DSTI – A24

03/11/2024



Datawarehouse and ETL Project

using SSIS & SQL server

**Mohamed jouhari – vincent pinneau**

christine ismail – abdifatah osman farah

November, 3rd 2024

Contents

[1. INTRODUCTION 2](#_Toc181391787)

[1.A. DATA 2](#_Toc181391788)

[1- Orders 2](#_Toc181391789)

[2- Order\_items 2](#_Toc181391790)

[3- Order\_payments 3](#_Toc181391791)

[4- Customers 3](#_Toc181391792)

[5- Sellers 3](#_Toc181391793)

[6- Products 3](#_Toc181391794)

[7- Product\_category\_name\_translation 3](#_Toc181391795)

[8- Geolocation 4](#_Toc181391796)

[9- Order\_reviews 4](#_Toc181391797)

[1.B. TOOL 4](#_Toc181391798)

[1.C. METHOD 4](#_Toc181391799)

[2. STAGING DATABASE 5](#_Toc181391800)

[2.A. Importing orders dataset 5](#_Toc181391801)

[2.B. Other datasets 6](#_Toc181391802)

[2.C. Creating SQL Staging database 7](#_Toc181391803)

[2.D. Specificity for OrderReviews dataset 8](#_Toc181391804)

[3. OPERATIONAL DATA STORE (ODS) 10](#_Toc181391805)

[3.A. Extracting staging data 10](#_Toc181391806)

[4. DATE WAREHOUSE 14](#_Toc181391807)

[4.A. MODELING 14](#_Toc181391808)

[4.B. DIMENSION TABLES 16](#_Toc181391809)

[1) Geolocation table 17](#_Toc181391810)

[2) Date table 17](#_Toc181391811)

[3) Other dimensions table 17](#_Toc181391812)

[4) Creation of a dimension table 18](#_Toc181391813)

[4.C. Fact table 18](#_Toc181391814)

[1) Total invoiced 18](#_Toc181391815)

[2) Date keys 19](#_Toc181391816)

[3) Order\_id / Order\_item\_id 19](#_Toc181391817)

[4) Adding the technical keys and checking the relations 19](#_Toc181391818)

[5. PIPELINE CONSTRUCTION 21](#_Toc181391819)

[6. AUTOMATIZATION 21](#_Toc181391820)

[6.A. deployment 21](#_Toc181391821)

[6.B scheduling 22](#_Toc181391822)

[7. USE CASES 23](#_Toc181391823)

[7.A. Most and least successful product categories 23](#_Toc181391824)

[7.B. Impact of payment methods on completion times 25](#_Toc181391825)

[7.C. factors influencing delivery delays 26](#_Toc181391826)

1. INTRODUCTION

This project is aimed to address a data warehouse modeling and ETL pipeline creation problem. We have at our disposal several data sources containing information about a South American e-commerce website. The dataset are in a CSV format. The objective is to create an automated data pipeline that will fetch those data sources and fill the relevant business data into a data warehouse. This data pipeline should be scheduled and work as a routine task to automate the solution for the business.

## 1.A. DATA

For this project, locally stored csv files will act as data sources. We have the following:

1. Orders

The orders data source is split into three different csv files: orders\_2016.csv, orders\_2017.csv, orders\_2018.csv. They contain the columns:

* Order\_id : This is a unique identifier for each order
* Customer\_id : A unique identifier for the customer who made the order
* Order\_status: The status of the order whether delivered, in process, etc…
* Order\_purchase\_timestamp: The date when the order was placed
* Order\_approved\_at: The date when the purchase was approved by the system
* Order\_delivered\_carrier\_date: The date when the order was delivered to the carrier
* Order\_delivered\_customer\_date: the date when the order was delivered to the customer
* Order\_estimated\_delivery\_date: the estimated date for the order cycle until delivered to customer

1. Order\_items

This data source is contained in the order\_items\_dataset.csv. It contains the following columns:

* Order\_id : a unique identifier for each order
* Order\_item\_id: a unique identifier for the item in the specific order

Those two features work as a composite key together to select a specific item from an order.

* Product\_id : a unique identifier for the product purchased
* Seller\_id : a unique identifier for the seller of the product
* Shipping\_limit\_date: Latest date by which the order must be shipped
* Price : the price of the item
* Freight\_value: the shipping cost of the item

1. Order\_payments

This data source can be found in the order\_payments\_dataset.csv and it contains the following columns:

* Order\_id: a unique identifier for the order
* Payment\_sequential: sequential number of the payment made for an item
* Payment\_type: type of payment used for the transaction (cash, credit card, etc)
* Payment\_installments: number of installments used for the transaction
* Payment\_value: total value of the payment in local currency

1. Customers

This data source is in customers\_dataset.csv.

