

Caso práctico: Almacén de datos para el análisis de indicadores de crisis en la eurozona

Solución PRA2 – Carga de datos

Índice

1. Carga de datos	2
2. Identificación de los procesos ETL	3
Bloque IN (de las fuentes a tablas intermedias)	4
Bloque TR (poblar las tablas de nuestro almacén)	5
3. Diseño de los procesos ETL	7
3.1. Creación de tablas intermedias (<i>staging area</i>)	7
3.2. Creación del modelo multidimensional	8
3.2.1 Dimensiones	8
3.2.2. Tablas de hechos	10
3.3 Creación del proceso de extracción, transformación y carga	11
3.3.1 Variables de entorno	11
3.3.2 Conexión a la base de datos SQL Server	12
3.3.3 Bloque IN	13
3.3.4 Bloque TR	45
3.3.5 Bloque TR_DIM	46
3.3.6 Bloque TR_FACT	65
4. Implementación de trabajos con procesos ETL	76

1. Carga de datos

Esta actividad consiste en realizar el diseño de los procesos de extracción, transformación y carga de los datos relativos a indicadores de crisis en la eurozona a partir de las fuentes de datos proporcionadas.

Los apartados que se han desarrollado en esta solución de la PRA2 son los siguientes:

- Identificación de los procesos de extracción, transformación y carga de datos (ETL) hacia el almacén de datos.
- Diseño y desarrollo de los procesos de ETL mediante las herramientas de diseño proporcionadas.
- Implementación con trabajos de los procesos de ETL.

2. Identificación de los procesos ETL

A la hora de diseñar el proceso de carga de un *data warehouse* no hay una única estrategia que sirva para todos los casos. Por un lado, es habitual estructurar los procesos de ETL sobre la base de las entidades de datos que se deben actualizar. Por otro lado, encontramos diferencias conceptuales entre la actualización de una dimensión y la de una tabla de hechos. Asimismo, la división del proceso en diferentes bloques de actualización facilitará diseñar el orden de ejecución y gestionar las dependencias. Cada uno de estos bloques de actualización se dividirá en las correspondientes etapas de extracción, transformación y carga.

Esta separación permite ejecutar los bloques de manera consecutiva, que es lo más habitual, pero también posibilita ejecutarlos de forma aislada si se requiere reprocesar alguno de los bloques por cualquier incidencia que se haya producido en la ejecución consecutiva. Cada uno de estos bloques de actualización se corresponderá con una transformación de la herramienta Pentaho Data Integration (PDI).

En el diseño de estos procesos deben considerarse una serie de factores:

- Orden de carga de los bloques.
- Ventana de tiempo disponible.
- Tipo de carga: inicial o incremental.
- Uso de un área intermedia o de maniobras (*staging area*).

En vuestro caso, y dentro del ámbito académico que nos ocupa, esta es una carga inicial, por lo que vuestros procesos no se diseñarán con el objetivo de repetirse de manera periódica. También es cierto que desconocéis la ventana de tiempo disponible (lo que no es una condición para esta actividad), pero en un contexto de producción este es un factor muy relevante. En lo relativo al área intermedia, hay que tener en cuenta que su uso, guardar la información de origen en bruto, puede facilitar mucho el trabajo de depuración de la información o de trazabilidad del dato, pudiendo ir desde el dato elaborado al dato en bruto.

En vuestro caso, identificaréis dos bloques y utilizaréis un prefijo en el nombre para identificarlos:

- **Bloque IN:** procesos de carga de los datos desde las fuentes a tablas intermedias en el área de maniobras (*staging area*). Estos procesos se distinguen por el prefijo: «IN_» en el nombre.
- **Bloque TR:** procesos de transformación para la carga de datos desde tablas intermedias a nuestro almacén según el modelo multidimensional diseñado. Se diferencian los procesos de ETL de transformación, para la carga de dimensiones; de los procesos de transformación, para la carga de las tablas de

hecho. Estos procesos se distinguen con el prefijo «TR_» en el nombre.

Bloque IN (de las fuentes a tablas intermedias)

Nombre ETL	Descripción	Orígenes de datos	Tabla destino (<i>stage</i>)
IN_COUNTRY	Carga de la información correspondiente a los países (nombre, ISO code, etc.)	countries.json	STG_COUNTRY
IN_HICP	Carga de la información relativa al indicador HICP	prc_hicp_mv12r_t abular.tsv	STG_HICP
IN_PGAS	Carga de la información relativa a la evolución del mercado energético europeo de gas por países	nrg_pc_202_tabul ar.tsv	STG_PGAS
IN_PELEC	Carga de la información relativa a la evolución del mercado energético europeo de la electricidad por países	nrg_pc_204_tabul ar.tsv	STG_PELEC
IN_COICOP	Carga de la información relativa al código y descripción de la finalidad de consumo	COICOP.xml	STG_COICOP
IN_CONSUMPTION	Carga de la información relativa al código y descripción de la franja de consumo de energía	consumption_ban d.csv	STG_CONSUMPTION

Bloque TR (poblar las tablas de nuestro almacén)

El bloque «TR_» de procesos de ETL para poblar el modelo multidimensional del almacén tiene dos partes diferenciadas: los procesos de carga y transformación de las dimensiones y los de las tablas de hechos. El orden de ejecución es importante para que la carga de datos sea correcta. Las dimensiones se cargarán primero y, después, las tablas de hechos, si no ha habido errores.

Los procesos del bloque de carga y transformación de las dimensiones son los siguientes:

Nombre ETL	Descripción	Tabla origen	Tabla destino
TR_DIM_DATE	Carga y transformación de la dimensión temporal	SQL	DIM_DATE
TR_DIM_DATE_SEMESTER	Carga y transformación de la dimensión temporal por semestres	SQL	DIM_DATE_SEMESTER
TR_DIM_COUNTRY	Carga de la dimensión con información de los países	STG_COUNTRY	DIM_COUNTRY
TR_DIM_COICOP	Carga de la información relativa al código y descripción de la finalidad de consumo	STG_COICOP	DIM_COICOP
TR_DIM_CURRENCY	Carga de la dimensión información la moneda en la que se expresa el precio de la energía	STG_PGAS STG_PEELEC	DIM_CURRENCY
TR_DIM_TAX	Carga de la dimensión con información relativa al tipo de impuesto en el que se expresa el precio de la energía	STG_PGAS STG_PEELEC	DIM_TAX
TR_DIM_UNIT	Carga la dimensión con información de la unidad de medida en la que se expresa el precio de la energía	STG_PGAS STG_PEELEC	DIM_UNIT
TR_DIM_PRODUCT	Carga la dimensión con información del tipo de energía a la que corresponde el precio	STG_PGAS STG_PEELEC	DIM_PRODUCT
TR_DIM_CONSUMPTION	Carga la dimensión con información relativa al código y descripción de la franja de consumo de energía	STG_CONSUMPTION	DIM_CONSUMPTION

Los procesos del bloque de carga y transformación de las tablas de hechos son:

Nombre ETL	Descripción	Tabla origen
TR_FACT_EUROZONE_INDICATORS	Carga de la información relativa al indicador HICP e indicadores de la eurozona	STG_HICP
TR_FACT_ENERGY_PRICE	Carga de la información relativa a la evolución de los precios de la energía (gas, electricidad, etc.)	STG_PGAS STG_PEELEC

Existen otras estrategias válidas que nos permitirán cargar los datos, ya sea organizando los procesos de otra manera o fusionándolos en un único proceso que lleve a cabo todas las tareas.

La opción de un único proceso de ETL se podría aplicar en nuestro caso, aunque no es recomendable en cargas más complejas y cambiantes.

3. Diseño de los procesos ETL

En este apartado, y teniendo en cuenta las consideraciones anteriores, vamos a diseñar e implementar los procesos de carga mediante la herramienta de diseño proporcionada: Pentaho Data Integration (PDI). Y, en particular, el programa de escritorio llamado Spoon, que corresponde al entorno gráfico IDE de desarrollo de los procedimientos de ETL.

Los procesos de ETL que diseñaremos en PDI consistirán en la definición de trabajos y transformaciones. Estas contienen la operativa de bajo nivel con las acciones que hay que realizar sobre los datos, mientras que los trabajos son procesos de alto nivel compuestos por flujos de transformaciones.

3.1. Creación de tablas intermedias (*staging area*)

El primer paso para la implementación del proceso de ETL consiste en la creación de las tablas intermedias en la *staging area*, que se llevará a cabo una única vez, mediante *scripts* sobre la base de datos, en nuestro caso SQL Server. Las tablas intermedias se utilizarán en los procesos IN, que permitirán cargar los datos desde las fuentes de datos.

IN_COUNTRY:

```
--IN_COUNTRY: Tabla de staging con la información de los países
IF OBJECT_ID(N'dbo.STG_COUNTRY', N'U') IS NOT NULL DROP TABLE dbo.STG_COUNTRY;
CREATE TABLE dbo.STG_COUNTRY(
    [name] nvarchar(100),
    [code] nvarchar(10) not null
)
```

IN_HICP:

```
--IN_HICP: Tabla de staging con los datos relativos al indicador HICP
IF OBJECT_ID(N'dbo.STG_HICP', N'U') IS NOT NULL DROP TABLE dbo.STG_HICP;
CREATE TABLE dbo.STG_HICP(
    freq nvarchar(50) not null,
    unit nvarchar(100) not null,
    coicop varchar(50) not null,
    geo nvarchar(50) not null,
    period int not null,
    HICP numeric(12,2) null
)
```

IN_PGAS:

```
--IN_PGAS: Tabla de staging con los datos relativos al precio del gas
IF OBJECT_ID(N'dbo.STG_PGAS', N'U') IS NOT NULL DROP TABLE dbo.STG_PGAS;
CREATE TABLE dbo.STG_PGAS(
    freq nvarchar(10) not null,
    product int not null,
    consom int not null,
    unit nvarchar(10) not null,
    tax nvarchar(10) not null,
    currency nvarchar(10) not null,
    geo nvarchar(10) not null,
)
```

```
period nvarchar(10) not null,
pgas numeric(12,4) null
)
```

IN_Pelec:

```
--IN_Pelec: Tabla de staging con los datos relativos al precio de la electricidad
IF OBJECT_ID(N'dbo.STG_Pelec', N'U') IS NOT NULL DROP TABLE dbo.STG_Pelec;
CREATE TABLE dbo.STG_Pelec(
    freq nvarchar(10) not null,
product int not null,
    consom int not null,
    unit nvarchar(10) not null,
    tax nvarchar(10) not null,
currency nvarchar(10) not null,
    geo nvarchar(10) not null,
period nvarchar(10) not null,
    pelec numeric(12,4) null
)
```

IN_COICOP:

```
--IN_COICOP: Tabla de staging con información de finalidad de consumo
IF OBJECT_ID(N'dbo.STG_COICOP', N'U') IS NOT NULL DROP TABLE dbo.STG_COICOP;
CREATE TABLE dbo.STG_COICOP(
    [code] nvarchar(25) not null,
Description nvarchar(300) null
)
```

IN_CONSUMPTION:

```
--IN_COICOP: Tabla de staging con información de la franja de consumo de la energía
IF OBJECT_ID(N'dbo.STG_CONSUMPTION', N'U') IS NOT NULL DROP TABLE dbo.STG_CONSUMPTION;
CREATE TABLE dbo.STG_CONSUMPTION(
    [code] nvarchar(25) not null,
Description nvarchar(250) null
)
```

Las tablas intermedias se han creado sin restricciones ni índices para facilitar la carga de datos desde las fuentes de origen.

3.2. Creación del modelo multidimensional

En este punto veréis los *scripts* de creación del modelo físico multidimensional que habéis diseñado para el almacén de datos, compuestos por las dimensiones y las tablas de hechos. En la creación, además de atributos y métricas, también se aplicarán las restricciones definidas y que son propias del modelo multidimensional, las claves primarias de las dimensiones y las foráneas de las tablas de hechos.

3.2.1. Dimensiones

DIM_DATE

```
CREATE TABLE [dbo].[DIM_DATE](
    [pk_date] Int NOT NULL,
    [Year] Int,
    [Mo] Int,
    [Quarter] Int,
CONSTRAINT [PK_DIM_DATE] PRIMARY KEY CLUSTERED
```



```
(
    [pk_date] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

DIM_DATE_SEMESTER

```
CREATE TABLE [dbo].[DIM_DATE_SEMESTER](
    [pk_date_semester] nvarchar(10) NOT NULL,
    [Year] Int,
    [Semester] nvarchar(5),
    CONSTRAINT [PK_DIM_DATE_SEMESTER] PRIMARY KEY CLUSTERED
(
    [pk_date_semester] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

DIM_COUNTRY

```
CREATE TABLE [dbo].[DIM_COUNTRY] (
    [pk_country] int NOT NULL,
    [code] nvarchar(10),
    [country_name] nvarchar(100),
    CONSTRAINT [PK_DIM_COUNTRY] PRIMARY KEY CLUSTERED
(
    [pk_country] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

DIM_COICOP

```
CREATE TABLE [dbo].[DIM_COICOP](
    [pk_coicop] int NOT NULL,
    [code] nvarchar(25),
    [coicop_name] nvarchar(300),
    CONSTRAINT [PK_DIM_COICOP] PRIMARY KEY CLUSTERED
(
    [pk_coicop] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

DIM_CURRENCY

```
CREATE TABLE [dbo].[DIM_CURRENCY](
    [pk_currency] int NOT NULL,
    [currency_name] nvarchar(10),
    CONSTRAINT [PK_DIM_CURRENCY] PRIMARY KEY CLUSTERED
(
    [pk_currency] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

DIM_TAX

```
CREATE TABLE [dbo].[DIM_TAX](
    [pk_tax] int NOT NULL,
    [tax_name] nvarchar(50),
```

```

CONSTRAINT [PK_DIM_TAX] PRIMARY KEY CLUSTERED
(
    [pk_tax] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

```

DIM_UNIT

```

CREATE TABLE [dbo].[DIM_UNIT](
    [pk_unit] int NOT NULL,
    [unit_name] nvarchar(25),
    CONSTRAINT [PK_DIM_UNIT] PRIMARY KEY CLUSTERED
(
    [pk_unit] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

```

DIM_PRODUCT

```

CREATE TABLE [dbo].[DIM_PRODUCT](
    [pk_product] int NOT NULL,
    [product_code] int,
    CONSTRAINT [PK_DIM_PRODUCT] PRIMARY KEY CLUSTERED
(
    [pk_product] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

```

DIM_CONSUMTION

```

CREATE TABLE [dbo].[DIM_CONSUMPTION](
    [pk_consumption] int NOT NULL,
    [code] nvarchar(25),
    [consumption_name] nvarchar(250),
    CONSTRAINT [PK_DIM_CONSUMPTION] PRIMARY KEY CLUSTERED
(
    [pk_consumption] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

```

3.2.2. Tablas de hechos

FACT_EUROZONE_INDICATORS

```

CREATE TABLE [dbo].[FACT_EUROZONE_INDICATORS] (
    [pk_id] int NOT NULL,
    [fk_date] int,
    [fk_country] int,
    [fk_coicop] int,
    [HICP] numeric(12,2),
    CONSTRAINT [PK_FACT_EUROZONE_INDICATORS] PRIMARY KEY CLUSTERED
(
    [pk_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

```

FACT_ENERGY_PRICE

```
CREATE TABLE [dbo].[FACT_ENERGY_PRICE] (
    [pk_id] int NOT NULL,
    [fk_date_semester] nvarchar(10),
    [fk_country] int,
    [fk_currency] int,
    [fk_tax] int,
    [fk_consumption] int,
    [fk_unit] int,
    [fk_product] int,
    [price] numeric(12,4)
CONSTRAINT [PK_FACT_ENERGY_PRICE] PRIMARY KEY CLUSTERED
(
    [pk_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Claves foráneas

```
ALTER TABLE FACT_EUROZONE_INDICATORS ADD FOREIGN KEY (fk_date) REFERENCES DIM_DATE(pk_date);
ALTER TABLE FACT_EUROZONE_INDICATORS ADD FOREIGN KEY (fk_country) REFERENCES DIM_COUNTRY(pk_country);
ALTER TABLE FACT_EUROZONE_INDICATORS ADD FOREIGN KEY (fk_coicop) REFERENCES DIM_COICOP(pk_coicop);

ALTER TABLE FACT_ENERGY_PRICE ADD FOREIGN KEY (fk_date_semester) REFERENCES
DIM_DATE_SEMESTER(pk_date_semester);
ALTER TABLE FACT_ENERGY_PRICE ADD FOREIGN KEY (fk_country) REFERENCES DIM_COUNTRY(pk_country);
ALTER TABLE FACT_ENERGY_PRICE ADD FOREIGN KEY (fk_currency) REFERENCES DIM_CURRENCY(pk_currency);
ALTER TABLE FACT_ENERGY_PRICE ADD FOREIGN KEY (fk_tax) REFERENCES DIM_TAX(pk_tax);
ALTER TABLE FACT_ENERGY_PRICE ADD FOREIGN KEY (fk_consumption) REFERENCES
DIM_CONSUMPTION(pk_consumption);
ALTER TABLE FACT_ENERGY_PRICE ADD FOREIGN KEY (fk_unit) REFERENCES DIM_UNIT(pk_unit);
ALTER TABLE FACT_ENERGY_PRICE ADD FOREIGN KEY (fk_product) REFERENCES DIM_PRODUCT(pk_product);
```

3.3. Creación del proceso de extracción, transformación y carga

Una vez que tengáis implementado el modelo físico del almacén, pasaréis a diseñar los procesos de ETL que permitirán poblar las tablas intermedias del área intermedia (*staging area*) y las tablas de dimensiones y de hechos del *data mart* que habéis diseñado.

Antes del diseño de las transformaciones, debéis definir en PDI las variables de entorno que usaréis en la implementación de los procesos de ETL, así como la conexión a la base de datos que utilizaréis en todos.

3.3.1. Variables de entorno

Es una buena práctica utilizar variables de entorno para evitar introducir errores en definiciones repetitivas durante la implementación de los procesos. PDI os permite

añadir variables personalizadas y propias de vuestros desarrollos en el archivo «kettle.properties».

En vuestro caso, podéis utilizar tres variables. Una para almacenar la ruta de las fuentes de datos y otras dos para reunir las cadenas de conexión a la base de datos, «CN_STAGE» (área intermedia o *staging area*) y «CN_DW» (*data warehouse*). Se podría crear un esquema *stage* en el SQL Server dentro de la base de datos asignada al estudiante para cargar las tablas intermedias (IN_) y definir la variable «CN_STAGE» haciendo referencia a este esquema, pero para simplificar la solución de la práctica se cargarán todas las tablas al esquema por defecto, *dbo*.

Variable	Valor
DIR_IN	F:\fuentes
CN_STAGE	jdbc:sqlserver://UCS1R1UOCSQL01:1433;databaseName=SOURCE_loginuoc;integratedSecurity=false
CN_DW	jdbc:sqlserver://UCS1R1UOCSQL01:1433;databaseName=SOURCE_loginuoc;integratedSecurity=false

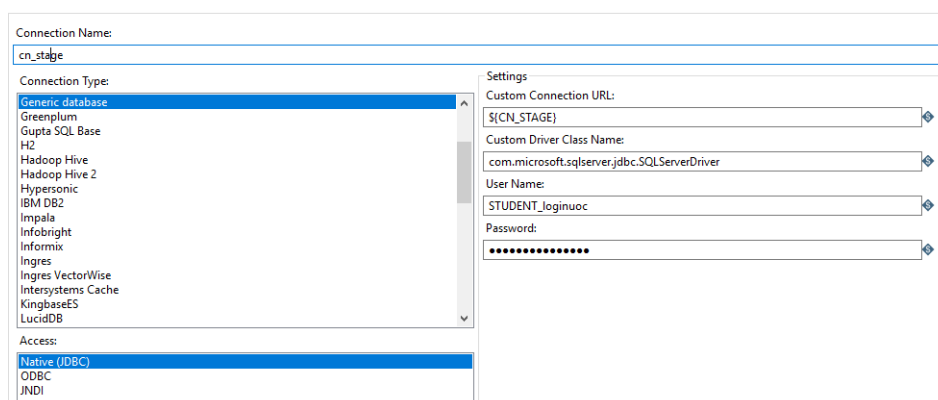
La referencia a las variables de entorno durante la implementación de los procesos se realiza mediante llaves, de esta manera: {DIR_IN}, {CN_STAGE}, {CN_DW}.

3.3.2. Conexión a la base de datos SQL Server

Otro paso previo que se debe realizar es crear las conexiones a las bases de datos que se usan en todas las transformaciones y trabajos de los procesos de carga.

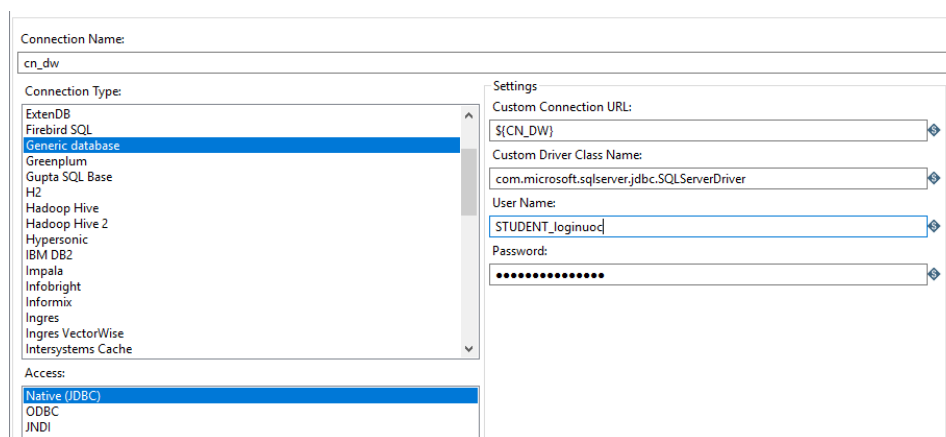
Se han definido dos conexiones diferentes, una para la base de datos del modelo multidimensional («BBDD») y otra para el área intermedia («STAGE»); de esta manera diferenciaréis claramente su uso, aunque físicamente se refieran al mismo esquema de la base de datos.

En la creación de la conexión al «STAGE», el nombre que utilizaréis es «cn_stage»:



The screenshot shows the 'Connection Name' field set to 'cn_stage'. Under 'Connection Type', 'Generic database' is selected. The 'Settings' tab is active, showing the 'Custom Connection URL' as '{CN_STAGE}', the 'Custom Driver Class Name' as 'com.microsoft.sqlserver.jdbc.SQLServerDriver', the 'User Name' as 'STUDENT_loginuoc', and the 'Password' field masked with dots. The 'Access' section shows 'Native (JDBC)' selected.

