**Cutlass**

**Technical Specification**

# Version History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version No. | Change | Section | Author | Date |
| 1.0 | Initial Draft |  | Fowziyah | 19/01/2015 |
| 2.0 | Updated |  | Saikrupa | 30/05/2016 |
| 3.0 | Updated discrepancies |  | Pavithra | 19/06/2018 |
| 4.0 | Azure components included |  | Chitra | 07/12/2018 |
| 5.0 | Modification to remove SSIS and include Data Brick components |  | Aishwarya K | 19/12/2019 |
| 6.0 | Azure DevOps implementation added |  | Thiru | 24/06/2021 |
| 7.0 | Modification to replace SQL JDBC connection with ADF activities (csv) |  | Pranav/ Anmol | 14/12/2021 |
| 8.0 | Removing DevOps Implementation |  | Pranav/ Manan | 27/05/2022 |
| 9.0 | Adding DevOps Implementation |  | Parv/SriHarsha | 09/08/2022 |
| 10.0 | Customer data transformation |  | Akshaya S | 26/12/2022 |
| 11.0 | Moving time implementation to ADB and settings to ADLS |  | Ayesha Isaac | 25/01/2023 |
| 12.0 | Removed SQL components and cleaned up document |  | Hasika Chitloor | 06/06/2023 |
| 13.0 | Added partitioning for delta tables, changed ADF pipeline structure and cleaned up document |  | SriHarsha Kavalipati | 07/05/2024 |

# Introduction

## Purpose

This document details the technical design of the Cutlass system. It is used in the first instance to define what technical components the system will have, how they should be built, what functions should be included and how they relate to each other. It acts as a detailed guide for developers to understand what is required to fulfil the functional specification and develop a coherent system to high quality standards.

The document is something that will change and grow as enhancements are added and change requests are implemented. Once the system is built it will continue to provide value by acting as a reference point for those who will support and maintain the system ongoing.

This document will be signed off as a technical development program by the Thorogood Project Manager and the CPG Ltd Project Manager. It will then be reviewed ongoing and updated at the end of the build phase.

## Scope

This document covers all components that Thorogood are designing and building.

* Azure Data Lake Storage (ADLS)
* Azure Data Factory (ADF)
* Azure Databricks (ADB)
* Reporting Tool (PowerBI)

# System Overview

The system being developed is an end-to-end system that involves loading data from flat files in the source to ADLS folders. The data ingested from source will be processed, converted to Parquet files and transformations will be performed on them. This will involve movement of data across various directories/layers in ADLS. The data stored in the final ADLS layer will then be used to build a report on Power BI for reporting purposes.

CUTLASS ARCHITECTURE

Graphical user interface

Description automatically generated

**Data Sources**

See below table for the list of source data for this system. The data will be in a raw format and will require cleansing.

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Description** | **Delivery Format** | **Frequency** |
| Dimension | Account Data | Files | Adhoc |
| Dimension | Product Data | Files | Adhoc |
| Fact | Invoiced Sales Data | Files | Monthly |

## Source: Product

The external product file will contain the below columns and the data type column indicates the expected data type of the column.

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Other Properties** |
| Source\_System\_Id | String |  |
| Product\_Code | String |  |
| Product\_Desc | String |  |
| Traded\_Unit | String |  |
| Outers\_Per\_Traded\_Unit | INT |  |
| Packs\_Per\_Traded\_Unit | INT |  |
| Sticks\_Per\_Traded\_Unit | INT |  |
| Grams\_Per\_Traded\_Unit | INT |  |
| Product\_Level | String | Product Level = Product\_SKU for SKU,  Product Level = Product\_Pack for Packing  Product Level = Product\_Brand for Brand  Product Level = Product\_House for House  Product Level = Product\_Group for Product Group |
| Par\_Row\_Id | String |  |

## Source: Account

The account file will contain the below columns and the data type column indicates the expected data type of the column.

|  |  |  |
| --- | --- | --- |
| Column | Data Type | Other Properties |
| RowId | String |  |
| AccountNo | String |  |
| AccountDesc | String |  |
| ParentRowId | String |  |
| DateOpened | TimeStamp |  |
| DateClosed | TimeStamp |  |

## Source: Invoice

The sales data will be made available in CSV files.

