DATAWAREHOUSING & ETL

Hénok Agbodjogbe
Michael Bruen
Théo Ourvoie
Yassine El Harrab
Loic Martins

Index

Introduction

The marketplace has a complex architecture because it must work with multiple sellers, multiple products, and thousands of customers. All these interactions generate a significant amount of data. To function well, the marketplace must leverage this data effectively to manage the platform and make the right decisions for future strategies.

This is why we need to create ETL processes and a data warehouse to have clean and structured data that can be used for analysis purposes.

To do it, we had a group composed of Hénok Agbodjogbe, Michael Bruen, Théo Ourvoie, Yassine El Harrab and Loic Martins.

At the organizational level, each member of the team worked on their own, and every week we had a meeting to take stock of the work done. Each member participated but Henok managed the technical part of the project.

Our project includes 4 stages:

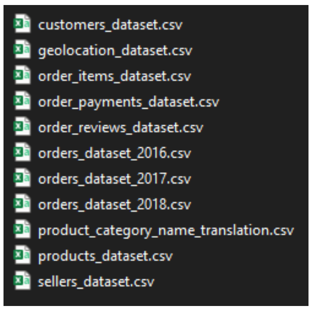
1. Data Understanding and Exploration
2. ETL Pipeline
3. Project Deployment
4. Analysis

I. Data Understanding and Exploration

1. Discovery

The exploration phase is one of the most important phases because it provides the foundations of our project.

We start with a large dataset including 11 Excel files:



Without opening the tables, we can note various points:

- We have a complete view of the data related to a marketplace: orders, products, sellers, customers, payment.
- We have 3 tables concerning the orders with different dates.
- We have 3 types of Orders table: the main table with the order, the reviews, the payments and the items.
- We have 2 tables related to the products: the main table and the categories table.

2. Exploration

To further understand the dataset, we need to have a better representation of each tables:

product_category_name_translation	products_dataset	sellers_dataset	orders_dataset
Mapping of product category names.	*Details about the products listed in the marketplace.*	*Sellers on the marketplace.*	*Details about customer orders.*
product_category_name product_category_name_english	product_id product_category_name product_name_lenght product_description_lenght	seller_id seller_zip_code_prefix seller_city seller_state	order_id customer_id order_status order_purchase_timestamp order_approved_at

	product_photos_qty product_weight_g product_length_cm product_height_cm product_width_cm		order_delivered_carrier_date order_delivered_customer_date order_estimated_delivery_date

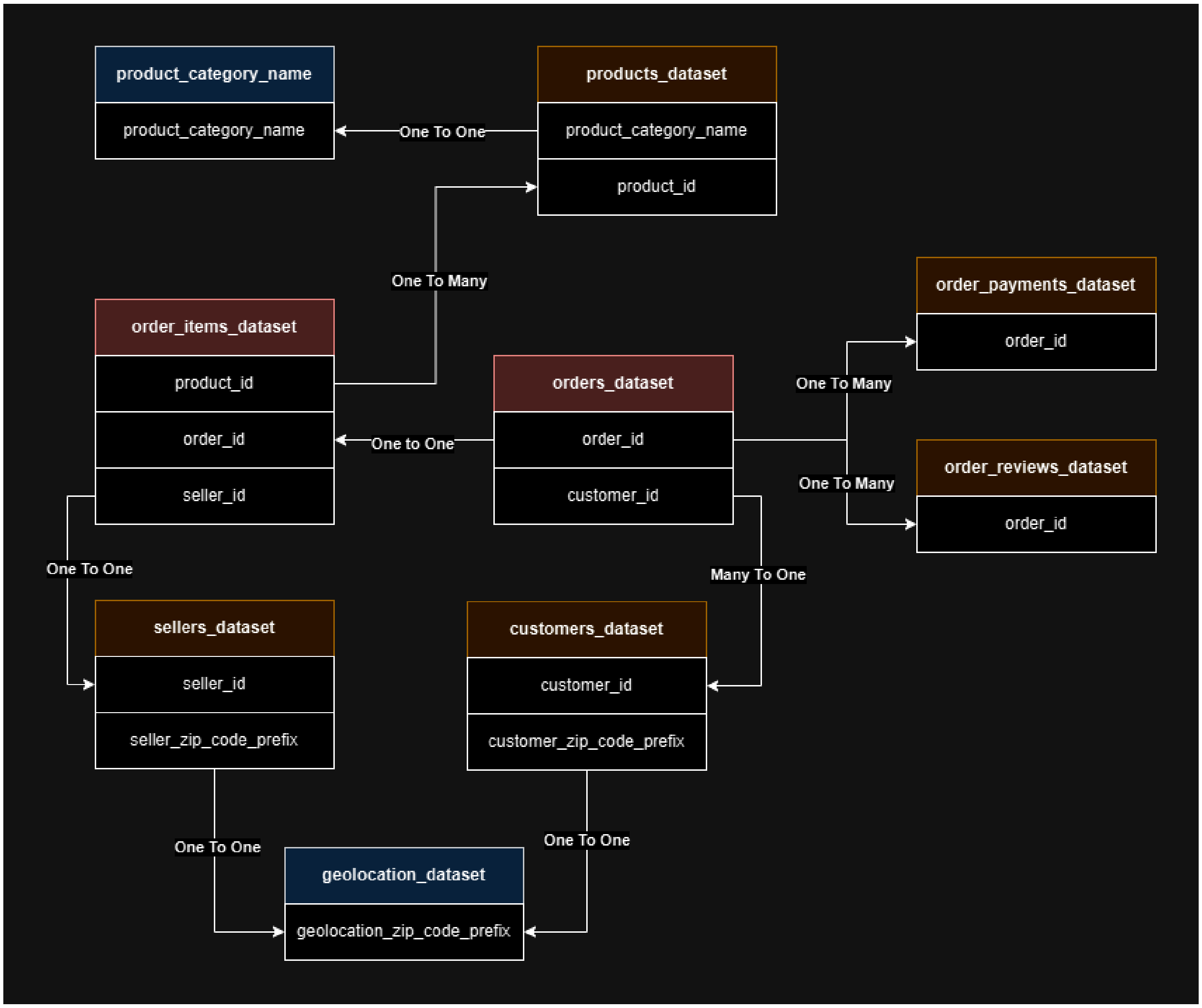
order_items_dataset	order_payments_dataset	order_reviews_dataset	geolocation_dataset
Information about the individual items.	*Details about payments made for orders.*	*Customer reviews for orders.*	*Geolocation information.*
order_id order_item_id product_id seller_id shipping_limit_date price freight_value	order_id payment_sequential payment_type payment_installments payment_value	review_id order_id review_score review_comment_title review_comment_message review_creation_date review_answer_timestamp	geolocation_zip_code_prefix geolocation_lat geolocation_lng geolocation_city geolocation_state

customers_datase
Customer information.
customer_id customer_unique_id customer_zip_code_prefix customer_city customer_state

Looking at these different tables, we can notice that:

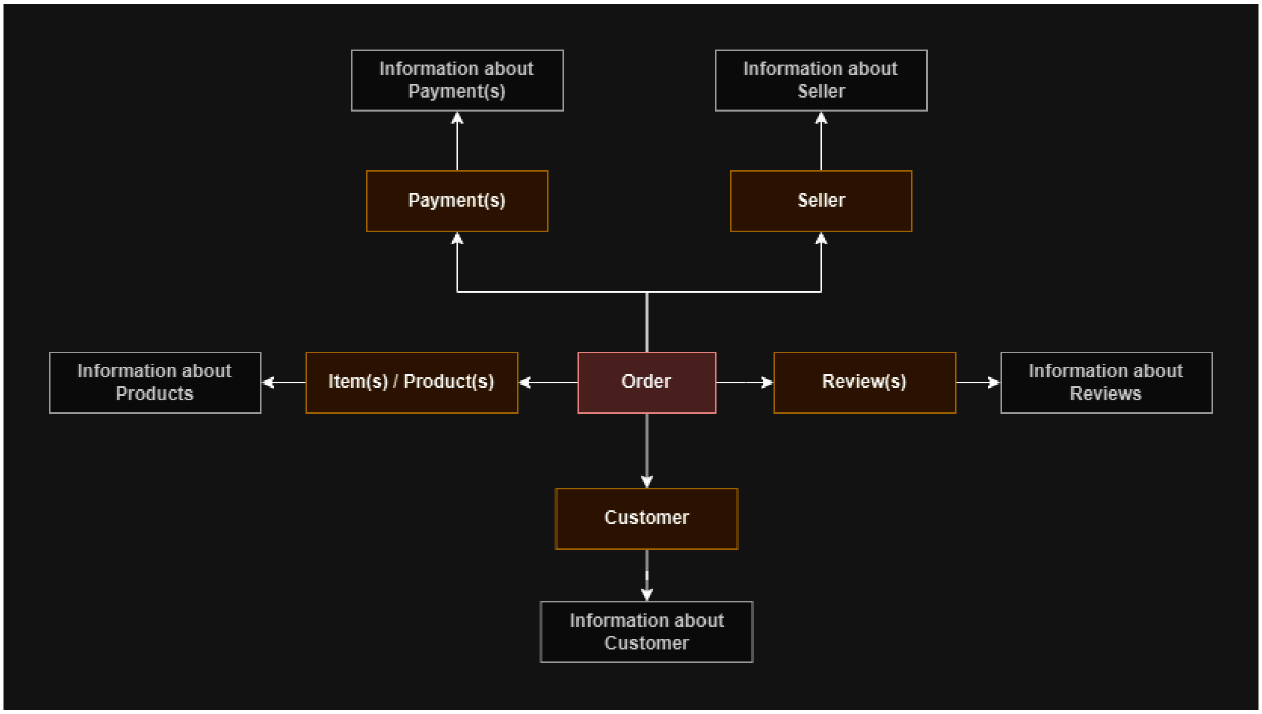
- The Order ID is our main field/variable because it is present in 4 tables.
- We can conclude that the dataset is centered around “the order”. When an order is placed, it creates an order_id.

We now have a better understanding of what the table contains but we need to understand their relationships and how they relate to each other:



Using this diagram, we can notice that:

- The Orders table is at the center of the whole dataset because this is where all the orders are recorded first.
- The Order Items table plays an important role because thank to it we can access the Sellers and Products tables.
- Using our first diagram we can create a map around the order:



- We can notice that for one order we have:
  - Item(s) / Product(s)
  - Customer
  - Review(s)
  - Payment (s)
  - Seller

II. ETL Pipeline

The pipeline will be done in three steps:

1. Data extraction will be done where the data will be extracted and placed in the staging area (STA). This will allow us to load the data as is, or with minimal changes.
2. The data transformation or Operational Data Store area (ODS) will allow us to clean and standardize the data.
   If the data does not pass the quality criteriums, they will be put in the “Technical_Rejects” table as technical rejects.
3. The data loading or Data WareHouse area (DWH) will organize the data in one fact table related to multiple dimensions tables. If records can’t be integrated in the schema, they will be put in the “Functional_Rejects” table as functional reject. Alternatively, some placeholder relations can be created.

There will be one STA and ODS package per file.

