Data Analysis Assignment

Date: 21 May 2025

Version 1.0

Business Requirement Objective (Summary):

The objective is to enable comprehensive reporting on customer transactions, product sales, and fulfillment status using the available orders, customers, and shipping tables. This includes supporting analysis of total spending by country for pending deliveries, detailed transaction and sales summaries for each customer and product, identifying top-selling products by country and age group, and pinpointing countries with the lowest sales and transaction volumes. All requirements are to be addressed by integrating and leveraging the existing source tables to ensure accurate and actionable business insights.

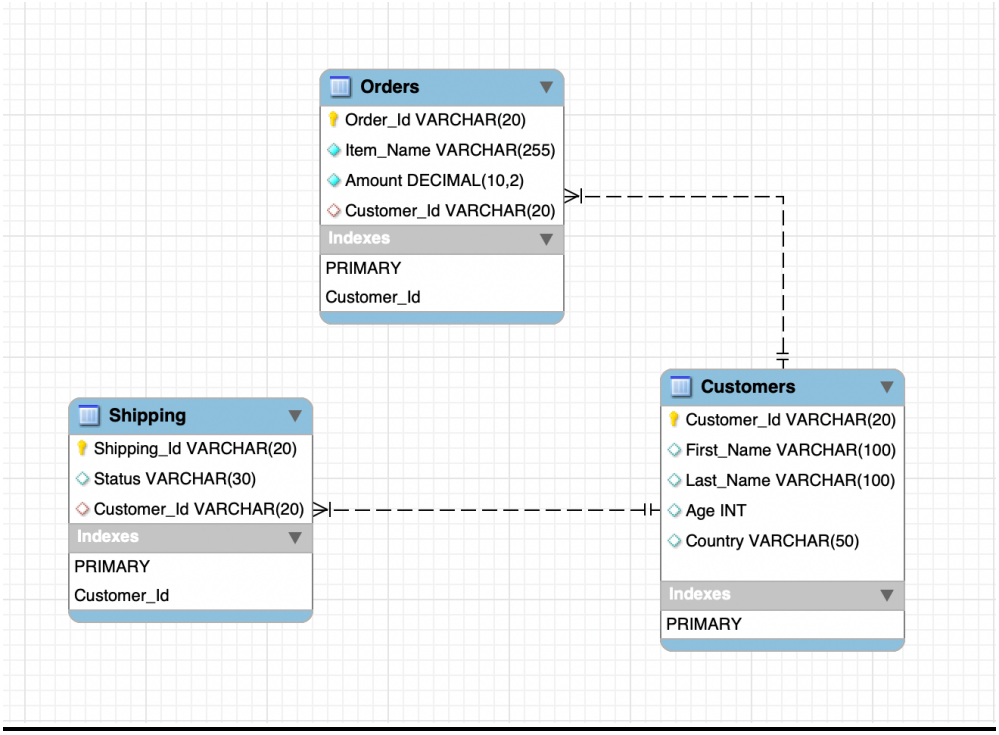
Details of Provided Data

For this analysis, we have been provided with three key datasets: orders, customers, and shipping. Each dataset comes as a structured CSV file containing relevant information to support sales, customer, and logistics analysis.

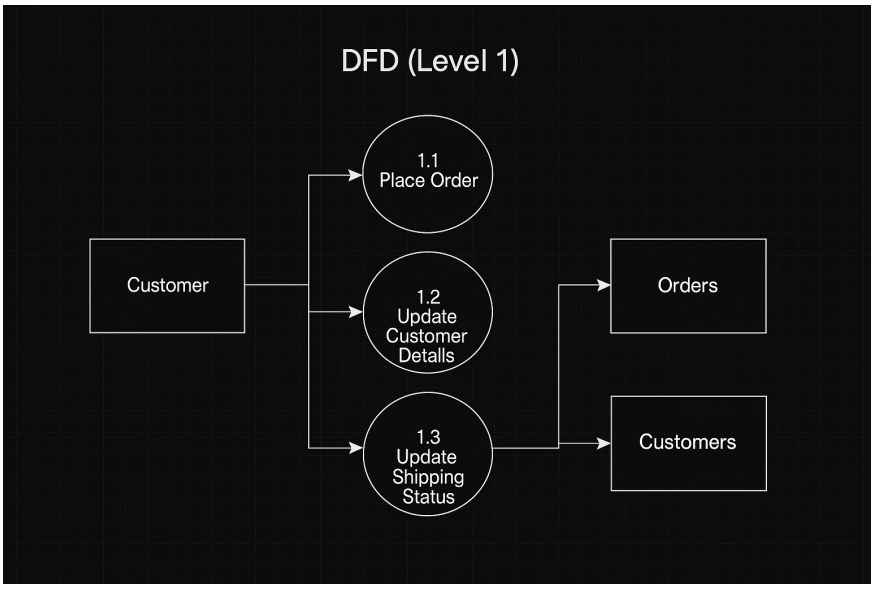
The customers dataset includes unique customer identifiers, names, age, and country, enabling segmentation and demographic analysis. The orders dataset contains order identifiers, customer references, product names, and the amount spent for each transaction, providing the basis for all sales and product performance reporting. The shipping dataset records each shipping event with its status (such as Pending or Delivered) and customer association, allowing for delivery tracking and fulfillment analysis.