These files are expected to be made available monthly. Each file will contain data for a single month.

The month for which the file contains data for can be idenfied from the file name.

|  |  |  |
| --- | --- | --- |
| Column | Data Type | Other Properties |
| Invoice Number | String |  |
| Line Item | String |  |
| Date | TimeStamp |  |
| Account | String |  |
| Product | String |  |
| Quantity | INT |  |
| Gross Value | Float |  |
| VAT | Float |  |

# Lakehouse Architecture

The system will have 3 layers/directories in ADLS – Bronze, Gold and Silver

## Bronze

* The Bronze layer in ADLS will act as a staging layer and will contain two sub-layers – Raw and Stage.
* At the beginning of the ETL process, the files are ingested into the Raw layer. The Raw folder will contain a direct dump of source data and this folder will act as an entry point into the lakehouse. This layer will have the data from source as-is in the same file format.
* The data in Raw layer is converted to delta table format and stored in the Stage layer and picked up for further transformations downstream.
* Depending on whether the file is in the expected format (as mentioned in the source file section in this document) and if it can successfully be ingested into the staging delta table, the files will be moved to the Archive directory in Bronze and failed loads will move files to the Reject folder.
* In addition, there will be a Config directory in the Bronze layer to store settings and configuration details required for the ingestion pipeline to run.
* Raw layer: Same format as source file
* Stage layer: Delta format

## Silver

* All the transformations in the lakehouse will happen in the silver layer
* As part of the data tranformations, a unique key will be created for each record in the dataset
* Silver layer: Delta format

## Gold

* The Gold layer will contain the final data in parquet format ready for Power BI Reporting, after all the required cleansing and transformations are done
* The parquet files in this directory will be the source for the Power BI report
* Gold layer: Parquet format

## Logs

This directory will contain various logs of the ingestion and transformation process. Logs to be written into delta table. Logs must be stored in the bronze layer.

#### LoadLog

This file is used to store logging information.

The requirement is to log and track the data everytime the ADF pipeline is run and important steps in ADB layer. This file will be used to store details of each load.

Places to log:

* Start and end of ADF pipeline runs
* Start and end of notebook runs
* Capture important steps in the notebook runs

# Azure Data Factory

ADF (in combination with ADB) will be used to load fact and dimension data from the source to different directories in the data lake.

Files to be provided/uploaded in Source Folder

Files should move to Archive and Reject Folders based on the successful completion or failure of the run

*Note: All linked services required to establish connection to Azure SQL DB and Azure Data Lake Store will be created upfront (using secrets stored in Azure Key Vault).*

## pipeline\_LoadDimAccount

ADF pipeline to be created to orchestrate the process of loading Account data

## pipeline\_LoadDimProduct

ADF pipeline to be created to orchestrate the process of loading Product data

## pipeline\_LoadDimTime

ADF pipeline to be created to orchestrate the process of loading Time data

## pipeline\_LoadFactInvoice

ADF Pipeline to orchestrate the process of loading Invoice data

File name to be expected in following format: Invoice\_YYYYMM.csv

(YYYY- year, MM-Month Number).

## pipeline\_WrapperPackage

An ADF pipeline will call 3 child pipelines:

1. (Pipeline\_LoadFactInvoice) – To load Invoice data

2. (Pipeline\_LoadDimProduct) – To load Product data

3. (Pipeline\_LoadDimAccount) – To load Account data

Note: The linked services required for creating the ADF pipeline will be already setup.

* Log the start of the package execution in LoadLog file.
* Call the dimension ADF pipelines
* Call fact ADF pipeline
* Log end of package execution.

# Azure Data Bricks

Data bricks will be used to load and transform data starting from the source to gold layer for all product, account and fact files. Notebooks should be written in PySpark.

Use ADB for the following:

* To generate time data and populate the time delta table
* To create and ingest data from source files into delta tables in the Bronze (Stage) layer
* To apply transformations and business logic in the Silver layer and Gold layer
* To clean and transform fact data by validating it against the dimension data by accessing the dimension parquet files in ADLS

The settings file needs to be accessed in every Notebook to retrieve paths and any other configuration information.

