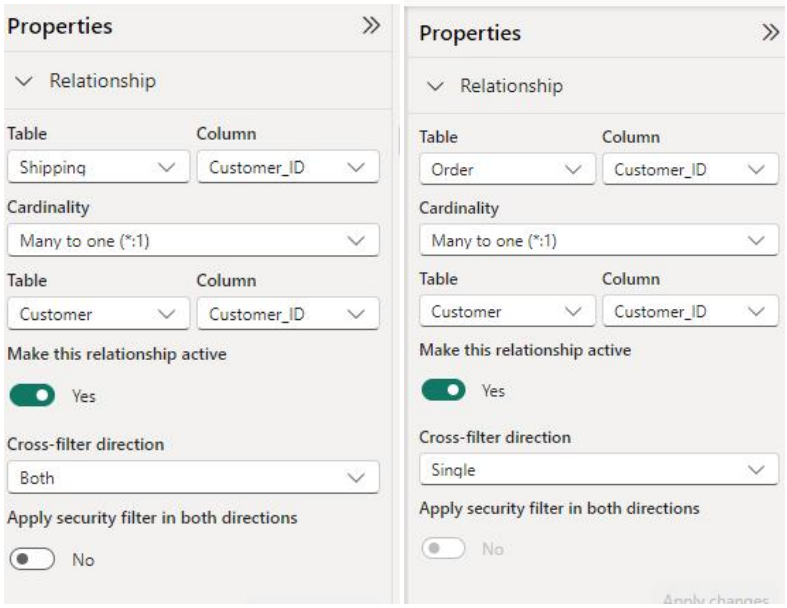
One-to-Many Relationship between Customer Table and Shipping Table: A single customer can have multiple shipping records associated with their orders.

This relationship is facilitated by the Customer_ID field in both tables.

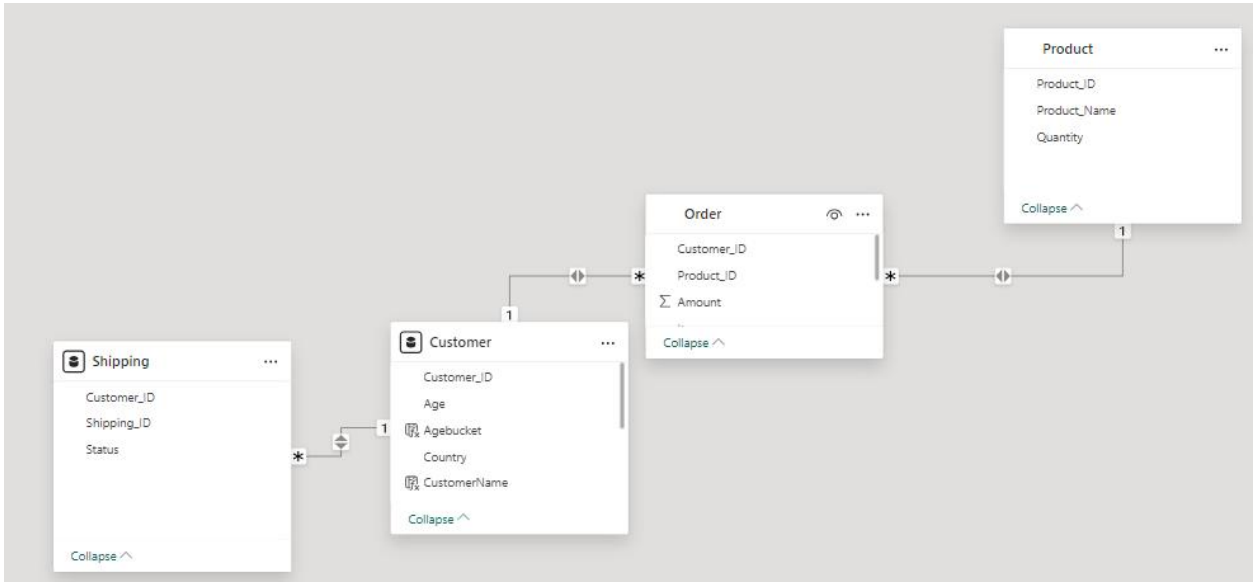
4. Create the Data Model

The data model can be visualized as follows:





Modified New Data Model suggested



Data Model Overview:

1. **Customer Table**:

    - Contains details such as `Customer_ID`, `Age`, `Country`, and `CustomerName`.
    - Primary Key: `Customer_ID`
2. **Order Table**:
    - Records orders with fields `Customer_ID`, `Product_ID`, and `Amount`.
    - Primary Key: Composite key on `Customer_ID` and `Product_ID`
3. **Product Table**:
    - Stores product information like `Product_ID`, `Product_Name`, and `Quantity`.
    - Primary Key: `Product_ID`
4. **Shipping Table**:
    - Tracks shipping status, linked by `Customer_ID` and `Shipping_ID`.
    - Primary Key: `Shipping_ID`

