# Retail Dataset Cleaning Step-by-Step Guide

---

## A. SETUP & BACKUPS

---

-- **A.1 Set database context**

-- Use the RetailDB database for all following operations.

```
USE RetailDB;
GO
```

**-- A.2 Inspect raw table (quick preview)**

-- Quick look at the original table for initial inspection.

```
SELECT * FROM [Online Retail];
```

**-- A.3 Create full backups / snapshots of the original data**

-- Keep two independent snapshots as recovery points before any modification.

```
SELECT * INTO dbo.OnlineRetail_backup FROM [Online Retail];
SELECT * INTO dbo.OnlineRetail_backup2 FROM [Online Retail];
```

---

## B. LOG / QUARANTINE TABLES (structures for outputs & audits)

---

**-- B.1 Create empty logging/quarantine tables (structure only)**

-- These will receive OUTPUT rows or quarantined records for later review.

```
SELECT TOP (0) * INTO dbo.OnlineRetail_DeletedLog FROM [Online Retail];
SELECT TOP (0) * INTO dbo.OnlineRetail_Quarantine FROM [Online Retail];
```

-- (ChangeLog is created below before procedures that write into it)

---

# C. INITIAL DATA QUALITY CHECKS

---

**-- C.1 High level null/invalid counts and quick diagnostics**

-- Count key problems before any modification so we have a baseline.

```
SELECT
    COUNT(*) AS TotalRows,
    SUM(CASE WHEN CustomerID IS NULL THEN 1 ELSE 0 END) AS NullCustomer,
    SUM(CASE WHEN Description IS NULL THEN 1 ELSE 0 END) AS NullDescription,
    SUM(CASE WHEN Quantity < 1 THEN 1 ELSE 0 END) AS NegativeQuantity,
    SUM(CASE WHEN Quantity IS NULL THEN 1 ELSE 0 END) AS NullQuantity,
    SUM(CASE WHEN UnitPrice IS NULL THEN 1 ELSE 0 END) AS NullPrice,
    SUM(CASE WHEN UnitPrice = 0 THEN 1 ELSE 0 END) AS ZeroUnitPrice,
    SUM(CASE WHEN Country IS NULL THEN 1 ELSE 0 END) AS NullCountry,
    SUM(CASE WHEN StockCode IS NULL THEN 1 ELSE 0 END) AS NullStockCode,
    SUM(CASE WHEN InvoiceNo IS NULL THEN 1 ELSE 0 END) AS NullInvoice,
    SUM(CASE WHEN InvoiceDate IS NULL THEN 1 ELSE 0 END) AS NullDate
FROM [Online Retail];
```

**-- C.2 Per-invoice full-null CustomerID detection**

```
SELECT InvoiceNo, COUNT(*) AS RowsPerInvoice
FROM [Online Retail]
GROUP BY InvoiceNo
HAVING SUM(CASE WHEN CustomerID IS NULL THEN 1 ELSE 0 END) = COUNT(*);
```

**-- C.3 Per-country distribution of missing CustomerID (prioritization)**

```
SELECT
    Country,
    COUNT(*) AS TotalRows,
    SUM(CASE WHEN CustomerID IS NULL THEN 1 ELSE 0 END) AS NullCust
FROM [Online Retail]
GROUP BY Country
ORDER BY NullCust DESC;
```