* Customer\_id : a unique identifier for the customer
* Customer\_unique\_id: an anonymized unique identifier for the customer
* Customer\_zip\_code\_prefix: ZIP code prefix of the customer
* Customer\_city : city of the customer
* Customer\_state: state of the customer

1. Sellers

This data source is located in sellers\_dataset.csv.

* Seller\_id : a unique identifier for the seller
* Seller\_zip\_code\_prefix: zip code of the seller location
* Seller\_city: city of the seller
* Seller\_state: state where the seller is located

1. Products

This contains information about the products that customers can purchase. This data source is enclosed in the products\_dataset.csv.

* product\_id: Unique identifier for each product.
* product\_category\_name: Category of the product (e.g., "perfumaria").
* product\_name\_lenght: The length of the product name in characters.
* product\_description\_lenght: The length of the product description in characters.
* product\_photos\_qty: Number of photos associated with the product.
* product\_weight\_g: Weight of the product in grams.
* product\_length\_cm: Length of the product in centimeters.
* product\_height\_cm: Height of the product in centimeters.
* product\_width\_cm: Width of the product in centimeters.

1. Product\_category\_name\_translation

This data source contains the translation of the original Portuguese names of the product categories in English, and it is found in the product\_category\_name\_translation.csv dataset.

* Product\_category\_name: name of the product category in original Portuguese language
* Product\_category\_name\_english: translation of the above

1. Geolocation

This contains general geolocation information with GPS coordinate.

* Geolocation\_zip\_code\_prefix: zip code of the described location
* Geolocation\_lat: latitutde
* Geolocation\_lng: longitude
* Geolocation\_city: city of the location
* Geolocation\_state: state of the location

1. Order\_reviews

This last data source contains information about order reviews made by customers on their order, including score, date, time, comment.

* review\_id: Unique identifier for the review.
* order\_id: Unique identifier for the order associated with the review.
* review\_score: Numerical score of the review (e.g., 1-5).
* review\_comment\_title: Title of the review comment.
* review\_comment\_message: Message content of the review.
* review\_creation\_date: Date when the review was created.
* review\_answer\_timestamp: Timestamp when the review was answered or

processed.

## 1.B. TOOL

To address this project, several tools will be used. The working environment will be on windows server. We will work with SQL Server Management Studio 20 (version 20.2) and Visual studio enterprise 2022 (version 17.11.4).

SQL Server Management Studio: this software allows us to manage one or several databases, scheduling routine tasks, etc..

Visual Studio Enterprise: This software will be the main environment in which we work this project out. A specific addon of VS Enterprise will be used, that is SSIS – SQL Server Integration Services. This combination will allow us to build a data pipeline and specific ETL job for the final datawarehouse.

## 1.C. METHOD

The data will travel through several database layers which will serve the purpose of extracting transforming and loading the data into the final modeled datawarehouse. In order to do that, we will need three main databases:

* a staging database : to fetch/retrieve the source raw data as they are
* an ods (operational data store): to transform, clean and process the raw data
* a datawarehouse database: the final database that will be used in the business, sales and other teams feeding on the data.
* An extra database “admin”: that will serve to welcome the data that had technical or functional issues.

Using the tool mentioned above, we will address the project from the source data in csv format.

The project timeline and process is described below:

# 2. STAGING DATABASE

In this part, we discuss how we built the staging database. The objective of the staging part is to welcome all the source data in raw form for further process. The source can be website, app data, APIs, IoT, data files, databases, cloud data, etc… Therefore, to retrieve what is interesting for the business, we need to have the data ready, thus staged in the staging database.

In this project, the source data are all from data files in CSV format. Each datasets represent a type of data, except for orders datasets which are triple. We will create one control flow for every dataset (and one slightly different for the orders dataset).

## 2.A. Importing orders dataset

To import orders dataset for the three years 2016, 2017 and 2018, we need to use a ‘Foreach Loop Container’ component in SSIS. This component allows to loop over data files and iterate over each of the set data files to retrieve its data and union them into a staging database “Orders” table.

A screenshot of a computer

Description automatically generated

In the container, we defined a data flow task in which we can set up a flat file connection.

A close-up of a computer screen

Description automatically generated

We first set the foreach container into “… file enumerator” . We then set the folder where the files are stored and we give an expression name, in our case, orders\_\*.csv so that the container can fetch all the files that begins with orders\_ and ends with .csv.

After setting this container, we create a ‘FilePath’ variable that we use in the flat file connection to indicate to the data flow task that the foreach loop will fetch the files.

A screenshot of a computer

Description automatically generated

Finally, we define in the flat file connection properties, a new “ConnectionString” property with the FilePath variable we created in the foreach loop container as an expression. That way, the data flow is set to fetch the designated files.