Together, these datasets offer a 360-degree view of the customer journey from purchase through to delivery. However, the data does not include a direct link between orders and shipping records, requiring additional logic to associate these for certain reporting needs. The assets have been supplied in CSV format to ensure compatibility with SQL databases and BI tools and have been used to build structured database tables and derived analytical views to support the business requirements.

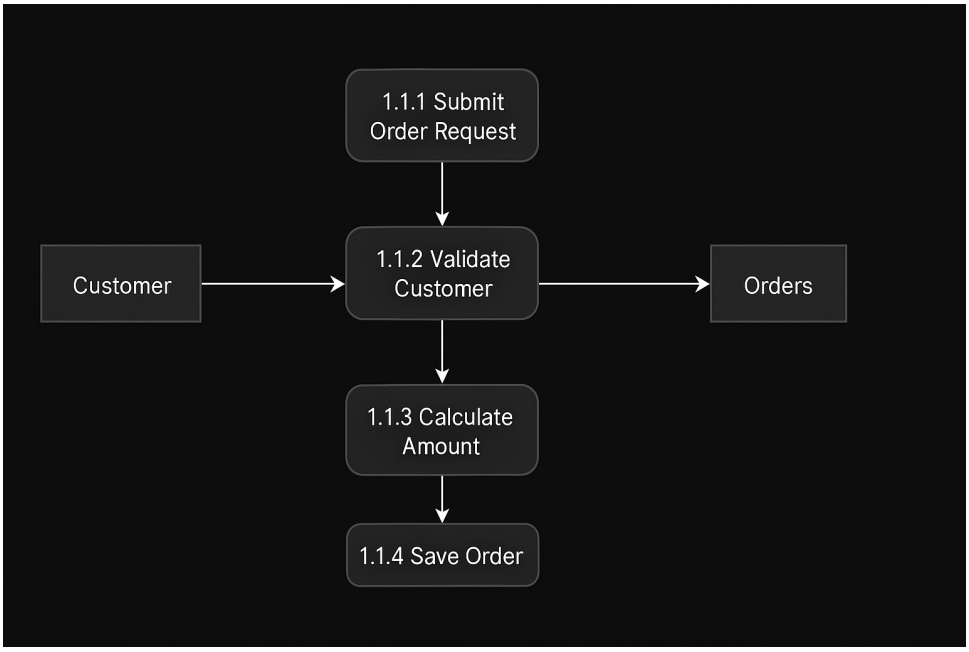
EER Data Model:



Data Flow Diagram: (Level 1)



Data Flow Diagram: (Level 2)



Data Table Design Queries:

```
CREATE TABLE Customers (
  Customer_Id VARCHAR(20) PRIMARY KEY NOT NULL,
  First_Name VARCHAR(100),
  Last_Name VARCHAR(100),
  Age INT,
  Country VARCHAR(50)
);
CREATE TABLE Orders (
    Order_Id VARCHAR(20) PRIMARY KEY NOT NULL,
    Item_Name VARCHAR(255) NOT NULL,
    Amount DECIMAL(10, 2) NOT NULL,
    Customer_Id VARCHAR(20),
    FOREIGN KEY (Customer_Id) REFERENCES Customers(Customer_Id)
);
CREATE TABLE Shipping (
  Shipping_Id VARCHAR(20) PRIMARY KEY NOT NULL,
  Status VARCHAR(30),
  Customer_Id VARCHAR(20),
  FOREIGN KEY (Customer_Id) REFERENCES Customers(Customer_Id)
);
```

