02/10/2023

SSIS & SQL Server project report

Valentina Rodriguez Lopez
DSTI – S23

Table of Content

Introduction

To be able to use and extract value from available data, it first needs to be integrated into an IT system. This imply that all the data coming from various sources need to be unified and standardized to be used by other programs. One way to achieve this is to implement a Data Warehouse.

In this project, we are going to design and implement a Data Warehouse with historical data of crimes in France. For this, we will use SQL Server and SSIS.

Data

Our data is composed of two Excel files.

The first file “Crimes_in_France_1996_2016.xls” has historical data of crimes in Metropolitan France, from 1996 to 2016. The data is arranged as follows [1]:

- Each column indicates the year and month statistics in the format YYYY_MM
- Each line represents a specific type of crime or offense
- Each tab (sheet) represents a French department

The second file “Departments mapping.csv” has the following data about the “Départements”:

- “Code Postal”, the zip code.
- “Département”, the name.
- “Indicatif Téléphonique”, the first digit of a fixed-line phone number.
- “Région”, the county.
- “Zone Vacances”, the code of zone associated with a holiday schedule.

Pipeline design

The pipeline will be done in three steps:

- The Staging area (STA) will allow to load the data as is, or with minimal changes.
- The Operational Data Store area (ODS) will allow to clean and standardize the data. If the data don’t pass the quality criteriums, they will be put in the “Technical_Rejects” table as technical rejects.
- The 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.

[1] « Subject of the project.docx », Emerick Duval.

There will be one STA and ODS packages per file.

Staging database

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

Crimes Table

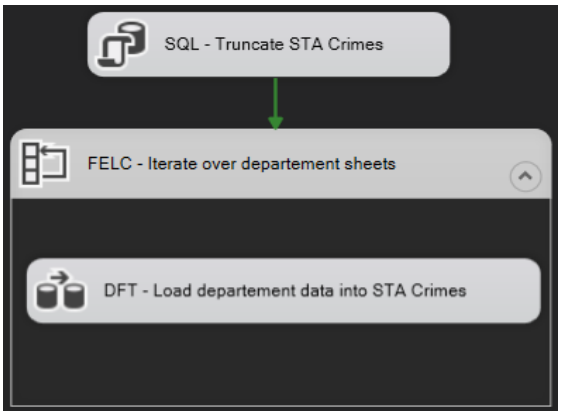
Here is an extract of one sheets of “Crimes_in_France_1996_2016.xls” :

Index	Libellé index	2016_03	2016_02	2016_01	2015_12	2015_11
1	Règlements de compte entre malfaireurs	5	3	1	0	6
2	Homicides pour voler et à l'occasion de vols	0	0	0	0	0
3	Homicides pour d'autres motifs	1	1	1	2	0
4	Tentatives d'homicides pour voler et à l'occasion de vols	0	0	0	1	0
5	Tentatives homicides pour d'autres motifs	13	11	6	9	11

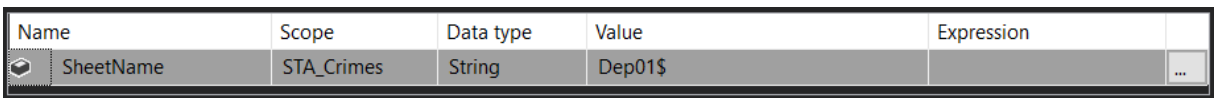
In addition, the data is separated into one sheet by “Département” :

To be able to use the data in the file, we need to put it into one table. This means that we need to extract the data from each data sheet, while also keeping track of the source sheet (“Département”).

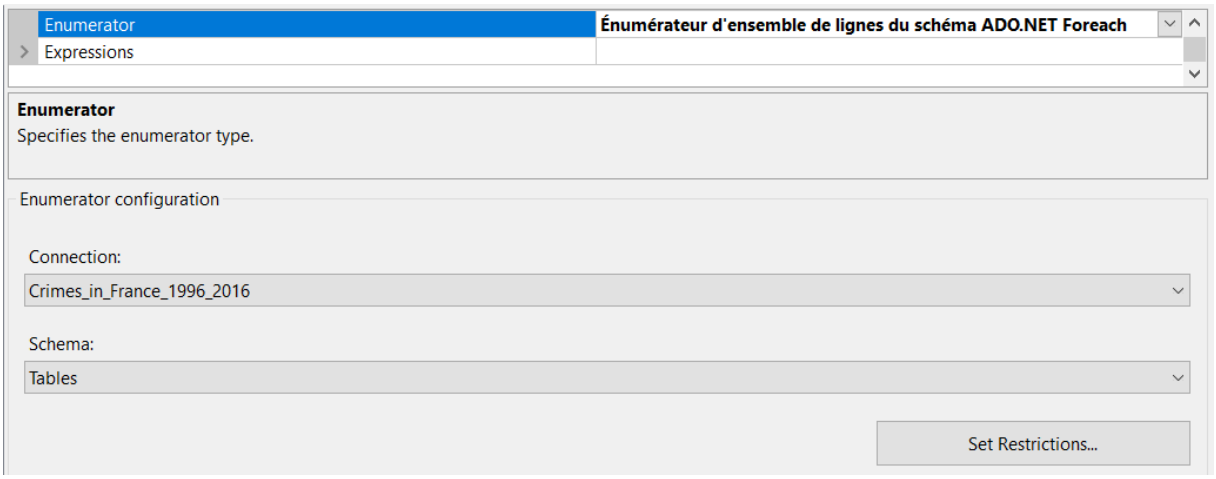
To go into all data sheets, we use a “Foreach Loop Container” where we put the extraction dataflow inside. The data flow in the container will be able to load one sheet at the time. Here, we also truncate the data from the previous runs.



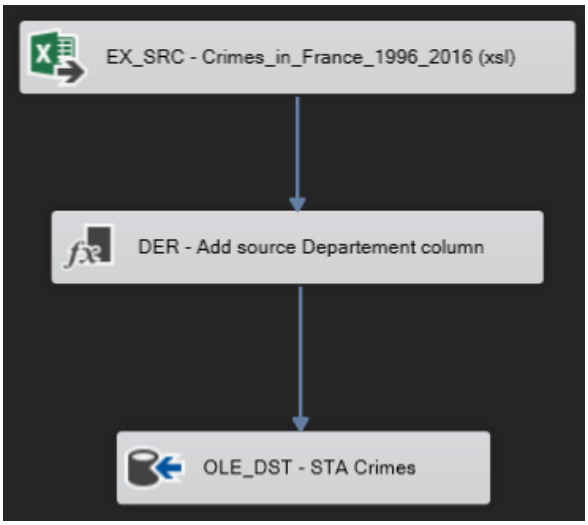
The foreach loop is iterating over a variable “SheetName” that allow to address each sheet separately. The variable is of the format “DepXX$” , with “XX” being the department number.



We use the enumerator “ADO.NET Schema Rowset Enumerator” to be able to iterate over the tables of the (Excel) data source.



Next, the dataflow is defined as follow :



To keep track of the source “Département” sheet, we define a derived column “Departement” using the variable “SheetName” as value.

Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
Departement	<add as new column>	@[User::SheetName]	chaîne Unicode [DT_...	6			

Once we have extracted the data, we can load it into our target table “Crimes” into the STA database.

First, we check if the table already exists and drop it. This is to prevent interference of the previous runs of the package.

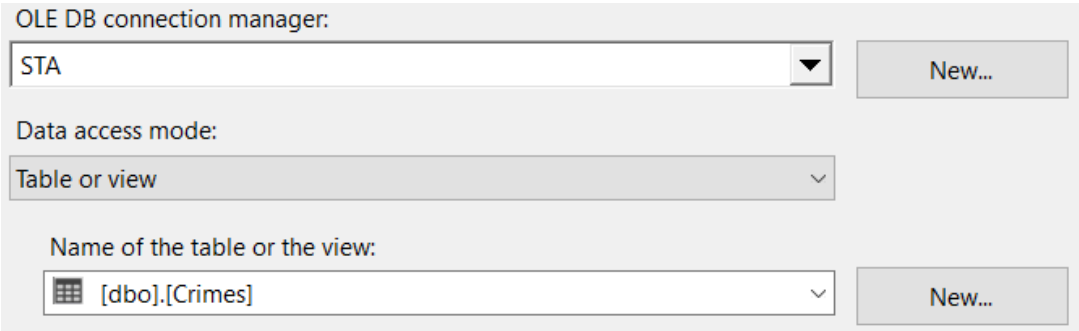
```
IF (EXISTS (SELECT *
                 FROM INFORMATION_SCHEMA.TABLES
                 WHERE TABLE_SCHEMA = 'dbo'
                 AND  TABLE_NAME = 'Crimes'
                 and TABLE_TYPE = 'BASE TABLE'))
    BEGIN
        DROP TABLE dbo.Crimes
    END
GO
```

Then we create the table with the following script:

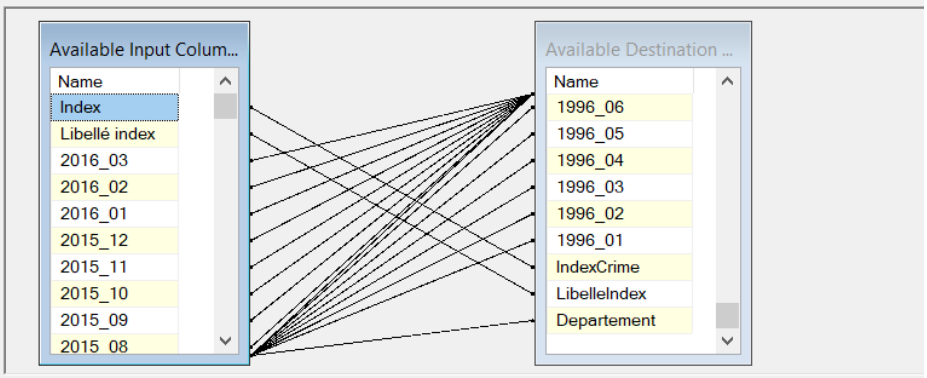
```
CREATE TABLE [dbo].[Crimes](
    [Departement] [nvarchar](255) NULL,
    [IndexCrime] [nvarchar](255) NULL,
    [LibelleIndex] [nvarchar](255) NULL,
    [2016_03] [nvarchar](255) NULL,
    [2016_02] [nvarchar](255) NULL,
    [2016_01] [nvarchar](255) NULL,
    [2015_12] [nvarchar](255) NULL,
    [2015_11] [nvarchar](255) NULL,
    [2015_10] [nvarchar](255) NULL,
                  •
                  •
                  •
    [1996_06] [nvarchar](255) NULL,
    [1996_05] [nvarchar](255) NULL,
    [1996_04] [nvarchar](255) NULL,
    [1996_03] [nvarchar](255) NULL,
    [1996_02] [nvarchar](255) NULL,
    [1996_01] [nvarchar](255) NULL
) ON [PRIMARY]
GO
```

All values are set to nvarchar(255) for now , to be able to accept all data.

