# Technical Design Document (TDD)

## Project: E-commerce Sales Data Processing using Databricks & Delta Lake

### 1. **Overview**

The goal of this project is to build a robust, scalable, and efficient data pipeline in **Databricks** to ingest, process, and analyze daily incoming e-commerce sales data. The pipeline will follow the Medallion Architecture (Bronze, Silver, Gold) using Delta Lake tables and support downstream analytics and reporting for business stakeholders.

### 2. **Data Sources**

| Source | Format | Frequency | Location Example |
| --- | --- | --- | --- |
| Orders | JSON | Daily | /mnt/raw/orders/ |
| Customers | Excel | Daily | /mnt/raw/customers/ |
| Products | CSV | Daily | /mnt/raw/products/ |

All data sources will be ingested as-is into the bronze layer with minimal transformations.

### 3. **Architecture**

Medallion Architecture (Delta Lake)

* **Bronze Layer**: Raw ingestion of data with ingestion metadata (source path, timestamp)
* **Silver Layer**: Cleansed and standardized data, with null checks, type casting, and deduplication
* **Gold Layer**: Aggregated, business-ready tables and pre-aggregated materialized views

### 4. **Bronze Layer Design**

All raw data is stored as Delta tables partitioned by ingest\_date.

| Table Name | Description | Partitioned By |
| --- | --- | --- |
| ecommerce.bronze\_\_orders | Raw orders from JSON | ingest\_date |
| ecommerce.bronze\_\_products | Raw products from CSV | ingest\_date |
| ecommerce.bronze\_\_customer | Raw customers from Excel | ingest\_date |

Each table includes metadata columns:

* ingest\_date: Date the data was ingested
* ingest\_ts: Timestamp of ingestion
* source\_file: Original file path

**Assumptions:**

* Orders data will be ingested on a daily basis and later processed using batch processing
* Mode of write is overwrite with daily partition by ingest\_date

**Out of Scope (current Solution):**

* Logging and audit tables
* Streaming data ingestion
* Slowly Changing Dimensions (SCD) implementation
* Incremental ingestion

### 5. **Silver Layer Design**

Cleansed and enriched tables ready for joining and analytical processing.

| Table Name | Description |
| --- | --- |
| ecommerce.silver\_\_customer | Cleaned customer data, names corrected and deduplicated |
| ecommerce.silver\_\_orders | Orders data with types cast and valid records only |
| ecommerce.silver\_\_products | Cleaned products with proper data types and pricing |
| ecommerce.silver\_\_enrichedOrderDetails | Enriched orders joined with customer and product tables |

#### Silver Transformations:

* Null handling (drop nulls in primary keys)
* De-duplication (using dropDuplicates)
* Casting and rounding (e.g., prices, discounts, profit)
* Date parsing and formatting
* Joining orders with customer and product

#### Silver Layer Unit Testing:

* Verify null filtering on primary keys (e.g., Customer\_ID, Product\_ID)
* Ensure de-duplication logic using sample test cases with duplicate rows
* Validate transformed columns (e.g., correct rounding of prices, correct year derivation)
* Confirm join results contain expected columns and data types

### 6. **Gold Layer Design**

Final business-layer aggregates and KPIs.

| Table Name | Description |
| --- | --- |
| ecommerce.gold\_\_profit\_by\_customer\_product\_year | Aggregated profit by customer, product category, and year |

SQL notebook for ad-hoc queries:

* data\_aggregated\_using\_sql\_gold\_layer
* Contains:
  + Profit by year
  + Profit by year and category
  + Profit by customer
  + Profit by customer + year

#### Gold Layer Unit Testing:

* Validate aggregation accuracy for profit calculations
* Test edge cases with missing customer or product dimensions
* Confirm grouped columns match expected types (e.g., year as Integer)
* Use predefined test data to validate SQL results

### 7. **Tools & Technologies**

* **Databricks**: Unified platform for development and orchestration
* **Delta Lake**: For ACID transactions and scalable data lakes
* **Spark (PySpark)**: Core transformation engine
* **Autoloader (Optional)**: For incremental file ingestion
* **Unity Catalog**: For data governance and access control (if available)

### 8. **Partitioning Strategy**

All bronze and silver tables are partitioned by ingest\_date or order\_year (for time-based access).

