



Project Requirements Document

Fernando Fontoura de Araujo

fontouraaraujo@gmail.com

+1 (778) 997 8877

Vancouver, Brith Columbia - Canada

08/28/2023



Summary

Introduction	5
Project Keywords	5
Specification	6
Purpose	6
Background.....	6
Assignment	6
From Shopify File.....	7
From Google Analytics File - Landing Page.....	7
Deliverables of the project.....	7
Phase 01: Construction of the Stage area.	8
JSON Shopify	8
CSV Customers	9
CSV Orders	10
Phase 02: Construction of the Data Warehouse	11
Data Warehouse Modeling – Star Schema	12
Constraints	12
Time Dimension	12
Customers Dimension	16
Orders Dimension (It's actually a Fact)	16
Visual Studio Packages	18
SQL Server Agent e Integration Services Catalogs.....	19
Google Analytics CSV	20
Phase 03: Analysis Services	23
Phase 04: Reporting Services	24



Phase 05: Backup Structure	27
Backup Scripts	27
Backup_SELL_DW.bat and Backup_SELL_STAGE.bat.....	27
BACKUP_SELL_DW.sql and BACKUP_SELL_STAGE.sql	27
Backup folder structure	28
Phase 06: Power BI	29
Dashboard Orders – Shopify	29
Orders	29
Unique users	29
New users	30
Repeating users.....	30
Revenue Total.....	30
Revenue only new user.....	31
Average time between the first and second purchase.....	31
Results comparing month vs previous month.	31
Results comparing the month vs the same month of the previous year.	32
Seasonality.....	32
Orders by Fulfillment Status	33
Moving Average 12 months	33
Dashboard Orders – Google Analytics.....	34
Sessions	34
Bounce Rate.....	34
AVG Time on Page	35
% Conversion Rate	35
% Click to rate	35



KPIs by Category and Value 36

Top 10 Values – By Number of Sessions..... 36

ABC Analysis Values - By Number of Sessions..... 37

Views for Power BI..... 38

Thanks 43



Introduction

This project documentation was developed as a free project, as a data analyst from the fictional company Sell Boardgames would do in his daily work.

In macro needs, the project was divided into 6 major phases.

Phase 01: Construction of the Stage area

Phase 02: Construction of the Datawarehouse

Phase 03: Analysis Services

Phase 04: Reporting Services

Phase 05: Backup Structure

Phase 06: Power BI

For the development of the business intelligence project, Python and tools from the Microsoft data environment were used, namely: SQL Server, Management Studio, Integration Services, Analysis Services, Reporting Services, and Power BI.

All files, packages, and scripts developed and/or used in this project will be sent along with this report.

Project Keywords

SQL Server - Power BI - DAX - Management Studio - SQL - Slowly Changing Dimension (SCD) - Data Warehouse - Importing data from JSON and CSV – ETL - Integration Services (SSIS) - Time Dimension – Python – Pandas – Numpy – PyODBC - Bulk Insert - Analysis Services (SSAS) - Star Schema - Stored Procedure – Cursors - Nonclustered Index - Stage Area - Error Handling - SQL Server Agent - Integration Services Catalogs - Reporting Services (SSRS) - Backup Structure - Batch Scripts - ABC Analysis - Common Table Expression (CTE) – Views - Data Analysis - Project Requirements Document



Specification

Purpose

The objective of this free project is to showcase job skills in relation to key requirements and responsibilities for the data analyst role.

As a data analyst at SELL, I became a specialist in the Business Intelligence platform. This project assignment is designed to test the core capabilities of the role required to take this ownership, including the ability to extract, transform, analyze data, and create reports.

Background

SELL is in a leading position as the largest online retailer of boardgames in the world and has used our knowledge of the space to expand into creating our own brand of boardgames. We have gotten to this position following our instinct, and intuitive knowledge of the market, putting customers first, and a simple yet effective strategy of focusing on content for users, SEO, and email marketing.

As a company, we want to be a high-performing, purpose-driven team that is informed by data and to use metrics to get better results helping us achieve our objectives. We know that we can do better for users by validating our insights with data and improving our understanding and performance through analytics and new ways of working.

Assignment

I used documents (json or csv file) with raw data for analysis, you can request these files if you want.

With the data provided, the initial goal is to calculate the following KPIs and create graphs to visualize the results.



From Shopify File

#Orders, #Unique users, #New users, #Repeating users, \$ Revenue Total, \$ Revenue only new user, Average time between the first and second purchase and any other KPIs that may be considered relevant.

I want to see the results comparing month vs previous month.

I want to see the results comparing the month vs the same month of the previous year.

From Google Analytics File - Landing Page

Sessions, %bounce Rate, AVG Time on page, % Conversion Rate, Click to rate and any other KPIs that may be considered relevant.

I want to see the results grouped by category and value.

From the raw data I need extract the category and the value, the position red is the category, and the blue is the value.

Deliverables of the project

I must provide the analysis created, it can be Excel or the BI tool of my choice.



Phase 01: Construction of the Stage area.

In the first stage of the project, the focus was on the treatment of received data files.

The first step was to create a model for the future Data Warehouse with an understanding of the available data. After this process, the staging database SELL_STAGE was created.

JSON Shopify

The first imported file was the JSON file through the procedure below.

```
CREATE PROCEDURE LOAD_ST_JSON AS
DECLARE @json AS NVARCHAR(MAX)

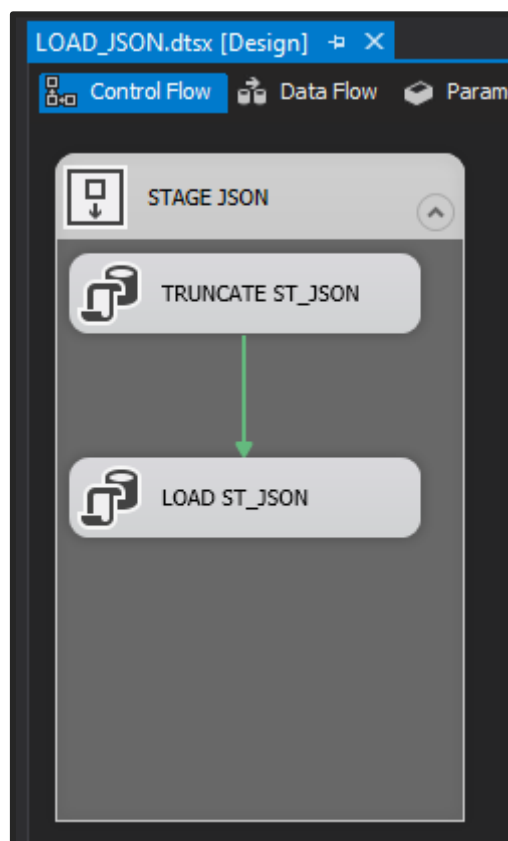
SELECT @json = BulkColumn
FROM OPENROWSET (BULK 'C:\SELL\Data\shopify_orders.json', SINGLE_CLOB)
as Datafile

INSERT INTO ST_JSON
SELECT
    convert(bigint,replace(tb.id,'gid://shopify/Order/','')) ID,
    tb.PROCESSED_AT,
    tb.CURRENCY_CODE,
    tb.CURRENT_SUBTOTAL_PRICE_SET_AMOUNT,
    tb.ORIGINAL_TOTAL_DUTIES_SET,
    tb.ORIGINAL_TOTAL_PRICE_SET_AMOUNT,
    tb.DISPLAY_FULFILLMENT_STATUS,
    convert(bigint,replace(tb.customer_id,'gid://shopify/Customer/',''))
CUSTOMER_ID,
    tb.CUSTOMER_CREATED_AT
FROM
    OPENJSON(@json)
WITH (
    id VARCHAR(200) '$.id',
    processed_at datetime '$.processedAt',
    currency_code VARCHAR(3) '$.currencyCode',
    current_subtotal_price_set_amount numeric(10,2)
    '$.currentSubtotalPriceSet.shopMoney.amount',
    original_total_duties_set VARCHAR(200) '$.originalTotalDutiesSet',
```




```
original_total_price_set_amount numeric(10,2)
'$.originalTotalPriceSet.shopMoney.amount',
display_fulfillment_status VARCHAR(20) '$.displayFulfillmentStatus',
customer_id VARCHAR(200) '$.customer.id',
customer_created_at datetime '$.customer.createdAt'
) AS tb
```

After importing the .csv files, I preferred to use this new path, leaving the JSON table only in the STAGE area, in addition to the SSIS process used for its import. SSIS Package: LOAD_JSON.dtsx.



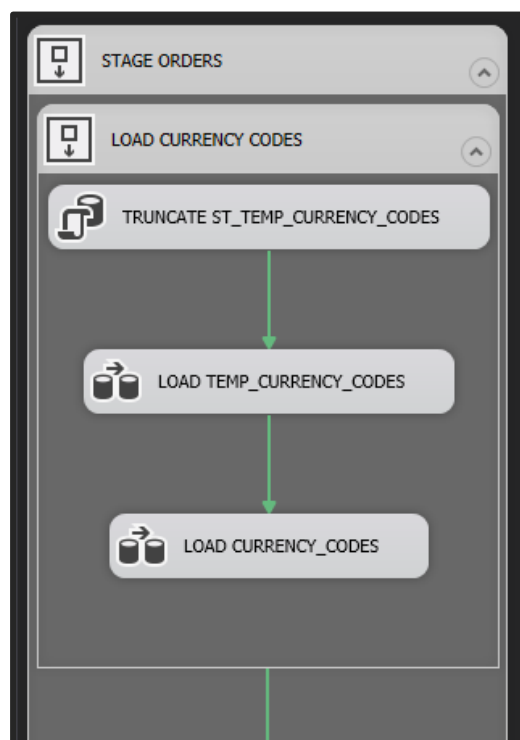
CSV Customers

Through an ETL process, the customers.csv file was imported into the Stage area and later into the DW in the same package, LOAD_CUSTOMERS.dtsx. An important point in loading the Stage area is to observe that the table is truncated in the first step of the package, this rule is applied by default.