## configSettings

The settings file (either a CSV/JSON) will be used to store configuration information to be used in ETL process. This table will be used to store the location of the source files, etc.

This table should be created in the Bronze layer. Add more columns, if relevant.

|  |  |
| --- | --- |
| **Column** | **Data Type** |
| SettingName | String |
| SettingValue | String |

## dimAccount

Create notebooks to transform account data. The Account data from the account flat file will be used to load the account dimension table into various layers in ADLS. The following steps will need to be performed:

### Bronze layer: Raw to Stage

* + Convert Account csv files (Raw layer) to Delta (Stage layer)
  + Validate if Account file has the correct format and data types. Archive / reject accordingly

### Bronze to Silver layer: Stage to Silver

* + Final silver layer to have additional columns:
    - AccountSkey - unique key, identifying each record (use *hash* function)
    - IsActive – to track the latest active records (used for SCD\*)
    - InsertedDateTime – Timestamp when record is inserted
    - EndDateTime – Timestamp when record is made inactive
  + Implement Slowly Changing Dimension logic  
    *A \*Slowly Changing Dimension (SCD) is a dimension that stores and manages both current and historical data over time in a data warehouse. It is considered and implemented as one of the most critical ETL tasks in tracking the history of dimension records.*

*SCD addresses the challenge of managing and preserving historical changes in dimensional data over time. Dimensions often undergo changes that need to be tracked and maintained for historical reporting and analysis purposes.*

*SCD provides a solution for handling these changes by creating a new row in the dimension table whenever a significant attribute value changes. This approach allows the retention of both the current and historical versions of a dimension record.*

* + - New accounts – If the account record is new and does not exist in our system, add a new record for the account in the account table with isActive = 1 (since this is the only active version of the account)
    - Existing account – If the source file contains a account that is already present in our account table but has changes in any attributes (like Description, DateOpened etc), insert new record for the account from the source data with isActive = 1 and mark the older record in the account table as isActive = 0
  + In case a parent account does not exist, then it should be mapped to “Unknown”
  + Using columns DateOpened and DateClosed, set AccountStatus as 0 for Closed, 1 for Open
* Handle exceptions, if any.

Silver layer schema:

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Other Properties** |
| AccountSkey | BIGINT | Unique Column |
| SourceSystemId | STRING | = RowID |
| AccountNo | STRING |  |
| AccountDesc | STRING |  |
| PrimaryGroupNo | STRING | Parent Row AccountNo |
| PrimaryGroupDesc | STRING | Parent AccountDesc |
| DateOpened | DATETIME |  |
| DateClosed | DATETIME |  |
| AccountStatus | BIT | 0 for Closed,1 for Open |
| IsActive | BIT | 1 for Active, 0 for Inactive |
| InsertedDateTime | DATETIME | Timestamp when record is inserted |
| EndDateTime | DATETIME | Timestamp when record is made inactive |

### Silver to Gold layer: Silver to Gold

* + Final gold layer to have additional columns:
    - Account = AccountNo + ‘-‘ + AccountDesc
    - AccountGroup = PrimaryGroupNo + ‘-‘ + PrimaryGroupDesc
  + Only Active accounts to come through in Gold layer

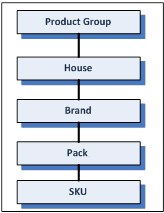
Gold layer schema:

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Other Properties** |
| AccountSkey | BIGINT |  |
| SourceSystemId | STRING |  |
| AccountNo | STRING |  |
| AccountDesc | STRING |  |
| PrimaryGroupNo | STRING |  |
| PrimaryGroupDesc | STRING |  |
| Account | STRING | AccountNo + ‘-‘ + AccountDesc |
| AccountGroup | STRING | PrimaryGroupNo + ‘-‘ + PrimaryGroupDesc |
| DateOpened | DATETIME |  |
| DateClosed | DATETIME |  |
| AccountStatus | BIT |  |

## dimProduct

Create notebooks to transform product data. The Product data from the product flat files will be used to load the product dimension table into various layers in ADLS.