1. Data Extraction

Here, the role of data extraction or the staging database is to store all the data coming from the different sources. We want to accept all available data.

a) Orders Table

Here is an extract of the Orders_2016.csv file:

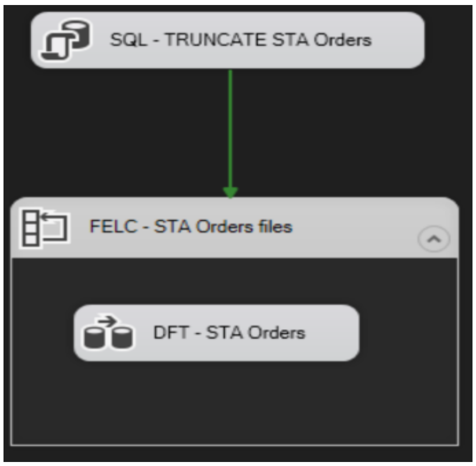
order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at
d3c8851a6651eeff2f73b0e011ac45d0	957f8e082185574de25992dc659ebbc0	processing	5/10/2016 22:44	6/10/2016 15:51
cbbb524a0e1646aa6cf7a3c0bbe517ad	dacb079d55ffb1d3955c5d923df3ebb7	delivered	5/10/2016 7:31	6/10/2016 2:46
ac2b7c522d811acba0aa270ed3e112e4	ef21aebbb093a6db29ccc6aa0b89c347	delivered	5/10/2016 15:08	6/10/2016 15:44
7033745709b7cf1bac7d2533663592de	7f0ca17bb33b230b47459437cf0682c7	delivered	4/10/2016 14:13	4/10/2016 14:46
5cd498954e2b37d71b315166809b4bd7	ff1a56726b7ea149c7423865609cc0c8	delivered	7/10/2016 0:54	8/10/2016 3:56

order_delivered_carrier_date	order_delivered_customer_date
10/10/2016 2:46	16/10/2016 14:36
10/10/2016 15:44	13/10/2016 15:44
8/10/2016 14:46	11/10/2016 14:46
25/10/2016 11:35	27/10/2016 17:32

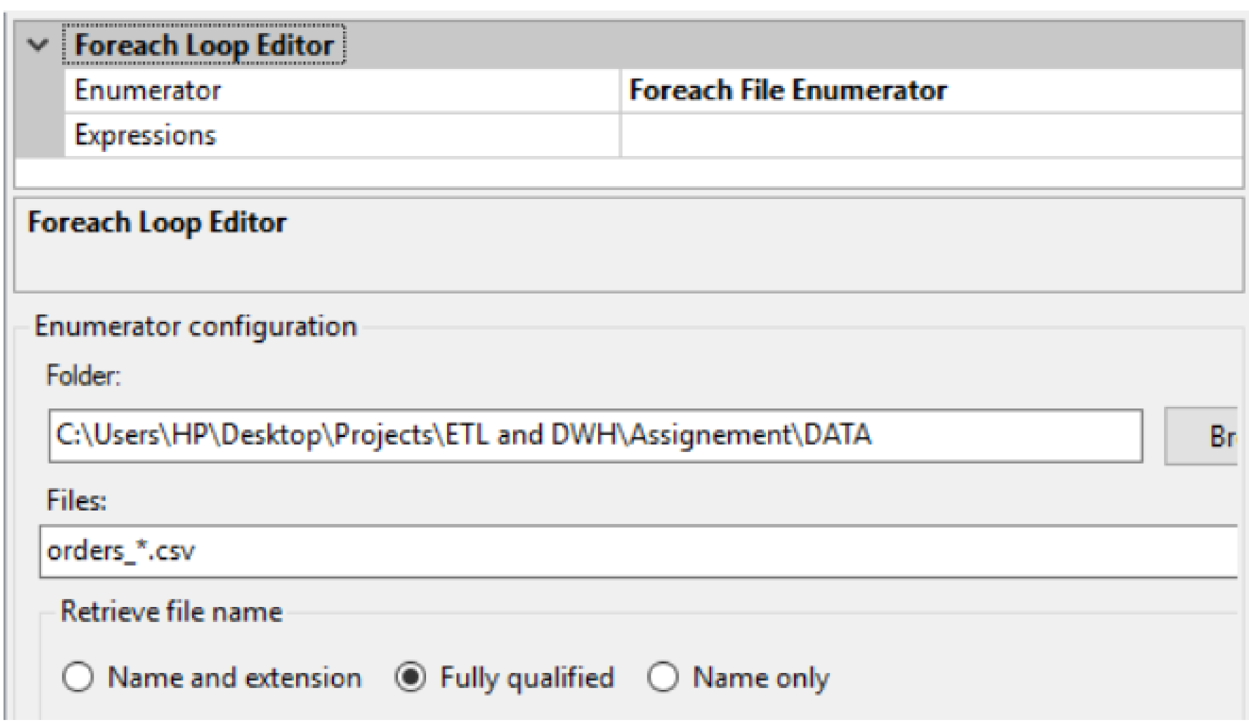
There are two other files with the same columns for orders 2017 and 2018.

To be able to use the orders data, we need to put it into one table. This means that we need to extract the data from each file.

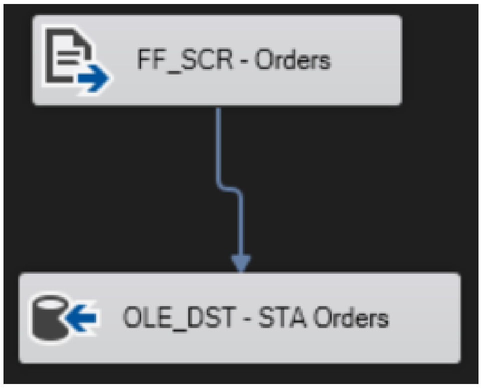
To go into all the orders files, we use a “Foreach Loop Container” where we put the extraction dataflow inside. The data flow in the container will be able to load one file at the time. Here, we also truncate the data from the previous runs.



The foreach loop is iterating over a variable “sales” that allow it to address each file separately. The variable is of the format “sales*” , with the star meaning it will access all files that start with sales.



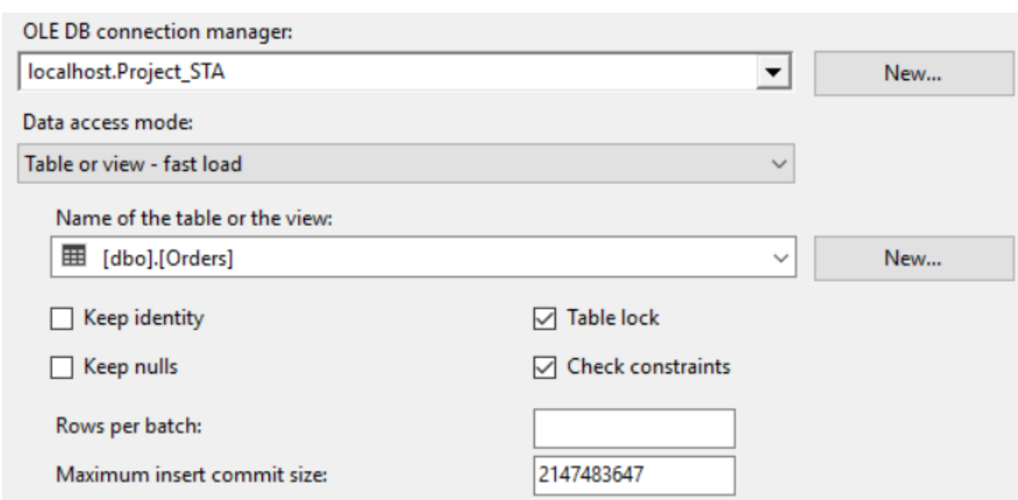
Next, the dataflow is defined as shown below:



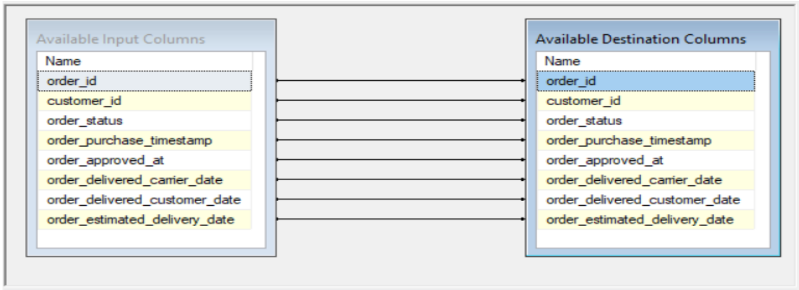
Once we have extracted the data, we can load it into our target table "Orders" in the STA database. We start this by creating a table with the following script:

```sql
USE PROJECT_STA
CREATE TABLE [dbo].[Orders] (
        [order_id] varchar(50),
    [customer_id] varchar(50),
    [order_status] varchar(50),
    [order_purchase_timestamp] varchar(50),
    [order_approved_at] varchar(50),
    [order_delivered_carrier_date] varchar(50),
    [order_delivered_customer_date] varchar(50),
    [order_estimated_delivery_date] varchar(50)
)
```

We then define the target destination.



And finally, we define the column mapping as follows:



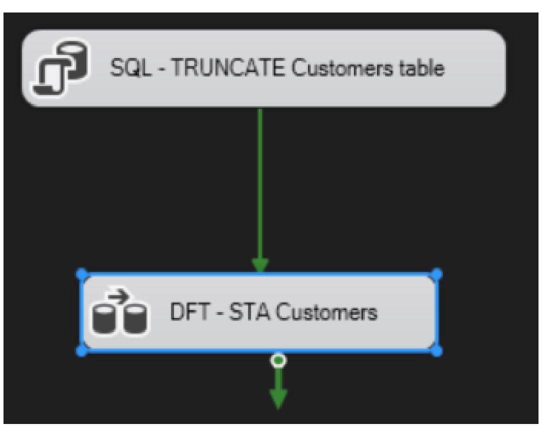
Input Column	Destination Column
order_id	order_id
customer_id	customer_id
order_status	order_status
order_purchase_timestamp	order_purchase_timestamp
order_approved_at	order_approved_at
order_delivered_carrier_date	order_delivered_carrier_date
order_delivered_customer_date	order_delivered_customer_date
order_estimated_delivery_date	order_estimated_delivery_date

Here are the first ten lines of the results table

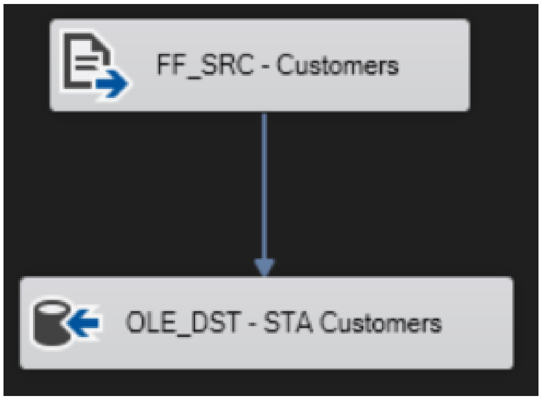
	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_
1	fc1d9429da446a147d438d41350a2932	8b13d2050940cb8d6a7203e69f279254	delivered	23/10/2017 16:29	24/10/2017 7:14	06/11/2017 22:11	18/11/2017 14:31	23/11/2017 0:00
2	8a6bf1083cc2efc062e037e474622de6	64f0ed97c13c1388088e93454db40488	delivered	20/03/2017 21:54	20/03/2017 21:54	25/03/2017 9:26	01/04/2017 8:47	10/04/2017 0:00
3	bcd37480e5f21eaf2b524c8e9724d099	fbc839b74c648a39b750799b58a13e0f	delivered	24/11/2017 22:55	25/11/2017 1:53	28/11/2017 21:26	07/12/2017 0:24	27/12/2017 0:00
4	b951cb6562f56cfb404cd77c81262f25	83116a97d5035c15ca0e75dc2b848bd9	delivered	26/12/2017 10:24	26/12/2017 10:36	27/12/2017 19:42	04/01/2018 21:27	24/01/2018 0:00
5	5bbe496e9e32884e4f0260f1b6bd6c3d	da6f87409fcc1f75e9bce0ff233ace2c	shipped	06/07/2017 14:42	06/07/2017 14:55	07/07/2017 12:04		28/07/2017 0:00
6	b825b53d8136d85eb4cbe60a1bd275f5	ec4f3f98da53dbaa4db16505b37f82c0	delivered	05/09/2017 14:48	06/09/2017 2:50	06/09/2017 21:22	12/09/2017 20:53	22/09/2017 0:00
7	b1e68c1ac52a1a5cb5051d07ae91ff06	cf34464d3b5d4a957dae9c0ef4a7d27e	delivered	18/03/2017 11:36	18/03/2017 11:36	21/03/2017 8:45	27/03/2017 13:22	26/04/2017 0:00
8	4516299f280a0f30720256ae0cd592d6	582ec2f9229bb3014866b31f73def34f	delivered	13/09/2017 18:12	13/09/2017 18:25	14/09/2017 18:53	18/10/2017 13:05	09/10/2017 0:00
9	7d63cb1d349e7bd0ee1edcc61ea71077	484711e3d14ac75fb874e949b3a28395	delivered	14/08/2017 15:30	14/08/2017 15:45	16/08/2017 17:04	24/08/2017 18:05	04/09/2017 0:00
10	b11073519dc9fe626df6281aa1965522	4f3ef3bae0d531a175d66f875cba84d1	delivered	16/11/2017 12:29	16/11/2017 13:10	21/11/2017 1:56	27/11/2017 16:22	05/12/2017 0:00

b) Customers Table

Next is the extraction of the data from “customers_dataset.csv”. For this table, we don’t need to add additional data. Here we just make sure to truncate the table “Customers” before running the package:



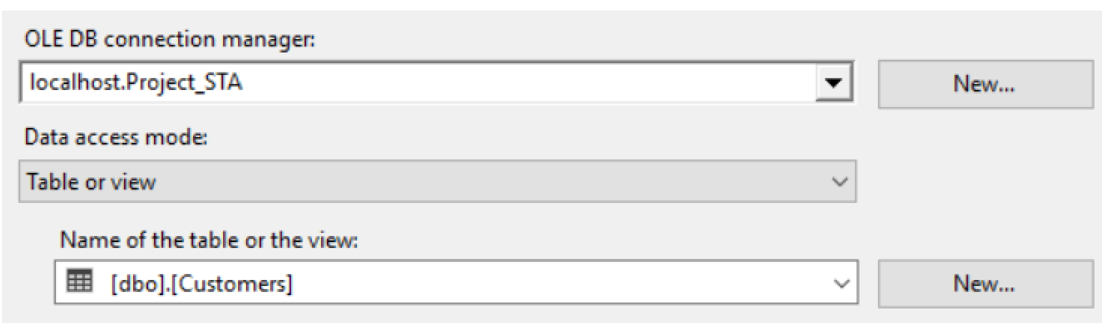
The dataflow is an import of a flat file:



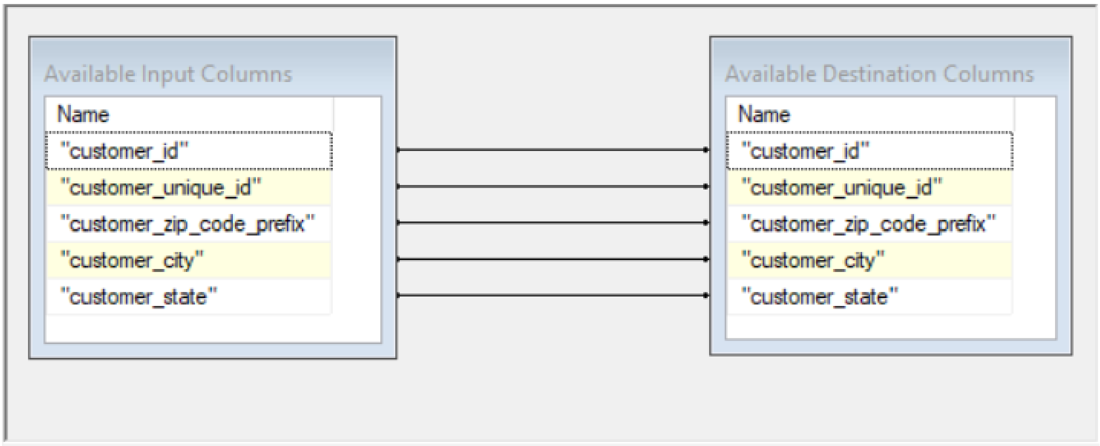
We create the destination table with the following command:

```
USE PROJECT_STA
CREATE TABLE [dbo].[Customers] (
    ["customer_id"] varchar(50),
    ["customer_unique_id"] varchar(50),
    ["customer_zip_code_prefix"] varchar(50),
    ["customer_city"] varchar(50),
    ["customer_state"] varchar(50)
)
```

The data is then loaded into the table “Customers”:



We define the column mapping as follows:



Input Column	Destination Column
"customer_id"	"customer_id"
"customer_unique_id"	"customer_unique_id"
"customer_zip_code_prefix"	"customer_zip_code_prefix"
"customer_city"	"customer_city"
"customer_state"	"customer_state"

Here is the first ten lines of the results:

	"customer_id"	"customer_unique_id"	"customer_zip_code_prefix"	"customer_city"	"customer_state"
1	a6fba43413c5d37fd7dc3173432f6ab2	212b273eed006f96498ccd8e91427e42	86840	faxinal	PR
2	c1bdc290c9ae68e5f0825b2dfed4d422	ae5df8e33ea496f14db277c8b8e77281	13098	campinas	SP
3	0a3e4f7efaf1befc4a84e1adcd9b70f5	273d136f7121c493a5c994c4838fa350	59196	pedro velho	RN
4	9c64abffb2de9888aa146da8c65a547f	18e362d58b54666925aacfd86ff84213	04944	sao paulo	SP
5	04edcf9c41ea594087f93171236b28e2	4e17943c8edf2f66d70f0e97b81aaf17	28085	campos dos goytacazes	RJ
6	1f954c777b0dbd306b017a3e4cd26475	d35862da9bdb38fad3994de72a77a3c2	38025	uberaba	MG
7	b769ee1c67d0e4f3c5b0abe1a1b958d4	3726ac7519981c0c5a86b98ca38297a4	28055	campos dos goytacazes	RJ
8	03bbe0ce5c28e05f22917607db798818	8f3dca4306d5a89e4ae2c65c110603a2	72465	brasilia	DF
9	cf833bbe73b3d8ddeb9889d4192b2496	03f923dbcd3b3c763b22b24db6072d27	13400	piracicaba	SP
10	9716c4a296c261a120d27d058e32ab38	0647d94e80c8033ef27686bdb447e105	36015	juiz de fora	MG

The rest of the datasets are extracted the same way as the customers one above.

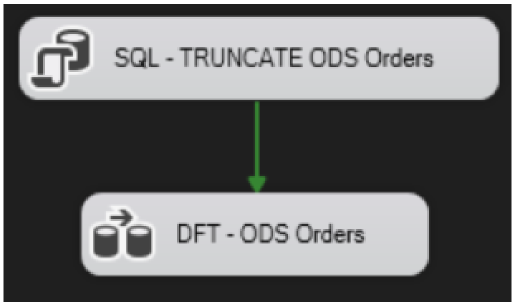
2. Data Transformation

The second step of the pipeline is to load usable data and transform it before putting it into the Operational Data Store (ODS). This means we need to transform the data into a usable format. We also need to clean and standardize the data. All the data that does not respect the “quality standards “will be rejected as a technical reject.

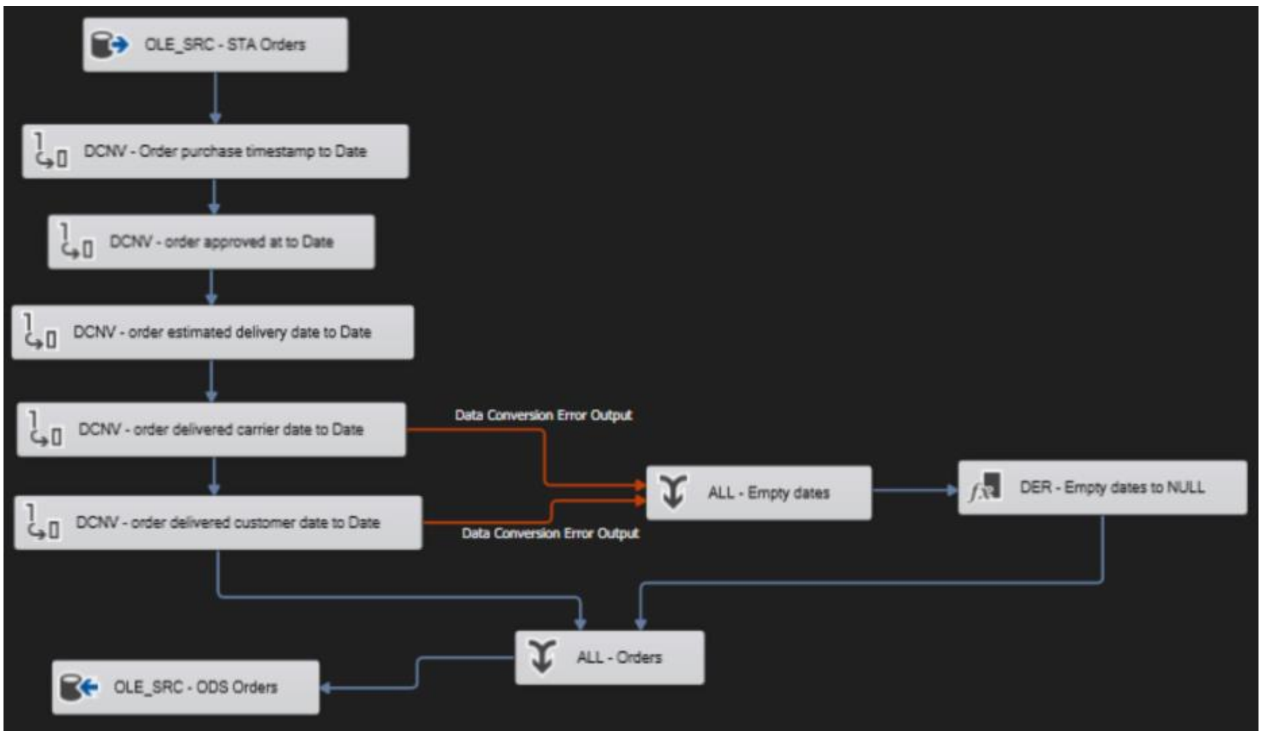
The quality standards are based on the “correctness” of the data. The output data must be consistent in data types and in values. We also need to ensure that the data can be used in queries, so we might need to reorganize and enrich the data.

a) Orders Table

Like before, we truncate the data from the previous runs.

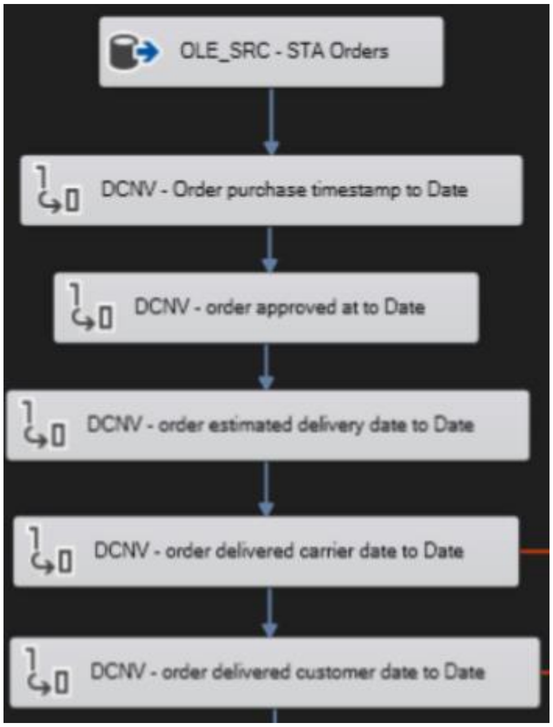


The data flow is defined below:



Below, the segments of this dataflow are explained:

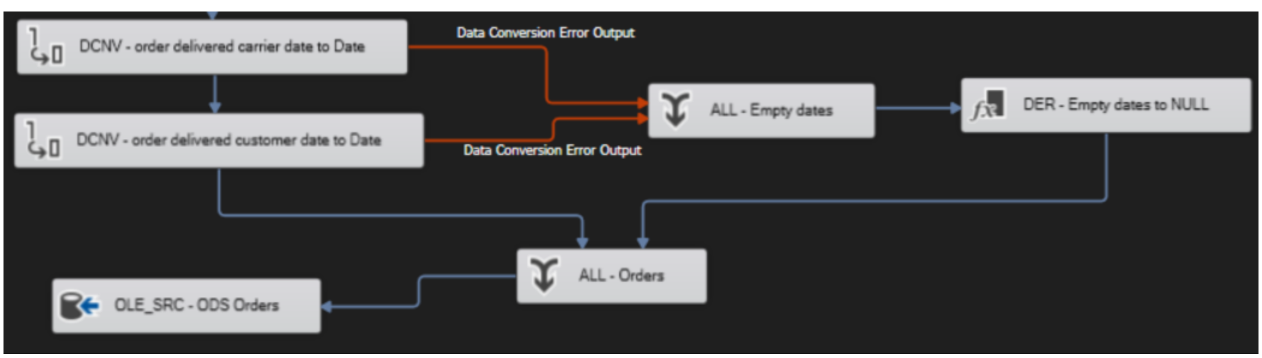
In the first step as shown below, we extract the data from STA orders. After this the columns that have date data are changed from datatype string to datatype date. Date is chosen instead of datetime because in the DWH later on we need to map this data with a column with the type date.



Within the tasks, the datatype of the column is being changed to date as shown below:

Input Column	Output Alias	Data Type
order_purchase_timestamp	order_purchase_timestamp_Date	date [DT_DATE]

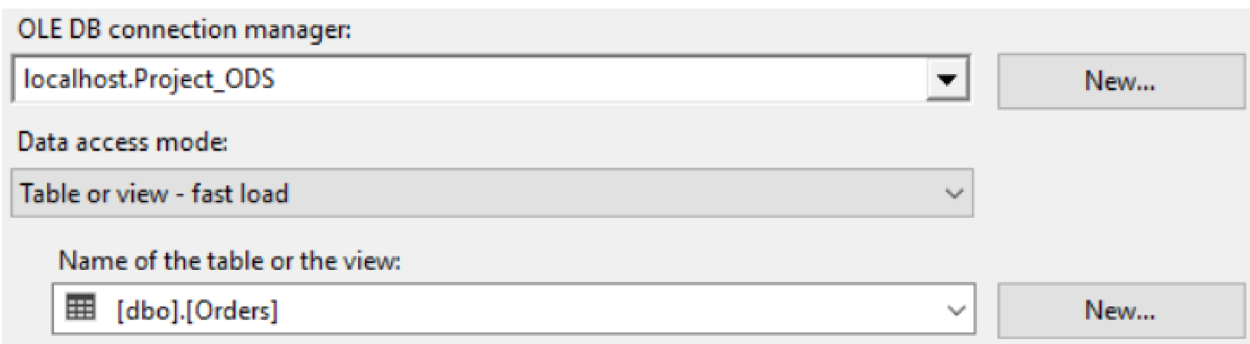
Order delivered carrier date and order delivered customer date have several rows that do not contain any data at all. To handle this, the rows that produce a data conversion error have their values changed to NULL before being fed back into the dataset. This is shown in the screenshots below.