CSV Orders

For the purposes of better work development, I chose to use the orders table as a future fact. Thinking about it I decided to normalize the Currency_Code and Fulfillment_Status fields into two separate tables. When performing this import, I chose to use temporary tables to receive the data before going to the staging table, so it was possible to apply a distinct to each field and insert only new records for each table into the stage.

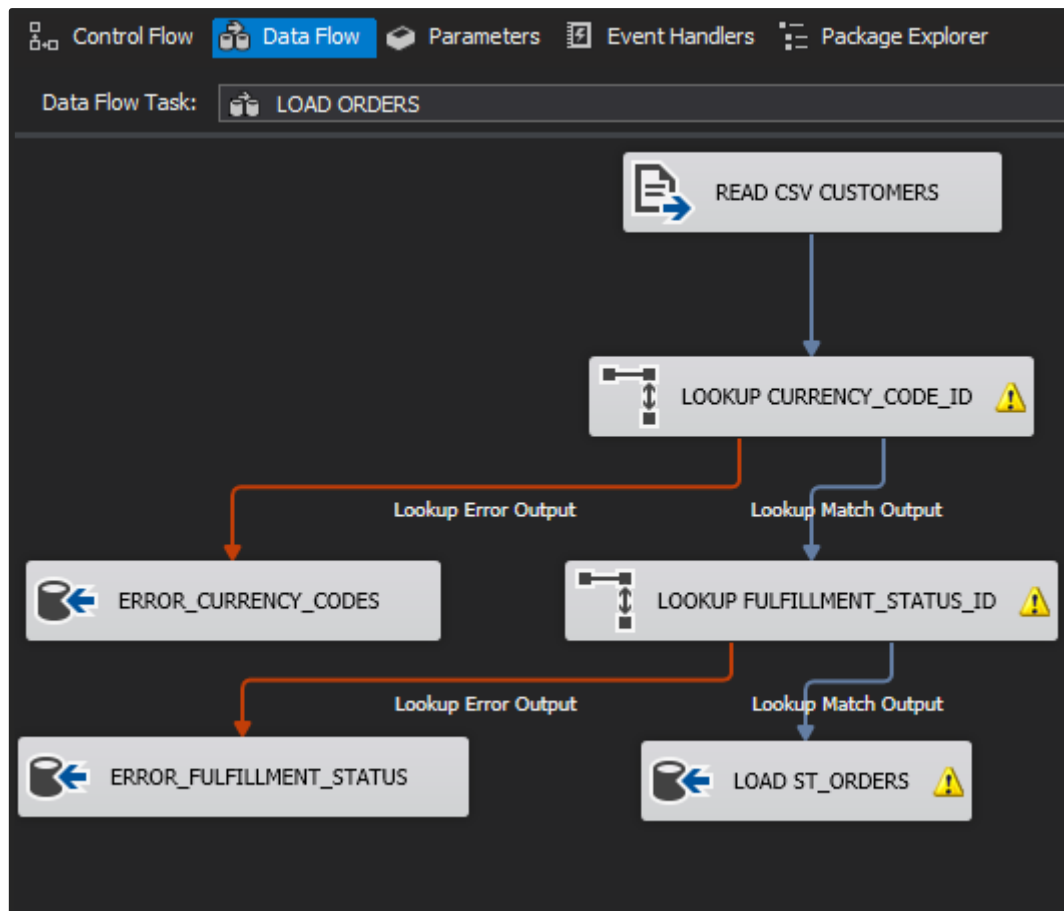


Script utilizado:

```
SELECT
    TSC.CURRENCY_CODE_NAME
FROM
    ST_TEMP_CURRENCY_CODES TSC
    LEFT JOIN ST_CURRENCY_CODES SC
        ON TSC.CURRENCY_CODE_NAME = SC.CURRENCY_CODE_NAME
WHERE
    SC.CURRENCY_CODE_NAME IS NULL
```



To load the orders table, lookup steps were used to search for the Ids generated in the normalized tables mentioned above. These Steps had error handling, redirecting log records to the SELL_ERROR database.



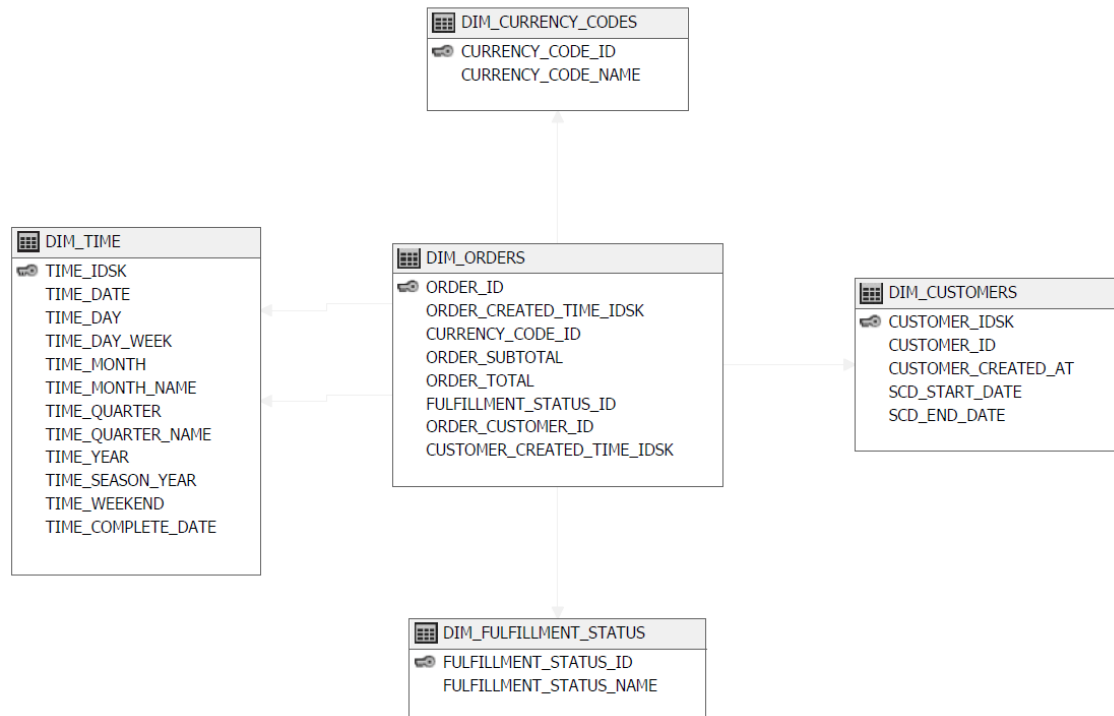
Phase 02: Construction of the Data Warehouse

In the first stage of the project, the focus was on the load coming from the Stage area to the DW still using SSIS.

At this stage, it was possible to see all the initial modeling of the Data Warehouse take shape with the creation of the SELL_DW database.



Data Warehouse Modeling – Star Schema



Constraints

Reference	Host Table	Host Field	Type
DIM_TIME	DIM_ORDERS	ORDER_CREATED_TIME_IDSK	FOREIGN KEY
DIM_CURRENCY_CODES	DIM_ORDERS	CURRENCY_CODE_ID	FOREIGN KEY
DIM_FULFILLMENT_STATUS	DIM_ORDERS	FULFILLMENT_STATUS_ID	FOREIGN KEY
DIM_CUSTOMERS	DIM_ORDERS	ORDER_CUSTOMER_ID	FOREIGN KEY
DIM_TIME	DIM_ORDERS	CUSTOMER_CREATED_TIME_IDSK	FOREIGN KEY

Time Dimension

The first addition to the DW for the purposes of enriching the analyzes was the creation of the time dimension through the script below, containing the insert and the cursor for calculating some columns.

```
-- STEP 1 – FIRST INSERT
DECLARE  @STARTDATE DATETIME
          , @ENDDATE DATETIME
          , @DATE DATETIME

PRINT GETDATE()

SELECT @STARTDATE = '1/1/1950'
```



```
, @ENDDATE = '1/1/2050'

SELECT @DATE = @STARTDATE

WHILE @DATE < @ENDDATE
BEGIN

    INSERT INTO DIM_TIME
    (
        time_date,
        time_day,
        time_day_week,
        time_month,
        time_month_name,
        time_quarter,
        time_quarter_name,
        time_year
    )
    SELECT @DATE AS time_date, DATEPART(DAY,@DATE) AS time_day,

        CASE DATEPART(DW, @DATE)

            WHEN 1 THEN 'Sunday'
            WHEN 2 THEN 'Monday'
            WHEN 3 THEN 'Tuesday'
            WHEN 4 THEN 'Wednesday'
            WHEN 5 THEN 'Thursday'
            WHEN 6 THEN 'Friday'
            WHEN 7 THEN 'Saturday'

        END AS time_day_week,

        DATEPART(MONTH,@DATE) AS time_month,

        CASE DATENAME(MONTH,@DATE)

            WHEN 'January' THEN 'January'
            WHEN 'February' THEN 'February'
            WHEN 'March' THEN 'March'
            WHEN 'April' THEN 'April'
            WHEN 'May' THEN 'May'
            WHEN 'June' THEN 'June'
            WHEN 'July' THEN 'July'
```



```
        WHEN 'August' THEN 'August'
        WHEN 'September' THEN 'September'
        WHEN 'October' THEN 'October'
        WHEN 'November' THEN 'November'
        WHEN 'December' THEN 'December'

    END AS time_month_name,

        DATEPART(qq,@DATE) time_quarter,

        CASE DATEPART(qq,@DATE)
        WHEN 1 THEN 'First'
        WHEN 2 THEN 'Second'
        WHEN 3 THEN 'Third'
        WHEN 4 THEN 'Fourth'
    END AS time_quarter_name
    , DATEPART(YEAR,@DATE) time_year

    SELECT @DATE = DATEADD(dd,1,@DATE)
END

UPDATE DIM_TIME
SET time_day = '0' + time_day
WHERE LEN(time_day) = 1

UPDATE DIM_TIME
SET time_month = '0' + time_month
WHERE LEN(time_month) = 1

UPDATE DIM_TIME
SET time_complete_date = time_year + time_month + time_day
GO

--STEP 2 - CURSOR
DECLARE C_TIME CURSOR FOR
    SELECT time_idsk, time_complete_date, time_day_week, time_year FROM
    DIM_TIME
DECLARE

        @ID INT,
        @DATE varchar(10),
        @DAYWEEK VARCHAR(20),
        @YEAR CHAR(4),
        @WEEKEND CHAR(3),
        @SEASON VARCHAR(15)
```



```
OPEN C_TIME
  FETCH NEXT FROM C_TIME
  INTO @ID, @DATE, @DAYWEEK, @YEAR
WHILE @@FETCH_STATUS = 0
BEGIN

    IF @DAYWEEK in ('Sunday','Saturday')
    SET @WEEKEND = 'Yes'
    ELSE
    SET @WEEKEND = 'No'

    IF @DATE BETWEEN CONVERT(CHAR(4),@YEAR)+'0923'
    AND CONVERT(CHAR(4),@YEAR)+'1220'
    SET @SEASON = 'Autumn'

    ELSE IF @DATE BETWEEN
CONVERT(CHAR(4),@YEAR)+'0321'
    AND CONVERT(CHAR(4),@YEAR)+'0620'
    SET @SEASON = 'Spring'

    ELSE IF @DATE BETWEEN
CONVERT(CHAR(4),@YEAR)+'0621'
    AND CONVERT(CHAR(4),@YEAR)+'0922'
    SET @SEASON = 'Summer'

    ELSE
    SET @SEASON = 'Winter'

    UPDATE DIM_TIME SET time_weekend = @WEEKEND
    WHERE time_idsk = @ID

    UPDATE DIM_TIME SET time_season_year = @SEASON
    WHERE time_idsk = @ID

  FETCH NEXT FROM C_TIME
  INTO @ID, @DATE, @DAYWEEK, @YEAR
END
CLOSE C_TIME
DEALLOCATE C_TIME
GO
```