We can now define the target destination.



And finally, we define the columns mapping as follow:



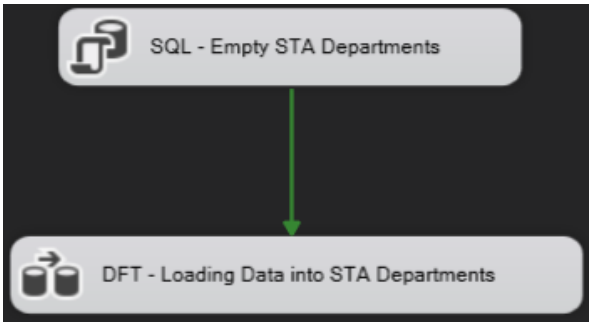
Input Column	Destination Column
1996_07	1996_07
1996_06	1996_06
1996_05	1996_05
1996_04	1996_04
1996_03	1996_03
1996_02	1996_02
1996_01	1996_01
Index	IndexCrime
Libellé index	LibelleIndex
Departement	Departement

Here is the first ten lines of the results:

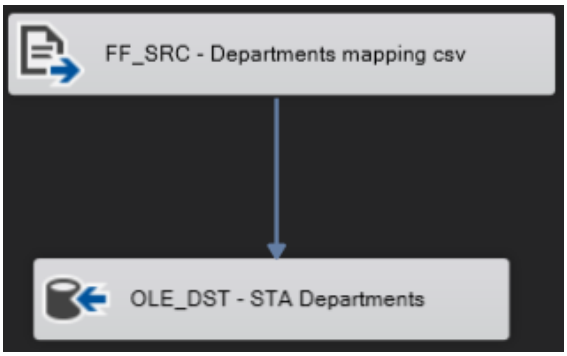
	Departement	IndexCrime	LibelleIndex	2016_03	2016_02	2016_01	2015_12	2015_11	2015_10	2015_09	2015_08	2015_07	2015_06	2015_05
1	Dep01$	1	Règlements de compte entre malfaiteurs	0	0	0	0	0	0	0	0	0	0	0
2	Dep01$	2	Homicides pour voler et à l'occasion de vols	0	0	0	0	0	0	0	0	0	0	0
3	Dep01$	3	Homicides pour d'autres motifs	1	1	0	0	0	0	0	0	0	0	0
4	Dep01$	4	Tentatives d'homicides pour voler et à l'occasion...	0	0	0	0	0	0	0	0	0	0	0
5	Dep01$	5	Tentatives homicides pour d'autres motifs	0	0	0	0	0	1	1	0	1	0	0
6	Dep01$	6	Coups et blessures volontaires suivis de mort	0	0	0	0	0	0	1	0	0	0	1
7	Dep01$	7	Autres coups et blessures volontaires criminels ...	120	117	137	130	111	114	123	102	156	153	134
8	Dep01$	8	Prises d'otages à l'occasion de vols	0	0	0	0	0	0	0	0	1	1	0
9	Dep01$	9	Prises d'otages dans un autre but	0	0	0	0	0	0	0	0	0	0	0
10	Dep01$	10	Sequestrations	4	2	2	1	3	1	1	0	1	3	0

Departments Table

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



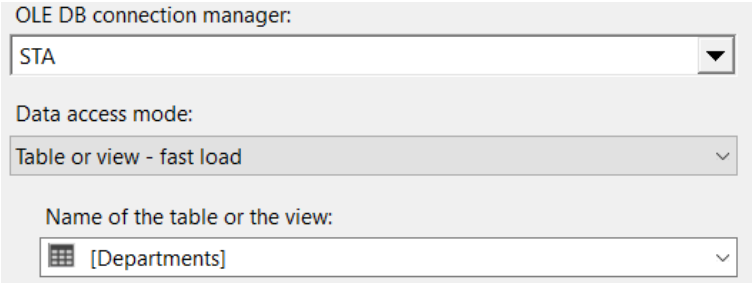
The dataflow is an import of a flat file :



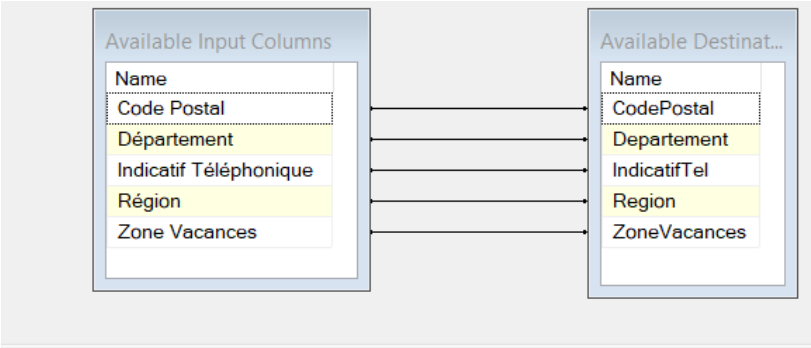
We need to create the destination table with the following command:

```
USE [STA]
GO

/****** Object:  Table [dbo].[Departments]    Script Date: 11/12/2021 10:06:14 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Departments](
    [CodePostal] [varchar](255) NULL,
    [Departement] [varchar](255) NULL,
    [IndicatifTel] [varchar](255) NULL,
    [Region] [varchar](255) NULL,
    [ZoneVacances] [varchar](255) NULL
) ON [PRIMARY]
GO
```

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



We also make sure to change some of the names to avoid problems with accents, spaces and SQL reserved words:



Input Column	Destination Column
Code Postal	CodePostal
Département	Departement
[Indicatif Téléphonique]	IndicatifTel
Région	Region
Zone Vacances	ZoneVacances

Here is the first ten lines of the results:

Results Messages

	CodePostal	Departement	IndicatifTel	Region	ZoneVacances
1	1	Ain	4	Auvergne-Rhône-Alpes	A
2	2	Aisne	3	Hauts-de-France	B
3	3	Allier	4	Auvergne-Rhône-Alpes	A
4	4	Alpes-de-Haute-Provence	4	Provence-Alpes-Côte d'Azur	B
5	5	Hautes-Alpes	4	Provence-Alpes-Côte d'Azur	B
6	6	Alpes-Maritimes	4	Provence-Alpes-Côte d'Azur	B
7	7	Ardèche	4	Auvergne-Rhône-Alpes	A
8	8	Ardennes	3	Grand-Est	B
9	9	Ariège	5	Occitanie	C
10	10	Aube	3	Grand-Est	B

Operational Data Store

The second step of the pipeline is to load usable data into the Operational Data Store. 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 do not respect the “quality standards “will be rejected as a technical reject.

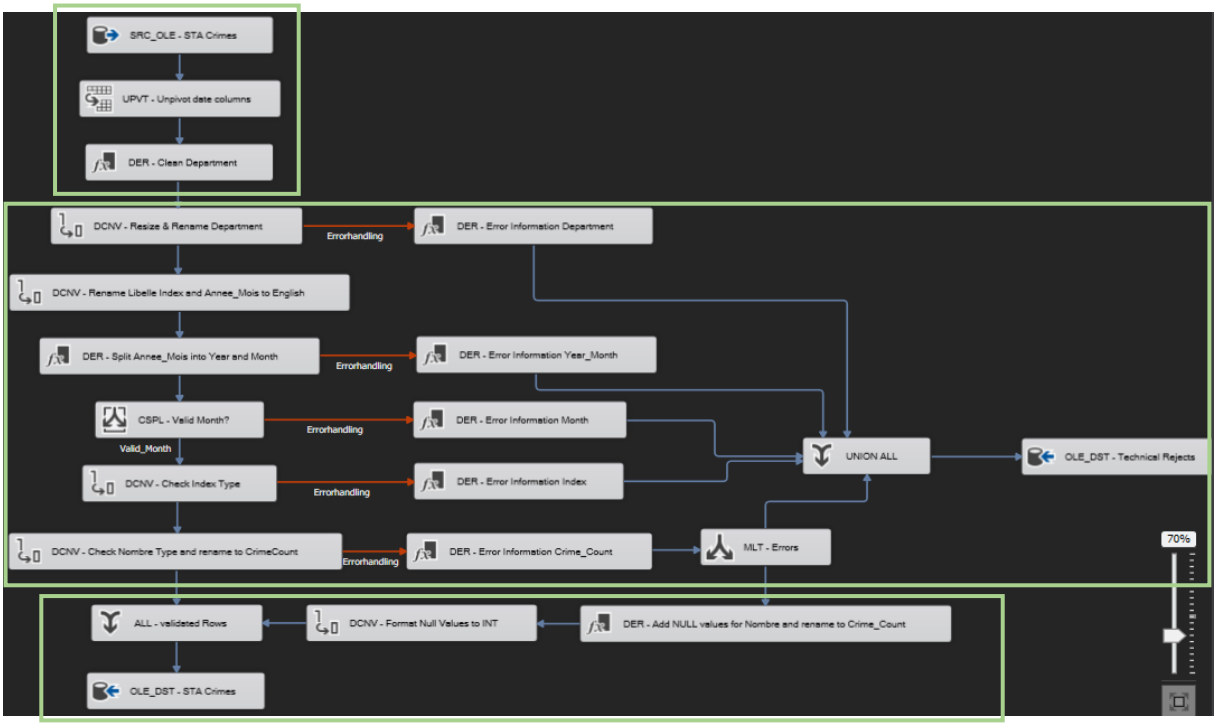
The quality standards are based on the “correctness” of the data. The output data must me consistent in data types and in values. We also need to ensure that the data can used in queries, so we might need to reorganize and enrich the data.