En la creación de la conexión al «DW», el nombre que le daréis es «cn_dw»:



3.3.3. Bloque IN

En el bloque IN solo aplicaréis las transformaciones básicas sobre el origen de datos. Transformaciones más complejas, como campos calculados, se realizarán en el bloque TR. El objetivo es disponer de una «copia rápida» de los orígenes de los datos normalizados. Esto os permitirá trazar el dato en caso de ser necesario.

Transformación IN_HICP

Fuente: prc_hicp_mv12r_tabular.tsv

Destino: STG_HICP (SQL Server)

A continuación, se describe parte del desarrollo de la transformación de «IN_HICP» (identificada en el primer punto de la guía) mediante Spoon. El objetivo es cargar uno de los orígenes de los datos identificados, «prc_hicp_mv12r_tabular.tsv», en la tabla «STG_HICP» del área intermedia (*staging area*). La tabla intermedia tendrá que haber sido creada con anterioridad en la base de datos analítica, cuyo *script* se habrá escrito en el apartado de «creación».

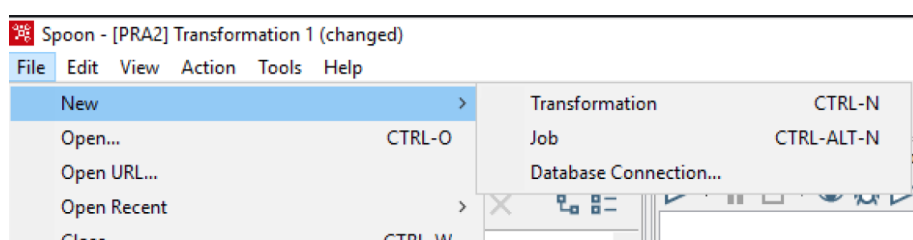
Para este caso práctico habéis utilizado fuentes externas (no operacionales) que emplearéis para descubrir el conocimiento mediante el análisis de los datos. Es muy habitual manipular los ficheros realizando manualmente una serie de acciones de preparación antes de su procesamiento (preprocesado).

La transformación de «IN_HICP» contiene las siguientes etapas:

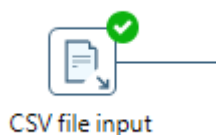
- lectura del fichero .tsv,
- separación de campos,
- normalización de filas,
- tratamiento de nulos y reemplazo de valores tipo «string»,
- conversiones de tipología de datos,
- carga a la tabla intermedia «STG_HICP».

A continuación, se detallan las diferentes etapas que implementaréis para poder realizar la carga de datos.

En primer lugar, se crea una nueva transformación.



En esta, como primer paso, realizaréis la entrada del fichero .tsv. Para llevarlo a cabo utilizaréis el tipo «CSV file input». En este paso deberéis indicar el fichero desde donde extraéis los datos. Para hacerlo, emplearéis la variable de entorno «DIR_IN» e indicaréis el tipo de motor que se deberá usar.



Para realizar correctamente la carga, deberéis indicar los parámetros correctamente, destacando en este caso lo siguiente:

- Indicar como delimitador TAB. Para ello, deberéis apretar en el botón «Insert TAB», que generará automáticamente un valor en blanco como se muestra en la imagen.
- Desmarcar «Lazy conversion?».

CSV file input

Step name: CSV file input

Filename: \${DIR_IN}\prc_hicp_mv12r_tabular.tsv Browse...

Delimiter: Insert TAB

Enclosure: "

NIO buffer size: 50000

Lazy conversion? ☐

Header row present? ☒

Add filename to result ☐

The row number field name (optional):

Running in parallel? ☐

New line possible in fields? ☐


Format: mixed

File encoding:

#	Name	Type	Format	Length	Precision	Currency	Decimal
1	freq,unit,coicop,geo\TIME_PERIOD	String		25		\$,
2	2020-01	String		6		\$,
3	2020-02	String		6		\$,
4	2020-03	String		6		\$,
5	2020-04	String		6		\$,
6	2020-05	String		6		\$,
7	2020-06	String		6		\$,
8	2020-07	String		6		\$,

En este paso, también tendréis que indicar los campos que vais a tratar con el botón «Get fields» y completareis su definición. Hay que especificar, donde se considere necesario, la precisión y la longitud de los campos.

Podéis realizar una visualización previa de los datos que se cargarán con el botón «Preview».

 Examine preview data

Rows of step: CSV file input (1000 rows)

#	freq,unit,coicop,geo\TIME_PERIOD	2020-01	2020-02	2020-03	2020-04	2020-05	2020-06	2020-07	2020-08	2020-
1	M,RCH_MV12MAVR,AP,AT	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2	2.2
2	M,RCH_MV12MAVR,AP,BE	1.7	1.7	1.7	1.7	1.7	1.6	1.6	1.5	1.5
3	M,RCH_MV12MAVR,AP,BG	2.6	2.6	2.6	2.5	2.4	2.3	2.3	2.2	2.0
4	M,RCH_MV12MAVR,AP,CH	:	:	:	:	:	:	:	:	:
5	M,RCH_MV12MAVR,AP,CY	2.1	1.8	1.7	1.5	1.3	0.8	0.2	-0.1	-0.5
6	M,RCH_MV12MAVR,AP,CZ	3.9	4.1	4.2	4.2	4.2	4.2	4.3	4.3	4.2
7	M,RCH_MV12MAVR,AP,DE	1.6	1.6	1.6	1.7	1.7	1.7	1.7	1.7	1.7
8	M,RCH_MV12MAVR,AP,DV	1.4	1.4	1.4	1.5	1.5	1.6	1.6	1.6	1.7

Como vemos en la imagen anterior, los campos de «string» siguen concatenados en uno solo, por lo que necesitaremos una transformación para separar columnas: «Split fields». Indicaremos el campo que queremos separar y el delimitador como se muestra seguidamente:

Split fields

Step name:

Field to split:

Delimiter:

Enclosure:

Fields

#	New field	ID	Remove ID?	Type	Length	Precision	Format	Group	Decimal	Currency	Nullif	Default	Trim type
1	freq		N	String									none
2	unit		N	String									none
3	coicop		N	String									none
4	geo\TIME_PERIOD		N	String									none

Una vez tengamos las columnas separadas, procederemos a convertir los campos de periodo en filas. Para ello usaremos la transformación de «Row normaliser»:

Row normaliser

Step name:

Type field:

Fields

#	Fieldname	Type	new field
1	2020-01	2020-01	Value
2	2020-02	2020-02	Value
3	2020-03	2020-03	Value
4	2020-04	2020-04	Value
5	2020-05	2020-05	Value
6	2020-06	2020-06	Value
7	2020-07	2020-07	Value
8	2020-08	2020-08	Value
9	2020-09	2020-09	Value
10	2020-10	2020-10	Value
11	2020-11	2020-11	Value
12	2020-12	2020-12	Value

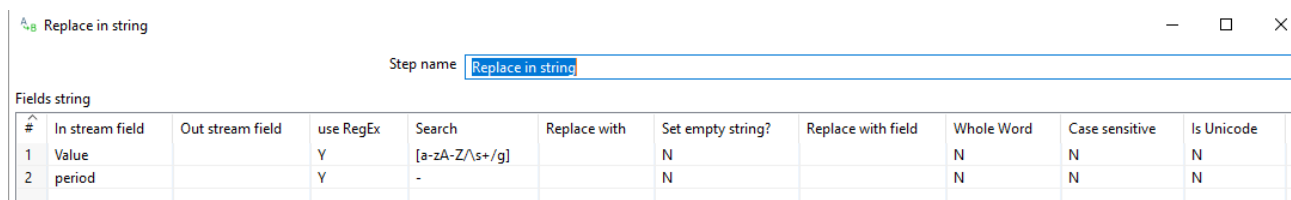
Execution Results

Logging Execution History Step Metrics Performance Graph Metrics Preview data

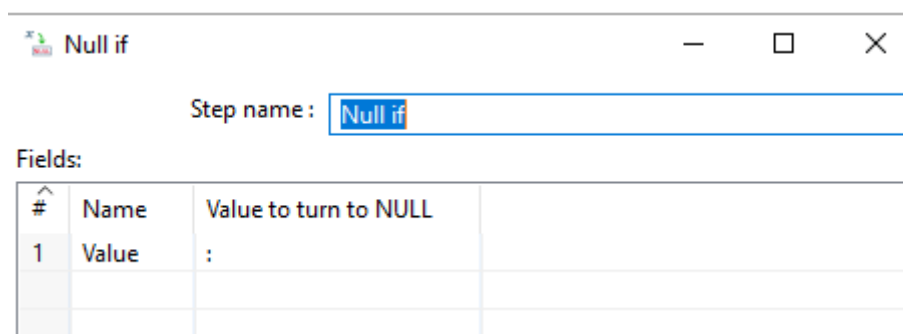
☒ First rows ☐ Last rows ☐ Off

#	freq	unit	coicop	geo\TIME_PERIOD	period	Value
1	M	RCH_MV12MAVR	AP	AT	2020-01	2.2
2	M	RCH_MV12MAVR	AP	AT	2020-02	2.2
3	M	RCH_MV12MAVR	AP	AT	2020-03	2.2
4	M	RCH_MV12MAVR	AP	AT	2020-04	2.2
5	M	RCH_MV12MAVR	AP	AT	2020-05	2.2
6	M	RCH_MV12MAVR	AP	AT	2020-06	2.2

Una vez convertidos los periodos en filas y con el valor de «Value» (HICP) para cada uno de ellos, vemos que los valores nulos tienen la expresión «:». Antes de reemplazar estos valores, nos aseguraremos de que no existen otros caracteres con la transformación «Replace in string»:



Y ahora sí podremos proceder a reemplazar el símbolo «:», por nulos mediante la transformación «Null if» sobre el campo «Value»:



Finalmente, ya podremos realizar la transformación del tipo de datos con el componente «Select values». Aprovecharemos para:

- Cambiar el nombre de los campos «geo» y «Value» como se muestra en la imagen.
- Repasaremos que todos los campos son de tipo «string», a excepción de «HICP», que será decimal, y «period», que será entero.

Select values

Step name:

Select & Alter Remove Meta-data

Fields to alter the meta-data for :

#	Fieldname	Rename ...	Type	Len...	Precisi...	Binary t...	Format
1	freq		String			N	
2	unit		String			N	
3	coicop		String			N	
4	geo\TIME_PERIOD	geo	String			N	
5	period		Integer	6		N	
6	Value	hicp	Number			N	#####.###

Get

Por último, cargaréis los datos en la tabla intermedia del *stage*, utilizando el paso «Table output». Para este paso es necesario especificar la conexión de la base de datos, que la podéis generar dentro del propio componente, pulsando el botón «New» e indicándole que queréis conectaros a Microsoft SQL Server y utilizar el acceso nativo con vuestras credenciales. También podréis utilizar las variables de entorno creadas previamente.

Database Connection

General
Advanced
Options
Pooling
Clustering

Connection name:

Connection type:
 MS SQL Server
 MS SQL Server (Native)
 MariaDB
 MaxDB (SAP DB)
 MonetDB
 MySQL
 Native Mondrian
 Neoview
 Netezza
 Oracle
 Oracle RDB
 Palo MOLAP Server
 Pentaho Data Services
 PostgreSQL

Access:
 Native (JDBC)
 ODBC
 JNDI

El paso de cargar los datos a la tabla intermedia del *stage* lo configuraréis como se os indica en el menú principal que veis a continuación. Es importante que tengáis la tabla de STG creada para poder insertar los datos preprocesados.

Table output

Step name: Insert into SQL Server

Connection: cn_stage [Edit...] [New...] [Wizard...]

Target schema: dbo [Browse...]

Target table: STG_HICP [Browse...]

Commit size: 1000

Truncate table: ☒

Ignore insert errors: ☐

Specify database fields: ☒

Main options Database fields

Fields to insert:

#	Table field	Stream field
1	freq	freq
2	unit	unit
3	coicop	coicop
4	geo	geo
5	period	period
6	hicp	hicp

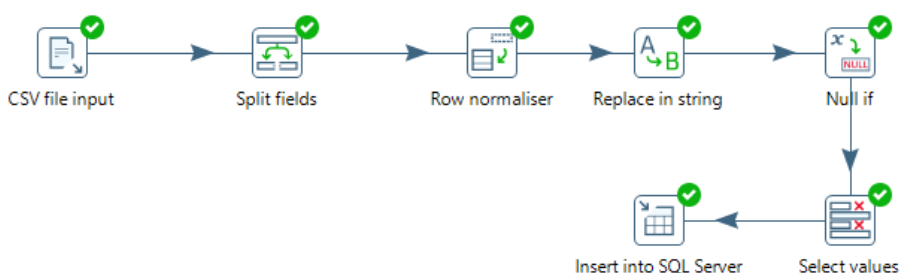
[Get fields]

[Enter field mapping]

Como veis en la imagen, para dejar la transformación preparada para posibles reprocesos, es necesario realizar un borrado previo para actualizar los datos en el caso de que tuvierais que efectuarlo. Para esto, activaréis el *check* «Truncate table» situado en los campos de la base de datos.

Spoon nos permite mapear los campos que vienen del flujo de datos de la ETL (*stream*) con los de la tabla destino, activando «Specify database fields». Si no se especifica, se van a insertar los registros mapeando solo los campos que tengan el mismo nombre.

El proceso de la transformación completa es el siguiente:



Y podemos comprobar en la base de datos el resultado del proceso realizado:

```

SELECT TOP (1000) [freq]
, [unit]
, [coicop]
, [geo]
, [period]
, [HICP]
FROM [SOURCE_loginuoc].[dbo].[STG_HICP]

```

freq	unit	coicop	geo	period	HICP
M	RCH_MV12MAVR	AP	CH	202102	-0.60
M	RCH_MV12MAVR	AP	DE	202204	1.90
M	RCH_MV12MAVR	AP	ES	202006	-0.90
M	RCH_MV12MAVR	AP	HR	202108	1.40
M	RCH_MV12MAVR	AP	IT	202210	18.00

Transformación IN_PGAS

Fuente: nrg_pc_202_tabular.tsv

Destino: STG_PGAS (SQL Server)

A continuación, se describe parte del desarrollo de la transformación de «IN_PGAS» (identificada en el primer punto de la guía) mediante Spoon. El objetivo es cargar uno de los orígenes de los datos identificados, «nrg_pc_202_tabular.tsv», en la tabla «STG_PGAS» del área intermedia (*staging area*). La tabla intermedia tendrá que haber sido creada con anterioridad en la base de datos analítica, cuyo *script* se habrá escrito en el apartado de «creación».

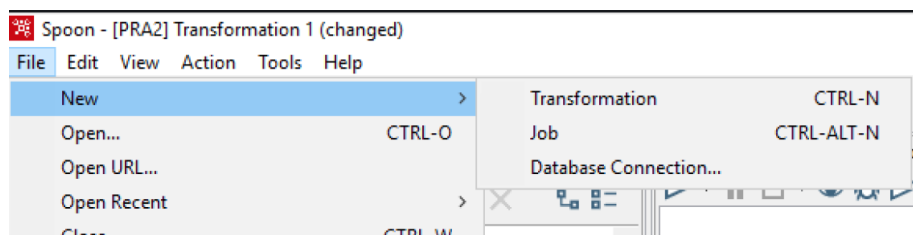
Para este caso práctico habéis utilizado fuentes externas (no operacionales) que emplearéis para descubrir el conocimiento mediante el análisis de los datos. Es muy habitual manipular los ficheros realizando manualmente una serie de acciones de preparación antes de su procesamiento (preprocesado).

La transformación de «IN_PGAS» contiene las siguientes etapas:

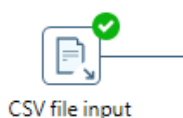
- lectura del fichero .tsv,
- separación de campos,
- normalización de filas,
- tratamiento de nulos y reemplazo de valores tipo «string»,
- conversiones de tipología de datos,
- carga a la tabla intermedia «STG_PGAS».

A continuación, se detallan las diferentes etapas que implementaréis para poder realizar la carga de datos.

En primer lugar, se crea una nueva transformación.



En esta, como primer paso, realizaréis la entrada del fichero .tsv. Para llevarlo a cabo utilizaréis el tipo «CSV input». En este paso deberéis indicar el fichero desde donde extraéis los datos. Para hacerlo, emplearéis la variable de entorno «DIR_IN» e indicaréis el tipo de motor que se deberá usar.



Para realizar correctamente la carga, deberéis indicar los parámetros correctamente, destacando en este caso lo siguiente:

- Indicar como delimitador TAB. Para ello, deberéis apretar en el botón «Insert TAB», que generará automáticamente un valor en blanco como se muestra en la imagen.
- Desmarcar «Lazy conversion?».

CSV file input

Step name: CSV file input

Filename: \${DIR_IN}nrg_pc_202_tabular.tsv Browse...

Delimiter: Insert TAB

Enclosure: "

NIO buffer size: 50000

Lazy conversion? ☐

Header row present? ☒

Add filename to result ☐

The row number field name (optional):

Running in parallel? ☐

New line possible in fields? ☐

Format: mixed

File encoding:

#	Name	Type	Format	Length	Precision	Curre
1	freq,product,consom,unit,tax,currency,geo\TIME_PERIOD	String		41		\$
2	2007-S1	String		9		\$
3	2007-S2	String		10		\$
4	2008-S1	String		10		\$
5	2008-S2	String		10		\$
6	2009-S1	String		10		\$
7	2009-S2	String		10		\$
8	2010-S1	String		10		\$
9	2010-S2	String		10		\$

En este paso, también tendréis que indicar los campos que vais a tratar con el botón «Get fields» y completareis su definición. Hay que especificar, donde se considere necesario, la precisión y la longitud de los campos.

Podéis realizar una visualización previa de los datos que se cargarán con el botón «Previsualizar filas».

[illegible]

Como vemos en la imagen anterior, los campos de «string» siguen concatenados en uno solo, por lo que requeriremos una transformación para separar columnas: «Split fields». Indicaremos el campo que queremos separar y el delimitador como se muestra seguidamente:

Split fields

Step name:

Field to split:

Delimiter:

Enclosure:

Fields

#	New field	ID	Remove ID?	Type	Length	Precision	Format	Group	Decimal	Currency	Nullif	Default	Trim type
1	freq		N	String									none
2	product		N	String									none
3	consom		N	String									none
4	unit		N	String									none
5	tax		N	String									none
6	currency		N	String									none
7	geo\TIME_PERIOD		N	String									none

Una vez tengamos las columnas separadas, procederemos a convertir los campos de periodo en filas. Para ello usaremos la transformación de «Row normaliser»:

Row normaliser

Step name:

Type field:

Fields

#	Fieldname	Type	new field
1	2007-S1	2007-S1	Value
2	2007-S2	2007-S2	Value
3	2008-S1	2008-S1	Value
4	2008-S2	2008-S2	Value
5	2009-S1	2009-S1	Value
6	2009-S2	2009-S2	Value
7	2010-S1	2010-S1	Value
8	2010-S2	2010-S2	Value
9	2011-S1	2011-S1	Value
10	2011-S2	2011-S2	Value
11	2012-S1	2012-S1	Value
12	2012-S2	2012-S2	Value
13	2013-S1	2013-S1	Value

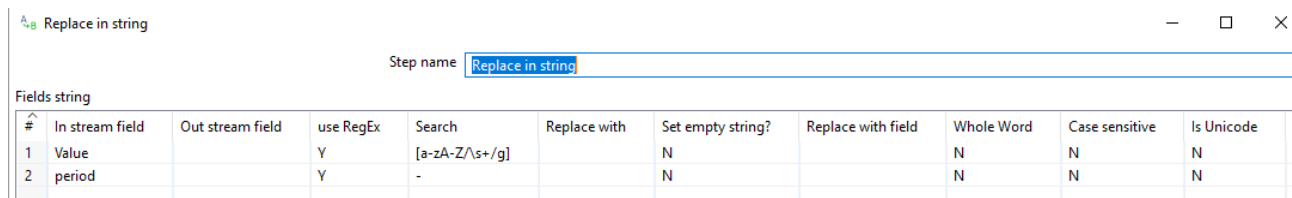
Execution Results

Logging Execution History Step Metrics Performance Graph Metrics Preview data

☒ First rows ☐ Last rows ☐ Off

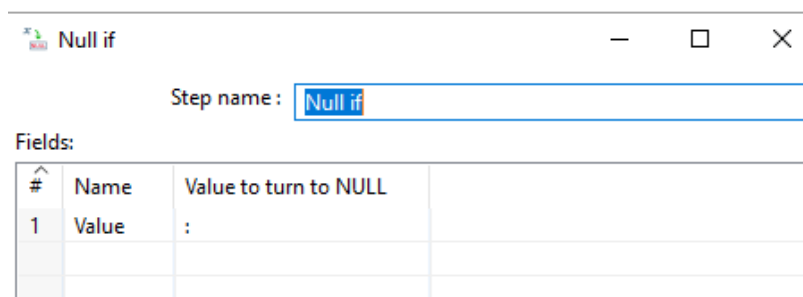
#	freq	product	consom	unit	tax	currency	geo\TIME_PERIOD	period	Value
1	S	4100	4141901	GJ_GCV	I_TAX	EUR	AT	2007-S1	:
2	S	4100	4141901	GJ_GCV	I_TAX	EUR	AT	2007-S2	20.9300
3	S	4100	4141901	GJ_GCV	I_TAX	EUR	AT	2008-S1	20.2600
4	S	4100	4141901	GJ_GCV	I_TAX	EUR	AT	2008-S2	20.9000
5	S	4100	4141901	GJ_GCV	I_TAX	EUR	AT	2009-S1	21.0200

Una vez convertidos los periodos en filas y con el valor de «Value» (PGAS) para cada uno de ellos, vemos que los valores nulos tienen la expresión «:». Antes de reemplazar estos valores, nos aseguraremos de que no existen otros caracteres con la transformación «Replace in string»:



#	In stream field	Out stream field	use RegEx	Search	Replace with	Set empty string?	Replace with field	Whole Word	Case sensitive	Is Unicode
1	Value		Y	[a-zA-Z/\s+/g]		N		N	N	N
2	period		Y	-		N		N	N	N

Y ahora sí podremos proceder a reemplazar el símbolo «:», por nulos mediante la transformación «Null if» sobre el campo que acabamos de crear «Value»:



#	Name	Value to turn to NULL
1	Value	:

Finalmente, ya podremos realizar la transformación del tipo de datos con el componente «Select values». Aprovecharemos para:

- Cambiar el nombre de los campos «geo» y «Value» como se muestra en la imagen.
- Repasaremos que todos los campos son de tipo «string», a excepción de «Value» (PGAS) que será decimal, y «product» y «consom», que serán «integer».
- Indicaremos decimales (.) para el campo de «Value» y especificaremos la precisión y formato en caso de que sea necesario.

