**E-Commerce Data Integration and ETL using SSIS**

**Name:** Kusha Patel

**Intern ID:** INT932

**Tool Used:** Microsoft SQL Server Integration Service (SSIS), Microsoft SQL Server Management Studio (SSMS)

**Submission Date:** 2nd, June 2025

**Table of Content**

|  |  |  |
| --- | --- | --- |
| **Sr No.** | **Content** | **Page No.** |
| 1 | Introduction | 3 |
| 2 | Objective | 4 |
| 3 | Tools & Technologies Used | 5 |
| 4 | Source Data Description | 6 |
| 5 | Data Warehouse Schema | 7 |
| 6 | Dimension Tables | 8 |
| 7 | Fact Tables | 10 |
| 8 | SSIS Packages | 12 |
|  | 8.1 Package 1: LoadDimensions.dtsx | 12 |
|  | 8.2 Package 2: LoadFactOrders.dtsx | 20 |
|  | 8.3 Package 3: LoadRatings.dtsx | 26 |
|  | 8.4 Package 4: LoadCampaignProductSubCategory.dtsx | 28 |
| 9 | Error Handling | 30 |
| 10 | Challenges and Solutions | 31 |
| 11 | Conclusion | 32 |

**Introduction**

In today’s digital economy, data is the lifeblood of business strategy, operational excellence, and customer satisfaction. Shop Smart, a rapidly expanding e-commerce platform, relies heavily on data-driven decision-making to understand customer behavior, track product performance, optimize marketing campaigns, and identify market trends. To support its global operations and growing analytics needs, a reliable, scalable, and auditable data integration solution is crucial.

This project presents the design and implementation of a robust ETL (Extract, Transform, Load) pipeline using SQL Server Integration Services (SSIS) to automate the ingestion, transformation, validation, and loading of raw data from multiple CSV sources into a well-structured Snowflake schema within a SQL Server Data Warehouse.

The ETL architecture emphasizes:

* Data quality enforcement through cleansing, validation, and error handling.
* Logging of bad records via centralized error description and audit tables.
* Modular SSIS package design that promotes reusability and scalability.
* Support for incremental data loading to optimize performance and ensure up-to-date analytics.

This structured and automated data flow empowers Shop Smart with consistent, accurate, and actionable data, ensuring reliable insights across key performance metrics such as sales trends, customer feedback, returns, and supplier efficiency.

**Objectives**

The objective of this project is to design and deploy a comprehensive, scalable ETL solution using SQL Server Integration Services (SSIS) that extracts raw e-commerce data from various CSV files, performs data quality checks and transformations, and loads it into a centralized SQL Server data warehouse using a Snowflake schema model.

Key objectives include:

* **Streamlining Data Integration**: Automate the extraction and loading of raw CSV data into staging, cleaned, and fact/dimension tables.
* **Ensuring Data Accuracy**: Apply strict validation rules, lookups, and transformations to eliminate inconsistencies and improve trust in the data.
* **Error Handling and Logging**: Capture invalid data with precise error descriptions in dedicated log tables (BadDataLog, Error Description) for auditing and debugging.
* **Implementing Reusable SSIS Frameworks**: Use a common control and data flow structure across all packages for consistency and maintainability.
* **Adopting Snowflake Schema Modelling**: Design dimension and fact tables that normalize data while supporting efficient querying and analytics.
* **Supporting Incremental Loads**: Enable partial data updates without reloading the entire dataset, preserving system performance.
* **Preparing for Scalability**: Structure the ETL process to handle additional data sources and business use cases in the future with minimal rework.

**Tools & Technologies Used**

**1. SQL Server Integration Services (SSIS)**

* **Purpose**: Core ETL tool for building packages to extract, transform, and load data.
* **Used for**:
  + Designing Control Flow and Data Flow tasks
  + Implementing data validation and transformation
  + Error handling, logging, and incremental loading

**2. Microsoft SQL Server**

* **Purpose**: Target database system for the data warehouse.
* **Used for**:
  + Hosting staging, cleaned, dimension, and fact tables
  + Snowflake schema implementation
  + Supporting BI reporting and querying

**3. Visual Studio (with SQL Server Data Tools - SSDT)**

* **Purpose**: Development environment for creating and debugging SSIS packages.
* **Used for**:
  + Package development and deployment
  + Designing control/data flow
  + Managing configurations and parameters

**4. CSV Files (Comma-Separated Values)**

* **Purpose**: Source data format for raw e-commerce data.
* **Used for**:
  + Simulating real-world business data from various domains (e.g., Customers, Products, Sales, etc.)
  + Input to the ETL pipeline

**5. SSIS Logging and Error Handling Mechanisms**

* **Purpose**: Capturing and storing errors and invalid data.
* **Used for**:
  + Redirecting bad data to BadDataLog and Error\_description\_\* tables
  + Ensuring traceability and auditability of failed records

**Source Data Description**

|  |  |  |
| --- | --- | --- |
| **CSV File Name** | **Columns** | **Description** |
| customer.csv | customer\_id(int), first\_name(string), last\_name(string), email(string), country(string) | Contains customer profile information. |
| supplier.csv | supplier\_id(int), supplier\_name(string), email(string) | Contains details about the suppliers |
| category.csv | category\_id(int), category\_name(string) | Represents high-level product categories. |
| subcategory.csv | subcategory\_id(int), subcategory\_name(string), category\_id(int) | Defines product subcategories and links them to parent categories. |
| product.csv | product\_id(int), name(string), price(float), description(string), subcategory\_id(int) | Includes information about all products available on the platform. |
| payment\_method.csv | payment\_method\_id(int), payment\_method(string) | Stores available payment methods used by customers. |
| marketing\_campaigns.csv | campaign\_id(int), campaign\_name(string), offer\_week(int) | Contains information about promotional campaigns run by the company. |
| orders.csv | order\_id\_surrogate(int), order\_id(int), customer\_id(int), order\_date(int), campaign\_id(int), amount(int), payment\_method\_id(int) | Captures overall order information, including customer, payment, and campaign details. |
| orderitem.csv | orderitem\_id(int), order\_id(int), product\_id(int), quantity(int), supplier\_id(int), subtotal(float), discount(float) | Represents individual items within an order, including quantities and discounts. |
| returns.csv | return\_id(int), order\_id(int), product\_id(int), return\_date(date), reason(string), amount\_refunded(float) | Records information about product returns. |
| customer\_product\_ratings.csv | customerproductrating\_id(int), customer\_id(int), product\_id(int), ratings(float), review(string), sentiment(string) | Captures product reviews and ratings submitted by customers. |
| campaign\_product\_subcatgeory.csv | campaign\_product\_subcategory\_id(int), campaign\_id(int), subcategory\_id(int), discount(float) | Maps marketing campaigns to specific product subcategories and the discounts applied. |

**Data Warehouse Schema**

The data warehouse for the Shop Smart e-commerce project is designed using a Snowflake Schema to ensure high performance, data normalization, and scalability in analytical workloads. The Snowflake model is an extension of the star schema, where dimension tables are further normalized into multiple related tables. This reduces data redundancy, improves consistency, and enables more efficient storage.

In this project, the schema is divided into dimension tables, which hold descriptive, lookup-type data, and fact tables, which store measurable, transactional data. This separation provides a clear and logical framework for analytics, where users can slice and dice metrics (like revenue, quantity, returns, etc.) based on dimensions (like customer, product, supplier, campaign, etc.).

Key design elements include:

* 5 Fact Tables: Representing orders, order items, product ratings, returns and campaign product subcategory.
* 7 Dimension Tables: Capturing descriptive details for customers, products, categories, subcategories, suppliers, marketing campaigns and payment method.

This architecture enables the business to perform comprehensive analysis on various KPIs including customer behaviour, product performance, campaign success, payment trends, and product return rates. The schema supports both historical and incremental loading and is optimized for SSIS-based ETL processes.

**Data Warehouse Schema Diagram**

[DimMarketingCampaign] [FactOrders] [DimPaymentMethod]

[FactCampaignProductSubcategory] [FactReturns] [FactOrderItems]

[DimProduct] [DimCustomer]

[DimSubcategory] [FactProductRatings] [DimSupplier]

[DimCategory]

**Dimension Tables**

This section provides details about each dimension table used in the Shop Smart data warehouse. Dimension tables store descriptive attributes related to fact data and are crucial for analytical queries. Each table description includes the source file, fields, and transformations applied during the ETL process.