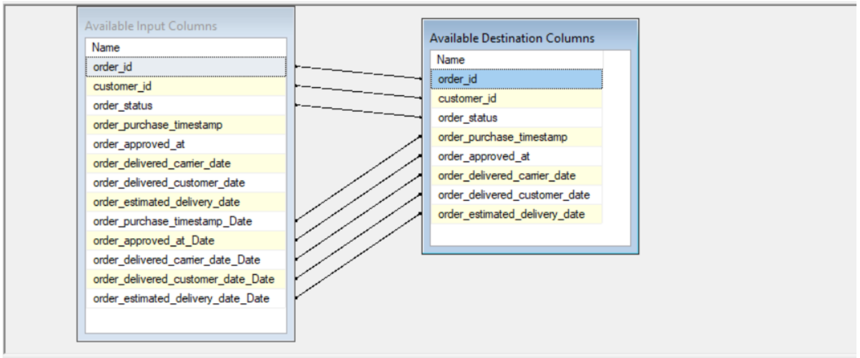


Derived Column Name	Derived Column	Expression	Data Type
order_delivered_carrier_date_NULL	<add as new column>	NULL(DT_DATE)	date [DT_DATE]
order_delivered_customer_date_NULL	<add as new column>	NULL(DT_DATE)	date [DT_DATE]

After all this is done the ODS Orders table is created and the data is inserted into the database as shown below.

```
USE Project_ODS
CREATE TABLE [dbo].[Orders] (
    [order_id] varchar(50),
    [customer_id] varchar(50),
    [order_status] varchar(50),
        [order_purchase_timestamp] Date,
    [order_approved_at] Date,
    [order_delivered_carrier_date] Date,
    [order_delivered_customer_date] Date,
    [order_estimated_delivery_date] Date
)
```





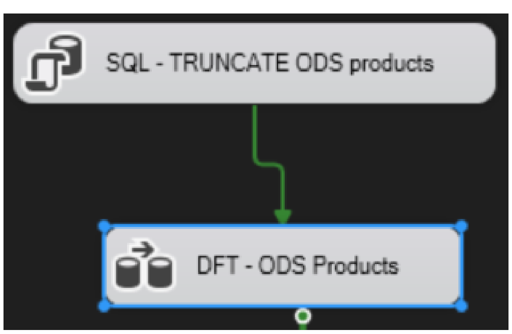
Input Column	Destination Column
order_id	order_id
customer_id	customer_id
order_status	order_status
order_purchase_timestamp_Date	order_purchase_timestamp
order_approved_at_Date	order_approved_at
order_delivered_carrier_date_Date	order_delivered_carrier_date
order_delivered_customer_date_Date	order_delivered_customer_date
order_estimated_delivery_date_Date	order_estimated_delivery_date

Here is the first ten lines of the results:

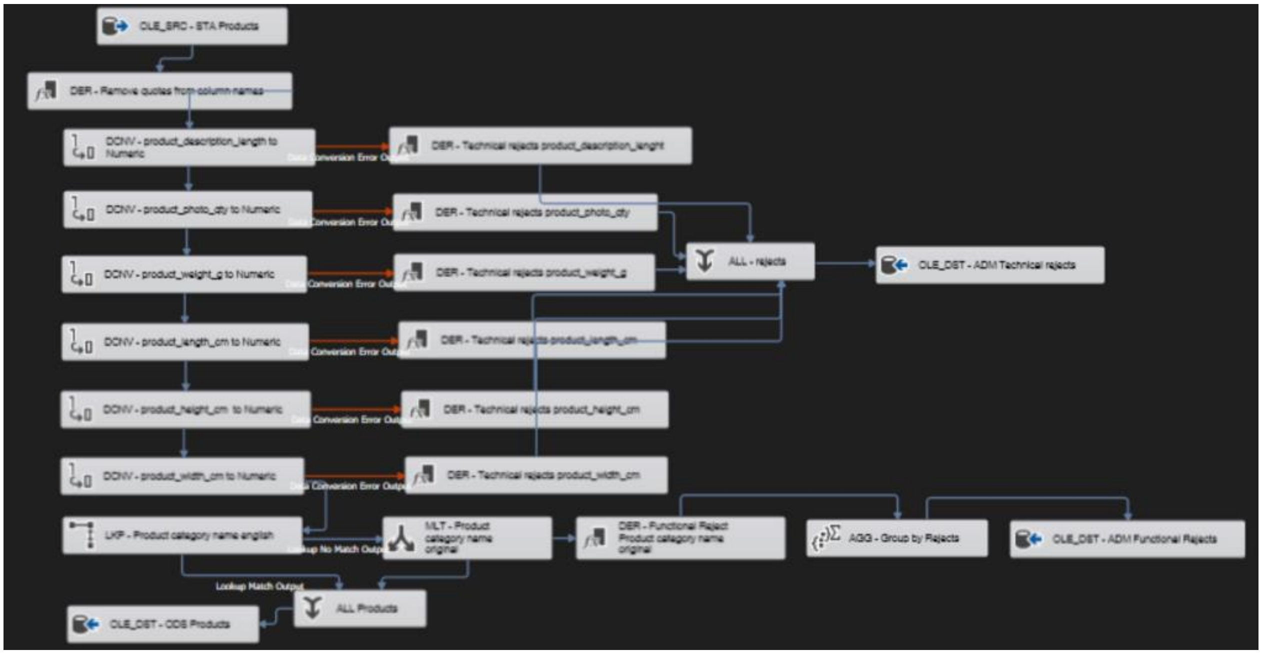
	order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier_date	order_delivered_customer_date	order_estimated_delivery_
1	fc1d9429da446a147d438d41350a2932	8b13d2050940cb8d6a7203e69f279254	delivered	2017-10-23	2017-10-24	2017-06-11	2017-11-18	2017-11-23
2	8a6bf1083cc2efc062e037e474622de6	64f0ed97c13c1388088e93454db40488	delivered	2017-03-20	2017-03-20	2017-03-25	2017-01-04	2017-10-04
3	bcd37480e5f21eaf2b524c8e9724d099	fbc839b74c648a39b750799b58a13e0f	delivered	2017-11-24	2017-11-25	2017-11-28	2017-07-12	2017-12-27
4	b951cb6562f56cfb404cd77c81262f25	83116a97d5035c15ca0e75dc2b848bd9	delivered	2017-12-26	2017-12-26	2017-12-27	2018-04-01	2018-01-24
5	b825b53d8136d85eb4cbe60a1bd275f5	ec4f3f98da53dbaa4db16505b37f82c0	delivered	2017-05-09	2017-06-09	2017-06-09	2017-12-09	2017-09-22
6	b1e68c1ac52a1a5cb5051d07ae91ff06	cf34464d3b5d4a957dae9c0ef4a7d27e	delivered	2017-03-18	2017-03-18	2017-03-21	2017-03-27	2017-04-26
7	4516299f280a0f30720256ae0cd592d6	582ec2f9229bb3014866b31f73def34f	delivered	2017-09-13	2017-09-13	2017-09-14	2017-10-18	2017-09-10
8	7d63cb1d349e7bd0ee1edcc61ea71077	484711e3d14ac75fb874e949b3a28395	delivered	2017-08-14	2017-08-14	2017-08-16	2017-08-24	2017-04-09
9	b11073519dc9fe626df6281aa1965522	4f3ef3bae0d531a175d66f875cba84d1	delivered	2017-11-16	2017-11-16	2017-11-21	2017-11-27	2017-05-12
10	c7b44077f6d014b67c3164e103e3bf02	19187bc9fab7457748d4dc42a9e1c6fe	delivered	2017-11-21	2017-11-21	2017-11-23	2017-11-30	2017-01-12

b) Products Table

Like before, we truncate the data from the previous runs.



The data flow is defined below:



Below, the segments of this dataflow are explained:

In the first step as shown below, we extract the data from STA products and remove the quotes from the column names.

Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
product_id	Replace 'product_id'	product_id	string [DT_STR]	50			1252 (ANSI - Latin I)
product_category_name	Replace 'product_category_name'	product_category_name	string [DT_STR]	50			1252 (ANSI - Latin I)
product_description_length	Replace 'product_description_length'	product_description_length	string [DT_STR]	50			1252 (ANSI - Latin I)
product_photos_qty	Replace 'product_photos_qty'	product_photos_qty	string [DT_STR]	50			1252 (ANSI - Latin I)
product_weight_g	Replace 'product_weight_g'	product_weight_g	string [DT_STR]	50			1252 (ANSI - Latin I)
product_length_cm	Replace 'product_length_cm'	product_length_cm	string [DT_STR]	50			1252 (ANSI - Latin I)
product_height_cm	Replace 'product_height_cm'	product_height_cm	string [DT_STR]	50			1252 (ANSI - Latin I)
product_width_cm	Replace 'product_width_cm'	product_width_cm	string [DT_STR]	50			1252 (ANSI - Latin I)

After this there are several columns that contain numerical data. These are changed as shown below.

Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
product_description_length	product_description_length_Numeric	numeric [DT_NUMERIC]		18	0	

If any of the data is unable to be changed to numeric, the error is tracked and put in the "technical_rejects" table.

Derived Column Name	Derived Column	Expression	Data Type	Length
Rejectdate	<add as new column>	GETDATE()	database timestamp [DT_DBTIMESTAMP]	
RejectPackageAndTask	<add as new column>	(DT_WSTR,100)@[System::PackageName] + " AND " + (DT_WSTR,100)@[System::TaskName]	Unicode string [DT_WSTR]	205
RejectColumn	<add as new column>	"product_description_length"	Unicode string [DT_WSTR]	26
RejectDescription	<add as new column>	"The value " + (DT_WSTR,100)product_description_length + " Is not a valid integer"	Unicode string [DT_WSTR]	133

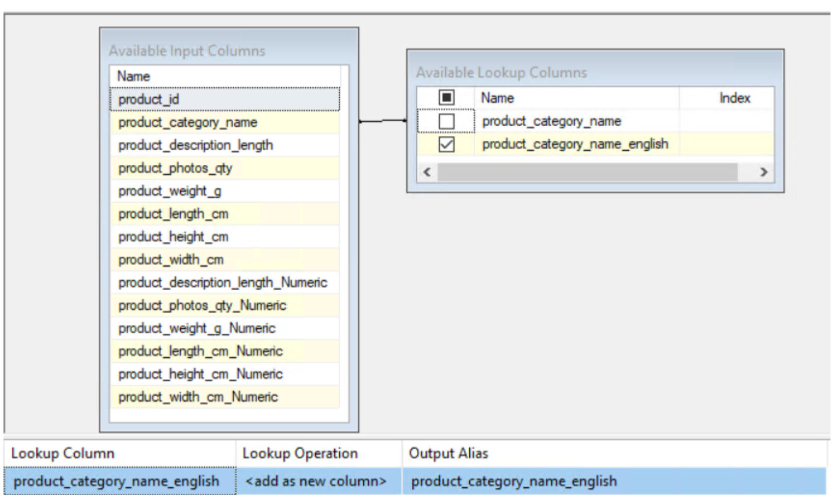
As seen above the data inserted in the technical rejects are:

4. The date of the error.
5. The column making the error.
6. An error message.
7. The package and the task causing the error.

The technical rejects table is created in the database using the following script:

```
USE [Project_ADM]
CREATE TABLE [dbo].[TechnicalRejects] (
    [Rejectdate] datetime,
    [RejectPackageAndTask] nvarchar(205),
    [RejectColumn] nvarchar(29),
    [RejectDescription] nvarchar(133)
)
```

This datatype conversion is performed on 6 columns. Afterwards a lookup is performed on the product category name column using the product category name translation dataset that was loaded into STA. this will change the Spanish names to English ones.



The product names that do not have a matching translation name get tracked and sent to the functional rejects table through the steps mapped out earlier and shown in more detail below.