Select values

Step name:

Select & Alter Remove Meta-data

Fields to alter the meta-data for:

#	Fieldname	Rename to	Type	Length	Precision	Binary t
1	freq		String			N
2	product		Integer			N
3	consom		Integer			N
4	unit		String			N
5	tax		String			N
6	currency		String			N
7	geo\TIME_PERIOD	geo	String			N
8	period		String	6		N
9	Value	pelec	Number		4	N

< >

Por último, cargaréis los datos en la tabla intermedia del *stage*, utilizando el paso «Table output». Para este paso es necesario especificar la conexión de la base de datos, que la podéis generar dentro del propio componente, pulsando el botón «New» e indicándole que queréis conectaros a Microsoft SQL Server y utilizar el acceso nativo con vuestras credenciales. También podréis utilizar las variables de entorno creadas previamente.

Database Connection

General
Advanced
Options
Pooling
Clustering

Connection name:

Connection type:
 MS SQL Server
 MS SQL Server (Native)
 MariaDB
 MaxDB (SAP DB)
 MonetDB
 MySQL
 Native Mondrian
 Neoview
 Netezza
 Oracle
 Oracle RDB
 Palo MOLAP Server
 Pentaho Data Services
 PostgreSQL

Access:
 Native (JDBC)
 ODBC
 JNDI

El paso de cargar los datos a la tabla intermedia del *stage* lo configuraréis como se os indica en el menú principal que veis a continuación. Es importante que tengáis la tabla de STG creada para poder insertar los datos preprocesados.

Table output

Step name:

Connection:

Target schema:

Target table:

Commit size:

Truncate table: ☒

Ignore insert errors: ☐

Specify database fields: ☒

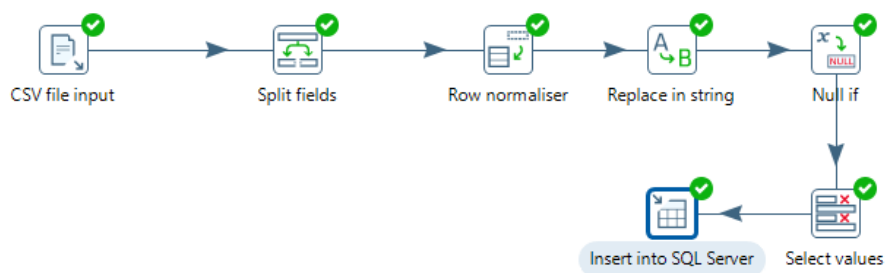
Main options Database fields

Fields to insert:

#	Table field	Stream field
1	freq	freq
2	product	product
3	consom	consom
4	unit	unit
5	tax	tax
6	currency	currency
7	geo	geo
8	period	period
9	pgas	pgas

Como veis en la imagen, para dejar la transformación preparada para posibles reprocesos, es necesario realizar un borrado previo para actualizar los datos en el caso de que tuvierais que efectuarlo. Para esto, activaréis el *check* «Truncate table» situado en los campos de la base de datos.

El proceso de la transformación completa es el siguiente:



Y podemos comprobar en la base de datos el resultado del proceso realizado:

```
SELECT TOP (1000) [freq]
, [product]
, [consom]
, [unit]
, [tax]
, [currency]
, [geo]
, [period]
, [pgas]
FROM [SOURCE_loginuoc].[dbo].[STG_PGAS]
```

	freq	product	consom	unit	tax	currency	geo	period	pgas
1	S	4100	4141901	GJ_GCV	X_VAT	EUR	DK	2011-S1	22.9153
2	S	4100	4141901	GJ_GCV	X_VAT	EUR	DK	2011-S2	22.8214
3	S	4100	4141901	GJ_GCV	X_VAT	EUR	DK	2012-S1	21.7755
4	S	4100	4141901	GJ_GCV	X_VAT	EUR	DK	2012-S2	21.2228
5	S	4100	4141901	GJ_GCV	X_VAT	EUR	DK	2013-S1	22.0686
6	S	4100	4141901	GJ_GCV	X_VAT	EUR	DK	2013-S2	21.7117

Transformación IN_PEEC

Fuente: nrg_pc_204_tabular.tsv

Destino: STG_PEEC (SQL Server)

A continuación, se describe parte del desarrollo de la transformación de «IN_PEEC» (identificada en el primer punto de la guía) mediante Spoon. El objetivo es cargar uno de los orígenes de los datos identificados, «nrg_pc_204_tabular.tsv», en la tabla «STG_PEEC» del área intermedia (*staging area*). La tabla intermedia tendrá que haber sido creada con anterioridad en la base de datos analítica, cuyo *script* se habrá escrito en el apartado de «creación».

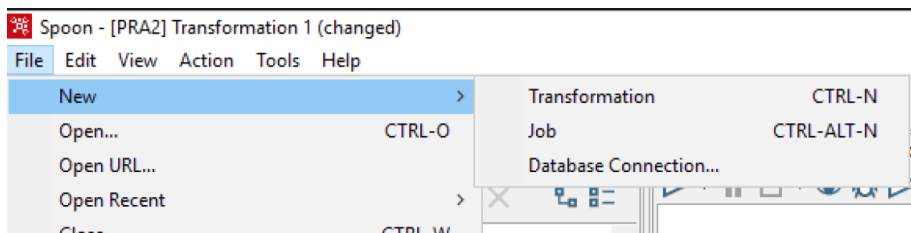
Para este caso práctico habéis utilizado fuentes externas (no operacionales) que emplearéis para descubrir el conocimiento mediante el análisis de los datos. Es muy habitual manipular los ficheros realizando manualmente una serie de acciones de preparación antes de su procesamiento (preprocesado).

La transformación de «IN_PEEC» contiene las siguientes etapas:

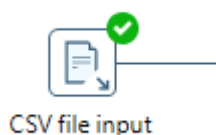
- lectura del fichero .tsv,
- separación de campos,
- normalización de filas,
- tratamiento de nulos y reemplazo de valores tipo «string»,
- conversiones de tipología de datos,
- carga a la tabla intermedia «STG_PEEC».

A continuación, se detallan las diferentes etapas que implementaréis para poder realizar la carga de datos.

En primer lugar, se crea una nueva transformación.



En esta, como primer paso, realizaréis la entrada del fichero .tsv. Para llevarlo a cabo utilizaréis el tipo «CSV input». En este paso deberéis indicar el fichero desde donde extraéis los datos. Para hacerlo, emplearéis la variable de entorno «DIR_IN» e indicaréis el tipo de motor que se deberá usar.



Para realizar correctamente la carga, deberéis indicar los parámetros correctamente, destacando en este caso lo siguiente:

- Indicar como delimitador TAB. Para ello, deberéis apretar en el botón «Insert TAB», que genera automáticamente un valor en blanco como se muestra en la imagen.
- Desmarcar «Lazy conversion?».

Step name: CSV file input

Filename: Browse...

Delimiter: Insert TAB

Enclosure:

NIO buffer size:

Lazy conversion? ☐

Header row present? ☒

Add filename to result ☐

The row number field name (optional):

Running in parallel? ☐

New line possible in fields? ☐

Format:

File encoding:

#	Name	Type	Format	Length	Precision	Curre
1	freq,product,consom,unit,tax,currency,geo\TIME_PERIOD	String		38		\$
2	2007-S1	String		7		\$
3	2007-S2	String		8		\$
4	2008-S1	String		8		\$
5	2008-S2	String		8		\$
6	2009-S1	String		8		\$

En este paso, también tendréis que indicar los campos que vais a tratar con el botón «Get fields» y completareis su definición. Hay que especificar, donde se considere necesario, la precisión y la longitud de los campos.

Podéis realizar una visualización previa de los datos que se cargarán con el botón «Preview».

Examine preview data

Rows of step: CSV file input (1000 rows)

#	freq	product	consom	unit	tax	currency	geo	TIME_PERIOD	2007-51	2007-52	2008-51	2008-52	2009-51	2009-52	2010-51	2010-52	2011-51	2011-52	2012-51	2012-52	2013-51	2013-52	2014-51	
1	:	S,6000	4161901	KWH,I	TAX	EUR,AL	:	:	:	:	:	:	:	:	:	:	0.1152	0.1157	0.1163	0.1167	0.1156	0.1154	0.1156	
2	:	S,6000	4161901	KWH,I	TAX	EUR,AT	:	:	:	0.2653	0.2650	0.2649	0.2727	0.2726	0.2538	0.2570	0.2946	0.2918	0.2978	0.2988	0.3181	0.3112	0.3175	
3	:	S,6000	4161901	KWH,I	TAX	EUR,BA	:	:	:	:	:	:	:	:	:	:	:	:	0.2164	0.1950	0.2270	0.2121	0.2269	
4	:	S,6000	4161901	KWH,I	TAX	EUR,BE	:	:	:	0.2443	0.2785	0.3593	0.2628	0.2780	0.2930	0.2852	0.3034	0.2965	0.2921	0.3102	0.2907	0.2929	0.2661	
5	:	S,6000	4161901	KWH,I	TAX	EUR,BG	:	:	:	0.0741	0.0741	0.0823	0.0844	0.0823	0.0823	0.0842	0.0840	0.0852	0.0857	0.0957	0.0957	0.0880	0.0851	
6	:	S,6000	4161901	KWH,I	TAX	EUR,CY	:	:	:	0.1674	0.1921	0.1992	0.1921	0.1601	0.2120	0.2224	0.2353	0.2657	0.3180	0.3179	0.3073	0.2731	0.2638	
7	:	S,6000	4161901	KWH,I	TAX	EUR,CZ	:	:	:	0.2106	0.2166	0.2492	0.2541	0.2517	0.2651	0.2612	0.2703	0.2883	0.2827	0.2915	0.2921	0.2922	0.2860	0.2587
8	:	S,6000	4161901	KWH,I	TAX	EUR,DE	:	:	:	0.3232	0.3347	0.3415	0.3480	0.3599	0.3567	0.3664	0.3725	0.3835	0.3831	0.3917	0.4028	0.4257	0.4267	0.4329
9	:	S,6000	4161901	KWH,I	TAX	EUR,DK	:	:	:	:	0.2676	0.2920	0.3070	0.2991	0.2847	0.2973	0.3010	0.3215	0.3283	0.3308	0.3278	0.3304	0.3217	0.3325
10	:	S,6000	4161901	KWH,I	TAX	EUR,EE	:	:	:	0.3230	0.2687	0.2763	0.2835	0.2811	0.2851	0.2953	0.2969	0.3032	0.3078	0.3041	0.3171	0.3320	0.3359	0.3479
11	:	S,6000	4161901	KWH,I	TAX	EUR,EL	:	:	:	:	0.0812	0.0838	0.0872	0.0945	0.0947	0.0995	0.1030	0.1008	0.1069	0.1143	0.1152	0.1424	0.1408	0.1347
12	:	S,6000	4161901	KWH,I	TAX	EUR,ES	:	:	:	:	0.1972	0.3320	0.1072	0.1940	0.1990	0.1575	0.1160	0.1508	0.1404	0.1160	0.1132	0.1924	0.1920	0.1930

Como vemos en la imagen anterior, los campos de «string» siguen concatenados en uno solo, por lo que requeriremos una transformación para separar columnas: «Split fields». Indicaremos el campo que queremos separar y el delimitador como se muestra a continuación:

Split fields

Step name:

Field to split:

Delimiter:

Enclosure:

Fields

#	New field	ID	Remove ID?	Type	Length	Precision	Format	Group	Decimal	Currency	Nullif	Default	Trim type
1	freq		N	String									none
2	product		N	String									none
3	consom		N	String									none
4	unit		N	String									none
5	tax		N	String									none
6	currency		N	String									none
7	geo\TIME_PERIOD		N	String									none

Una vez tengamos las columnas separadas, procederemos a convertir los campos de periodo en filas. Para ello usaremos la transformación de «Row normaliser»:

Row normaliser

Step name:

Type field:

Fields

#	Fieldname	Type	new field
1	2007-S1	2007-S1	Value
2	2007-S2	2007-S2	Value
3	2008-S1	2008-S1	Value
4	2008-S2	2008-S2	Value
5	2009-S1	2009-S1	Value
6	2009-S2	2009-S2	Value
7	2010-S1	2010-S1	Value
8	2010-S2	2010-S2	Value
9	2011-S1	2011-S1	Value
10	2011-S2	2011-S2	Value
11	2012-S1	2012-S1	Value
12	2012-S2	2012-S2	Value
13	2013-S1	2013-S1	Value
14	2013-S2	2013-S2	Value
15	2014-S1	2014-S1	Value

Execution Results

<div> Logging Execution History Step Metrics Performance Graph Metrics Preview data </div>									
<input checked="" type="radio"/> First rows <input type="radio"/> Last rows <input type="radio"/> Off									
#	freq	product	consom	unit	tax	currency	geo\TIME_PERIOD	period	Value
7	S	6000	4161901	KWH	I_TAX	EUR	AL	2010-S1	:
8	S	6000	4161901	KWH	I_TAX	EUR	AL	2010-S2	:
9	S	6000	4161901	KWH	I_TAX	EUR	AL	2011-S1	0.1152
10	S	6000	4161901	KWH	I_TAX	EUR	AL	2011-S2	0.1157
11	S	6000	4161901	KWH	I_TAX	EUR	AL	2012-S1	0.1163
12	S	6000	4161901	KWH	I_TAX	EUR	AL	2012-S2	0.1167
13	S	6000	4161901	KWH	I_TAX	EUR	AL	2013-S1	0.1156

Una vez convertidos los periodos en filas y con el valor de «Value» («PELEC») para cada uno de ellos, vemos que los valores nulos tienen la expresión «:». Antes de reemplazar estos valores, nos aseguraremos de que no existen otros caracteres con la transformación «Replace in string». En este fichero, a diferencia de los anteriores, veremos que existe el valor «e» en algunos registros dentro del campo de «Value», el cual quedará eliminado gracias a esta transformación:

<div> Replace in string <div>Step name: Replace in string</div> </div>										
Fields string										
#	In stream field	Out stream field	use RegEx	Search	Replace with	Set empty string?	Replace with field	Whole Word	Case sensitive	Is Unicode
1	Value		Y	[a-zA-Z/\s+ /g]		N		N	N	N
2	period		Y	-		N		N	N	N

Y ahora sí podremos proceder a reemplazar el símbolo «:», por nulos mediante la transformación «Null if» sobre el campo que acabamos de crear «Value»:

<div> Null if <div>Step name: Null if</div> </div>		
Fields:		
#	Name	Value to turn to NULL
1	Value	:

Finalmente, ya podremos realizar la transformación del tipo de datos con el componente «Select values». Aprovecharemos para:

- Cambiar el nombre de los campos «geo» y «Value» como se muestra en la imagen.
- Repasaremos que todos los campos son de tipo «string», a excepción de «Value» (PELEC), que será decimal, y «product» y «consom», que

serán «integer».

- Indicaremos decimales (.) para el campo de «Value» y especificaremos la precisión y formato en caso de que sea necesario.

Select values

Step name

Select & Alter Remove Meta-data

Fields to alter the meta-data for :

#	Fieldname	Rename to	Type	Length	Precision	Binary t
1	freq		String			N
2	product		Integer			N
3	consom		Integer			N
4	unit		String			N
5	tax		String			N
6	currency		String			N
7	geo\TIME_PERIOD	geo	String			N
8	period		String	7		N
9	Value	pgas	Number		4	N

Por último, cargaréis los datos en la tabla intermedia del stage, utilizando el paso «Table output». Para este paso es necesario especificar la conexión de la base de datos, que la podéis generar dentro del propio componente, pulsando el botón «New» e indicándole que queréis conectaros a Microsoft SQL Server y utilizar el acceso nativo con vuestras credenciales. También podréis utilizar las variables de entorno creadas previamente.

Database Connection

General
Advanced
Options
Pooling
Clustering

Connection name:

Connection type:
 MS SQL Server
 MS SQL Server (Native)
 MariaDB
 MaxDB (SAP DB)
 MonetDB
 MySQL
 Native Mondrian
 Neoview
 Netezza
 Oracle
 Oracle RDB
 Palo MOLAP Server
 Pentaho Data Services
 PostgreSQL

Access:
 Native (JDBC)
 ODBC
 JNDI

El paso de cargar los datos a la tabla intermedia del *stage* lo configuraréis como se os indica en el menú principal que veis a continuación. Es importante que tengáis la tabla de STG creada para poder insertar los datos preprocesados.

Table output

Step name: Insert into SQL Server

Connection: cn_stage [Edit...] [New...] [W...]

Target schema: dbo [Browse]

Target table: STG_Pelec [Browse]

Commit size: 1000

Truncate table: ☒

Ignore insert errors: ☐

Specify database fields: ☒

Main options | Database fields

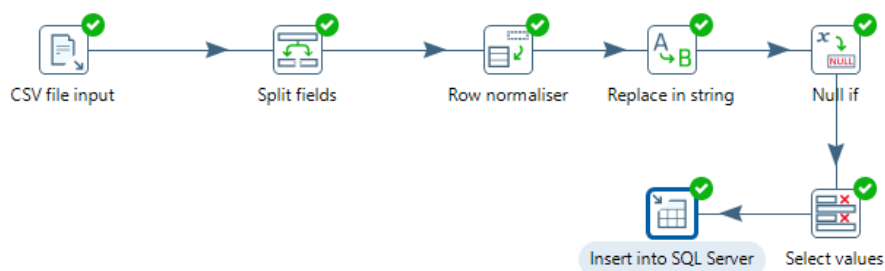
Fields to insert:

#	Table field	Stream field
1	freq	freq
2	product	product
3	consom	consom
4	unit	unit
5	tax	tax
6	currency	currency
7	geo	geo
8	period	period
9	pelec	pelec

[Get] [Enter field map]

Como veis en la imagen, para dejar la transformación preparada para posibles reprocesos, es necesario realizar un borrado previo para actualizar los datos en el caso de que tuvierais que efectuarlo. Para esto, activaréis el *check* «Truncate table» situado en los campos de la base de datos.

El proceso de la transformación completa es el siguiente:



Y podemos comprobar en la base de datos el resultado del proceso realizado:

```
SELECT TOP (1000) [freq]
      ,[product]
      ,[consom]
      ,[unit]
      ,[tax]
      ,[currency]
      ,[geo]
      ,[period]
      ,[pelec]
FROM [SOURCE_loginuoc].[dbo].[STG_PELEC]
```

	freq	product	consom	unit	tax	currency	geo	period	pelec
10	S	6000	4161901	KWH	I_TAX	NAC	FR	2010S2	0.2397
11	S	6000	4161901	KWH	I_TAX	NAC	LU	2020S1	0.3550
12	S	6000	4161901	KWH	I_TAX	NAC	SI	2016S2	0.1863
13	S	6000	4161901	KWH	I_TAX	PPS	BG	2011S1	0.1773
14	S	6000	4161901	KWH	I_TAX	PPS	EU27_2020	2017S2	0.3490

Transformación IN_COUNTRY

Fuente: countries.json

Destino: STG_COUNTRY (SQL Server)

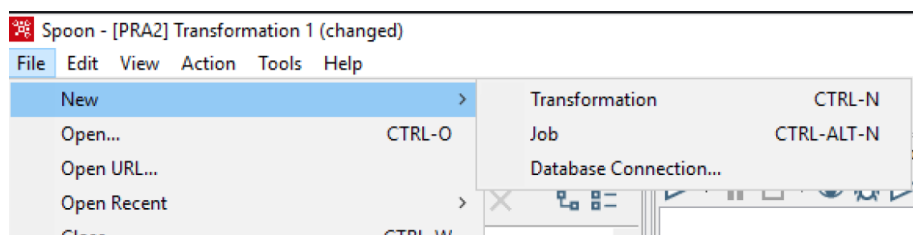
A continuación, se describe parte del desarrollo de la transformación de «IN_COUNTRY» mediante Spoon. El objetivo es cargar uno de los orígenes de los datos identificados, «countries.json», en la tabla STG_COUNTRY del área intermedia (*staging area*). La tabla intermedia tendrá que haber sido creada con anterioridad en la base de datos analítica, cuyo *script* se habrá escrito en el apartado de «creación».

La transformación de «IN_COUNTRY» contiene las siguientes etapas:

- lectura del fichero .json,
- carga a la tabla intermedia «STG_COUNTRY».

A continuación, se detallan las diferentes etapas que implementaréis para poder realizar la carga de datos.

En primer lugar, se crea una nueva transformación.



En esta, como primer paso, realizaréis la entrada del fichero .json. Para llevarlo a cabo utilizaréis el tipo «JSON input». En este paso deberéis indicar el fichero desde donde extraéis los datos. Podéis utilizar la variable de entorno «DIR_IN» o introducir la ruta directamente.

JSON input

Step name: Read countries.json

Get fields

Field is from a previous step: ☐

Select field:

Use field as file names: ☐

Read source as URL: ☐

Do not pass field downstream: ☐

File or directory: Add

Regular Expression

Exclude Regular Expression

Selected files:

#	File/Directory	Wildcard (RegExp)	Exclude wildcard	Required	Include subfol
1	\$(DIR_IN)\countries.json			N	N

Show filename(s)...

En este paso, tendréis que indicar los campos que vais a tratar con el botón «Get fields» y completaráis su definición.

JSON input

Step name: Read countries.json

#	Name	Path	Type	Format	Length	Precision	Currency	Decimal	Group	Trim type	Repeat
1	name	\$..name	String							both	N
2	code	\$..code	String							both	N

Podéis realizar una visualización previa de los datos que se cargarán con el botón «Preview».