-- **C.4 Percentiles and min/max for quantity (absolute) and unit price**

```
WITH CTE AS (
    SELECT UnitPrice, ABS(Quantity) AS AbsQty
    FROM [Online Retail]
),
```

```
Percentiles AS (
    SELECT
        PERCENTILE_CONT(0.95) WITHIN GROUP (ORDER BY AbsQty) OVER() AS P95AbsQty,
        PERCENTILE_CONT(0.99) WITHIN GROUP (ORDER BY AbsQty) OVER() AS P99AbsQty,
        PERCENTILE_CONT(0.95) WITHIN GROUP (ORDER BY UnitPrice) OVER() AS P95Price,
        PERCENTILE_CONT(0.99) WITHIN GROUP (ORDER BY UnitPrice) OVER() AS P99Price
    FROM CTE
)
SELECT
    (SELECT MIN(AbsQty) FROM CTE) AS MinQty,
    (SELECT MAX(AbsQty) FROM CTE) AS MaxQty,
    (SELECT MIN(UnitPrice) FROM CTE) AS MinPrice,
    (SELECT MAX(UnitPrice) FROM CTE) AS MaxPrice,
    MAX(P95AbsQty) AS P95AbsQty,
    MAX(P99AbsQty) AS P99AbsQty,
    MAX(P95Price) AS P95Price,
    MAX(P99Price) AS P99Price
FROM Percentiles;

-- C.5 Show rows with the overall min or max quantity (investigate extremes)
SELECT *
FROM [Online Retail]
WHERE Quantity IN (
    SELECT MAX(Quantity) FROM [Online Retail]
    UNION
    SELECT MIN(Quantity) FROM [Online Retail]
);
```

---

## D. CLEANING TRANSACTION: delete extreme erroneous rows (with audit)

---

```
-- Begin transaction to wrap the main cleaning operations (commit later)
BEGIN TRAN;
```

-- **D.1 Remove extreme negative quantities that do not have a matching opposite row.**

```
-- Preserve deleted rows by OUTPUT into DeletedLog created earlier.
```

```
DELETE FROM [Online Retail]
OUTPUT deleted.* INTO dbo.OnlineRetail_DeletedLog
WHERE Quantity < 0
  AND ABS(Quantity) > 1000
  AND NOT EXISTS (
      SELECT 1
      FROM [Online Retail] R2
      WHERE R2.CustomerID = [Online Retail].CustomerID
        AND R2.StockCode = [Online Retail].StockCode
        AND R2.UnitPrice = [Online Retail].UnitPrice
        AND R2.InvoiceDate = [Online Retail].InvoiceDate
        AND R2.Quantity = ABS([Online Retail].Quantity)
  );
```

---

## E. CHANGE LOG & PROCEDURES (for tracked updates)

---

**-- E.1 Recreate the ChangeLog table (drop if exists) — used by procedures for audit**
**DROP TABLE IF EXISTS dbo.OnlineRetail_ChangeLog;**

```
CREATE TABLE dbo.OnlineRetail_ChangeLog (
    OldInvoiceNo NVARCHAR(50),
    NewInvoiceNo NVARCHAR(50),
    OldStockCode NVARCHAR(50),
    NewStockCode NVARCHAR(50),
    OldDescription NVARCHAR(4000),
    NewDescription NVARCHAR(4000),
    OldQuantity BIGINT,
    NewQuantity BIGINT,
    OldInvoiceDate DATE,
    NewInvoiceDate DATE,
    OldUnitPrice FLOAT,
    NewUnitPrice FLOAT,
    OldCustomerID NVARCHAR(50),
    NewCustomerID NVARCHAR(50),
    OldCountry NVARCHAR(50),
    NewCountry NVARCHAR(50),
    ChangeDate DATETIME DEFAULT GETDATE()
);
```

**-- E.2 Stored procedure: simple dynamic update with OUTPUT into change log**

```
CREATE OR ALTER PROCEDURE usp_UpdateOnlineRetail
    @ColumnName NVARCHAR(128),
```

```
    @NewValue NVARCHAR(MAX),
    @Condition NVARCHAR(MAX)
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @SQL NVARCHAR(MAX);

    SET @SQL = N'
        UPDATE dbo.[Online Retail]
        SET ' + QUOTENAME(@ColumnName) + N' = @NewValue
        OUTPUT
            deleted.InvoiceNo,   inserted.InvoiceNo,
            deleted.StockCode,   inserted.StockCode,
            deleted.Description, inserted.Description,
            deleted.Quantity,    inserted.Quantity,
            deleted.InvoiceDate, inserted.InvoiceDate,
            deleted.UnitPrice,   inserted.UnitPrice,
            deleted.CustomerID,  inserted.CustomerID,
            deleted.Country,     inserted.Country,
            GETDATE()
        INTO dbo.OnlineRetail_ChangeLog
        (
            OldInvoiceNo, NewInvoiceNo,
            OldStockCode, NewStockCode,
            OldDescription, NewDescription,
            OldQuantity, NewQuantity,
            OldInvoiceDate, NewInvoiceDate,
            OldUnitPrice, NewUnitPrice,
            OldCustomerID, NewCustomerID,
            OldCountry, NewCountry,
            ChangeDate
        )
        WHERE ' + @Condition + ';';

                EXEC sp_executesql @SQL, N'@NewValue NVARCHAR(MAX)',
@NewValue=@NewValue;
END;
GO
```