**Product Hierarchy:**



The following steps will need to be performed to achieve required structure:

### Bronze layer: Raw to Stage

* + Convert Product csv files (Raw layer) to Delta (Stage layer)
  + Validate if Product file has the correct format and data types. Archive / reject accordingly

### Bronze to Silver layer: Stage to Silver

* + Products to follow the hierarchy mentioned using Product Level
  + New products will be inserted into the dimension table
  + Changes to product description and other attributes will be updated in the target Silver table (No SCD required. If there are changes, product dimension can be updated)
  + In case mapping does not exist, then it should be mapped to ‘Unknown’
  + Final silver layer to have additional columns:
    - ProductSkey - unique key, identifying each record (use *hash* function)
    - InsertedDateTime – Timestamp when record is inserted
    - ModifiedDateTime – Timestamp when record is modified
  + Handle exceptions, if any

Silver layer schema:

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Other Properties** |
| ProductSkey | BIGINT | Unique key |
| SourceSystemSKUId | STRING | = RowID |
| SkuCode | STRING |  |
| SkuDesc | STRING |  |
| PackingCode | STRING |  |
| PackingDesc | STRING |  |
| BrandCode | STRING |  |
| BrandDesc | STRING |  |
| HouseCode | STRING |  |
| HouseDesc | STRING |  |
| ProductGroupCode | STRING |  |
| ProductGroupDesc | STRING |  |
| TradedUnit | STRING |  |
| OutersPerTradedUnit | INT |  |
| PacksPerTradedUnit | INT |  |
| SticksPerTradedUnit | INT |  |
| GramsPerTradedUnit | FLOAT |  |
| InsertedDateTime | DATETIME | Inserted timestamp |
| ModifiedDateTime | DATETIME | Modified timestamp |

### Silver to Gold layer: Silver to Gold

* + Add new columns (logic mentioned in schema table)

Gold layer schema:

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Other Properties** |
| ProductSkey | BIGINT | Unique key |
| SourceSystemSKUId | STRING |  |
| SkuCode | STRING |  |
| SKU | STRING | SkuCode + ‘-‘ + SkuDesc |
| PackingCode | STRING |  |
| Packing | STRING | PackingCode + ‘-‘ + PackingDesc |
| BrandCode | STRING |  |
| Brand | STRING | BrandCode + ‘-‘ + BrandDesc |
| HouseCode | STRING |  |
| House | STRING | HouseCode + ‘-‘ + HouseDesc |
| ProductGroupCode | STRING |  |
| ProductGroup | STRING | ProductGroupCode + ‘-‘ + ProductGroupDesc |
| TradedUnit | STRING |  |
| OutersPerTradedUnit | INT |  |
| PacksPerTradedUnit | INT |  |
| SticksPerTradedUnit | INT |  |
| GramsPerTradedUnit | FLOAT |  |

## dimTime

* Create logic to populate the dimTime delta table in Silver layer and create a Gold layer object as well.
* This will populate the time table with 10 years of time data. It accepts a date as widget or can be read from settings
* Time table should have continuous dates at any given point. If the input date (start date) parameter is provided in such a way that it creates a gap in dates in the time table, the gap should automatically be filled. There should also be no duplicate entries.

E.g. If date table has dates from 2021 to 2030 and if input argument for next run is 1st Jan 2000, then just populating 10 years time series would mean time table will have dates from 2000 to 2009 and 2021 to 2030. Instead, it should ensure all dates from 2000 to 2030 are present in the time table.

*Time notebook to be written in Spark SQL*

Silver Schema

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Other Properties** |
| TimeSkey | INT | Unique to identify record |
| Date | DATETIME |  |
| DayOfWeek | STRING |  |
| CurrentDay | BIT | Flag indicating latest day for which fact data exists |
| WorkingDay | BIT |  |
| MonthId | INT |  |
| MonthDesc | STRING |  |
| QuarterId | SMALLINT |  |
| QuarterDesc | STRING |  |
| Year | INT |  |

Gold Schema

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Other Properties** |
| TimeSkey | INT | Unique to identify record |
| Date | DATETIME |  |
| DayOfWeek | STRING |  |
| CurrentDay | BIT | Flag indicating latest day for which fact data exists |
| WorkingDay | BIT |  |
| MonthId | INT |  |
| MonthDesc | STRING |  |
| QuarterId | SMALLINT |  |
| QuarterDesc | STRING |  |
| Year | INT |  |

## factInvoice

The factInvoice table is used to store the invoice data.