A screenshot of a computer

Description automatically generated

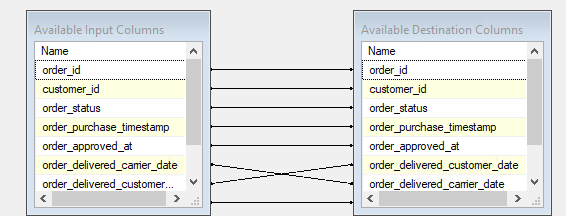
## 2.B. Other datasets

For other datasets, the data flow is easier, as there is only one of each dataset. Therefore, we need to fetch only one dataset for each data flow. A simple data flow with a flat file source component is enough. We set the flat file connection to fetch that specific file. We use a latin-1 (1252) encoding. After the data is retrieved, we need to load them into the staging databases.

A screenshot of a computer

Description automatically generated

To do that, we create an OLE DB Destination component where we will load all the data loaded from flat source file and we connect it to the right table in the staging database. We need to “map” the source file variables to the column in the database table.



Thus why, the database table must exist before completing this step. (The step to create the table in the sql database is described in the next part).

A screenshot of a computer screen

Description automatically generated

## 2.C. Creating SQL Staging database

To connect the OLE Destination SSIS component to the SQL Server database, we first need to create the databases in SQL Server Management Studio. We create 4 databases named:

* S24\_EXAM\_STA
* S24\_EXAM\_ODS
* S24\_EXAM\_DWH
* S24\_EXAM\_ADMIN

Taking the Orders dataset as an example, in the S24\_EXAM\_STA, we use the following query to create the orders table that will contain the data from the orders dataset source files:

A screenshot of a computer code

Description automatically generated

All the SQL Queries used to create the different tables can be found in appendix.

## 2.D. Specificity for OrderReviews dataset

This source dataset shows a singularity. In fact, in general for the project, we used a latin-1 encoding that can read most of latin language special characters. But for the specific orderreviews data, it cannot. Therefore, in the specific orderReviews data flow, we set up the flat file connection to encode in UTF-8. We also set all the variables for this dataset to WSTR,255 (instead of normal STR).

A screenshot of a computer

Description automatically generated

Therefore, we use a very specific SQL Query for this table in terms of ‘data type’.

A computer code with text

Description automatically generated

We defined all the variables as nvarchar(255) which is equivalent to WSTR,255 in SSIS, that can contain UTF-8 encoded string. In fact, SSIS specific flat file connection only support the same encoding for all variables. We cannot define different encodings for different variables of the same data source.

The right data types will be set up right in the ODS database before loading them in the data warehouse.

A screenshot of a computer

Description automatically generated

The first lines of the result using a data viewer functionality in SSIS:

A screenshot of a computer

Description automatically generated

The first lines of the result directly in the staging database in the sql server:

A screenshot of a computer

Description automatically generated

# 3. OPERATIONAL DATA STORE (ODS)

This part of the pipeline aims to operate all the transformation, cleaning and processing of the data as we want them to be. Its output will be directly ready to be loaded into the data warehouse.

In this part of the project, we still require to have a control flow for every dataset.

The control flow begins with creating a data flow task that will contain all our extracting, loading, data transformation task, just like the staging part.

## 3.A. Extracting staging data

The data flow task for our Orders ODS table looks like this:

A screenshot of a computer

Description automatically generated

It begins with data extraction and finishes with data loading.

To extract the staging data, we now don’t need to fetch from the flat source file anymore. Instead, we work directly with the staging database we created previously. To extract the data from there, we use a ‘OLE DB Source’ SSIS component. This component allows to retrieve data from a specific table in the database.

One only needs to specify the database and the table:

A white and grey rectangular object

Description automatically generated with medium confidence

Then, in the columns tab, we can review all the variables that can be imported from the database table. For our project, we decided to retrieve all columns except :

* Order\_delivered\_carrier\_date
* Order\_estimated\_delivery\_date

Those two variables, we won’t include them in the final model of the data warehouse as we decided them to be useless for our specific business needs. Therefore, they don’t need any transformation or extra steps.

A screenshot of a computer

Description automatically generated

After extracting the data, we converted the date from string to datetime format. We also resize all the variables to optimize the database. We use the minimal required size for every variables.

A screenshot of a computer

Description automatically generated

In the transformation process, we also considered the possible existence of technical issues with raw data that could pose a problem for our process. For that purpose, we earlier created a STA Admin database that will welcome all the data that had a technical or a functional issue. In this specific data flow for Orders data, we create a derived column for every date conversion task and union them and load them to the staging admin database.

In each derived column, we defined 4 variables that are needed to assess where the problem comes from to be able to solve it later on.