Data Accuracy, Completeness, and Reliability of Source Data:

Data Quality Checks Summary:

Check Type	Validation Outcome
1. Customers Table	
Null Check	No Nulls in First_Name, Last_Name, Age, Country, Customer_Id
Value Range	Age criteria verified (Age < 0 OR Age > 120)
Duplicates	No duplicate records found
Foreign Key	All Order Customer_Id exists in Customers Table
Relationship Validation (Cross Table)	Identified 90 Customers without Orders
2. Orders Table	
Null Check	No NULLs in Order_Id, Item_Name, Amount, Customer_Id
Value Range	Amount are positive (Amount < 0)
Duplicates	No Duplicate Order IDs
Foreign Key	Every Order is map to an existing Customer
Relationship Validation (Cross Table)	Identified 94 Orders without corresponding Shipping
3. Shipping Table	
Null Check	No NULLs in Shipping_Id,Status or Customer_Id
Domain Check	Status column hold only 'Pending' or 'Delivered' values
Duplicates	No duplicate Shipping IDs
Foreign Key	Every shipment is map to a valid customer
Relationship Validation (Cross Table)	Identified 96 Customers without Shipping
4. Orders - Shipping (Customer-based joins)	
Orders without matching shipping records	Identified - 94 records
Shipments without corresponding orders	Identified - 98 Records

Summary Metrics	Value (from SQL Output)
Total Customers	250
Total Orders	250
Total Shipping Records	250
Unique Customers	250
Unique Countries	3
Unique Items	8
Customers with Orders Only	61
Customers with Shipping Only	55
Customers with Orders & Shipping Both	99

Key Insights:

- All tables are clean: no nulls, no invalid domains, no bad foreign keys.
- Some customers exist in only one of Orders or Shipping, which may or may not be acceptable depending on business rules. *
- All Id's are unique — no duplication detected.

Important Observation:

- There are no references between Shipping Data and Order Data. Hence even a Junction table cannot be created to map and understand which Shipping Id belongs to which Order Id as One Customer has multiple orders, and one customer has multiple Shipments which is validated by existing Customer_Id column in both tables.

- No Quantity information is available in any of the table

Proposed Domain Model

The way we set up our data model is meant to make sense of customer, sales, and shipping information—even though these data sources didn't always connect perfectly. Because there wasn't a direct way to link orders and shipments, we used careful logic to match them based on what makes sense in real business situations, like matching the most recent order to a pending shipment.

In proposed model, every customer connects to their orders and shipping records. Where there was no natural link, we created a safe way to pair orders and shipping using rules that avoid any confusing or messy connections. We also built special views (like SalesDetails and OrderShippingMatched) on top of the basic tables, which help pull out all the important details needed for reporting—such as what each customer bought, which products sell the most, and which orders are still waiting to be delivered.

Assumptions and Logic:

- To handle missing quantity information required for business requirement, we assume Quantity as 1 for each order
- To manage the missing linkage between Orders and Shipping Table we incorporate below logic as best possible way to enable report and achieve business requirement
    - Assigns the first order (Ascending) to the first shipping (Ascending) for every customer.
    - **Latest order** is associated with **Pending** shipping where applicable.
    - **Oldest order** aligns with **Delivered** where applicable.
    - Only includes customers where you can safely match 1 order to 1 shipping
    - Avoids Cartesian joins or unsafe many-to-many mappings

Anticipated Datasets Details With Technical Specifications:

Depending on the assumptions and possible logic below are 2 best possible Datasets which leverages all the business requirements:

- SalesDetails
- OrderShippingMatched

1. **SalesDetails Analytical View**

Objective:

Create a derived SQL view called SalesDetails to provide an enriched, reporting-ready dataset that combines order information with customer demographics and calculated fields.

SalesDetails is an analytic view that combines order and customer information to provide enriched transaction records for each sale. It includes derived fields such as full customer name, age categories, order sequence, and order value type, enabling detailed customer and product analysis for business intelligence and reporting.

