**Introduction**

This document describes the approach that was followed to build a DataWarehouse model and populate it with data.

The challenge was to create a Datawarehouse model using Kimball’s methodology.

Kimball’s methodoly is known as the bottom-up approach. It emphasizes building data marts first using a dimensional model, usually with star schemas or snowflake schemas, which are optimized for easy querying and reporting.

**Design Phase**

Having this in mind the approach was first to analyze the given file (Invoices\_Year\_2009-2010.csv) and see how we can design a model suited for the data given.

The columns contained in the file are shown below:

* Invoice
* StockCode
* Description
* Quantity
* Price
* Customer ID
* Country
* InvoiceDate

The Quantity and Price columns are containing metrics that are related with Invoice column and the InvoiceDate column which is the actual date that a purchase took place.

Take this as our main assumption we can say that a FactTable can be created with the columns: Invoice, InvoiceDate, Quantity and Price.

Furthermore, StockCode and Description columns can construct the DimProducts and Customer ID column can construct the DimCustomers.

In addition, two dimensions will be created DimTime and DimCountries.

* DimTime will contain date and time information for the last 30 years.
* DimCountries will contain all the countries of the world.

These two dimensions will help to validate the source data columns from Invoice source table, InvoiceDate and Country for consistency.

Based on the above the approach that followed is outlined in the below star schema.

A screenshot of a computer

Description automatically generated

We can see that all tables are having as primary key a surrogate\_key (sk\_id) which is a unique auto-generated key which is created when the data are inserted in the table and serves the uniqueness of the record.

Furthermore, the loading strategy is first to load the Dimension tables to generate the surrogate keys for all records and then based on these keys to load the Fact table and fill the corresponding columns referring to Dimensions with these keys. Also, the use of hash\_key logic took place for capturing the changes of a row.

The loading approach that was used for Dimensions was the SCD2 (Slowly Changing Dimension – type 2). This approach preserves historical information meaning that if a change occurs the old record will close (update end\_date with current date and set flag is\_active = 0) and a new one is inserted taking as end\_dt the value ‘9999-12-31’ and flag is\_active = 1).

Below we can see the BetssonDB and the tables contained.

A screenshot of a computer

Description automatically generated

**Implementation Phase**

To achieve the Star Schema described above we had to define some areas responsible for extracting, transforming and loading (ETL) the data. Three areas were created to serve this purpose: source area, staging area and target area.

**Python Solution responsible for Source Area**

First, we had to start investigating the columns to see the data quality and then load the data to source area of the database.

For the above step a Python solution was used. The project was built in functional mode since it wasn’t so big and complex (if this was the case an OOP approach would be more efficient). However, the database related staff was built as a Class (db\_connect.py) since more than one db instances can be created. A utils.py file was created containing all the functions needed for manipulating the data.

The first thing to be done was to acquire the source file and load it. The pandas library was chosen based on its overall manipulation capabilities. When loading the file some parameters were passed like:

* Encoding='ISO-8859-1'

This was applied because an error occurred when reading the file saying that “*can't decode byte 0xa3 in position 52440: invalid start byte*” so encoding ISO-8859-1 applied to bypass such errors.

* dtype=source\_schema

Define an initial schema helps ensure consistency, avoids potential errors, and can optimize performance.

* parse\_dates=['InvoiceDate']

Convert InvoiceDate column to Python datetime objects.

This simplifies handling dates and enables time-based operations in the dataframe.

The second thing was an inspection of the datatypes containing in the columns of the source file, if there were different datatypes in a column and if there were duplicated (in the whole file or in a subset of it).

A test\_utils.py file was created, containing functions that are responsible for performing the above checks. Through tests.py file all these functions are called along with some custom checks. Basic aspects of data quality are covered (in the future more checks can be added if necessary).

The tests performed arose some issues regarding the variety of datatypes that some columns contain. For example, the column Description had both ‘str’ and ‘float’ datatypes. Since Description column contains descriptions, we took the assumption that the column should be a ‘str’ datatype. So, for this column we found the ‘float’ datatypes and convert them to ‘str’ datatypes. Also, for the column Invoice from the investigation we saw that in general the column was consisted by 6digit integer but there were cases that a character was in front of the 6digt number (example. C489518). In here we took the assumption that the letter was entered by mistake, and we eliminate it. These transformations along with others have been performed in the main.py file. All these findings should be sent to the business departments to confirm/suggest approaches.