A screenshot of a computer

Description automatically generated

* Rejects\_Date: uses the expression GETDATE(). This will retrieve the date when this problem occurred.
* RejectPackageAndTask: retrieves system information about the package name and package task where the issue occurred
* RejectColumn: retrieves the specific column concerned with this problem
* RejectDescription: a short description of the technical problem.

The same derived column was created for all three conversion steps, then we merged them using a ‘UNION ALL’ SSIS component.

Note: the data tasks are set to ‘fail component’ by default, therefore, the data flow will stop if an error occurs. To redirect the errors to the sta admin database, we set up the conversion steps to redirect any errors to the derived columns first:

A screen shot of a computer error

Description automatically generated

After uniting all the incoming eventual errors, we load them into the STA Admin : technical rejects database table.



A white and grey rectangular object

Description automatically generated with medium confidence

Again, the database and the table need to exist before loading them. An SQL Query step is done before:

A screenshot of a computer code

Description automatically generated

And then a mapping step from the incoming data in the pipeline and the SQL destination database table:

A screenshot of a computer

Description automatically generated

The successful transformed data are then sent to the destination ODS database.

A white and grey rectangular object

Description automatically generated with medium confidence

As before, the database and the table must exist before this data task is created.

A computer code with text

Description automatically generated

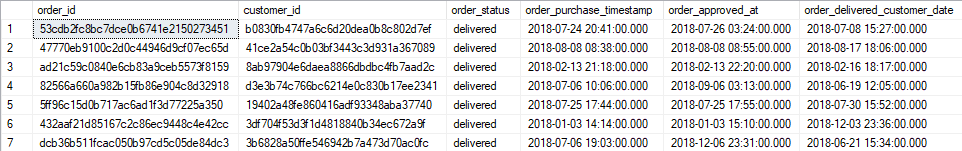
In this SQL query, we define the right data type for every variable. As we converted our dates into datetime format, we defined datetime data type in the sql query.

Now that the table exists, we can do the mapping between our transformed data and the destination variables in the sql table.

A screenshot of a computer

Description automatically generated

The ODS Orders table output looks like this:



# 4. DATE WAREHOUSE

## 4.A. MODELING

In this section, we are addressing the date warehousing part. We first modeled the data warehousing on paper taking into account what would be interesting business wise.

In the model, we adopted a snowflake architecture for our data warehouse.

A central fact table that includes the following data :

* Order\_id
* Oder\_item\_id

Those two previous variables act together as a composite key

* Customer\_key (surrogate)
* Seller\_key (surrogate)
* Product\_key (surrogate)
* Review\_key (surrogate)
* Order\_status
* Order\_purchase\_date (surrogate key from dimension date table)
* Order\_approved\_date (surrogate key from dimension date table)
* Order\_delivered\_customer\_date (surrogate key from dimension date table)
* Payment\_type
* Total\_invoiced (a manual calculation)

Around this fact table there are 6 dimension tables giving the context information of the observations in the fact table.

DimCustomer table:

* Customer\_key (surrogate key incrementing integer)
* Customer\_id (initial primary key of customer data)
* Customer\_unique\_id (anonymized unique id for customer)
* Geolocation\_key (which is a surrogate key for another dimension table)

DimSeller table:

* Seller\_key (surrogate)
* Seller\_id
* Geolocation\_key (which is a surrogate key for another dimension table)

DimProducts table:

* Product\_key (surrogate)
* Product\_id
* Product\_category\_name
* Product\_category\_name\_english
* Product\_photos\_qty
* Product\_weight\_g
* Product\_length\_cm
* Product\_height\_cm
* Product\_width\_cm
* Price
* Freight\_value

DimReviews:

* Review\_key (surrogate)
* DateKey (surrogate for dimension date table)
* Review\_id
* Review\_comment\_title
* Review\_comment\_message
* Review\_score

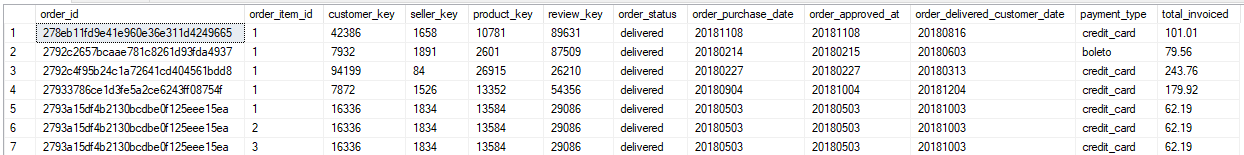
DimDate:

* DateKey
* Date
* Day
* DaySuffix
* Weekday
* WeekDayName
* WeekDayName\_Short
* WeekDayName\_FirstLetter
* DOWInMonth
* DayOfYear
* WeekOfMonth
* WeekOfYear
* Month
* MonthName
* MonthName\_Short
* MonthName\_FirstLetter
* Quarter
* QuarterName
* Year
* MMYYYY
* MonthYear
* IsWeekend

DimGeolocation

* Geolocation\_key
* Geolocation\_zip\_code\_prefix
* Geolocation\_lat
* Geolocation\_lng
* Geolocation\_city
* Geolocation\_state

The fact table first rows in the data warehouse looks like this:



## 4.B. DIMENSION TABLES

1. Geolocation table

We created a geolocation dimension table that includes all the geolocation details for a specific geolocation\_key. This table, as described in the snowflake architecture, is used to describe other dimension table (and not the fact table directly). For example, the customer dimension table includes the columns:



The geolocation\_key points to a specific row in the geolocation dimension table where details about the customer location can be found. This geolocation key is also used for seller table to indicate the location of the seller through a geolocation key.

The advantage of this, is to normalize and increase the granularity of our data warehouse, making it more readable and logical. The inconvenient is a reduction in the performance as more joins will have to be used when queries about customer or seller location are made.

Since, the raw data had several gps coordinates for same zip code prefix/city/state, we average the gps coordinate grouping by zip code/city/state and rounded them to 8 decimals. In fact, 8 decimals give already a precision of a location to the cm.

1. Date table

The date dimension table contains all the information of “time” that we might need, including month, week of the year, date in different format and a dateKey (surrogate key) that we reference in the other tables.

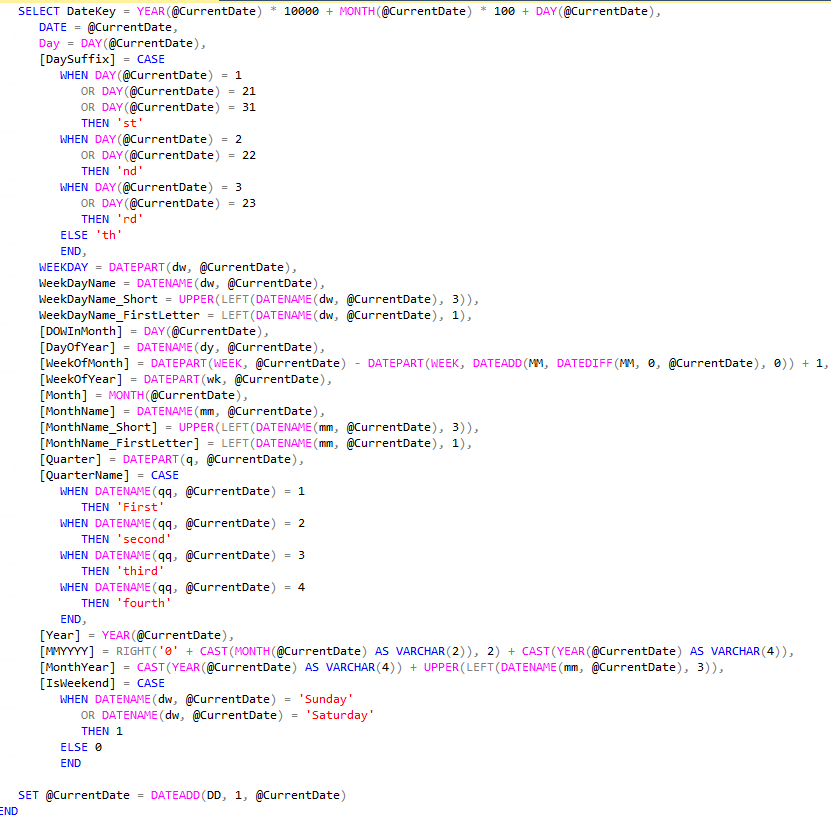
This table is also used to describe other dimension table such as DimReview table as well as to describe directly the fact table.

We do not download it, but we create it by using an SQL script.

We start by creating a table :



Then we fill the table by using the following query:



The first rows of the DimDate table look like this:

A screenshot of a computer

Description automatically generated

We can update this table changing the variable @EndDate at the beginning of the script to have the more recent dates.

1. Other dimensions table

All the context of a fact or an observation in the fact table can be found in the following dimension table:

* DimCustomer : table containing infos on customer
* DimSeller: seller informations
* DimProducts: product information (this contextualize the order\_items)
* DimReviews: review information of an order. The reviews are made for every order and not product. Therefore, when several items were purchased in a unique order, a duplicate of this will be found in the fact table, as it references the same review.

1. Creation of a dimension table

For our dimension tables, we chose the SCD1 strategy where we update and replace the old data when there are new modifications brought to a row.

We start by creating the DimTable with an SQL script where we create the technical keys.