### Bronze layer: Raw to Stage

* + Convert Fact files (Raw layer) to Delta (Stage layer)
  + Validate if Invoice files have the correct format and data types. Archive / reject accordingly
  + Validate the stage data for any blanks in the dimension columns. If any dimension column is found to have a blank, then stop the load and move the file to reject, else proceed with the data load

### Bronze to Silver layer: Stage to Silver

* + Expected format of silver layer invoice table is delta
  + Silver delta table must be partitioned on Year and Month and only the relevant partitions have to be refreshed based on availability of data in raw layer
  + New invoice data comes in, it needs to be inserted into silver table
  + If invoice data for the month already exists, replace the entire month’s data with the data from incoming file
  + The fact data to be joined with the dimension data to get appropriate dimension surrogate keys
  + In case a mapping does not exist, the corresponding fact record to be **moved** to the exception table
  + Additional columns to be added in silver layer:
    - AccountSkey – get the Key column value corresponding to account
    - ProductSkey – get Key column value corresponding to product
    - TimeSkey – get Key column corresponding to time
    - InsertedDateTime - Timestamp when record is inserted

Silver layer schema

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Other Properties** |
| FactInvoiceSkey | BIGINT | Unique |
| AccountSkey | BIGINT |  |
| ProductSkey | BIGINT |  |
| TimeSkey | INT |  |
| InvoiceNumber | STRING |  |
| InvoiceLineItemNumber | STRING |  |
| Quantity | FLOAT |  |
| GrossValue | FLOAT |  |
| VATvalue | FLOAT |  |
| NetValue | FLOAT | = GrossValue - VAT |
| InsertedDateTime | DATETIME |  |

### Silver to Gold layer: Silver to Gold

* + Restrict data in gold layer to latest 3 years

Gold layer schema

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Other Properties** |
| FactInvoiceSkey | BIGINT | Unique |
| AccountSkey | BIGINT |  |
| ProductSkey | BIGINT |  |
| TimeSkey | INT |  |
| InvoiceNumber | STRING |  |
| InvoiceLineItemNumber | STRING |  |
| Quantity | FLOAT |  |
| GrossValue | FLOAT |  |
| NetValue | FLOAT |  |
| VATvalue | FLOAT |  |

## Logging Mechanism

The following notebook should be created for logging purposes

### NBLoadLog

Use this notebook to define the logging function. Create UDF that can be called in different notebooks at different places in the load capturing the information required.

This UDF accepts relevant parameters as per logging requirement and will log the data in CSV files in bronze layer. Logs to be stored in csv format.

The requirement is to log and track the data everytime the ADF pipe is run or an important notebook step is run and this file will be used to store details of each load.

Places to log:

* Start and end of ADF pipeline runs
* Start and end of Notebook runs
* Capture important steps in the Notebook runs

|  |  |  |
| --- | --- | --- |
| **Column** | **Data Type** | **Comments** |
| LogDateTime | DATETIME |  |
| LogType | STRING | Component Name |
| LogMessage | STRING | Step Name |
| LogMessageDetails | STRING | Details about the Step |
| LogRowCount | INT |  |
| LogNotebookPath | INT |  |
| LogClusterName | STRING |  |
| LogClusterID | STRING |  |
| LogRunID | INT |  |
| LogUserID | STRING |  |

# Azure DevOps

## Overview

Azure DevOps should be used to deploy the Cutlass solution (ADF & ADB) to production environment. Azure DevOps organisation, production environment and the necessary Service Principles for the pipelines to work properly will be made ready as part of the training. The training DevOps organisation & project (provided to the trainees on day-2 of the training) should be used to implement the case study.