Examine preview data

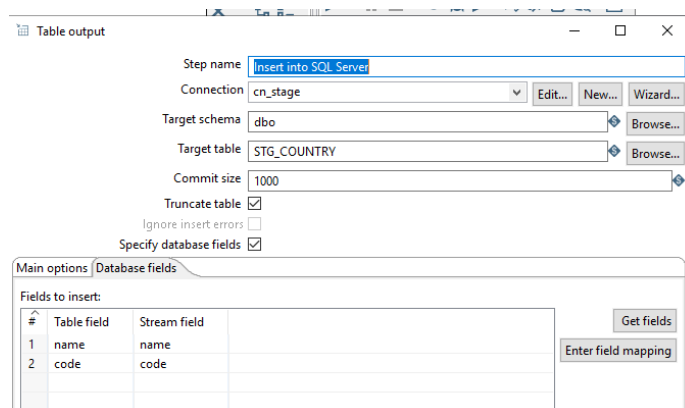
Rows of step: Read countries.json (243 rows)

#	name	code
1	Afghanistan	AF
2	Åland Islands	AX
3	Albania	AL
4	Algeria	DZ
5	American Samoa	AS
6	Andorra	AD
7	Angola	AO
8	Anguilla	AI

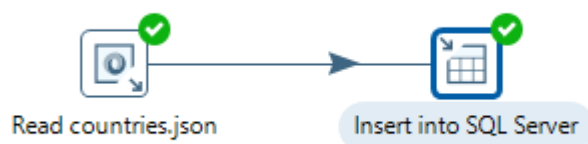
Dado que en este fichero countries.json no tenemos ningún valor nulo, no será necesario realizar la gestión de los mismos.

Por último, ya se puede realizar la carga en la tabla intermedia del *stage*, utilizando el componente «Table output».

En este paso se debe crear la conexión en la base de datos (como se ha hecho en transformaciones anteriores), hacer el mapeo de los campos e indicar que antes de realizar el *insert* se debe hacer un *truncate* de la tabla. A continuación, se puede observar esta configuración.



El proceso de la transformación completa es el siguiente:



Y podemos comprobar en la base de datos el resultado del proceso realizado:

```
SELECT TOP (1000) [name]
, [code]
FROM [SOURCE_loginuoc].[dbo].[STG_COUNTRY]
```

name	code
Afghanistan	AF
Aland Islands	AX
Albania	AL
Algeria	DZ
American Samoa	AS

Transformación IN_COICOP

Fuente: COICOP.xml

Destino: STG_COICOP (SQL Server)

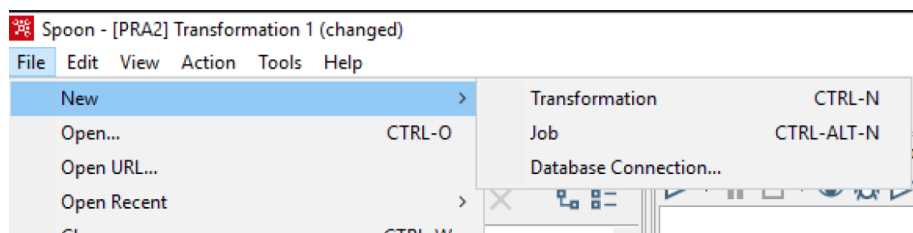
A continuación, se describe parte del desarrollo de la transformación de «IN_COICOP» mediante Spoon. El objetivo es cargar uno de los orígenes de los datos identificados, «COICOP.xml», en la tabla STG_COICOP del área intermedia (*staging area*). La tabla intermedia tendrá que haber sido creada con anterioridad en la base de datos analítica, cuyo *script* se habrá escrito en el apartado de «creación».

La transformación de «IN_COICOP» contiene las siguientes etapas:

- lectura del fichero .xml,
- filtración de valores nulos,
- carga a la tabla intermedia «STG_COICOP».

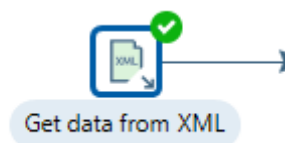
A continuación, se detallan las diferentes etapas que implementaréis para poder realizar la carga de datos.

En primer lugar, se crea una nueva transformación.



En esta, como primer paso, realizaréis la entrada del fichero .xml. Para llevarlo a

cabo utilizaréis la transformación «Get data from XML». En este paso deberéis indicar el fichero desde donde extraéis los datos. Podéis utilizar la variable de entorno «DIR_IN» o introducir la ruta directamente.



Get data from XML

Step name: Get data from XML

File Content Fields Additional output fields

XML source from field

XML source is defined in a field? ☐

XML source is a filename? ☐

Read source as Url ☐

get XML source from a field

File or directory Add Browse

Regular Expression

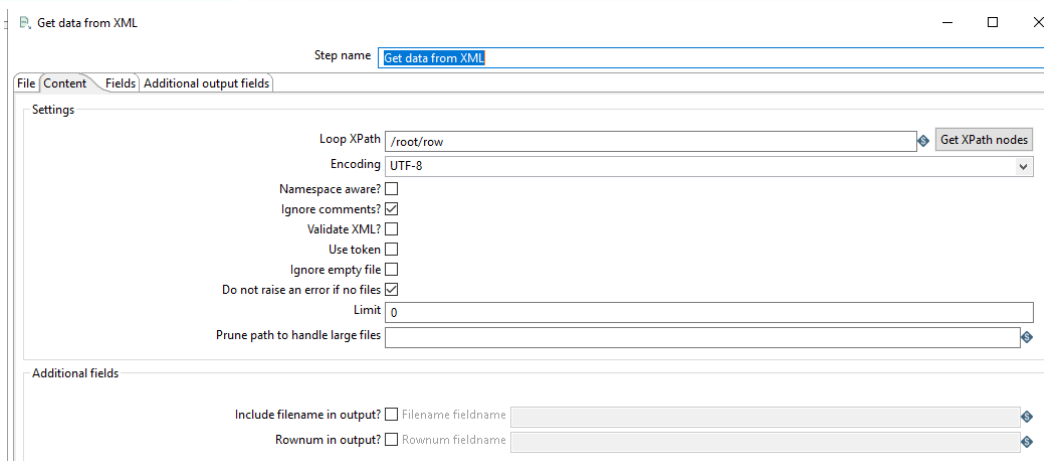
Exclude Regular Expression

Selected files:

#	File/Directory	Wildcard (RegExp)	Exclude wildcard	Required	Include subfolders
1	\${DIR_IN}\COICOP.xml			N	N
2				N	N

Delete Edit

En este paso, tendréis indicar por un lado el «Loop XPath» y deberemos ignorar los posibles comentarios que haya dentro del fichero. En la imagen podéis ver la configuración que acabamos de comentar:



Step name: **Get data from XML**

File | Content | **Fields** | Additional output fields

Settings

Loop XPath: Get XPath nodes

Encoding:

Namespace aware? ☐

Ignore comments? ☒

Validate XML? ☐

Use token ☐

Ignore empty file ☐

Do not raise an error if no files ☒

Limit:

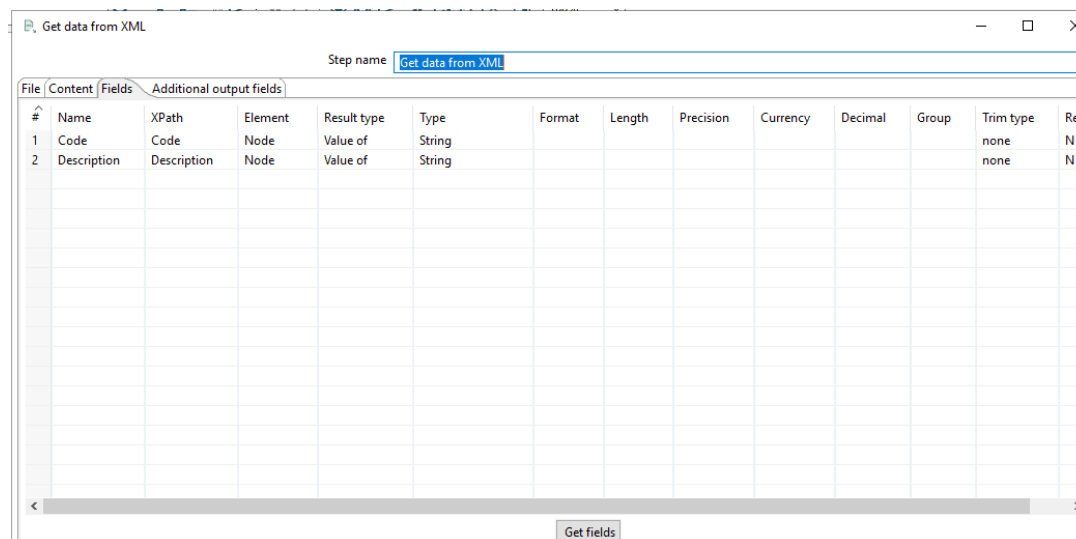
Prune path to handle large files:

Additional fields

Include filename in output? ☐ Filename fieldname:

Rownum in output? ☐ Rownum fieldname:

Seguidamente, deberemos seleccionar que campos queremos traernos a Spoon mediante el botón de «Get fields»:




Step name: **Get data from XML**

File | Content | **Fields** | Additional output fields

#	Name	XPath	Element	Result type	Type	Format	Length	Precision	Currency	Decimal	Group	Trim type	Rep
1	Code	Code	Node	Value of	String							none	N
2	Description	Description	Node	Value of	String							none	N

Get fields

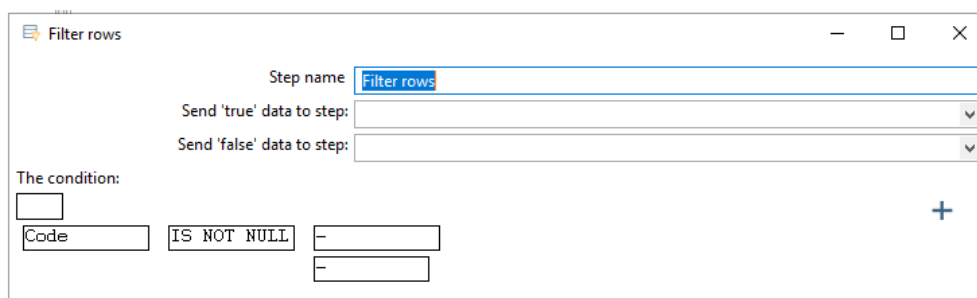
Por último, y para terminar este paso, podéis realizar una visualización previa de los datos que se cargarán con el botón «Preview».

 Examine preview data

Rows of step: Get data from XML (100 rows)

#	Code	Description
1	TOT_X_CP041_042	Total except actual rents
2	CP01113	Bread
3	CP01116	Pasta products and couscous
4	CP01118	Other cereal products
5	CP01126_01127	Dried, salted or smoked meat and edible meat offal
6	CP01128A	Other preserved or processed meat and meat preparations
7	CP01128B	Other fresh, chilled or frozen edible meat
8	CP01131_01132	Fresh, chilled or frozen fish
9	CP01133_01134	Fresh, chilled or frozen seafood
10	CP01135	Dried, smoked or salted fish and seafood
11	CP01136	Other preserved or processed fish and seafood and fish and seafood preparations
12	CP0116A	Citrus fruits (fresh, chilled or frozen)
13	CP0116B	Bananas (fresh, chilled or frozen)

Si hemos analizado el fichero .xml previamente, deberíamos haber visto que tiene algunas filas con valores nulos. Dado que la tabla que hemos creado en nuestra base de datos SQL (STG_COICOP) no admite valores nulos, deberemos filtrarlos mediante el paso «Filter rows»:



Finalmente, ya se puede realizar la carga en la tabla intermedia del stage, utilizando el componente «Table output».

En este paso se debe crear la conexión en la base de datos (como se ha hecho en transformaciones anteriores), hacer el mapeo de los campos e indicar que antes de realizar el *insert* se debe hacer un *truncate* de la tabla. A continuación, se puede observar esta configuración.

Table output

Step name:

Connection:

Target schema:

Target table:

Commit size:

Truncate table: ☒

Ignore insert errors: ☐

Specify database fields: ☒

Main options Database fields

Fields to insert:

#	Table field	Stream field
1	Code	Code
2	Description	Description

El proceso de la transformación completo es el siguiente:



Y podemos comprobar en la base de datos el resultado del proceso realizado:

```
SELECT TOP (1000) [code]
, [Description]
FROM [SOURCE_loginuoc].[dbo].[STG_COICOP]
```

code	Description
TOT_X_CP041_042	Total except actual rents
CP01113	Bread
CP01116	Pasta products and couscous
CP01118	Other cereal products
CP01126_01127	Dried, salted or smoked meat and edible meat offal
CP01128A	Other preserved or processed meat and meat prepa...
CP01128B	Other fresh, chilled or frozen edible meat

Transformación IN_CONSUMPTION

Fuente: consumption_band.csv

Destino: STG_CONSUMPTION (SQL Server)

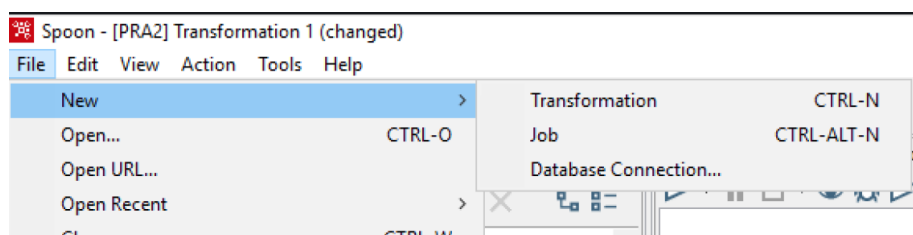
A continuación, se describe parte del desarrollo de la transformación de «IN_CONSUMPTION» mediante Spoon. El objetivo es cargar uno de los orígenes de los datos identificados, «consumption_band.csv», en la tabla «STG_CONSUMPTION» del área intermedia (*staging area*). La tabla intermedia tendrá que haber sido creada con anterioridad en la base de datos analítica, cuyo *script* se habrá escrito en el apartado de «creación».

La transformación de «IN_CONSUMPTION » contiene las siguientes etapas:

- lectura del fichero .csv,
- carga a la tabla intermedia «STG_CONSUMPTION».

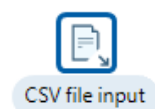
A continuación, se detallan las diferentes etapas que implementaréis para poder realizar la carga de datos.

En primer lugar, se crea una nueva transformación.



En esta, como primer paso, realizaréis la entrada del fichero .csv. Para llevarlo a cabo utilizaréis el tipo «CSV input». En este paso deberéis indicar el fichero desde donde extraéis los datos. Podéis utilizar la variable de entorno «DIR_IN» o introducir la ruta directamente.

Asimismo, en cuanto al tratamiento del fichero, el delimitador del .csv en este caso será de «;», como podemos apreciar en la imagen:



CSV file input

Step name: CSV file input

Filename: \${DIR_IN}\consumption_band.csv Browse...

Delimiter: ; Insert TAB

Enclosure: "

NIO buffer size: 50000

Lazy conversion? ☐

Header row present? ☒

Add filename to result ☐

The row number field name (optional)

Running in parallel? ☐

New line possible in fields? ☐

Format: mixed

File encoding

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Trim type
1	Code	Integer	#	15	0	\$,	.	none
2	Description	String		46		\$,	.	none

Podemos previsualizar los resultados y veremos que el fichero contiene dos columnas:



Examine preview data

Rows of step: CSV file input (8 rows)

#	Code	Description
1	4161901	Band DA : Consumption < 1 000 kWh
2	4161902	Band DB : 1 000 kWh < Consumption < 2 500 kWh
3	4161903	Band DC : 2 500 kWh < Consumption < 5 000 kWh
4	4161904	Band DD : 5 000 kWh < Consumption < 15 000 kWh
5	4161905	Band DE : Consumption > 15 000 kWh
6	4141901	Band D1 : Consumption < 20 GJ
7	4141902	Band D2 : 20 GJ < Consumption < 200 GJ
8	4141903	Band D3 : Consumption > 200 GJ

Dado que en este fichero «consumption_band.csv» no tenemos ningún valor nulo, no será necesario realizar la gestión de los mismos.

Por último, ya podremos realizar la carga en la tabla intermedia del stage, utilizando el componente «Table output».

En este paso se debe crear la conexión en la base de datos (como se ha hecho en transformaciones anteriores), hacer el mapeo de los campos e indicar que antes de realizar el *insert* se debe hacer un *truncate* de la tabla. A continuación, se puede observar esta configuración.

Table output

Step name: Insert into SQL Server

Connection: cn_stage [Edit...] [New...] [Wizard...]

Target schema: dbo [Browse...]

Target table: STG_CONSUMPTION [Browse...]

Commit size: 1000

Truncate table: ☒

Ignore insert errors: ☐

Specify database fields: ☒

Main options Database fields

Fields to insert:

#	Table field	Stream field
1	Code	Code
2	Description	Description

[Get fields] [Enter field mapping]

El proceso de la transformación completa es el siguiente:



Y podemos comprobar en la base de datos el resultado del proceso realizado:

```

SELECT TOP (1000) [code]
, [Description]
FROM [SOURCE_loginuoc].[dbo].[STG_CONSUMPTION]
  
```

code	Description
4161901	Band DA : Consumption < 1 000 kWh
4161902	Band DB : 1 000 kWh < Consumption < 2 500 kWh
4161903	Band DC : 2 500 kWh < Consumption < 5 000 kWh
4161904	Band DD : 5 000 kWh < Consumption < 15 000 kWh

3.3.4. Bloque TR

El bloque TR contiene los procesos de ETL, que se encargan de la carga inicial de datos, desde las tablas intermedias pobladas, con los procesos del bloque IN, hasta el modelo multidimensional del almacén que habéis diseñado, compuesto por dimensiones y tablas de hechos.

Este bloque se divide, a su vez, en dos subbloques: por un lado, los procesos para la carga de dimensiones y, por el otro, los procesos para la carga de tablas de hechos. Esta división permite el retroceso de dichos procesos en caso de error y un mejor entendimiento de la implementación de los procesos.

Debido a que en Spoon, antes de realizar cualquier tipo de unión o intersección de dos flujos de datos, es necesario ordenarlos por el mismo campo clave, en esta solución se obviarán las transformaciones básicas de ordenación. Tampoco se detallarán las transformaciones de selección de campos o de modificación de tipos de campos.

3.3.5. Bloque TR_DIM

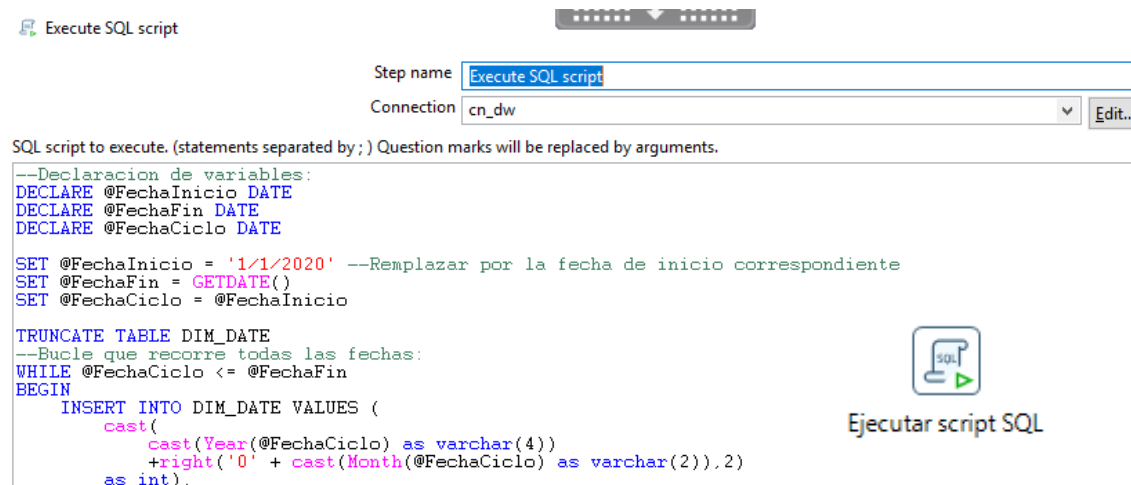
Este bloque contiene las transformaciones para la carga inicial de las dimensiones en el almacén desde las tablas intermedias «IN_» del staging area.

Se tendrá en cuenta que en una carga inicial pueden ejecutarse las transformaciones de carga de dimensiones las veces que sean necesarias.

Transformación TR_DIM_DATE

En esta transformación se cargará la dimensión «Fecha». La carga de esta dimensión es algo diferente a la carga de las dimensiones de datos. Hay distintas opciones para cargar las tablas de tiempo. En esta solución, dado que se creará una dimensión temporal sencilla, se utilizará un *script* SQL para generar todos los registros necesarios.