Data Flow Design:

1. **Customer Interaction**:
    - **Customer** places an order, which links to the **Order** table via `Customer_ID`.
    - The customer's demographic information (e.g., age, country) can be used to analyze purchasing patterns or generate targeted offers.
2. **Order Processing**:
    - Once the **Order** is placed, it's linked to both the **Customer** and **Product** tables using `Customer_ID` and `Product_ID`.
    - The **Order** table tracks how much was spent (`Amount`) and what was ordered (linked via `Product_ID`).
3. **Product Information**:
    - Every order references the **Product** table for details like `Product_Name` and `Quantity` available.
    - If a product's stock (`Quantity`) is low, the system could flag it for replenishment.
4. **Shipping Status**:
    - After the order is placed, **Shipping** details are recorded using the `Shipping_ID` linked to the `Customer_ID`.
    - The shipping status is monitored, and customers are notified based on the `Status` field in the **Shipping** table.

Data Flow Diagram:

- **Step 1**: **Customer** places an **Order** → Data flows from the **Customer** table to the **Order** table.
- **Step 2**: The **Order** references the **Product** table for item details → **Order** table fetches product info (e.g., `Product_Name`, `Quantity`).
- **Step 3**: Once the order is confirmed, the **Shipping** details are updated, linking the **Customer** to the shipping process.
- **Step 4**: The **Order** is fulfilled, and the shipping status is monitored and updated.

1. Customer Order Initiation

- **Trigger**: A customer initiates an order on the platform.
- **Data Flow**:
  - The system checks the **Customer** table using the `Customer_ID` to verify the customer's details.
  - The **Customer** data (such as `CustomerName`, `Age`, and `Country`) is retrieved to ensure the customer is legitimate and to personalize the order experience (e.g., special discounts based on age, region-specific offers).
- **Process Checks**:
  - The system ensures that the **Customer_ID** is valid (foreign key constraint).
  - If the **Customer_ID** is not found, the system can prompt the user to register or correct their details.

Here's a more detailed breakdown of the data flow between entities in your model, with additional processes and checks incorporated at each step:

1. Customer Order Initiation

- **Trigger**: A customer initiates an order on the platform.
- **Data Flow**:
  - The system checks the **Customer** table using the `Customer_ID` to verify the customer's details.
  - The **Customer** data (such as `CustomerName`, `Age`, and `Country`) is retrieved to ensure the customer is legitimate and to personalize the order experience (e.g., special discounts based on age, region-specific offers).
- **Process Checks**:
  - The system ensures that the **Customer_ID** is valid (foreign key constraint).
  - If the **Customer_ID** is not found, the system can prompt the user to register or correct their details.

---

2. Product Selection and Order Creation

- **Trigger**: The customer selects a product and places it in their order.
- **Data Flow**:
  - The `Product_ID` is used to query the **Product** table, retrieving the product's details like `Product_Name` and checking stock availability (`Quantity`).
  - If sufficient quantity is available, the system creates an entry in the **Order** table, associating it with the corresponding `Customer_ID` and `Product_ID`. The order amount (`Amount`) is calculated and stored in the **Order** table.
- **Process Checks**:
  - **Product Availability**: The system checks the **Product** table for the `Quantity` field. If stock is below the required amount, the system can flag the order and notify the customer of the insufficient stock.

    - **Order Integrity**: A composite key (Customer_ID + Product_ID) ensures that duplicate orders for the same product are handled correctly.
- **Order Confirmation**:
    - Once validated, the order is confirmed, and the **Order** table is updated with `Customer_ID`, `Product_ID`, and `Amount`.

3. Inventory Management

- **Trigger**: After the order is created, the system must adjust inventory levels.
- **Data Flow**:
    - The **Product** table is updated, and the `Quantity` of the ordered product is decremented by the quantity in the order.
    - This helps keep real-time stock levels for products.
- **Process Checks**:
    - **Stock Recheck**: After the order, the system checks if the updated stock level (`Quantity`) is below a certain threshold, possibly triggering a reordering process to replenish stock.

4. Shipping Process

- **Trigger**: Once the order is confirmed, the shipping process is initiated.
- **Data Flow**:
    - The **Shipping** table is updated with a new entry using the `Customer_ID` and a newly generated `Shipping_ID`.
    - The system assigns a `Status` to the shipment (e.g., "Pending", "Shipped", "Delivered").
- **Process Checks**:
    - **Shipping Validation**: The system checks if the `Customer_ID` exists in the **Customer** table before creating the shipping record.
    - **Order Verification**: The order linked to the shipment is verified by cross-referencing the **Order** table to ensure all products ordered are ready for shipment.