**-- E.3 Use the procedure to set NULL or blank CustomerID in the original table to 'UNKNOWN'**

```
EXEC usp_UpdateOnlineRetail
    @ColumnName = 'CustomerID',
    @NewValue = 'UNKNOWN',
    @Condition = 'CustomerID IS NULL OR LTRIM(RTRIM(CustomerID)) = ''';
```

---

# F. PREPARE CLEAN WORKING TABLE(S)

---

**-- F.1 Create a single cleaned working table (OnlineRetail_Clean)**

```
IF OBJECT_ID('dbo.OnlineRetail_Clean') IS NOT NULL DROP TABLE dbo.OnlineRetail_Clean;
CREATE TABLE dbo.OnlineRetail_Clean (
  RowID BIGINT IDENTITY(1,1) PRIMARY KEY,
  InvoiceNo NVARCHAR(50),
  StockCode NVARCHAR(50),
  Description NVARCHAR(400),
  Quantity INT,
  InvoiceDate DATETIME,
  UnitPrice DECIMAL(18,4),
  CustomerID NVARCHAR(50),
  Country NVARCHAR(100),
  LoadDate DATETIME DEFAULT GETDATE()
);
```

**-- F.2 Load valid/parsible rows into OnlineRetail_Clean (trim + type conversion)**

```
-- Rows that fail conversion remain in the original table and will be quarantined below.

INSERT INTO dbo.OnlineRetail_Clean
      (InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, Country)
SELECT
   LTRIM(RTRIM(InvoiceNo)),
   LTRIM(RTRIM(StockCode)),
   NULLIF(LTRIM(RTRIM(Description)), ''),
   TRY_CONVERT(INT, LTRIM(RTRIM(Quantity))),
   TRY_CONVERT(DATETIME, LTRIM(RTRIM(InvoiceDate))),
   TRY_CONVERT(DECIMAL(18,4), REPLACE(LTRIM(RTRIM(UnitPrice)), ',', '')),
   NULLIF(LTRIM(RTRIM(CustomerID)), ''),
   LTRIM(RTRIM(Country))
FROM [Online Retail]
WHERE TRY_CONVERT(INT, LTRIM(RTRIM(Quantity))) IS NOT NULL
  AND TRY_CONVERT(DATETIME, LTRIM(RTRIM(InvoiceDate))) IS NOT NULL
   AND TRY_CONVERT(DECIMAL(18,4), REPLACE(LTRIM(RTRIM(UnitPrice)), ',', '')) IS NOT NULL;
```

**-- F.3 Report counts: valid vs invalid (how many rows parsed successfully)**

```
SELECT
    COUNT(*) AS TotalData,
    SUM(CASE WHEN TRY_CONVERT(INT, LTRIM(RTRIM(Quantity))) IS NOT NULL
             AND TRY_CONVERT(DATETIME, LTRIM(RTRIM(InvoiceDate))) IS NOT NULL
               AND TRY_CONVERT(DECIMAL(18,4), REPLACE(LTRIM(RTRIM(UnitPrice)), ',',
'')) IS NOT NULL
            THEN 1 ELSE 0 END) AS ValidData,
    SUM(CASE WHEN TRY_CONVERT(INT, LTRIM(RTRIM(Quantity))) IS NULL
             OR TRY_CONVERT(DATETIME, LTRIM(RTRIM(InvoiceDate))) IS NULL
             OR TRY_CONVERT(DECIMAL(18,4), REPLACE(LTRIM(RTRIM(UnitPrice)), ',', ''))
IS NULL
            THEN 1 ELSE 0 END) AS InvalidData
FROM [Online Retail];
```

