

BASES DE DATOS ANALÍTICAS

Práctica 3

Diciembre 2025



Víctor Suesta Arribas

ÍNDICE

ÍNDICE

Ejercicio 1

- [1.1 Configuración de Spoon](#)
- [1.2 Preparación de la capa Staging en PostgreSQL](#)
- [1.3 Transformación IN_DGT_TYPE \(carga de dgt_type.csv → stg_dgt_type\)](#)
- [1.4 Transformación IN_MUNICIPALITIES \(carga de municipalities.csv → stg_municipalities\)](#)
- [1.5 Transformación IN_VEHICLES \(carga de vehicles.csv → stg_vehicles\)](#)
- [1.6 Conclusión](#)

Ejercicio 2

- [2.1 Perfilado de dbo.stg_municipalities](#)
- [2.2 Perfilado de dbo.stg_vehicles](#)
- [2.3 Conclusión](#)

Ejercicio 3

- [3.1 Transformación DIM_DGT_TYPE](#)
- [3.2 Transformación DIM_DATE](#)
- [3.3 Transformación DIM_VEHICLE](#)
- [3.4 Transformación FACT_VEHICLE_REGISTRATION](#)
- [3.5 Job JOB_LOAD](#)
- [3.6 Ejercicios a realizar](#)
- [3.7 Conclusión](#)

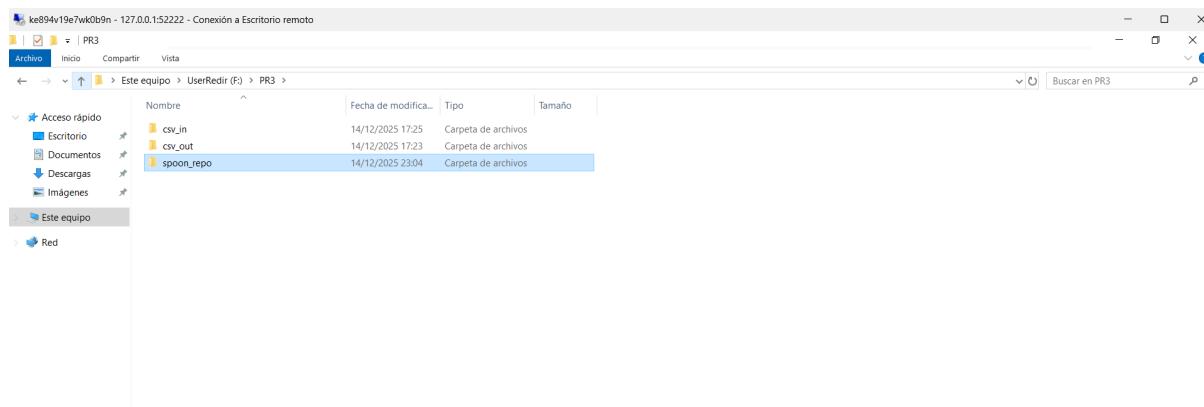
Ejercicio 1

En este ejercicio mi objetivo ha sido familiarizarme con **Pentaho Spoon** y, con esa herramienta, **diseñar y ejecutar procesos ETL** para cargar en la capa de *staging* de **PostgreSQL** los tres ficheros CSV que genera el sistema operacional: **dgt_type.csv**, **municipalities.csv** y **vehicles.csv**. Además, he dejado evidencias tanto de la ejecución en Spoon como de la validación posterior en base de datos.

1.1 Configuración de Spoon

1.1.1 Creación del repositorio

Lo primero que hice fue abrir Spoon en la VDI y crear un repositorio de tipo **File Repository** con el nombre **PR3**, seleccionando una ubicación persistente para guardar de forma estable todas las transformaciones.



Evidencia (Figura 1): creación del repositorio PR3 y selección de la ubicación.

1.1.2 Definición de la variable DIR_IN

Para evitar errores al repetir rutas en varios pasos, definí una variable de entorno en el fichero **kettle.properties**:

- **DIR_IN = F:\Fuentes**

A partir de ese momento, referencié la ruta con **\${DIR_IN}** dentro de los pasos de lectura de ficheros CSV.

| Enter the values for the kettle.properties file | | |
|---|---|----------------------|
| # | Variable name | Description |
| 1 | DIR_IN | Value F:/PR3/csv_in |
| 2 | DIR_OUT | Value F:/PR3/csv_out |
| 3 | KETTLE_AGGREGATION_ALL_NULLS_ARE_ZERO | N |
| 4 | KETTLE_AGGREGATION_MIN_NULL_IS_VALUE | N |
| 5 | KETTLE_ALLOW_EMPTY_TABLE_NAMES_AND_TYPES | false |
| 6 | KETTLE_BATCHING_BONKET | N |
| 7 | KETTLE_CARTE_JETTY_ACCEPTORS | |
| 8 | KETTLE_CARTE_JETTY_ACCEPT_QUEUE_SIZE | |
| 9 | KETTLE_CARTE_JETTY_RES_MAX_IDLE_TIME | |
| 10 | KETTLE_CARTE_OBJECT_TIMEOUT_MINUTES | 1440 |
| 11 | KETTLE_CHANGELOG_DDL | |
| 12 | KETTLE_CHANNEL_LOG_CRFMA | |
| 13 | KETTLE_CHANNEL_LOG_TABLE | |
| 14 | KETTLE_COMPATIBILITY_DB_JONOR_TIMEZONE | N |
| 15 | KETTLE_COMPATIBILITY_IMPORT_PATH_ADDITION_ON_VARIABLES | N |
| 16 | KETTLE_COMPATIBILITY_INVOKE_FILES_WITH_OR_WITHOUT_FILE_EXTENSION | Y |
| 17 | KETTLE_COMPATIBILITY_MERGE_ROWS_USE_REFERENCE_STREAM_WHEN_IDENTICAL | N |
| 18 | KETTLE_COMPATIBILITY_PUB_OLD_NAMING_MODE | N |
| 19 | KETTLE_COMPATIBILITY_TEXT_FILE_OUTPUT_APPEND_NO_HEADER | N |
| 20 | KETTLE_CORE_JOBENTRIESFILE | |
| 21 | KETTLE_CORE_STEPSFILE | |
| 22 | KETTLE_DATA_REFINERY_HTTP_CLIENT_TIMEOUT | 2000 |
| 23 | KETTLE_DEFAULT_BIGNUMBER_FORMAT | |
| 24 | KETTLE_DEFAULT_DATEFORMAT | |
| 25 | KETTLE_DEFAULT_INTEGER_FORMAT | |
| 26 | KETTLE_DEFAULT_NUMBER_FORMAT | |
| 27 | KETTLE_DEFAULT_SERVER_ENCODING | |
| 28 | KETTLE_DEFAULT_TIMESTAMP_FORMAT | |
| 29 | KETTLE_DISABLE_CONSOLE_LOGGING | N |
| 30 | KETTLE_DO_NOT_NORMALIZE_ALL_STRING_TO_EMPTY | N |
| 31 | KETTLE_ENABLE_THREADS_THREADSACES_ONLY_STRING_TO_EMPTY | N |
| 32 | KETTLE_EMPTY_STRING_DIFFERS_FROM_NULL | N |
| 33 | KETTLE_FAIL_ON_LOGGING_ERROR | N |
| 34 | KETTLE_FILE_OUTPUT_MAX_STREAM_COUNT | 1024 |
| 35 | KETTLE_FILE_OUTPUT_MAX_STREAM_SIZE | N |

Evidencia (Figura 2): fichero `kettle.properties` con la variable `DIR_IN` definida.

1.2 Preparación de la capa Staging en PostgreSQL

Antes de cargar datos, creé las tablas de *staging* en PostgreSQL, ya que los procesos ETL van a insertar los datos en estas tablas intermedias.

Las tablas creadas han sido:

- dbo.stg_dgt_type
 - dbo.stg_municipalities
 - dbo.stg_vehicles

The screenshot shows the pgAdmin 4 interface with a connection to a PostgreSQL database named 'SOURCE_suave/STUDENT_suave@PostgreSQL_PR1'. The left sidebar displays the database schema, including 'Schemas' (2), 'Tables' (5), and 'Views' (1). The 'Tables' section is currently selected, showing tables like 'aggregates', 'collections', 'domains', 'fts_configurations', 'fts_dictionaries', 'fts_fts_parsers', 'fts_templates', 'foreign_tables', 'functions', 'materialized_views', 'operators', 'procedures', 'sequences', and 'municipalities'. The 'stg_dgt_type' table is highlighted with a blue selection bar.

The main pane shows the following SQL code:

```
1 DROP TABLE IF EXISTS dbo.stg_dgt_type;
2
3 CREATE TABLE dbo.stg_dgt_type (
4     cod_type_dgt integer,
5     des_type_dgt varchar(22),
6     cod_subtype_dgt varchar(2),
7     des_subtype_dgt varchar(37)
8 );
9
```

Below the code, the 'Data Output' tab is active, showing the message: 'Query returned successfully in 75 msec.' The status bar at the bottom indicates 'Total rows: 1 of 1' and 'Query complete 00:00:00.075'. A green checkmark in the bottom right corner also states 'Query returned successfully in 75 msec.'

Evidencia (Figura 3): ejecución en pgAdmin del CREATE TABLE de dbo.stg_dgt_type.

The screenshot shows the pgAdmin 4 interface with a connection to 'ke894v19e7wk0b9n - 127.0.0.1:52222 - Conexión a Escritorio remoto'. The left sidebar shows the database schema with 'Schemas (2)' expanded, revealing 'dbo' which contains 'Tables (5)' including 'stg_municipalities'. The main window displays a query editor with the following SQL code:

```

1 DROP TABLE IF EXISTS dbo.stg_municipalities;
2
3 CREATE TABLE dbo.stg_municipalities (
4     cod_ine      bigint,
5     cod_geo      integer,
6     province    varchar(11),
7     name_muni   varchar(30),
8     population_muni integer,
9     surface      numeric(10,4),
10    perimeter   integer,
11    cod_in_capital bigint,
12    capital     varchar(42),
13    population_capital integer,
14    longitude_etr893 numeric(12,9),
15    latitude_etr893 numeric(11,8),
16    altitude     integer
17 );

```

The status bar at the bottom indicates 'Query returned successfully in 456 msec.' and 'Ln 19, Col 1'.

Evidencia (Figura 4): ejecución en pgAdmin del CREATE TABLE de dbo.stg_municipalities.

The screenshot shows the pgAdmin 4 interface with a connection to 'ke894v19e7wk0b9n - 127.0.0.1:52222 - Conexión a Escritorio remoto'. The left sidebar shows the database schema with 'Schemas (2)' expanded, revealing 'dbo' which contains 'Tables (5)' including 'stg_municipalities' and 'stg_vehicles'. The main window displays a query editor with the following SQL code:

```

1 DROP TABLE IF EXISTS dbo.stg_vehicles;
2
3 CREATE TABLE dbo.stg_vehicles (
4     id integer NOT NULL,
5     municipality varchar(5),
6     brand varchar(25),
7     model varchar(25),
8     registration_date date,
9     new_used boolean,
10    subtype varchar(2)
11 );

```

The status bar at the bottom indicates 'Query returned successfully in 289 msec.'