**1. DimCustomer**

* **Source**: customer.csv
* **Columns**: customer\_id, first\_name, last\_name, email, country
* **Transformation**:
  + Trimmed whitespace from names and email
  + Validated email format
  + Converted country names to title case
  + Removed duplicates

**2. DimProduct**

* **Source**: product.csv
* **Columns**: product\_id, name, price, description, subcategory\_id
* **Transformation**:
  + Removed HTML tags or special characters from description
  + Converted prices to float
  + Trimmed string fields
  + Checked foreign key integrity with subcategory

**3. DimSubcategory**

* **Source**: subcategory.csv
* **Columns**: subcategory\_id, subcategory\_name, category\_id
* **Transformation**:
  + Normalized subcategory names
  + Removed duplicates
  + Validated foreign key with category

**4. DimCategory**

* **Source**: category.csv
* **Columns**: category\_id, category\_name
* **Transformation**:
  + Title-cased category names
  + Removed duplicates
  + Trimmed string fields

**5. DimSupplier**

* **Source**: supplier.csv
* **Columns**: supplier\_id, supplier\_name, email
* **Transformation**:
  + Validated and standardized email addresses
  + Trimmed and formatted supplier names
  + Removed null or invalid entries

**6. DimPaymentMethod**

* **Source**: payment\_method.csv
* **Columns**: payment\_method\_id, payment\_method
* **Transformation**:
  + Standardized naming (e.g., Credit Card, PayPal)
  + Removed special characters
  + Trimmed and formatted strings

**7. DimMarketingCampaign**

* **Source**: marketing\_campaigns.csv
* **Columns**: campaign\_id, campaign\_name, offer\_week
* **Transformation**:
  + Standardized date formats for offer\_week
  + Trimmed and capitalized campaign\_name
  + Removed duplicates

**Fact Table**

Fact tables store measurable, quantitative data about business processes. In the Shop Smart SSIS ETL Project, five fact tables have been created. Each one is described below, including source files, key columns, and transformations applied during the ETL process.

**1. FactOrders**

* **Source**: orders.csv
* **Columns**:
  + order\_id\_surrogate, order\_id, order\_date, amount
  + campaign\_id, customer\_id, payment\_method\_id
* **Transformation**:
  + Converted order\_date to datetime format
  + Validated customer\_id, campaign\_id, payment\_method\_id as foreign keys
  + Filtered invalid or incomplete records
  + Ensured amount is a positive value

**2. FactOrderItems**

* **Source**: orderitem.csv
* **Columns**:
  + orderitem\_id, quantity, subtotal, discount
  + supplier\_id, order\_id, product\_id
* **Transformation**:
  + Validated order\_id, product\_id, and euplerid as foreign keys
  + Removed duplicate entries
  + Checked discount ranges (0–100%)
  + Cleaned malformed or null entries

**3. FactReturns**

* **Source**: returns.csv
* **Columns**:
  + return\_id, return\_date, reason, amount\_refunded
  + order\_id, product\_id
* **Transformation**:
  + Converted return\_date to datetime format
  + Filtered out negative refund amounts
  + Mapped order\_id and product\_id as foreign keys
  + Standardized reason text

**4. FactCustomerRatings**

* **Source**: customer\_product\_ratings.csv
* **Columns**:
  + customerproductrating\_id, ratings, review, sentiment
  + customer\_id, product\_id
* **Transformation**:
  + Normalized ratings values between 0 and 5
  + Cleaned and tokenized review text
  + Standardized sentiment to "Positive", "Negative", "Neutral"
  + Validated customer\_id and product\_id as foreign keys

**5. FactCampaignSubcategory**

* **Source**: campaign\_product\_subcategory.csv
* **Columns**:
  + campaign\_product\_subcategory\_id, discount
  + campaign\_id, subcategory\_id
* **Transformation**:
  + Ensured valid links to dim\_Campaign and dim\_Subcategory
  + Converted discount to numeric format and validated range
  + Removed duplicates and null entries

**SSIS Packages**

In this section, we detail the design, purpose, and flow of each SSIS (SQL Server Integration Services) package developed for the Shop Smart ETL Project. Each package is modular, focusing on specific tasks such as loading dimension or fact tables. The packages ensure data validation, error handling, transformation logic, and logging are consistently applied.

**8.1 LoadDimensions.dtsx**

**Purpose:**

This package extracts data from source CSV files and loads the cleaned data into all 7-dimension tables: DimCustomer, DimProduct, DimCategory, DimSubcategory, DimSupplier, DimPaymentMethod, and DimMarketingCampaign.

**Control Flow:**

* Execute SQL Task → Truncate all dimension staging tables
* Data Flow Tasks → One per dimension table
* Execute SQL Task → Insert audit logs



Figure 1: Load Dimensions Control Flow

**8.1.1 DimCustomer**

**Data Flow:**

This data flow loads customer data from customer.csv into the StgCustomer table. It begins by handling nulls and performing type conversions, followed by formatting customer names and emails. Data is then validated, sorted, and separated based on email validity. Valid records are inserted into the staging table, while invalid records are logged for error review.

**Transformations:**

* Handle Null & Empty: Cleanses incoming data by replacing or flagging null or empty fields like email or names.
* Data Conversion: Converts source data types to match the expected types in the staging table (e.g., string to int or float).
* CustomerID Formation: Standardizes the format of customer\_id, ensuring consistent identifiers across the system.

Like: "CUST\_ID\_" + RIGHT ("000" + customer\_id,3)

* Proper Case Format: Converts names to proper case (e.g., "john doe" → "John Doe") to improve readability and consistency.
* New Added Column: Adds calculated or audit columns such as Full Name
* Sort CustomerID: Sorts data by customer\_id for consistency before validation and loading.
* Script Component – Email Validation: Validates email syntax using a script component to ensure correct formatting (e.g., @domain.com check).
* Conditional Split – Based on Email Validity: Splits the flow into valid and invalid email paths. Valid records go to staging and invalid ones go to error logging.

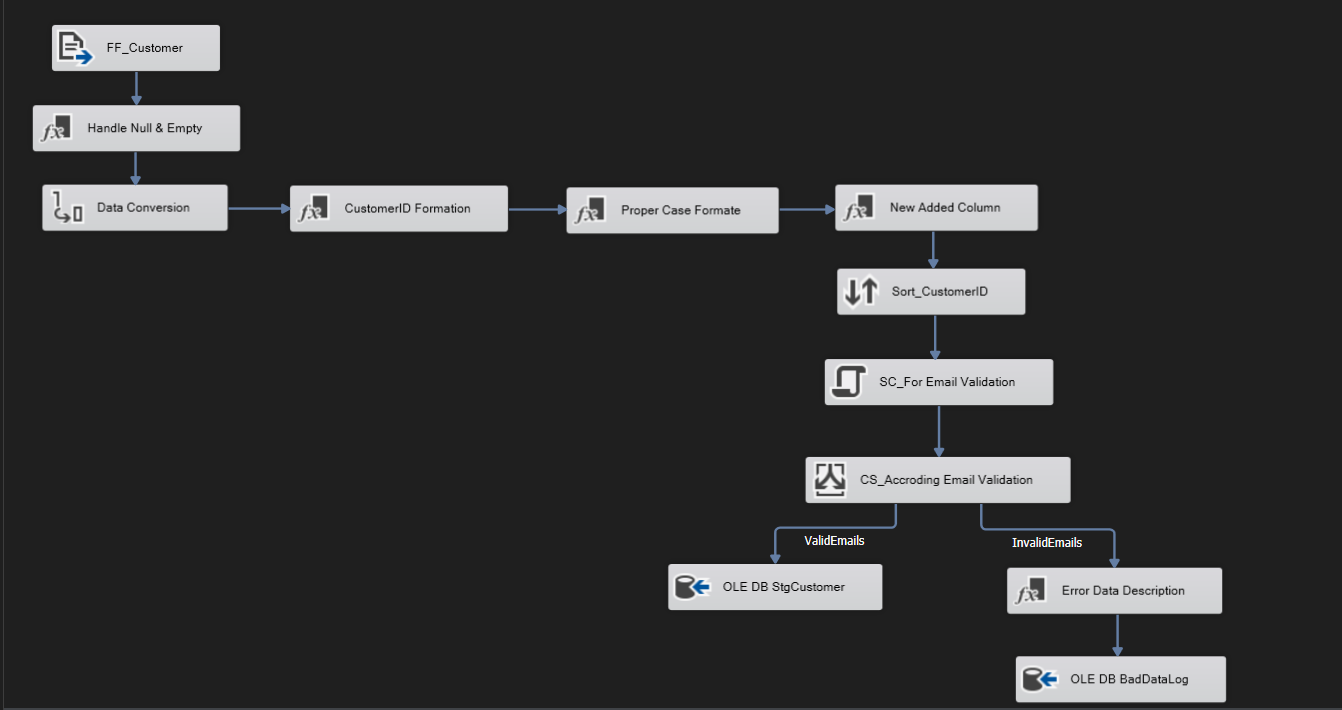
**Logging & Error Handling:**

Valid Emails: Inserted into the staging table StgCustomer using OLE DB Destination.