Derived Column Name	Derived Column	Expression	Data Type	Length
RejectDate	<add as new column>	GETDATE()	database timestamp [DT_DBTIMESTAMP]	
RejectPackageAndTask	<add as new column>	(DT_WSTR,100)@[System::PackageName] + " AND " + (DT_WSTR,100)@[System::TaskName]	Unicode string [DT_WSTR]	205
RejectColumn	<add as new column>	"Product_category_name"	Unicode string [DT_WSTR]	21
RejectDescription	<add as new column>	(DT_WSTR,100)product_category_name + " has not translation in english"	Unicode string [DT_WSTR]	131

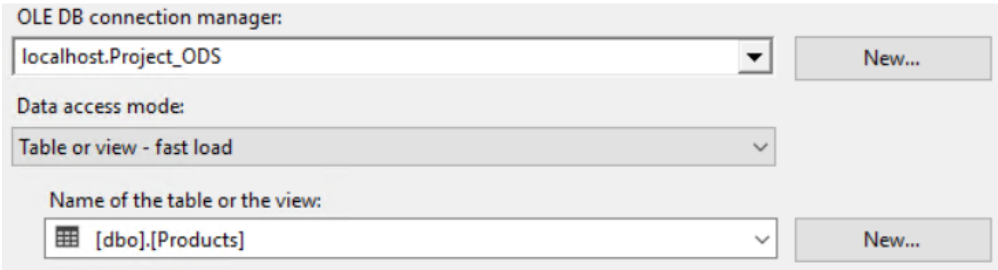
Available Input Columns

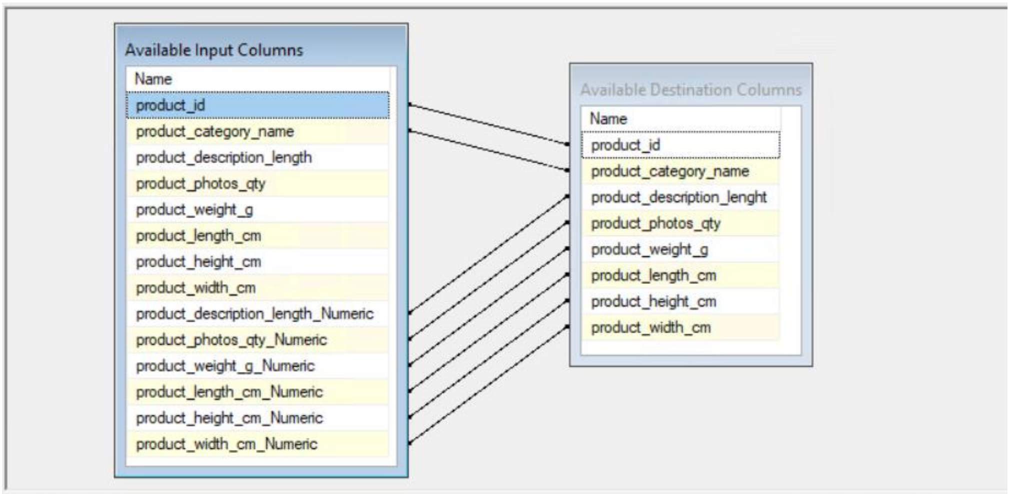
	Name
[ ]	product_photos_qty
[ ]	product_weight_g
[ ]	product_length_cm
[ ]	product_height_cm
[ ]	product_width_cm
[ ]	product_description_leng...
[ ]	product_photos_qty_Nu...
[ ]	product_weight_g_Numeric
[ ]	product_length_cm_Num...
[ ]	product_height_cm_Num...
[ ]	product_width_cm_Nume...
[x]	RejectDate
[x]	RejectPackageAndTask
[x]	RejectColumn
[x]	RejectDescription

Input Column	Output Alias	Operation
(*)	Count all	Count all
RejectDate	RejectDate	Group by
RejectPackageAndTask	RejectPackageAndTask	Group by
RejectColumn	RejectColumn	Group by
RejectDescription	RejectDescription	Group by

```sql
USE [Project_ADM]
CREATE TABLE [dbo].[FunctionalRejects] (
    [RejectDate] datetime,
    [RejectPackageAndTask] nvarchar(205),
    [RejectColumn] nvarchar(29),
    [RejectDescription] nvarchar(131),
        [Count] int
)
```

These functional rejects are also fed back into the dataset with the use of a multi. This is done for potential analysis later on translation and sales. After all this is done the data is loaded into an ODS products table as shown in the steps below.



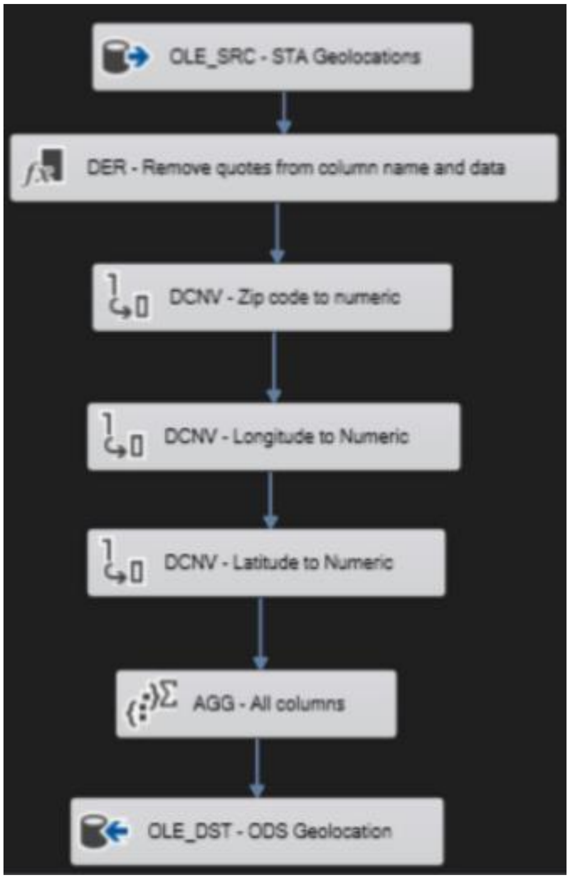


Input Column	Destination Column
product_id	product_id
product_category_name	product_category_name
product_description_length_Numeric	product_description_lenght
product_photos_qty_Numeric	product_photos_qty
product_weight_g_Numeric	product_weight_g
product_length_cm_Numeric	product_length_cm
product_height_cm_Numeric	product_height_cm
product_width_cm_Numeric	product_width_cm

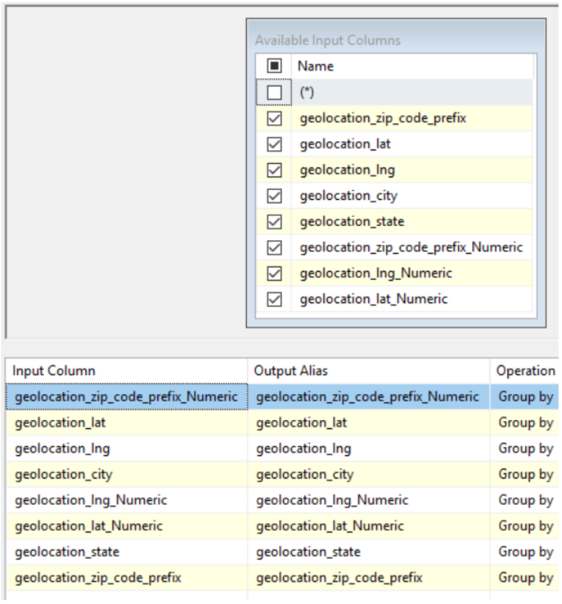
	product_id	product_category_name	product_description_lenght	product_photos_qty	product_weight_g	product_length_cm	product_height_cm	product_width_cm
1	b5694f734bee2683d25bb188a7f45b40	stationery	684	2	100	16	2	11
2	41b24e4f49504cc023666ed7cde3cf16	sports_leisure	628	7	1800	25	30	40
3	e987fc6af8290fcf1db0510d52b65795	health_beauty	99	2	250	20	10	15
4	6e2d41323c4b6d240b530b9e68e83de1	watches_gifts	626	1	400	23	15	19
5	e76572c2ae4cede1e3b2ead512423efe	bed_bath_table	352	2	450	45	11	11
6	e97ee7673459375fac14fe9a01a64e6c	toys	515	1	289	17	15	17
7	d212ad4ac52091c8d50442a89fbbc7af	bed_bath_table	160	1	350	16	16	16
8	99f83ce7f14247f186cb52d73bdd6e79	construction_tools_construction	453	1	100	16	7	12
9	9539cddd99ab3f8ca77b6cad0a0899ea	sports_leisure	1067	4	433	23	12	17
10	9fae04deab3bb1c353c27c50cbf3b42b	toys	927	4	450	21	36	11

c) Geolocation Table

This package starts with a truncate the same as the previous packages. The DFT is shown below:



In the first step, we extract the data from STA Geolocation and remove the quotes from the column names the same as previous packages. After this there are several columns that contain numerical data. These are changed the same way as the previous package without errors. Once this is completed, a group by is used to aggregate all the data and remove duplicates.

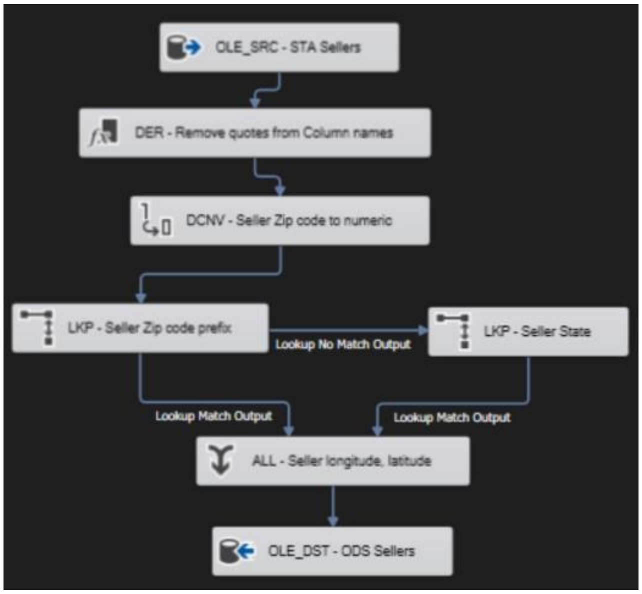


After all is completed, the data is loaded into an ODS geolocation table in the same steps as shown previously. The script for the table is shown below.

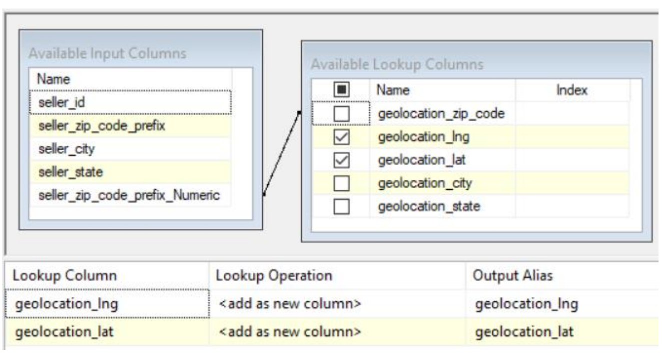
```
USE [Project_ODS]
CREATE TABLE dbo.Geolocation (
    [geolocation_zip_code] numeric(18,0),
    [geolocation_lng] decimal(18,10),
    [geolocation_lat] decimal(18,10),
        [geolocation_city] varchar(50),
        [geolocation_state] varchar(50),
)
```

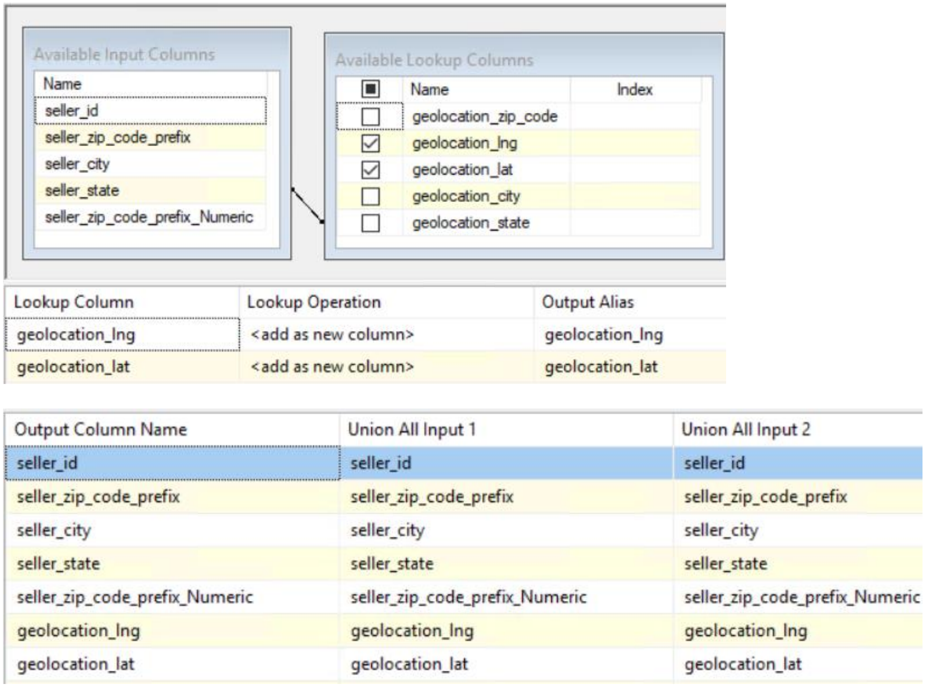
d) Sellers Table

This package starts with a truncate the same as the previous packages. The DFT is shown below:



In the first step, we extract the data from STA Geolocation and remove the quotes from the column names the same as previous packages. After this the zip code column is changed to numerical data using a data conversion as shown in previous packages. After this a lookup is performed on the seller zip code column using the geolocation dataset that was loaded into ODS. The data that had no matching geolocation data for zip code are then matched with state to get a geolocation.





