SQL Documentation

Data Cleaning (01_clean.sql)

Purpose: Clean null values from the Online Retail dataset

Output: A cleaned table called OnlineRetail_Clean

The purpose of this script is to prepare the Online Retail transactional dataset for downstream analysis and modelling. It ensures that the data is reliable, standardised, and representative of actual sales activity.

Step 0: Create Backup of Original Data

A full copy of the raw dataset is created to preserve the original state and provide a recovery point if needed.

Step 1: Identify and Remove Duplicate Records

Duplicate rows are identified and removed. These likely stem from data entry errors.

Step 2: Investigate Missing Values in Key Columns

Essential fields such as InvoiceNo, CustomerID, and Description are evaluated for null values. These columns are necessary for customer tracking, product identification, and sales aggregation. Null columns will be removed.

Step 3: Fill Missing Descriptions

For rows where the product description is missing, the most frequent description associated with the corresponding StockCode is used. This prevents unnecessary loss of otherwise usable records.

Step 4: Remove Remaining Nulls

After attempting to fill in missing data, any remaining rows with nulls in critical fields are removed.

Step 5: Save Intermediate Cleaned Table

The dataset at this stage contains no nulls or duplicates and is stored for further cleaning.

Step 6: Validate Column Format and Consistency

Columns such as InvoiceNo, StockCode, and CustomerID are checked for structural consistency (e.g. length, formatting). Outliers and potential anomalies (e.g. postage charges) are flagged.

Step 7: Standardise Product Descriptions

Product names are cleaned by removing excess spaces and inconsistent punctuation. Only product records are kept. Where a product code has multiple names, the most common cleaned version is applied across the dataset.

Step 8: Save Cleaned Table

The result is stored as a new table with standardised product descriptions and only product transactions.

Step 9: Review Column Ranges and Extremes

Date ranges, pricing extremes, and other checks are performed to identify any unusual values that may still exist.

Step 10: Investigate Near-Duplicates

Rows that differ by only one attributes (e.g. timestamp, quantity, or unit price) are examined in more detail. Some are legitimate (e.g. Variation in UnitPrice due to discounts). Transactions for the same InvoiceNo, StockCode, and UnitPrice that appear as separate rows are grouped, and their quantities summed. This ensures accurate product-level totals per transaction.

Step 11: Confirm uniqueness of key columns

Rows that differ only in timestamp (by one minute or less) are merged.

Step 12: Final Uniqueness Check

A final validation is conducted to ensure each record is uniquely identified by a combination of InvoiceNo, StockCode, and UnitPrice. This combination proved to be the most consistent unique key throughout the cleaning process.

Data Type Standardisation (02_datatypes.sql)

Purpose: Standardise column data types in OnlineRetail_Clean

Output: Updates the table with appropriate numeric formats

Step 1: Inspected current data types

Checked data types of all columns in the table.

Step 2: Validated data before conversion

Used ISNUMERIC() to ensure columns could safely be cast to numeric types without errors.

Step 3: Updated column types

Set appropriate types:

- InvoiceNo → INT
- Quantity → INT
- CustomerID → INT
- UnitPrice → DECIMAL(10, 2) for accurate monetary values

Step 4: Verified changes

Re-ran schema check to confirm updates were applied successfully.

Calculated Metrics (03_calculated_metrics.sql)

Purpose: Create transaction-level and customer-level metrics to prepare for business analysis and segmentation

Output: Two tables - OnlineRetail_Transactions and Customer_RFM

Step 1: Created transaction-level metrics

Added TotalSpend (Quantity × UnitPrice) and InvoiceYearMonth (for time-based analysis) to each row to enable spend tracking and monthly trend analysis. Saved table as OnlineRetail_Transactions.

Step 2: Computed Recency, Frequency, and Monetary (RFM) per customer

- **Recency:** Days since each customer's last purchase
- **Frequency:** Count of unique invoices (distinct purchases)
- **Monetary:** Total spend across all transactions

Saved table as Customer_RFM.

Customer Segmentation (04_customer_segments.sql)

Purpose: Segment customers based on Recency, Frequency, and Monetary scores
Output: A Customer_Segments table with RFM scores and segment labels

Step 1: Scored R, F, M metrics using quartiles

Assigned customers a score from 1 (lowest) to 4 (highest) for Recency, Frequency, and Monetary values to standardise comparisons.

Step 2: Created behavioural segments based on combined RFM scores

Used logical rules considering all three scores to classify customers into meaningful groups:

	R-score	F-score	M-score	Business Meaning
1. *Champions*	4	4	4	Recent, frequent, high spend
2. *Loyal Customers*	>=3	>=3		Frequent buyers
3. *Big Spenders*	>=2	>=2	4	High spend, regular customers
4. *At Risk*	<=2	>=2		Haven't purchased in a while
5. *Needs Attention*	<=2	<=2	<=2	Generally low across the board
6. *Lost*	1	1	1	Inactive, low across the board
7. *Others*				

Customers are defined by the first matched tier group. E.g. 3-3-4 would be classed as a Loyal Customer, not Big Spender.

Step 3: Saved final segmentation results

Stored the customer data including scores and segment labels in a new table.