Invalid Emails: Sent to a derived column task (Error Data Description) to annotate the error reason, then logged into BadDataLog table for review.

Logs help in tracing malformed or missing data entries and allow future correction or cleanup tasks.

* Rows with missing foreign keys are redirected to error output
* Errors are logged using the script in Log\_Table.sql
* Audit info (row count, date/time) is captured

 Figure 2: Dim Customer Data Flow

**8.1.2 DimSupplier**

**Data Flow:**

This data flow loads supplier data from supplier.csv (Flat File Source: FF\_Supplier) into the StgSupplier table. It starts by handling null values and performing data type conversions. Supplier IDs are formatted, names are standardized to proper case, and emails are validated. Data is then conditionally split based on email validity. Valid entries are inserted into the staging table, while invalid ones are logged for error tracking.

**Transformation:**

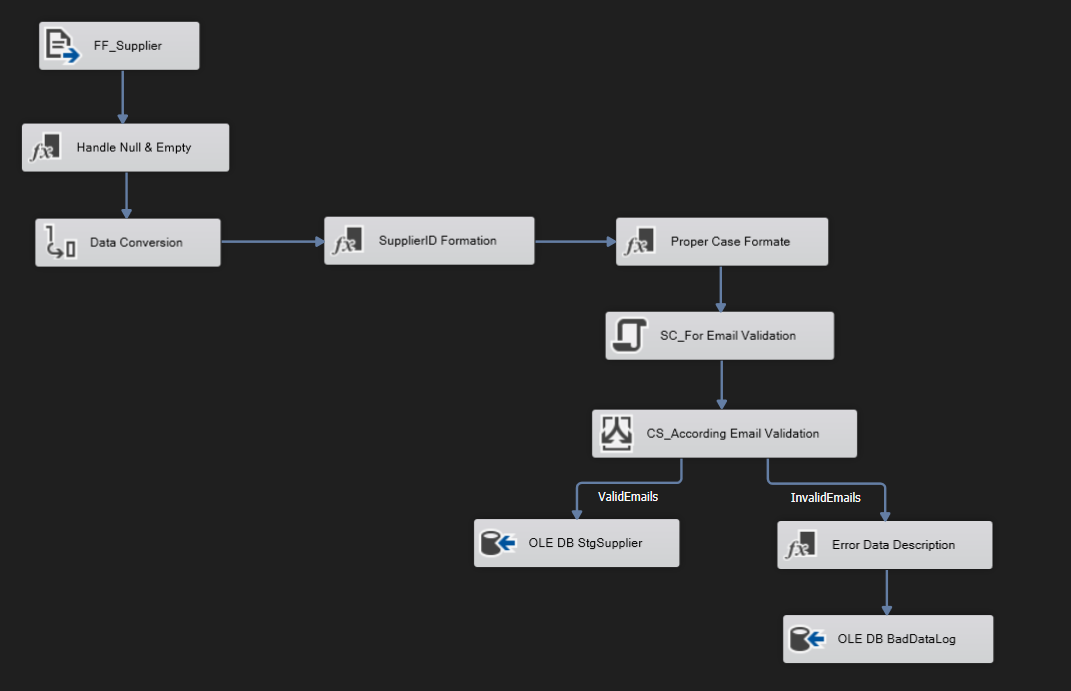
* Handle Null & Empty: Cleanses incoming data by replacing or flagging null/empty values (e.g., email, supplier name).
* Data Conversion: Converts data types from the flat file to match SQL Server types expected in the staging table.
* SupplierID Formation: Formats supplier identifiers into a consistent structure (eg. : "SUP\_ID\_" + RIGHT ("000" + supplier\_id,3))
* Proper Case Format: Standardizes name fields to proper case for readability (e.g., “acme ltd” → “Acme Ltd”).
* Script Component – Email Validation (SC\_For Email Validation): Uses custom script logic to validate email format (e.g., checks presence of @ and .com).
* Conditional Split – Based on Email Validity (CS\_According Email Validation)  
  Splits the flow into valid and invalid email paths. Valid records go to staging and invalid ones go to error logging.
* Error Data Description (Derived Column): Annotates invalid records with error reason (e.g., “Invalid Email Format”).

**Logging & Error Handling:**

Valid Emails: Inserted into StgSupplier using an OLE DB Destination.

Invalid Emails: Logged with description into BadDataLog for audit and correction.

Audit Logging: Additional logging may include row counts, timestamps, and error types.

****Figure 3: Dim Supplier Data Flow

**8.1.3 DimCategory**

**Data Flow:**

This data flow loads category data from category.csv (Flat File Source: FF\_Category) into the StgCategory staging table. It focuses on forming clean and readable category identifiers and standardizing the name format before loading into the staging database.

**Transformation:**

* CategoryID Formation: Generates standardized category identifiers, such as:  
  "CAT\_ID\_" + RIGHT ("000" + category\_id,3) This ensures a consistent ID format across the data warehouse.
* Proper Case Format: Converts category names to proper case (e.g., "electronic devices" → "Electronic Devices") for consistency and readability.

**Logging & Error Handling:**

OLE DB Destination – StgCategory: Transformed category records are loaded into the staging table StgCategory.

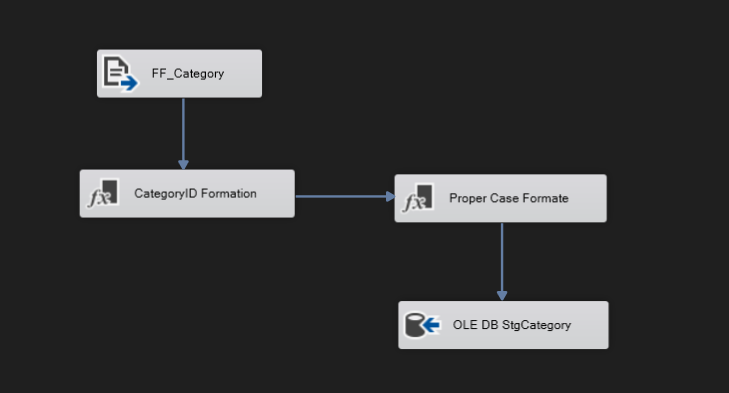


Figure 4: Dim Category Data Flow

**8.1.4 DimSubcategory**

**Data Flow:**

This data flow extracts subcategory data from subcategory.csv (Flat File Source: FF\_Subcategory) and loads it into the StgSubcategory table. It includes detailed validation, error handling, and lookup processes to ensure accurate data integrity with related categories.

**Transformation:**

* ID Formation: Generates standardized subcategory identifiers (e.g., "SUB\_CAT\_ID\_" + RIGHT ("000" + subcategory\_id,3)).
* Proper Case Format: Converts subcategory names into proper case for consistent formatting and readability.
* CS\_Check\_CategoryID\_NULL: Conditional Split to handle records with:
  + NullCategoryID: CategoryID is NULL
  + NonNullCategoryID: CategoryID has a valid value
* LU\_For CategoryID: Lookup transformation to verify if the given CategoryID exists in the reference Category table.
* ErrorDescription\_CategoryID: Adds error details to records where CategoryID is either NULL or not found in lookup.
* UA\_ValidID and NULL CategoryID: Union All transformation that merges:
  + Valid CategoryID records from Lookup Match Output
  + Records with NULL CategoryID (if allowed for business reasons)

**Logging & Error Handling:**

OLE DB Destination: StgSubcategory: Loads validated and transformed subcategory records into the StgSubcategory staging table.

OLE DB BadDataLog: Captures and stores invalid records (with NULL or unmatched CategoryIDs) along with error descriptions for auditing or debugging purposes.