A screenshot of a computer program

Description automatically generated

Then we load the data in this new table following this process:

A screenshot of a computer

Description automatically generated

In the first lookup, if the new ID does not exist in the old data, we add the full row to our table.

**[SCREENSHOT of the first lookup mapping]**

**@Mohamed: The packages with SCD couldn’t be retrieved from Visual studio (the virtual machine doesn’t start anymore).**

The idea was to add a column modification\_date. LKP between ODS customers and DWH DimCustomers. If customer ID matches but a change in one of the other column (geolocation especially), a new row will be added with modification\_date put to NULL. The older row will have modification\_date put to GETDATE().

**[SCREENSHOT of the second lookup mapping]**

We use the following SQL statement to update the existing rows that have been modified:

**@Mohamed: [SCREENSHOT of the SQL script]**

**@Vincent: [maybe also SCREENSHOT of the mapping of the parameters for the update?]**

The first lines of one of our dimension table looks like this:

**[SCREENSHOT of first rows of a dimension table from SQL server**]

## 4.C. Fact table

1. Total invoiced

Regarding, the variable total\_invoiced, this is a calculated column. In fact, in the original flat source files, we have access to the payment value by order, however an order can contain several items. Our granularity works at the order-item level, so we needed a way to obtain the price value per item. We can find from the order\_item source file, the information about the price and freight value of each item. Those two values were summed together to get the total value per item. The final payment\_value of the order is just the sum of the total\_invoiced for every items contained in the order.

1. Date keys

Regarding date, we decided to work with date and not datetime. Meaning that during the creation of the datawarehouse in SQL database, we changed the data type for date columns to date unlike we did with ODS where we kept the datetime. Indeed, depending on business needs, we decided to keep the datetime format in the ODS database, meaning it will be less costly to impute changed to retrieve the time when the business needs it.

Also for the dimension date table, we worked with the table seen in class that supports only date (day) at most (the SQL query can be found in appendix). To work with datekeys and keeping time, the dimension date table must be changed to support time slices.

1. Order\_id / Order\_item\_id

In the fact table, the primary key is a composite one, composed of the order\_id and the order\_item\_id which is just a repeated integer. The combination of both variables is necessary and sufficient to uniquely identify a row in the fact table.

A screenshot of a computer

Description automatically generated

For example the row #10 and #11 share the same order\_id but not the same order\_item\_id.

### Adding the technical keys and checking the relations

To create the fact table we added the technical keys created in the different dimension tables and checked the existence of the relations between the fact and the dimension tables.

To do so, we followed the following process:

A screenshot of a computer

Description automatically generated

**[SCREENSHOT of the whole process of one id lookup + management of functional rejects]**

The look up component allows us to check the relationship between the tables, but also to add the technical key to the fact table.

A screenshot of a computer

Description automatically generated

**[SCREENSHOT of the LookUp Mapping]**

When we do the lookup to check the relationship with the IDs, we send the missing relations to a functional reject table that we added to the admin database.

We created the functional reject table with the following script:

A computer code with text

Description automatically generated

**[SCREENSHOT of the SQL script creating the functional reject table]**

Then, just like we did in ODS, we use derived columns where we define variables that are needed to assess where the problem comes from to be able to solve it later on.

A screenshot of a computer

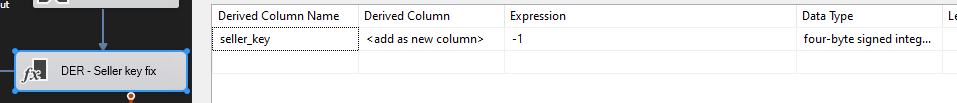
Description automatically generated

**[SCREENSHOT OF the creation of the derived columns]**

Before sending the errors to the functional reject table, we check if there isn’t any duplicates to avoid create redundant error messages.

**[SCREENSHOT of the aggregation component]**

Also, even if a relation is missing, we still want to keep the data in our fact table to not lose information, so we reincorporate the rows that were sent to the functional reject pipeline back in our table by giving it a key with a default value (-1).



**[SCREENSHOT of the creation of the derived column with -1]**

Once all the relations have been checked and the technical keys added, we can run an SQL script to create our fact table and then load our data into it.