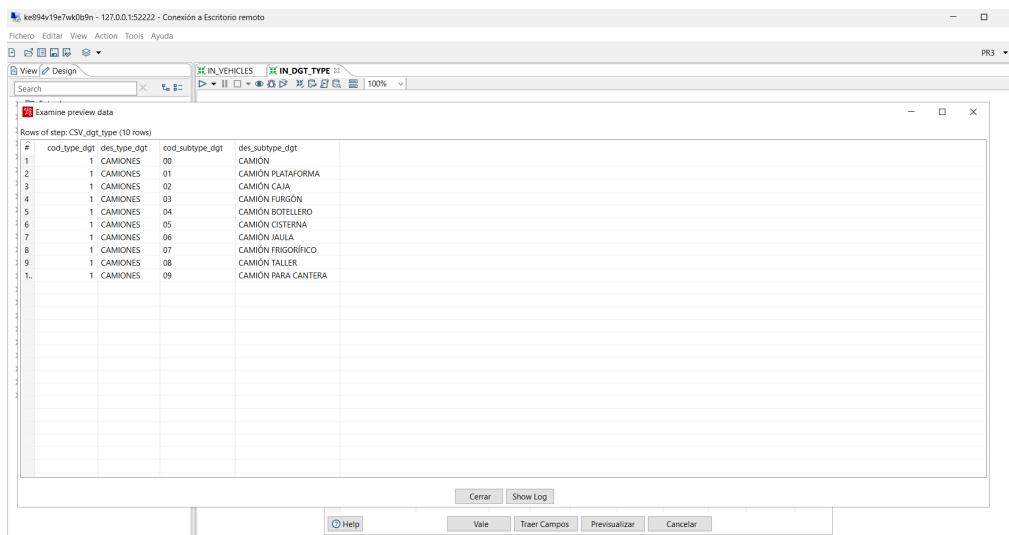
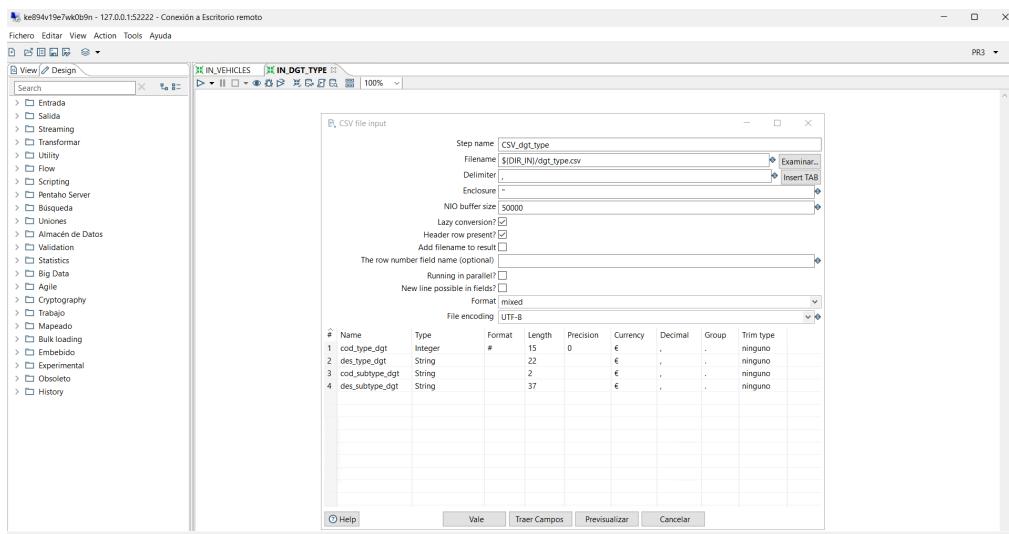
Evidencia (Figura 5): ejecución en pgAdmin del CREATE TABLE de dbo.stg_vehicles.

1.3 Transformación IN_DGT_TYPE (carga de dgt_type.csv → stg_dgt_type)

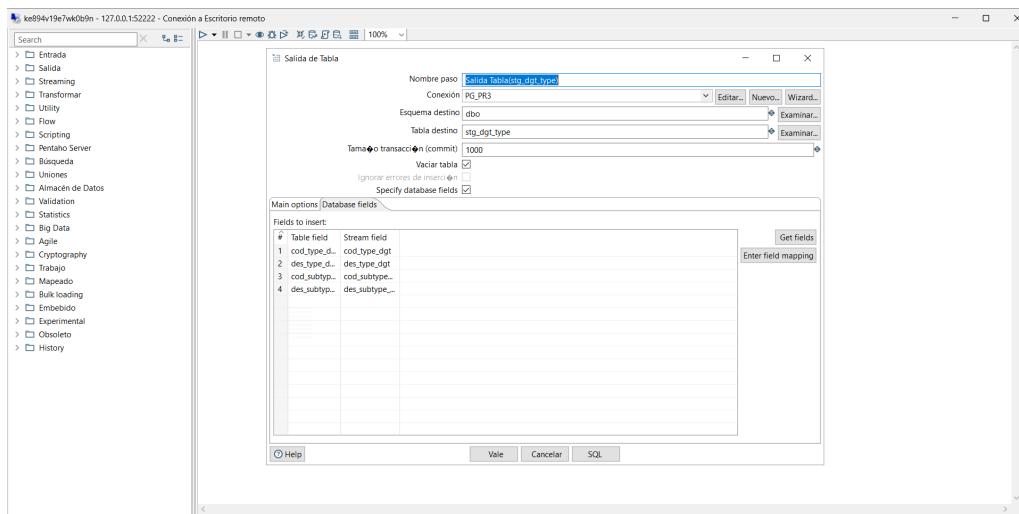
Para cargar el fichero `dgt_type.csv` construí una transformación con dos pasos principales:

1. **CSV file input:** lectura del fichero `DIR_IN\dgt_type.csv` indicando delimitador por coma, cabecera y codificación **UTF-8**. Con **Get Fields** obtuve la estructura y con **Preview** comprobé que los datos se estaban leyendo correctamente.

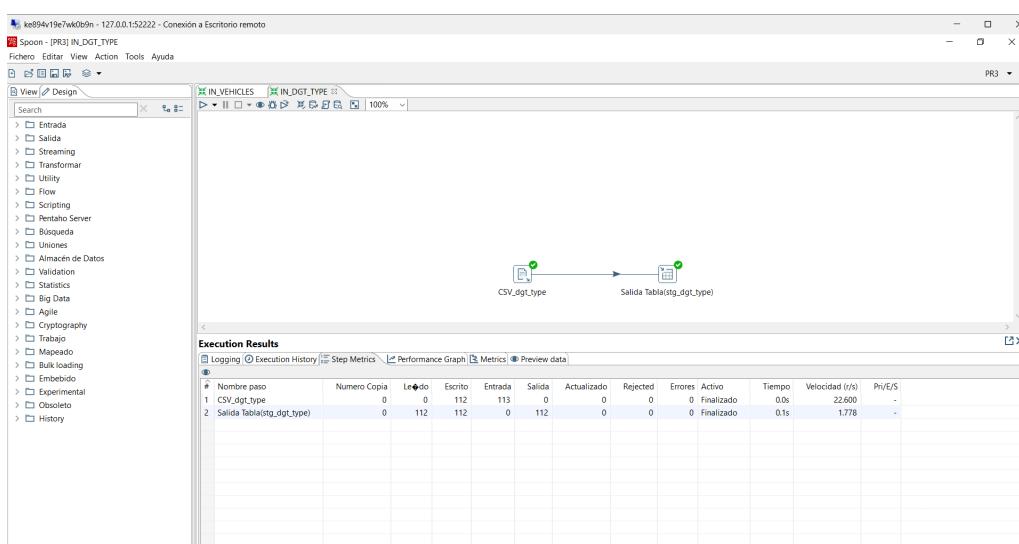
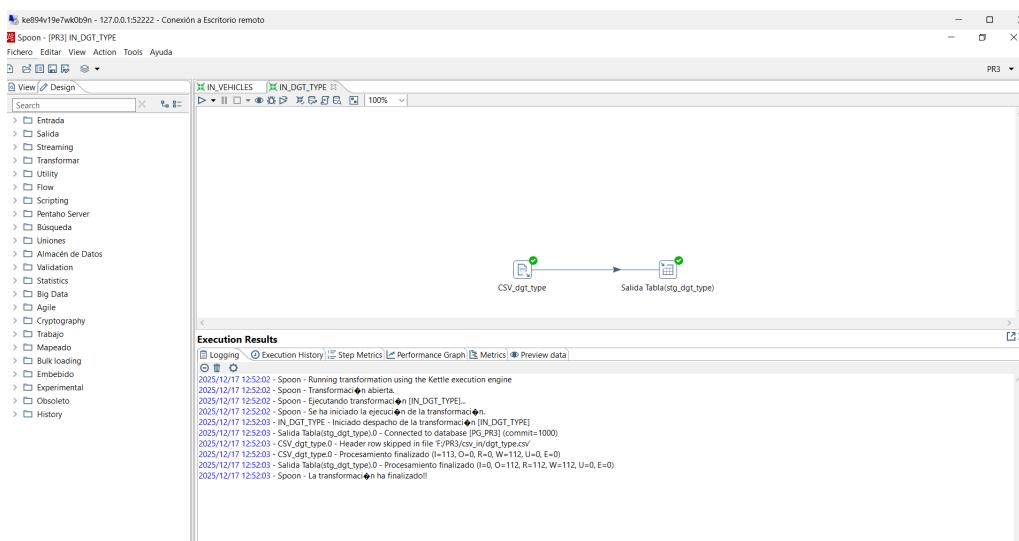
2. Table output: carga en la tabla `dbo.stg_dgt_type`, activando **Truncate table** para vaciar la tabla antes de insertar y realizando el mapeo de campos para que coincidieran con los nombres de la tabla.



Evidencia (Figuras 6.1 y 6.2): configuración y previsualización del paso CSV file input.



Evidencia (Figura 7): configuración del Table Output (tabla `stg_dgt_type`, truncate y mapeo).

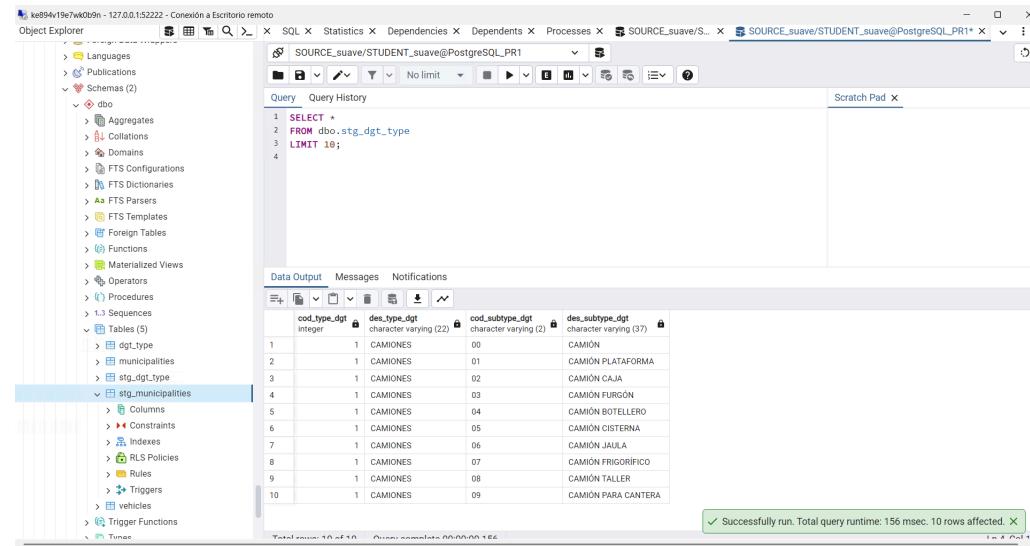


Evidencia (Figuras 8.1 y 8.2): métricas y log de ejecución sin errores.