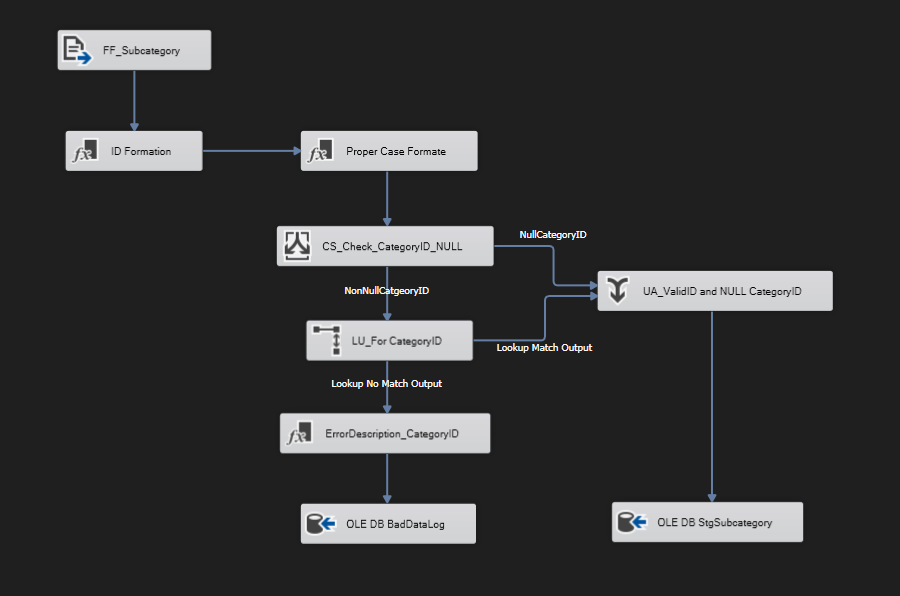


Figure 5: Dim Subcategory Data Flow

**8.1.5 DimProduct**

**Data Flow:**

This data flow loads product data from a source file (FF\_Product) into the StgProduct staging table. The flow starts by converting data types to match the staging table’s requirements, followed by cleansing and standardizing product information. Key steps include handling missing or empty values, ensuring a standardized product ID, and formatting product names to proper case.

**Transformation:**

* Handle Null & Empty: Replaces or flags null or empty fields in the product data, ensuring consistency in required fields like product name or description.
* Data Conversion: Converts incoming data to match the destination types in the StgProduct table (e.g., converting numeric fields to integers, text fields to appropriate string lengths).
* ProductID Formation: Standardizes product identifiers by forming a consistent product ID (e.g., "PROD\_ID\_" + RIGHT ("000" + product\_id,3)).
* Proper Case Format: Converts product names and descriptions to proper case to ensure uniform formatting.
* OLE DB Destination: Loads the cleansed and transformed product data into the StgProduct table in the staging database.

**Logging & Error Handling:**

* Data with missing or invalid entries (e.g., empty required fields) can be redirected to error output if configured in the Data Conversion or Handle Null transformations.
* Errors can be logged in a similar way as the customer data flow, by using a script or a Derived Column to annotate the error reason, and writing invalid records to a BadDataLog table for future review.
* Audit information, such as row counts and load timestamps, should be captured to support data lineage and troubleshooting.

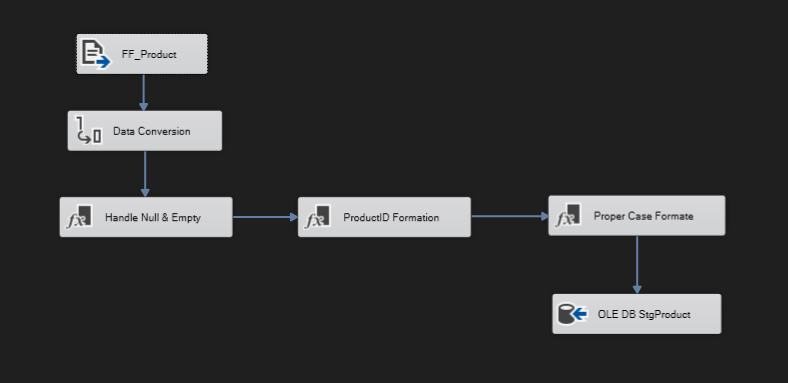


Figure 6: Dim Product Data Flow

**8.1.6 DimPaymentMethod**

**Data Flow:**

This data flow loads payment method data from a source file (FF\_PaymentMethod) into the StgPaymentMethod staging table. The flow starts by converting ID for proper Id formation, followed by formatting product names to proper case.

**Transformation:**

* PaymentMethodID Formation: Standardizes product identifiers by forming a consistent ID (e.g., "PAY\_MET\_ID\_" + RIGHT ("00" + payment\_method\_id,2)).
* Proper Case Format: Converts product names and descriptions to proper case to ensure uniform formatting.
* OLE DB Destination: Loads the cleansed and transformed product data into the StgProduct table in the staging database.

**Logging & Error Handling:**

OLE DB Destination – StgPaymentMethod: Transformed PaymentMethod records are loaded into the staging table.

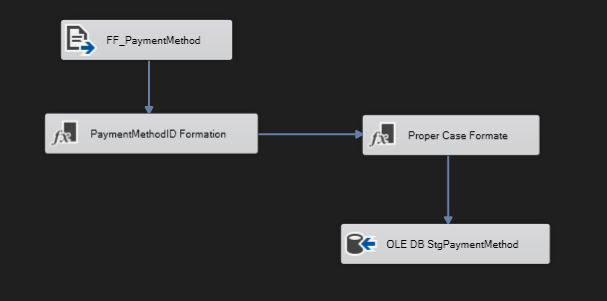


Figure 7: Dim Payment Method Data Flow

**8.1.7 DimMarketingCampaign**

**Data Flow:**

This data flow loads marketing Campaign data from a source file (FF\_MarketingCampaign) into the StgMarketingCampaign staging table. The flow starts by converting ID for proper Id formation, followed by formatting campaign names to proper case using script component and after that at last data conversion for match data type properly into data warehouse.

**Transformation:**

* MarketingCampaignID Formation: Standardizes product identifiers by forming a consistent ID (e.g., "CMP\_ID\_" + RIGHT ("00" + campaign\_id,2)
* Proper Case Format: Converts product names and descriptions to proper case to ensure uniform formatting. Using script.
* Data conversion: For set proper match data type as per data warehouse.
* OLE DB Destination: Loads the cleansed and transformed product data into the StgProduct table in the staging database

**Logging & Error Handling:**

OLE DB Destination – StgMarketingCampaigns: Transformed records are loaded into the staging table.

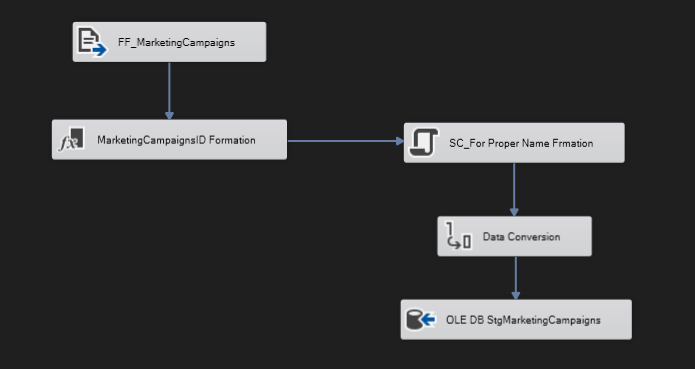


Figure 8: Dim Marketing Campaign Data Flow

**8.2 LoadFactOrders.dtsx**

**Purpose:**

This package extracts data from source CSV files and loads the cleaned data into all 3-fact tables: FactOrders, FactOrderItem, FactReturns

**Control Flow:**

* Execute SQL Task → Truncate all fact staging tables
* Data Flow Tasks → One per fact table
* Execute SQL Task → Insert audit logs

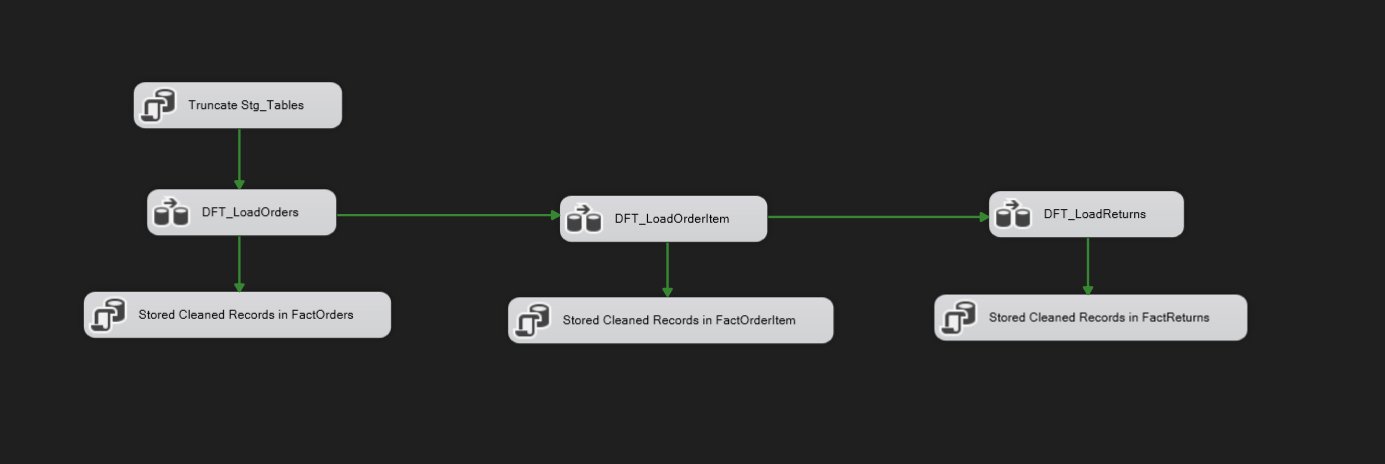


Figure 9: LoadFactOrders Control Flow

**8.2.1 FactOrders**

**Data Flow:**

This data flow loads order data from a source file (FF\_LoadOrders) into the StgFactOrders staging table. It includes extensive validation and error handling, particularly around foreign keys (CustomerID, CampaignID, and PaymentMethodID). The flow ensures data consistency and integrity by handling nulls, performing data type conversions, forming IDs, and verifying foreign key references before inserting valid records into the staging table. Invalid records are logged for error review.

