

# Sales Data-Analysis

## Data Verification

- Three distinct data sources—Order (CSV), Shipping (JSON), and Customers (Excel)—have been provided for analysis.
- Data normalization and verification were conducted using Python to ensure consistency across formats thus enabling streamlined and accurate analytical processing.
- Below Python commands were utilized to structure and prepare it for further business intelligence activities.

```
>>> import pandas as pd
>>> cust = pd.read_excel("C:/Users/anm/Downloads/Customer.xls",engine='xlrd')
>>> cust.to_csv("C:/Users/anm/Downloads/Customer.csv", index=False)
>>> order=pd.read_csv("C:/Users/anm/Downloads/Order.csv")
>>> Shipping=pd.read_json("C:/Users/anm/Downloads/Shipping.json")
>>> Shipping.to_csv("C:/Users/anm/Downloads/Shipping.csv")
>>>
```

- Next, we check for Missing Values in all 3 data sources

```
print(order.isnull().sum())
Order_ID      0
Item          0
Amount        0
Customer_ID   0
dtype: int64
print(Shipping.isnull().sum())
Shipping_ID    0
Status         0
Customer_ID    0
dtype: int64
print(cust.isnull().sum())
Customer_ID    0
First          0
Last           0
Age            0
Country        0
dtype: int64
```

- Duplicate records were proactively identified and removed, where present, to eliminate data redundancy and enhance the overall quality

```
print("Orders:", order.duplicated().sum())
Orders: 0
print("Shipping", Shipping.duplicated().sum())
Shipping 0
print("Customers:", cust.duplicated().sum())
Customers: 0
```

- Data types for each column are verified

```
print("Orders data types:\n", order.dtypes)
Orders data types:
Order_ID      int64
Item          object
Amount        int64
Customer_ID   int64
dtype: object
print("Customers data types:\n", cust.dtypes)
Customers data types:
Customer_ID   int64
First         object
Last          object
Age           int64
Country       object
dtype: object
print("Shipping data types:\n", Shipping.dtypes)
Shipping data types:
Shipping_ID   int64
Status        object
Customer_ID   int64
dtype: object
```

- Each data source should have a Primary Key column which is Non-Null and Unique, next step is to ensure this constraint is met in all 3 data sources. We compare the number of Unique and non-null values with the complete row count in each.

```
print("Unique Values in Orders:\n",order.nunique().sort_values(ascending=False))

Unique Values in Orders:
Order_ID      250
Customer_ID    160
Amount         9
Item           8
dtype: int64
print("Orders Total Row Count:", len(order))

Orders Total Row Count: 250
print("Unique Values in Shipping:\n",Shipping.nunique().sort_values(ascending=False))

Unique Values in Shipping:
Shipping_ID    250
Customer_ID    154
Status         2
dtype: int64
print("Shipping Total Row Count:", len(Shipping))

Shipping Total Row Count: 250
print("Unique Values in Customers:\n",cust.nunique().sort_values(ascending=False))

Unique Values in Customers:
Customer_ID    250
Last           189
First          171
Age            62
Country        3
dtype: int64
print("Customers Total Row Count:", len(cust))

Customers Total Row Count: 250
```

- Hence, by observation **Order\_ID** from Orders, **Shipping\_ID** from Shipping and **Customer\_ID** from Customers data can be used as **Primary Keys**.
- Next, we check the referential Integrity Constraint, to ensure there is no Customer ID in Orders whose details are not present in Customers data and similarly in Shipping data

```
print("Invalid Customers:",Shipping[~Shipping['Customer_ID'].isin(cust['Customer_ID'])])

Invalid Customers: Empty DataFrame
Columns: [Shipping_ID, Status, Customer_ID]
Index: []
print("Invalid Customers:",order[~order['Customer_ID'].isin(cust['Customer_ID'])])

Invalid Customers: Empty DataFrame
Columns: [Order_ID, Item, Amount, Customer_ID]
Index: []
```