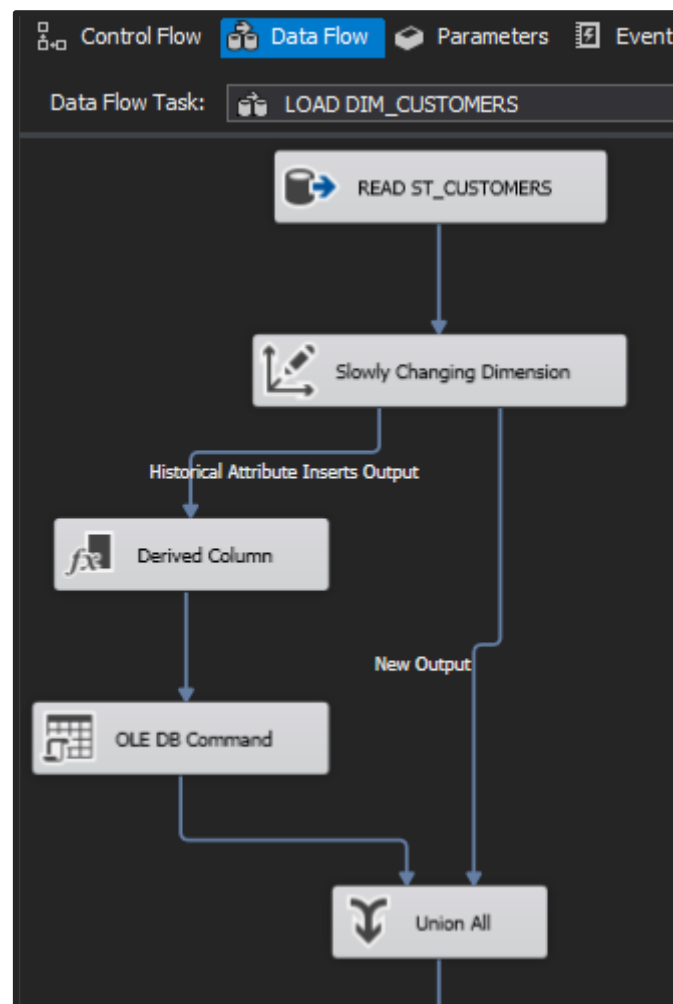


Customers Dimension

For illustrative purposes, I added the use of the Slowly Changing Dimension (SCD) to the customer's dimension. This tool allows database records not to lose references when a customer or product changes status.

It is quite common to find this type of inconsistency in modeling where the place of purchase is defined by the customer's address. In this example, a customer would have moved from Vancouver to Toronto in January 2023.

All purchases prior to the customer switch would now appear as the City of Toronto sales, not just subsequent purchases. With Slowly Changing Dimension (SCD) the database automatically creates a new record for the same customer in the customer's table, maintaining consistency.



Orders Dimension (It's actually a Fact)

To load the fact table I created a procedure for reading the data from the Stage area and comparing it with what would be inserted in the DW. This Procedure was executed in SSIS.



```
CREATE PROC LOAD_ORDERS
AS

DECLARE @FINAL_DATE DATETIME
DECLARE @START_DATE DATETIME

set @FINAL_DATE = (SELECT MAX(TIME_DATE)
FROM SELL_DW.DBO.DIM_TIME T)

set @START_DATE = (SELECT MAX(TIME_DATE)
FROM SELL_DW.DBO.DIM_ORDERS FT
JOIN SELL_DW.DBO.DIM_TIME T ON
(FT.CUSTOMER_CREATED_TIME_IDSK=T.TIME_IDSK))

IF @START_DATE IS NULL
BEGIN
    set @START_DATE = (SELECT MIN(TIME_DATE)
FROM SELL_DW.DBO.DIM_TIME T)
END

INSERT INTO SELL_DW.DBO.DIM_ORDERS(
    ORDER_ID,
    ORDER_CREATED_TIME_IDSK,
    CURRENCY_CODE_ID,
    ORDER_SUBTOTAL,
    ORDER_TOTAL,
    FULFILLMENT_STATUS_ID,
    ORDER_CUSTOMER_ID,
    CUSTOMER_CREATED_TIME_IDSK
)
SELECT
    O.ORDER_ID AS ORDER_ID,
    T1.TIME_IDSK AS ORDER_CREATED_TIME_IDSK,
    CC.CURRENCY_CODE_ID AS CURRENCY_CODE_ID,
    O.ORDER_SUBTOTAL AS ORDER_SUBTOTAL,
    O.ORDER_TOTAL AS ORDER_TOTAL,
    FS.FULFILLMENT_STATUS_ID AS FULFILLMENT_STATUS_ID,
    C.CUSTOMER_IDSK as ORDER_CUSTOMER_ID,
    T2.TIME_IDSK as CUSTOMER_CREATED_TIME_IDSK

FROM
    SELL_STAGE.DBO.ST_ORDERS O

    INNER JOIN DBO.DIM_CUSTOMERS C
```



```
on (O.ORDER_CUSTOMER_ID=C.CUSTOMER_ID)

INNER JOIN DBO.DIM_FULFILLMENT_STATUS FS
on (O.FULFILLMENT_STATUS_ID=FS.FULFILLMENT_STATUS_ID)

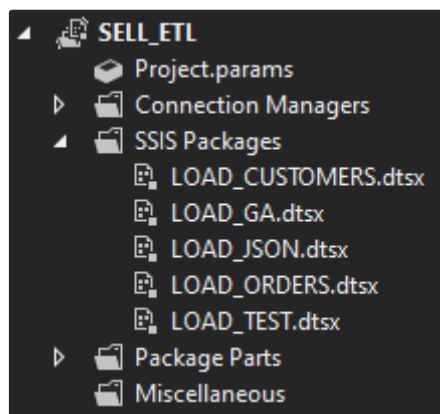
INNER JOIN DBO.DIM_CURRENCY_CODES CC
on (O.CURRENCY_CODE_ID=CC.CURRENCY_CODE_ID)

INNER JOIN DBO.DIM_TIME T1
ON (CONVERT(VARCHAR, T1.TIME_DATE,102) =
CONVERT(VARCHAR,
O.ORDER_CREATED_AT,102))
and O.ORDER_CREATED_AT BETWEEN @START_DATE AND
@FINAL_DATE

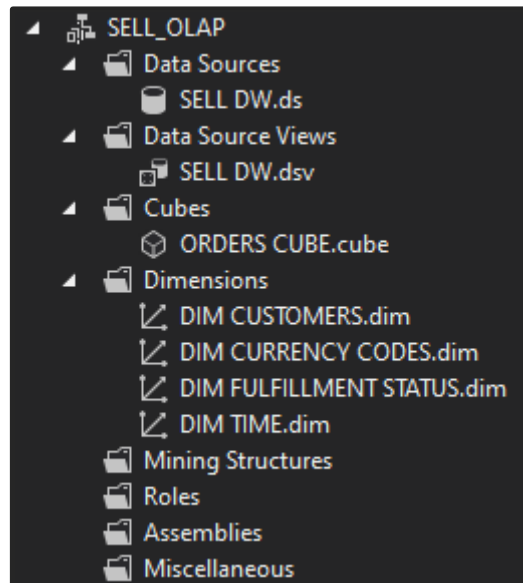
INNER JOIN DBO.DIM_TIME T2
ON (CONVERT(VARCHAR, T2.TIME_DATE,102) =
CONVERT(VARCHAR,
O.ORDER_CUSTOMER_CREATED_AT,102))
and O.ORDER_CUSTOMER_CREATED_AT BETWEEN @START_DATE
AND @FINAL_DATE
GO
```

Visual Studio Packages

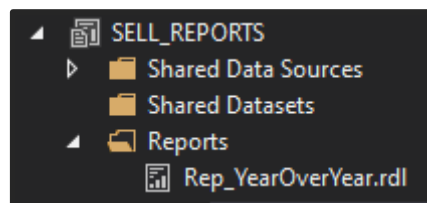
- ETL Packages



- OLAP Packages

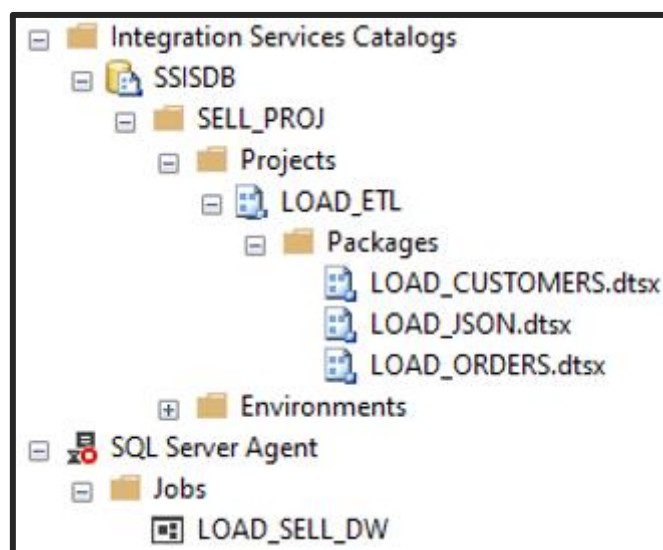


- Report Packages



SQL Server Agent e Integration Services Catalogs

Management Studio tools were used to automate the loads of packages in a specific job for the Data Warehouse.