5. Status Updates and Tracking

- **Trigger**: Throughout the shipping process, the status of the shipment is updated.
- **Data Flow**:
    - The `Status` in the **Shipping** table is updated as the order moves through different stages (e.g., "Processing", "In Transit", "Delivered").
    - Customers can query the **Shipping** table using their `Customer_ID` and `Shipping_ID` to track their order status.
- **Process Checks**:
    - **Status Monitoring**: The system ensures that status transitions are sequential (e.g., "Shipped" must come after "Processing").
    - If any errors occur (e.g., failed delivery attempts), the system flags the shipment for manual review and notifies the customer.

6. Post-Delivery and Customer Feedback

- **Trigger**: After delivery, the system may initiate feedback collection or further actions.
- **Data Flow**:
  - Once the **Shipping** status is marked as "Delivered", the system logs this status for future analysis (e.g., delivery time, performance, regional delivery success).
  - The system can trigger a request for customer feedback, sending surveys or post-purchase inquiries based on the `Customer_ID` and associated order.
- **Process Checks**:
  - **Delivery Confirmation**: The system ensures the delivery status is accurate before marking the order complete.
  - **Customer Follow-Up**: If feedback is collected, it could be tied back to the `Customer_ID` for service improvement analytics.

Optional Enhancements for Data Flow

- **Data Quality Assurance**:
  - Implement validation checks to ensure that no data discrepancies (e.g., mismatched IDs, inconsistent statuses) affect the flow of information across tables.
  - Enforce data integrity rules like referential constraints between **Order**, **Product**, **Customer**, and **Shipping** tables.
- **Optimization**:
  - **Batch Processing**: For high-volume transactions (e.g., large sales events), the system can process orders in batches to optimize database performance.
  - **Data Caching**: Frequently accessed data (like product details) can be cached to reduce database load during product lookups.
- **Error Handling**:
  - The system should log any failures in order placement, shipping status updates, or stock updates. These logs could trigger automated recovery processes or escalate to manual interventions.

Mapping Document for Changes Between the Existing Model and the New Model

1. Customer Table

- **Current Fields (Both Models)**:
  - `Customer_ID`: Unique identifier for the customer.
  - `Age`: Customer's age.
  - `Country`: Customer's country.
  - `CustomerName`: Name of the customer.
- **Additional Fields in Existing Model**:

    - `First`: First name of the customer.
    - `Last`: Last name of the customer.
    - `IsNull`: (Purpose unclear, but possibly a placeholder or validation field).
    - **Action**: These fields are removed in the new model. The Data Engineer should drop the `First`, `Last`, and `IsNull` fields.
- **Unchanged Fields**:
    - `Customer_ID`, `Age`, `Country`, and `CustomerName` remain the same in both models.

2. Order Table

- **Existing Model Fields**:
    - `Order_ID`: Unique identifier for each order.
    - `Amount`: Amount spent by the customer in the order.
    - `Item`: Product or service purchased.
- **New Model Changes**:
    - `Order_ID`: Removed in the new model.
    - `Product_ID`: Added in the new model to establish a link to the **Product** table.
    - `Item`: Removed.
    - **Action**:
        - Drop `Order_ID` and `Item` fields.
        - Add `Product_ID` field to link orders to products.
    - **Amount** remains unchanged.

3. Product Table

- **Existing Model**: No **Product** table in the existing model.
- **New Model**:
    - **Product Table** is introduced, containing:
        - `Product_ID`: Unique identifier for products.
        - `Product_Name`: Name of the product.
        - `Quantity`: Stock quantity of the product.
    - **Action**: Data Engineer should create a new **Product** table with these fields, and link it to the **Order** table using `Product_ID`.

4. Shipping Table

- **Current Fields (Both Models)**:
    - `Shipping_ID`: Unique identifier for each shipment.
    - `Customer_ID`: Links the shipping record to the customer.
    - `Status`: Tracks the status of the shipment (e.g., delivered, pending).
- **Unchanged**: No change is required to the **Shipping** table. It remains the same in both models.

5. Relationships and Keys

- **Existing Model**:
    - **Customer → Order**: Linked by `Customer_ID`.

    - **Customer → Shipping**: Linked by `Customer_ID`.
- **New Model**:
    - **Customer → Order → Product**:
        - `Customer_ID` continues to link **Customer** to **Order**.
        - `Product_ID` is added to link **Order** to **Product**.
    - **Customer → Shipping**: Remains unchanged.
- **Action**:
    - Update the **Order** table to have a `Product_ID` foreign key referencing the **Product** table.
    - Ensure existing foreign key relationships for `Customer_ID` between **Customer**, **Order**, and **Shipping** tables remain intact.

Summary of Changes

Table	Field/Action	Type	Description
Customer	`No Changes`		No Changes required
Order	`Product_ID` → **Add**	Attribute Addition	Add `Product_ID` to link **Order** to **Product** table.
Product	**Create Table**: `Product_ID`, `Product_Name`, `Quantity`	New Table	Introduce a new **Product** table.
Relationships	Update relationship **Customer → Order → Product**	Foreign Key	Establish a link between **Order** and **Product** using `Product_ID`.