**Transformation:**

* NULL Handling: Replaces or flags null or empty fields in the incoming order data to ensure that required fields are properly filled.
* Data Conversion: Converts source data types to match the destination table’s data types (e.g., numeric conversions, date parsing).
* ID’s Formation: Standardizes the format of order identifiers or any related keys, such as zero-padding numeric IDs.
* CS\_Check\_\*\_NULL: Conditional Splits for each foreign key (CustomerID, CampaignID, PaymentMethodID). These tasks check for null or missing foreign key references and split the flow accordingly.
  + E.g., CS\_Check\_CustomerID\_NULL splits rows into those with a valid/non-null CustomerID and those with a null CustomerID.
* LU\_For \* ID: Lookup transformations validate foreign keys against their respective dimension tables (e.g., LU\_For CustomerID, LU\_For CampaignID, LU\_For Payment\_method\_id). This ensures the referenced keys exist in the dimension tables.
  + Lookup Match Output: If a match is found, the foreign key is valid.
  + Lookup No Match Output: If no match is found, the record is considered invalid and is redirected to error handling.
* ErrorDescription\_\*: Derived Columns that annotate the error reason for missing or invalid foreign keys.
  + E.g., ErrorDescription\_CustomerID, ErrorDescription\_CampaignID, and ErrorDescription\_PaymentMethodID provide a human-readable description of the error for logging.
* UA\_ValidID and Null \* ID: Union All transformations that consolidate valid foreign key records and those with null references. These form the final set of valid records.
* UA\_ErrorRecords: Union All that gathers all invalid records (from various lookup error paths) and consolidates them for error logging.
* OLE DB Destination (BadDataLog): All invalid records are written to a BadDataLog table for future review and correction.
* OLE DB Destination (StgFactOrders): Loads the validated and clean order data into the staging table StgFactOrders.

**Logging & Error Handling:**

* Foreign key lookups (CustomerID, CampaignID, PaymentMethodID) ensure referential integrity by verifying that each foreign key exists in the dimension tables.
* Errors are annotated using Derived Columns and written to BadDataLog for analysis.
* Audit information (row count, load timestamp, etc.) should be captured for monitoring and troubleshooting.

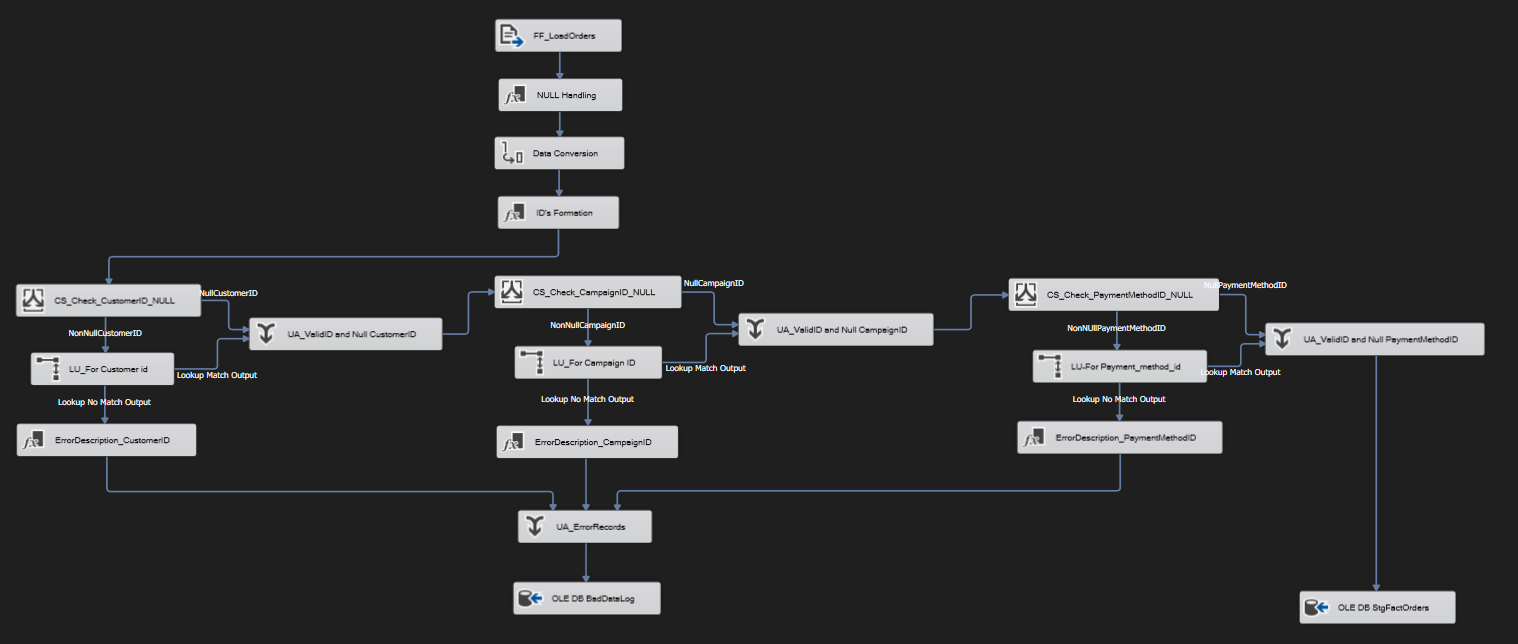


Figure 10: Fact Orders Data Flow

**8.2.2 FactOrderItem**

**Data Flow:**

This data flow loads order item data from a source file (FF\_Orderitem) into the StgFactOrderitem staging table. It rigorously validates foreign key references (OrderID, ProductID, SupplierID) and logs any records that fail these checks. Like the FactOrders load, it ensures data consistency and completeness before inserting the validated records into the staging table.

**Transformation:**

* ID Formations: Standardizes and forms the IDs in the dataset for consistency and integration.
* Data Conversion: Converts source data types (e.g., string to integer, date formats) to match the destination table’s data types.
* CS\_Check\_\*\_NULL: Conditional Splits for checking each foreign key’s presence.
  + CS\_Check\_OrderID\_NULL
  + CS\_Check\_ProductID\_NULL
  + CS\_Check\_SupplierID\_NULL
* Each Conditional Split routes rows with non-null IDs to the lookup transformations for foreign key validation, and routes nulls to be flagged.
* LU\_For \* ID: Lookup transformations validate the presence of foreign keys in their respective dimension tables:
  + LU\_For OrderID
  + LU\_For ProductID
  + LU\_For SupplierID
* Lookup No Match Output: Rows with invalid foreign keys are annotated with an error description (e.g., ErrorDescription\_OrderID, ErrorDescription\_ProductID, ErrorDescription\_SupplierID).
* ErrorDescription\_\*: Derived Columns that record error descriptions for downstream error handling and logging.
* UA\_ValidID and NULL \* ID: Union All transformations combine:
  + Rows with valid foreign keys.
  + Rows with NULL foreign key references (sometimes valid depending on business rules).
* UA\_ErrorRecords: Union All transformation that aggregates all rows with invalid foreign keys (from no match outputs).
* OLE DB Destination (BadLogData): Stores these error records into a BadLogData table for further investigation and correction.
* OLE DB Destination (StgFactOrderitem): Loads the fully validated records (including rows with valid foreign keys and those with acceptable nulls) into the staging table.

**Logging & Error Handling:**

* Each foreign key reference (OrderID, ProductID, SupplierID) is rigorously validated.
* Lookup failures generate detailed error descriptions using Derived Columns.
* All invalid records are logged to BadLogData, providing a robust error tracking mechanism.
* Valid records are consolidated and inserted into the staging fact table StgFactOrderitem.