Google Analytics CSV

To import the file 'Google Analytics.csv' I chose to process the data using Python with the Pandas and NumPy libraries. Finally, I used PyODBC to insert each line directly into the DW.

This model inserts line by line which made the import take 10 minutes. In a routine process, I would do a bulk insert using the SQLAlchemy library.

Python script used to import file:

```
import pandas as pd
import pyodbc
import numpy as np

server = 'localhost'
database = 'SELL_DW'
username = 'sa'
password = '@1q2w3e4r'
driver= '{ODBC Driver 13 for SQL Server}'
cnxn0 =
pyodbc.connect('DRIVER='+driver+';SERVER='+server+';PORT=1433;DATABASE='
+database+';UID='+username+';PWD='+ password)
cnxn1 =
pyodbc.connect('DRIVER='+driver+';SERVER='+server+';PORT=1433;DATABASE='
+database+';UID='+username+';PWD='+ password)
print('Started')

df = pd.read_csv('c:/SELL/Data/Google Analytics.csv',quotechar='')
df.rename(columns = {'Date':'GA_DATE'}, inplace = True)
df.rename(columns = {'Entrances':'GA_ENTRANCES'}, inplace = True)
df.rename(columns = {'Landing Page':'GA_LANDING_PAGE'}, inplace = True)
df.rename(columns = {'Sessions':'GA_SESSIONS'}, inplace = True)
df.rename(columns = {'Bounces':'GA_BOUNCES'}, inplace = True)
df.rename(columns = {'clicks':'GA_CLICKS'}, inplace = True)
df.rename(columns = {'impressions':'GA_IMPRESSIONS'}, inplace = True)
df.rename(columns = {'Time On page (seconds)':'GA_TIME_ON_PAGE'}, inplace =
True)
df.rename(columns = {'Pageviews':'GA_PAGEVIEWS'}, inplace = True)
df.rename(columns = {'Exits':'GA_EXITS'}, inplace = True)
df = df.fillna(0)

df['GA_LANDING_PAGE_T'] = df['GA_LANDING_PAGE'].str.split(' ').str[0]
df['GA_LANDING_PAGE_T2'] = df['GA_LANDING_PAGE_T'].str.split('https').str[0]
```



```
df['GA_URL_CLEAR'] = np.where(df['GA_LANDING_PAGE_T2'].str.find('?')>0,
df['GA_LANDING_PAGE_T2'].str.split('?').str[0],
    np.where(df['GA_LANDING_PAGE_T2'].str.find('#') > 0,
df['GA_LANDING_PAGE_T2'].str.split('#').str[0],
    df['GA_LANDING_PAGE_T2']))

df['GA_URL_CLEAR_T'] =
df['GA_URL_CLEAR'].str.replace('www.sellboardgames.com/', '', regex=True)
df['GA_CATEGORY'] = df['GA_URL_CLEAR_T'].str.split('/').str[0]

df['GA_VALUE'] = df['GA_URL_CLEAR_T'].str.split('/').str[-1]

cursor0 = cnxn0.cursor()
for index,row in df.iterrows():
    cursor0.execute("INSERT INTO
dbo.DW_FACT_GA(GA_DATE,GA_ENTRANCES,GA_LANDING_PAGE,GA_SES
SIONS,GA_BOUNCES,GA_CLICKS,GA_IMPRESSIONS,GA_TIME_ON_PAGE,G
A_PAGEVIEWS,GA_EXITS,GA_URL_CLEAR,GA_CATEGORY,GA_VALUE)
values (?,?,?,?,?,?,?,?,?,?,?,?,?)",
row['GA_DATE'],row['GA_ENTRANCES'],row['GA_LANDING_PAGE'],row['GA_S
SESSIONS'],row['GA_BOUNCES'],row['GA_CLICKS'],row['GA_IMPRESSIONS'],row
['GA_TIME_ON_PAGE'],row['GA_PAGEVIEWS'],row['GA_EXITS'],row['GA_URL_
CLEAR'],row['GA_CATEGORY'],row['GA_VALUE'])
    cnxn0.commit()
cursor0.close()

cursor1 = cnxn1.cursor()
cursor1.execute("EXEC LOAD_GA_AGGREGATE")
cnxn1.commit()
cursor1.close()
print('Finished')
```

At the end of the script and run a procedure that creates the aggregation tables in the data warehouse. This Procedure was executed in .py.

```
USE SELL_DW
GO

CREATE PROC LOAD_GA_AGGREGATE
AS

drop table LOAD_GA_AGGREGATE_1
drop table LOAD_GA_AGGREGATE_2
```





```
select
    row_number() over (order by TOTAL_GA_SESSIONS desc) LINHA,
    GA_VALUE,
    TOTAL_GA_SESSIONS,
    TOTAL_SESSION_GROUP,
    TOTAL_GA_SESSIONS/(TOTAL_SESSION_GROUP*1.0) PCT_SESSION
into LOAD_GA_AGGREGATE_1
from
    DASH_GA_AGGREGATE
    cross join (select sum(TOTAL_GA_SESSIONS) TOTAL_SESSION_GROUP
from DASH_GA_AGGREGATE) t
```

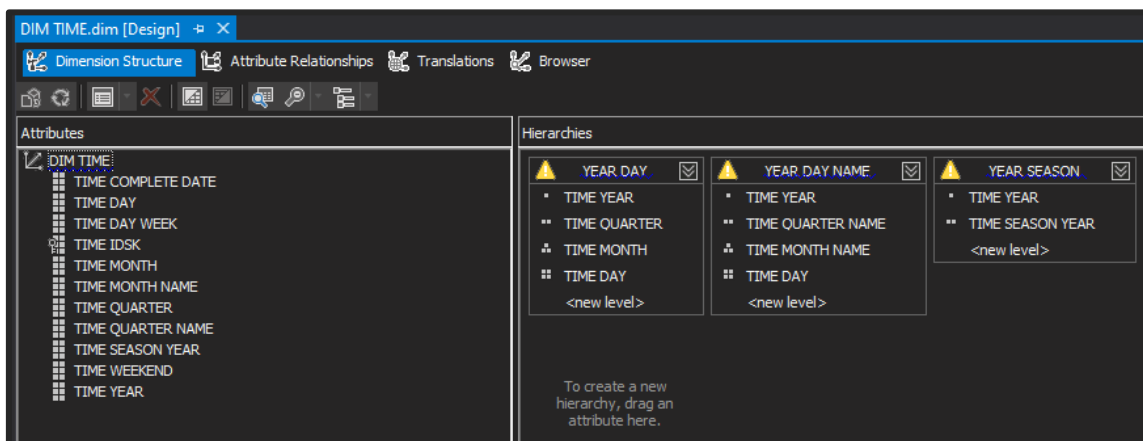
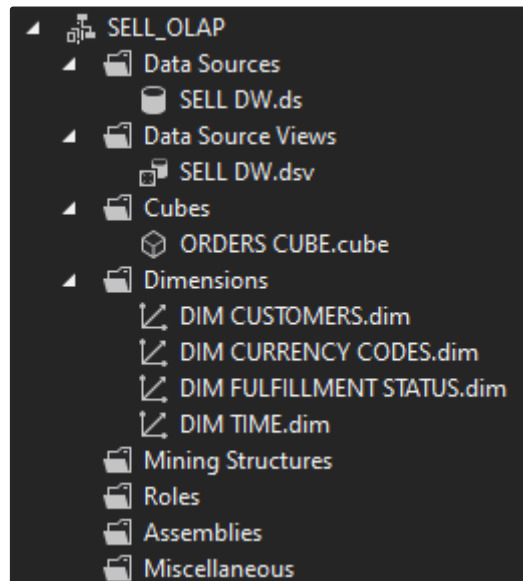
```
select
    t1.LINHA,sum(t2.PCT_SESSION) TOTAL_PCT_SESSION,
    (case
        when sum(t2.PCT_SESSION)<= 0.8 then 'A'
        when sum(t2.PCT_SESSION)<=0.95 then 'B'
        else 'C'
    end
    ) PARETO
into LOAD_GA_AGGREGATE_2
from
    LOAD_GA_AGGREGATE_1 t1
    inner join LOAD_GA_AGGREGATE_1 t2 on t1.LINHA >= t2.LINHA
group by
    t1.LINHA
```

GO



Phase 03: Analysis Services

The next step was completely focused on creating the cube in Analysis Services. This cube will be the basis for analysis in Power BI and its strong point is the hierarchies provided by the Time dimension.