Será necesario indicar manualmente una fecha de inicio; la fecha de fin será el día que se ejecute el *script* SQL. En este caso, como se quiere tener el detalle por meses, el *script* de carga será rápido:



Execute SQL script

Step name: Execute SQL script

Connection: cn_dw

SQL script to execute. (statements separated by ;) Question marks will be replaced by arguments.

```
--Declaracion de variables:
DECLARE @FechaInicio DATE
DECLARE @FechaFin DATE
DECLARE @FechaCiclo DATE

SET @FechaInicio = '1/1/2020' --Reemplazar por la fecha de inicio correspondiente
SET @FechaFin = GETDATE()
SET @FechaCiclo = @FechaInicio

TRUNCATE TABLE DIM_DATE
--Bucle que recorre todas las fechas:
WHILE @FechaCiclo <= @FechaFin
BEGIN
    INSERT INTO DIM_DATE VALUES (
        cast(
            cast(Year(@FechaCiclo) as varchar(4))
            +right('0' + cast(Month(@FechaCiclo) as varchar(2)),2)
            as int),
        cast(
            cast(Day(@FechaCiclo) as varchar(2))
            +right('0' + cast(Month(@FechaCiclo) as varchar(2)),2)
            as int),
        cast(
            cast(Year(@FechaCiclo) as varchar(4))
            +right('0' + cast(Month(@FechaCiclo) as varchar(2)),2)
            as int),
        cast(
            cast(Day(@FechaCiclo) as varchar(2))
            +right('0' + cast(Month(@FechaCiclo) as varchar(2)),2)
            as int)
    )
    SET @FechaCiclo = DATEADD(MONTH, 1, @FechaCiclo)
END
```

Ejecutar script SQL

El *script* SQL completo es el siguiente:

```
--Declaración de variables:
DECLARE @FechaInicio DATE
DECLARE @FechaFin DATE
DECLARE @FechaCiclo DATE

SET @FechaInicio = '1/1/2020' --Reemplazar por la fecha de inicio
correspondiente
SET @FechaFin = GETDATE()
SET @FechaCiclo = @FechaInicio
TRUNCATE TABLE DIM_DATE
--Bucle que recorre todas las fechas:
```

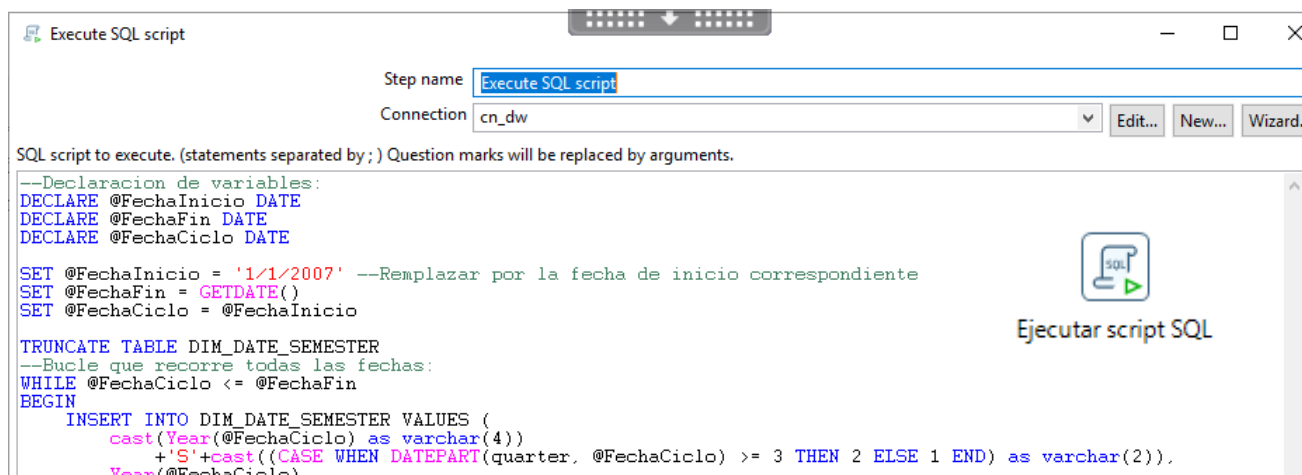
```

WHILE @FechaCiclo <= @FechaFin
BEGIN
    INSERT INTO DIM_DATE VALUES (
        cast(
            cast(Year(@FechaCiclo) as varchar(4))
            +right('0' + cast(Month(@FechaCiclo) as
varchar(2)),2)
            as int),
        Year(@FechaCiclo),
        Month(@FechaCiclo),
        datepart(quarter, @FechaCiclo)
    )
    -- Incrementar la FechaCiclo en un mes
    SET @FechaCiclo = DateAdd(m, 1, @FechaCiclo)
END

```

Transformación TR_DIM_DATE_SEMESTER

Para generar esta tabla de dimensiones, y al igual que en la tabla de dimensiones de «DIM_DATE», hay que ejecutar un *script* SQL donde deberemos indicar la fecha de inicio. En este caso, se quiere tener el detalle por semestres:



El *script* SQL completo es el siguiente:

```

--Declaración de variables:
DECLARE @FechaInicio DATE
DECLARE @FechaFin DATE
DECLARE @FechaCiclo DATE

SET @FechaInicio = '1/1/2007' --Reemplazar por la fecha de inicio
correspondiente

SET @FechaFin = GETDATE()
SET @FechaCiclo = @FechaInicio

TRUNCATE TABLE DIM_DATE_SEMESTER
--Bucle que recorre todas las fechas:
WHILE @FechaCiclo <= @FechaFin
BEGIN
    INSERT INTO DIM_DATE_SEMESTER VALUES (
        cast(Year(@FechaCiclo) as varchar(4))
        + 'S'+cast((CASE WHEN DATEPART(quarter, @FechaCiclo) >= 3 THEN 2 ELSE 1 END) as varchar(2)),
        Year(@FechaCiclo)
    )
    -- Incrementar la FechaCiclo en un mes
    SET @FechaCiclo = DateAdd(m, 1, @FechaCiclo)
END

```

```
INSERT INTO DIM_DATE_SEMESTER VALUES (

    cast(Year(@FechaCiclo) as varchar(4))
    + 'S'+cast((CASE WHEN DATEPART(quarter, @FechaCiclo)
>= 3 THEN 2 ELSE 1 END) as varchar(2)),
    Year(@FechaCiclo),
    ('S'+cast((CASE WHEN DATEPART(quarter,@FechaCiclo) >= 3
THEN 2 ELSE 1 END) as varchar(2)))
)
-- Incrementar la FechaCiclo en seis meses
SET @FechaCiclo = DateAdd(m, 6, @FechaCiclo)
END
```

Transformación TR_DIM_COUNTRY

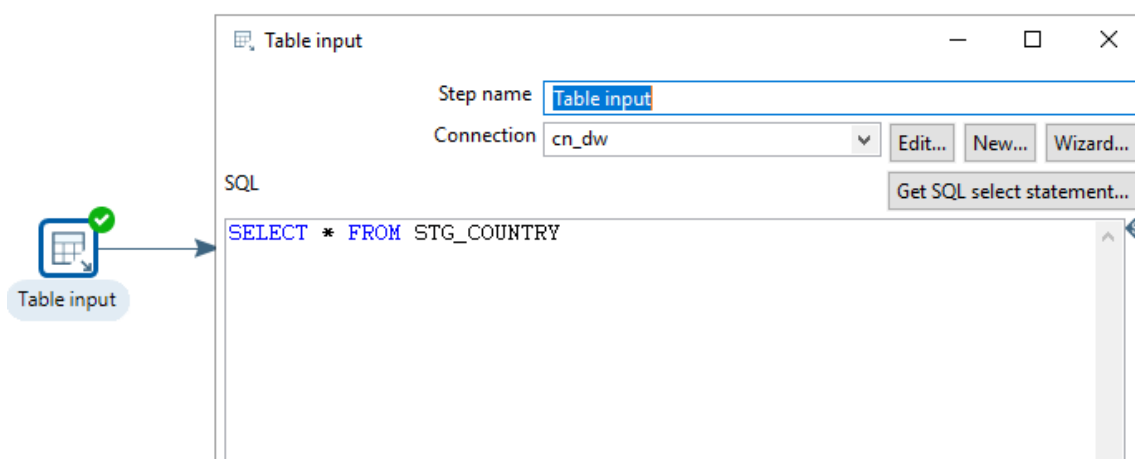
Fuente: STG_COUNTRY (SQL Server)

Destino: DIM_COUNTRY (SQL Server)

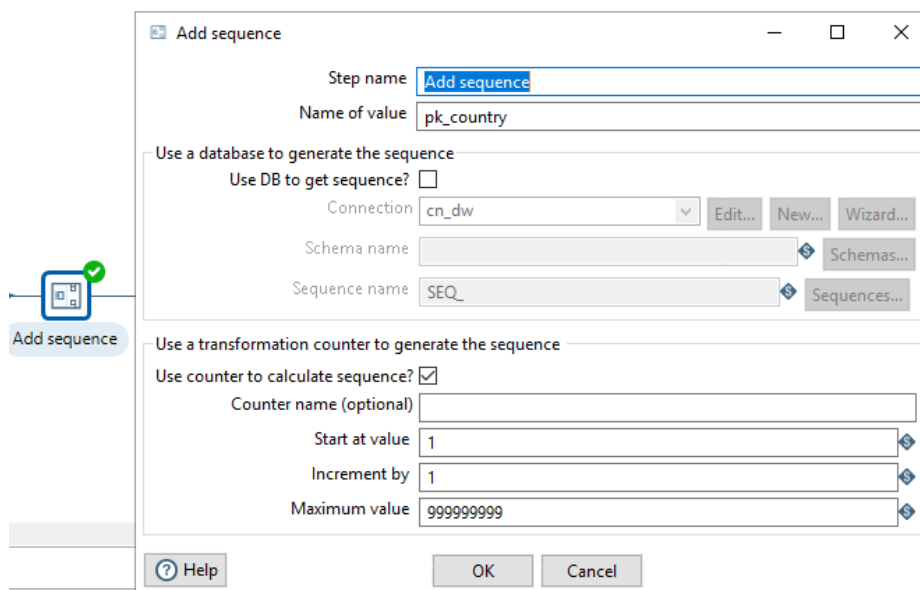
A continuación, se describe el desarrollo de la transformación de «TR_DIM_COUNTRY» mediante Spoon. En esta transformación se cargarán los datos disponibles en la tabla «STG_COUNTRY» en la tabla de dimensión «DIM_COUNTRY». Para hacerlo, seguiremos los siguientes tres pasos:

- leer los datos de «STG_COUNTRY»,
- añadir secuencia,
- insertar en la tabla «DIM_COUNTRY».

Para realizar la lectura se ha utilizado el componente «Table input», que permite realizar una *query* directamente en la tabla de la BBDD. En este caso, se quieren todos los campos.



Con los datos cargados en Spoon, se utiliza el componente «Add sequence» para añadir una secuencia que actuará como «PK» de la tabla.



Add sequence

Step name:

Name of value:

Use a database to generate the sequence

Use DB to get sequence? ☐

Connection:

Schema name:

Sequence name:

Use a transformation counter to generate the sequence

Use counter to calculate sequence? ☒

Counter name (optional):

Start at value:

Increment by:

Maximum value:

Seguidamente, se realiza la carga de los datos en la tabla de dimensión correspondiente. Deberéis mapear los campos de la tabla de *staging* con la dimensión final, ya que puede que haya algún cambio de nombre. En este caso, la columna «name» de «STG_COUNTRY» se llamará «country_name» en «DIM_COUNTRY».

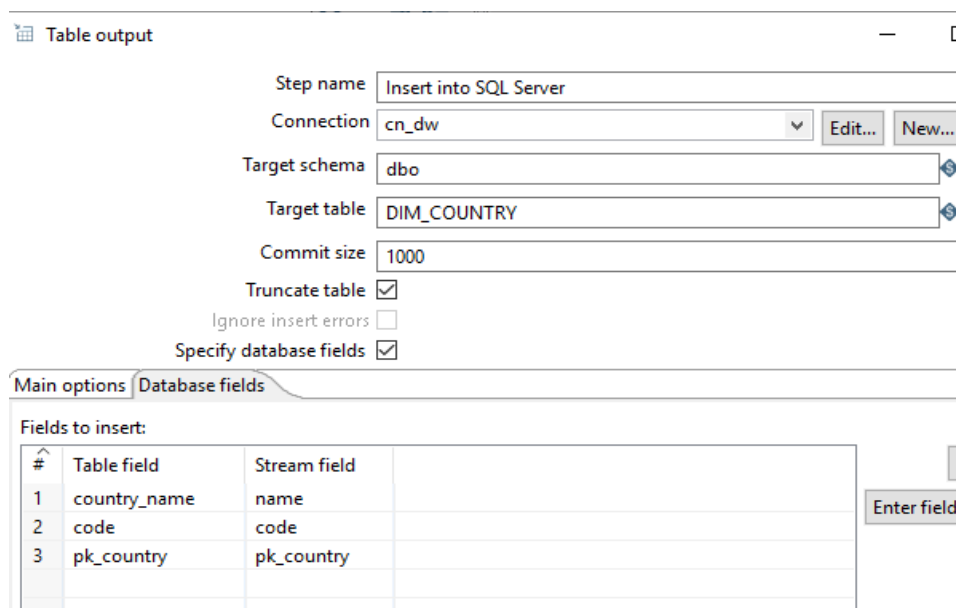


Table output

Step name:

Connection:

Target schema:

Target table:

Commit size:

Truncate table: ☒

Ignore insert errors: ☐

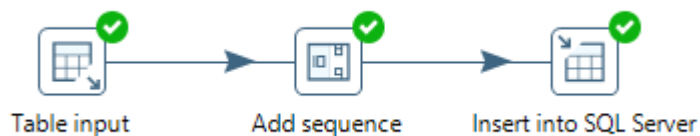
Specify database fields: ☒

Main options **Database fields**

Fields to insert:

#	Table field	Stream field
1	country_name	name
2	code	code
3	pk_country	pk_country

El proceso final de transformación quedaría de la siguiente manera:



Execution Results												
Logging Execution History Step Metrics Performance Graph Metrics Preview data												
#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected	Errors	Active	Time	Speed (r/s)
1	Table input	0	0	250	250	0	0	0	0	Finished	0.0s	10,870
2	Add sequence	0	250	250	0	0	0	0	0	Finished	0.0s	9,259
3	Insert into SQL Server	0	250	250	0	250	0	0	0	Finished	0.1s	4,237

Finalmente, comprobamos el resultado en la tabla de la base de datos:

```

SELECT TOP (1000) [pk_country]
, [code]
, [country_name]
FROM [SOURCE_loginuoc].[dbo].[DIM_COUNTRY]
  
```

100 %

Results Messages

	pk_country	code	country_name
1	1	AF	Afghanistan
2	2	AX	Åland Islands
3	3	AL	Albania
4	4	DZ	Algeria
5	5	AS	American Samoa
6	6	AD	Andorra
7	7	AO	Angola

Transformación TR_DIM_COICOP

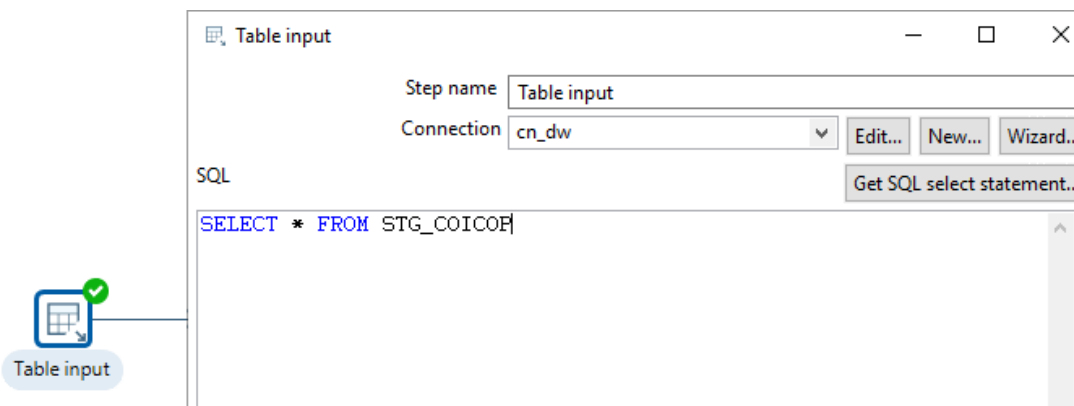
Fuente: STG_COICOP (SQL Server)

Destino: DIM_COICOP (SQL Server)

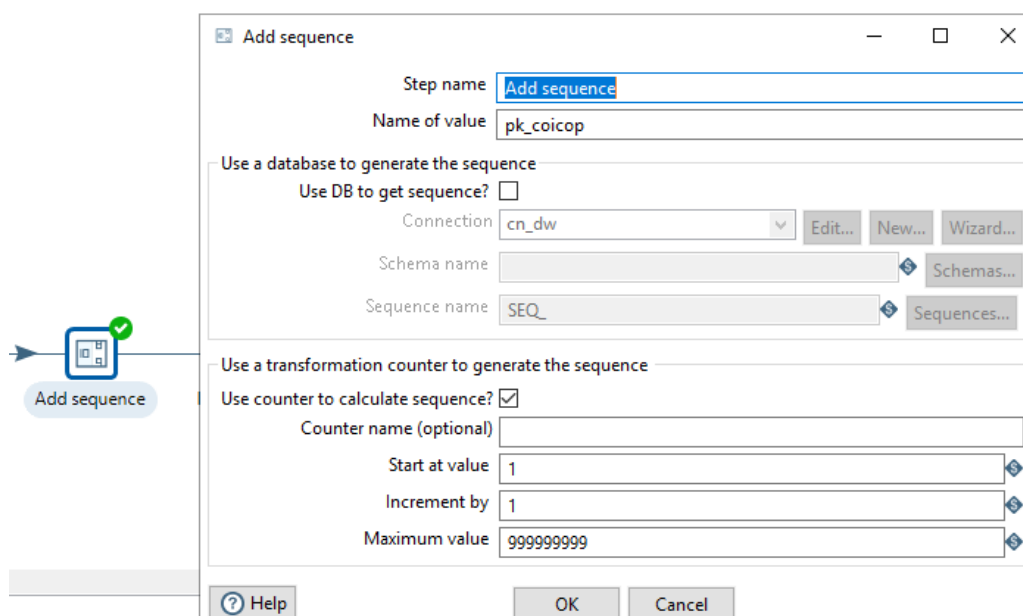
A continuación, se describe el desarrollo de la transformación de «TR_DIM_COICOP» mediante Spoon. En esta transformación se cargarán los datos disponibles en la tabla «STG_COICOP» en la tabla de dimensión «DIM_COICOP». Para hacerlo, seguiremos los siguientes tres pasos:

- leer los datos de «STG_COICOP»,
- añadir secuencia,
- insertar en la tabla «DIM_COICOP».

Para realizar la lectura se ha utilizado el componente «Table input», que permite realizar una *query* directamente en la tabla de la BBDD. En este caso, se quieren todos los campos.



Con los datos cargados en Spoon, se utiliza el componente «Add sequence» para añadir una secuencia que actuará como «PK» de la tabla.



Seguidamente, se realiza la carga de los datos en la tabla de dimensión correspondiente. Deberéis mapear los campos de la tabla de *staging* con la dimensión final, ya que puede que haya algún cambio de nombre. En este caso, la columna «description» de «STG_COICOP» se llamará «coicop_name» en «DIM_COICOP».

Table output

Step name: Insert into SQL Server

Connection: cn_dw [Edit...] [New...] [Wizard...]

Target schema: dbo [Browse...]

Target table: DIM_COICOP [Browse...]

Commit size: 1000

Truncate table: ☒

Ignore insert errors: ☐

Specify database fields: ☒

Main options Database fields

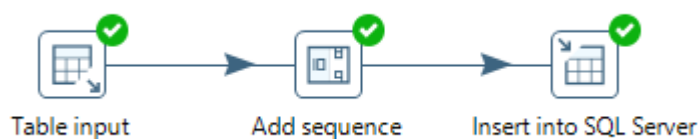
Fields to insert:

#	Table field	Stream field
1	code	code
2	coicop_name	Description
3	pk_coicop	pk_coicop

[Get fields]

[Enter field mapping]

El proceso final de transformación quedaría de la siguiente manera:



Execution Results

Logging Execution History Step Metrics Performance Graph Metrics Preview data												
#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected	Errors	Active	Time	Speed (r/s)
1	Table input	0	0	951	951	0	0	0	0	Finished	0.0s	73,154
2	Add sequence	0	951	951	0	0	0	0	0	Finished	0.0s	43,227
3	Insert into SQL Server	0	951	951	0	951	0	0	0	Finished	0.1s	11,188

Finalmente, comprobamos el resultado en la tabla de la base de datos:

```
SELECT TOP (1000) [pk_coicop]
, [code]
, [coicop_name]
FROM [SOURCE_loginuoc].[dbo].[DIM_COICOP]
```

pk_coicop	code	coicop_name
1	TOT_X_CP041_042	Total except actual rents
2	CP01113	Bread
3	CP01116	Pasta products and couscous
4	CP01118	Other cereal products
5	CP01126_01127	Dried, salted or smoked meat and edible meat offal

Transformación TR_DIM_CURRENCY

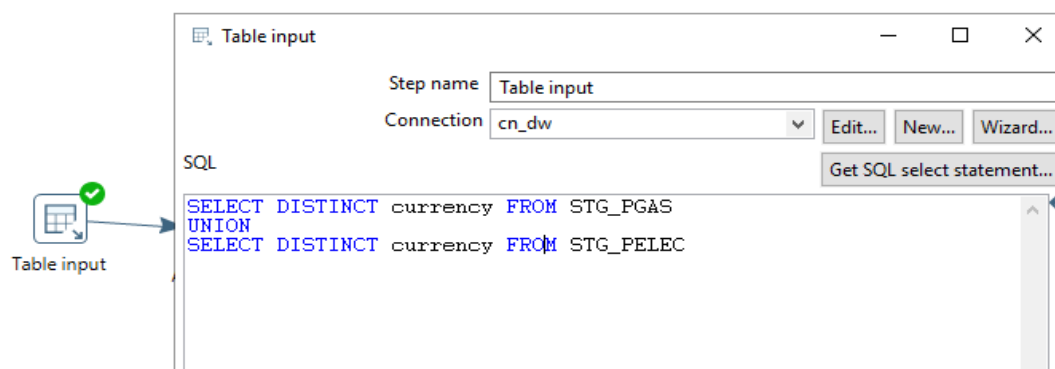
Fuente: STG_PGAS, STG_Pelec (SQL Server)

Destino: DIM_CURRENCY (SQL Server)

A continuación, se describe el desarrollo de la transformación de «TR_DIM_CURRENCY» mediante Spoon. En esta transformación se cargarán los datos disponibles en las tablas «STG_PGAS» y «STG_Pelec» en la tabla de dimensión «DIM_CURRENCY». Para hacerlo, seguiremos los siguientes tres pasos:

- leer los datos de «STG_PGAS» y «STG_Pelec»,
- añadir secuencia,
- insertar en la tabla «DIM_CURRENCY».

Para realizar la lectura se ha utilizado el componente «Table input», que permite realizar una *query* directamente en la tabla de la BBDD. Vemos que en este caso hemos obtenido todos los valores distintos del campo «currency» de las tablas «STG_PGAS» y «STG_Pelec» y los hemos unido en una misma «Table input».

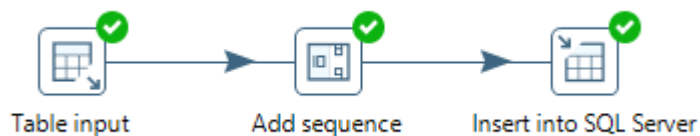


Con los datos cargados en Spoon, se utiliza el componente «Add sequence» para añadir una secuencia que actuará como «PK» de la tabla.

Seguidamente, se realiza la carga de los datos en la tabla de dimensión correspondiente. Deberéis mapear los campos de la tabla de *staging* con la dimensión final, ya que puede que haya algún cambio de nombre. En este caso, la columna «currency» de las tablas «STG_PGAS» y «STG_Pelec» se llamará «currency_name» en «DIM_TAX».