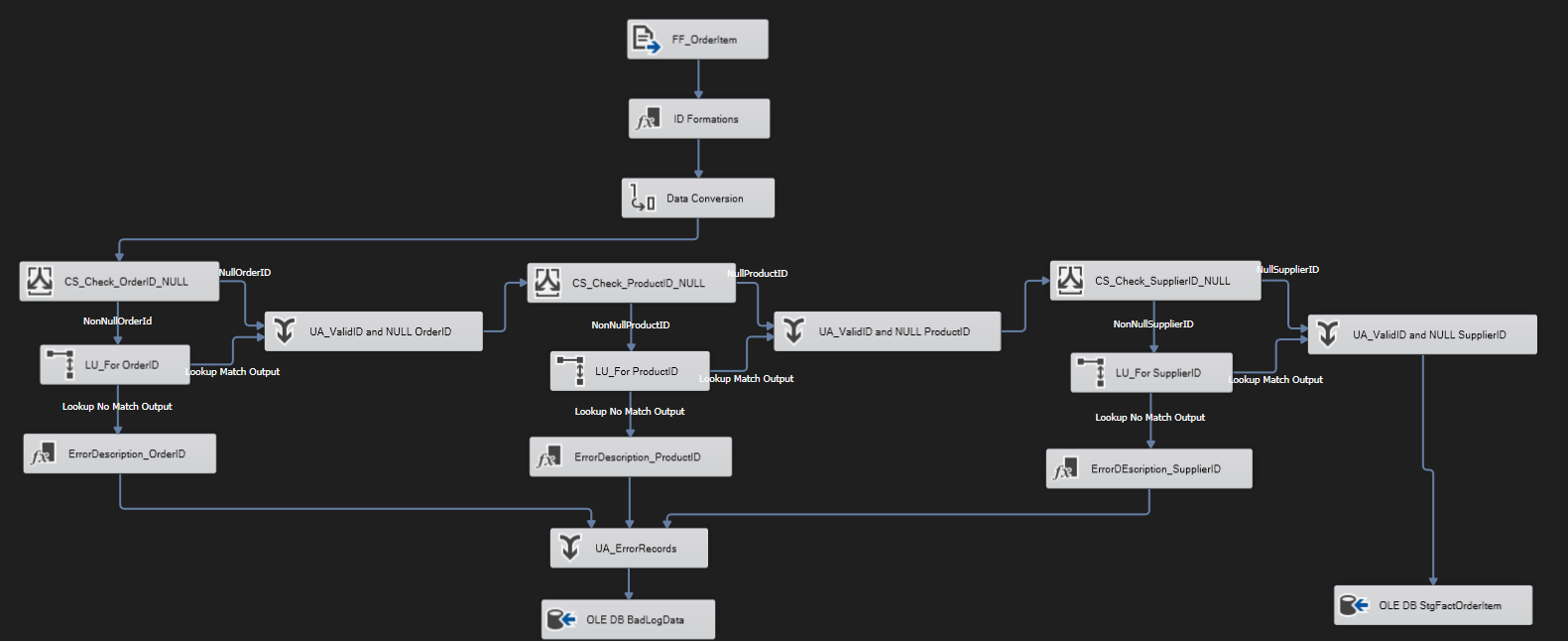


Figure 11: Fact Order Item Data Flow

**8.2.3 FactReturns**

**Data Flow:**

This data flow loads returns data from the source file (FF\_Returns) into the StgFactReturns staging table, ensuring the integrity and completeness of the data by validating the foreign key references (OrderID and ProductID) and managing error records.

**Transformation:**

* ID’s Formation: This step standardizes and structures the IDs for downstream processing, ensuring consistency.
* Data Conversion: Converts data types in the source file to match the data types in the destination table.
* CS\_Check\_OrderID\_NULL: Separates rows with a NULL OrderID and those with a non-NULL OrderID.
  + NonNullOrderID rows flow to LU\_For OrderID for validation.
  + NullOrderID rows are routed directly to UA\_ValidID and NULL OrderID.
* LU\_For OrderID: Validates the OrderID by checking against the corresponding dimension table. Valid OrderIDs proceed to the valid ID union (UA\_ValidID and NULL OrderID). Invalid OrderIDs are flagged and routed to ErrorDescription\_OrderID for error annotation.
* ErrorDescription\_OrderID: Adds a descriptive error message for rows with invalid OrderIDs for further analysis and logging.
* CS\_Check\_ProductID\_NULL: Separates rows with a NULL ProductID and those with a non-NULL ProductID.
  + NonNullProductID rows flow to LU\_For ProductID for validation.
  + NullProductID rows are routed directly to UA\_ValidID and NULL ProductID.
* LU\_For ProductID: Validates the ProductID by checking against the product dimension table.
  + Valid ProductIDs proceed to the valid ID union (UA\_ValidID and NULL ProductID).
  + Invalid ProductIDs are flagged and routed to ErrorDescription\_ProductID.
* ErrorDescription\_ProductID: Adds a descriptive error message for rows with invalid ProductIDs for downstream error handling.
* UA\_ValidID and NULL OrderID: Union All transformation that consolidates rows with valid OrderIDs and those with NULL OrderIDs (if allowed).
* UA\_ValidID and NULL ProductID: Union All transformation that consolidates rows with valid ProductIDs and those with NULL ProductIDs (if allowed).
* UA\_ErrorRecords: Union All transformation that aggregates all error records from invalid OrderIDs and ProductIDs.
* OLE DB Destination (BadLogData): Stores all error records (invalid OrderID or ProductID) in the BadLogData table, providing a robust error handling mechanism.
* OLE DB Destination (StgFactReturns): Inserts fully validated data (including valid rows and rows with acceptable NULL foreign keys) into the StgFactReturns staging table.

**Logging & Error Handling:**

* The process logs detailed error descriptions for both OrderID and ProductID validation failures, supporting thorough error tracking and resolution.
* Valid records and acceptable null foreign key rows are inserted into the staging table.
* Invalid records are captured in the BadLogData table for further investigation and correction.

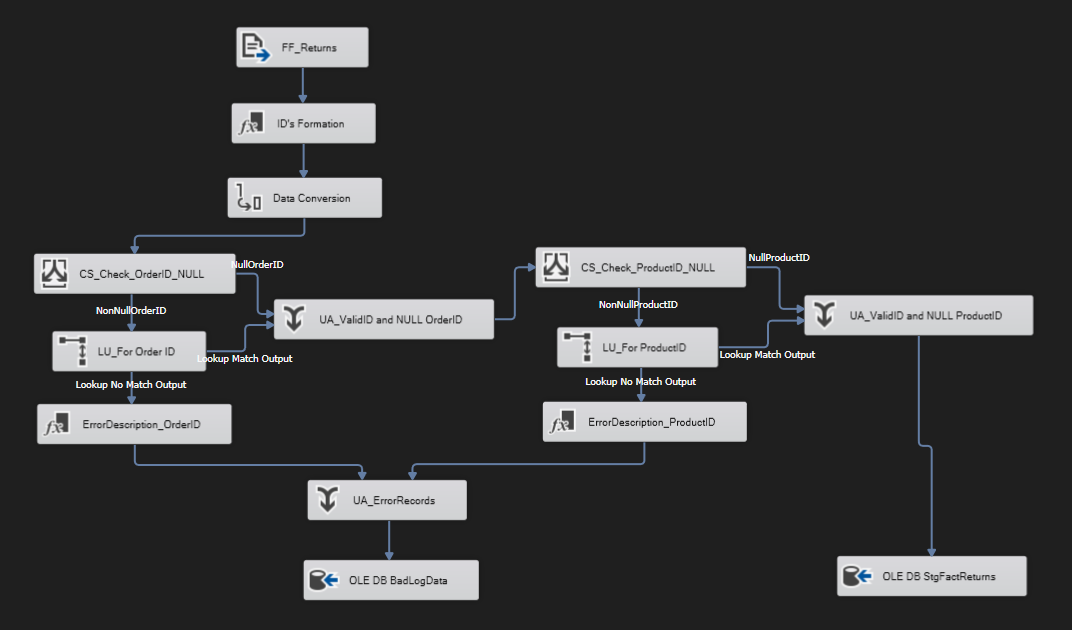


Figure 12: Fact Returns Data Flow

**8.3 LoadFactRating.dtsx**

**Purpose:**

This package extracts data from source CSV files and loads the cleaned data into all 1-fact tables: FactCustomerRating

**Control Flow:**

* Execute SQL Task → Truncate all fact staging tables
* Data Flow Tasks → One per fact table
* Execute SQL Task → Insert audit logs

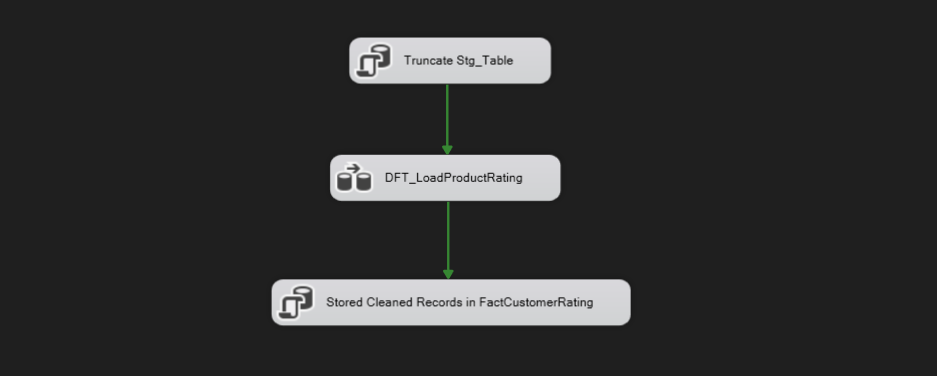


Figure 13: Load Rating Control Flow

**8.3.1 FactCustomerRating**

**Data Flow:**

This data flow processes and loads customer rating data from the source file (FF\_CustomerRating) into the StgFactCustomerRating staging table, performing comprehensive validation for CustomerID and ProductID foreign keys and managing any erroneous records.