The third task was the creation of the dataframes for loading the data into source schema in the database. The invoice source file after the tests and the basic transformation that took place was separated into smaller dataframes each one designed for the dimensions and fact tables.

Based on the start schema designed the following dataframes were created:

* product (DimProducts)
* customer (DimCustomers)
* invoice (FactInvoice)

For DimCountry and DimTime as mentioned above another approach took place.

Furthermore, three tables were created in the source schema of the database:

* src.Products
* src.Customers
* src.Invoice

These three tables if they don’t exist are created based on the create statements in params.py file, then are being truncated and then the data based on the dataframes are inserted.

The create statements, the truncations and the insertions are managed by the db\_connect.py class.

**MSSQL responsible for Staging and Target Area**

After loading the data in the source area of the database the Python program ends its job, and the rest transformations are being handled from MSSQL. This decision took place because as we talk about relational model and tables SQL can be more performant that classic Python (if we had the option of using Spark the processing would be continued with PySpark).

In the folder sql\_scripts there are the scripts used for creating the database schemas, tables along with the ETL operations that take place. The scripts are ordered based on the sequence that should run. Ideally the SQL code and the Python code would run from an ETL tool (data factory, snowflake) or from an orchestration tool such as Airflow.

* The script starting with 00 is responsible for creating the schemas stg and trg in database.
* The scripts starting with 01 are responsible for creating the tables in stg and trg schema.
* The scripts starting with 02 are responsible for inserting the values in DimTime table, in DimCountries table and in the ErrorCodes table.
* The scripts starting with 03 are responsible for inserting the erroneous values from the src tables for further investigation from business departments.
* The scripts starting with 04 are responsible for inserting the cleaned data from src schema to stg schema. In this step the hash\_key functionality introduced.
* The scripts starting with 05 are responsible for merging the dimension data from the corresponding stg tables to the target Dimensions.
* The script starting with 06 is responsible for merging the fact data from the corresponding stg table to the target Fact table.
* The script starting with 07 is showing the aggregations with the data of the target Fact table.

**Observations & Findings**

From the investigation we saw that some columns had invalid data or data that needs to be sent to the business departments for further investigation.

Below are the findings that have been found and needs further investigation.

* Someentries in the column Country contains countries that are invalid or empty string. For example, names like Bermuda and Hong Kong need to be corrected from the biusiness. All the invalid dates took in the Fact Table the sk\_id = 999999 (INVALID). The invalid records were inserted in table dbo.CountriesInvalid.

Also, some entries like ‘U.K’ where translated to ‘United Kingdom’. The translations can be found in the script 04.Populate\_StgCountries.sql.

* In CustomerID column the value ‘TEST’ found, and empty strings located. Also, these errors handled with sk\_id = 999999 (INVALID). These cases need further clarification from business. The records were inserted in table dbo.CustomersInvalid.
* Some StockCodes were duplicated. The duplicated records were found and inserted in table dbo.ProductsDuplicates for further investigation. Also, these errors handled with sk\_id = 999999 (INVALID).
* Another finding is that the related description column contained empty strings. The records were inserted in table dbo.ProductsNoDesc for further investigation.
* Some values in the quality column are negative. These values were inserted in table dbo.InvoiceQualityNegative along with the columns stockcode and the price for the data to be more meaningful when the investigation occurs.
* Some values in the price column are negative. These values were inserted in table dbo.InvoicePriceNegative along with the columns stockcode and the price for the data to be more meaningful when the investigation occurs.

**Aggregation for Reporting Purposes**

The aggregations that have been done are included in the file 07.Aggregations.sql.

Below are the aggregations that have been done:

* Top Selling Products (By Quantity)
* Revenue Contribution by Country
* Revenue by Product
* Weighted Average Price
* Total Revenue by Day
* Top 10 Clients Based on Purchases

These are some calculations that can be done for reporting purposes, different departments in a company might need other calculation fitting their needs.