**-- F.4 Move rows that did not parse into quarantine (structure preserved from original)**

```
INSERT INTO dbo.OnlineRetail_Quarantine
SELECT *
FROM dbo.[Online Retail] r
WHERE NOT EXISTS (
  SELECT 1 FROM dbo.OnlineRetail_Clean c
  WHERE c.InvoiceNo = LTRIM(RTRIM(r.InvoiceNo))
    AND c.StockCode = LTRIM(RTRIM(r.StockCode))
    -- join logic can be extended for uniqueness
);
```

---

# G. PRODUCT METADATA & DESCRIPTION CLEANING

---

**-- G.1 Build ProductMaster: choose the most frequent description per StockCode (for imputation)**

```
IF OBJECT_ID('dbo.ProductMaster') IS NOT NULL DROP TABLE dbo.ProductMaster;

SELECT
    x.StockCode,
    TOP_Desc = MAX(x.Description)
INTO dbo.ProductMaster
FROM (
    SELECT
        StockCode,
        Description,
        COUNT(*) AS cnt,
```

```
        ROW_NUMBER() OVER (PARTITION BY StockCode ORDER BY COUNT(*) DESC)
rn
    FROM dbo.OnlineRetail_Clean
    WHERE Description <> ''
    GROUP BY StockCode, Description
) x
WHERE rn = 1
GROUP BY x.StockCode;
```

**-- G.2 Fill missing descriptions in clean table from ProductMaster**

```
UPDATE c
SET c.Description = pm.TOP_Desc
FROM dbo.OnlineRetail_Clean c
JOIN dbo.ProductMaster pm ON c.StockCode = pm.StockCode
WHERE (c.Description IS NULL OR c.Description = '')
  AND pm.TOP_Desc IS NOT NULL;

-- G.3 Standardize descriptions to upper-case and trimmed form
UPDATE dbo.OnlineRetail_Clean
SET Description = UPPER(LTRIM(RTRIM(Description)));
```

---

## H. FLEXIBLE UPDATE PROC (for complex updates against the clean table)

---

**-- H.1 Create a flexible update procedure that logs changes into OnlineRetail_ChangeLog**

```
CREATE OR ALTER PROCEDURE usp_UpdateOnlineRetail_Flex
    @ColumnName NVARCHAR(128),   -- Column to update
    @UpdateQuery NVARCHAR(MAX),  -- Query/expression for new value (can reference c)
    @Condition NVARCHAR(MAX)     -- WHERE condition to select rows
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @SQL NVARCHAR(MAX);

    SET @SQL = N'
    UPDATE c
    SET c.' + QUOTENAME(@ColumnName) + N' = ' + @UpdateQuery + N'
    OUTPUT
        deleted.InvoiceNo,   inserted.InvoiceNo,
        deleted.StockCode,   inserted.StockCode,
```

```
        deleted.Description, inserted.Description,
        deleted.Quantity,    inserted.Quantity,
        deleted.InvoiceDate, inserted.InvoiceDate,
        deleted.UnitPrice,   inserted.UnitPrice,
        deleted.CustomerID,  inserted.CustomerID,
        deleted.Country,     inserted.Country,
        GETDATE()
    INTO dbo.OnlineRetail_ChangeLog
    (
        OldInvoiceNo, NewInvoiceNo,
        OldStockCode, NewStockCode,
        OldDescription, NewDescription,
        OldQuantity, NewQuantity,
        OldInvoiceDate, NewInvoiceDate,
        OldUnitPrice, NewUnitPrice,
        OldCustomerID, NewCustomerID,
        OldCountry, NewCountry,
        ChangeDate
    )
    FROM dbo.OnlineRetail_Clean c
    WHERE ' + @Condition + ';';

    EXEC sp_executesql @SQL;
END;
GO
```