**Transformations:**

* ID’s Formation: Standardizes and formats IDs to ensure consistency for downstream processing.
* Data Conversion: Converts data types in the source file to align with the staging table’s data types.
* Derived Column: Adds or modifies columns based on expressions or logic, if needed (e.g., calculating rating score adjustments or adding metadata).
* CS\_Check\_CustomerID\_NULL: Checks for NULL CustomerID values.
  + NonNullCustomerID rows are routed to the LU\_For CustomerID lookup for validation.
  + NullCustomerID rows are routed directly to UA\_ValidID and Null CustomerID.
* LU\_For CustomerID: Validates CustomerID against the customer dimension table.
  + Valid rows flow to the UA\_ValidID and Null CustomerID.
  + Invalid CustomerIDs are routed to ErrorDescription\_CustomerID for annotation.
* ErrorDescription\_CustomerID: Adds descriptive error messages for records with invalid CustomerID values.
* CS\_Check\_ProductID\_NULL: Checks for NULL ProductID values.
  + NonNullProductID rows are routed to LU\_For ProductID lookup for validation.
  + NullProductID rows go directly to UA\_ValidID and Null ProductID.
* LU\_For ProductID: Validates ProductID against the product dimension table.
  + Valid rows flow to the UA\_ValidID and Null ProductID.
  + Invalid ProductIDs are routed to ErrorDescription\_ProductID.
* ErrorDescription\_ProductID: Adds descriptive error messages for records with invalid ProductID values.

**Logging & Error Handling:**

* UA\_ValidID and Null CustomerID / UA\_ValidID and Null ProductID: Union All transformations that merge valid and NULL CustomerID and ProductID rows, respectively, for downstream processing.
* UA\_ErrorRecords: Aggregates all error records (invalid CustomerIDs and ProductIDs) into a unified flow for error handling and storage.
* OLE DB Destination (BadLogData): Stores all error records into the BadLogData table for further investigation and resolution.
* OLE DB Destination (StgFactCustomerRating): Loads fully validated customer rating data (including those with acceptable NULL CustomerID or ProductID) into the StgFactCustomerRating staging table.

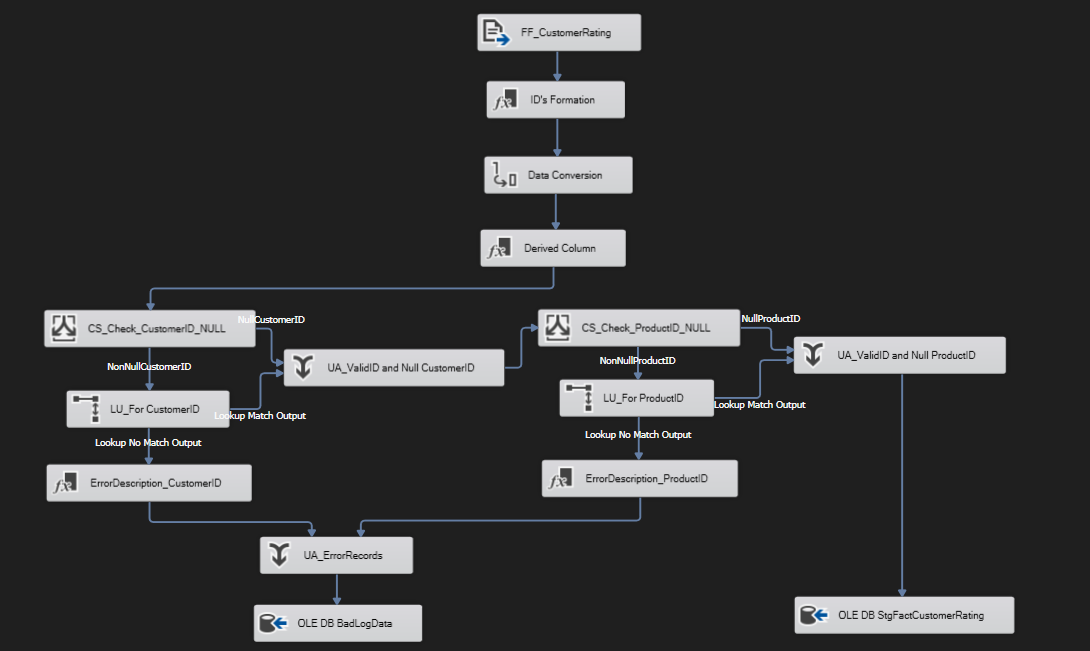


Figure 14: Fact Customer Rating Data Flow

**8.4 LoadFactCampaignProductSubcategory.dtsx**

**Purpose:**

This package extracts data from source CSV files and loads the cleaned data into all 3-fact tables: FactOrders, FactOrderItem, FactReturns

**Control Flow:**

* Execute SQL Task → Truncate all fact staging tables
* Data Flow Tasks → One per fact table
* Execute SQL Task → Insert audit logs

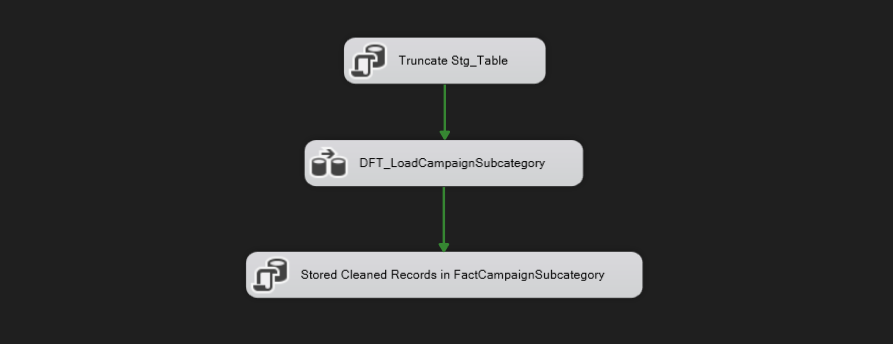


Figure 15: Load Fact Campaign Product Subcategory Control Flow

**8.4.1 FactCampaignProductSubcategory**

**Data Flow:**

This ETL pipeline extracts campaign subcategory data from the source file (FF\_CampaignSubcategory), performs data cleaning and validation, and loads validated records into the StgFactCampaignSubcategory table. Invalid data is logged for further review.

**Transformations:**

* NULL Handling: Cleans up NULL values in the source file to ensure proper downstream processing.
* Data Conversion: Converts source column data types to match the staging table’s expected data types.
* ID’s Formation: Constructs or standardizes IDs (e.g., trimming or concatenating fields to create a composite key if necessary).
* CS\_Check\_CampaignID\_NULL: Conditional Split that identifies rows with NULL CampaignIDs and separates them from rows with valid IDs.
  + NullCampaignID rows go directly to UA\_ValidID and NULL CampaignID.
  + NonNullCampaignID rows proceed to the lookup transformation for validation.
* LU\_For CampaignID: Looks up CampaignID in the campaign dimension table to confirm it’s valid.
  + Lookup Match Output (valid CampaignIDs) goes to UA\_ValidID and NULL CampaignID.
  + Lookup No Match Output (invalid CampaignIDs) proceeds to ErrorDescription\_CampaignID.
* ErrorDescription\_CampaignID: Adds descriptive error messages for invalid CampaignID records for easy diagnosis.
* CS\_Check\_SubCategoryID\_NULL: Conditional Split that separates rows with NULL SubCategoryID values.
  + NullSubCategoryID rows go directly to UA\_ValidID and NULL SubCategoryID.
  + NonNullSubCategoryID rows are validated in the lookup step.
* LU\_For SubCategoryID: Validates SubCategoryID by looking it up in the subcategory dimension table.
  + Lookup Match Output (valid SubCategoryIDs) goes to UA\_ValidID and NULL SubCategoryID.
  + Lookup No Match Output (invalid SubCategoryIDs) goes to ErrorDescription\_SubCategoryID.
* ErrorDescription\_SubCategoryID: Adds descriptive error messages for invalid SubCategoryID records for easier error tracking.