Validación en PostgreSQL

Tras ejecutar la transformación, validé la carga desde pgAdmin con:

```
SELECT COUNT(*) AS n FROM dbo.stg_dgt_type;  
SELECT * FROM dbo.stg_dgt_type LIMIT 10;
```



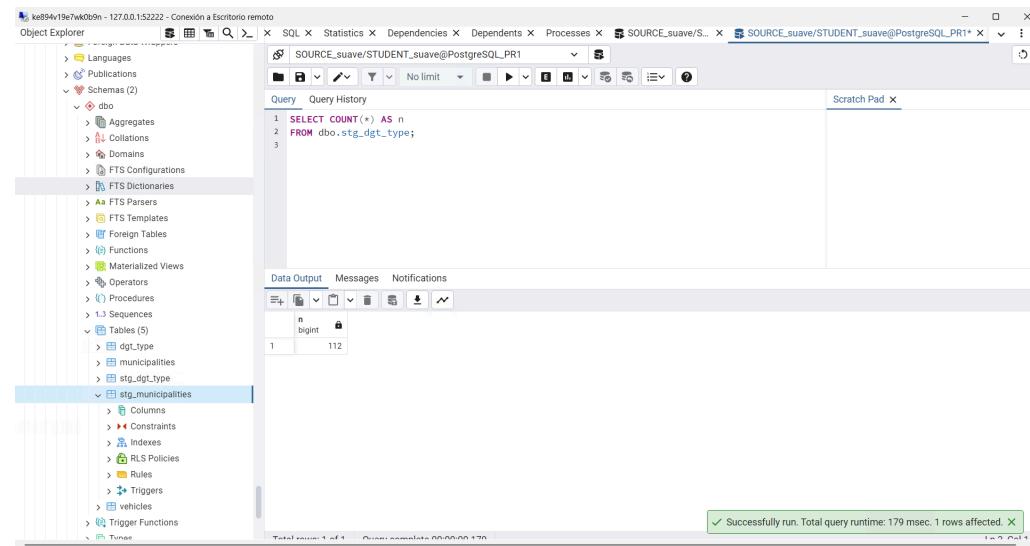
The screenshot shows the pgAdmin interface with a connection to 'ke89419e7wk0b9n - 127.0.0.1:52222'. The Object Explorer on the left shows the database structure, including the 'stg_municipalities' table under the 'dbo' schema. The main window displays a query in the 'Query' tab:

```
1 SELECT *  
2 FROM dbo.stg_dgt_type  
3 LIMIT 10;
```

The 'Data Output' tab shows the results of the query:

| cod_type_dgt | des_type_dgt | cod_subtype_dgt | des_subtype_dgt |
|--------------|--------------|-----------------|---------------------|
| 1 | CAMIONES | 00 | CAMION |
| 2 | CAMIONES | 01 | CAMION PLATAFORMA |
| 3 | CAMIONES | 02 | CAMION CAJA |
| 4 | CAMIONES | 03 | CAMION FURGON |
| 5 | CAMIONES | 04 | CAMION BOTELERO |
| 6 | CAMIONES | 05 | CAMION CISTERNA |
| 7 | CAMIONES | 06 | CAMION JAULA |
| 8 | CAMIONES | 07 | CAMION FRIGORIFICO |
| 9 | CAMIONES | 08 | CAMION TALLER |
| 10 | CAMIONES | 09 | CAMION PARA CANTERA |

A green success message at the bottom right indicates: 'Successfully run. Total query runtime: 156 msec. 10 rows affected.'



The screenshot shows the pgAdmin interface with the same connection. The Object Explorer on the left shows the database structure, including the 'stg_municipalities' table under the 'dbo' schema. The main window displays a query in the 'Query' tab:

```
1 SELECT COUNT(*) AS n  
2 FROM dbo.stg_dgt_type;  
3
```

The 'Data Output' tab shows the results of the query:

| n |
|-----|
| 112 |

A green success message at the bottom right indicates: 'Successfully run. Total query runtime: 179 msec. 1 rows affected.'

Evidencia (Figuras 9.1 y 9.2): resultados de validación (conteo y muestra de registros).

1.4 Transformación IN_MUNICIPALITIES (carga de **municipalities.csv** → **stg_municipalities**)

Para el fichero **municipalities.csv** seguí el mismo enfoque, creando una transformación con:

- CSV file input:** lectura de \${DIR_IN}\municipalities.csv, con delimitador coma, cabecera y codificación UTF-8, usando **Get Fields** y **Preview** para verificar la lectura.
- Table output:** inserción en **dbo.stg_municipalities**, marcando **Truncate table** y ajustando el mapeo de campos con la tabla de destino.

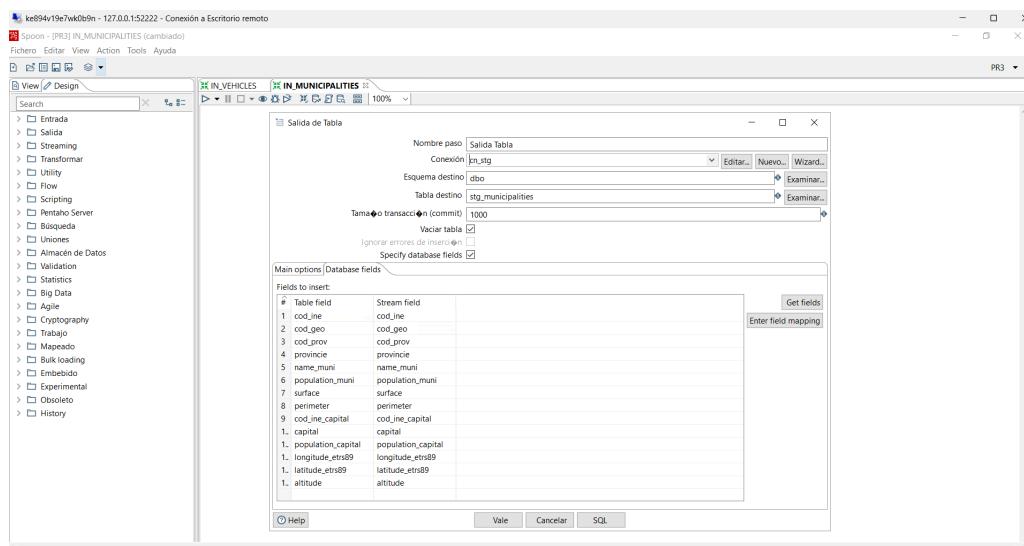
The screenshot shows the Apache Spoon interface for a Data Integration job. It consists of two main windows:

- Top Window (CSV file input configuration):**
 - Step name: CSV_municipalities
 - Filename: \${DIR_IN}/municipalities.csv
 - Delimeter: ,
 - Encoding: UTF-8
 - Format: mixed
 - File encoding: UTF-8
 - Fields definition table:

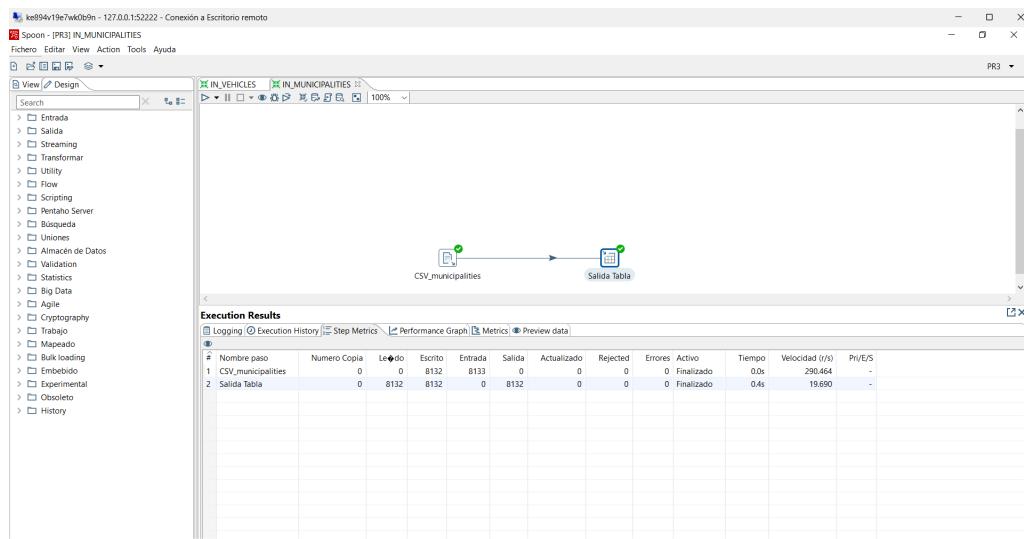
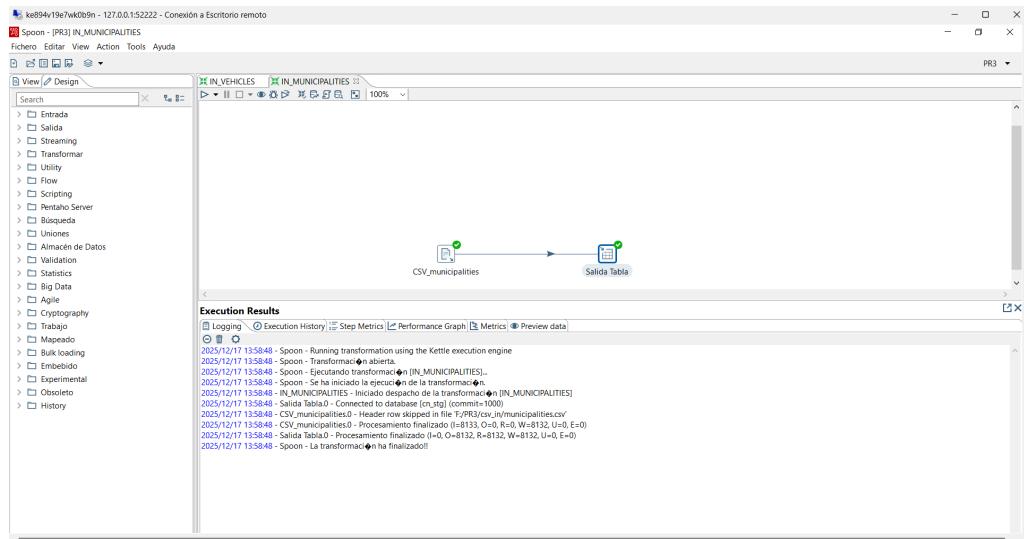
| # | Name | Type | Format | Length | Precision | Currency | Decimal | Group | Trim type |
|----|--------------------|---------|--------|--------|-----------|----------|---------|-------|-----------|
| 1 | cod_ue | Integer | # | 15 | 0 | € | , | . | nninguo |
| 2 | cod_geo | Integer | # | 15 | 0 | € | , | . | nninguo |
| 3 | cod_prov | Integer | # | 15 | 0 | € | , | . | nninguo |
| 4 | province | String | | 11 | | € | , | . | nninguo |
| 5 | name_muni | String | | 30 | | € | , | . | nninguo |
| 6 | population_muni | Integer | # | 15 | 0 | € | , | . | nninguo |
| 7 | surface | Number | #.# | 10 | 4 | € | , | . | nninguo |
| 8 | perimeter | Integer | # | 15 | 0 | € | , | . | nninguo |
| 9 | cod_ue_capital | Integer | # | 15 | 0 | € | , | . | nninguo |
| 10 | capital | String | | 42 | | € | , | . | nninguo |
| 11 | population_capital | Integer | # | 15 | 0 | € | , | . | nninguo |
| 12 | longitude_etrs89 | Number | #.# | 12 | 9 | € | , | . | nninguo |
| 13 | latitude_etrs89 | Number | #.# | 11 | 8 | € | , | . | nninguo |
| 14 | altitude | Integer | # | 15 | 0 | € | , | . | nninguo |
- Bottom Window (Preview data):**
 - Job: ke994v19e7wk0t9n - 127.0.0.1:52222 - Conexión a Escritorio remoto
 - Table: IN_MUNICIPALITIES (cambiado)
 - Fields: cod_ue, cod_geo, cod_prov, province, name_muni, population_muni, surface, perimeter, cod_ue_capital, capital, population_capital, longitude_etrs89, latitude_etrs89, altitude
 - Preview data table:

| # | cod_ue | cod_geo | cod_prov | province | name_muni | population_muni | surface | perimeter | cod_ue_capital | capital | population_capital | longitude_etrs89 | latitude_etrs89 | altitude |
|----|------------|---------|----------|-------------|-----------------------|-----------------|---------|-----------|----------------|------------------------|--------------------|------------------|-----------------|----------|
| 1 | 1001000000 | 1010 | 1 | Araba/Álava | Alegia-Dulantzi | 2975 | 1994.6 | 53569 | 1001000101 | Alegia-Dulantzi | 2860 | -2.5 | 43.8 | 568 |
| 2 | 1003000000 | 1020 | 1 | Araba/Álava | Amurrio | 10313 | 9629.7 | 65381 | 1003000201 | Amurrio | 9338 | -3 | 43.1 | 219 |
| 3 | 1003000000 | 1030 | 1 | Araba/Álava | Aramaio | 1409 | 7309 | 42097 | 1003000601 | Ibarra | 758 | -2.6 | 43.1 | 333 |
| 4 | 1004000000 | 1040 | 1 | Araba/Álava | Artziniega | 1832 | 2728.7 | 22886 | 1004000101 | Artziniega | 1697 | -3.1 | 43.1 | 210 |
| 5 | 1006000000 | 1060 | 1 | Araba/Álava | Armiñón | 232 | 1297.3 | 24707 | 1006000101 | Armiñón | 113 | -2.9 | 42.7 | 467 |
| 6 | 1008000000 | 1080 | 1 | Araba/Álava | Arratzua-Ubarrundia | 1043 | 5753.2 | 67180 | 1008000505 | Durana | 389 | -2.6 | 42.9 | 528 |
| 7 | 1009000000 | 1090 | 1 | Araba/Álava | Asparrena | 1609 | 6510.8 | 53754 | 1009000401 | Arria | 1207 | -2.3 | 42.9 | 602 |
| 8 | 1010000000 | 1100 | 1 | Araba/Álava | Ayala/Aiara | 2918 | 14096.9 | 64211 | 1010002001 | Arespaldisa/Respaldisa | 414 | -3 | 43.1 | 306 |
| 9 | 1011000000 | 1110 | 1 | Araba/Álava | Baños de Ebro/Mafueta | 295 | 950.4 | 18666 | 1011000101 | Baños de Ebro/Mafueta | 295 | -2.7 | 42.5 | 421 |
| 10 | 1013000000 | 1130 | 1 | Araba/Álava | Barrenda | 897 | 9729.7 | 57551 | 1013001401 | Ozeta | 210 | -2.5 | 42.9 | 575 |

Evidencia (Figuras 10.1 y 10.2): configuración y previsualización del CSV file input.



Evidencia (Figura 11): configuración del Table Output (tabla `stg_municipalities`, truncate y mapeo).



Evidencia (Figuras 12.1 y 12.2): métricas y log de ejecución sin errores.

Validación en PostgreSQL

Después de ejecutar, comprobé la carga con:

```
SELECT COUNT(*) AS n FROM dbo.stg_municipalities;  
SELECT * FROM dbo.stg_municipalities LIMIT 10;
```

The screenshot shows the Object Explorer on the left with the database 'dbo' selected. The central pane displays a query window with the following SQL code:

```
1 SELECT *  
2 FROM dbo.stg_municipalities  
3 LIMIT 10;
```

The Data Output tab shows the results of the query, which lists 10 rows of data from the 'stg_municipalities' table. The columns include: cod_muni (bigint), cod_geo (integer), cod_prov (integer), provincia (character varying(50)), name_muni (character varying(100)), population_muni (integer), surface (numeric(10,4)), perimeter (integer), cod_line_capital (bigint), and capital (character varying(50)). The results show data for municipalities like Araba/Alava, Amurrio, Aramalo, Artziniega, Armiñón, Arratzua-Ubarundia, Alegria-Dulantzi, Ibarra, and Durana.

The screenshot shows the Object Explorer on the left with the database 'dbo' selected. The central pane displays a query window with the following SQL code:

```
1 SELECT COUNT(*) AS n  
2 FROM dbo.stg_municipalities  
3
```

The Data Output tab shows the results of the query, which displays a single row with the value 'n' (bigint) set to 8132. A message at the bottom right indicates 'Successfully run. Total query runtime: 301 msec. 1 rows affected.'

Evidencia (Figuras 13.1 y 13.2): resultados de validación (conteo y muestra de registros).

1.5 Transformación IN_VEHICLES (carga de vehicles.csv → stg_vehicles)

En esta parte diseñé la transformación para cargar el fichero `vehicles.csv`, que es el más voluminoso y el que requiere un ajuste adicional por el campo `new_used`.

Diseño de la transformación

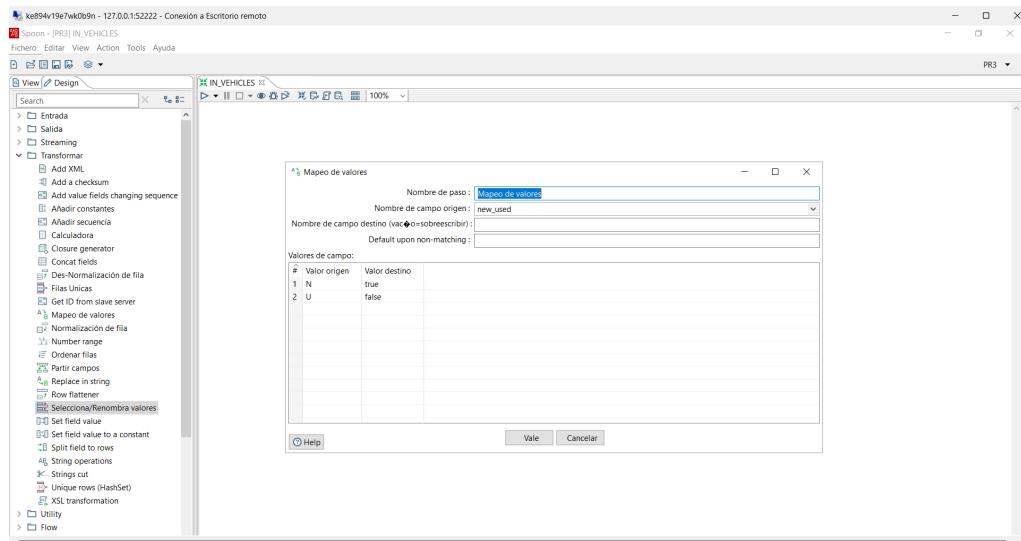
La transformación quedó estructurada en los siguientes pasos:

1. **CSV file input:** lectura de `${DIR_IN}\vehicles.csv` con delimitador coma, cabecera y codificación **UTF-8**. Obtuve los campos con **Get Fields** y realicé una **previsualización** para comprobar que la lectura era correcta.
2. **Mapeo de valores (new_used):** transformé los valores del fichero (`N` / `U`) a valores booleanos para que fueran coherentes con el tipo de dato de la tabla en PostgreSQL:
 - o `N → true`
 - o `U → false`
3. **Selecciona/Renombra valores:** utilicé este paso para asegurar que `new_used` quedaba definido como **Boolean** en el flujo antes de cargar en la tabla.
4. **Table output:** carga en `dbo.stg_vehicles`, activando **Truncate table**, configurando el **commit** y dejando el mapeo de campos alineado con la tabla.

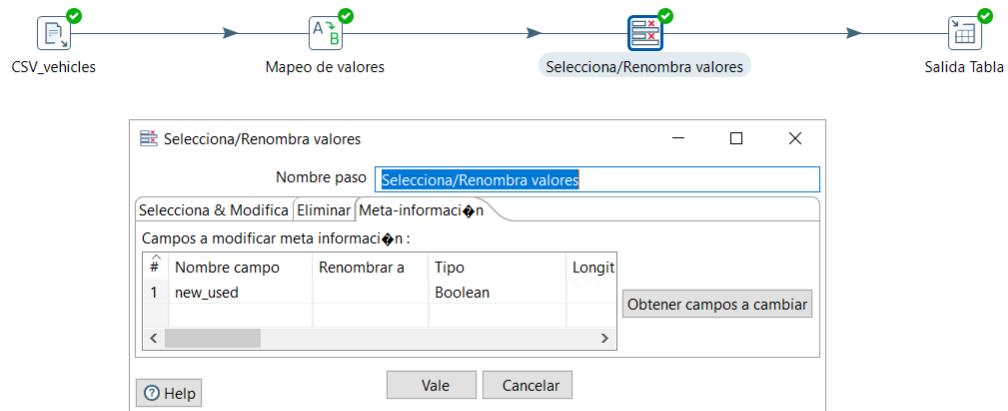
The screenshot shows the Talend Data Integration environment. On the left, the 'Design' tab is selected, displaying a tree view of the transformation steps: Entrada, Salida, Streaming, Transformar, and specific steps like CSV file input, Mapeo de valores, and Table output. The 'CSV file input' step is currently active, showing its configuration details. The 'Step name' is 'CSV_vehicles', 'Filename' is '\$DIR_IN\vehicles.csv', 'Delimiter' is ',', 'Enclosure' is ''', 'NIO buffer size' is 50000, and 'File encoding' is UTF-8. The 'Get Fields' section lists seven fields: id (Integer), municipality (String), brand (String), model (String), registration_date (Date), new_used (String), and subtype (String). The 'Preview' tab is also visible, showing a sample of 10 rows from the CSV file. The rows are as follows:

| # | id | municipality | brand | model | registration_date | new_used | subtype |
|----|--------|--------------|-------|----------------|-------------------|----------|---------|
| 1 | 303304 | B156 | FORD | TRANSIT CUSTOM | 2020-03-01 | N | 20 |
| 2 | 303305 | B156 | FORD | TRANSIT CUSTOM | 2020-03-01 | N | 20 |
| 3 | 303306 | B156 | FORD | TRANSIT CUSTOM | 2020-03-01 | N | 20 |
| 4 | 303307 | B156 | FORD | TRANSIT CUSTOM | 2020-03-01 | N | 20 |
| 5 | 303308 | B156 | FORD | TRANSIT CUSTOM | 2020-03-01 | N | 20 |
| 6 | 303309 | B156 | FORD | TRANSIT CUSTOM | 2020-03-01 | N | 20 |
| 7 | 303310 | B156 | FORD | TRANSIT CUSTOM | 2020-03-01 | N | 20 |
| 8 | 303311 | B156 | FORD | TRANSIT CUSTOM | 2020-03-01 | N | 20 |
| 9 | 303312 | B156 | FORD | TRANSIT CUSTOM | 2020-03-01 | N | 20 |
| 10 | 303313 | B156 | FORD | TRANSIT CUSTOM | 2020-03-01 | N | 20 |

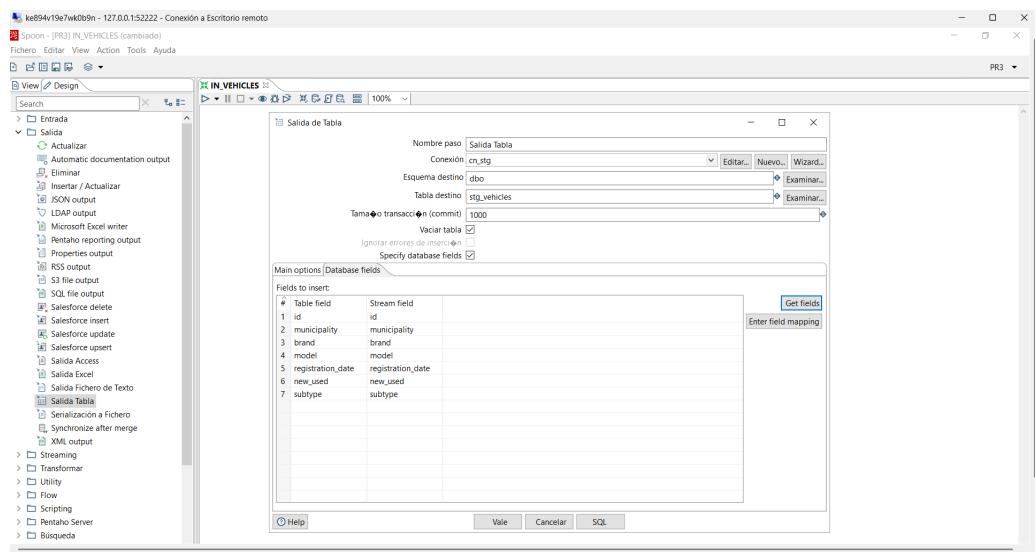
Evidencia (Figuras 14.1 y 14.2): configuración y preview del CSV file input de `vehicles.csv`.