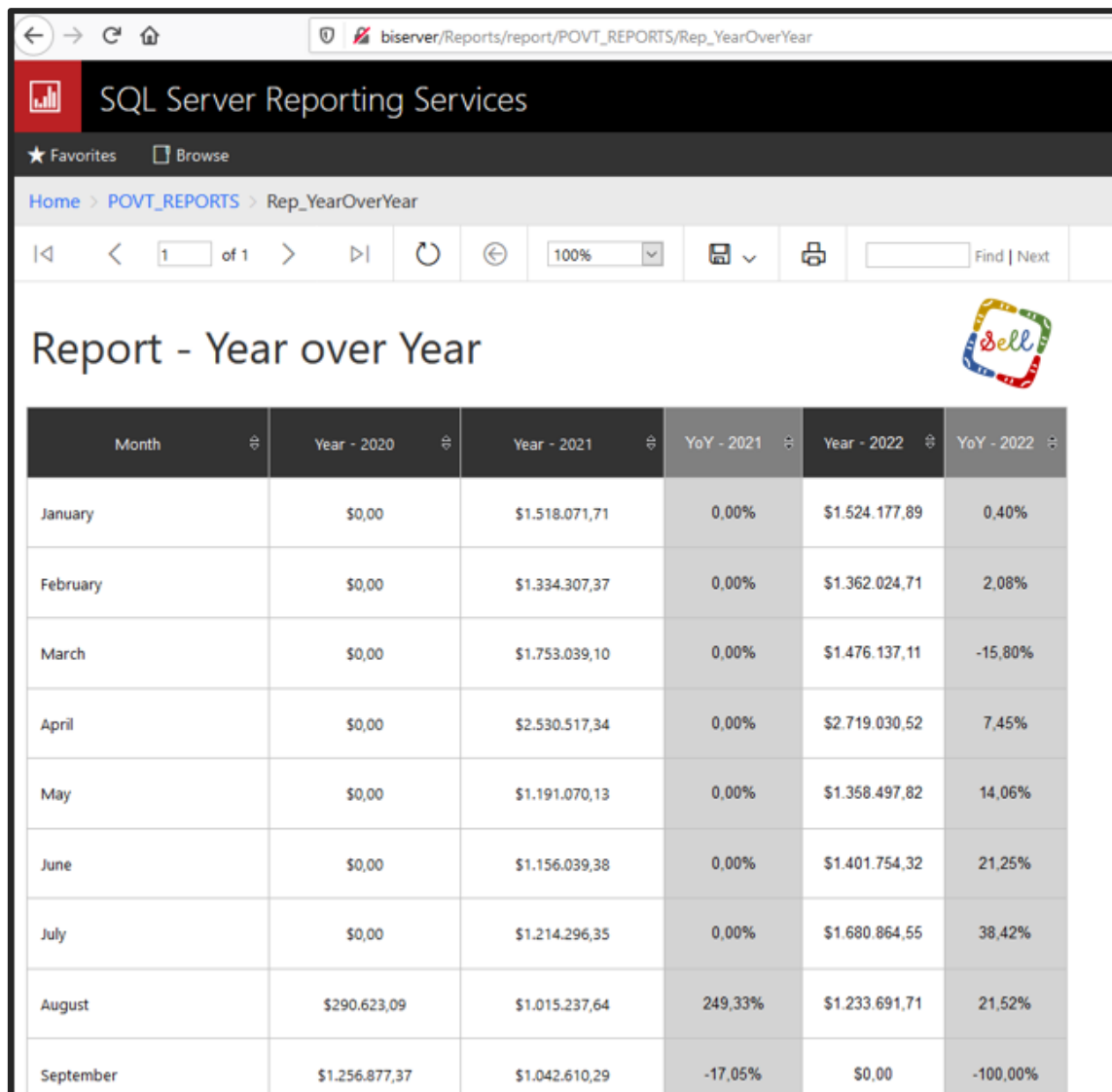




Phase 04: Reporting Services

In Reporting Services, the focus was on setting up the entire structure and developing a first report layout, following the same as in this document.

In order to start solving the proposed business problems, at this stage the developed report seeks to provide the results comparing the month vs the same month of the previous year.



The screenshot shows a web browser displaying a report from SQL Server Reporting Services. The browser address bar shows the URL: biserver/Reports/report/POVT_REPORTS/Rep_YearOverYear. The report title is 'Report - Year over Year'. The report contains a table with 6 columns: Month, Year - 2020, Year - 2021, YoY - 2021, Year - 2022, and YoY - 2022. The data is presented for the months of January through September.

Month	Year - 2020	Year - 2021	YoY - 2021	Year - 2022	YoY - 2022
January	\$0,00	\$1.518.071,71	0,00%	\$1.524.177,89	0,40%
February	\$0,00	\$1.334.307,37	0,00%	\$1.362.024,71	2,08%
March	\$0,00	\$1.753.039,10	0,00%	\$1.476.137,11	-15,80%
April	\$0,00	\$2.530.517,34	0,00%	\$2.719.030,52	7,45%
May	\$0,00	\$1.191.070,13	0,00%	\$1.358.497,82	14,06%
June	\$0,00	\$1.156.039,38	0,00%	\$1.401.754,32	21,25%
July	\$0,00	\$1.214.296,35	0,00%	\$1.680.864,55	38,42%
August	\$290.623,09	\$1.015.237,64	249,33%	\$1.233.691,71	21,52%
September	\$1.256.877,37	\$1.042.610,29	-17,05%	\$0,00	-100,00%



To illustrate the layout highlighting the company's colors and differentiating the YoY columns to seek user focus.

Report - Year over Year

Month	Year - 2020	Year - 2021	YoY - 2021	Year - 2022	YoY - 2022
January	\$0,00	\$1.518.071,71	0,00%	\$1.524.177,89	0,40%
February	\$0,00	\$1.334.307,37	0,00%	\$1.362.024,71	2,08%
March	\$0,00	\$1.753.039,10	0,00%	\$1.476.137,11	-15,80%
April	\$0,00	\$2.530.517,34	0,00%	\$2.719.030,52	7,45%
May	\$0,00	\$1.191.070,13	0,00%	\$1.358.497,82	14,08%
June	\$0,00	\$1.156.039,38	0,00%	\$1.401.754,32	21,25%
July	\$0,00	\$1.214.296,35	0,00%	\$1.680.884,55	38,42%
August	\$290.623,09	\$1.015.237,64	249,33%	\$1.233.891,71	21,52%
September	\$1.256.877,37	\$1.042.610,29	-17,05%	\$0,00	-100,00%
October	\$1.414.594,88	\$1.224.508,02	-13,44%	\$0,00	-100,00%
November	\$2.465.751,53	\$2.937.240,44	19,12%	\$0,00	-100,00%
December	\$1.866.572,25	\$1.921.891,12	2,98%	\$0,00	-100,00%

1

For the data in this report, the following view was developed using the T-SQL pivot resource.

```
CREATE VIEW REP_YEAR_BY_YEAR AS
select
    TIME_MONTH,
    TIME_MONTH_NAME,
```



```
isnull([2020],0) Y_2020,
isnull([2021],0) Y_2021,
isnull([2022],0) Y_2022
from
(
select
    TIME_YEAR,
    TIME_MONTH,
    TIME_MONTH_NAME,
    sum(ORDER_TOTAL) ORDER_TOTAL
from
    DIM_ORDERS o
    inner join DIM_TIME t on o.ORDER_CREATED_TIME_IDSK =
t.TIME_IDSK
group by
    TIME_YEAR,
    TIME_MONTH,
    TIME_MONTH_NAME
) t
PIVOT
(
    SUM(ORDER_TOTAL)
FOR TIME_YEAR IN
    (
        [2020],
        [2021],
        [2022]
    )
) AS pivot_table
```



Phase 05: Backup Structure

One of the tasks I developed was the backup structure for the databases. This structure was important for information security during the project development period and the bank's availability for the SELL team.

This backup model is simple but very efficient in developments like this.

Backup Scripts

Batch .bat files have been developed that execute SQL scripts and maintain the record in a log file. The developed scripts follow.

Backup_SELL_DW.bat and Backup_SELL_STAGE.bat

These files are similar, only the target database changes. These scripts were programmed to run daily.

```
echo Start Backup SELL_DW .....  
%date:~0,2%/ %date:~3,2%/ %date:~6,4%-%time:~1,7% >> c:\BACKUP\logbackup.txt
```

```
sqlcmd -S localhost -e -d SELL_DW -U sa -P "#####" -i  
"C:\BACKUP\BACKUP_SELL_DW.SQL"
```

```
:: move C:\BACKUP\BACKUP_TEMP\*.bak "C:\BACKUP"
```

```
:: forfiles -p "C:\BACKUP\" -d -15 -c "cmd /c del /f /q @path"
```

```
echo Finish Backup SELL_DW .....  
%date:~0,2%/ %date:~3,2%/ %date:~6,4%-%time:~1,7% >> c:\BACKUP\logbackup.txt
```

BACKUP_SELL_DW.sql and BACKUP_SELL_STAGE.sql

SQL scripts responsible for backing up the database.

```
DECLARE @ARQUIVO CHAR(250)
```

```
SET @ARQUIVO = 'C:\BACKUP\' + 'SELL_DW_' +
```

```
SUBSTRING(CONVERT(CHAR(10),GETDATE(),103),7,4) + '_' +  
SUBSTRING(CONVERT(CHAR(10),GETDATE(),103),4,2) + '_' +  
SUBSTRING(CONVERT(CHAR(10),GETDATE(),103),1,2) +  
'_' +  
SUBSTRING(CONVERT(CHAR(12),GETDATE(),114),1,2) + '_' +
```

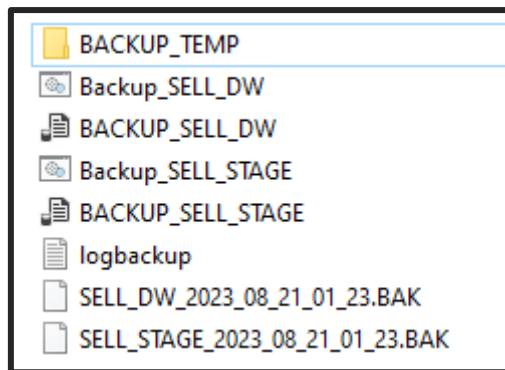


```
SUBSTRING(CONVERT(CHAR(12),GETDATE(),114),4,2) + '.BAK'
```

```
BACKUP DATABASE SELL_DW TO DISK = @ARQUIVO WITH NOFORMAT,  
INIT, NAME = N'C:\BACKUP', SKIP, NOREWIND, NOUNLOAD, STATS = 10  
GO
```

Backup folder structure

Structure with batch files, SQL scripts, log and backups generated with date and time nomenclature.