Crimes Table

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

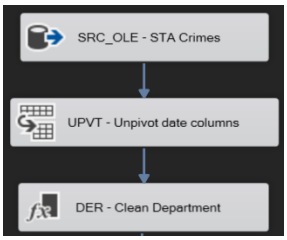
Here we can see a warning, this is due to a possible truncation on the error messages.

The data flow is defined as follow:



We are going to detail the three segments of this dataflow.

In the first segment bellow, we change the format of the table into one that is easier to query:



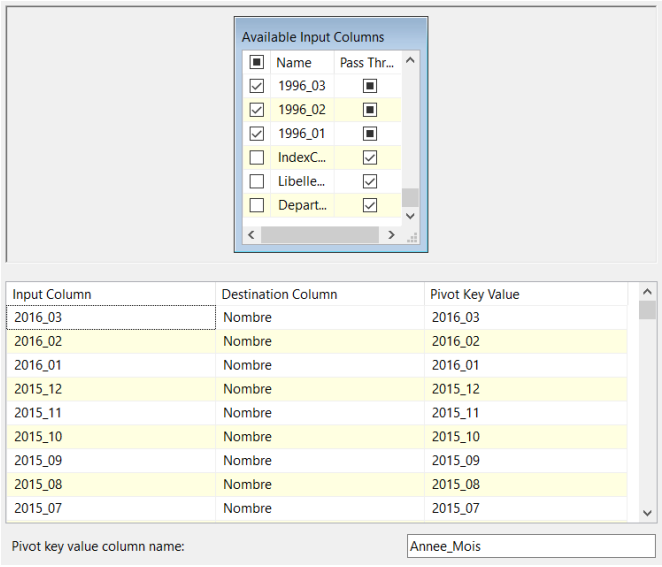
The initial data format, of the Crimes table, have the time series data organized into columns and the categories are defined in the rows:

Libellé index	2016_03	2016_02	2016_01
Règlements de compte entre malfaiteurs	8	6	11
Homicides pour voler et à l'occasion de vols	3	1	5

The desired output would have one row per date, with the associated category and value:

Libellé index	Période	Valeur
Règlements de compte entre malfaiteurs	2016_03	8
Règlements de compte entre malfaiteurs	2016_02	6
Règlements de compte entre malfaiteurs	2016_01	11
Homicides pour voler et à l'occasion de vols	2016_03	3
Homicides pour voler et à l'occasion de vols	2016_02	1
Homicides pour voler et à l'occasion de vols	2016_01	5

This operation is called “Unpivot” and we apply it to the date columns:

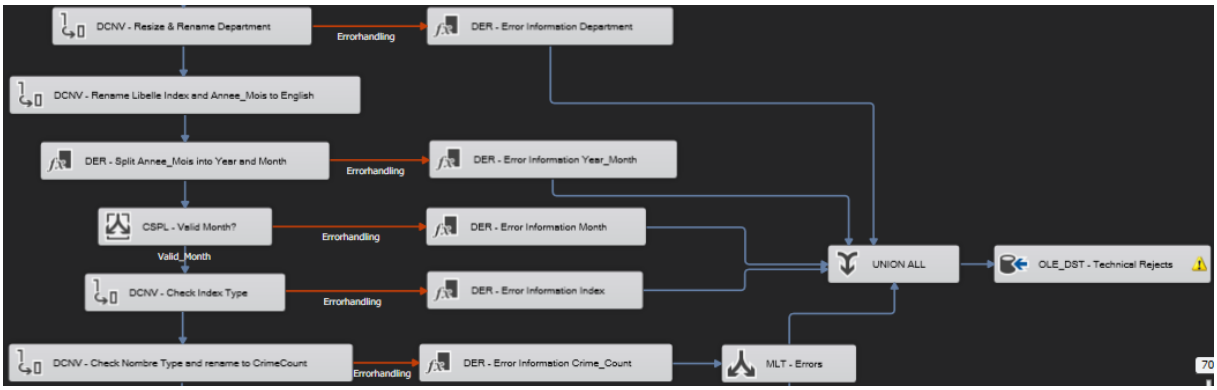


For this, we need to define wich collumns will be expanded. In our case, we need to expand the dates and move the data to a new column “Nombre”. We also rename the date column “Annee_Mois”.

Then, we clean the “Department” column that has still the format of the variable. We only keep the two characters of the department (because of “2A” and “2B”, we choose characters):

Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
Departement	Replace 'Departement'	LEFT(RIGHT(Departement,3),2)	chaîne Unicode [DT_WSTR]	255			

In the second segment, we are cleaning and enriching the data.



We have multiple checks and the errors are redirected to the “Technical_Rejects” Table created as follow:

```
USE [STA]
GO

/****** Object:  Table [dbo].[Technical Rejects]    Script Date: 11.11.2021 09:11:32 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Technical_Rejects](
    [Error_date] [datetime] NULL,
    [Error_Column] [nvarchar](255) NULL,
    [Error_Message] [nvarchar](255) NULL,
    [Error_Source] [nvarchar](255) NULL,
    [Rejected_Row] [int] NULL
) ON [PRIMARY]
GO
```

First, we rename and resize the “Departement” column. To keep track of the changes and distinguish the columns, we add a suffix with the change in length or type.

Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
Departement	Department_10	chaîne Unicode [DT_W...	10			

If that step fails, we track the error and put it in the “Technical_Rejects” table.

Derived Column Name	Derived Column	Expression	Data Type	Length
Error_date	<add as new column>	GETDATE()	horodateur base de données [DT_DBTIMESTAMP]	
Error_Column	<add as new column>	"Department"	chaîne Unicode [DT_WSTR]	10
Error_Message	<add as new column>	"The Department" + Departement + "is not a valid Department"	chaîne Unicode [DT_WSTR]	294
Error_Source	<add as new column>	@[System::PackageName] + "|" + @[System::TaskName]	chaîne Unicode [DT_WSTR]	42
Rejected_Row	<add as new column>	1	entier signé (4 bits) [DT_I4]	

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

- The date of the error.
- The column making the error.
- An error message.
- The package and the task causing the error.
- A rejection code: 1 for reject; 0 for a warning.

Next, we rename and resize the columns

Input Column	Output Alias	Data Type	Length
LibelleIndex	Index_Description	chaîne Unicode [DT_WSTR]	255
Annee_Mois	Year_Month_10	chaîne Unicode [DT_WSTR]	10

Then, we enrich the data by splitting the column “Year_Month_10” into “Year” and “Month” columns. We explain this choice in the Data Warehouse section.

Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
Year	<add as new column>	(DT_I4)LEFT(Year_Month_10,4)	entier signé (4 bits) [DT_I4]				
Month	<add as new column>	(DT_I4)RIGHT(Year_Month_10...	entier signé (4 bits) [DT_I4]				

In case of an error, we track it like before:

Derived Column Name	Derived Column	Expression	Data Type	Length
Error_date	<add as new column>	GETDATE()	horodateur base de données [DT_DBTIMESTAMP]	
Error_Column	<add as new column>	"Month"	chaîne Unicode [DT_WSTR]	5
Error_Message	<add as new column>	"The Month" + Annee_Mois + "is not a valid Month"	chaîne Unicode [DT_WSTR]	284
Error_Source	<add as new column>	@[System::PackageName] + "|" + @[System::TaskName]	chaîne Unicode [DT_WSTR]	42
Rejected_Row	<add as new column>	1	entier signé (4 bits) [DT_I4]	

We also do an additional check on the months to make sure that the value makes sense (range 0-12):

Order	Output Name	Condition
1	Valid_Month	Month >= 1 && Month <= 12

In case of an error:

Derived Column Name	Derived Column	Expression	Data Type	Length
Error_date	<add as new column>	GETDATE()	horodateur base de données [DT_DBTIMESTAMP]	
Error_Column	<add as new column>	"Month"	chaîne Unicode [DT_WSTR]	5
Error_Message	<add as new column>	"The Month" + Year_Month_10 + "is not a valid Month"	chaîne Unicode [DT_WSTR]	39
Error_Source	<add as new column>	@[System::PackageName] + "|" + @[System::TaskName]	chaîne Unicode [DT_WSTR]	42
Rejected_Row	<add as new column>	1	entier signé (4 bits) [DT_I4]	

Next, we resize the “IndexCrime” column:

Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
IndexCrime	IndexCrime_10	chaîne Unicode [DT_WSTR]	10			

In case of an error:

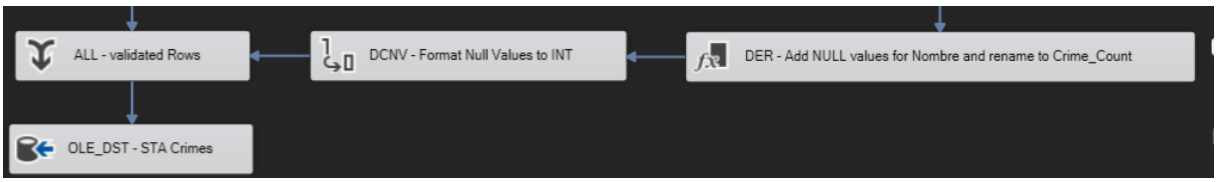
Derived Column Name	Derived Column	Expression	Data Type	Length	P
Error_date	<add as new column>	GETDATE()	horodateur base de données [DT_DBTIMESTAMP]		
Error_Column	<add as new column>	"IndexCrime"	chaîne Unicode [DT_WSTR]	10	
Error_Message	<add as new column>	"The Index" + IndexCrime + "is not a valid INTEGER"	chaîne Unicode [DT_WSTR]	286	
Error_Source	<add as new column>	@[System::PackageName] + "|" + @[System::TaskName]	chaîne Unicode [DT_WSTR]	42	
Rejected_Row	<add as new column>	1	entier signé (4 bits) [DT_I4]		

Finally, we rename and resize the “Nombre” column containing the numerical data about the crimes. The last check is of Warning level, so we integrate it in the database with a null value. We choose to do this to not lose the record as it could generate errors in the data integration (see DWH_Crimes) and we can still manage it in the queries.

Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
Nombre	CrimeCount_INT	entier signé (4 bits) [D...				

Derived Column Name	Derived Column	Expression	Data Type	Length
Error_date	<add as new column>	GETDATE()	horodateur base de données [DT_DBTIMESTAMP]	
Error_Column	<add as new column>	"Nombre"	chaîne Unicode [DT_WSTR]	6
Error_Message	<add as new column>	"The Nombre" + Nombre + "is not a valid INTEGER"	chaîne Unicode [DT_WSTR]	287
Error_Source	<add as new column>	@[System::PackageName] + "|" + @[System::TaskName]	chaîne Unicode [DT_WSTR]	42
Rejected_Row	<add as new column>	0	entier signé (4 bits) [DT_I4]	

In the last segment, we want to insert a null value. So, we first generate the “NULL” then we convert it to the destination type.



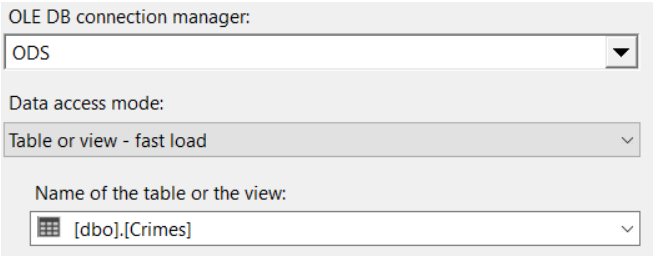
Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
CrimeCount_NULL	<add as new column>	NULL(DT_I4)	entier signé (4 bits) [DT_I4]				

Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
CrimeCount_NULL	CrimeCount_NULL_INT	entier signé (4 bits) [DT_I4]				

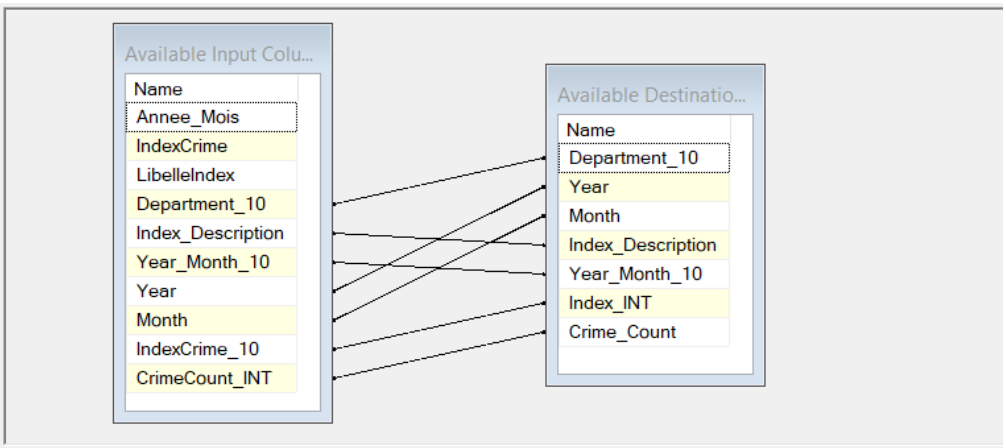
Now we can create the ODS “Crimes” table as follow:

```
USE [ODS]
GO

/****** Object:  Table [dbo].[Crimes]    Script Date: 11.11.2021 09:08:03 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Crimes](
    [Department_10] [nvarchar](10) NULL,
    [Year_Month_10] [nvarchar](10) NULL,
    [Year] [int] NULL,
    [Month] [int] NULL,
    [IndexCrime_10] [int] NULL,
    [Index_Description] [nvarchar](255) NULL,
    [CrimeCount_INT] [int] NULL
) ON [PRIMARY]
GO
```

The last task is to insert the data in the ODS database:



The final mapping is:



Input Column	Destination Column
Department_10	Department_10
Year	Year
Month	Month
Index_Description	Index_Description
Year_Month_10	Year_Month_10
IndexCrime_10	Index_INT
CrimeCount_INT	Crime_Count

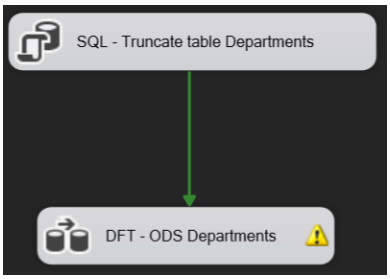
Here is the first ten lines of the results:

Results | Messages

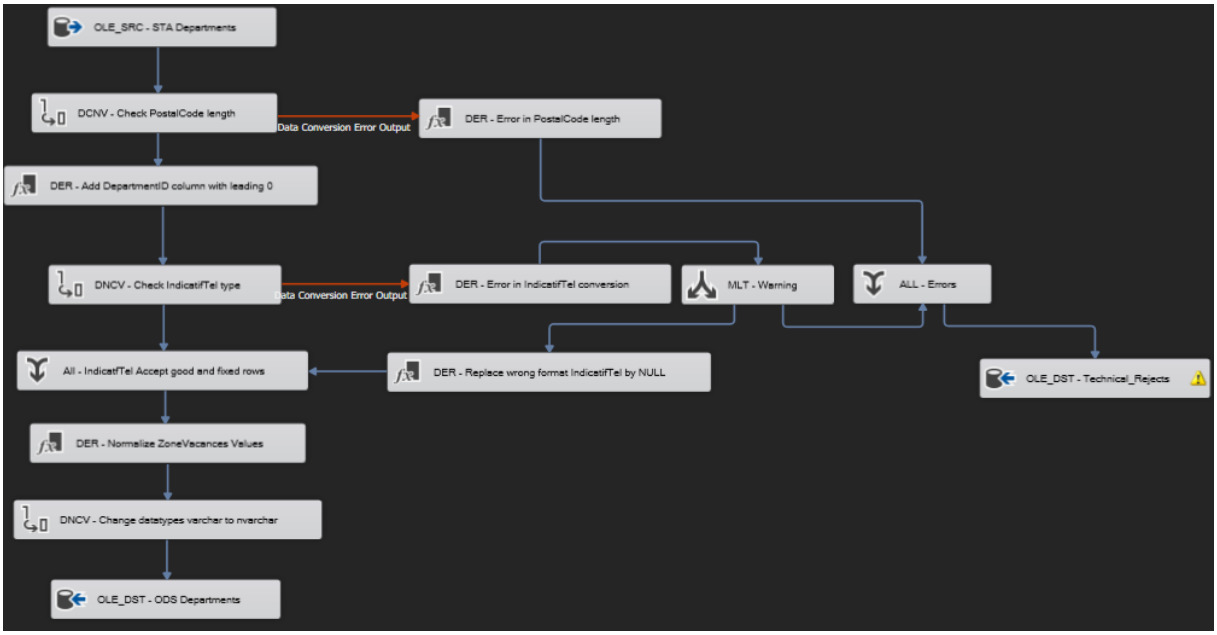
	Department_10	Year_Month_10	Year	Month	Index_INT	Index_Description	Crime_Count
1	75	2009_04	2009	4	40	Vols simples sur exploitations agricoles	0
2	75	2009_05	2009	5	40	Vols simples sur exploitations agricoles	0
3	75	2009_06	2009	6	40	Vols simples sur exploitations agricoles	0
4	75	2009_07	2009	7	40	Vols simples sur exploitations agricoles	0
5	75	2009_08	2009	8	40	Vols simples sur exploitations agricoles	0
6	75	2009_09	2009	9	40	Vols simples sur exploitations agricoles	0
7	75	2009_10	2009	10	40	Vols simples sur exploitations agricoles	0
8	75	2009_11	2009	11	40	Vols simples sur exploitations agricoles	0
9	75	2009_12	2009	12	40	Vols simples sur exploitations agricoles	0
10	75	2010_01	2010	1	40	Vols simples sur exploitations agricoles	0

Departments Table

Now we need to process the data from the “Departments” table. We also have a warning for the same reason as before.



We process here is similar, we process the data, collect the errors, and reintegrate some of the data into the table with null values.



We import the data from the staging area, then the first processing step is to clean some of the data.

First, we resize the reference of the department at two characters and handle the associated errors.

Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
CodePostal	CodePostal_VAR2	chaîne [DT_STR]	2			1252 (ANSI - latin I)

Derived Column Name	Derived Column	Expression	Data Type	Length
Error_Date	<add as new column>	GETDATE()	horodateur base de données [DT_DBTIMESTAMP]	
Error_Column	<add as new column>	"PostalCode"	chaîne Unicode [DT_WSTR]	10
Error_Message	<add as new column>	"The PostalCode " + CodePostal + " have a wrong format"	chaîne Unicode [DT_WSTR]	290
Error_Source	<add as new column>	@[System::PackageName] + "/" + @[System::TaskName]	chaîne Unicode [DT_WSTR]	37
Rejected	<add as new column>	1	entier signé (4 bits) [DT_I4]	

Next, to have a valid zip code, we need to add a leading zero.

Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
DepartmentID_NVAR2	<add as new column>	UPPER(RIGHT(REPLICATE("0",2) + CodePostal_VAR2,2))	chaîne Unicode [DT_...	2			

Then we process the phone number data.

Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
IndicatifTel	IndicatifTel_NUM1	numérique [DT_NUMERIC]		1	0	

Derived Column Name	Derived Column	Expression	Data Type	Length
Error_Date	<add as new column>	GETDATE()	horodateur base de données [DT_DBTIMESTAMP]	
Error_Column	<add as new column>	"IndicatifTel"	chaîne Unicode [DT_WSTR]	12
Error_Message	<add as new column>	"The IndicatifTel " + IndicatifTel + " is not a valis Integer"	chaîne Unicode [DT_WSTR]	295
Error_Source	<add as new column>	@[System::PackageName] + "/" + @[System::TaskName]	chaîne Unicode [DT_WSTR]	37
Rejected	<add as new column>	0	entier signé (4 bits) [DT_I4]	

Here, we consider that the leading “0” can be replaced with “+33”. So, the real data is one number. In addition, if we have an error, we replace it with a null value to not lost the entire department.

Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
IndicatifTel_NUM1_N...	<add as new column>	NULL(DT_NUMERIC,1,0)	numérique [DT_NUMERIC]		1	0	

Then we normalize the holiday areas as one upper character (i.e., “S” instead of “Special”).

Derived Column Name	Derived Column	Expression	Data Type	Length
ZoneVacance_VAR1	<add as new column>	(LEN(ZoneVacances) > 1) ? UPPER(LEFT(ZoneVacances,1)) : UPPER(ZoneVacances)	chaîne Unicode [DT_WSTR]	255

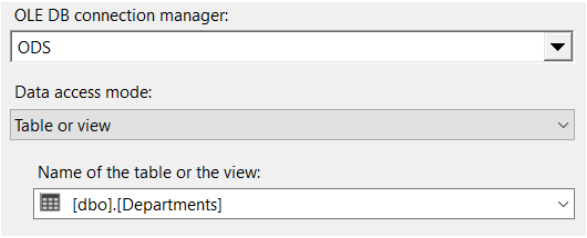
The final cleaning step is to make sure that all the text data is of nvarchar type.

Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
Departement	Departement_NVAR100	chaîne Unicode [DT_WSTR]	100			
Region	Region_NVAR100	chaîne Unicode [DT_WSTR]	100			
ZoneVacance_VAR1	ZoneVacance_NVAR1	chaîne Unicode [DT_WSTR]	1			
CodePostal_VAR2	CodePostal_NVAR2	chaîne Unicode [DT_WSTR]	2			

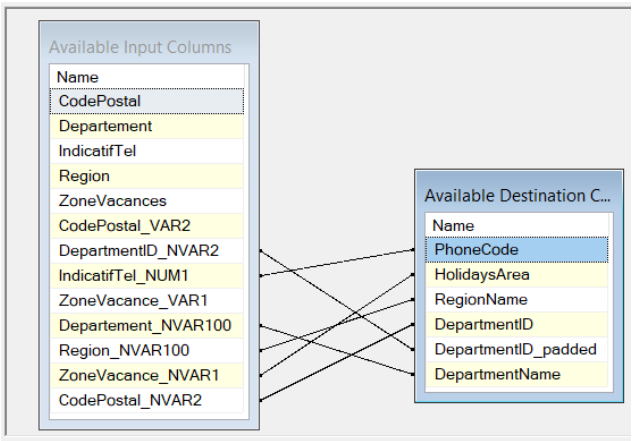
Finally, we can create the ODS “Departments” table with the following script:

```
USE [ODS]
GO

/****** Object:  Table [dbo].[Departments]    Script Date: 11/12/2021 10:09:13 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Departments](
    [DepartmentID] [nvarchar](2) NULL,
    [DepartmentID_padded] [nvarchar](2) NULL,
    [DepartmentName] [nvarchar](100) NULL,
    [PhoneCode] [numeric](1, 0) NULL,
    [RegionName] [nvarchar](100) NULL,
    [HolidaysArea] [nvarchar](1) NULL
) ON [PRIMARY]
GO
```

Then, we load the data into the table.



The final mapping is as follow:



Input Column	Destination Column
IndicatifTel_NUM1	PhoneCode
ZoneVacance_NVAR1	HolidaysArea
Region_NVAR100	RegionName
CodePostal_NVAR2	DepartmentID
DepartmentID_NVAR2	DepartmentID_padded
Departement_NVAR100	DepartmentName

Here is the first ten lines of the results:

Results Messages

	DepartmentID	DepartmentID_padded	DepartmentName	PhoneCode	RegionName	HolidaysArea
1	1	01	Ain	4	Auvergne-Rhône-Alpes	A
2	2	02	Aisne	3	Hauts-de-France	B
3	3	03	Allier	4	Auvergne-Rhône-Alpes	A
4	4	04	Alpes-de-Haute-Provence	4	Provence-Alpes-Côte d'Azur	B
5	5	05	Hautes-Alpes	4	Provence-Alpes-Côte d'Azur	B
6	6	06	Alpes-Maritimes	4	Provence-Alpes-Côte d'Azur	B
7	7	07	Ardèche	4	Auvergne-Rhône-Alpes	A
8	8	08	Ardennes	3	Grand-Est	B
9	9	09	Ariège	5	Occitanie	C
10	10	10	Aube	3	Grand-Est	B

All the errors are sent to the same “Technical_Rejects” table.

Datawarehouse

The last step in the data pipeline is to integrate the data into the Data WareHouse. One common database schema 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.

Database design

For our data, we can consider that the important data are the statistics about the crimes. Therefore, we choose to build the fact table with the “Crimes” table.

For the dimensions, one common dimension is the “Date” dimension. It allows to describe the dates with multiple temporal aggregate categories (year, month, quarter). The table will contain multiple variation 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”.

The second dimension we choose is the Department dimension. It will help to describe geographical data. We choose to use an incremental indices technical key for the relation to the fact table. There is also a business key with a two-character zip code of the department.

We could consider an additional dimension describing the crimes in more details. However, with the current available data, we choose to keep everything in the fact table.

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 an TSQL script to make it.

First, we create an empty table:

```
USE DWH
GO

IF ( EXISTS (SELECT *
                 FROM INFORMATION_SCHEMA.TABLES
                 WHERE TABLE_SCHEMA = 'dbo'
                 AND  TABLE_NAME = 'DimMonth'
                 and TABLE_TYPE = 'BASE TABLE'))
    BEGIN
            DROP TABLE dbo.DimMonth
      END
GO

CREATE TABLE dbo.DimMonth (
   [MonthKey] INT NOT NULL PRIMARY KEY,
   [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,
   [YYYY_MM] NVARCHAR(10) NOT NULL,
   [MonthYear] CHAR(7) NOT NULL
   )
GO
```

The available data with different formats are:

- MonthKey, the technical key.
- Month (1,2,3…)
- MonthName (January, February, March…)
- MonthName_Short (JAN, FEB, MAR…)
- MonthName_FirstLetter (J, F, M…)
- Quarter (1,2…)
- QuarterName (first, second…)
- Year (1996, 1997…)
- MMYYYY (011996, 021996…)
- YYYY_MM (1996_01, 1996_02)
- MonthYear (1996JAN, 1996FEB…)

To build that data we use the following script:

```
SET NOCOUNT ON


DECLARE @CurrentDate DATE = '1996-01-01'
DECLARE @EndDate DATE = '2016-12-31'

WHILE @CurrentDate < @EndDate
BEGIN
     INSERT INTO [dbo].[DimMonth] (
      [MonthKey],
      [Month],
      [MonthName],
      [MonthName_Short],
      [MonthName_FirstLetter],
      [Quarter],
      [QuarterName],
      [Year],
      [MMYYYY],
    [YYYY_MM],
      [MonthYear]
      )
    Select
          MonthKey = YEAR(@CurrentDate) * 100 + MONTH(@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)),
          [YYYY_MM] = concat ( YEAR(@CurrentDate) , '_',FORMAT(@CurrentDate,'MM')),
          [MonthYear] = CAST(YEAR(@CurrentDate) AS VARCHAR(4)) + UPPER(LEFT(DATENAME(mm, @CurrentDate), 3))

      SET @CurrentDate = DATEADD(MM, 1, @CurrentDate)

END
```

Here is the first ten lines of the results:

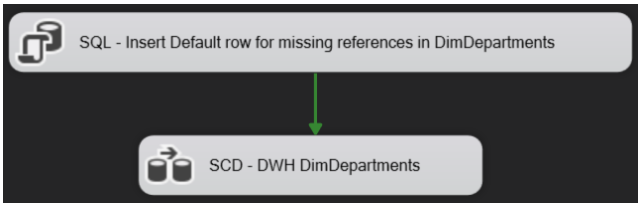
Results | Messages

	MonthKey	Month	MonthName	MonthName_Short	MonthName_FirstLetter	Quarter	QuarterName	Year	MMYYYY	YYYY_MM	MonthYear
1	199601	1	January	JAN	J	1	first	1996	011996	1996_01	1996JAN
2	199602	2	February	FEB	F	1	first	1996	021996	1996_02	1996FEB
3	199603	3	March	MAR	M	1	first	1996	031996	1996_03	1996MAR
4	199604	4	April	APR	A	2	second	1996	041996	1996_04	1996APR
5	199605	5	May	MAY	M	2	second	1996	051996	1996_05	1996MAY
6	199606	6	June	JUN	J	2	second	1996	061996	1996_06	1996JUN
7	199607	7	July	JUL	J	3	third	1996	071996	1996_07	1996JUL
8	199608	8	August	AUG	A	3	third	1996	081996	1996_08	1996AUG
9	199609	9	September	SEP	S	3	third	1996	091996	1996_09	1996SEP
10	199610	10	October	OCT	O	4	fourth	1996	101996	1996_10	1996OCT

To accept new data in the data warehouse, this dimension can be updated before the beginning of a new year. We would need to change the “@EndDate” variable.