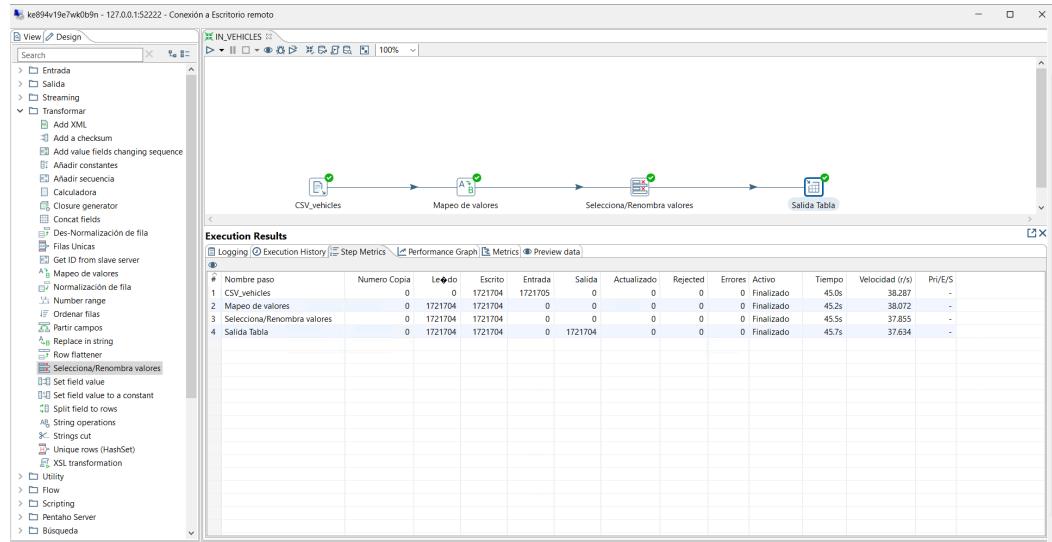
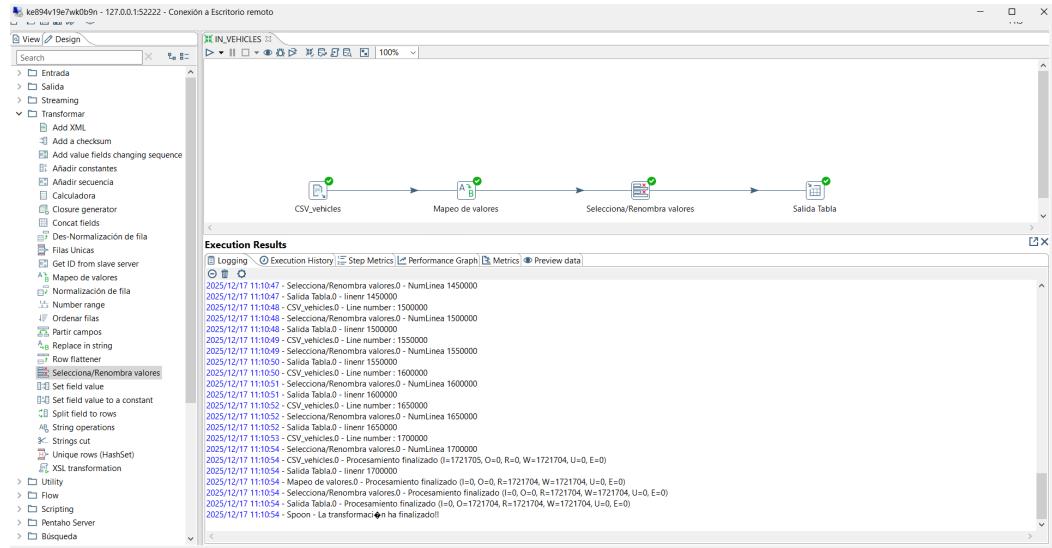
Evidencia (Figura 15): mapeo de valores para `new_used` (`N`→`true`, `U`→`false`).



Evidencia (Figura 16): paso de selección/renombrado asegurando el tipo booleano de `new_used`.



Evidencia (Figura 17): configuración del Table Output (tabla stg_vehicles, truncate y mapeo).



Evidencia (Figuras 18.1 y 18.2): Step Metrics final con registros leídos/escritos y log sin errores.

Validación en PostgreSQL

Tras la ejecución, validé que la carga fuese completa y que `new_used` quedara con valores coherentes:

```
SELECT COUNT(*) AS n FROM dbo.stg_vehicles;
```

```
SELECT new_used, COUNT(*)
FROM dbo.stg_vehicles
GROUP BY new_used
ORDER BY new_used;
```

The screenshot shows the pgAdmin 4 interface with a connection to 'SOURCE_suave/STUDENT_suave@PostgreSQL_PR1'. The left sidebar shows the schema structure under 'dbo'. The main window has a 'Query' tab with the following SQL code:

```
1 SELECT * FROM dbo.stg_vehicles LIMIT 10;
```

The results pane displays 10 rows of data from the 'stg_vehicles' table, showing columns: id, municipality, brand, model, registration_date, new_used, and subtype. All 'new_used' values are set to 'true'. A message at the bottom right indicates the query was successfully run with a runtime of 329 msec and 10 rows affected.

The screenshot shows the pgAdmin 4 interface with the same connection. The left sidebar shows the schema structure under 'dbo'. The main window has a 'Query' tab with the following SQL code:

```
1 SELECT COUNT(*) AS n FROM dbo.stg_vehicles;
2
3 SELECT new_used, COUNT(*)
4   FROM dbo.stg_vehicles
5  GROUP BY new_used
6 ORDER BY new_used;
```

The results pane shows two rows of data. The first row is for 'false' with a count of 28044. The second row is for 'true' with a count of 1693660. A message at the bottom right indicates the query was successfully run with a runtime of 610 msec and 2 rows affected.

Evidencia (Figuras 19.1 y 19.2): resultados de validación (conteo total y distribución por `new_used`).

1.6 Conclusión

Con las tres transformaciones ejecutadas, he podido cargar correctamente los ficheros `dgt_type.csv`, `municipalities.csv` y `vehicles.csv` en la capa de *staging* de PostgreSQL. La ejecución en Spoon finaliza sin errores y las consultas en pgAdmin confirman que los datos están insertados en las tablas `stg_` correspondientes y con el formato esperado, especialmente en el caso de `vehicles` con la conversión del campo `new_used` a booleano.

Ejercicio 2

En este ejercicio mi objetivo ha sido **evaluar la calidad y consistencia** de los datos cargados en la capa *staging* (PostgreSQL) para **anticipar problemas** antes de construir la capa de integración / modelo analítico. Para ello he realizado un **perfilado sistemático** sobre `dbo.stg_municipalities` y `dbo.stg_vehicles`, revisando: volumetría, completitud (nulos), cardinalidad (valores únicos), rangos y longitudes, atípicos básicos y coherencia de claves/códigos entre tablas.

2.1 Perfilado de dbo.stg_municipalities

2.1.1 M1 — Volumetría total

En primer lugar calculé el número total de registros en `dbo.stg_municipalities` para disponer de una referencia base sobre la que interpretar nulos y porcentajes.

psu340j5z3bv - 127.0.0.1:16120\$ Conexión a Escritorio remoto

Object Explorer SOURCE_stmstns SOURCE_suave

Query Query History

```
1 SELECT COUNT(*) AS total_rows
2 FROM dbo.stg_municipalities;
3
```

Data Output Messages Notifications

| total_rows | bright |
|------------|--------|
| 1 | 8132 |

Total rows: 1 of 1 Query complete: 00:00:00.274 ✓ Successfully run. Total query runtime: 374 msec. 1 rows affected. ✘

Evidencia (Figura M1): resultado de COUNT(*) con la volumetría total de dbo.stg_municipalities.

2.1.2 M2 — Análisis de calidad por columna (completitud, nulos, únicos, longitudes y min–max)

A continuación ejecuté un bloque de perfilado por columna para cuantificar:

- **No nulos y nulos** (y su %).
 - **Valores únicos** (cardinalidad).
 - Para variables de texto: **mínimo/máximo de longitud**.
 - Para variables numéricas/fecha: **mínimo/máximo**.

Este análisis me permite detectar rápidamente columnas problemáticas (por ejemplo, alta proporción de nulos, cardinalidades sospechosas o rangos incoherentes).

| | columna | tipo | no_nulos | nulos | porcentaje_nulos | valores_unicos | min_length | max_length | min_valor | max_valor |
|----|--------------------|---------|----------|-------|------------------|----------------|------------|------------|---------------|-------------|
| 1 | altitude | numeric | 8132 | 0 | 0.00 | 1361 | [null] | [null] | 1 | 1695 |
| 2 | cod_geo | numeric | 8132 | 0 | 0.00 | 8124 | [null] | [null] | 0 | 52001 |
| 3 | cod_line | numeric | 8132 | 0 | 0.00 | 8132 | [null] | [null] | 1001000000 | 52001000000 |
| 4 | cod_ine_capital | numeric | 8132 | 0 | 0.00 | 8132 | [null] | [null] | 1001000101 | 52001000301 |
| 5 | cod_prov | numeric | 8132 | 0 | 0.00 | 52 | [null] | [null] | 1 | 52 |
| 6 | latitude_etrs89 | numeric | 8132 | 0 | 0.00 | 8131 | [null] | [null] | 27.70386872 | 43.74035104 |
| 7 | longitude_etrs89 | numeric | 8132 | 0 | 0.00 | 8132 | [null] | [null] | -17.998381480 | 4.239666290 |
| 8 | perimeter | numeric | 8132 | 0 | 0.00 | 7590 | [null] | [null] | 0 | 624452 |
| 9 | population_capital | numeric | 8132 | 0 | 0.00 | 3100 | [null] | [null] | 0 | 3332035 |
| 10 | population_muni | numeric | 8132 | 0 | 0.00 | 3569 | [null] | [null] | 3 | 3332035 |
| 11 | surface | numeric | 8132 | 0 | 0.00 | 8128 | [null] | [null] | 0.0000 | 175022.9100 |
| 12 | capital | text | 8132 | 0 | 0.00 | 8069 | 2 | 47 | [null] | [null] |
| 13 | name_muni | text | 8132 | 0 | 0.00 | 8115 | 2 | 47 | [null] | [null] |
| 14 | province | text | 8132 | 0 | 0.00 | 52 | 4 | 22 | [null] | [null] |

Evidencia (Figura M2): tabla completa del perfilado por columnas (nulos, % nulos, únicos, longitudes y min/max) para `dbo.stg_municipalities`.