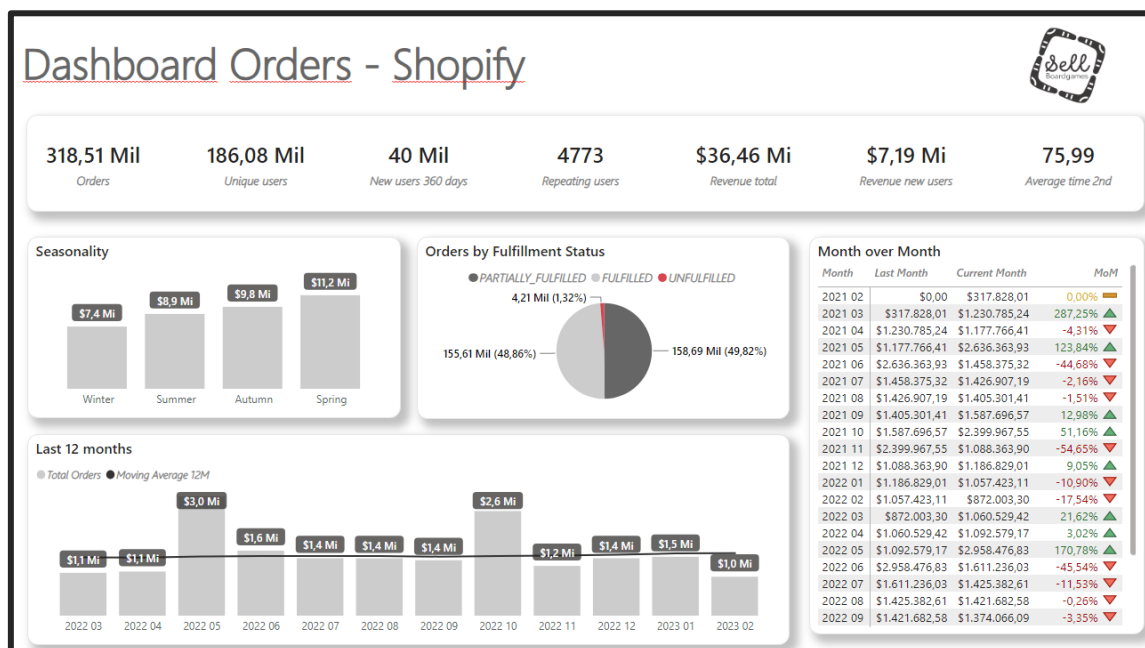


Phase 06: Power BI

After all the developments related to data manipulation and governance, we finally arrived at the last tool to be used in this project.

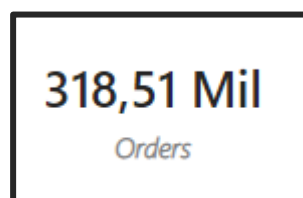
During the period waiting for the google analytics file, I made a second version of the dashboard, changing only the look.

Dashboard Orders – Shopify



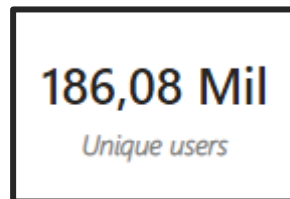
Orders

This indicator is calculated with total purchases. This indicator has been placed on a card in the top bar.



Unique users

This indicator is calculated with the distinct count in the CUSTOMER_ID of the DIM_ORDERS dimension. This indicator has been placed on a card in the top bar.

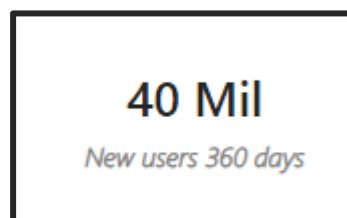


New users

I defined this metric as the number of customers who made their first purchase in the last 360 days. This indicator has been placed on a card in the top bar.

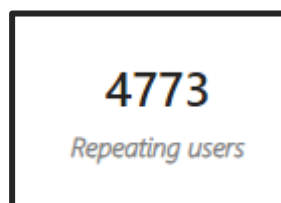
DAX function used to generate the measure:

FIRST_PURCHASE_360DAYS =
calculate(DISTINCTCOUNT(DASH_BASE_CUSTOMERS_ORDERS[CUSTOMER_ID]);DASH_BASE_CUSTOMERS_ORDERS[number_purchase]=1;datediff(DASH_BASE_CUSTOMERS_ORDERS[TIME_DATE];NOW();DAY)<=360)



Repeating users

This indicator is calculated with the distinct count in the CUSTOMER_ID of the DASH_REPEATING_USERS view. This indicator has been placed on a card in the top bar.



Revenue Total

This indicator is calculated with a sum on the ORDER_TOTAL of the DIM_ORDERS dimension. This indicator has been placed on a card in the top bar.



\$36,46 Mi

Revenue total

Revenue only new user

This indicator is calculated with a sum in the ORDER_TOTAL of the DASH_PURCHASE_NEW_USERS view. This indicator has been placed on a card in the top bar.

\$7,19 Mi

Revenue new users

Average time between the first and second purchase

I used AVERAGEX to perform the average between the dates of the first and second purchase. This indicator has been placed on a card in the top bar.

DAX function used to generate the measure:

```
AVERAGE_TIME =  
calculate(AVERAGEX(DASH_BASE_CUSTOMERS_ORDERS;datediff(DASH_BASE_CUSTOMERS_ORDERS[  
LAST_SELL];DASH_BASE_CUSTOMERS_ORDERS[  
TIME_DATE];DAY));DASH_BASE_CUSTOMERS_ORDERS[NUMBER_PURCHASE]=2)
```

75,99

Average time 2nd

Results comparing month vs previous month.

To assemble this table I used the view DASH_MOVING_12M which has the total sales in the current month and the previous month.



DAX function used to generate the measure:

MoM =

$$\text{if}(\text{sum}(\text{DASH_MOVING_12M}[\text{ORDERS_1}])=0;0;\text{sum}(\text{DASH_MOVING_12M}[\text{ORDERS}])/\text{sum}(\text{DASH_MOVING_12M}[\text{ORDERS_1}])-1)$$

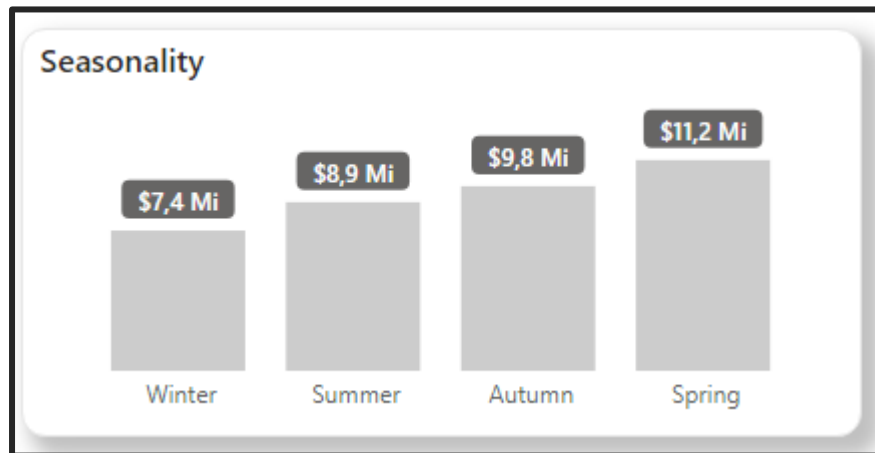
Month	Last Month	Current Month	MoM
2021 02	\$0,00	\$317.828,01	0,00% ▬
2021 03	\$317.828,01	\$1.230.785,24	287,25% ▲
2021 04	\$1.230.785,24	\$1.177.766,41	-4,31% ▼
2021 05	\$1.177.766,41	\$2.636.363,93	123,84% ▲
2021 06	\$2.636.363,93	\$1.458.375,32	-44,68% ▼
2021 07	\$1.458.375,32	\$1.426.907,19	-2,16% ▼
2021 08	\$1.426.907,19	\$1.405.301,41	-1,51% ▼
2021 09	\$1.405.301,41	\$1.587.696,57	12,98% ▲
2021 10	\$1.587.696,57	\$2.399.967,55	51,16% ▲
2021 11	\$2.399.967,55	\$1.088.363,90	-54,65% ▼
2021 12	\$1.088.363,90	\$1.186.829,01	9,05% ▲
2022 01	\$1.186.829,01	\$1.057.423,11	-10,90% ▼
2022 02	\$1.057.423,11	\$872.003,30	-17,54% ▼
2022 03	\$872.003,30	\$1.060.529,42	21,62% ▲
2022 04	\$1.060.529,42	\$1.092.579,17	3,02% ▲
2022 05	\$1.092.579,17	\$2.958.476,83	170,78% ▲
2022 06	\$2.958.476,83	\$1.611.236,03	-45,54% ▼
2022 07	\$1.611.236,03	\$1.425.382,61	-11,53% ▼
2022 08	\$1.425.382,61	\$1.421.682,58	-0,26% ▼
2022 09	\$1.421.682,58	\$1.374.066,09	-3,35% ▼

Results comparing the month vs the same month of the previous year.

Delivered in previous chapter using Reporting Services.

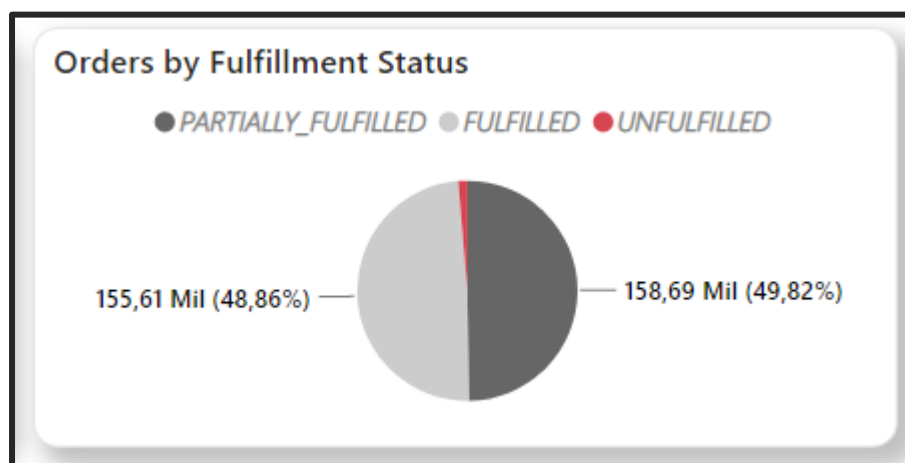
Seasonality

I used the TIMES_SEASON_YEAR column to get this view. It is exciting to observe the summer with the lowest sales, my expectation was that it would be the greatest period.