#	Table field	Stream field
1	currency_name	currency
2	pk_currency	pk_currency

El proceso final de transformación quedaría de la siguiente manera:



Finalmente, comprobamos el resultado en la tabla de la base de datos:

```

SELECT TOP (1000) [pk_currency]
, [currency_name]
FROM [SOURCE_loginuoc].[dbo].[DIM_CURRENCY]
  
```

	pk_currency	currency_name
1	1	EUR
2	2	NAC
3	3	PPS

Transformación TR_DIM_TAX

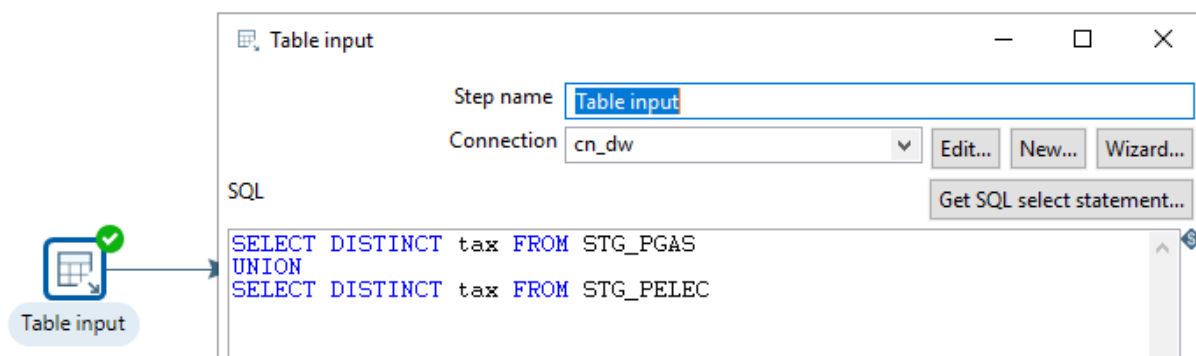
Fuente: STG_PGAS, STG_PELEC (SQL Server)

Destino: DIM_TAX (SQL Server)

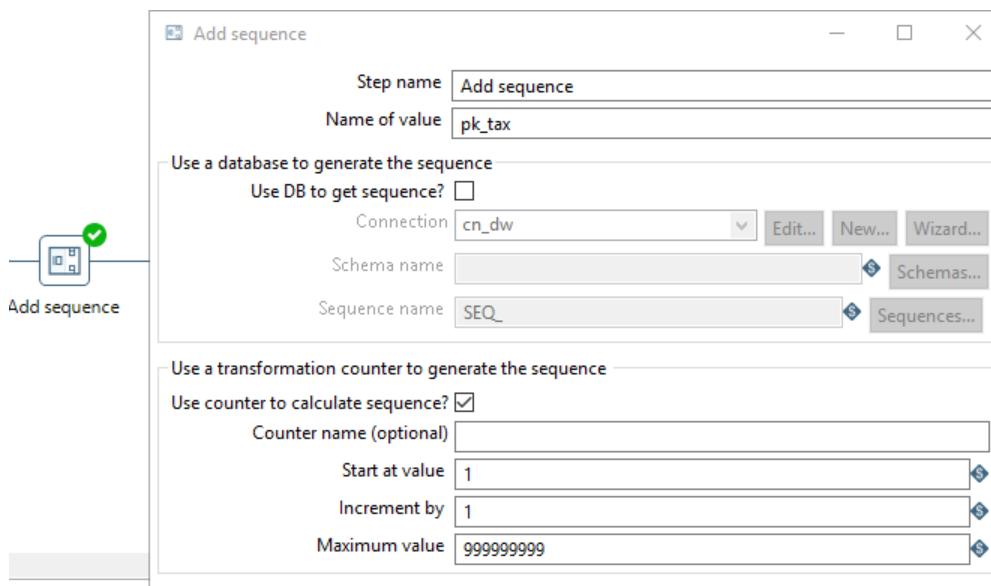
A continuación, se describe el desarrollo de la transformación de «TR_DIM_TAX» mediante Spoon. En esta transformación se cargarán los datos disponibles en las tablas «STG_PGAS» y «STG_PELEC» en la tabla de dimensión «DIM_TAX». Para hacerlo, seguiremos los siguientes tres pasos:

- leer los datos de «STG_PGAS» y «STG_PELEC»,
- añadir secuencia,
- insertar en la tabla «DIM_TAX».

Para realizar la lectura se ha utilizado el componente «Table input», que permite realizar una *query* directamente en la tabla de la BBDD. Vemos que en este caso hemos obtenido todos los valores distintos del campo «tax» de las tablas «STG_PGAS» y «STG_PELEC» y los hemos unido en una misma «Table input».



Con los datos cargados en Spoon, se utiliza el componente «Add sequence» para añadir una secuencia que actuará como «PK» de la tabla.



Seguidamente, se realiza la carga de los datos en la tabla de dimensión correspondiente. Deberéis mapear los campos de la tabla de *staging* con la dimensión final, ya que puede que haya algún cambio de nombre. En este caso, la columna «tax» de las tablas «STG_PGAS» y «STG_Pelec» se llamará «tax_name» en «DIM_TAX».

Table output

Step name:

Connection:

Target schema:

Target table:

Commit size:

Truncate table: ☒

Ignore insert errors: ☐

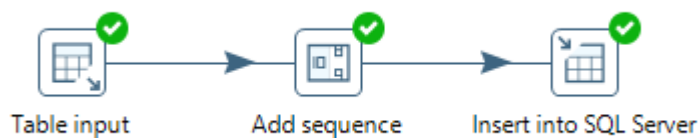
Specify database fields: ☒

Main options | **Database fields**

Fields to insert:

#	Table field	Stream field
1	tax_name	tax
2	pk_tax	pk_tax

El proceso final de transformación quedaría de la siguiente manera:



Finalmente, comprobamos el resultado en la tabla de la base de datos:

```

SELECT TOP (1000) [pk_tax]
, [tax_name]
FROM [SOURCE_loginuoc].[dbo].[DIM_TAX]
  
```

.00 %

Results | **Messages**

	pk_tax	tax_name
1	1	I_TAX
2	2	X_TAX
3	3	X_VAT

Transformación TR_DIM_UNIT

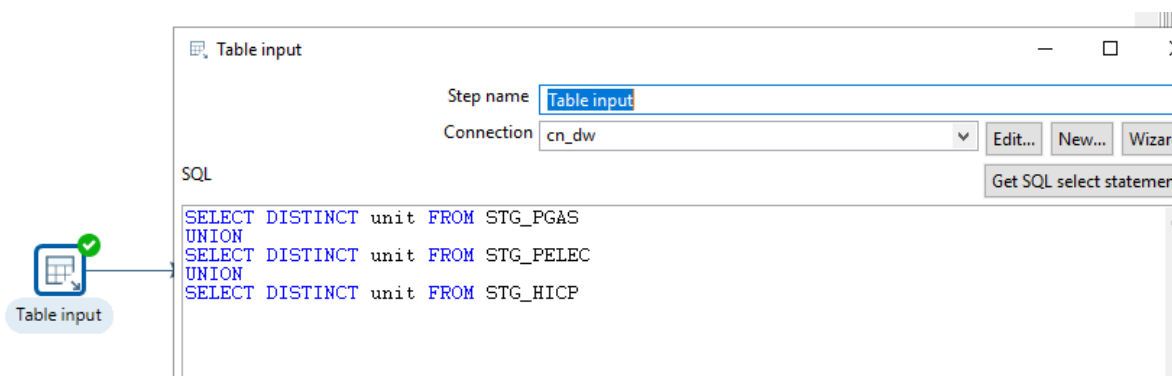
Fuente: STG_PGAS, STG_PEEC, STG_HICP (SQL Server)

Destino: DIM_UNIT (SQL Server)

A continuación, se describe el desarrollo de la transformación de «TR_DIM_UNIT» mediante Spoon. En esta transformación se cargarán los datos disponibles en las tablas «STG_PGAS», «STG_PEEC» y «STG_HICP» en la tabla de dimensión «DIM_UNIT». Para hacerlo, seguiremos los siguientes tres pasos:

- leer los datos de «STG_PGAS», «STG_PEEC» y «STG_HICP»,
- añadir secuencia,
- insertar en la tabla «DIM_UNIT».

Para realizar la lectura se ha utilizado el componente «Table input», que permite realizar una *query* directamente en la tabla de la BBDD. Vemos que en este caso hemos obtenido todos los valores distintos del campo «unit» de las tablas «STG_PGAS», «STG_PEEC» y «STG_HICP», y los hemos unido en una misma «Table input».



Con los datos cargados en Spoon, se utiliza el componente «Add sequence» para añadir una secuencia que actuará como «PK» de la tabla.

Add sequence

Step name: Add sequence

Name of value: pk_unit

Use a database to generate the sequence

Use DB to get sequence? ☐

Connection: cn_dw

Schema name:

Sequence name: SEQ_

Use a transformation counter to generate the sequence

Use counter to calculate sequence? ☒

Counter name (optional):

Start at value: 1

Increment by: 1

Maximum value: 999999999

Seguidamente, se realiza la carga de los datos en la tabla de dimensión correspondiente. Deberéis mapear los campos de la tabla de *staging* con la dimensión final, ya que puede que haya algún cambio de nombre. En este caso, la columna «unit» de las tablas «STG_PGAS», «STG_PELLEC» y «STG_HICP» se llamará «unit_name» en «DIM_UNIT».

Table output

Step name: Insert into SQL Server

Connection: cn_dw

Target schema: dbo

Target table: DIM_UNIT

Commit size: 1000

Truncate table: ☒

Ignore insert errors: ☐

Specify database fields: ☒

Main options | Database fields

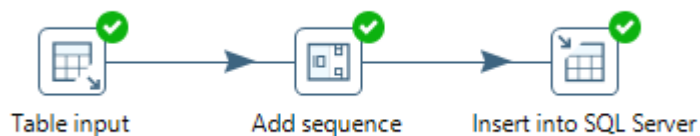
Fields to insert:

#	Table field	Stream field
1	unit_name	unit
2	pk_unit	pk_unit

Get fields

Enter field mapping

El proceso final de transformación quedaría de la siguiente manera:



Finalmente, comprobamos el resultado en la tabla de la base de datos:

```

SELECT TOP (1000) [pk_unit]
, [unit_name]
FROM [SOURCE_loginuoc].[dbo].[DIM_UNIT]
  
```

	pk_unit	unit_name
1	1	GJ_GCV
2	2	KWH
3	3	RCH_MV12MAVR

Transformación TR_DIM_PRODUCT

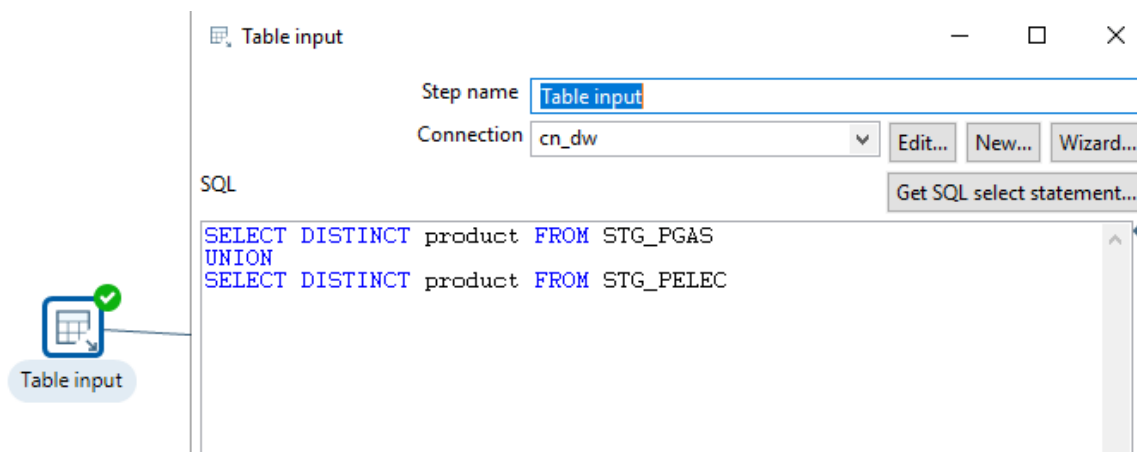
Fuente: STG_PGAS, STG_PEELEC (SQL Server)

Destino: DIM_PRODUCT (SQL Server)

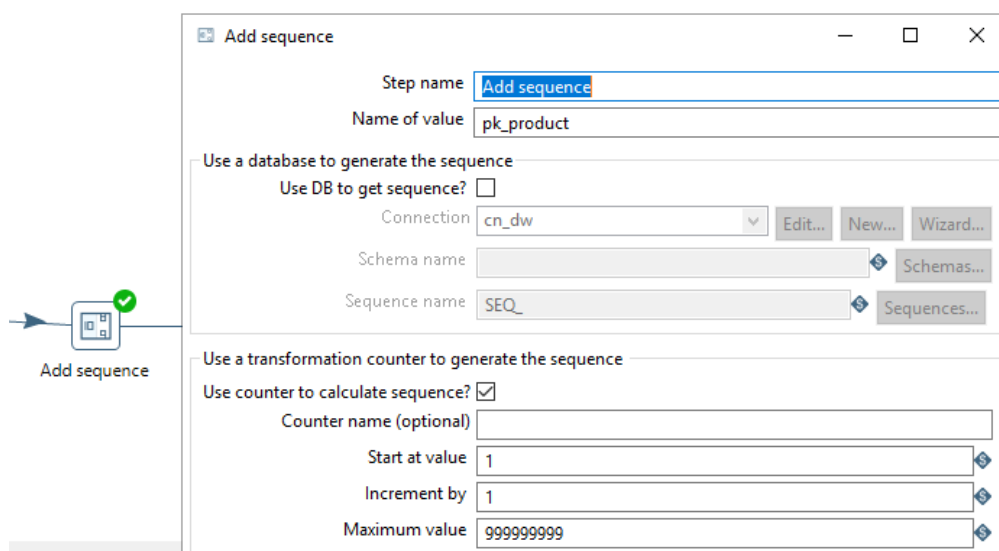
A continuación, se describe el desarrollo de la transformación de «TR_DIM_PRODUCT» mediante Spoon. En esta transformación se cargarán los datos disponibles en las tablas «STG_PGAS» y «STG_PEELEC» en la tabla de dimensión «DIM_PRODUCT». Para hacerlo, seguiremos los siguientes tres pasos:

- leer los datos de «STG_PGAS» y «STG_PEELEC»,
- añadir secuencia,
- insertar en la tabla «DIM_PRODUCT».

Para realizar la lectura se ha utilizado el componente «Table input», que permite realizar una *query* directamente en la tabla de la BBDD. Vemos que en este caso hemos obtenido todos los valores distintos del campo «product» de las tablas «STG_PGAS» y «STG_PEELEC» y los hemos unido en una misma «Table input».



Con los datos cargados en Spoon, se utiliza el componente «Add sequence» para añadir una secuencia que actuará como «PK» de la tabla.



Seguidamente, se realiza la carga de los datos en la tabla de dimensión correspondiente. Deberéis mapear los campos de la tabla de *staging* con la dimensión final, ya que puede que haya algún cambio de nombre. En este caso, la columna «product» de las tablas «STG_PGAS» y «STG_PEEC» se llamará «product_code» en «DIM_PRODUCT».

Table output

Step name: Insert into SQL Server

Connection: cn_dw [Edit... New... Wizard...]

Target schema: dbo [Browse...]

Target table: DIM_PRODUCT [Browse...]

Commit size: 1000

Truncate table: ☒

Ignore insert errors: ☐

Specify database fields: ☒

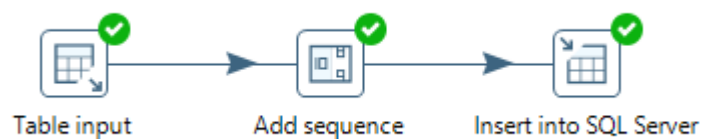
Main options | Database fields

Fields to insert:

#	Table field	Stream field
1	product_code	product
2	pk_product	pk_product

[Get fields] [Enter field mapping]

El proceso final de transformación quedaría de la siguiente manera:



Finalmente, comprobamos el resultado en la tabla de la base de datos:

```

SELECT TOP (1000) [pk_product]
, [product_code]
FROM [SOURCE_loginuoc].[dbo].[DIM_PRODUCT]
  
```

100 %

Results | Messages

	pk_product	product_code
1	1	4100
2	2	6000

Transformación TR_DIM_CONSUMPTION

Fuente: STG_CONSUMPTION (SQL Server)

Destino: DIM_CONSUMPTION (SQL Server)

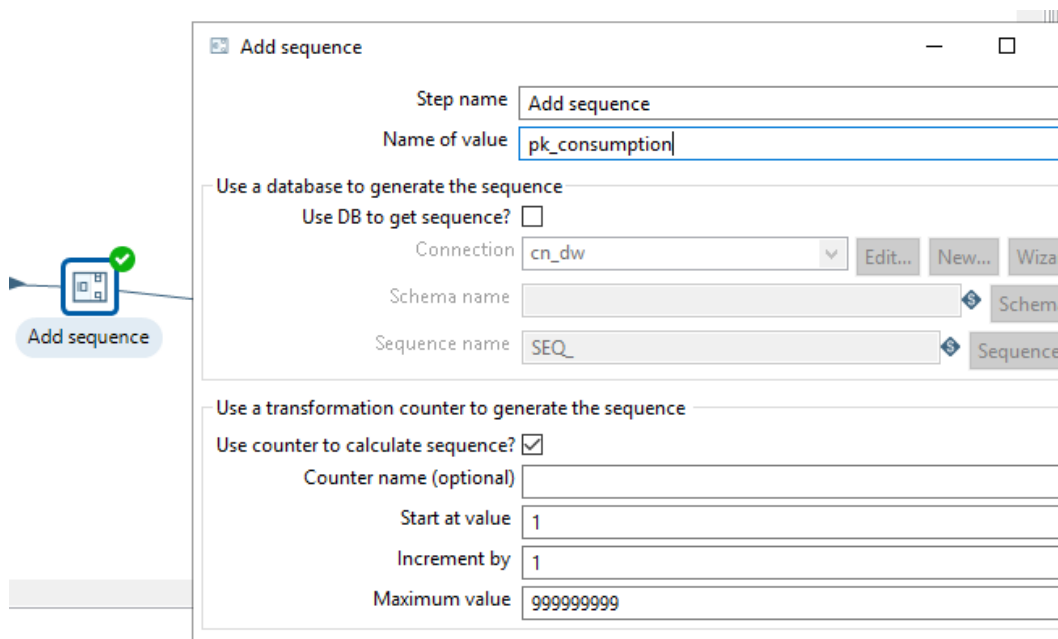
A continuación, se describe el desarrollo de la transformación de «TR_DIM_CONSUMPTION» mediante Spoon. En esta transformación se cargarán los datos disponibles en la tabla «STG_CONSUMPTION» en la tabla de dimensión «DIM_CONSUMPTION». Para hacerlo, seguiremos los siguientes tres pasos:

- leer los datos de «STG_CONSUMPTION»,
- añadir secuencia,
- insertar en la tabla «DIM_CONSUMPTION».

Para realizar la lectura se ha utilizado el componente «Table input», que permite realizar una *query* directamente en la tabla de la BBDD. En este caso, se quieren todos los campos.



Con los datos cargados en Spoon, se utiliza el componente «Add sequence» para añadir una secuencia que actuará como «PK» de la tabla.



Add sequence

Step name: Add sequence

Name of value: pk_consumption

Use a database to generate the sequence

Use DB to get sequence? ☐

Connection: cn_dw

Schema name:

Sequence name: SEQ_

Use a transformation counter to generate the sequence

Use counter to calculate sequence? ☒

Counter name (optional):

Start at value: 1

Increment by: 1

Maximum value: 99999999

Seguidamente, se realiza la carga de los datos en la tabla de dimensión correspondiente. Deberéis mapear los campos de la tabla de *staging* con la dimensión final, ya que puede que haya algún cambio de nombre. En este caso, la columna «description» de «STG_CONSUMPTION» se llamará «consumption_name» en «DIM_CONSUMPTION».

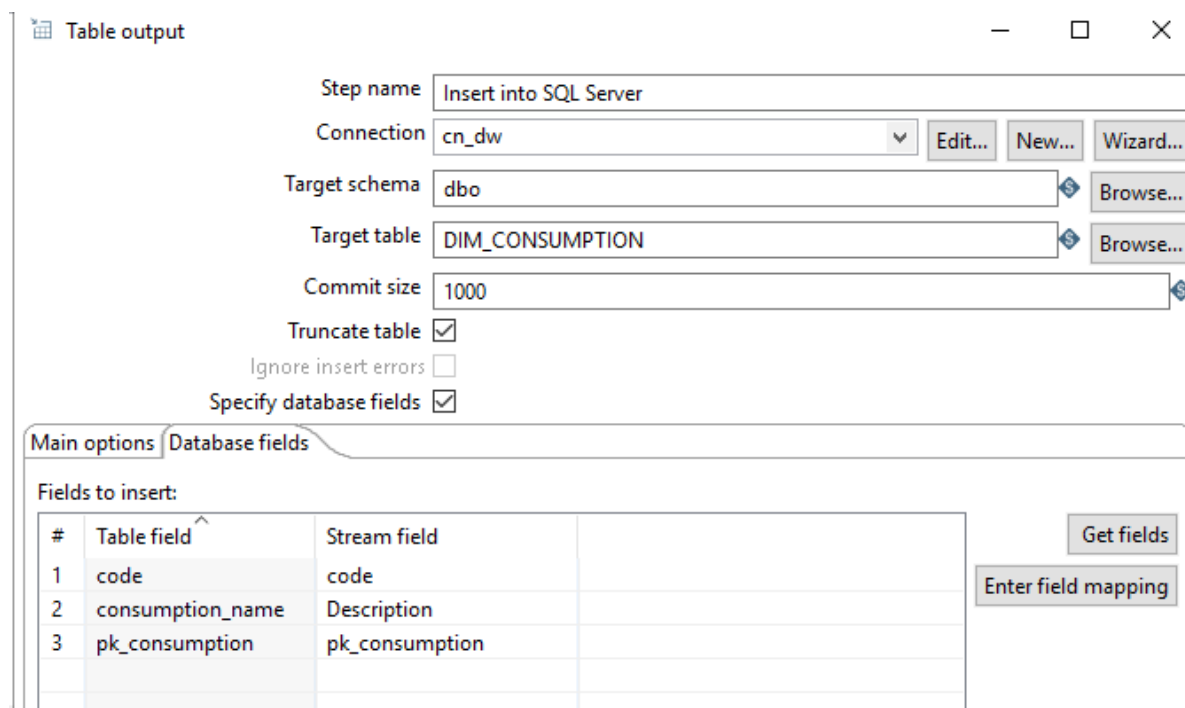


Table output

Step name: Insert into SQL Server

Connection: cn_dw

Target schema: dbo

Target table: DIM_CONSUMPTION

Commit size: 1000

Truncate table: ☒

Ignore insert errors: ☐

Specify database fields: ☒

Main options | Database fields

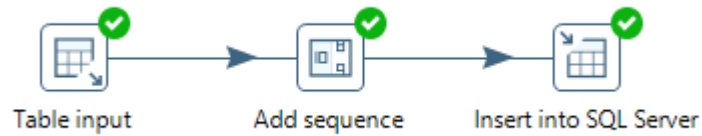
Fields to insert:

#	Table field	Stream field
1	code	code
2	consumption_name	Description
3	pk_consumption	pk_consumption

Get fields

Enter field mapping

El proceso final de transformación quedaría de la siguiente manera:



Execution Results													
Logging Execution History Step Metrics Performance Graph Metrics Preview data													
#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected	Errors	Active	Time	Speed (r/s)	in
1	Table input	0	0	8	8	0	0	0	0	Finished	0.0s	348	
2	Add sequence	0	8	8	0	0	0	0	0	Finished	0.0s	308	
3	Insert into SQL Server	0	8	8	0	8	0	0	0	Finished	0.1s	138	

Finalmente, comprobamos el resultado en la tabla de la base de datos:

```

SELECT TOP (1000) [pk_consumption]
, [code]
, [consumption_name]
FROM [SOURCE_loginuoc].[dbo].[DIM_CONSUMPTION]
  
```

	pk_consumption	code	consumption_name
1	1	4161901	Band DA : Consumption < 1 000 kWh
2	2	4161902	Band DB : 1 000 kWh < Consumption < 2 500 kWh
3	3	4161903	Band DC : 2 500 kWh < Consumption < 5 000 kWh
4	4	4161904	Band DD : 5 000 kWh < Consumption < 15 000 kWh
5	5	4161905	Band DE : Consumption < 15 000 kWh

3.3.6. Bloque TR_FACT

Este bloque contiene las transformaciones para la carga inicial de las tablas de hecho al almacén desde las tablas intermedias «STG_» del *staging area*.