## Git Repository

A git repository should be used to version control all the source code/ solution from ADB and ADF. The solution should be store in the repository as per the following folder structure. From the root of the repository,

* *<<repo root>>/ADF/* - folder that contains the ADF solution
* *<<repo root>>/ADB/* - folder that contains the ADB solution

The repository should atleast have the two branches below:

### Develop Branch

The develop branch is used to store the solution that is being developed. This branch should be used in ADF as the develop branch. ADB should commit/push the notebooks into this branch.

### Main Branch

This branch is used to store the stable/final versions of the solution that is ready-to-be deployed or already deployed to production environment. No commits or pushes should be made directly to this branch. Any change to the solution should be made as a new commit/push to the develop branch and a pull request should be created to bring the changed from develop branch to the main branch. The Devops Pipelines below should use this branch for deployment.

## DevOps Pipelines

This section describes in detail the pipelines that are required to be created to implement a DevOps setup for Cutlass.

* All pipelines must be YAML based pipelines.
* All pipelines must have proper names as per below.
* All pipeline yaml files must have proper names as per below and should be stored in *<<repo root>>/.azure-pipelines* folder

The following pipelines must be implemented:

### PIPELINE\_ADF (.azure-pipelines/pipeline\_adf.yml)

* A YAML pipeline to deploy the ADF solution to the production ADF.
* This pipeline should be run on the main branch.
* This pipeline should be triggered automatically once there is any change to the <<repo root>>/ADF/ folder in the main branch

### PIPELINE\_ADB (.azure-pipelines/pipeline\_adb.yml)

* A YAML pipeline to deploy the ADB notebooks to the production ADB workspace.
* This pipeline should be run on the main branch.
* This pipeline should be triggered automatically once there is any change to the <<repo root>>/ADB/ folder in the main branch

# Modelling in PBI

This section describes in detail the various tables that are required in the model.

## Account

The account dimension table will include the following columns and will also contain a hierarchy, the details of which are given below.

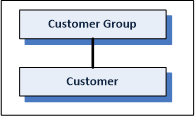
It will be based on gold layer dimAccount.

Columns

|  |
| --- |
| Attribute Name |
| AccountSkey |
| SourceSystemId |
| Account |
| AccountStatus |
| Account Group |

Hierarchy

The Account Group hierarchy will consist of two levels as shown below.



## Product

The product dimension table will include the following product columns and will include a “Product” hierarchy.

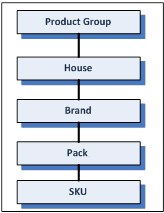
The product dimension table will include the following product columns and will include a “Product” hierarchy

It will be based on gold layer dimProduct

***Columns***

|  |
| --- |
| Column |
| ProductSkey |
| SourceSystemSKUId |
| SkuCode |
| SKU |
| PackingCode |
| Packing |
| BrandCode |
| Brand |
| HouseCode |
| House |
| ProductGroupCode |
| ProductGroup |
| TradedUnit |
| OutersPerTradedUnit |
| PacksPerTradedUnit |
| SticksPerTradedUnit |
| GramsPerTradedUnit |

***Hierarchy***



## Time

The time dimension table will have the following columns and will contain a “Time” hierarchy. It will be based on gold layer dimTime.

***Columns***

|  |
| --- |
| Column |
| TimeSkey |
| Date |
| DayOfWeek |
| CurrentDay |
| WorkingDay |
| Month |
| Quarter |
| Year |

***Hierarchy***



## Invoice

The invoice table will be based on the gold layer factInvoice

|  |
| --- |
| **Column** |
| AccountSkey |
| ProductSkey |
| TimeSkey |
| InvoiceNumber |
| InvoiceLineItemNumber |
| Quantity |
| GrossValue |
| NetValue |
| VATValue |

### Measures

This section explains each of the measures to be created in the invoice table.