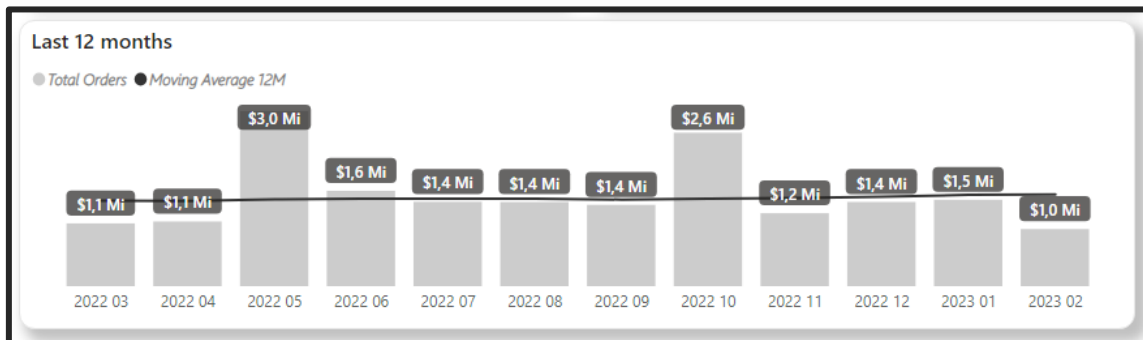
Orders by Fulfillment Status

An important graph to monitor the number of Unfulfilled sales in the middle of all orders.

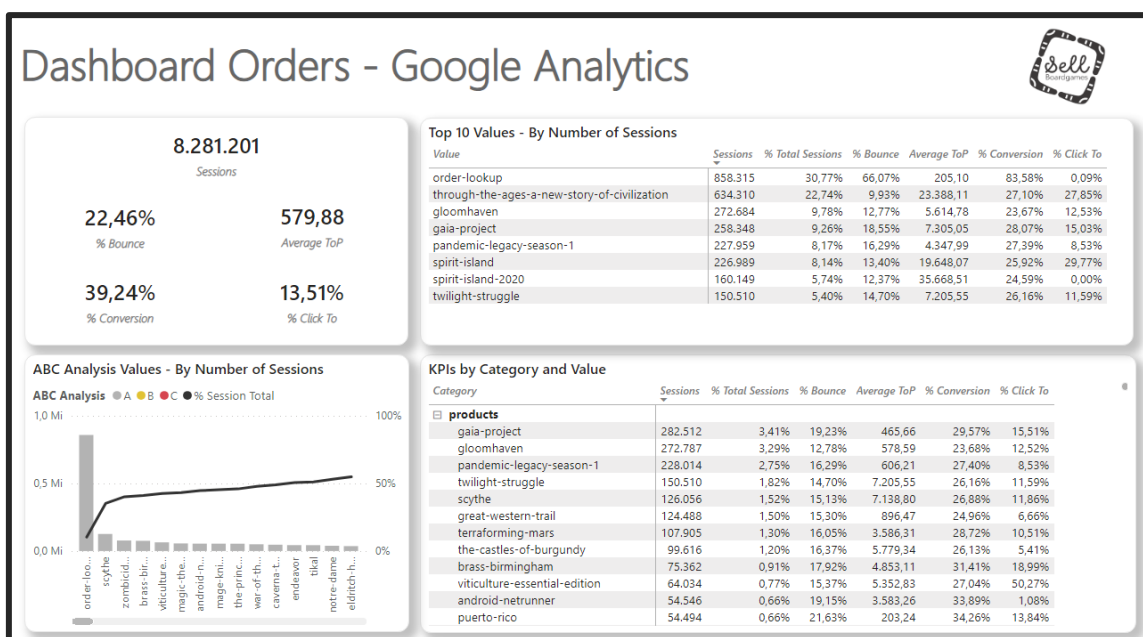


Moving Average 12 months

The 12M moving average is a way to look at the company's trend. With the data presented, we see stability in sales with small but continuous growth in these 12 months.

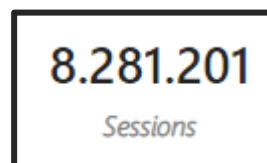


Dashboard Orders – Google Analytics



Sessions

This indicator is calculated with total sessions. This indicator has been placed on a card in the top left.



Bounce Rate

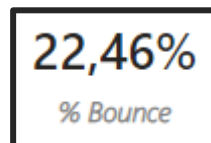
This indicator is calculated with total bounces divided by total entrances. This indicator has been placed on a card in the top left.





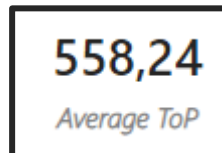
DAX function used to generate the measure:

`BOUNCE_RATE =
sum(DASH_GA_AGGREGATE[TOTAL_GA_BOUNCES])/sum(DASH_GA_AGGREGATE[TOTAL_GA_ENTRANCES])`



AVG Time on Page

This indicator is calculated with the avg aggregation in SLQ view. This indicator has been placed on a card in the top left.

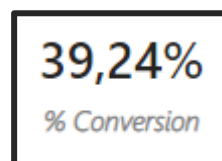


% Conversion Rate

This indicator is calculated with total sessions divided by total entrances. This indicator has been placed on a card in the top left.

DAX function used to generate the measure:

`CONVERSION_RATE =
sum(DASH_GA_AGGREGATE[TOTAL_GA_SESSIONS])/sum(DASH_GA_AGGREGATE[TOTAL_GA_ENTRANCES])`



% Click to rate

This indicator is calculated with total clicks divided by total sessions. This indicator has been placed on a card in the top left.

DAX function used to generate the measure:



CLICKTO_RATE =
sum(DASH_GA_AGGREGATE[TOTAL_GA_CLICKS])/sum(DASH_GA_AGGREGATE[TOTAL_GA_SESSIONS])

13,51%
% Click To

KPIs by Category and Value

This table allows viewing the indicators opened by category hierarchy followed by values.

Category	Sessions	% Total Sessions	% Bounce	Average ToP	% Conversion	% Click To
products						
gaia-project	282.512	3,41%	19,23%	465,66	29,57%	15,51%
gloomhaven	272.787	3,29%	12,78%	578,59	23,68%	12,52%
pandemic-legacy-season-1	228.014	2,75%	16,29%	606,21	27,40%	8,53%
twilight-struggle	150.510	1,82%	14,70%	7.205,55	26,16%	11,59%
scythe	126.056	1,52%	15,13%	7.138,80	26,88%	11,86%
great-western-trail	124.488	1,50%	15,30%	896,47	24,96%	6,66%
terraforming-mars	107.905	1,30%	16,05%	3.586,31	28,72%	10,51%
the-castles-of-burgundy	99.616	1,20%	16,37%	5.779,34	26,13%	5,41%
brass-birmingham	75.362	0,91%	17,92%	4.853,11	31,41%	18,99%
viticulture-essential-edition	64.034	0,77%	15,37%	5.352,83	27,04%	50,27%
android-netrunner	54.546	0,66%	19,15%	3.583,26	33,89%	1,08%
puerto-rico	54.494	0,66%	21,63%	203,24	34,26%	13,84%

Top 10 Values – By Number of Sessions

In this table, the focus is on identifying the 10 values with the greatest impact on the business according to the number of sessions. It is possible to observe all the main indicators.

Value	Sessions	% Total Sessions	% Bounce	Average ToP	% Conversion	% Click To
order-lookup	858.315	30,77%	66,07%	205,10	83,58%	0,09%
through-the-ages-a-new-story-of-civilization	634.310	22,74%	9,93%	23.388,11	27,10%	27,85%
gloomhaven	272.684	9,78%	12,77%	5.614,78	23,67%	12,53%
gaia-project	258.348	9,26%	18,55%	7.305,05	28,07%	15,03%
pandemic-legacy-season-1	227.959	8,17%	16,29%	4.347,99	27,39%	8,53%
spirit-island	226.989	8,14%	13,40%	19.648,07	25,92%	29,77%
spirit-island-2020	160.149	5,74%	12,37%	35.668,51	24,59%	0,00%
twilight-struggle	150.510	5,40%	14,70%	7.205,55	26,16%	11,59%

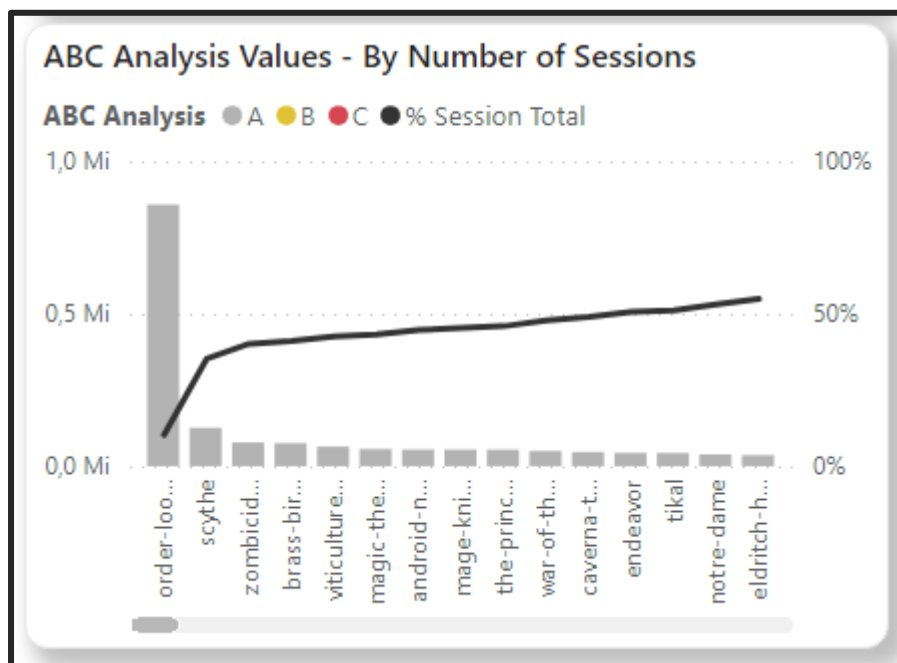


ABC Analysis Values - By Number of Sessions

It is important to use the ABC analysis to prioritize the main points that generate returns and, above all, to remove items that are not in demand by users. In its precept, 20% of the items correspond to approximately 80% of the result.