Integration of the Departments dimension

In ODS we have clean data about the departments. However, there is the possibility that an error in the fact table could prevent to make a link to the fact table. To handle this, we add a default (-1) value for the department key.



First, we need to create the table.

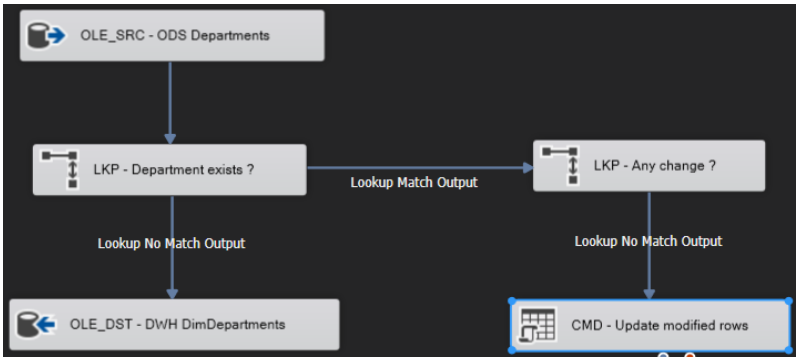
```
USE [ODS]
GO

/****** Object:  Table [dbo].[Departments]    Script Date: 11/12/2021 10:09:13 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Departments](
    [DepartmentID] [nvarchar](2) NULL,
    [DepartmentID_padded] [nvarchar](2) NULL,
    [DepartmentName] [nvarchar](100) NULL,
    [PhoneCode] [numeric](1, 0) NULL,
    [RegionName] [nvarchar](100) NULL,
    [HolidaysArea] [nvarchar](1) NULL
) ON [PRIMARY]
GO
```

We decide to create a row that will be a reference for any unknown department found in other data. It is created at pre-sql step of Department Dimension.
We decide to give it a special DepartmentKey -1.

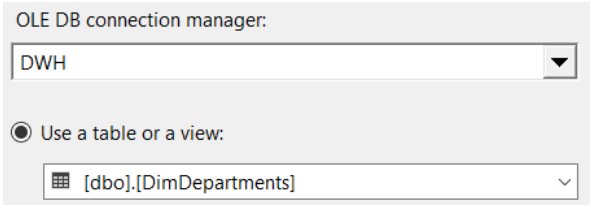
Script testing if this DepartmentKey exist and create it if not found:

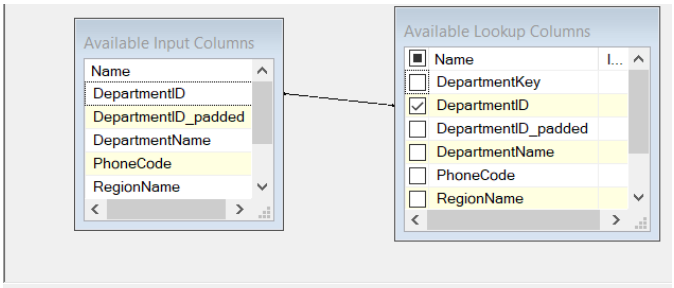
```
SET IDENTITY_INSERT dbo.DimDepartments  ON
GO

IF NOT EXISTS (SELECT * FROM dbo.DimDepartments WHERE DepartmentKey = -1)
BEGIN
    INSERT INTO dbo.DimDepartments  (DepartmentKey, DepartmentID,DepartmentID_padded,DepartmentName,PhoneCode,RegionName,HolidaysArea)
    VALUES (-1, Null, Null, 'Unknown Department',Null,'Unknown Region',Null)
END
go

SET IDENTITY_INSERT dbo.DimDepartments OFF
GO
```

Then we can load the data with the process bellow:



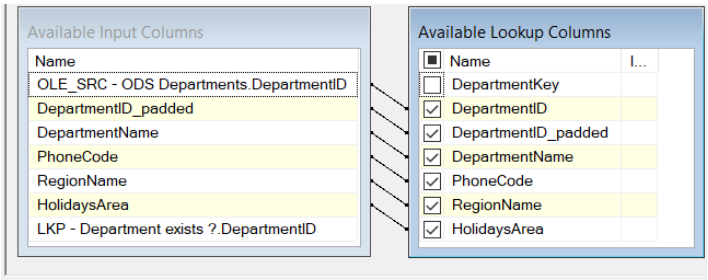
We need to make sure that we can make joins between the fact table and the dimension table, so we check the link with “DepartmentID”.





Lookup Column	Lookup Operation	Output Alias
DepartmentID	<add as new column>	DepartmentID

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.



Lookup Column	Lookup Operation	Output Alias
DepartmentID	<add as new column>	DepartmentID
DepartmentID_padded	<add as new column>	DepartmentID_padded
DepartmentName	<add as new column>	DepartmentName
PhoneCode	<add as new column>	PhoneCode
RegionName	<add as new column>	RegionName
HolidaysArea	<add as new column>	HolidaysArea

Here is the first ten lines of the results:

Results | Messages

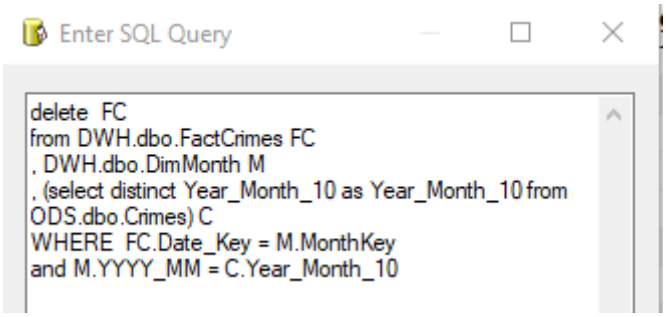
	DepartmentKey	DepartmentID	DepartmentID_padded	DepartmentName	PhoneCode	RegionName	HolidaysArea
1	-1	NULL	NULL	Unknown Department	NULL	Unknown Region	NULL
2	1	1	01	Ain	4	Auvergne-Rhône-Alpes	A
3	2	2	02	Aisne	3	Hauts-de-France	B
4	3	3	03	Allier	4	Auvergne-Rhône-Alpes	A
5	4	4	04	Alpes-de-Haute-Provence	4	Provence-Alpes-Côte d'Azur	B
6	5	5	05	Hautes-Alpes	4	Provence-Alpes-Côte d'Azur	B
7	6	6	06	Alpes-Maritimes	4	Provence-Alpes-Côte d'Azur	B
8	7	7	07	Ardèche	4	Auvergne-Rhône-Alpes	A
9	8	8	08	Ardennes	3	Grand-Est	B
10	9	9	09	Ariège	5	Occitanie	C

Integration of the Crimes Facts table

Now that we finally have our dimensions tables, we can build our fact table while checking valid relations with the dimensions.

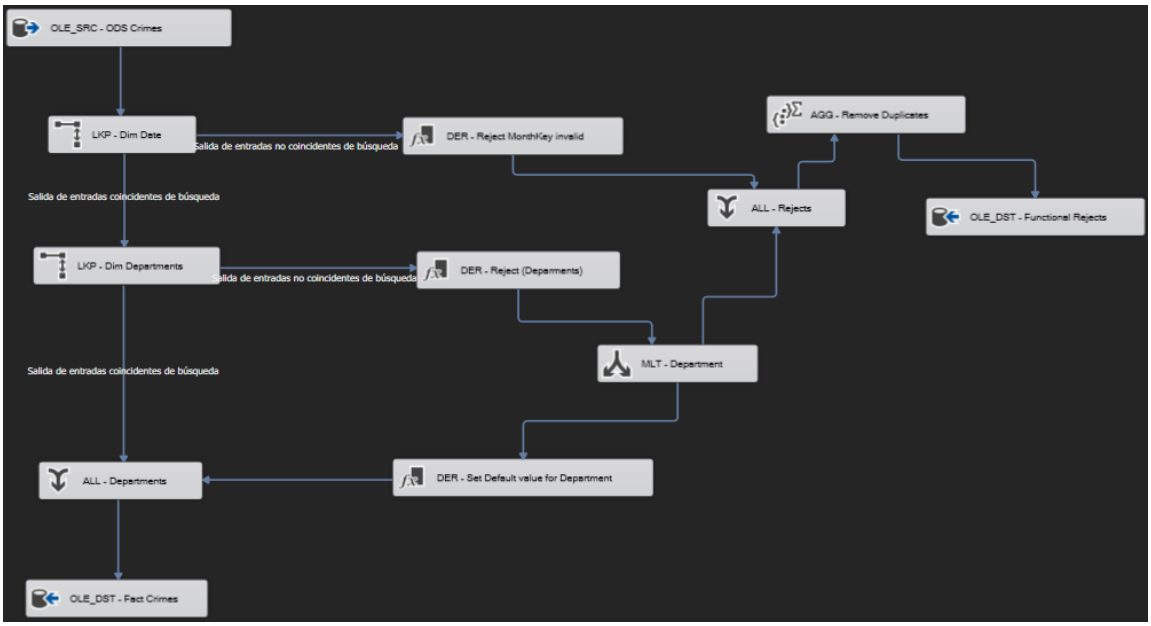
Delete strategy:

As this is a fact table it is supposed to cumulate historical data with new data coming based on a regular basis. The original spreadsheet is organized by departments and by months, probably the data ingestion is made on a monthly basis. But the spreadsheet presents much more than a month of data.
We choose to consider that data about a month X coming in the spreadsheet can be either new data or updated data. So, a month Y not present in the spreadsheet have to be consider as stable in the fact table.
What about the departments? can we consider that if a department is missing in next spreadsheet, we have to keep the data in Fact table about it?
We would have answer Yes but it requires to be able to identify the department to be able to delete only those coming from the new spreadsheet. But we don't keep the original Crimes department information in the fact table, we replace it by a surrogate key to "DimDepartment" table.
We choose to delete in "FactCrimes" table the data based on "Year_Month" present in ODS Crimes table.