Con la implementación y ejecución de los procesos de carga de dimensiones tendrás una gran cantidad de datos en vuestro modelo dimensional y podréis pasar entonces a añadir los datos al modelo de hechos, haciendo referencia a las dimensiones disponibles mediante sus claves foráneas.

Transformación TR_FACT_EUROZONE_INDICATORS

La parte principal en la carga de las tablas de hechos es la búsqueda de los valores de las claves foráneas en las tablas de dimensiones cargadas anteriormente.

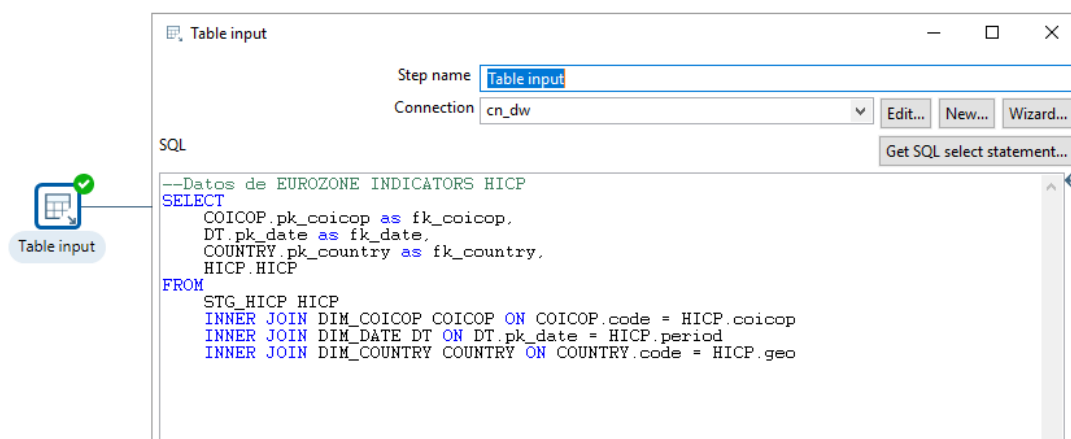
De la misma manera que en la carga de las dimensiones, siempre que sea necesario unir dos o más flujos de datos o realizar una unión, los datos deben estar ordenados por la misma clave.

Existen varias soluciones para tratar los valores nulos en claves foráneas: eliminar los registros «incompletos» (asumiendo una pérdida de datos), asignar valores constantes y buscar registros *NI*, *NA* dinámicamente, etc. En este caso, dado que nuestro objetivo es analizar el indicador HICP para valorar cada uno de los países, hemos decidido eliminar registros que no tengamos el campo informado.

En esta transformación se han realizado diferentes pasos para llegar a cargar la tabla de hechos. Concretamente, los pasos seguidos son los siguientes:

- carga de datos («STG_HICP») incorporando claves foráneas,
- filtrado de nulos,
- nueva secuencia,
- carga en la tabla «FACT_EUROZONE_INDICATORS».

El primer paso ha sido traernos los datos de *staging* de la tabla «STG_HICP» con el módulo de «Table input». Como podéis apreciar en la imagen, solo seleccionaremos las columnas necesarias para nuestro posterior análisis (el cual quedó definido en la PRA1).

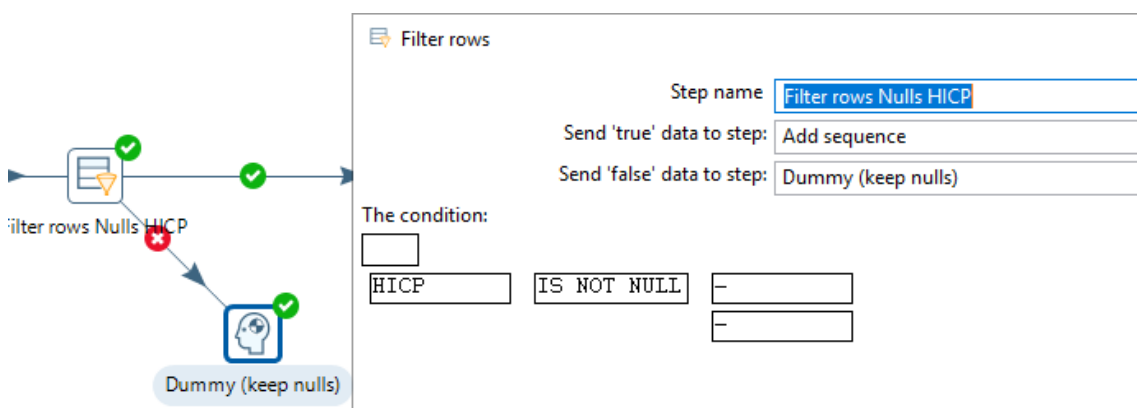


El módulo anterior ha ejecutado la siguiente consulta:

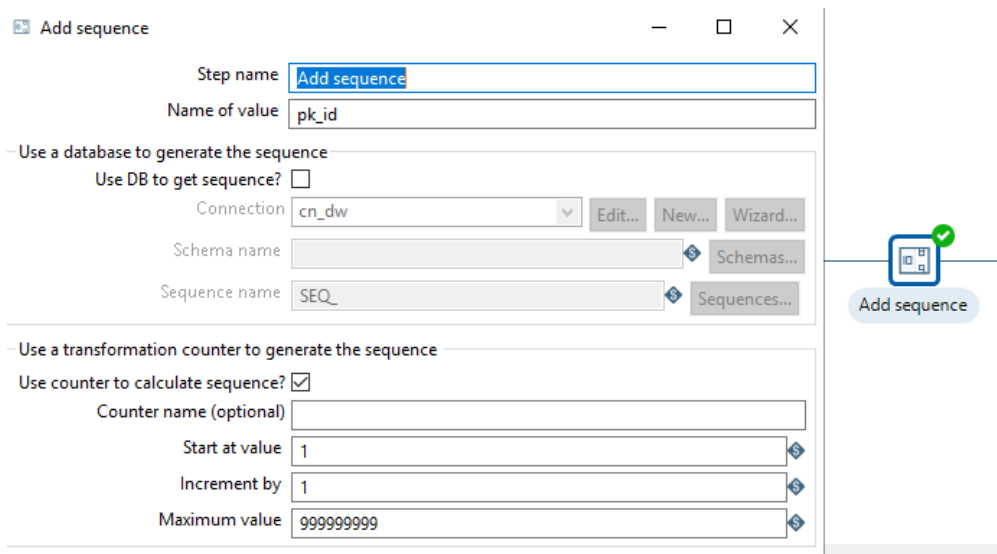
```
--Datos de EUROZONE INDICATORS HICP
SELECT
  COICOP.pk_coicop as fk_coicop,
  DT.pk_date as fk_date,
  COUNTRY.pk_country as fk_country,
  HICP.HICP
FROM
  STG_HICP HICP
INNER JOIN DIM_COICOP COICOP ON COICOP.code = HICP.coicop
INNER JOIN DIM_DATE DT ON DT.pk_date = HICP.period
INNER JOIN DIM_COUNTRY COUNTRY ON COUNTRY.code = HICP.geo
```

Una vez cargada la información, y dado que el objetivo de esta tabla será el análisis del indicador «HICP», procederemos a eliminar los registros nulos en este campo. Además, también se perderán los registros sin país informado, dado que no tiene mucho sentido agrupar los registros sin este dato debido a que el indicador de «HICP» puede variar enormemente entre regiones.

Opcionalmente, podemos mandar los registros filtrados a un *flow* tipo «Dummy» si queremos revisar qué registros no se insertarán posteriormente en nuestra base de datos.



Con los conjuntos de datos ya juntos en un mismo *set*, se ha procedido a añadir la secuencia que actuará como clave primaria para después realizar el *insert* a BBDD con el mapeo de datos correspondiente. El nombre de este nuevo campo será «pk_id»:



Add sequence

Step name:

Name of value:

Use a database to generate the sequence

Use DB to get sequence? ☐

Connection: Edit... New... Wizard...

Schema name: Schemas...

Sequence name: Sequences...

Use a transformation counter to generate the sequence

Use counter to calculate sequence? ☒

Counter name (optional):

Start at value:

Increment by:

Maximum value:

Add sequence

Seguidamente, se realiza la carga de los datos en la tabla de hechos correspondiente en el SQL Server. Podéis mapear las columnas con la tabla «FACT_EUROZONE_INDICATORS» para asegurarnos de que se insertarán correctamente:

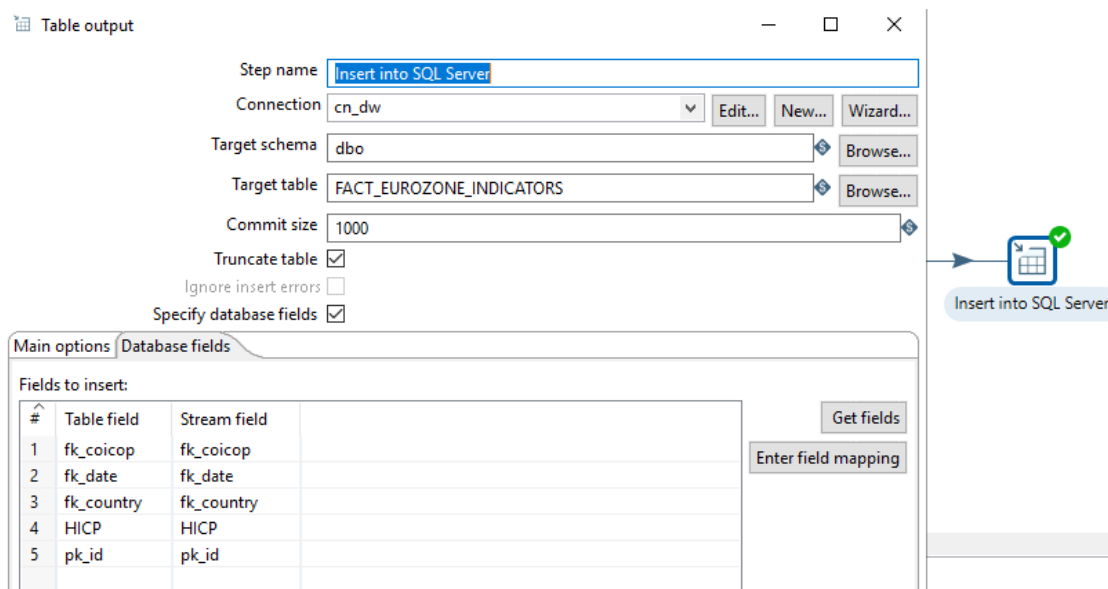


Table output

Step name:

Connection: Edit... New... Wizard...

Target schema: Browse...

Target table: Browse...

Commit size:

Truncate table: ☒

Ignore insert errors: ☐

Specify database fields: ☒

Main options Database fields

Fields to insert:

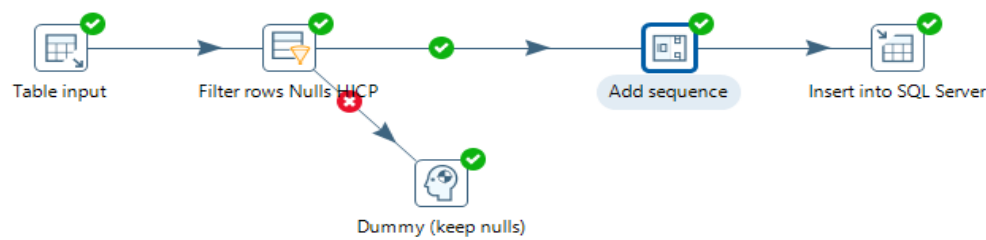
#	Table field	Stream field
1	fk_coicop	fk_coicop
2	fk_date	fk_date
3	fk_country	fk_country
4	HICP	HICP
5	pk_id	pk_id

Get fields

Enter field mapping

Insert into SQL Server

El proceso final de transformación quedaría de la siguiente manera:



Execution Results

#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected	Errors	Active	Time	Speed (r/s)	input/output
1	Table input	0	0	510300	510300	0	0	0	0	Finished	15.0s	33,961	-
2	Filter rows Nulls HICP	0	510300	510300	0	0	0	0	0	Finished	15.3s	33,423	-
3	Add sequence	0	491014	491014	0	0	0	0	0	Finished	15.5s	31,656	-
4	Insert into SQL Server	0	491014	491014	0	491014	0	0	0	Finished	15.8s	31,170	-
5	Dummy (keep nulls)	0	19286	19286	0	0	0	0	0	Finished	15.3s	1,262	-

Comprobamos el resultado en la tabla de la base de datos:

```

SELECT TOP (1000) [pk_id]
      ,[fk_date]
      ,[fk_country]
      ,[fk_coicop]
      ,[HICP]
FROM [SOURCE_loginuoc].[dbo].[FACT_EUROZONE_INDICATORS]
  
```

Results

pk_id	fk_date	fk_country	fk_coicop	HICP
1	202102	210	315	-0.60
2	202204	81	315	1.90
3	202006	203	315	-0.90
4	202108	55	315	1.40
5	202210	108	315	18.00

Transformación TR_FACT_ENERGY_PRICE

De la misma manera que hicimos para los indicadores de la eurozona, la parte principal en la carga de las tablas de hechos es la búsqueda de los valores de las claves foráneas en las tablas de dimensiones cargadas anteriormente.

Existen varias soluciones para tratar los valores nulos en claves foráneas: eliminar los registros «incompletos» (asumiendo una pérdida de datos), asignar valores constantes y buscar registros *NI*, *NA* dinámicamente, etc. En este caso, dado que nuestro objetivo es analizar la evolución de precios del gas y la electricidad, hemos decidido eliminar registros donde no tengamos los campos de «PGAS» y «PELEC» informados.

En esta transformación se han realizado diferentes pasos para llegar a cargar la tabla de hechos. Concretamente, los pasos seguidos son los siguientes:

- carga de datos («STG_PGAS» y «STG_Pelec») incorporando claves foráneas,
- unión de las dos tablas en una misma antes de tratar los datos,
- filtrado de nulos,
- orden por periodo y generación de clave primaria,
- carga en la tabla «FACT_ENERGY_PRICE».

El primer paso ha sido traernos los datos de *staging* de las tablas «STG_PGAS» y «STG_Pelec» con el módulo de «Table input». Como podéis apreciar en la imagen, solo seleccionaremos las columnas necesarias para nuestro posterior análisis (el cual quedó definido en la PRA1). En este caso solo tuvimos que excluir la columna «freq».

Además, aprovecharemos para cambiar de nombre a algunas columnas, en especial las columnas de precios (*pgas* y *pelec*) las cuales renombraremos como «price» de cara a facilitar la unión entre tablas que realizaremos posteriormente:

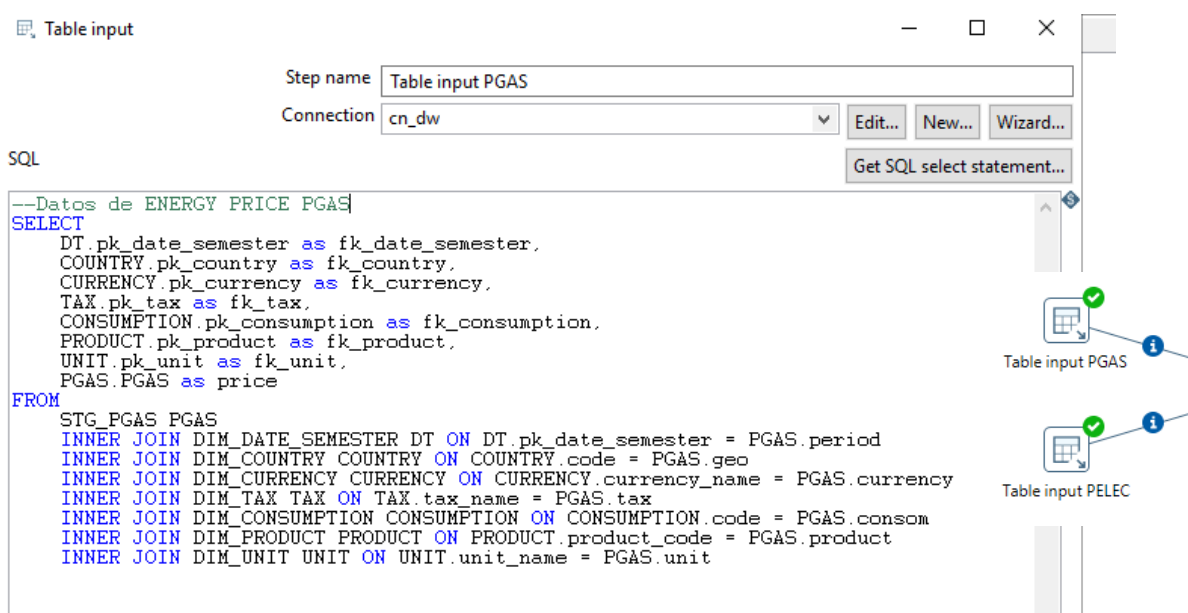


Table input

Step name: Table input PGAS

Connection: cn_dw

SQL

```
--Datos de ENERGY PRICE PGAS
SELECT
  DT.pk_date_semester as fk_date_semester,
  COUNTRY.pk_country as fk_country,
  CURRENCY.pk_currency as fk_currency,
  TAX.pk_tax as fk_tax,
  CONSUMPTION.pk_consumption as fk_consumption,
  PRODUCT.pk_product as fk_product,
  UNIT.pk_unit as fk_unit,
  PGAS.PGAS as price
FROM
  STG_PGAS PGAS
  INNER JOIN DIM_DATE_SEMESTER DT ON DT.pk_date_semester = PGAS.period
  INNER JOIN DIM_COUNTRY COUNTRY ON COUNTRY.code = PGAS.geo
  INNER JOIN DIM_CURRENCY CURRENCY ON CURRENCY.currency_name = PGAS.currency
  INNER JOIN DIM_TAX TAX ON TAX.tax_name = PGAS.tax
  INNER JOIN DIM_CONSUMPTION CONSUMPTION ON CONSUMPTION.code = PGAS.consom
  INNER JOIN DIM_PRODUCT PRODUCT ON PRODUCT.product_code = PGAS.product
  INNER JOIN DIM_UNIT UNIT ON UNIT.unit_name = PGAS.unit
```

Table input PGAS

Table input PELEC

Table input

Step name Table input PELEC

Connection cn_dw

Get SQL select statement...