|  |  |
| --- | --- |
| Calculated Measure | Definition |
| Invoice Volume (Quantity) | Sum of Quantity |
| Gross Invoice Value | Sum of GrossValue |
| Net Invoice Value | Sum of NetValue |
| VAT Value | Sum of VATValue |
| Latest Month Invoice Volume | Total Invoice Volume for the Latest Month available |
| Last Year Total Invoice Volume | Total Invoice Volume for the Previous year based on current selection. |
| Last Year YTD Invoice Volume | Total Invoice Volume from the start of the previous year up till the current month (in the previous year) based on current selection. |
| % of Last Year Total Invoice Volume | Last Year YTD Invoice Volume /Last Year Total Invoice Volume |
| Last Year YTD Net Invoice Value | Net Invoice Value for the previous year up till the current month (in the previous year) based on the current selection |
| YTD Invoice Volume | Total Invoice Volume for the current year up till the current month. |
| YTD Net Invoice Value | Total Net Invoice Value for the current year up till the current month. |
| MAT Net Invoice Value | Moving Annual Total of last 52 weeks based on the selected date |
| Latest Available Date | Display the date for latest date available for fact data |

# Reporting in PBI

**Build a report to answer the following questions:**

1. What is my Total Net Invoice Value for the current selection?
2. What is my Latest Month Net Invoice Value ? Also show which month this is referring to?
3. How has my Gross Value changed over the months of the selected year?
4. A table with YTD Invoice volume of selected year and % change in YTD invoice volume at Product – House level where the values are highlighted in different colours based on following rule:
   * <= -50% - Red
   * -49% to 49% - Amber
   * > =50% - Green
5. How has my Invoice Volume been distributed across different levels of the product hierarchy?
6. How has my Net Invoice Value been distributed across different levels of the account hierarchy?

*Add slicers for Year/Quarter, active Account Group/Account hierarchy and House/Brand. Default the report to select one year.*

Drill through from the chart built for question 5 to the Product report.

Drill through from the chart built for question 6 to the Account report

**Build a Product specific report:**

1. Total Invoice Volume, Last Year Total Invoice Volume for the selected product – should not change with filter selections
2. Trend of Net Invoice Value/Gross Invoice Value for this product over time (allowing drilldown to lowest level of time hierarchy)
3. Top 5 active accounts purchasing this product based on Net Invoice Value/Gross Invoice Value.

*User should be able to switch between Net Invoice and Gross Invoice from a parameter.*

*Only relevant slicers to be present.*

**Build a Account specific report:**

1. Total Invoice Volume, Total Gross Invoice Value, Total Net Invoice Value, Total VAT Value for the selected account
2. Split of the Product Group/Product House based on number of different type of products bought by the account. (Drilldown between Product Group -> Product House)
3. Top 5 SKUs sold by this account based on Net Invoice Value and their corresponding YTD net invoice sales value and Last year YTD net invoice sales value.

*Only relevant slicers to be present.*

**In general:**

1. Be conscious of fonts, colours, and chart placements
2. Format slicers appropriately
3. Publish to ‘My Workspace’ and test that your report and interactions between your visualizations work there too.

# Azure Data Lake Storage

The ADLS folder hierarchy **should strictly follow** the below structure.

**Bronze**

* Raw
  + Dimension
    - Product
    - Account
  + Fact
    - Invoice
* Stage
  + Dimension
    - Product
    - Account
  + Fact
    - Invoice
* Config
* Archive
  + Dimension
    - Product
    - Account
  + Fact
    - Invoice
* Reject
  + Dimension
    - Product
    - Account
  + Fact
    - Invoice
* Logs (folder structure defined in the logging UDF)

**Silver**

* Dimension
  + Product (Delta Table)
  + Account (Delta Table)
* Fact
  + Invoice **(Delta table to be partitioned based on Year and Month)**
    - YYYY
      * MM

**Gold**

* Dimension
  + Product (Parquet File)
  + Account (Parquet File)
* Fact
  + Invoice (Parquet File)

# Best Practices

## Azure Data Factory

The following standards should be followed while developing the ADF objects

* All datasets should be named as “dataset\_<dim/fact>\_<source>”
* All pipelines should be named as “pipeline\_<load/transform>\_<name>”
* The description property should be updated for all objects (datasets, pipeline, activities, etc.)
* All activities should be named in a logical manner “<name of activity>\_<details”. For example, “copy\_invoice\_adls\_to\_stage”.