**-- H.2 Use flexible proc to fill CustomerID from other rows in same invoice when available**

```
EXEC usp_UpdateOnlineRetail_Flex
    @ColumnName = 'CustomerID',
    @UpdateQuery = '(SELECT MAX(t.CustomerID)
                     FROM dbo.OnlineRetail_Clean t
                     WHERE t.InvoiceNo = c.InvoiceNo
                       AND t.CustomerID IS NOT NULL)',
    @Condition = '(c.CustomerID IS NULL OR LTRIM(RTRIM(c.CustomerID)) = "")
                  AND EXISTS (
                    SELECT 1
                    FROM dbo.OnlineRetail_Clean t
                    WHERE t.InvoiceNo = c.InvoiceNo
                      AND t.CustomerID IS NOT NULL
                  )';
```

**-- H.3 Fallback: set remaining blank/null CustomerID to 'UNKNOWN' in the clean table (logged)**

```
EXEC usp_UpdateOnlineRetail_Flex
    @ColumnName = 'CustomerID',
```

```
    @UpdateQuery = "'UNKNOWN'",
    @Condition = 'NULLIF(LTRIM(RTRIM(CustomerID)), "") = NULL';
```

---

# I. DEDUPLICATION & DELETION LOGGING

---

```
-- Remove exact duplicates in the clean table, keeping one, and log deleted duplicates into
DeletedLog

 ;WITH d AS (
   SELECT *,
        ROW_NUMBER() OVER (
                PARTITION BY InvoiceNo, StockCode, Quantity, UnitPrice, InvoiceDate,
 CustomerID
           ORDER BY (SELECT 1)
        ) AS rn
   FROM dbo.OnlineRetail_Clean
 )
 DELETE d
 OUTPUT
    DELETED.InvoiceNo,
    DELETED.StockCode,
    DELETED.Description,
    DELETED.Quantity,
    DELETED.InvoiceDate,
    DELETED.UnitPrice,
    DELETED.CustomerID,
    DELETED.Country
 INTO dbo.OnlineRetail_DeletedLog
        (InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID,
 Country)
 WHERE d.rn > 1;
```

---

# J. TRANSACTION LABELING, RETURNS MATCHING & ADJUSTMENTS

---

**-- J.1 Add TransactionType and classify simple returns/sales by InvoiceNo prefix or negative qty**

```
ALTER TABLE dbo.OnlineRetail_Clean
ADD TransactionType NVARCHAR(20);

UPDATE dbo.OnlineRetail_Clean
SET TransactionType = CASE WHEN LEFT(InvoiceNo,1)='C' OR Quantity < 0 THEN
'RETURN' ELSE 'SALE' END;
```

**-- J.2 Pair returns with sales (within ±2 days, same product/customer/price) to classify valid returns vs adjustments**

```
;WITH returns AS (
      SELECT RowID, StockCode, ABS(Quantity) AS ReturnQty, UnitPrice, CustomerID,
InvoiceDate
   FROM dbo.OnlineRetail_Clean
   WHERE TransactionType = 'RETURN'
),
sales AS (
   SELECT RowID, StockCode, Quantity AS SaleQty, UnitPrice, CustomerID, InvoiceDate
   FROM dbo.OnlineRetail_Clean
   WHERE TransactionType = 'SALE'
),
candidates AS (
   SELECT r.RowID AS ReturnRow, s.RowID AS SaleRow,
        r.ReturnQty, s.SaleQty,
        ROW_NUMBER() OVER (
          PARTITION BY r.RowID
          ORDER BY ABS(DATEDIFF(day, s.InvoiceDate, r.InvoiceDate))
        ) AS rn
   FROM returns r
   JOIN sales s
     ON r.StockCode = s.StockCode
    AND r.UnitPrice = s.UnitPrice
    AND r.CustomerID = s.CustomerID
    AND ABS(DATEDIFF(day, s.InvoiceDate, r.InvoiceDate)) <= 2
),
bestMatch AS (
   SELECT *
   FROM candidates
   WHERE rn = 1
)
SELECT b.ReturnRow, b.SaleRow,
      CASE
        WHEN b.ReturnQty <= b.SaleQty THEN 'RETURN'
        ELSE 'STOCK_ADJUSTMENT'
      END AS NewType
INTO #ReturnClass
FROM bestMatch b;
```