Source Tables

- orders
- customers

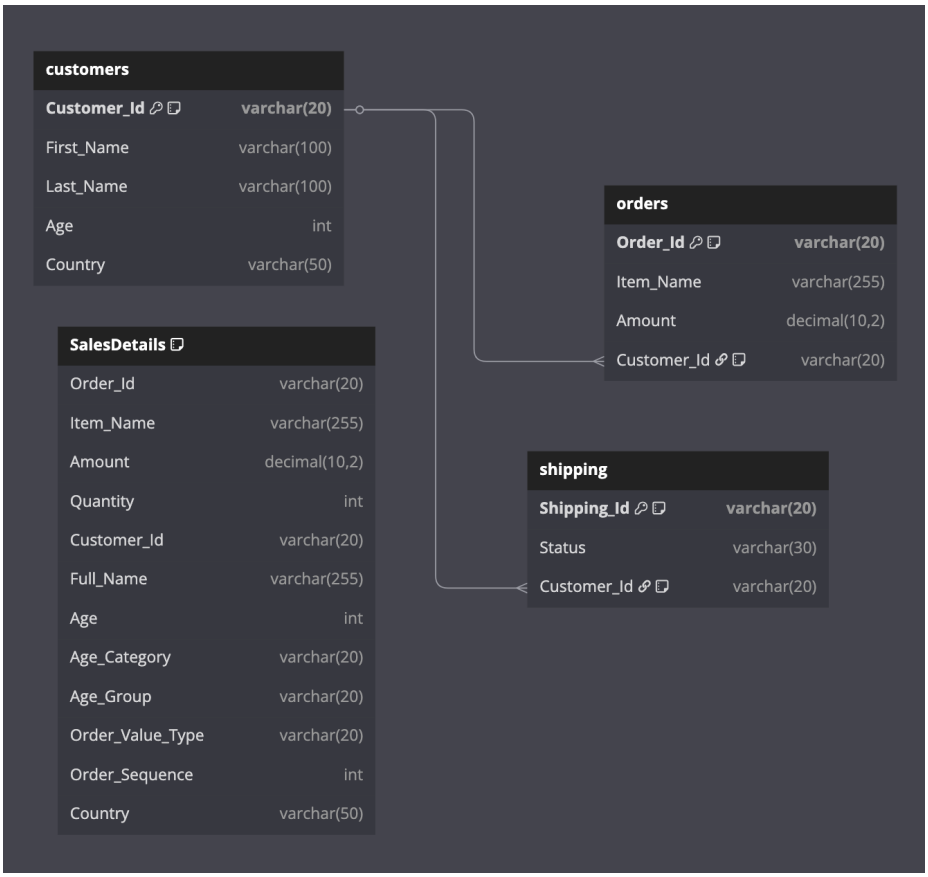
Transformation & Logic

1. **Join Logic:**
    - Join orders to customers using Customer_Id as the foreign key.
2. **Selected & Derived Columns:**
    - Order_Id: Taken from orders.
    - Item_Name: Taken from orders.
    - Amount: Taken from orders.
    - Quantity: Set to 1 by default (as quantity data is not available in the source).
    - Customer_Id: Taken from customers.
    - Full_Name: Concatenate First_Name and Last_Name from customers.
    - Age: Taken from customers.
    - Age_Category:
        - 'Below 30' if Age < 30,
        - '30 and Above' otherwise.
    - Age_Group (Buckets):
        - 'Under 18' if Age < 18
        - '18-29' if Age between 18 and 29
        - '30-49' if Age between 30 and 49
        - '50+' if Age >= 50
    - Order_Value_Type:
        - 'High Value' if Amount >= 500
        - 'Standard Value' otherwise.
    - Order_Sequence:
        - Sequential number of the order per customer (ordered by Order_Id ASC).
    - Country: Taken from customers.

SQL Implementation: SalesDetails:

```
CREATE VIEW SalesDetails AS
SELECT
  o.Order_Id,
  o.Item_Name,
  o.Amount,
  1 AS Quantity,
  c.Customer_Id,
  CONCAT(c.First_Name, ' ', c.Last_Name) AS Full_Name,
  c.Age,
  CASE WHEN c.Age < 30 THEN 'Below 30' ELSE '30 and Above' END AS Age_Category,
  CASE
    WHEN c.Age < 18 THEN 'Under 18'
    WHEN c.Age BETWEEN 18 AND 29 THEN '18-29'
    WHEN c.Age BETWEEN 30 AND 49 THEN '30-49'
    WHEN c.Age >= 50 THEN '50+'
  END AS Age_Group,
  CASE WHEN o.Amount >= 500 THEN 'High Value' ELSE 'Standard Value' END AS Order_Value_Type,
  ROW_NUMBER() OVER (PARTITION BY o.Customer_Id ORDER BY o.Order_Id ASC) AS Order_Sequence,
  c.Country
FROM Orders o
JOIN Customers c ON o.Customer_Id = c.Customer_Id;
```

Data Model SalesDetails:



Testing & Validation Requirements (QA Checklist)

- All columns should be populated as per logic above.
- Full_Name correctly concatenates first and last names.
- Age_Category and Age_Group are assigned based on customer age.