2.1.3 M3 — Comprobaciones de posibles valores atípicos (numéricas)

Para complementar el perfilado anterior, realicé comprobaciones directas sobre condiciones anómalas típicas:

- Valores **negativos** en población.
- Valores **cero o negativos** en variables donde no tendría sentido (superficie, perímetro).
- Coordenadas fuera de rangos generales (`longitude_etrs89` y `latitude_etrs89`).

psu340jdj5zx3bv - 127.0.0.1:51205 - Conexión a Escritorio remoto

Object Explorer SOURCE_stomans SOURCE_suave

```

1 -- negativos o ceros donde no tendría sentido
2 SELECT *
3 FROM dbo.stg_municipalities
4 WHERE population_muni < 0
5 OR population_capital < 0
6 OR surface <= 0
7 OR perimeter <= 0;
8

```

Data Output Messages Notifications

| | cod_line | cod_geo | cod_prov | provincia | name_muni | population_muni | surface | perimeter | cod_line_capital | capital |
|---|------------|---------|----------|-----------|-----------|-----------------|---------|-----------|------------------|----------|
| 1 | 4891600000 | 0 | 48 | Bizkaia | Usansolo | 4512 | 0.0000 | 0 | 489160001 | Usansolo |

Total rows: 1 of 1 Query complete: 00:00:00 387 Successfully run. Total query runtime: 387 msec. 1 rows affected.

psu340jdj5zx3bv - 127.0.0.1:51205 - Conexión a Escritorio remoto

Object Explorer SOURCE_stomans SOURCE_suave

```

1 -- coordenadas fuera de rangos generales
2 SELECT *
3 FROM dbo.stg_municipalities
4 WHERE longitude_etr89 NOT BETWEEN -180 AND 180
5 OR latitude_etr89 NOT BETWEEN -90 AND 90;
6

```

Data Output Messages Notifications

| | cod_line | cod_geo | cod_prov | provincia | name_muni | population_muni | surface | perimeter | cod_line_capital | capital |
|---|----------|---------|----------|-----------|-----------|-----------------|---------|-----------|------------------|---------|
| 1 | | | | | | | | | | |

Total rows: 0 of 0 Query complete: 00:00:00 460 Successfully run. Total query runtime: 460 msec. 0 rows affected.

Evidencia (Figuras M3.1 y M3.2): resultado de las consultas de detección de atípicos (0 filas o filas encontradas).

2.1.4 M4 — Consistencia/estandarización en texto (mayúsculas/minúsculas y espacios)

Con el objetivo de anticipar problemas de integración (joins) y de calidad dimensional, verifiqué si existen variantes textuales debidas a mayúsculas/minúsculas o espacios (normalización con `UPPER(TRIM())`), usando como ejemplo `provincie`.

```

1 SELECT UPPER(TRIM(provincie)) AS provincie_norm,
2        COUNT(DISTINCT provincie) AS variantes
3  FROM dbo.stg_municipalities
4 GROUP BY 1
5 HAVING COUNT(DISTINCT provincie) > 1
6 ORDER BY variantes DESC, provincie_norm
7 LIMIT 20;
8

```

| provincie_norm | variantes |
|----------------|-----------|
| BOLIVIA | 28 |

Successfully run. Total query runtime: 195 msec. 0 rows affected.

Evidencia (Figura M4): resultado del análisis de variantes en **provincie** (agrupación por **UPPER(TRIM(provincie))**) y recuento de variantes).

2.1.5 M5 — Caso detectado: **surface/perimeter = 0** (cuantificación y detalle)

Como parte del control de valores no plausibles, cuantifíqué específicamente los registros con **surface = 0** o **perimeter = 0** y luego inspeccioné el/los registros para documentar el caso y justificar acciones posteriores (por ejemplo, tratar ceros como “no informado”, revisar fuente o transformar a **NULL** en integración).

Evidencia (Figuras M5.1 y M5.1):

```

1 -- M5.1: cuántos casos hay
2 SELECT
3    COUNT(*) AS casos_surface_o_perimeter_cero
4  FROM dbo.stg_municipalities
5 WHERE surface = 0 OR perimeter = 0;

```

| casos_surface_o_perimeter_cero |
|--------------------------------|
| 1 |

Successfully run. Total query runtime: 400 msec. 1 rows affected.

Resultado del conteo de casos con **surface = 0 OR perimeter = 0**.

```

-- M5.2: vea qué filas son
SELECT
    cod_ine, cod_geo, province, name_muni,
    population_muni, surface, perimeter
FROM dbo.stg_municipalities
WHERE surface = 0 OR perimeter = 0;

```

| | cod_ine | cod_geo | province | name_muni | population_muni | surface | perimeter |
|---|------------|---------|----------|-----------|-----------------|---------|-----------|
| 1 | 4891600000 | 0 | Bizkaia | Usansolo | 4512 | 0.0000 | 0 |

Successfully run. Total query runtime: 161 msec. 1 rows affected.

Detalle del/los registros afectados con campos clave.

2.1.6 M6 — Duplicidades en identificadores (control de integridad)

Para comprobar posibles problemas de integridad y evitar duplicidades en dimensiones, verifíqué duplicados en los identificadores principales:

- Duplicados por `cod_ine`.
- Duplicados por `cod_geo`.

The screenshot shows the pgAdmin 4 interface with two panes. The left pane is the Object Explorer displaying various database schemas and objects. The right pane is the Query Editor showing the results of a SQL query.

```

-- duplicados por cod_ine
SELECT cod_ine, COUNT(*) AS n
FROM dbo.stg_municipalities
GROUP BY cod_ine
HAVING COUNT(*) > 1
ORDER BY n DESC;

```

The Data Output pane shows the results:

| cod_ine | n |
|---------|--------|
| bignum | bignum |

Total rows: 0 of 0 Query complete 00:00:00.378 Ln 6, Col 1

The screenshot shows the pgAdmin 4 interface with two panes. The left pane is the Object Explorer displaying various database schemas and objects. The right pane is the Query Editor showing the results of a SQL query.

```

-- duplicados por cod_geo
SELECT cod_geo, COUNT(*) AS n
FROM dbo.stg_municipalities
GROUP BY cod_geo
HAVING COUNT(*) > 1
ORDER BY n DESC;

```

The Data Output pane shows the results:

| cod_geo | n |
|---------|--------|
| integer | bignum |
| 1 | 0 |
| | 9 |

Total rows: 1 of 1 Query complete 00:00:00.202 Ln 6, Col 1

Evidencia (Figuras M6.1 y M6.2): resultado de las consultas de duplicados para `cod_ine` y `cod_geo` (0 filas si no hay duplicados, o listado si existen).

2.1.7 M7 — Coherencia de la “capital” (referencia interna en staging)

Finalmente revisé la coherencia interna de `cod_ine_capital`, comprobando si las referencias de capital existen como `cod_ine` dentro de la propia tabla `dbo.stg_municipalities`. Este control es relevante para asegurar consistencia antes de crear relaciones entre dimensiones.

```

1 SELECT cod_ine_capital, COUNT(*) AS n_refs
2 FROM dbo.stg_municipalities
3 WHERE cod_ine_capital IS NOT NULL
4 AND cod_ine_capital NOT IN (SELECT cod_ine FROM dbo.stg_municipalities)
5 GROUP BY cod_ine_capital
6 ORDER BY n_refs DESC;
7

```

| cod_ine_capital | n.refs |
|-----------------|--------|
| 49145003031 | 1 |
| 21061000101 | 1 |
| 42174000301 | 1 |
| 45028000201 | 1 |
| 1054000701 | 1 |
| 4402800101 | 1 |
| 21006000101 | 1 |
| 1615200101 | 1 |
| 1043000301 | 1 |
| 2803100101 | 1 |
| 1905700101 | 1 |
| 43098003031 | 1 |
| 4514600101 | 1 |
| 8256000102 | 1 |

Total rows: 1000 of 8132 Query complete 00:00:00.414 1 in 7 Col

Evidencia (Figura M7): resultado de la consulta de `cod_ine_capital` referenciando valores inexistentes en `cod_ine` (0 filas idealmente; si hay filas, queda documentado como incidencia).

2.2 Perfilado de `dbo.stg_vehicles`

2.2.1 V1 — Volumetría total

Calculé el número total de registros en `dbo.stg_vehicles`, que además sirve como referencia para interpretar nulos y porcentajes en el resto de validaciones.

```

1 SELECT COUNT(*) AS total_rows
2 FROM dbo.stg_vehicles;
3

```

| total_rows |
|------------|
| 1721704 |

Total rows: 1 of 1 Query complete 00:00:01.582 1 in 7 Col

Evidencia (Figura V1): resultado de `COUNT(*)` con la volumetría total de `dbo.stg_vehicles`.

2.2.2 V2 — Análisis de calidad por columna (completitud, nulos, únicos, longitudes y min–max)

Ejecuté un perfilado por columna equivalente al realizado en municipalities, adaptado a las columnas reales de `stg_vehicles`:

- Para columnas de texto (`municipality`, `brand`, `model`, `subtype`): no nulos, únicos, longitudes mín/máx.
- Para `id`: no nulos, únicos, min/max.
- Para `registration_date`: no nulos, únicos, min/max.
- Para `new_used`: no nulos y cardinalidad (true/false).

The screenshot shows the SSMS interface with the following details:

- Object Explorer:** Shows the database structure under the `SOURCE_stomash` and `SOURCE_suave` schemas.
- Query Editor:** Contains the following T-SQL code:

```
84    ca.min_length,
85    ca.max_length,
86    ca.min_valor,
87    ca.max_valor
88    FROM col_analysis ca
89    CROSS JOIN total_rows tr
90    ORDER BY ca.tipo, ca.columna;
```
- Data Output:** Displays the results of the query in a grid format. The columns are: columna, tipo, no_nulos, nulos, porcentaje_nulos, valores_unicos, min_length, max_length, min_valor, max_valor. The data rows are:

| columna | tipo | no_nulos | nulos | porcentaje_nulos | valores_unicos | min_length | max_length | min_valor | max_valor |
|-------------------|---------|----------|-------|------------------|----------------|------------|------------|------------|------------|
| new_used | boolean | 1721704 | 0 | 0.00 | 2 | [null] | [null] | [null] | [null] |
| registration_date | date | 1721704 | 0 | 0.00 | 694 | [null] | [null] | 2020-01-01 | 2024-12-11 |
| id | numeric | 1721704 | 0 | 0.00 | 1721704 | [null] | [null] | 1 | 3471091 |
| brand | text | 1721704 | 0 | 0.00 | 127 | 2 | 22 | [null] | [null] |
| model | text | 1721704 | 0 | 0.00 | 1045 | 1 | 22 | [null] | [null] |
| municipality | text | 1721704 | 0 | 0.00 | 27 | 5 | 5 | [null] | [null] |
| subtype | text | 1721704 | 0 | 0.00 | 47 | 2 | 2 | [null] | [null] |

Status Bar: Successfully run. Total query runtime: 9 secs 316 msec. 7 rows affected.

Evidencia (Figura V2): tabla completa del perfilado por columnas para `dbo.stg_vehicles`.

2.2.3 V3 — Controles de particularidades (integridad y plausibilidad)

Para reforzar el perfilado, realicé comprobaciones específicas:

V3.1 — Duplicados de `id`

Verifiqué si el `id` se repite.

```

psu340jj5z3bv - 127.0.0.1:61205 - Conexión a Escritorio remoto
Object Explorer SOURCE_stoman SOURCE_suave
Query History
1 SELECT id, COUNT(*) AS n
2 FROM dbo.stg_vehicles
3 GROUP BY id
4 HAVING COUNT(*) > 1
5 ORDER BY n DESC;
6

Data Output Messages Notifications
id n
1 28044

Total rows: 0 of 0 Query complete at 00:00:01.829 ✓ Successfully run. Total query runtime: 1 secs 832 msec. 0 rows affected. In 6 Ctrl

```

Evidencia (Figura V3.1): resultado de duplicados por `id` (0 filas o listado de ids duplicados).