**-- J.3 Apply classifications from #ReturnClass**

```
UPDATE c
SET TransactionType = 'RETURN'
FROM dbo.OnlineRetail_Clean c
JOIN #ReturnClass rc ON c.RowID = rc.ReturnRow
WHERE rc.NewType = 'RETURN';

UPDATE c
SET TransactionType = 'STOCK_ADJUSTMENT'
FROM dbo.OnlineRetail_Clean c
JOIN #ReturnClass rc ON c.RowID = rc.ReturnRow
WHERE rc.NewType = 'STOCK_ADJUSTMENT';
```

**-- J.4 Any remaining RETURN rows without a matched sale -> STOCK_ADJUSTMENT**

```
UPDATE c
SET TransactionType = 'STOCK_ADJUSTMENT'
FROM dbo.OnlineRetail_Clean c
WHERE TransactionType = 'RETURN'
  AND RowID NOT IN (SELECT ReturnRow FROM #ReturnClass);

-- J.5 Quick summaries for QA: counts per TransactionType and return vs adjustment totals
SELECT TransactionType, COUNT(*) AS TotalRows
FROM dbo.OnlineRetail_Clean
GROUP BY TransactionType;

SELECT
      SUM(CASE WHEN TransactionType = 'RETURN' THEN 1 ELSE 0 END) AS
ValidReturn,
    SUM(CASE WHEN TransactionType = 'STOCK_ADJUSTMENT' THEN 1 ELSE 0 END)
AS Adjustments
FROM dbo.OnlineRetail_Clean;
```

---

# K. EXTREME ADJUSTMENT QUARANTINE & ZERO-PRICE HANDLING

---

**-- K.1 Compute P99 quantity threshold (for capping extreme adjustments)**

```
DECLARE @P99Qty FLOAT;
SELECT @P99Qty = PERCENTILE_CONT(0.99) WITHIN GROUP (ORDER BY
ABS(Quantity)) OVER() FROM dbo.OnlineRetail_Clean;
```

**-- K.2 Cap (quarantine) extreme adjustments by setting qty to P99 (preserving originals was done earlier via OUTPUT)**

```
UPDATE dbo.OnlineRetail_Clean
SET Quantity = SIGN(Quantity) * CAST(@P99Qty AS INT)
WHERE TransactionType = 'STOCK_ADJUSTMENT' AND ABS(Quantity) > @P99Qty;
```

**-- K.3 Flag rows with zero UnitPrice for inspection/imputation**

```
ALTER TABLE dbo.OnlineRetail_Clean ADD Flag_ZeroPrice TINYINT DEFAULT 0;
UPDATE dbo.OnlineRetail_Clean SET Flag_ZeroPrice = 1 WHERE UnitPrice = 0;
```

**-- K.4 Aggregate zero-price occurrences by product to prioritize imputation**

```
SELECT StockCode,
      COUNT(*) ZeroCount,
      SUM(CASE WHEN Quantity > 0 THEN 1 ELSE 0 END) AS ZeroSalesCount
FROM dbo.OnlineRetail_Clean
WHERE Flag_ZeroPrice = 1
GROUP BY StockCode
ORDER BY ZeroSalesCount DESC;
```

**-- K.5 Compute median price per StockCode to use for imputing zero prices**

```
SELECT StockCode, PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY UnitPrice)
OVER (PARTITION BY StockCode) AS MedianPrice
INTO #MedianPerProduct2
FROM dbo.OnlineRetail_Clean
WHERE UnitPrice > 0;
```