**Logging & Error Handling:**

* Error Description Columns: Dedicated error description transformations (ErrorDescription\_CampaignID and ErrorDescription\_SubCategoryID) provide clear explanations of validation failures.
* BadLogData Table
  + All erroneous records are captured and stored in the BadLogData table.
  + This ensures no loss of data, supporting error analysis and fixes.
* NULL Handling: Explicit pathways handle records with NULL values, ensuring compliance with the data model.
* Consolidated Error Reporting: Errors from both CampaignID and SubCategoryID validation steps are combined and logged for transparency and debugging.

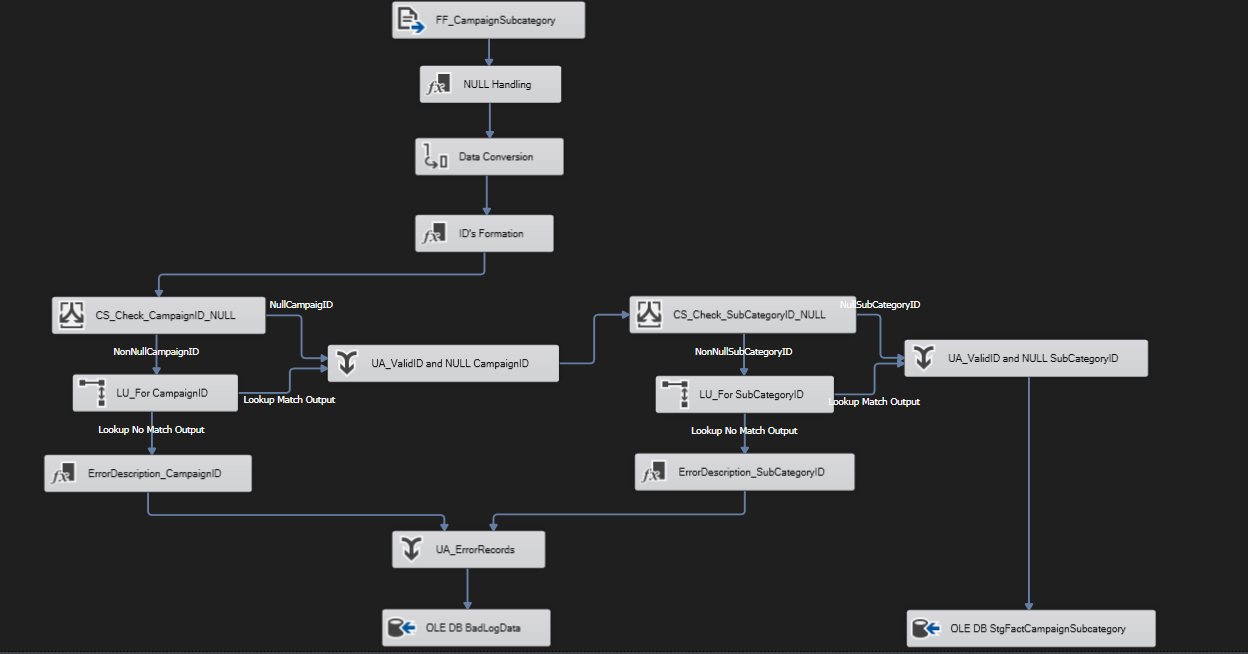


Figure 16: Fact Campaign Product Subcategory Data Flow

**Error Handling**

To ensure robust data quality and traceability within the ETL process, this project incorporates a centralized error logging mechanism using a custom-built SQL table named BadDataLog. This table serves as a unified repository to capture, store, and track all rows of data that fail during the ETL pipeline across multiple flat file sources.

CREATE TABLE BadDataLog (

log\_id INT IDENTITY (1,1) PRIMARY KEY,

table\_name NVARCHAR (100) NOT NULL,

error\_description NVARCHAR(MAX) NOT NULL,

error\_data NVARCHAR(MAX) NOT NULL,

log\_date DATETIME DEFAULT GETDATE()

)

**Purpose and Usage:** The BadDataLog table is designed to record and diagnose errors encountered during the SSIS data flow for different source files (CSV files). Instead of halting the entire ETL process due to data inconsistencies or transformation issues, erroneous rows are redirected to this table for further inspection, ensuring:

* Smooth ETL execution with minimal interruption
* Centralized tracking of data quality issues
* Ease of troubleshooting and validation
* Auditing and transparency for rejected records

**Integration in ETL Process**

In SSIS, each Data Flow Task handling flat file inputs is configured to redirect erroneous rows to a Derived Column transformation that adds appropriate error\_description values. These rows are then inserted into the BadDataLog table via an **OLE DB Destination** component.

For example:

* **Source File:** FF\_Subcategory.csv
* **Error:** CategoryID not found in lookup
* **Logged Info:**
  + table\_name: StgSubcategory
  + error\_description: Invalid CategoryID
  + error\_data: "CategoryID"

**Challenge & Solution**

**Challenge 1: Empty CampaignID in Source Data**

* **Problem**: Some campaign records have a NULL value for OrderID, which could violate referential integrity or lead to orphaned records.
* **Solution**:
  + Used Conditional Split (CS\_Check\_CampaignID\_NULL) to separate NULL and non-NULL values.
  + Allowed NULL values to proceed through a controlled path (Union All) if business logic permits storing subcategories with no associated category.

**Challenge 2: Invalid or Non-Existent CategoryID**

* **Problem**: Some subcategory records reference a CategoryID that does not exist in the parent Category table.
* **Solution**:
  + Implemented Lookup Transformation (LU\_For CategoryID) to validate the CategoryID.
  + Records without a matching CategoryID were redirected to Error Handling flow using ErrorDescription\_CategoryID.

**Challenge 3: Improper Case Formatting of Names**

* **Problem**: Subcategory names were inconsistently cased in the flat file (e.g., uppercase/lowercase/mixed).
* **Solution**:
  + Applied Derived Column Transformation (Proper Case Formate) to standardize the text using proper case logic

**Challenge 4: Tracking and Logging Bad Data**

* **Problem**: Losing invalid records during transformation would make auditing difficult.
* **Solution**:
  + Redirected invalid rows to a dedicated destination: OLE DB BadDataLog.
  + Included descriptive error messages for easier debugging and resolution.

**Challenge 5: Combining Valid and Allowable Null Data**

* **Problem**: Need to combine rows with valid CategoryID and rows with NULL values.
* **Solution**:
  + Used Union All Transformation (UA\_ValidID and NULL CategoryID) to merge both data streams before final loading into the staging table.

**Conclusion**

The completion of this E-Commerce Data Warehousing project marks a significant milestone in enabling data-driven decision-making for the Shop Smart platform. Through the end-to-end implementation of an SSIS-based ETL pipeline, raw and unstructured data from multiple CSV files spanning customers, orders, products, returns, suppliers, campaigns, and more was successfully extracted, validated, transformed, and loaded into a well-organized SQL Server Data Warehouse following a snowflake schema.

Each data domain was carefully handled: customer and supplier details were cleansed and validated for consistency; products and subcategories were transformed for analytical hierarchy; orders, returns, and ratings were linked with campaigns and payment data to generate rich insights across business functions. With the use of modular SSIS packages, we ensured that data quality rules were applied uniformly and that logs were maintained for audit and error tracking through dedicated staging, clean, and log tables.

Special attention was given to common real-world data issues such as null values, type mismatches, duplicate records, and invalid foreign keys. These were systematically resolved using transformations like conditional splits, lookups, data conversions, script components, and derived columns ensuring clean, reliable data loading into both dimension and fact tables.

Moreover, the logging mechanism captured both successful and failed records, providing traceability and easing future debugging. By designing the warehouse with scalability in mind, the system is capable of handling incremental loads, future data expansion, and additional modules as the business grows.

Ultimately, this project not only delivered a technically robust data warehouse but also empowered the business with a foundation to generate advanced dashboards, KPI tracking, and BI reporting. It enables stakeholders to gain meaningful insights into customer behavior, sales performance, campaign effectiveness, and product returns driving strategic planning and operational excellence. This project stands as a comprehensive solution for modern e-commerce data management—scalable, reliable, and built for analytics.