This strategy will ensure not to duplicate data coming this month with data coming next month.

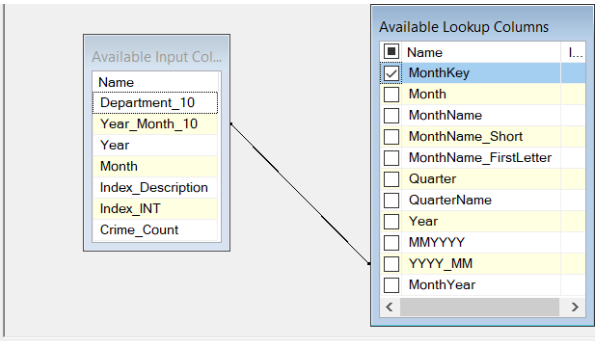
Once we made sure that we have a clean table to insert the data into the data warehouse, we verify and load the data as follow:



In case of an error in the processing, we generate a functional reject and insert it into a table “Functional_Rejects” created by the following script:

```
USE [STA]
GO

/****** Object:  Table [dbo].[Functional_Rejects]    Script Date: 14/11/2021 16:39:31 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Functional_Rejects](
    [Error_Date] [datetime] NULL,
    [Error_Column] [nvarchar](255) NULL,
    [Error_Message] [nvarchar](1000) NULL,
    [Error_Source] [nvarchar](255) NULL,
    [Rejected_Row] [int] NULL,
    [Nb_Rows] [int] NULL
) ON [PRIMARY]
GO
```

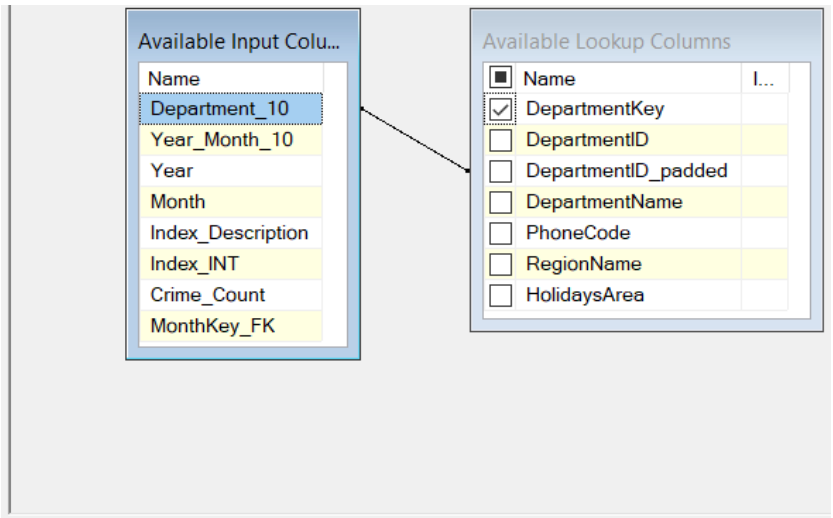
The first dimension we check is “DimMonth”. We check it with the business key between “Year_Month_10” and “YYYY_MM”. We also add a technical key “MonthKey_FK”.



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

Derived Column Name	Derived Column	Expression	Data Type	Length
Error_Date	<add as new column>	GETDATE()	horodateur base de données [DT_DBTIMESTAMP]	
Error_Column	<add as new column>	"Year_Month_10"	chaîne Unicode [DT_WSTR]	13
Error_Message	<add as new column>	"The date " + Year_Month_10 + " cannot be found in DimDate"	chaîne Unicode [DT_WSTR]	46
Error_Source	<add as new column>	@[System::PackageName] + "	" + @[System::TaskName]	chaîne Unicode [DT_WSTR]	45
Reject_Row	<add as new column>	1	entier signé (4 bits) [DT_I4]	

Similarly, we check the relation with “DimDepartment” with the business key “Department_10” and “DepartmentID_padded”. We also add the technical key “DepartmentKey”.



Lookup Column	Lookup Operation	Output Alias
DepartmentKey	<add as new column>	DepartementKey

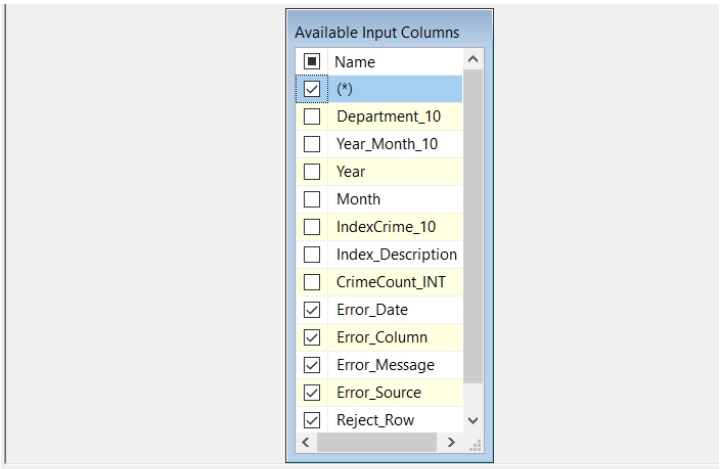
In case of an error, we have the following:

Derived Column Name	Derived Column	Expression	Data Type	Length
Error_Date	<add as new column>	GETDATE()	horodateur base de données [DT_DBTIMESTAMP]	
Error_Column	<add as new column>	"Department_10"	chaîne Unicode [DT_WSTR]	13
Error_Message	<add as new column>	"The Department " + Department_10 + " cannot be found in DimDepartments"	chaîne Unicode [DT_WSTR]	59
Error_Source	<add as new column>	@[System::PackageName] + "|" + @[System::TaskName]	chaîne Unicode [DT_WSTR]	43
Reject_Row	<add as new column>	0	entier signé (4 bits) [DT_I4]	

In this case, we will integrate the record with the default value (-1) for the Department.

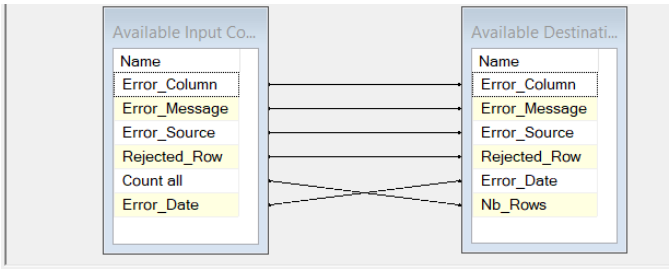
Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
DepartmentKey_Default	<add as new column>	-1	entier signé (4 bits) [DT_I4]				

Since we have a lot of records, a missing department can generate a lot of data. To alleviate this problem, we check for duplicates during insertions in “Functional_Rejects”.



Input Column	Output Alias	Operation
Error_Column	Error_Column	GROUP BY
Error_Message	Error_Message	GROUP BY
Error_Source	Error_Source	GROUP BY
Reject_Row	Rejected_Row	GROUP BY
(*)	Count all	COUNT ALL
Error_Date	Error_Date	GROUP BY

As shown here, we are keeping track of the number of duplicates:



Input Column	Destination Column
Error_Column	Error_Column
Error_Message	Error_Message
Error_Source	Error_Source
Rejected_Row	Rejected_Row
Error_Date	Error_Date
Count all	Nb_Rows

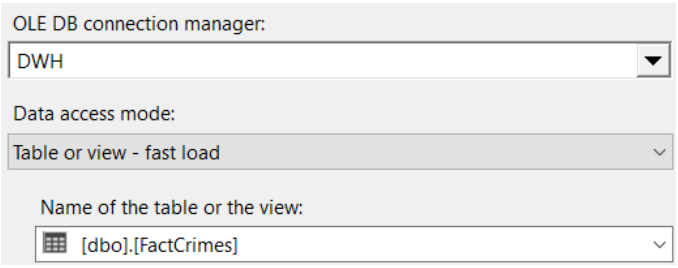
Here is the first ten lines of the results of “Functional_Rejects”:

Results Messages

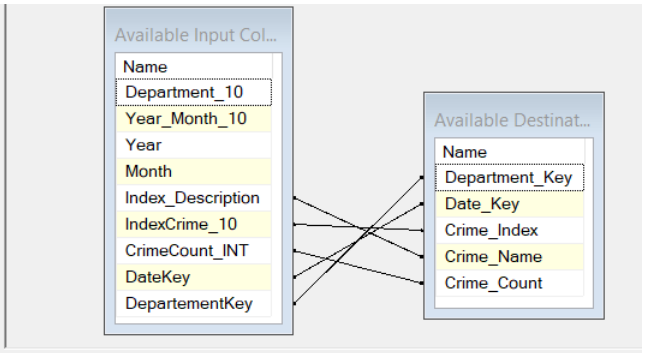
	Error_Date	Error_Column	Error_Message	Error_Source	Rejected_Row	Nb_Rows
1	2021-11-19 15:20:02.983	Department_10	The Department 87 cannot be found in DimDepartme...	Package1|DFT - Loading into DWH Fact Crimes	0	21
2	2021-11-19 15:20:02.980	Department_10	The Department 87 cannot be found in DimDepartme...	Package1|DFT - Loading into DWH Fact Crimes	0	357
3	2021-11-19 15:20:02.980	Department_10	The Department 87 cannot be found in DimDepartme...	Package1|DFT - Loading into DWH Fact Crimes	0	211
4	2021-11-19 15:20:02.977	Department_10	The Department 87 cannot be found in DimDepartme...	Package1|DFT - Loading into DWH Fact Crimes	0	476
5	2021-11-19 15:20:02.977	Department_10	The Department 87 cannot be found in DimDepartme...	Package1|DFT - Loading into DWH Fact Crimes	0	284
6	2021-11-19 15:20:02.977	Department_10	The Department 87 cannot be found in DimDepartme...	Package1|DFT - Loading into DWH Fact Crimes	0	306
7	2021-11-19 15:20:02.803	Department_10	The Department 87 cannot be found in DimDepartme...	Package1|DFT - Loading into DWH Fact Crimes	0	155
8	2021-11-19 15:20:02.803	Department_10	The Department 87 cannot be found in DimDepartme...	Package1|DFT - Loading into DWH Fact Crimes	0	564
9	2021-11-19 15:20:02.800	Department_10	The Department 87 cannot be found in DimDepartme...	Package1|DFT - Loading into DWH Fact Crimes	0	703
10	2021-11-19 15:20:02.800	Department_10	The Department 87 cannot be found in DimDepartme...	Package1|DFT - Loading into DWH Fact Crimes	0	654