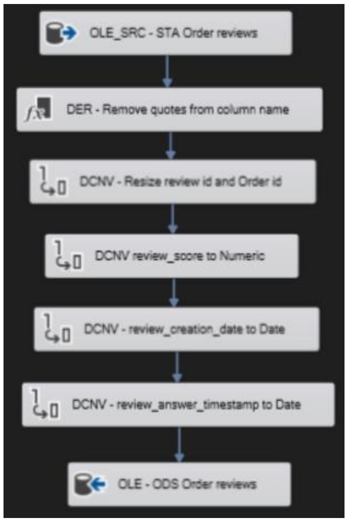
At the end of this process, all the sellers will have geolocation data attached to them. The reason for this is to have the potential to evaluate the distance between sellers and customers in analysis. This also means you can create maps with your data during analysis. After all this the data is loaded into a table in ODS the same way as the previous packages.

e) Customers Table

The customers table follows the same steps as the sellers table above with its own respective data. Zip code is changed to numeric, geolocation data is added for the customers using a lookup from the geolocation dataset, and the data is loaded into a table in ODS.

f) Order Reviews Table

This package starts with a truncate the same as the previous packages. The DFT is shown below:



In the first step, we extract the data from STA Customer_reviews and remove the quotes from the column names the same as previous packages. After this, data conversion is done on various columns in the same steps as done in previous packages. Review ID and order ID have their length changed to 50, review score is changed to numeric, and review creation date and review answer timestamp is changed to date. After all this is done the data is loaded into a table in ODS the same way as previous packages.

Order payments and order items follow the same steps as order reviews above. Data is loaded from STA, quotes are removed from the column names, datatypes are changed on the columns required, and the data is loaded into ODS.

3. Data Loading

The last step in the data pipeline is to load or integrate the data into the Data Warehouse (DWH). One common database schema that is used is the Star schema. In this schema, we have one main table called the “Fact Table” That is surrounded by “Dimension Tables”. The fact table contains the most important data called “facts”, whereas the dimensions tables give addition descriptive information. We will design our database around this schema. Our design will have orders as our fact table with everything else being a dimension table. Order items, order reviews, and order payments are part of our orders fact table however due to limitations with the data, we decided to leave them as separate tables. They effectively work as 1 large fact table broken up into different sections.

A common dimension table that we added into the data is the “Date” dimension. It allows you to describe the dates with multiple temporal aggregate categories (year, month, quarter). The table will contain multiple variations of those three data to be able to adapt to various styles of queries. We define the relation to the fact table with a technical key with the format “YYYYMM”.

a) Integration of the Date Dimension

This dimension is describing dates, in particular months. Since we don’t need external data to build this dimension, we use a SQL script to make it.

First, we create an empty table:

```sql
USE [Project_DWH]
CREATE TABLE [dbo].[DimDate] (
    DateKey INT NOT NULL PRIMARY KEY,
    [Date] DATE NULL,
    [Day] TINYINT NOT NULL,
    [DaySuffix] CHAR(2) NOT NULL,
    [Weekday] TINYINT NOT NULL,
    [WeekDayName] VARCHAR(10) NOT NULL,
    [WeekDayName_Short] CHAR(3) NOT NULL,
    [WeekDayName_FirstLetter] CHAR(1) NOT NULL,
    [DOWInMonth] TINYINT NOT NULL,
    [DayOfYear] SMALLINT NOT NULL,
    [WeekOfMonth] TINYINT NOT NULL,
    [WeekOfYear] TINYINT NOT NULL,
    [Month] TINYINT NOT NULL,
    [MonthName] VARCHAR(10) NOT NULL,
    [MonthName_Short] CHAR(3) NOT NULL,
    [MonthName_FirstLetter] CHAR(1) NOT NULL,
    [Quarter] TINYINT NOT NULL,
    [QuarterName] VARCHAR(6) NOT NULL,
    [Year] INT NOT NULL,
    [MMYYYY] CHAR(6) NOT NULL,
    [MonthYear] CHAR(7) NOT NULL,
    IsWeekend BIT NOT NULL,
    )


    GO
```

Then to build the data, we use the following script:

```sql
  SET NOCOUNT ON

TRUNCATE TABLE DimDate


DECLARE @CurrentDate DATE = '2016-01-01'
DECLARE @EndDate DATE = '2021-12-31'

--Insertion of a row to get a DateKey for the NULL Dates in Orders (Not delivered yet orders)
INSERT INTO [dbo].[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]
```

```sql
VALUES (
   -1,
   NULL,
   0,
   '',
   0,
   ''
   ,
   '',
   '',
   0,
   0,
   0,
   0,
   0,
   '',
   '',
   '',
   0,
   '',
   0,
   '',
   '',
   0
)

WHILE @CurrentDate < @EndDate
BEGIN
```

```sql
BEGIN
   INSERT INTO [dbo].[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]
      )
      SELECT DateKey = YEAR(@CurrentDate) * 10000 + MONTH(@CurrentDate) * 100 + DAY(@CurrentDate),
         DATE = @CurrentDate,
         Day = DAY(@CurrentDate),
         [DaySuffix] = CASE
            WHEN DAY(@CurrentDate) = 1
               OR DAY(@CurrentDate) = 21
               OR DAY(@CurrentDate) = 31
               THEN 'st'
            WHEN DAY(@CurrentDate) = 2
               OR DAY(@CurrentDate) = 22
               THEN 'nd'
            WHEN DAY(@CurrentDate) = 3
               OR DAY(@CurrentDate) = 23
               THEN 'rd'
            ELSE 'th'
            END,
         WEEKDAY = DATEPART(dw, @CurrentDate),
         WeekDayName = DATENAME(dw, @CurrentDate),
         WeekDayName_Short = UPPER(LEFT(DATENAME(dw, @CurrentDate), 3)),
         WeekDayName_FirstLetter = LEFT(DATENAME(dw, @CurrentDate), 1),
         [DOWInMonth] = DAY(@CurrentDate),
         [DayOfYear] = DATENAME(dy, @CurrentDate),
         [WeekOfMonth] = DATEPART(WEEK, @CurrentDate) - DATEPART(WEEK, DATEADD(MM, DATEDIFF(MM, 0, @CurrentDate), 0)) + 1,
         [WeekOfYear] = DATEPART(wk, @CurrentDate),
         [Month] = MONTH(@CurrentDate),
         [MonthName] = DATENAME(mm, @CurrentDate),
         [MonthName_Short] = UPPER(LEFT(DATENAME(mm, @CurrentDate), 3)),
         [MonthName_FirstLetter] = LEFT(DATENAME(mm, @CurrentDate), 1),
         [Quarter] = DATEPART(q, @CurrentDate),
         [QuarterName] = CASE
            WHEN DATENAME(qq, @CurrentDate) = 1
               THEN 'First'
            WHEN DATENAME(qq, @CurrentDate) = 2
               THEN 'second'
            WHEN DATENAME(qq, @CurrentDate) = 3
               THEN 'third'
            WHEN DATENAME(qq, @CurrentDate) = 4
               THEN 'fourth'
            END,
         [Year] = YEAR(@CurrentDate),
         [MMYYYY] = RIGHT('0' + CAST(MONTH(@CurrentDate) AS VARCHAR(2)), 2) + CAST(YEAR(@CurrentDate) AS VARCHAR(4)),
         [MonthYear] = CAST(YEAR(@CurrentDate) AS VARCHAR(4)) + UPPER(LEFT(DATENAME(mm, @CurrentDate), 3)),
         [IsWeekend] = CASE
            WHEN DATENAME(dw, @CurrentDate) = 'Sunday'
               OR DATENAME(dw, @CurrentDate) = 'Saturday'
               THEN 1
            ELSE 0
            END

      SET @CurrentDate = DATEADD(DD, 1, @CurrentDate)
   END
```

Here is the first ten lines of the results:

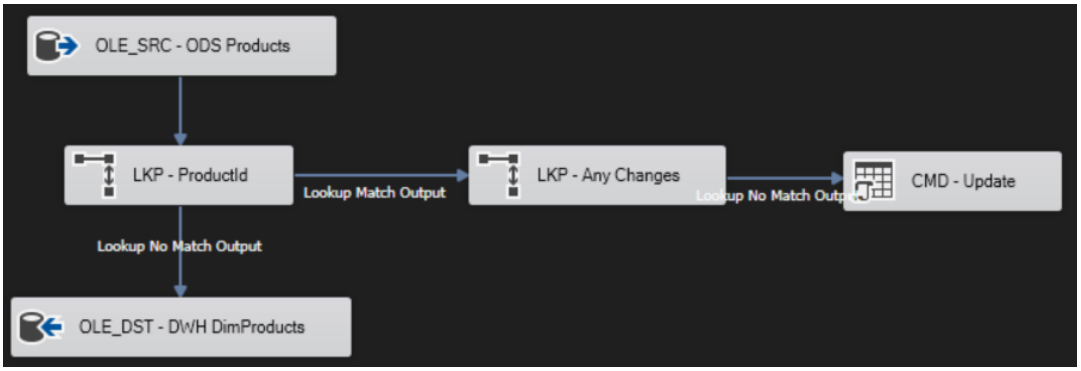
Results | Messages

	DateKey	Date	Day	DaySuffix	Weekday	WeekDayName	WeekDayName_Short	WeekDayName_FirstLetter	DOWInMonth	DayOfYear	WeekOfMonth	WeekOfYear	Month	MonthName	MonthName_Short
1	-1	NULL	0		0				0	0	0	0	0		
2	20160101	2016-01-01	1	st	6	Friday	FRI	F	1	1	1	1	1	January	JAN
3	20160102	2016-01-02	2	nd	7	Saturday	SAT	S	2	2	1	1	1	January	JAN
4	20160103	2016-01-03	3	rd	1	Sunday	SUN	S	3	3	2	2	1	January	JAN
5	20160104	2016-01-04	4	th	2	Monday	MON	M	4	4	2	2	1	January	JAN
6	20160105	2016-01-05	5	th	3	Tuesday	TUE	T	5	5	2	2	1	January	JAN
7	20160106	2016-01-06	6	th	4	Wednesday	WED	W	6	6	2	2	1	January	JAN
8	20160107	2016-01-07	7	th	5	Thursday	THU	T	7	7	2	2	1	January	JAN
9	20160108	2016-01-08	8	th	6	Friday	FRI	F	8	8	2	2	1	January	JAN
10	20160109	2016-01-09	9	th	7	Saturday	SAT	S	9	9	2	2	1	January	JAN

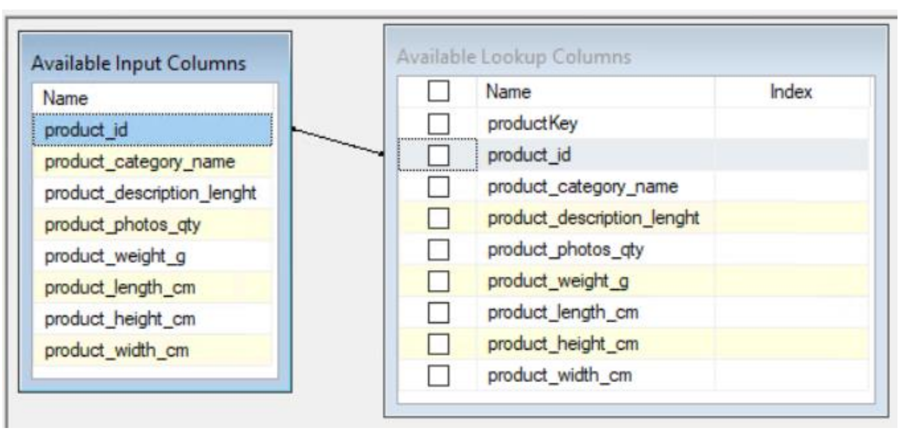
To accept new data in the data warehouse, this dimension can be updated. We would need to change the “@EndDate” variable to include all the dates required.

b) Integration of the Products Dimension

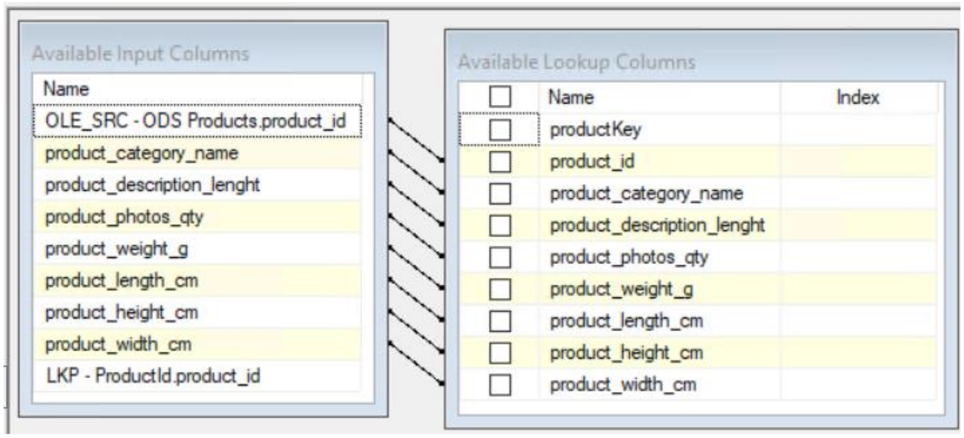
We load this data with the process below:



We need to make sure that we can make joins between the fact table and the dimension table, so we check the link with “ProductID”.