SQL

```
--Datos de ENERGY PRICE PELEC
SELECT
  DT.pk_date_semester as fk_date_semester,
  COUNTRY.pk_country as fk_country,
  CURRENCY.pk_currency as fk_currency,
  TAX.pk_tax as fk_tax,
  CONSUMPTION.pk_consumption as fk_consumption,
  PRODUCT.pk_product as fk_product,
  UNIT.pk_unit as fk_unit,
  PELEC.PELEC as price
FROM
  STG_PELEC PELEC
  INNER JOIN DIM_DATE_SEMESTER DT ON DT.pk_date_semester = PELEC.period
  INNER JOIN DIM_COUNTRY COUNTRY ON COUNTRY.code = PELEC.geo
  INNER JOIN DIM_CURRENCY CURRENCY ON CURRENCY.currency_name = PELEC.currency
  INNER JOIN DIM_TAX TAX ON TAX.tax_name = PELEC.tax
  INNER JOIN DIM_CONSUMPTION CONSUMPTION ON CONSUMPTION.code = PELEC.consom
  INNER JOIN DIM_PRODUCT PRODUCT ON PRODUCT.product_code = PELEC.product
  INNER JOIN DIM_UNIT UNIT ON UNIT.unit_name = PELEC.unit
```

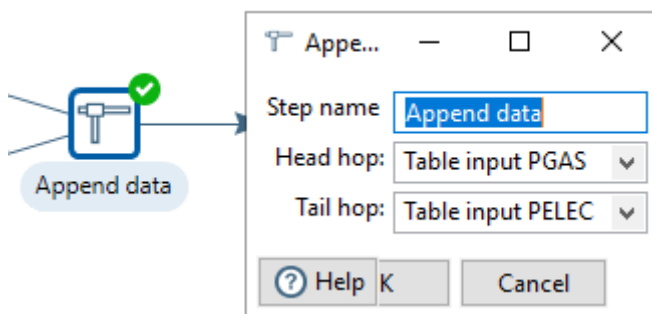
Cada uno de los módulos ha ejecutado una de estas consultas respectivamente:

```
--Datos de ENERGY PRICE PGAS
SELECT
  DT.pk_date_semester as fk_date_semester,
  COUNTRY.pk_country as fk_country,
  CURRENCY.pk_currency as fk_currency,
  TAX.pk_tax as fk_tax,
  CONSUMPTION.pk_consumption as fk_consumption,
  PRODUCT.pk_product as fk_product,
  UNIT.pk_unit as fk_unit,
  PGAS.PGAS as price
FROM
  STG_PGAS PGAS
  INNER JOIN DIM_DATE_SEMESTER DT ON DT.pk_date_semester = PGAS.period
  INNER JOIN DIM_COUNTRY COUNTRY ON COUNTRY.code = PGAS.geo
  INNER JOIN DIM_CURRENCY CURRENCY ON CURRENCY.currency_name =
PGAS.currency
  INNER JOIN DIM_TAX TAX ON TAX.tax_name = PGAS.tax
  INNER JOIN DIM_CONSUMPTION CONSUMPTION ON CONSUMPTION.code = PGAS.consom
  INNER JOIN DIM_PRODUCT PRODUCT ON PRODUCT.product_code = PGAS.product
  INNER JOIN DIM_UNIT UNIT ON UNIT.unit_name = PGAS.unit
```

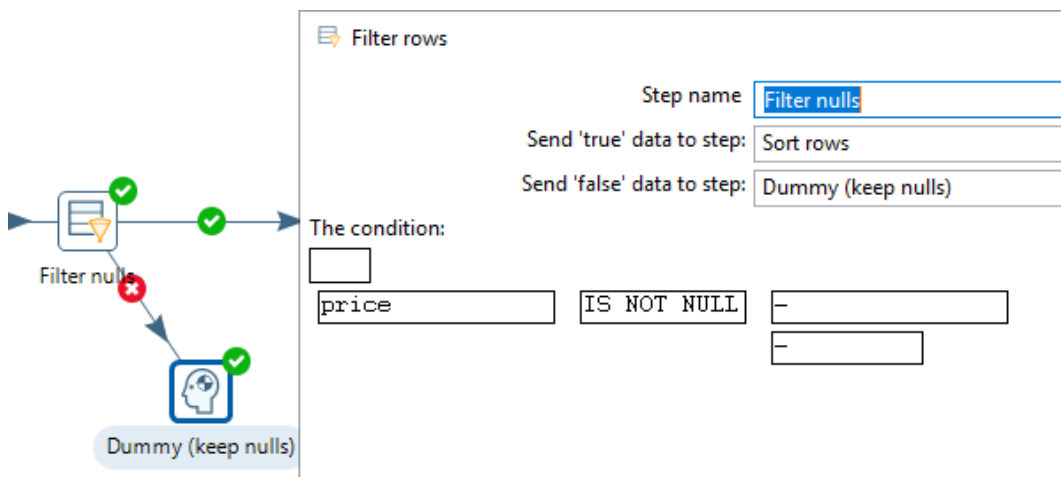
```
--Datos de ENERGY PRICE PELEC
SELECT
  DT.pk_date_semester as fk_date_semester,
  COUNTRY.pk_country as fk_country,
  CURRENCY.pk_currency as fk_currency,
  TAX.pk_tax as fk_tax,
  CONSUMPTION.pk_consumption as fk_consumption,
  PRODUCT.pk_product as fk_product,
  UNIT.pk_unit as fk_unit,
  PELEC.PELEC as price
FROM
  STG_PELEC PELEC
  INNER JOIN DIM_DATE_SEMESTER DT ON DT.pk_date_semester = PELEC.period
  INNER JOIN DIM_COUNTRY COUNTRY ON COUNTRY.code = PELEC.geo
  INNER JOIN DIM_CURRENCY CURRENCY ON CURRENCY.currency_name =
PELEC.currency
```

```
INNER JOIN DIM_TAX TAX ON TAX.tax_name = PELEC.tax
INNER JOIN DIM_CONSUMPTION CONSUMPTION ON CONSUMPTION.code =
PELEC.consom
INNER JOIN DIM_PRODUCT PRODUCT ON PRODUCT.product_code = PELEC.product
INNER JOIN DIM_UNIT UNIT ON UNIT.unit_name = PELEC.unit
```

Antes del tratamiento de datos, procederemos a juntar nuestros dos *datasets* en uno solo, ya que tienen las mismas columnas e información, a excepción del precio (gas y electricidad). Para ello usaremos la transformación de «Append data»:



Una vez cargada la información, y dado que el objetivo de esta tabla será el análisis del indicador «price», procederemos a eliminar los registros nulos en este campo. Opcionalmente, podemos mandar los registros filtrados a un *flow* tipo «Dummy» si queremos revisar que registros no se insertarán posteriormente en nuestra base de datos.

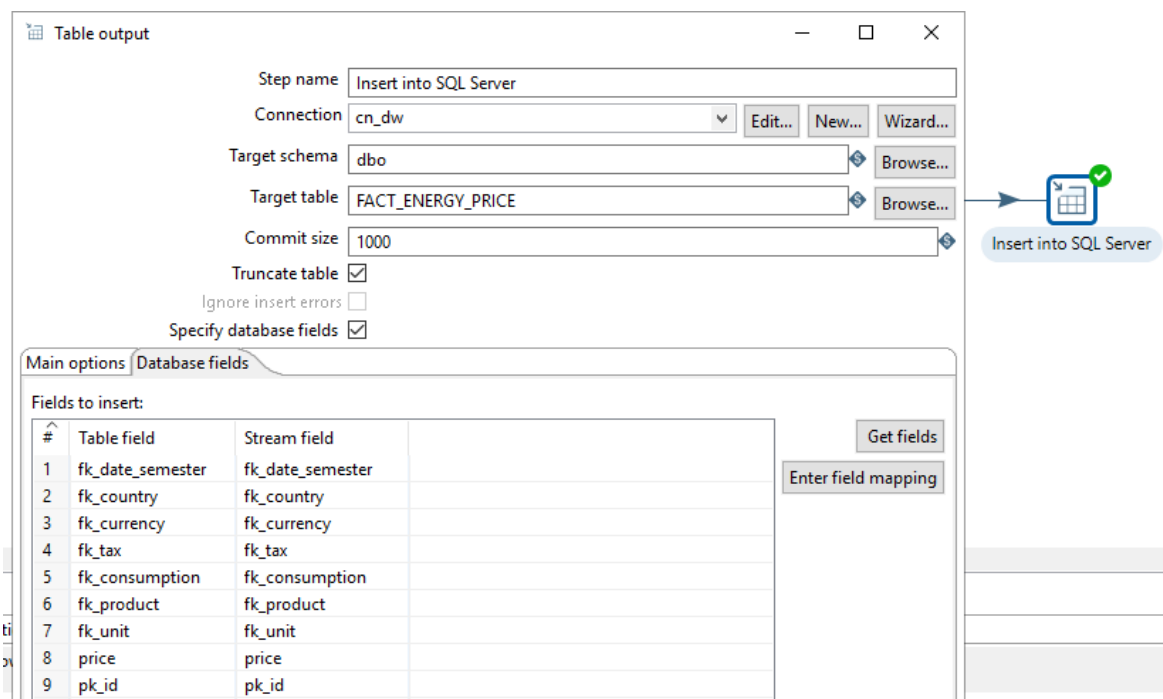


Seguidamente ya podremos ordenar nuestros datos por el campo de periodo antes de añadir la clave primaria. Esto lo haremos mediante el paso de «Sort rows»:

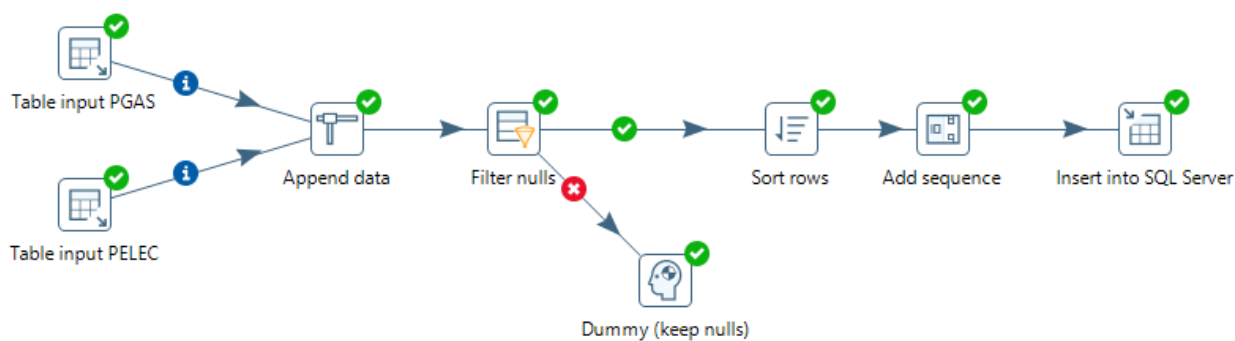
#	Fieldname	Ascending	Case sensitive compare?	Sort based on current locale?	Collator Strength	Presorted?
1	fk_date_semester	Y	N	N	0	N

Después de ordenar nuestros datos por periodo, se ha procedido a añadir la secuencia que actuará como clave primaria para después realizar el *insert* a BBDD con el mapeo de datos correspondiente. El nombre de este nuevo campo será «pk_id»:

Seguidamente, se realiza la carga de los datos en la tabla de hechos correspondiente en el SQL Server. Podéis mapear las columnas con la tabla «FACT_ENERGY_PRICE» para asegurarnos de que se insertan correctamente:



El proceso final de transformación quedaría de la siguiente manera:



Execution Results

Logging Execution History Step Metrics Performance Graph Metrics Preview data

#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected	Errors	Active	Time	Speed (r/s)	input/output
1	Table input PGAS	0	0	55800	55800	0	0	0	0	Finished	0.6s	99,821	-
2	Table input PELEC	0	0	57195	57195	0	0	0	0	Finished	0.9s	61,237	-
3	Append data	0	112995	112995	0	0	0	0	0	Finished	1.1s	103,951	-
4	Filter nulls	0	112995	112995	0	0	0	0	0	Finished	1.1s	103,475	-
5	Sort rows	0	97870	97870	0	0	0	0	0	Finished	3.1s	31,308	-
6	Dummy (keep nulls)	0	15125	15125	0	0	0	0	0	Finished	1.1s	13,456	-
7	Add sequence	0	97870	97870	0	0	0	0	0	Finished	3.4s	29,197	-
8	Insert into SQL Server	0	97870	97870	0	97870	0	0	0	Finished	3.6s	27,399	-

Comprobamos el resultado en la tabla de la base de datos:

```

SELECT TOP (1000) [pk_id]
, [fk_date_semester]
, [fk_country]
, [fk_currency]
, [fk_tax]
, [fk_consumption]
, [fk_unit]
, [fk_product]
, [price]
FROM [SOURCE_loginuoc].[dbo].[FACT ENERGY PRICE]

```

100 %

Results Messages

	pk_id	fk_date_semester	fk_country	fk_currency	fk_tax	fk_consumption	fk_unit	fk_product	price
1	1	2017S1	99	3	1	6	1	1	15.8245
2	2	2017S1	127	3	1	6	1	1	28.8659
3	3	2017S1	130	3	1	6	1	1	29.8446
4	4	2017S1	176	3	1	6	1	1	31.3714
5	5	2017S1	209	3	1	6	1	1	34.5225
6	6	2017S1	222	3	1	6	1	1	14.5082

4. Implementación de trabajos con procesos ETL

Habrá que tener en cuenta los siguientes bloques de procesos implementados:

- Bloque «IN_»: procesos de ETL de transformación y carga al área intermedia.
- Bloque «TR_DIM»: procesos de ETL de transformación y carga de dimensiones.
- Bloque «TR_FACT»: procesos de ETL de transformación y carga de hechos.

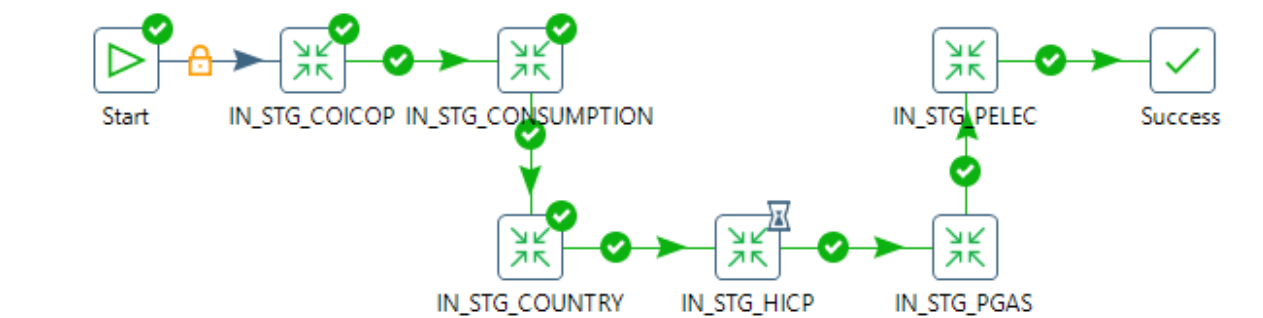
Diseñaréis los trabajos (*jobs*) mediante PDI; estos van a permitir la ejecución secuencial de todos los procesos de ETL incluidos en cada bloque definido.

Cada trabajo contiene como pasos cada una de las transformaciones implementadas en el apartado anterior de diseño de ETL.

JOB_IN

El trabajo (*job*) «JOB_IN» procesa todas las transformaciones del bloque «IN_» para la carga de datos desde las fuentes de datos proporcionadas al área intermedia (*staging area*).

El diseño completo del trabajo (*job*) «JOB_IN» es el siguiente:



Los pasos incluidos en el trabajo «JOB_IN» son:

- inicio del *job*,
- ejecución de las transformaciones «IN_» de carga del *staging area*,
- finalización del *job*.

El resultado de la ejecución de la transformación completa es el siguiente:

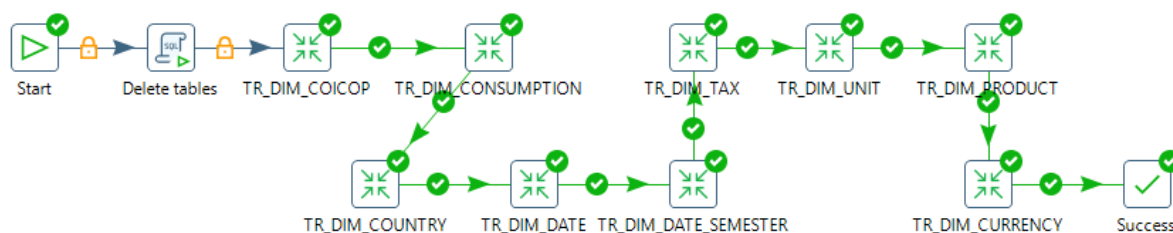
Job / Job Entry	Comment	Result	Reason
▼ JOB_IN			
Job: JOB_IN	Start of job execution		start
Start	Start of job execution		start
Start	Job execution finished	Success	
IN_STG_COICOP	Start of job execution		Followed unconditional link
IN_STG_COICOP	Job execution finished	Success	
IN_STG_CONSUMPTION	Start of job execution		Followed link after success
IN_STG_CONSUMPTION	Job execution finished	Success	
IN_STG_COUNTRY	Start of job execution		Followed link after success
IN_STG_COUNTRY	Job execution finished	Success	
IN_STG_HICP	Start of job execution		Followed link after success
IN_STG_HICP	Job execution finished	Success	
IN_STG_PGAS	Start of job execution		Followed link after success
IN_STG_PGAS	Job execution finished	Success	
IN_STG_PELEC	Start of job execution		Followed link after success
IN_STG_PELEC	Job execution finished	Success	
Success	Start of job execution		Followed link after success

Se observa el procesamiento con éxito de todos los pasos del «JOB_IN» correspondientes a la ejecución de todas las transformaciones que están incluidas en el trabajo.

JOB_TR_DIMS

El trabajo (*job*) «JOB_TR_DIMS» procesa todas las transformaciones del bloque «TR_DIMS» para la carga de datos, desde las tablas intermedias hasta las tablas de dimensiones del almacén.

El diseño completo del trabajo (*job*) «JOB_TR_DIMS» es el siguiente:



Los pasos incluidos en el trabajo «JOB_TR_DIMS» son:

- inicio del *job*,
- borrado de todas las tablas (esto permite la recarga inicial en caso de ser necesario. Aunque es importante respetar el orden de borrado, según las relaciones definidas entre tablas, en este caso en particular, dado que no hay relaciones entre tablas de dimensión, no influirá el orden),
- ejecución secuencial de todas las transformaciones «TR_DIM» (extracción, transformación y carga de dimensiones),
- finalización del *job*.

El resultado de la ejecución de la transformación completa es el siguiente:

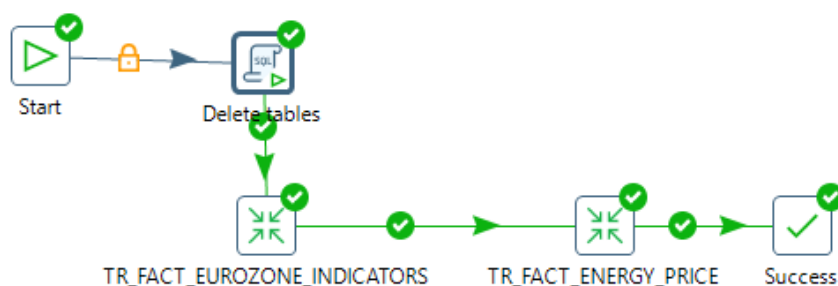
Job / Job Entry	Comment	Result	Reason
▼ JOB_TR_DIMS			
Job: JOB_TR_DIMS	Start of job execution		start
Start	Start of job execution		start
Start	Job execution finished	Success	
SQL	Start of job execution		Followed unconditional link
SQL	Job execution finished	Success	
TR_DIM_COICOP	Start of job execution		Followed unconditional link
TR_DIM_COICOP	Job execution finished	Success	
TR_DIM_CONSUMPTION	Start of job execution		Followed link after success
TR_DIM_CONSUMPTION	Job execution finished	Success	
TR_DIM_COUNTRY	Start of job execution		Followed link after success
TR_DIM_COUNTRY	Job execution finished	Success	
TR_DIM_DATE	Start of job execution		Followed link after success
TR_DIM_DATE	Job execution finished	Success	
TR_DIM_DATE_SEMESTER	Start of job execution		Followed link after success
TR_DIM_DATE_SEMESTER	Job execution finished	Success	
TR_DIM_TAX	Start of job execution		Followed link after success
TR_DIM_TAX	Job execution finished	Success	
TR_DIM_UNIT	Start of job execution		Followed link after success
TR_DIM_UNIT	Job execution finished	Success	
TR_DIM_PRODUCT	Start of job execution		Followed link after success
TR_DIM_PRODUCT	Job execution finished	Success	
TR_DIM_CURRENCY	Start of job execution		Followed link after success
TR_DIM_CURRENCY	Job execution finished	Success	
Success	Start of job execution		Followed link after success
Success	Job execution finished	Success	
Job: JOB_TR_DIMS	Job execution finished	Success	finished

Se observa el procesamiento con éxito de todos los pasos del «JOB_TR_DIMS», correspondientes a la ejecución de todas las transformaciones que están incluidas en el trabajo.

JOB_TR_FACTS

El trabajo (*job*) «JOB_TR_FACTS» procesa todas las transformaciones del bloque «TR_FACT» para la carga de datos desde las tablas intermedias a las tablas de hechos del almacén.

El diseño completo del trabajo (*job*) «JOB_TR_FACTS» es el siguiente:



Los pasos incluidos en el trabajo «JOB_TR_FACTS» son:

- inicio del *job*,
- eliminación de tablas,
- ejecución de las transformaciones «TR_FACT»,
- finalización del *job*.

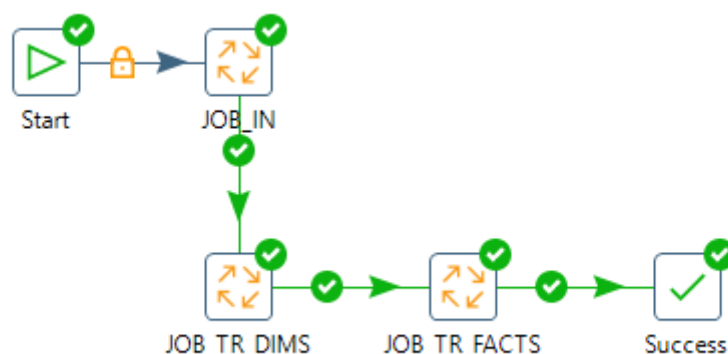
El resultado de la ejecución de la transformación completa es el siguiente:

Job / Job Entry	Comment	Result	Reason
▼ JOB_TR_FACTS			
Job: JOB_TR_FACTS	Start of job execution		start
Start	Start of job execution		start
Start	Job execution finished	Success	
Delete tables	Start of job execution		Followed unconditional link
Delete tables	Job execution finished	Success	
TR_FACT_EUROZONE_INDICAT	Start of job execution		Followed link after success
TR_FACT_EUROZONE_INDICAT	Job execution finished	Success	
TR_FACT_ENERGY_PRICE	Start of job execution		Followed link after success
TR_FACT_ENERGY_PRICE	Job execution finished	Success	
Success	Start of job execution		Followed link after success
Success	Job execution finished	Success	
Job: JOB_TR_FACTS	Job execution finished	Success	finished

JOB_CARGA_DW

El trabajo (*job*) «JOB_CARGA_DW» orquesta todos los trabajos anteriores en un único proceso.

El diseño completo del trabajo (*job*) «JOB_CARGA_DW» es el siguiente:



Los pasos incluidos en el trabajo «JOB_CARGA_DW» son:

- inicio del *job*,
- ejecución orquestada de los *jobs* de carga de todas las transformaciones («JOB_IN», «JOB_TR_DIM», «JOB_TR_FACT»),
- finalización del *job*.

El resultado de la ejecución de la transformación completa es el siguiente:

Job / Job Entry	Comment	Result	Reason
▼ JOB_CARGA_DW			
Job: JOB_CARGA_DW	Start of job execution		start
Start	Start of job execution		start
Start	Job execution finished	Success	
JOB_IN	Start of job execution		Followed unconditional link
> Job: JOB_IN			
JOB_IN	Job execution finished	Success	
JOB_TR_DIMS	Start of job execution		Followed link after success
> Job: JOB_TR_DIMS			
JOB_TR_DIMS	Job execution finished	Success	
JOB_TR_FACTS	Start of job execution		Followed link after success
> Job: JOB_TR_FACTS			
JOB_TR_FACTS	Job execution finished	Success	
Success	Start of job execution		Followed link after success
Success	Job execution finished	Success	
Job: JOB_CARGA_DW	Job execution finished	Success	finished

Se observa el procesamiento con éxito de todos los pasos del «JOB_CARGA_DW», correspondientes a la ejecución de todas las transformaciones que están incluidas en el trabajo.