**-- K.6 Impute zero UnitPrice for SALE rows using median per product and mark as imputed (Flag_ZeroPrice = 2)**

```
UPDATE c
SET c.UnitPrice = m.MedianPrice,
   c.Flag_ZeroPrice = 2
FROM dbo.OnlineRetail_Clean c
JOIN (
   SELECT StockCode,
        PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY UnitPrice)
          OVER (PARTITION BY StockCode) AS MedianPrice,
         ROW_NUMBER() OVER (PARTITION BY StockCode ORDER BY StockCode) AS
rn
   FROM dbo.OnlineRetail_Clean
   WHERE UnitPrice > 0
) m ON c.StockCode = m.StockCode AND m.rn = 1
WHERE c.UnitPrice = 0
  AND c.TransactionType = 'SALE'
```

```
  AND c.Quantity > 0;
```

---

# L. DERIVED COLUMNS, VIEWS AND INDEXES FOR ANALYSIS

---

**-- L.1 Add a computed persisted LineAmount column to simplify reporting**

```
ALTER TABLE dbo.OnlineRetail_Clean
ADD LineAmount AS (Quantity * UnitPrice) PERSISTED;
```

**-- L.2 Add date parts and a CustomerType field for reporting**

```
ALTER TABLE dbo.OnlineRetail_Clean ADD InvoiceDate_Date DATE, InvoiceYear INT,
InvoiceMonth INT, CustomerType NVARCHAR(20);
UPDATE dbo.OnlineRetail_Clean
SET InvoiceDate_Date = CONVERT(date, InvoiceDate),
    InvoiceYear = YEAR(InvoiceDate),
    InvoiceMonth = MONTH(InvoiceDate),
      CustomerType = CASE WHEN CustomerID='UNKNOWN' THEN 'UNKNOWN' ELSE
'KNOWN' END;
```

**-- L.3 View: Net sales per product per day (SALE + RETURN as net)**

```
CREATE OR ALTER VIEW dbo.vNetSales AS
SELECT StockCode, Description, InvoiceDate_Date,
      SUM(LineAmount) AS NetRevenue,
      SUM(Quantity) AS NetQuantity
FROM dbo.OnlineRetail_Clean
WHERE TransactionType IN ('SALE','RETURN')
GROUP BY StockCode, Description, InvoiceDate_Date;
```

**-- L.4 View: Sales and Returns breakdown per product/day**

```
CREATE OR ALTER VIEW dbo.vSalesReturn AS
SELECT
    StockCode,
    Description,
    InvoiceDate_Date,
      SUM(CASE WHEN TransactionType = 'SALE' THEN LineAmount ELSE 0 END) AS
SaleRevenue,
     SUM(CASE WHEN TransactionType = 'RETURN' THEN LineAmount ELSE 0 END) AS
ReturnRevenue,
    SUM(LineAmount) AS NetRevenue,
```

```
      SUM(CASE WHEN TransactionType = 'SALE' THEN Quantity ELSE 0 END) AS SaleQuantity,
      SUM(CASE WHEN TransactionType = 'RETURN' THEN Quantity ELSE 0 END) AS ReturnQuantity,
    SUM(Quantity) AS NetQuantity
  FROM dbo.OnlineRetail_Clean
  WHERE TransactionType IN ('SALE','RETURN')
  GROUP BY StockCode, Description, InvoiceDate_Date;
```

**-- L.5 View: Stock adjustments summary**

```
  CREATE OR ALTER VIEW dbo.vStockAdjustments AS
  SELECT StockCode, Description, InvoiceDate_Date,
      SUM(LineAmount) AS AdjAmount, SUM(Quantity) AS AdjQty
  FROM dbo.OnlineRetail_Clean
  WHERE TransactionType = 'STOCK_ADJUSTMENT'
  GROUP BY StockCode, Description, InvoiceDate_Date;
```

