

1) Source Data Verification — Approach & Findings

Tools / Setup

I used **SQLiteOnline** (<https://sqliteonline.com/>) to ensure the process was simple and reproducible.

Source Files & Formats (as Received)

The dataset was provided as three separate files in different formats:

- **Customer.xls**: An Excel workbook with a single sheet containing columns: Customer_ID, First, Last, Age, Country.
- **Order.csv**: A CSV file with columns: Order_ID, Item, Amount, Customer_ID.
- **Shipping.json**: A JSON array of objects with fields: Shipping_ID, Status, Customer_ID.

Handling Mixed Formats in SQLiteOnline

I handled the different file formats by uploading each file directly into **SQLiteOnline**, creating three separate staging tables: customers, orders, and shipping. This approach, which matched column names to the source headers, ensured traceability.

- For the Excel (.xls) file, I used SQLiteOnline's direct file upload feature to ingest the first sheet as a flat table.
- For the CSV, I confirmed the **UTF-8 encoding** and **comma delimiter** were correct before importing.
- For the JSON, the file was imported as a single array of records, with each object becoming a single row in the shipping table.

Import Checks at Load Time

During the ingestion process, I performed the following checks to ensure data integrity:

- I **verified row counts** for each table to confirm they matched the file expectations.
- I confirmed that **key columns were non-null** after the import.
- I **spot-checked a few rows** in each table to ensure data types were parsed as intended (e.g., Amount as numeric, Age as an integer, and Status as text).
- I kept the column names identical to the source files for easy traceability.

Staging tables created:

- **customers**: Customer_ID, First, Last, Age, Country
- **orders**: Order_ID, Item, Amount, Customer_ID
- **shipping**: Shipping_ID, Status, Customer_ID

What I Checked (and Why)

1. **Row/Column Sanity & Nulls:** To confirm the data feeds loaded correctly and no crucial fields were missing.
2. **Primary Key Uniqueness:** To ensure each table had a stable natural key from the source.
3. **Referential Integrity:** To verify that foreign keys are correctly resolved (orders.customer_id → customers.customer_id, shipping.customer_id → customers.customer_id).
4. **Domain / Business Rules:** To validate that values were within valid ranges (e.g., Amount > 0, Status was limited to expected values like 'Pending' or 'Delivered').
5. **Text Hygiene:** To identify and address "leetspeak" or stray characters in names (e.g., N!cole, L@rry, R0bert, A1cia).
6. **Structural Anomalies:** To detect issues like multiple shipping rows per customer or customers without corresponding orders/shipping data.

SQL I Ran

Nulls / Shape

SQL

SELECT

```
(SELECT COUNT(*) FROM customers) AS customers_rows,  
(SELECT COUNT(*) FROM orders) AS orders_rows,  
(SELECT COUNT(*) FROM shipping) AS shipping_rows;
```

-- Example for checking specific column nulls

-- SELECT SUM(CASE WHEN Customer_ID IS NULL THEN 1 ELSE 0 END) FROM customers;

Primary-Key Uniqueness

SQL

```
SELECT Customer_ID, COUNT(*) c FROM customers GROUP BY 1 HAVING c > 1;  
SELECT Order_ID, COUNT(*) c FROM orders GROUP BY 1 HAVING c > 1;  
SELECT Shipping_ID, COUNT(*) c FROM shipping GROUP BY 1 HAVING c > 1;
```

Referential Integrity

SQL

-- Orders → Customers

```
SELECT o.*
FROM orders o
LEFT JOIN customers c
  ON c.Customer_ID = o.Customer_ID
WHERE c.Customer_ID IS NULL;
```

-- Shipping → Customers

```
SELECT s.*
FROM shipping s
LEFT JOIN customers c
  ON c.Customer_ID = s.Customer_ID
WHERE c.Customer_ID IS NULL;
```

Business Rules

SQL

-- Amount must be positive

```
SELECT COUNT(*) FROM orders WHERE Amount <= 0;
```

-- Status limited to two values

```
SELECT DISTINCT Status FROM shipping;
```

Name Hygiene

SQL

-- Has digits

```
SELECT Customer_ID, First FROM customers WHERE First GLOB '[0-9]*';
SELECT Customer_ID, Last FROM customers WHERE Last GLOB '[0-9]*';
```

-- Has disallowed special characters (blacklist pattern)

```
SELECT Customer_ID, First FROM customers
WHERE First GLOB '*[!@#$$%^&*()_+=\[\];:~"'\.,/<>?\]\]*';
SELECT Customer_ID, Last FROM customers
WHERE Last GLOB '*[!@#$$%^&*()_+=\[\];:~"'\.,/<>?\]\]*';
```

Coverage / Structural Checks

SQL

-- Customers with no orders

```
SELECT COUNT(*) AS customers_without_orders
FROM customers c
LEFT JOIN orders o
  ON o.Customer_ID = c.Customer_ID
WHERE o.Customer_ID IS NULL;
```

-- Customers with no shipping rows

```
SELECT COUNT(*) AS customers_without_shipping
FROM customers c
LEFT JOIN shipping s
  ON s.Customer_ID = c.Customer_ID
WHERE s.Customer_ID IS NULL;
```

-- Multiple shipping rows per customer

```
SELECT Customer_ID, COUNT(*) AS shipping_rows
FROM shipping
GROUP BY Customer_ID
HAVING COUNT(*) > 1;
```

What I Found

- **Loads & Nulls:** All three files loaded cleanly with no `NULL` values in the key fields. Each file had 250 rows.
- **Primary Keys:** No duplicate `Customer_ID`, `Order_ID`, or `Shipping_ID` values were found.
- **Foreign Keys:** 100% of the `orders.customer_id` and `shipping.customer_id` values successfully matched a customer, indicating no orphaned rows.
- **Domain Checks:**
 - All `Amount` values were positive.
 - `Status` values were limited to 'Pending' and 'Delivered'.
- **Name Hygiene:** I did find a few names with digits or symbols (e.g., N!cole, L@rry, R0bert, Al1cia). I developed a repeatable transformation to map these characters to letters.
- **Structural Anomaly:** A significant finding was that the **shipping table is at the customer level, not the order level**. This means a customer can have multiple shipping rows, which can cause double-counting of order amounts if not handled carefully in joins.
 - My **interim approach** was to collapse shipping to one row per customer for reporting purposes.
 - **Note:** For future robustness, the `shipping` table should ideally include `Order_ID` (and/or timestamps) to accurately model shipping at the order level.

Bottom Line (Data Quality Stance)

- **Accuracy:** The data is highly accurate, with unique keys, resolving foreign keys, and clean numeric and categorical domains.
- **Completeness:** There were no missing rows in the core join paths, and I accounted for customers without orders or shipping rows.
- **Reliability:** Data distributions were sane. The primary modeling observation is the customer-level grain of the `shipping` table, which was addressed in the dataset design.