V3.2 — Distribución de `new_used`

Comprobé la distribución de valores del booleano `new_used` para validar coherencia y detectar posibles nulos o categorías no esperadas.

```

psu340jj5z3bv - 127.0.0.1:61205 - Conexión a Escritorio remoto
Object Explorer SOURCE_stoman SOURCE_suave
Query History
1 SELECT new_used, COUNT(*) AS n
2 FROM dbo.stg_vehicles
3 GROUP BY new_used
4 ORDER BY new_used;
5

Data Output Messages Notifications
new_used n
false 28044
true 1693660

Total rows: 2 of 2 Query complete at 00:00:00.402 ✓ Successfully run. Total query runtime: 503 msec. 2 rows affected. In 6 Ctrl

```

Evidencia (Figura V3.2): tabla con el recuento por `new_used` (true/false/(null)).

V3.3 — Fechas fuera de rango lógico

Verifiqué:

- Matrículas en el futuro (`registration_date > CURRENT_DATE`).
- Fechas muy antiguas (`registration_date < 1900-01-01`).

nt922tb62df3n31 - 127.0.0.1:54735 - Conexión a Escritorio remoto

Objetos explorados SOURCE_stellmont SOURCE_schical SOURCE_sergiosalas SOURCE_sfalconna SOURCE_sfemandez36 SOURCE_sjimenezcorra SOURCE_smunozno SOURCE_spalomosa SOURCE_sronquillolog SOURCE_ssamperp SOURCE_stomans SOURCE_suave

SOURCE_suave

Query Query History

```
1 SELECT COUNT(*) AS fechas_futuras
2 FROM dbo.stg_vehicles
3 WHERE registration_date > CURRENT_DATE;
```

Data Output Messages Notifications

| fechas_futuras |
|----------------|
| 0 |

Total rows: 1 of 1 Query complete 00:00:22.383 ✓ Successfully run. Total query runtime: 22 secs 383 msec. 1 rows affected. Ln 3, Col 4

nt922tb62df3n31 - 127.0.0.1:54735 - Conexión a Escritorio remoto

Objetos explorados SOURCE_stellmont SOURCE_schical SOURCE_sergiosalas SOURCE_sfalconna SOURCE_sfemandez36 SOURCE_sjimenezcorra SOURCE_smunozno SOURCE_spalomosa SOURCE_sronquillolog SOURCE_ssamperp SOURCE_stomans SOURCE_suave

SOURCE_suave

Query Query History

```
1 SELECT COUNT(*) AS fechas_muy_antiguas
2 FROM dbo.stg_vehicles
3 WHERE registration_date < DATE '1900-01-01';
```

Data Output Messages Notifications

| fechas_muy_antiguas |
|---------------------|
| 0 |

Total rows: 1 of 1 Query complete 00:00:07.198 ✓ Successfully run. Total query runtime: 7 secs 198 msec. 1 rows affected. Ln 3, Col 4

Evidencia (Figuras V3.3.1 y V3.3.2): resultados de ambos conteos (fechas futuras y fechas muy antiguas).

V3.4 — Variantes textuales en **brand** (normalización)

Analicé si existen variantes de marca debidas a mayúsculas/minúsculas o espacios, que podrían dificultar agrupaciones y análisis posteriores.

```

1 SELECT UPPER(TRIM(brand)) AS brand_norm,
2      COUNT(DISTINCT brand) AS variantes
3   FROM dbo.stg_vehicles
4      GROUP BY 1
5      HAVING COUNT(DISTINCT brand) > 1
6      ORDER BY variantes DESC, brand_norm
7      LIMIT 20;

```

brand_norm variantes
text bigint

✓ Successfully run. Total query runtime: 2 secs 495 msec. 0 rows affected.

Evidencia (Figura V3.4): resultado del análisis de variantes de **brand** mediante **UPPER(TRIM(brand))**.

2.2.4 V4 — Validación de **municipality** (**vehicles**) vs **stg_municipalities**

Dado que **municipality** es un campo crítico para relacionar vehículos con municipios, validé su formato y su capacidad de emparejar con la tabla de municipios.

V4.1 — Formato del campo **municipality** (vacíos / numéricos / no numéricos)

Cuantifiqué:

- Vacíos (**NULL** o cadena vacía).
- Valores compuestos solo por dígitos.
- Valores no numéricos (si existieran).

```

1 SELECT
2     COUNT(*) AS total,
3     COUNT(*) FILTER (WHERE municipality IS NULL OR btrim(municipality) = '') AS vacios,
4     COUNT(*) FILTER (WHERE municipality ~ '^\\d+$') AS solo_digitos,
5     COUNT(*) FILTER (WHERE municipality !~ '^\\d+$' AND municipality IS NOT NULL AND btrim(municipality) ~ '^\\d+$') AS no_numericos
6 FROM dbo.stg_vehicles;

```

| | total | vacios | solo_digitos | no_numericos |
|---|---------|--------|--------------|--------------|
| 1 | 1721704 | 0 | 1721704 | 0 |

Successfully run. Total query runtime: 1 secs 335 msec. 1 rows affected.

Evidencia (Figura V4.1): salida con `total`, `vacios`, `solo_digitos` y `no_numericos`.

V4.2 — Conteo de “no match” contra `cod_geo` (si es numérico)

Asumiendo que `municipality` representa un código numérico, calculé cuántos registros de `vehicles` no encuentran correspondencia en `dbo.stg_municipalities.cod_geo`.

```

1 WITH v AS (
2     SELECT
3         municipality,
4         CASE WHEN municipality ~ '^\\d+$' THEN municipality::int END AS municipality_int
5     FROM dbo.stg_vehicles
6 )
7 SELECT
8     COUNT(*) AS no_match
9 FROM v
10 LEFT JOIN dbo.stg_municipalities m
11     ON v.municipality_int = m.cod_geo
12 WHERE v.municipality_int IS NOT NULL
13     AND m.cod_geo IS NULL;
14

```

| | no_match |
|---|----------|
| 1 | 0 |

Successfully run. Total query runtime: 1 secs 778 msec. 1 rows affected.

Evidencia (Figura V4.2): resultado del conteo `no_match` tras el join con `cod_geo`.

V4.3 — Identificación de códigos no emparejados (top 20)

Para documentar y facilitar corrección, listé los valores de `municipality` que no emparejan (top 20 por frecuencia).

```

psu340jj5zx3bvh - 127.0.0.1:61205 - Conexión a Escritorio remoto
Object Explorer SOURCE_stomans SOURCE_suave
Query History
1 WITH v AS (
2   SELECT
3     municipality,
4     CASE WHEN municipality ~ '^\\d+$' THEN municipality::int END AS municipality_int
5   FROM dbo.stg_vehicles
6 )
7   SELECT
8   v.municipality,
9   COUNT(*) AS n
10  FROM v
11 LEFT JOIN dbo.stg_municipalities m
12   ON v.municipality_int = m.cod_geo
13 WHERE v.municipality_int IS NOT NULL
14   AND m.cod_geo IS NULL
15 GROUP BY v.municipality
16 ORDER BY n DESC
17 LIMIT 20;
18
Data Output Messages Notifications
municipality character varying (5) n bigint

```

Successfully run. Total query runtime: 1 secs 711 msec. 0 rows affected.

Evidencia (Figura V4.3): listado top de códigos `municipality` sin correspondencia y su frecuencia.

2.2.5 V5 — Validación de `subtype` (`vehicles`) contra `stg_dgt_type`

Para asegurar coherencia con el catálogo DGT, validé el campo `subtype` de `vehicles` contra `stg_dgt_type` (normalizando con `UPPER(TRIM())`).

V5.1 — Conteo de valores de `subtype` sin correspondencia

Calculé cuántos registros de `stg_vehicles` tienen un `subtype` que no existe en `stg_dgt_type`.

```

psu340jj5zx3bvh - 127.0.0.1:61205 - Conexión a Escritorio remoto
Object Explorer SOURCE_stomans SOURCE_suave
Query History
1 WITH d AS (
2   SELECT DISTINCT UPPER(TRIM(cod_subtype_dgt)) AS code_norm
3   FROM dbo.stg_dgt_type
4 )
5   SELECT
6   COUNT(*) AS no_match
7   FROM dbo.stg_vehicles v
8   LEFT JOIN d
9   ON UPPER(TRIM(v.subtype)) = d.code_norm
10 WHERE d.code_norm IS NULL;
11
Data Output Messages Notifications
no_match bigint
1 0

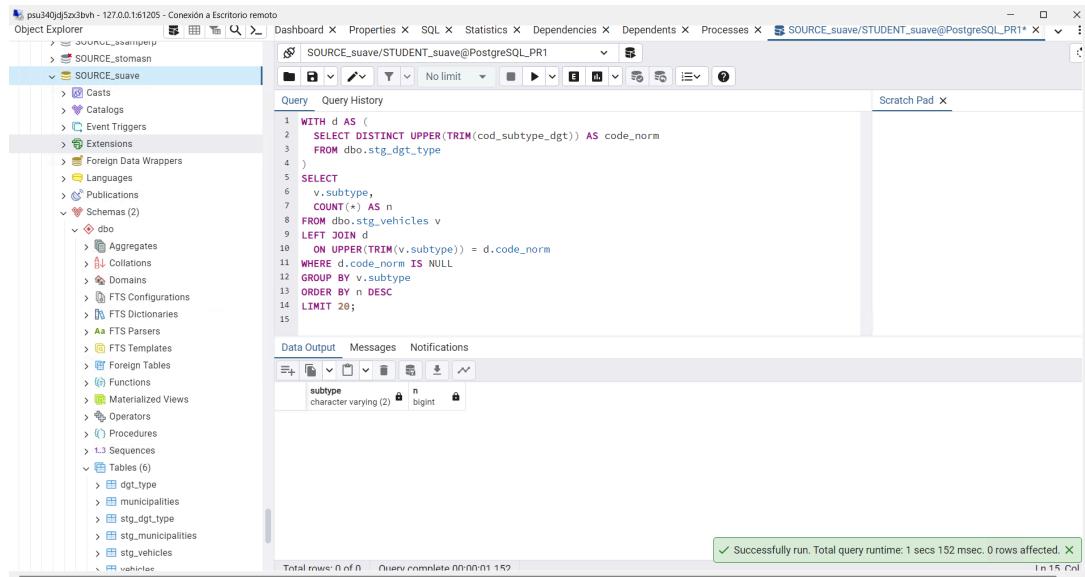
```

Successfully run. Total query runtime: 1 secs 30 msec. 1 rows affected.

Evidencia (Figura V5.1): resultado del conteo `no_match` para `subtype` contra `stg_dgt_type`.