**-- L.6 Example checks (now that views exist)**

```
  SELECT TOP 20 * FROM dbo.vSalesReturn ORDER BY ReturnRevenue ASC;
  SELECT * FROM dbo.vStockAdjustments;
```

**-- L.7 Create helpful indexes to speed queries on the clean table**

```
  CREATE INDEX IX_ORC_InvoiceDate ON dbo.OnlineRetail_Clean (InvoiceDate);
  CREATE INDEX IX_ORC_StockCode ON dbo.OnlineRetail_Clean (StockCode);
  CREATE INDEX IX_ORC_CustomerID ON dbo.OnlineRetail_Clean (CustomerID);
```

---

# M. DATA QUALITY REPORTING & FINALIZATION

---

**-- M.1 Create DataQualityReport table if missing (store before/after metrics)**

```
  IF OBJECT_ID('dbo.DataQualityReport') IS NULL
  BEGIN
    CREATE TABLE dbo.DataQualityReport (
      ReportDate DATETIME DEFAULT GETDATE(),
      Phase NVARCHAR(50),
      TotalRows INT,
      NullCustomer INT,
      ZeroPrice INT,
      NegativeQty INT
    );
```

```
END

-- M.2 Insert an after-cleaning snapshot

INSERT INTO dbo.DataQualityReport (Phase, TotalRows, NullCustomer, ZeroPrice, NegativeQty)
SELECT 'AfterCleaning',
  (SELECT COUNT(*) FROM dbo.OnlineRetail_Clean),
  (SELECT COUNT(*) FROM dbo.OnlineRetail_Clean WHERE CustomerID='UNKNOWN'),
  (SELECT COUNT(*) FROM dbo.OnlineRetail_Clean WHERE UnitPrice = 0),
  (SELECT COUNT(*) FROM dbo.OnlineRetail_Clean WHERE Quantity < 0);

-- M.3 Insert a before-clean snapshot (baseline)

INSERT INTO dbo.DataQualityReport (Phase, TotalRows, NullCustomer, ZeroPrice, NegativeQty)
SELECT 'BeforeCleaning',
  (SELECT COUNT(*) FROM [Online Retail]),
  (SELECT COUNT(*) FROM [Online Retail] WHERE CustomerID IS NULL),
  (SELECT COUNT(*) FROM [Online Retail] WHERE UnitPrice = 0),
  (SELECT COUNT(*) FROM [Online Retail] WHERE Quantity < 0);

-- M.4 Final counts for quick QA
SELECT
  (SELECT COUNT(*) FROM dbo.OnlineRetail_Clean) AS RowsClean,
  (SELECT COUNT(*) FROM dbo.OnlineRetail_DeletedLog) AS RowsDeletedLog,
  (SELECT COUNT(*) FROM dbo.OnlineRetail_Quarantine) AS RowsQuarantine;

-- M.5 Commit the cleaning transaction to persist changes
COMMIT TRAN;

-------------------------------------------------------------------------------
```

# N. FINAL ANALYSIS VIEW

```
-------------------------------------------------------------------------------

-- N.1 Create a flattened view for downstream analysis and BI

CREATE OR ALTER VIEW dbo.vOnlineRetail_Analysis AS
SELECT
    InvoiceNo,
    StockCode,
    Description,
    Quantity,
    UnitPrice,
    LineAmount,
```

```
    InvoiceDate,
    CONVERT(date, InvoiceDate) AS InvoiceDate_Date,
    YEAR(InvoiceDate) AS InvoiceYear,
    MONTH(InvoiceDate) AS InvoiceMonth,
    CustomerID,
     CASE WHEN CustomerID = 'UNKNOWN' THEN 'UNKNOWN' ELSE 'KNOWN' END AS
CustomerType,
    Country,
    TransactionType
FROM dbo.OnlineRetail_Clean;
```

**-- N.2 Final check: show the analysis view**

```
SELECT * FROM dbo.vOnlineRetail_Analysis;
```