- Above output indicates that Customers table has information about all customers who have placed an order or whose order has been shipped
- Upon validation, it was observed that certain customers listed in the Orders dataset do not have corresponding Shipping details, and vice versa, indicating incomplete data linkage across sources.

```
print("Invalid Customers:",Shipping[~Shipping['Customer_ID'].isin(order['Customer_ID'])])

Invalid Customers:
   Shipping_ID  Status  Customer_ID
0            1  Pending          173
1            2  Pending          155
4            5  Delivered          72
8            9  Pending          199
12           13  Delivered          141
..          ...
238          239  Pending          173
240          241  Pending           22
243          244  Pending           21
244          245  Delivered          199
249          250  Delivered           27

[98 rows x 3 columns]
print("Invalid Customers:",order[~order['Customer_ID'].isin(Shipping['Customer_ID'])])

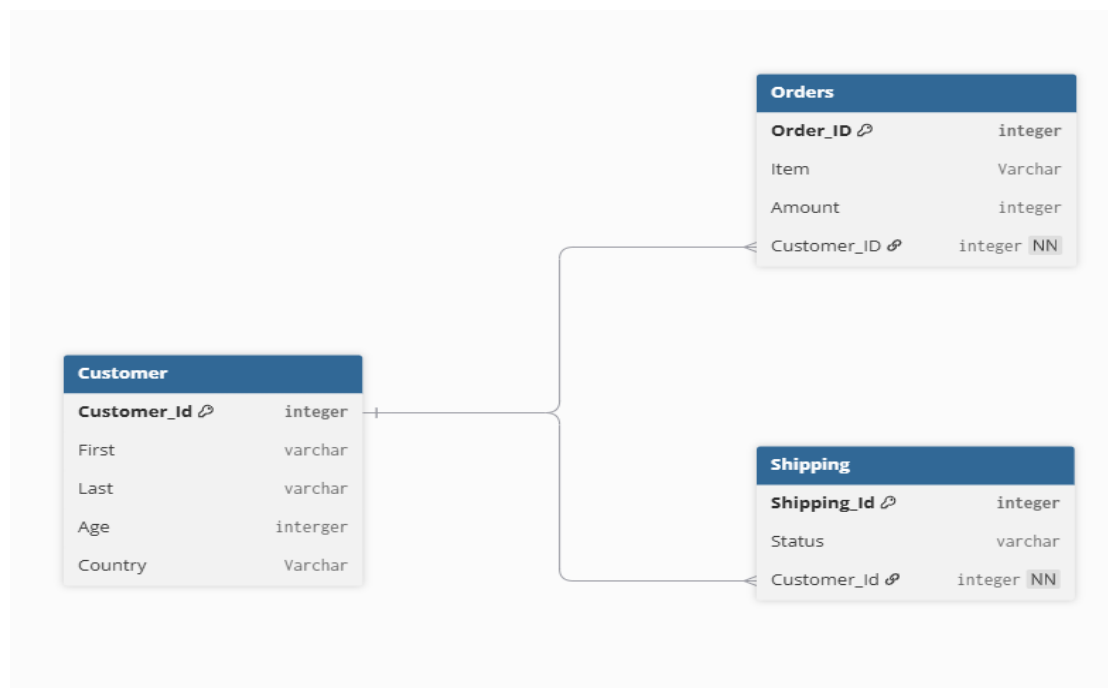
Invalid Customers:
   Order_ID  Item  Amount  Customer_ID
0          1  Keyboard    400          139
1          2   Mouse    300          250
2          3  Monitor  12000          239
3          4  Keyboard    400          153
4          5  Mousepad    250          153
..        ...
230        231  Keyboard    400          162
232        233  Monitor  12000          206
237        238  Mousepad    200          200
242        243  Monitor  12000          123
248        249   DDR RAM   1500          176

[94 rows x 4 columns]
```

Conclusions: The 3 datasets have non-null values with no duplicates and **Order\_ID** from Orders, **Shipping\_ID** from Shipping and **Customer\_ID** from Customers data can be used as **Primary Keys**. With **Customer\_ID** being the foreign key in Orders and Shipping data. The datatype of each column is known and also we noticed that Orders and Shipping data cannot be a complete join.