- Order_Value_Type reflects the correct threshold (>= 500).
- Order_Sequence is sequential for each customer and resets per customer.
- No duplicates unless customer has multiple orders.
- Each Order_Id in SalesDetails should exist in the orders table.

Acceptance Criteria

- The view returns all expected columns for every order-customer combination.
- All derived columns (Full_Name, Age_Category, etc.) are correct according to the business logic.
- The view supports downstream aggregation/reporting without additional transformation.
- Performance is reasonable on full dataset (no unnecessary cross joins or cartesian products).

2. **OrderShippingMatched Analytical View:**

Objective:

Create a SQL view named OrderShippingMatched that provides a clean, business-safe link between orders and shipping records, incorporating customer information, for delivery status analysis and reporting.

OrderShippingMatched is a reporting view that safely links orders and shipping records for customers where a strict one-to-one match is possible. It supports delivery-status-driven analysis by associating each order with the appropriate shipping status (delivered or pending), ensuring accurate aggregation of spending and fulfillment across countries and customers.

- **B**ridged orders and shipping by Customer_Id in the absence of a direct Order-to-Shipping link.
- Assigns the first order (ascending Order_Id) to the first shipping (ascending Shipping_Id) for every customer to establish a consistent baseline pairing.
- Associates the latest order (descending Order_Id) with the pending shipping status, ensuring that unfulfilled shipments are matched to the most recent order activity.
- Aligns the oldest order (ascending Order_Id) with the delivered shipping status, so fulfilled deliveries reflect the earliest order.
- Restricts matches to customers who have exactly one order and one shipping record, ensuring only safe, unambiguous 1-to-1 mappings are included.
- Avoids any many-to-many or cartesian joins to maintain data integrity and ensure reporting accuracy.

This approach provides a reliable, business-safe linkage for reporting and analytics where no natural key exists between orders and shipping, and all associations are based on defensible business logic rather than arbitrary or ambiguous matching.

Source Tables

- orders
- shipping
- customers

Transformation & Logic

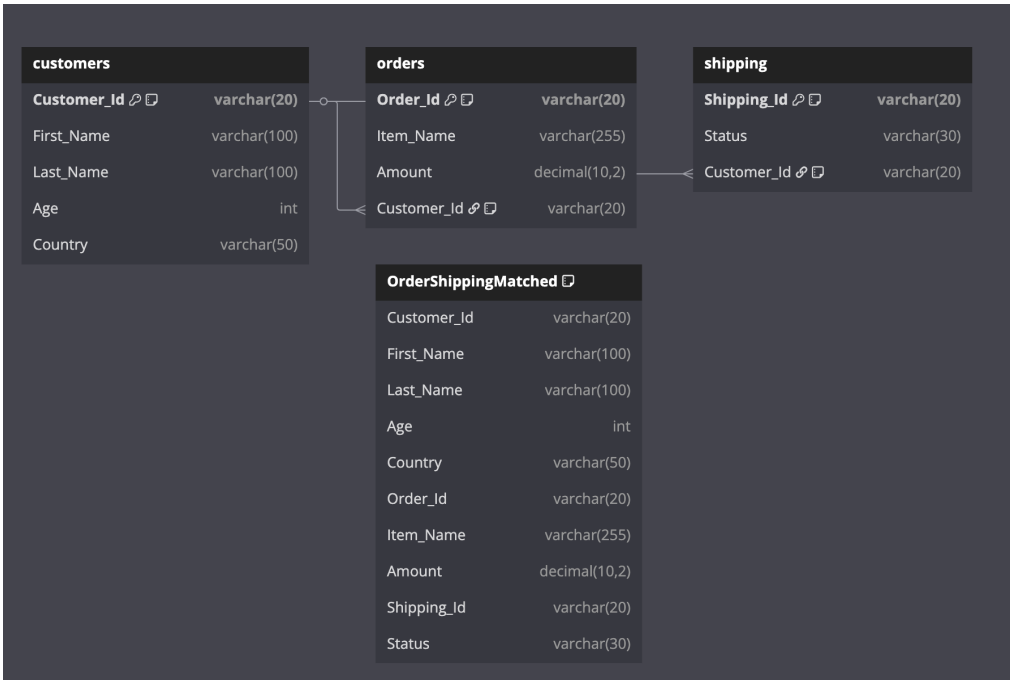
1. **Bridge Logic:**
   - Since there is no direct link between orders and shipping, join them through Customer_Id **only for customers who have exactly one order and one shipping record** (safe 1:1 matches).
   - For customers with more than one order or shipping record, do not include them in the view.
2. **Status-based Matching:**
   - **Delivered:** Assign the earliest order (by ascending Order_Id) to the earliest delivered shipping record (by ascending Shipping_Id).
   - **Pending:** Assign the latest order (by descending Order_Id) to the latest pending shipping record (by descending Shipping_Id).
3. **Selected & Derived Columns:**
   - Customer_Id: Customer reference, as matched above.
   - First_Name, Last_Name, Age, Country: Taken from customers.
   - Order_Id, Item_Name, Amount: Taken from orders.
   - Shipping_Id, Status: Taken from shipping.
   - Any other necessary fields as required by reporting.