A screenshot of a computer code

Description automatically generated

**[SCREENSHOT of SQL script creating the fact table]**

A screenshot of a computer

Description automatically generated

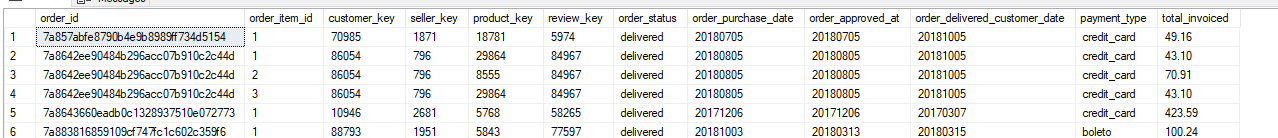
**[SCREENSHOT of the OLE DB Connection manager + choice of table]**

A screenshot of a computer

Description automatically generated

**[SCREENSHOT of mapping in OLE DB DESTINATION]**

This is what the first rows of our fact table look like:



**[SCREENSHOT of the first rows of the fact table in SQL Server]**

# 5. PIPELINE CONSTRUCTION

The step before automatization of this process is to build the pipeline, in other words, to construct the links between every single step of the whole process. This is done directly in SSIS.

Instead of using data flow task as it was done before, we now use sequence container (SEQC). SEQC can contain several packages and link them together in a specific order. That way, the whole pipeline process can be started all together without starting every single package at a time. Also, it the pipeline will proceeds in the specific order of the package that was defined in the sequence container.

This is important especially at the data warehouse step where the fact table data flow can only work after the dimension tables have been created and populated. Therefore, the dimension tables package must start first and foremost in the DWH step.

A screenshot of a computer

Description automatically generated

[SCREENSHOT with the updated version]

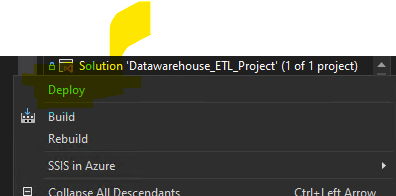
The sequence container have also more useful perks, such as working flow indication (failed, completed, success..) and more.

# 6. AUTOMATIZATION

## 6.A. deployment

The final step of the project is to create a routine job in SQL server that will launch the whole pipeline periodically at defined time periods.

To do that, we first deployed the project from Visual Studio solution explorer side page.

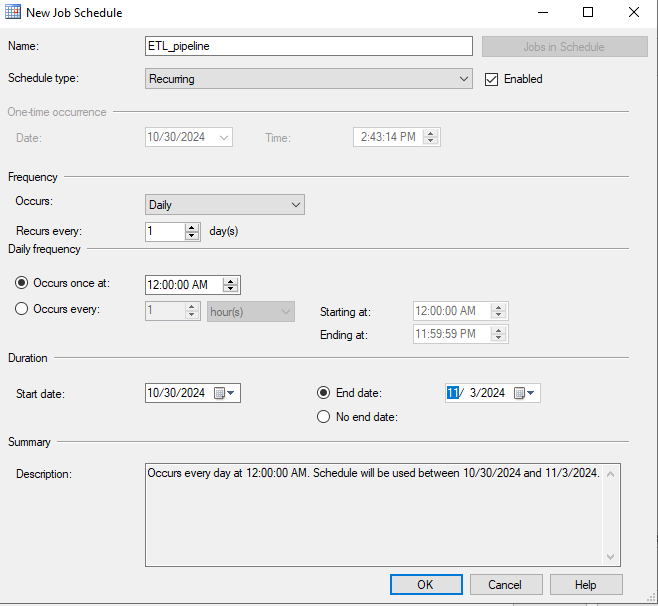


The deployment can be done with Azure, but in the scope of this project, we deployed the project directly to SQL server. We created a folder: S24\_EXAM. After deploying to SQL server, in SQL Server management studio we can see the folder S24\_EXAM in the integration services catalogs, SSISDB.

## 6.B scheduling

Once the project is deployed we can create job and schedule to implement the pipeline in a routine task.

We use for that, SQL server agent, we create a new job that we call S24\_EXAM\_ETL\_PIPELINE and we select the package called ETL , that contains all the sequence container of all packages in the pipeline as described in previous section. We then create a new schedule in the Schedules tab.



After this step is done, the project is completed and the pipeline will run every day at 12AM fetching the source file and proceeding to all the steps in the pipeline to filling the data warehouse.

# 7. USE CASES

Now that the data warehouse is ready to be used by business, we can do some analysis on the data. We will treat 3 use cases of the data warehouse.

## 7.A. Most and least successful product categories

We would like to know which product categories have the highest or lowest customer satisfaction.

Using the DWH, we can query using the following SQL:

**USE [S24\_EXAM\_DWH]**

**SELECT**

**dp.product\_category\_name\_english,**

**AVG(dr.review\_score) AS avg\_review\_score**

**FROM**

**FactOrders fo**

**JOIN**

**DimProducts dp ON fo.product\_key = dp.product\_key**

**JOIN**

**DimReviews dr ON fo.review\_key = dr.review\_key**

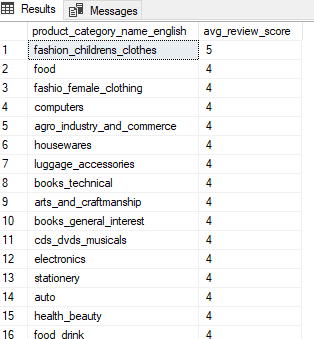
**GROUP BY**

**dp.product\_category\_name\_english**

**ORDER BY**

**avg\_review\_score DESC;**

This can provide a ranking of the different product categories according to their average customer review score.