## ER Diagram

The below represents an Entity-Relation Diagram which serves as a reference for the Data Engineer to build the tables.



### Customers:

- The Customer\_ID is the primary key.
- A customer can have multiple orders and shipping records, but each record in Orders and Shipping refers to exactly one customer (1:N relationship).

### Orders:

- The Order\_ID is the primary key.
- Each order is associated with a Customer\_ID (foreign key from Customers).
- A customer can place multiple orders, but an order is associated with exactly one customer.

### Shipping:

- The Shipping\_ID is the primary key.
- Each shipping record is associated with a Customer\_ID (foreign key from Customers).
- A customer can have multiple shipping records, but a shipping record is associated with exactly one customer.

- As depicted in the diagram, the Customer\_ID fields in both the Orders and Shipping dataset's function acts as foreign key columns referencing the Customers dataset. These fields are expected to be non-null, to maintain referential integrity.
- Additionally, Quality Assurance Engineers .

## Functional Document for DE and QA

Create a summary table that aggregates customer order details with the shipping status for each customer. This would help understand the customer order behaviour, including the number of items ordered, item category and whether the order has shipped. The summary table will consolidate data from the CUSTOMERS\_TEST, ORDERS\_TEST SHIPPING\_TEST tables. The QA Engineer is needed to verify the accuracy of the data, ensuring that the referential integrity is met.

### Data Sources:

- Customer Table: Customers(Excel)
- Orders Table: Order(CSV)
- Shipping Table: Shipping(json)

### Target Tables:

- CUSTOMERS\_TEST
- SHIPPING\_TEST
- ORDERS\_TEST
- Aggregate Table: Customers\_Aggregate

### Table Schema:

Column Name	Data Type	Description	Notes / Transformations
Customer_id	INTEGER	Unique customer identifier	Direct from CUSTOMERS_TEST.customer_id
Customer_Name	VARCHAR(60)	Customer's full name	Concatenate CUSTOMERS_TEST.First + CUSTOMERS_TEST.Last
Total_Spent	INTEGER	Total amount spent by the customer	Sum of ORDERS_TEST.Amount for each Customer
Order_Count	INTEGER	Total Number of orders placed by the customer	Count of distinct ORDERS_TEST.Order_id for each Customer
Shipping_Status	VARCHAR(30)	Status of the customer's latest order	Latest SHIPPING_TEST.Status for each customer_id

### Data Transformations:

1. Join the CUSTOMERS\_TEST table with ORDERS\_TEST the table to get customer details (first name, last name) and the order amounts.
2. Aggregate the total amount spent (total\_spent) by summing up the amount field in the ORDERS\_TEST table for each customer\_id.
3. Count the number of orders (order\_count) by counting distinct order\_ids in the ORDERS\_TEST table.
4. Determine the shipping status (shipping\_status) for each customer by joining the SHIPPING\_TEST table.
5. **Concatenate FIRST\_NAME and LAST\_NAME** from the CUSTOMERS\_TEST table to create a full name for each customer.

Data Checks for QA:

- Customer\_ID column should be unique and non-null
- All Customer\_IDs in Shipping and Orders should be present in Customers table
- Order\_ID and Shipping\_ID column should be non-null
- Total\_Spent should be a positive value
- Ensure that these columns adhere to the defined data types during testing to uphold data consistency