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

```
USE [DWH]
GO

/****** Object:  Table [dbo].[FactCrimes]    Script Date: 14/11/2021 16:40:31 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[FactCrimes](
    [Department_Key] [int] NULL,
    [Date_Key] [int] NULL,
    [Crime_Index] [int] NULL,
    [Crime_Name] [nvarchar](255) NULL,
    [Crime_Count] [int] NULL
) ON [PRIMARY]
GO
```



During the mapping we do a final change to the names.



Input Column	Destination Column
DepartementKey	Department_Key
DateKey	Date_Key
IndexCrime_10	Crime_Index
Index_Description	Crime_Name
CrimeCount_INT	Crime_Count

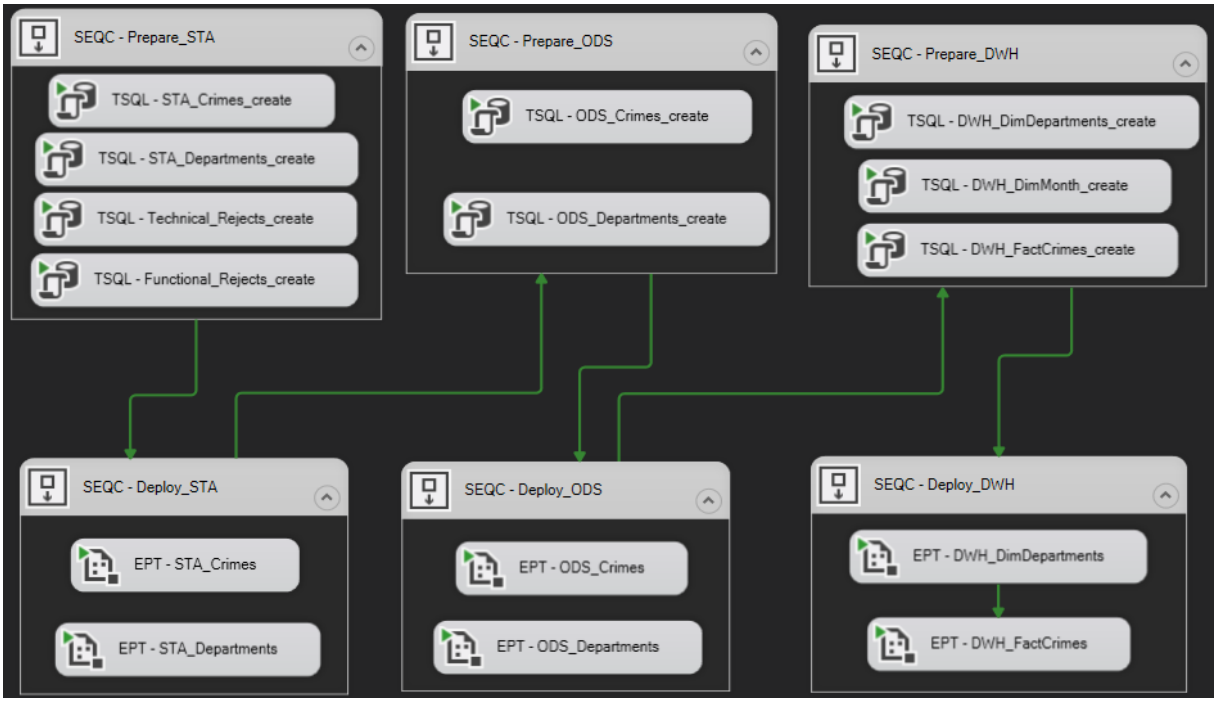
Here is the first ten lines of the results of the facts table “FactCrimes”:

Results | Messages

	Department_Key	Date_Key	Crime_Index	Crime_Name	Crime_Count
1	44	199603	69	Infractions aux conditions générales d'entrée et...	9
2	44	199604	69	Infractions aux conditions générales d'entrée et...	1
3	44	199605	69	Infractions aux conditions générales d'entrée et...	6
4	44	199606	69	Infractions aux conditions générales d'entrée et...	10
5	44	199607	69	Infractions aux conditions générales d'entrée et...	4
6	44	199608	69	Infractions aux conditions générales d'entrée et...	6
7	44	199609	69	Infractions aux conditions générales d'entrée et...	3
8	44	199610	69	Infractions aux conditions générales d'entrée et...	8
9	44	199611	69	Infractions aux conditions générales d'entrée et...	4
10	44	199612	69	Infractions aux conditions générales d'entrée et...	12

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 “ETL_Pipeline” with the DAG of the tasks.



Using this package allow to use one interface for the entire pipeline. The creation of the tables is done with TSQL tasks and the package execution tasks launches the data flows.

Use Case

Once we have deployed the data warehouse, we can start to query it for making analysis reports. For instance, we can generate a view that can then be used in a BI tool to report on the geographical and temporal distribution of crimes.

The following view get geographical data from “DimDepartment” and temporal data from “DimMonth”.

```
CREATE VIEW V_All_Crimes
as SELECT dbo.DimDepartments.DepartmentName
, dbo.DimDepartments.RegionName
, dbo.DimMonth.Month
, dbo.DimMonth.MonthName
, dbo.FactCrimes.Crime_Index
, dbo.FactCrimes.Crime_Name
, SUM(dbo.FactCrimes.Crime_Count) AS Total
FROM dbo.DimDepartments
INNER JOIN dbo.FactCrimes
ON dbo.DimDepartments.DepartmentKey = dbo.FactCrimes.Department_Key
INNER JOIN dbo.DimMonth
ON dbo.FactCrimes.Date_Key = dbo.DimMonth.MonthKey
GROUP BY dbo.DimDepartments.DepartmentName
, dbo.DimDepartments.RegionName
, dbo.DimMonth.Month
, dbo.DimMonth.MonthName
, dbo.FactCrimes.Crime_Index
, dbo.FactCrimes.Crime_Name
```

Then, we aggregate by month and region to analyse if there is any “seasonal” pattern and variation across regions. Here we could use numerical and literal labels for months and crimes categories.

Here is the first ten lines of the results of the facts view “V_All_Crimes”:

Results | Messages

	DepartmentName	RegionName	Month	MonthName	Crime_Index	Crime_Name	Total
1	Hautes-Pyrénées	Occitanie	3	March	48	Harcèlements sexuels et autres agressions sexuel...	13
2	Territoire de Belfort	Bourgogne-Franche-Comté	2	February	31	Vols avec entrée par ruse en tous lieux	15
3	Tarn	Occitanie	7	July	91	Escroqueries et abus de confiance	1136
4	Somme	Hauts-de-France	10	October	44	Recels	551
5	Hautes-Alpes	Provence-Alpes-Côte d'Azur	12	December	4	Tentatives d'homicides pour voler et à l'occasion d...	0
6	Charente	Nouvelle-Aquitaine	6	June	6	Coups et blessures volontaires suivis de mort	0
7	Lot	Occitanie	12	December	67	Autres destructions er dégradations de biens privés	182
8	Territoire de Belfort	Bourgogne-Franche-Comté	11	November	35	Vols d'automobiles	291
9	Cantal	Auvergne-Rhône-Alpes	5	May	93	Travail clandestin	53
10	Cantal	Auvergne-Rhône-Alpes	2	February	96	Index non utilisé	0

Conclusion

We made some choices:

Months

This table is a regular time dimension. Here, we limit the grain to month because the fact data is at this granularity. All possible values are known, so it's not necessary to create a reference for unknown month.

Department

This table is Dimension data. It is almost stable. Therefore, we consider that it is not necessary to keep any old version of data
We can't truncate the whole "DimDepartment" table because it is reference for other tables. So, it is necessary to merge existing data with new one to accept changes.
It is also necessary to create a special key for unknown Department found in Crimes.

Crimes

This table is the fact table. Our design choice was to replace the value for month and year by the computed key "YYYYMM" of "DimMonths". The Department reference is replaced by the surrogate "DepartmentKey" found in "DimDepartment", with a default value for missing references.

This data model we can answer questions about Crimes using the two dimensions: Time and Department. By using different levels of aggregations, we can extract data per quarter or year and per region. Views are a good tool to prepare data for BI tools.

Future expansions

In the future, it is possible that crimes will be classified by consequences like "prison" and/or "fine", or regrouped in categories "aggressions", "vols", …
Having more information about crime types will drive to create a new dimension "DimCrimesTypes". Then, we would need to change the "FactCrimes" table structure to reflect this new dimension.

If we want to create it today, we would create a new package using ODS "Crimes" table. This package will extract crimes index and descriptions and make a set of distinct crimes to form it as a dimension. This would require a lot of checks to unify it correctly. The merge of data coming from ODS "Crimes" with a "DimCrimesType" will be a source of issues too.

Another expansion can be a fact table of contacts by department. The current design will adapt easily to this kind of evolution.