For the dimensions tables, we also need to have an update policy. We choose the SCD1 strategy, implemented by checking is there is any change and updating the table if necessary.



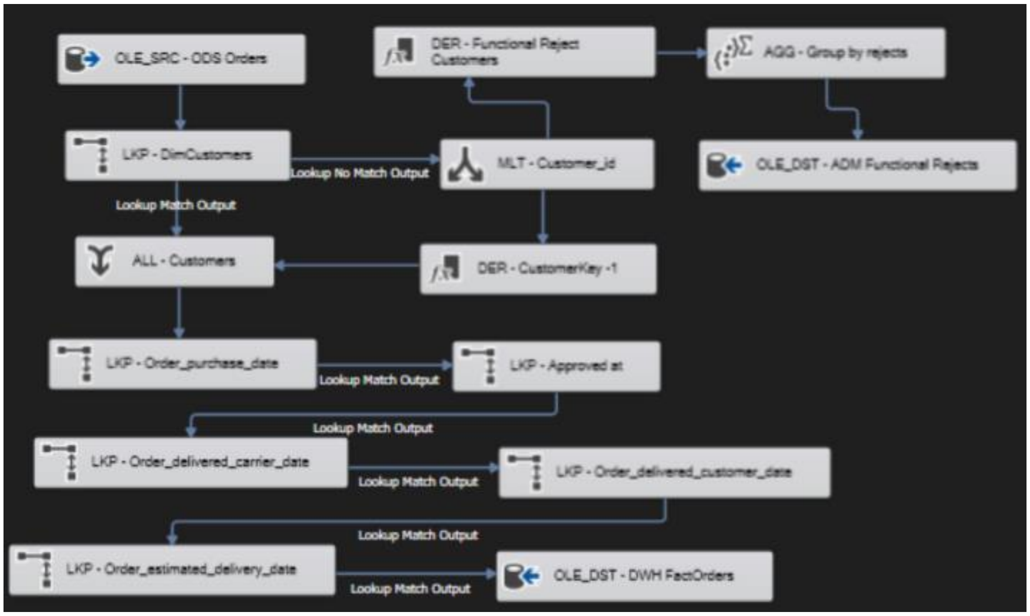
Here is the first ten lines of the results:

	productKey	product_id	product_category_name	product_description_lenght	product_photos_qty	product_weight_g	product_length_cm	product_height_cm	product_width_cm
1	1	b5694f734bee2683d25bb188a7f45b40	stationery	684	2	100	16	2	11
2	2	41b24e4f49504cc023666ed7cde3cf16	sports_leisure	628	7	1800	25	30	40
3	3	e987fc6af8290fcf1db0510d52b65795	health_beauty	99	2	250	20	10	15
4	4	6e2d41323c4b6d240b530b9e68e83de1	watches_gifts	626	1	400	23	15	19
5	5	e76572c2ae4cede1e3b2ead512423efe	bed_bath_table	352	2	450	45	11	11
6	6	e97ee7673459375fac14fe9a01a64e6c	toys	515	1	289	17	15	17
7	7	d212ad4ac52091c8d50442a89fbbc7af	bed_bath_table	160	1	350	16	16	16
8	8	99f83ce7f14247f186cb52d73bdd6e79	construction_tools_construction	453	1	100	16	7	12
9	9	9539cddd99ab3f8ca77b6cad0a0899ea	sports_leisure	1067	4	433	23	12	17
10	10	9fae04deab3bb1c353c27c50cbf3b42b	toys	927	4	450	21	36	11

The other Dimension tables follow the same structure as the one above.

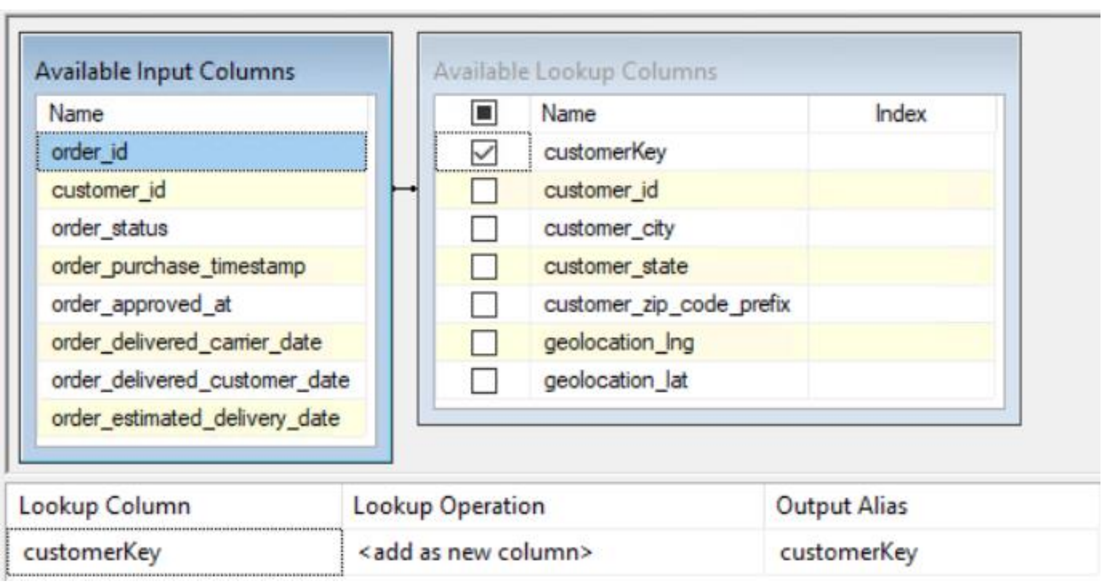
c) Integration of the orders fact table

Now that we finally have our dimensions tables, we can build our fact table while checking valid relations with the dimensions. We verify and load the data as follows:



In case of an error in the processing, we generate a functional reject and insert it into a table “Functional_Rejects”.

The first dimension we check is “DimCustomers”. We check it with the key between “customer_id” and “customer_id”. We also add a technical key “customerKey_FK”.



We tack the errors with the same method that in ODS. For a missing relation with “DimCustomers”, we generate the following data:

Derived Column Name	Derived Column	Expression	Data Type	Length
RejectDate	<add as new column>	GETDATE()	database timestamp [DT_DBTIMESTAMP]	
RejectPackageAndTask	<add as new column>	(DT_WSTR,100)@[System::PackageName] + " AND " + (DT_WSTR,100)@[System::TaskName]	Unicode string [DT_WSTR]	205
RejectColumn	<add as new column>	"customer_id"	Unicode string [DT_WSTR]	11
RejectDescription	<add as new column>	"This customer_id" + (DT_WSTR,50)customer_id + " doesn't exist in our DimCustomer table"	Unicode string [DT_WSTR]	106

We then group by the rejects:

Available Input Columns

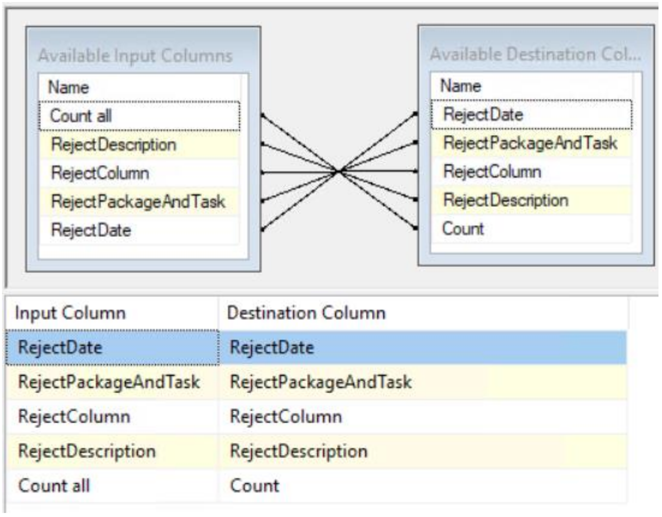
☐	Name
☑	(*)
☐	order_id
☐	customer_id
☐	order_status
☐	order_purchase_timestamp
☐	order_approved_at
☐	order_delivered_carrier_d...
☐	order_delivered_custome...
☐	order_estimated_delivery...
☑	RejectDate
☑	RejectPackageAndTask
☑	RejectColumn
☑	RejectDescription

Input Column	Output Alias	Operation
(*)	Count all	Count all
RejectDescription	RejectDescription	Group by
RejectColumn	RejectColumn	Group by
RejectPackageAndTask	RejectPackageAndTask	Group by
RejectDate	RejectDate	Group by

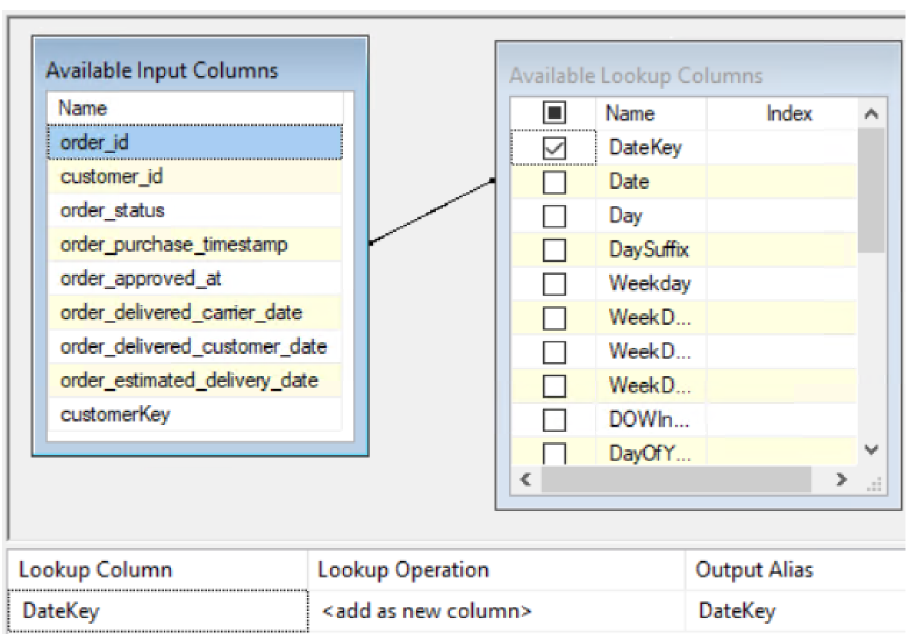
In this case, we will also integrate the record with the default value (-1) for the Customer:

Derived Column Name	Derived Column	Expression	Data Type
customerKey	<add as new column>	-1	four-byte signed integer [DT_I4]

As shown here, we are keeping track of the rejects as they are sent into functional rejects:

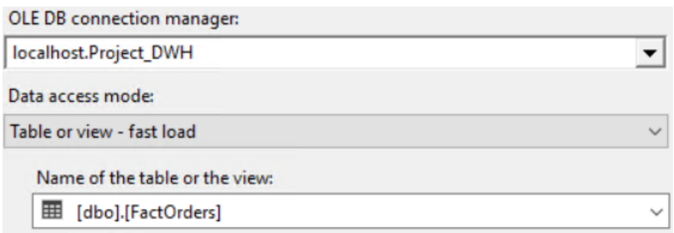


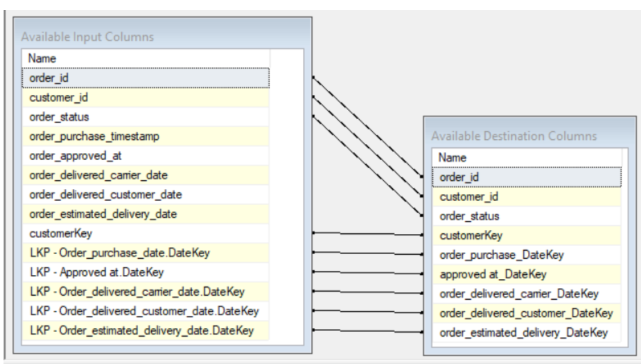
Next for all the columns containing dates, a lookup is performed using the Dimdate table.