SQL Implementation: OrderShippingMatched:

```
CREATE VIEW OrderShippingMatched AS
WITH order_ranked AS (
  SELECT
    Order_Id, Customer_Id, Item_Name, Amount,
    ROW_NUMBER() OVER (PARTITION BY Customer_Id ORDER BY Order_Id ASC) AS rn_asc,
    ROW_NUMBER() OVER (PARTITION BY Customer_Id ORDER BY Order_Id DESC) AS rn_desc
  FROM Orders
),
shipping_ranked AS (
  SELECT
    Shipping_Id, Customer_Id, Status,
    ROW_NUMBER() OVER (
      PARTITION BY Customer_Id, Status
      ORDER BY
        CASE WHEN Status = 'Delivered' THEN Shipping_Id END ASC,
        CASE WHEN Status = 'Pending' THEN Shipping_Id END DESC
    ) AS rn_by_status,
    ROW_NUMBER() OVER (PARTITION BY Customer_Id ORDER BY Shipping_Id ASC) AS rn_asc
  FROM Shipping
),
orders_1 AS (
  SELECT Customer_Id
  FROM Orders
  GROUP BY Customer_Id
  HAVING COUNT(*) = 1
),
shipping_1 AS (
  SELECT Customer_Id
  FROM Shipping
  GROUP BY Customer_Id
  HAVING COUNT(*) = 1
),
customers_with_1_each AS (
  SELECT o.Customer_Id
  FROM orders_1 o
  INNER JOIN shipping_1 s ON o.Customer_Id = s.Customer_Id
),
delivered_match AS (
```

```
  SELECT
    o.Order_Id, o.Item_Name, o.Amount, o.Customer_Id,
    s.Shipping_Id, s.Status
  FROM order_ranked o
  JOIN shipping_ranked s
    ON o.Customer_Id = s.Customer_Id
   AND o.rn_asc = 1
   AND s.Status = 'Delivered' AND s.rn_by_status = 1
   AND o.Customer_Id IN (SELECT Customer_Id FROM customers_with_1_each)
),
pending_match AS (
  SELECT
    o.Order_Id, o.Item_Name, o.Amount, o.Customer_Id,
    s.Shipping_Id, s.Status
  FROM order_ranked o
  JOIN shipping_ranked s
    ON o.Customer_Id = s.Customer_Id
   AND o.rn_desc = 1
   AND s.Status = 'Pending' AND s.rn_by_status = 1
   AND o.Customer_Id IN (SELECT Customer_Id FROM customers_with_1_each)
),
combined_matches AS (
  SELECT * FROM delivered_match
  UNION
  SELECT * FROM pending_match
)
SELECT
  cm.Customer_Id,
  c.Country,
  cm.Order_Id,
  cm.Item_Name,
  cm.Amount,
  1 AS Quantity,
  cm.Shipping_Id,
  cm.Status
FROM combined_matches cm
JOIN Customers c ON cm.Customer_Id = c.Customer_Id;
```

Data Model OrderShippingMatched:



Testing & Validation Requirements (QA Checklist)

- Only customers with exactly one order and one shipping record appear in the view.
- For 'Delivered' status: The oldest order is matched with the earliest delivered shipping.
- For 'Pending' status: The latest order is matched with the latest pending shipping.
- Customer details match correctly from the customers table.
- Each order and shipping ID in the view must exist in the respective base tables.
- Quantity is set to 1 for every row.
- No customer is duplicated for multiple order-shipping pairings.