In the use cases, we are more interested in two specific categories, the top one and the last one. In that case, we can modify the query as such:

**WITH RankedCategories AS (**

**SELECT**

**dp.product\_category\_name\_english,**

**AVG(dr.review\_score) AS avg\_review\_score,**

**ROW\_NUMBER() OVER (ORDER BY AVG(dr.review\_score) DESC) AS rn\_desc,**

**ROW\_NUMBER() OVER (ORDER BY AVG(dr.review\_score) ASC) AS rn\_asc**

**FROM**

**FactOrders fo**

**JOIN**

**DimProducts dp ON fo.product\_key = dp.product\_key**

**JOIN**

**DimReviews dr ON fo.review\_key = dr.review\_key**

**GROUP BY**

**dp.product\_category\_name\_english**

**)**

**SELECT**

**product\_category\_name\_english,**

**avg\_review\_score**

**FROM**

**RankedCategories**

**WHERE**

**rn\_desc = 1 OR rn\_asc = 1;**

This query uses the ‘VIEW’ functionality in TSQL. The result of the query is :

A screenshot of a computer

Description automatically generated

So the category that had much customer satisfaction is ‘fashion\_childrens\_clothes’ and the one that had the least customer satisfaction is ‘home\_comfort\_2’.

## 7.B. Impact of payment methods on completion times

Another use case:

How do different payment methods impact order completion times ?

We can query the DWH as such:

**USE [S24\_EXAM\_DWH]**

**SELECT**

**fo.Payment\_type,**

**AVG(DATEDIFF(day, ddp.Date, ddd.Date)) AS avg\_completion\_time\_days**

**FROM**

**FactOrders fo**

**JOIN**

**DimDate ddp ON fo.Order\_purchase\_date = ddp.DateKey**

**JOIN**

**DimDate ddd ON fo.Order\_delivered\_customer\_date = ddd.DateKey**

**WHERE**

**fo.Order\_status = 'Delivered'**

**GROUP BY**

**fo.Payment\_type**

**ORDER BY**

**avg\_completion\_time\_days;**

That way we can get the average completion time depending on the payment type. Here is the result:

A screenshot of a computer

Description automatically generated

## 7.C. factors influencing delivery delays

A more broad use case is to investigate the factors that influence delivery delays. For such analysis, we investigated separately suspect factors and try to see if there are big differences in delivery delays between sub groups of those factors.

For example, the weight of the product can be an interesting factor to look at considering delivery delays.

**SELECT**

**delays.product\_weight\_g,**

**AVG(delivery\_delay\_days) AS avg\_delay\_days,**

**ROUND(STDEV(delays.delivery\_delay\_days),2) AS stddev\_delay\_days**

**FROM**

**(SELECT**

**fo.Order\_id,**

**fo.Order\_item\_id,**

**DATEDIFF(day, da.Date, dd.Date) AS delivery\_delay\_days,**

**dp.product\_category\_name\_english,**

**dc.geolocation\_key AS customer\_location,**

**fo.payment\_type,**

**dp.product\_weight\_g**

**FROM**

**FactOrders fo**

**JOIN**

**DimDate da ON fo.order\_approved\_at = da.DateKey**

**JOIN**

**DimDate dd ON fo.order\_delivered\_customer\_date = dd.DateKey**

**JOIN**

**DimProducts dp ON fo.product\_key = dp.product\_key**

**JOIN**

**DimCustomers dc ON fo.customer\_key = dc.customer\_key**

**WHERE**

**DATEDIFF(day, da.Date, dd.Date) > 0) AS delays**

**GROUP BY**

**delays.product\_weight\_g**

**ORDER BY**

**avg\_delay\_days DESC;**

**GO**

The result is a ranking of average delivery delays in descending order of the product\_weight.

A screenshot of a data

Description automatically generated

As analyst, we could plot this table as scatter plot to check for any linearity, or correlation between package weight and delivery delay. For such result, we see that the weight of the package has in fact no influence on the delivery delay.

Customer location (no real meaning because too few observations like only one, for most location) , but maybe investigate the location that had more than 5 observations.

Payment type, answered by previous use case.

Product dimension (product width, or another Liter)

CCL:

Improvement (for more viability)

Date 🡪 include date time

Get more information about customer /sellers like first , last name, address, email, phone, etc.