### 9. **Data Quality & Testing**

* Unit tests for each transformation (TDD)
* Validate:
  + Column types and schema
  + Null filtering
  + Deduplication logic
  + Aggregation accuracy
* Tools: chispa, pytest
* Silver and Gold layers are tested with both valid and edge case input scenarios

### 10. **Future Enhancements**

* Add CDC-based ingestion (using AutoLoader or DLT)
* Implement Incremental and SCD
* Implement Delta Live Tables for declarative pipelines
* Add DLT expectations or Data Quality Frame work as gate keeper after each Transformation Layer and stored the result in Table for reporting or fixing.
* Integrate with BI tools like Power BI/Tableau for direct Gold Layer access
* Track data lineage using Unity Catalog
* Use Pytest or unittest Module for unit testing coverage

### 11. **Security & Governance**

* Use Unity Catalog for table-level access control
* Use RBAC for workspace permissions
* Store secrets in Azure Key Vault / Databricks Secrets

### 12. **Prerequisites**

* **Excel Reading**:
  + Install library: com.crealytics:spark-excel\_2.12:0.13.5 or compatible version for your Spark runtime.
  + Configure .option("header", "true").option("inferSchema", "true") when reading Excel files.

### 13. **Observation (Not implemented in current solution)**

* Phone numbers are not correct, need lookup table or logic to clean it up.
* Ship Mode does not have uniform values from days\_to\_ship (difference of ship\_date and order\_date). Example: Same day was having 0 and 1, First class was having 1,2,3,4 and second class also having 2,3,4 kind of value. We should have a calculated value in Silver to make it uniform.
* Null product category is there in data (need logics or updated mapping to check it)
* Look up table country,state,city will be good to make changes if corrupt or non-uniform data are coming from source.
* Email ID was not matching with Customer Name (might be data problem)

### 14. Project Folder Structure

project\_root/  
│  
├── bronze\_layer/  
│ └── data\_ingestion\_bronze\_layer.ipynb  
│ # Ingests raw data into Bronze Delta Tables  
│  
├── silver\_layer/  
│ ├── customers\_silver\_layer.ipynb  
│ ├── orders\_silver\_layer.ipynb  
│ ├── products\_silver\_layer.ipynb  
│ └── enrich\_order\_details\_silver\_layer.ipynb  
│ # Cleans and transforms data into Silver layer  
│  
├── gold\_layer/  
│ └── final\_aggregations.sql  
│ # SQL scripts or notebooks with final business-level aggregations  
│  
├── utility\_functions/  
│ ├── silver\_layer\_functions.ipynb  
│ └── gold\_layer\_functions.ipynb  
│ # Shared PySpark functions used in Silver and Gold layers  
│  
├── configs/  
│ └── variables  
│ # Contains path definitions and environment-specific configurations  
│  
├── unit\_testing/  
│ ├── sl\_ut\_clean\_customer\_names.ipynb  
│ ├── sl\_ut\_clean\_orders.ipynb  
│ ├── sl\_ut\_clean\_products.ipynb  
│ ├── sl\_ut\_enrich\_order\_details.ipynb  
│ └── gl\_ut\_aggregated\_profit.ipynb  
│ # Unit tests for Silver and Gold layer transformations  
│  
├── orchestrator/  
│ └── orchestrate\_layer\_stages.ipynb  
│ # Notebook to execute Bronze → Silver → Gold layers end-to-end

**Folder Descriptions:**

* **bronze\_layer/**: Contains the raw data ingestion notebook.
* **silver\_layer/**: Contains notebooks for transforming and cleaning customer, order, and product data.
* **gold\_layer/**: Contains SQL-based final aggregations or reporting layer transformations.
* **utility\_functions/**: Houses reusable PySpark transformation functions for Silver and Gold layers.
* **configs/**: Stores configuration files (like input paths) required during ingestion.
* **unit\_testing/**: Includes detailed unit test notebooks for validating silver and gold layer logic.
* **Orchestor/**: End to end execution

### 15. **Conclusion**

This TDD outlines the design for an e-commerce data pipeline using Databricks and Delta Lake. By leveraging a robust medallion architecture and using TDD and Delta best practices, we ensure scalability, performance, and trustworthiness in our data platform.