## Part-2

### Business Reporting Queries

To execute the Queries 3 tables : CUSTOMERS\_TEST, ORDERS\_TEST, SHIPPING\_TEST is created as below using SQL

```
CREATE OR REPLACE TABLE EXA_DM.CUSTOMERS_TEST (  
  CUSTOMER_ID VARCHAR(30) PRIMARY KEY ,  
  FIRST_NAME VARCHAR(30),  
  LAST_NAME VARCHAR(30),  
  AGE INTEGER,  
  COUNTRY VARCHAR(100)  
);  
  
IMPORT INTO EXA_DM.CUSTOMERS_TEST  
FROM LOCAL CSV FILE 'C:/Users/anm/Downloads/Customer.csv' skip=1;  
  
CREATE OR REPLACE TABLE EXA_DM.ORDERS_TEST (  
  ORDER_ID INTEGER PRIMARY KEY ,  
  ITEM VARCHAR(30),  
  AMOUNT INTEGER,  
  CUSTOMER_ID INTEGER);  
  
IMPORT INTO EXA_DM.ORDERS_TEST  
FROM LOCAL CSV FILE 'C:/Users/anm/Downloads/Order.csv' skip=1;  
  
CREATE OR REPLACE TABLE EXA_DM.SHIPPING_TEST (  
  SHIPPING_ID INTEGER PRIMARY KEY ,  
  STATUS VARCHAR(30),  
  CUSTOMER_ID INTEGER);  
  
IMPORT INTO EXA_DM.SHIPPING_TEST  
FROM LOCAL CSV FILE 'C:/Users/anm/Downloads/Shipping.csv'  
SKIP= 1;
```

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

```
SELECT DISTINCT COUNTRY,SUM(AMOUNT) FROM EXA_DM.CUSTOMERS_TEST C JOIN EXA_DM.ORDERS_TEST O ON  
C.CUSTOMER_ID=O.CUSTOMER_ID JOIN EXA_DM.SHIPPING_TEST S ON C.CUSTOMER_ID=S.CUSTOMER_ID WHERE S.STATUS='Pending' GROUP BY 1;
```

DISTINCT COUNTRY,SUM(AMOUNT); FROM EXA_DM.CUSTOMERS_TEST		Enter a SQL expression to filter results (use Ctrl+Space)
A-Z COUNTRY	123 SUM(O.AMOUNT)	
UK	136,300	
USA	65,500	
UAE	53,800	

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

```

SELECT DISTINCT FIRST_NAME, LAST_NAME, ITEM, COUNT(DISTINCT ORDER_ID) AS TRANSACTIONS, COUNT(DISTINCT SHIPPING_ID) AS QUANTITY, SUM(AMOUNT) AS AMOUNT
FROM EXA_DM.CUSTOMERS_TEST C JOIN EXA_DM.ORDERS_TEST O ON C.CUSTOMER_ID=O.CUSTOMER_ID JOIN EXA_DM.SHIPPING_TEST S ON C.CUSTOMER_ID=S.CUSTOMER_ID
GROUP BY 1,2,3 ORDER BY 3 DESC;

```

A-Z FIRST_NAME	A-Z LAST_NAME	A-Z ITEM	123 TRANSACTIONS	123 QUANTITY	123 AMOUNT
Mark	R0berts	Monitor	1	3	36,000
R0bert	Shepherd	Monitor	1	2	24,000
Stacey	Welch	Monitor	1	2	24,000
Zachary	Williams	Monitor	1	2	24,000
Yesenia	White	Monitor	1	2	24,000
Amber	Banks	Monitor	1	2	24,000
Michael	Mann	Harddisk	1	3	15,000
Michele	Maxwell	Harddisk	1	3	15,000
Omar	Martin	Monitor	1	1	12,000
Erin	Taylor	Monitor	1	1	12,000
Steve	Braun	Monitor	1	1	12,000
Margaret	Hardy	Monitor	1	1	12,000
Miranda	Williams	Monitor	1	1	12,000
Philip	Newton	Monitor	1	1	12,000
Javier	Jones	Monitor	1	1	12,000
Christy	Rodriguez	Monitor	1	1	12,000
Regina	Wong	Monitor	1	1	12,000
Jodi	Gonzalez	Harddisk	1	2	10,000
Carolyn	Green	Harddisk	1	2	10,000
Amber	Banks	Harddisk	1	2	10,000
Michael	Williams	Harddisk	1	2	10,000
Michelle	Edwards	Harddisk	1	1	5,000
Adrian	West	Harddisk	1	1	5,000
Jose	Poole	Harddisk	1	1	5,000
Jessica	Welch	Harddisk	1	1	5,000
Xavier	Miles	Harddisk	1	1	5,000
Robin	Snyder	Harddisk	1	1	5,000
Taylor	Reed	Harddisk	1	1	5,000
Sandra	Mcmahon	Harddisk	1	1	5,000
Andrea	Velasquez	Harddisk	1	1	5,000
Jonathan	Middleton	DDR RAM	1	3	4,500