Finally we can create and load the fact table into the data warehouse.

```
USE [Project_DWH]
CREATE TABLE [dbo].[FactOrders] (
    [order_id] varchar(50),
    [customer_id] varchar(50),
    [order_status] varchar(50),
    [customerKey] int,
    [order_purchase_DateKey] int,
    [approved at_DateKey] int,
    [order_delivered_carrier_DateKey] int,
    [order_delivered_customer_DateKey] int,
    [order_estimated_delivery_DateKey] int
)
```





Input Column	Destination Column
order_id	order_id
customer_id	customer_id
order_status	order_status
customerKey	customerKey
LKP - Order_purchase_date.DateKey	order_purchase_DateKey
LKP - Approved at.DateKey	approved at_DateKey
LKP - Order_delivered_carrier_date.DateKey	order_delivered_carrier_DateKey
LKP - Order_delivered_customer_date.DateKey	order_delivered_customer_DateKey
LKP - Order_estimated_delivery_date.DateKey	order_estimated_delivery_DateKey

Here is the first ten lines of the results of the facts table “orders”:

Results | Messages

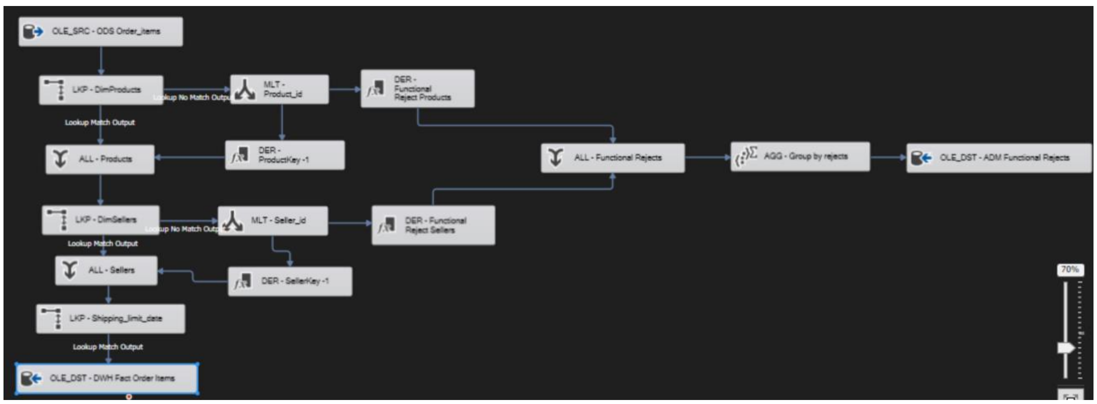
	order_id	customer_id	order_status	customerKey	order_purchase_DateKey	approved at_DateKey	order_delivered_carrier_DateKey	order_delivered_customer_DateKey
1	fc1d9429da446a147d438d41350a2932	8b13d2050940cb8d6a7203e69f279254	delivered	3566	20171023	20171024	20170611	20171118
2	8a6bf1083cc2efc062e037e474622de6	64f0ed97c13c1388088e93454db40488	delivered	16866	20170320	20170320	20170325	20170104
3	bcd37480e5f21eaf2b524c8e9724d099	fbc839b74c648a39b750799b58a13e0f	delivered	47155	20171124	20171125	20171128	20170712
4	b951cb6562f56cfb404cd77c81262f25	83116a97d5035c15ca0e75dc2b848bd9	delivered	64538	20171226	20171226	20171227	20180401
5	b825b53d8136d85eb4cbe60a1bd275f5	ec4f3f98da53dbaa4db16505b37f82c0	delivered	79211	20170509	20170609	20170609	20171209
6	b1e68c1ac52a1a5cb5051d07ae91ff06	cf34464d3b5d4a957dae9c0ef4a7d27e	delivered	32037	20170318	20170318	20170321	20170327
7	4516299f280a0f30720256ae0cd592d6	582ec2f9229bb3014866b31f73def34f	delivered	28312	20170913	20170913	20170914	20171018
8	7d63cb1d349e7bd0ee1edcc61ea71077	484711e3d14ac75fb874e949b3a28395	delivered	72746	20170814	20170814	20170816	20170824
9	b11073519dc9fe626df6281aa1965522	4f3ef3bae0d531a175d66f875cba84d1	delivered	22825	20171116	20171116	20171121	20171127
10	c7b44077f6d014b67c3164e103e3bf02	19187bc9fab7457748d4dc42a9e1c6fe	delivered	37298	20171121	20171121	20171123	20171130

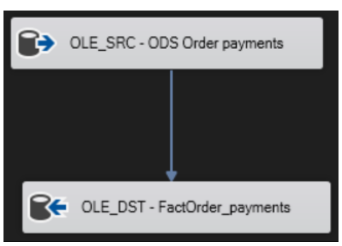
The rest of the fact tables follow the same structure as above. This structure in summary is:

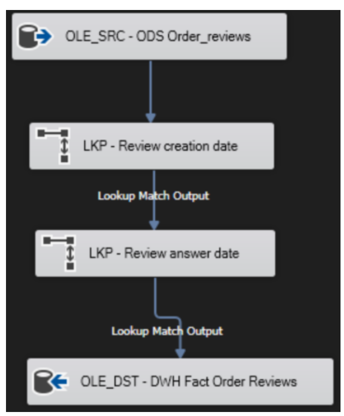
For the dimension(s) it contains:

- Lookup with the Dimension primary key and grab the surrogate key. In case of a no match, redirect the row to functional error and use a multicast to, at the same time follow the row in the main stream but the surrogate key is -1
- Lookup with Dimdate for each date it contains.
- In what is inserted in the DWH Fact table, the primary keys of the dimensional columns are replaced with their surrogate key. The dates are replaced with datekey.

Their dataflows are shown below:







III. Project Deployment

We have defined all required steps to deploy our data warehouse with the available data. To ensure that the pipeline is executed in a reproducible manner, we define an additional package “Project_ETL” with containers of all the tasks.

Using this package allows us to use one interface for the entire pipeline. The containers make sure everything is launched in the correct order.

VI. Analysis

In this part, we decided to choose 4 business questions:

1. Which product categories have the highest customer satisfaction?
2. Which month do customers order the most?
3. How many orders have not been delivered?
4. In which of the company's cities do we have the biggest profits?
5.

To answer each question, we built a specific query.

1. Which product categories have the highest customer satisfaction?

It's always important to know which products are popular with customers. This data can be used by the marketing or business team to make decisions:

a) Query

```
SELECT TOP(5) prod.product_category_name, rev.review_score, COUNT(rev.review_score) as
number_of_reviews
FROM
DimProducts as prod
JOIN FactOrder_items as ord ON prod.productKey = ord.productKey
JOIN FactOrder_reviews as rev on ord.order_id = rev.order_id
WHERE rev.review_score = 5
GROUP BY prod.product_category_name, rev.review_score
ORDER BY number_of_reviews DESC
```

b) Result

	product_category_name	review_score	number_of_reviews
1	health_beauty	5	11716
2	bed_bath_table	5	11570
3	sports_leisure	5	10242
4	furniture_decor	5	8904
5	computers_accessories	5	8400

As far as our results are concerned, we can see that the health and beauty category is the best-selling, which seems logical. But the second category seems more surprising and unclear. This category may be linked to a famous company that sells products for the home.

2. Which month do customers order the most?

Knowing the months in which customers buy the most enables the company to make strategic and financial decisions. For example, to advertise or restock certain products.

a) Query

```
SELECT COUNT(order_id) as nb_orders, MonthName

FROM FactOrders as ord

JOIN DimDate as date_ ON ord.[approved at_DateKey] = date_.DateKey

GROUP BY MonthName

ORDER BY COUNT(order_id) DESC
```

b) Result

	nb_orders	MonthName
1	10450	July
2	9857	May
3	9239	March
4	9073	August
5	8992	June
6	8777	November
7	8572	April
8	8388	January
9	8009	February
10	6140	October
11	5903	December
12	5881	September

Surprisingly, July is the month when customers buy the most. We can see that the top 5 months are in summer or spring.

3. How many orders have not been delivered?

One of the most important parts of a marketplace is delivery. This means that the order has been completed. When the order is not fulfilled, for whatever reason, it means that the marketplace loses a sale. So it's important to know how many orders are not delivered.

a) Query

```
SELECT

COUNT(order_id) AS total_orders,

COUNT(CASE WHEN order_status != 'delivered' THEN 1 END) AS not_delivered_orders

FROM

  FactOrders;
```

** this query was created with the help of ChatGPT. After 1 hour of trying to figure out why the query wasn't working, we asked ChatGPT to help us understand our mistake: we hadn't put the CASE WHEN in the right place.*

b) Result

	total_orders	not_delivered_orders
1	99281	2817

The number of orders approved but not delivered represents around 3%. Depending on the company's wishes, this figure may seem high.

4. In which of the company's cities do we have the biggest profits?

It will be interesting to find out which city sellers make the most profit. We could find correlations between the city and profits.

a) Query

```
SELECT seller.seller_city, SUM(ord_items.price) as benef

FROM DimSellers as seller

JOIN FactOrder_items as ord_items ON seller.sellerKey = ord_items.sellerKey

GROUP BY seller_city

ORDER BY benef DESC    FactOrders;
```

b) Result

	seller_city	benef
1	sao paulo	5405756.28
2	ibitinga	1249185.88
3	curitiba	941519.64
4	rio de janeiro	716827.18
5	guarulhos	658988.76
6	ribeirao preto	551952.88
7	itaquaquecetuba	461136.24
8	guariba	458945.26
9	santo andre	457123.20
10	lauro de freitas	451050.10
11	piracicaba	425355.64
12	belo horizonte	415342.46

Looking at the data, we can see that the sellers are located in Brazil and that the first city, Sao Paulo, has a significant lead over the other cities. We can assume that there are more companies located in Sao Paulo. It will be interesting to identify this number.

5. Are top-rated products with a low average distance between customer and seller?

The result of this query can help us understand whether the distance between the customer and the salesperson can have an impact on the price of the product. Distance is important because it can impact the delivery delay or the condition of the parcel on arrival.

To calculate the distance we use this formula:

Distance

This uses the '**haversine**' formula to calculate the great-circle distance between two points – that is, the shortest distance over the earth's surface – giving an 'as-the-crow-flies' distance between the points (ignoring any hills they fly over, of course!).

Haversine formula:

$$a = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$$

$$c = 2 \cdot \mathrm{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

where: φ is latitude, λ is longitude, R is earth's radius (mean radius = 6,371km); note that angles need to be in radians to pass to trig functions!

a) Query

WITH lat_lng_radians AS

(

SELECT RADIANS(sellers.geolocation_lat) as seller_lat, RADIANS(sellers.geolocation_lng) as seller_lng,

```
	RADIANS(customers.geolocation_lat) as customer_lat,
RADIANS(customers.geolocation_lng) as customer_lng, order_reviews.review_score,
customers.customer_id

	FROM dbo.FactOrders as orders

		JOIN dbo.DimCustomers as customers ON orders.customerKey =
customers.customerKey

		JOIN dbo.FactOrder_items as order_items ON order_items.order_id =
orders.order_id

		JOIN dbo.DimSellers as sellers ON sellers.sellerKey = order_items.sellerKey

		JOIN dbo.FactOrder_reviews as order_reviews ON order_reviews.order_id =
orders.order_id

	WHERE order_status = 'delivered'

)


SELECT AVG(

			ROUND(2 * 6371 * ASIN(

			SQRT(

				POWER(SIN((customer_lat - seller_lat)/2), 2) + COS(seller_lat) *
COS(customer_lat) * POWER(SIN((customer_lng - seller_lng)/2), 2)

				)

			), 1)

		) as mean_distance_km, review_score

FROM lat_lng_radians

GROUP BY review_score

ORDER BY review_score DESC
```

b) Result

	mean_distance_km	review_score
1	569.374808858838	5
2	620.466844515795	4
3	629.699567146411	3
4	645.225628548257	2
5	659.915650099402	1

We can see that the highest score, 5, is linked to the short distance between customer and salesperson. Using other variables, we can deduce that one of the most important factors impacting evaluation is delivery time.