In this chart, we have the Values in descending order in the columns with the growth of the percentage of sessions in the line.

The columns are colored gray for curve A, yellow for curve B and red for curve C..





Views for Power BI

To carry out the Power BI dashboards, I created 6 views treating the data still in the DW.

- DASH_BASE_CUSTOMERS_ORDERS – Organizing the already filtered data with fulfillment <> of unfulfilled. I created two items focused on its performance.
- DASH_REPEATING_USERS - Filtering only customers who made at least 3 purchases in the last 360 days.
- DASH_PURCHASE_NEW_USERS – Filtering only sales from customers who made their first Purchase less than 360 days ago.
- DASH_MOVING_12M – They are used for the 12-month Moving Average and Month over Month comparison.
- DASH_GA_AGGREGATE - Performs the aggregation of the Google Analytics file.
- DASH_GA_AGGREGATE_PARETO - Prepares data for ABC Analysis.

```
CREATE NONCLUSTERED INDEX [IX_DIM_TIME] ON [dbo].[DIM_TIME]
(TIME_IDSK ASC)INCLUDE(TIME_DATE)
drop index [IX_DIM_ORDERS]
```

```
CREATE NONCLUSTERED INDEX [IX_DIM_ORDERS] ON [dbo].[DIM_ORDERS]
(FULFILLMENT_STATUS_ID ASC,ORDER_CREATED_TIME_IDSK
ASC,ORDER_CUSTOMER_ID ASC)INCLUDE(ORDER_TOTAL)
```

```
CREATE VIEW DASH_BASE_CUSTOMERS_ORDERS as
WITH CTE_BASE_CUSTOMERS_ORDERS as (
SELECT
    CUSTOMER_ID,
    TIME_DATE,
    ORDER_TOTAL
FROM
    (select
    ORDER_TOTAL,ORDER_CREATED_TIME_IDSK,ORDER_CUSTOMER_ID from
    DIM_ORDERS WITH(INDEX([IX_DIM_ORDERS])) where
    FULFILLMENT_STATUS_ID <> 3) o
    INNER JOIN
        (select TIME_IDSK,TIME_DATE from DIM_TIME
        WITH(INDEX([IX_DIM_TIME]))
        WHERE
            TIME_IDSK >= (select min(ORDER_CREATED_TIME_IDSK)
from DIM_ORDERS) and
            TIME_IDSK < (select max(ORDER_CREATED_TIME_IDSK)
from DIM_ORDERS)
        ) t on o.ORDER_CREATED_TIME_IDSK = t.TIME_IDSK
    inner join DIM_CUSTOMERS c on o.ORDER_CUSTOMER_ID =
c.CUSTOMER_IDSK
```



```
)
select
    CUSTOMER_ID,
    TIME_DATE,
    concat(year(TIME_DATE),'',(case when month(TIME_DATE)<10 then '0' else"
end),month(TIME_DATE)) YEAR_MONTH,
    concat(year(TIME_DATE),(case when month(TIME_DATE)<10 then '0' else"
end),month(TIME_DATE)) ORDER_YEAR_MONTH,
    ROW_NUMBER() over(partition by CUSTOMER_ID order by
CUSTOMER_ID,TIME_DATE) NUMBER_PURCHASE,
    LAG(CUSTOMER_ID, 1) over(order by CUSTOMER_ID,TIME_DATE) AS
LAST_CUSTOMER,
    LAG(TIME_DATE, 1) over(order by CUSTOMER_ID,TIME_DATE) AS
LAST_SELL,
    ORDER_TOTAL
from
    CTE_BASE_CUSTOMERS_ORDERS
```

```
CREATE VIEW DASH_REPEATING_USERS as
select
    CUSTOMER_ID,
    count(CUSTOMER_ID) PURCHASES_LAST360DAYS
from
    DASH_BASE_CUSTOMERS_ORDERS
where
    DATEDIFF(day,time_date,getdate()) <= 360
group by
    CUSTOMER_ID
having
    count(CUSTOMER_ID)>=3
```

```
CREATE VIEW DASH_PURCHASE_NEW_USERS as
select
    t.CUSTOMER_ID,
    sum(ORDER_TOTAL) ORDER_TOTAL
from
    DASH_BASE_CUSTOMERS_ORDERS t
inner join
    (
        select
            CUSTOMER_ID
        from
            DASH_BASE_CUSTOMERS_ORDERS
        where
            NUMBER_PURCHASE = 1
```



```

        and DATEDIFF(day,time_date,getdate()) <= 360
    ) t1 on t.CUSTOMER_ID = t1.CUSTOMER_ID
group by
    t.CUSTOMER_ID

```

```

CREATE VIEW DASH_MOVING_12M as
select

```

```

    YEAR_MONTH,
    ORDER_YEAR_MONTH,
    isnull(ORDERS,0) ORDERS,
    isnull(LAG(ORDERS, 1) over(order by YEAR_MONTH),0) AS ORDERS_1,
    isnull(LAG(ORDERS, 2) over(order by YEAR_MONTH),0) AS ORDERS_2,
    isnull(LAG(ORDERS, 3) over(order by YEAR_MONTH),0) AS ORDERS_3,
    isnull(LAG(ORDERS, 4) over(order by YEAR_MONTH),0) AS ORDERS_4,
    isnull(LAG(ORDERS, 5) over(order by YEAR_MONTH),0) AS ORDERS_5,
    isnull(LAG(ORDERS, 6) over(order by YEAR_MONTH),0) AS ORDERS_6,
    isnull(LAG(ORDERS, 7) over(order by YEAR_MONTH),0) AS ORDERS_7,
    isnull(LAG(ORDERS, 8) over(order by YEAR_MONTH),0) AS ORDERS_8,
    isnull(LAG(ORDERS, 9) over(order by YEAR_MONTH),0) AS ORDERS_9,
    isnull(LAG(ORDERS, 10) over(order by YEAR_MONTH),0) AS ORDERS_10,
    isnull(LAG(ORDERS, 11) over(order by YEAR_MONTH),0) AS ORDERS_11,
    isnull(ORDERS,0)+
    isnull(LAG(ORDERS, 1) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 2) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 3) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 4) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 5) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 6) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 7) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 8) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 9) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 10) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 11) over(order by YEAR_MONTH),0) AS TOTAL_12M
,
    (isnull(ORDERS,0)+
    isnull(LAG(ORDERS, 1) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 2) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 3) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 4) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 5) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 6) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 7) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 8) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 9) over(order by YEAR_MONTH),0)+
    isnull(LAG(ORDERS, 10) over(order by YEAR_MONTH),0)+

```




```
isnull(LAG(ORDERS, 11) over(order by YEAR_MONTH),0))/12 AS
AVG_12M,
(case when isnull(LAG(ORDERS, 1) over(order by YEAR_MONTH),0) = 0
then 0 else isnull(ORDERS,0)/isnull(LAG(ORDERS, 1) over(order by
YEAR_MONTH),0)-1 end) AS DELTA_MONTH
from
(
SELECT
YEAR_MONTH,
ORDER_YEAR_MONTH,
sum(ORDER_TOTAL) ORDERS
FROM
DASH_BASE_CUSTOMERS_ORDERS
group by
YEAR_MONTH,
ORDER_YEAR_MONTH
) t
```

```
CREATE VIEW DASH_GA_AGGREGATE as
select
GA_CATEGORY,
GA_VALUE,
GA_URL_CLEAR,
sum(GA_ENTRANCES) TOTAL_GA_ENTRANCES,
sum(GA_SESSIONS) TOTAL_GA_SESSIONS,
sum(GA_BOUNCES) TOTAL_GA_BOUNCES,
sum(GA_CLICKS) TOTAL_GA_CLICKS,
sum(GA_IMPRESSIONS) TOTAL_GA_IMPRESSIONS,
sum(GA_TIME_ON_PAGE) TOTAL_GA_TIME_ON_PAGE,
sum(GA_PAGEVIEWS) TOTAL_GA_PAGEVIEWS,
sum(GA_EXITS) TOTAL_GA_EXITS,
avg(GA_ENTRANCES) AVG_GA_ENTRANCES,
avg(GA_SESSIONS) AVG_GA_SESSIONS,
avg(GA_BOUNCES) AVG_GA_BOUNCES,
avg(GA_CLICKS) AVG_GA_CLICKS,
avg(GA_IMPRESSIONS) AVG_GA_IMPRESSIONS,
avg(GA_TIME_ON_PAGE) AVG_GA_TIME_ON_PAGE,
avg(GA_PAGEVIEWS) AVG_GA_PAGEVIEWS,
avg(GA_EXITS) AVG_GA_EXITS
from
DW_FACT_GA
group by
GA_CATEGORY,
GA_VALUE,
GA_URL_CLEAR
```



```
CREATE VIEW DASH_GA_AGGREGATE_PARETO as
select
    GA_VALUE,
    TOTAL_GA_SESSIONS,
    PCT_SESSION,
    TOTAL_PCT_SESSION,
    PARETO
from
    teste_CTE_GA_AGGREGATE t1
    inner join teste_CTE_GA_AGGREGATE_2 t2 on t1.linha = t2.linha
```



Thanks

Thank you for the opportunity, it was an honor to be able to analyze all the data from this fictitious company and to be able to show a little of my work. Thank you for accompanying the whole project and I am at your disposal if you want a presentation or ask any questions.

Warmly Regards,
Fernando Araujo