3. the maximum product purchased for each country.

```

SELECT
  COUNTRY, ITEM, PURCHASES
FROM
(
  SELECT
    DISTINCT COUNTRY,
    ITEM,
    COUNT(DISTINCT ORDER_ID) AS PURCHASES,
    MAX(COUNT(DISTINCT ORDER_ID)) OVER(PARTITION BY COUNTRY) AS MAX_PURCHASE
  FROM
    EXA_DM.CUSTOMERS_TEST C
  JOIN EXA_DM.ORDERS_TEST O ON
    C.CUSTOMER_ID = O.CUSTOMER_ID
  JOIN EXA_DM.SHIPPING_TEST S ON
    C.CUSTOMER_ID = S.CUSTOMER_ID
  GROUP BY
    1,
    2
)
WHERE PURCHASES=MAX_PURCHASE
;

```

A-Z COUNTRY	A-Z ITEM	123 PURCHASES
UK	Keyboard	13
USA	Mousepad	10
UAE	Keyboard	11
UK	Mousepad	13

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

```

SELECT
  ITEM,AGE_CATEGORY,PURCHASES
FROM
(
  SELECT
    DISTINCT ITEM ,
    CASE WHEN AGE<=30 THEN 'LESS THAN 30'
    WHEN AGE>=30 THEN 'GREATER THAN 30' END AS AGE_CATEGORY,
    COUNT(DISTINCT ORDER_ID) AS PURCHASES,
    MAX(COUNT(DISTINCT ORDER_ID))OVER(PARTITION BY ITEM,CASE WHEN AGE<=30 THEN 'LESS THAN 30'
    WHEN AGE>=30 THEN 'GREATER THAN 30' END) AS MAX_PURCHASE
  FROM
    EXA_DM.CUSTOMERS_TEST C
  JOIN EXA_DM.ORDERS_TEST O ON
    C.CUSTOMER_ID = O.CUSTOMER_ID
  JOIN EXA_DM.SHIPPING_TEST S ON
    C.CUSTOMER_ID = S.CUSTOMER_ID
  GROUP BY
    1,2)
WHERE PURCHASES=MAX_PURCHASE
;

```

ITEM	AGE_CATEGORY	PURCHASES
Webcam	GREATER THAN 30	13
Mousepad	GREATER THAN 30	20
Headset	LESS THAN 30	5
Harddisk	LESS THAN 30	2
Headset	GREATER THAN 30	12
Monitor	GREATER THAN 30	9
Webcam	LESS THAN 30	4
Mouse	GREATER THAN 30	10
Mouse	LESS THAN 30	5
DDR RAM	LESS THAN 30	5
DDR RAM	GREATER THAN 30	12
Keyboard	LESS THAN 30	14
Mousepad	LESS THAN 30	9
Keyboard	GREATER THAN 30	17
Monitor	LESS THAN 30	6
Harddisk	GREATER THAN 30	13

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

```

SELECT
  COUNTRY,PURCHASES,AMT
FROM
(
  SELECT ROW_NUMBER()OVER (ORDER BY SUM(AMOUNT) ) AS RN,
    COUNTRY ,
    COUNT(DISTINCT ORDER_ID) AS PURCHASES,
    MIN(COUNT(DISTINCT ORDER_ID))OVER(PARTITION BY COUNTRY) AS MIN_PURCHASE,
    SUM(AMOUNT) AS AMT
  FROM
    EXA_DM.CUSTOMERS_TEST C
  JOIN EXA_DM.ORDERS_TEST O ON
    C.CUSTOMER_ID = O.CUSTOMER_ID
  JOIN EXA_DM.SHIPPING_TEST S ON
    C.CUSTOMER_ID = S.CUSTOMER_ID
  GROUP BY
    2
  ORDER BY AMT ASC)
WHERE PURCHASES=MIN_PURCHASE AND RN=1
;

```

COUNTRY	PURCHASES	AMT
UAE	30	67,400