Acceptance Criteria

- The view returns one row per safe order-shipping match for eligible customers.
- All columns are populated correctly according to the logic above.
- The view supports reporting of pending delivery amounts by country and other fulfillment analytics without further transformation.
- No ambiguous, cartesian, or many-to-many matches are present.

Business Reporting Requirements:

1. the total amount spent and the country for the Pending delivery status for each country.

```
SELECT
  Country,
  SUM(Amount) AS Total_Amount_Spent
FROM
  OrderShippingMatched
WHERE
  Status = 'Pending'
GROUP BY
  Country;
```

2. The maximum product purchased for each country

```
WITH CountryProductSales AS (
  SELECT
    Country,
    Item_Name,
    SUM(Quantity) AS Total_Quantity
  FROM SalesDetails
  GROUP BY Country, Item_Name
)
SELECT
  cps.Country,
  cps.Item_Name,
  cps.Total_Quantity
FROM CountryProductSales cps
JOIN (
    SELECT
      Country,
      MAX(Total_Quantity) AS Max_Quantity
    FROM CountryProductSales
    GROUP BY Country
) maxprod
```

```
ON cps.Country = maxprod.Country
AND cps.Total_Quantity = maxprod.Max_Quantity;
```

3. the total number of transactions, total quantity sold, and total amount spent for each customer, along with the product details.

```
SELECT
Customer_Id,
  Full_Name,
  Country,
  Item_Name,
  COUNT(Order_Id) AS Total_Transactions,
  SUM(Quantity) AS Total_Quantity_Sold,
  SUM(Amount) AS Total_Amount_Spent
FROM SalesDetails
GROUP BY Customer_Id, Full_Name, Country, Item_Name
ORDER BY Customer_Id, Item_Name;
```

4. The most purchased product based on the age category less than 30 and above 30.

```
WITH AgeCategoryProductSales AS (
  SELECT
    Age_Category,
    Item_Name,
    SUM(Quantity) AS Total_Quantity
  FROM SalesDetails
  GROUP BY Age_Category, Item_Name
)
SELECT
  acps.Age_Category,
  acps.Item_Name,
  acps.Total_Quantity
FROM AgeCategoryProductSales acps
JOIN (
    SELECT
      Age_Category,
      MAX(Total_Quantity) AS Max_Quantity
    FROM AgeCategoryProductSales
    GROUP BY Age_Category
) maxcat
ON acps.Age_Category = maxcat.Age_Category
AND acps.Total_Quantity = maxcat.Max_Quantity;
```

5. The country that had minimum transactions and sales amount.

```
-- Minimum Transactions
SELECT
  Country,
  COUNT(Order_Id) AS Total_Transactions
FROM SalesDetails
GROUP BY Country
ORDER BY Total_Transactions ASC
LIMIT 1;

-- Minimum Sales Amount
SELECT
  Country,
  SUM(Amount) AS Total_Sales_Amount
FROM SalesDetails
GROUP BY Country
ORDER BY Total_Sales_Amount ASC
LIMIT 1;
```

Implementation Steps:

- Converted all data files to CSV format for easy import and processing.
- Standardized and updated column names and aliases across all datasets for consistency.
- Created the data model and Data Flow Diagram (DFD) to visualize table structures and relationships.
- Designed and built the database tables (orders, customers, and shipping), carefully defining data types, primary keys, and foreign keys to enforce data integrity.
- Imported data into the created tables, ensuring completeness and accuracy.
- Audited the tables to identify missing data points and analyze relationships between the datasets.
- Performed data quality checks using SQL queries to uncover insights and validate the reliability of the data.
- Planned a unified dataset approach to fulfill key business requirements.
- Developed queries and logic to address the missing link between the shipping and orders tables.
- Created 2 Analytical Datasets as Views to address business requirements
- Created technical specifications, QA Checklist and UAT criteria's
- Created SQL queries for each of business requirements using analytical datasets.