V5.2 — Listado de subtypes problemáticos (top 20)

Listé los valores de `subtype` que no emparejan para identificar rápidamente qué códigos requieren revisión o normalización.



```

psu340jd5zx3bvh - 127.0.0.1:61205 - Conexión a Escritorio remoto
Object Explorer SOURCE_stomash SOURCE_suave
Query History
1 WITH d AS (
2     SELECT DISTINCT UPPER(TRIM(cod_subtype_dgt)) AS code_norm
3     FROM dbo.stg_dgt_type
4 )
5 SELECT
6     v.subtype,
7     COUNT(*) AS n
8     FROM dbo.stg_vehicles v
9     LEFT JOIN d
10    ON UPPER(TRIM(v.subtype)) = d.code_norm
11   WHERE d.code_norm IS NULL
12   GROUP BY v.subtype
13   ORDER BY n DESC
14   LIMIT 20;
15

Data Output Messages Notifications
+-----+-----+
| subtype | n |
| character varying (2) | bigint |
+-----+-----+

```

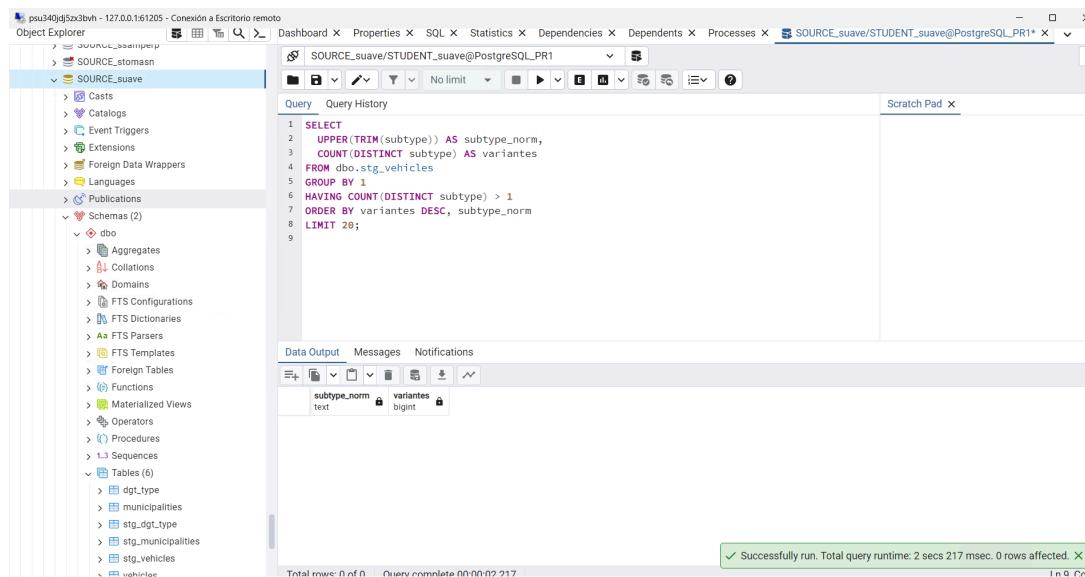
Total rows: 0 of 0 Query complete 00:00:01.152 1 in 15 Col

Successfully run. Total query runtime: 1 secs 152 msec. 0 rows affected. X

Evidencia (Figura V5.2): top 20 de `subtype` sin correspondencia y su frecuencia.

V5.3 — Variantes en `subtype` (mayúsculas/minúsculas/espacios)

Analicé si un mismo subtype aparece escrito con variantes (casos típicos de normalización), lo que puede explicar discrepancias de matching si no se normaliza.



```

psu340jd5zx3bvh - 127.0.0.1:61205 - Conexión a Escritorio remoto
Object Explorer SOURCE_stomash SOURCE_suave
Query History
1 SELECT
2     UPPER(TRIM(subtype)) AS subtype_norm,
3     COUNT(DISTINCT subtype) AS variantes
4     FROM dbo.stg_vehicles
5     GROUP BY 1
6     HAVING COUNT(DISTINCT subtype) > 1
7     ORDER BY variantes DESC, subtype_norm
8     LIMIT 20;
9

Data Output Messages Notifications
+-----+-----+
| subtype_norm | variantes |
| text          | bigint     |
+-----+-----+

```

Total rows: 0 of 0 Query complete 00:00:02.217 1 in 15 Col

Successfully run. Total query runtime: 2 secs 217 msec. 0 rows affected. X

Evidencia (Figura V5.3): resultado del análisis de variantes de `subtype` mediante `UPPER(TRIM(subtype))`.

2.3 Conclusión

Con las comprobaciones M1–M7 y V1–V5.3 he podido realizar un perfilado completo de la capa *staging*:

- He documentado la **volumetría** de `municipalities` y `vehicles`.
- He evaluado la **completitud**, cardinalidad y rangos por columna, identificando posibles puntos de atención.
- He validado plausibilidad básica (atípicos) y **normalización textual** (posibles variantes).
- He verificado coherencias clave para la integración: `municipality` ↔ `cod_geo` y `subtype` ↔ catálogo DGT.
- He dejado evidencias trazables para justificar decisiones de transformación en la siguiente fase (p. ej., normalización con `TRIM/UPPER`, tratamiento de ceros no plausibles y gestión de no matches).

Ejercicio 3

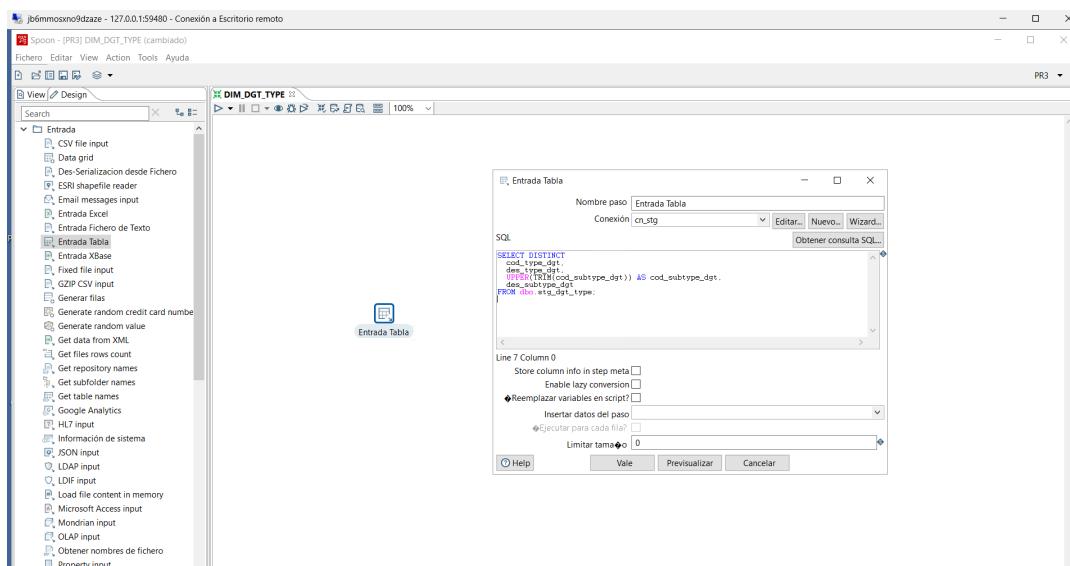
En este ejercicio mi objetivo ha sido profundizar en el uso de **Pentaho Spoon (PDI)** para diseñar y ejecutar procesos de integración que permitan trasladar los datos desde la capa **staging** (PostgreSQL) hacia el **modelo estrella** del Data Warehouse. Para ello he reproducido las transformaciones descritas en la guía (3.1–3.5) y, adicionalmente, he implementado lo solicitado en el apartado 3.6: creación de la dimensión **DIM_MUNICIPALITY**, adaptación de la **FACT** para vincularla con dicha dimensión, y actualización del **JOB_LOAD**.

3.1 Transformación DIM_DGT_TYPE

En esta transformación he cargado los datos de `dbo.stg_dgt_type` a `dbo.dim_dgt_type` mediante los pasos: **Table input** → **Add sequence** → **Table output**, aplicando además normalización del campo `cod_subtype_dgt` con `UPPER()` (y en mi caso también `TRIM()` para robustez).

3.1.1 Lectura de datos en `stg_dgt_type` (**Table input**) + normalización

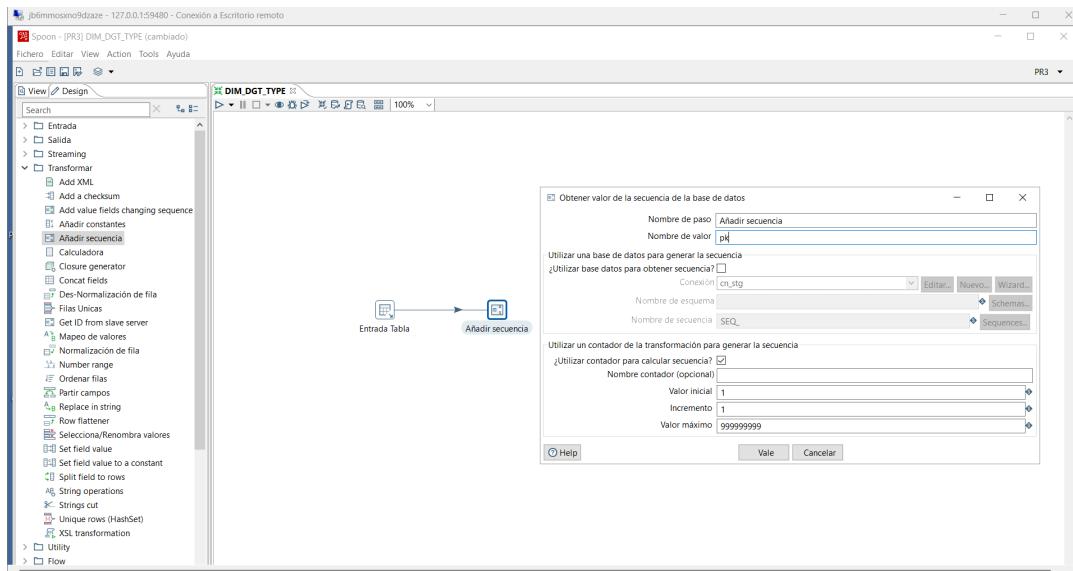
En primer lugar configuro un **Table input** sobre `dbo.stg_dgt_type`, aplicando `SELECT DISTINCT` para evitar duplicidades lógicas y normalizando `cod_subtype_dgt` con `UPPER(TRIM(...))`, con el objetivo de unificar variantes equivalentes (por ejemplo, `s3` vs `S3`) antes de cargar la dimensión.



Evidencia (Figura D2): configuración del paso *Table input* con la consulta SQL visible.

3.1.2 Añadir secuencia (Add sequence) para PK sustituta

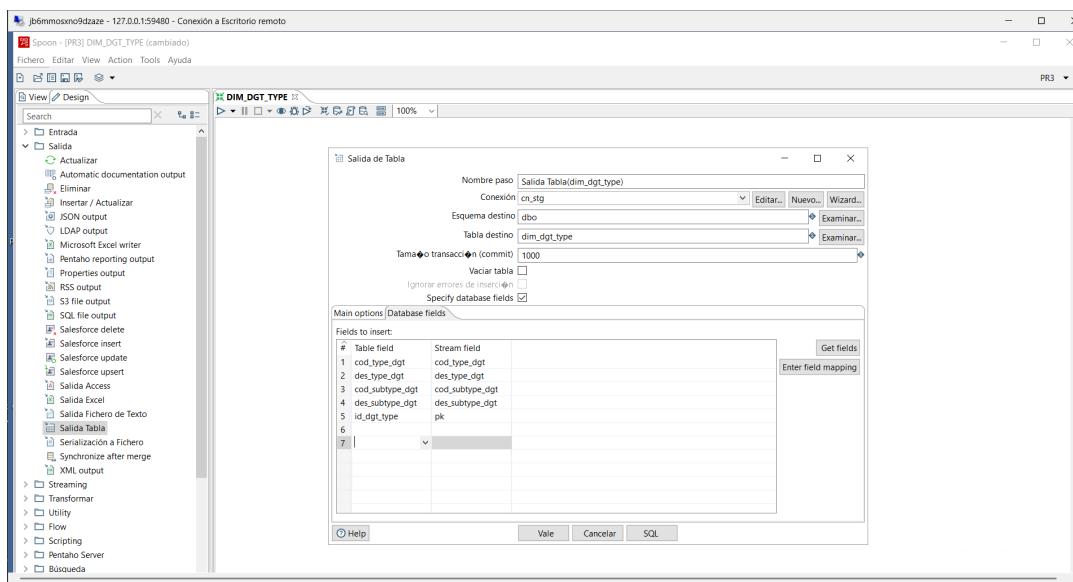
A continuación utilicé **Add sequence** para generar una clave sustituta incremental que actuase como PK de la dimensión.



Evidencia (Figura D3): configuración del paso *Add sequence*.

3.1.3 Carga en dim_dgt_type (Table output) + mapeo

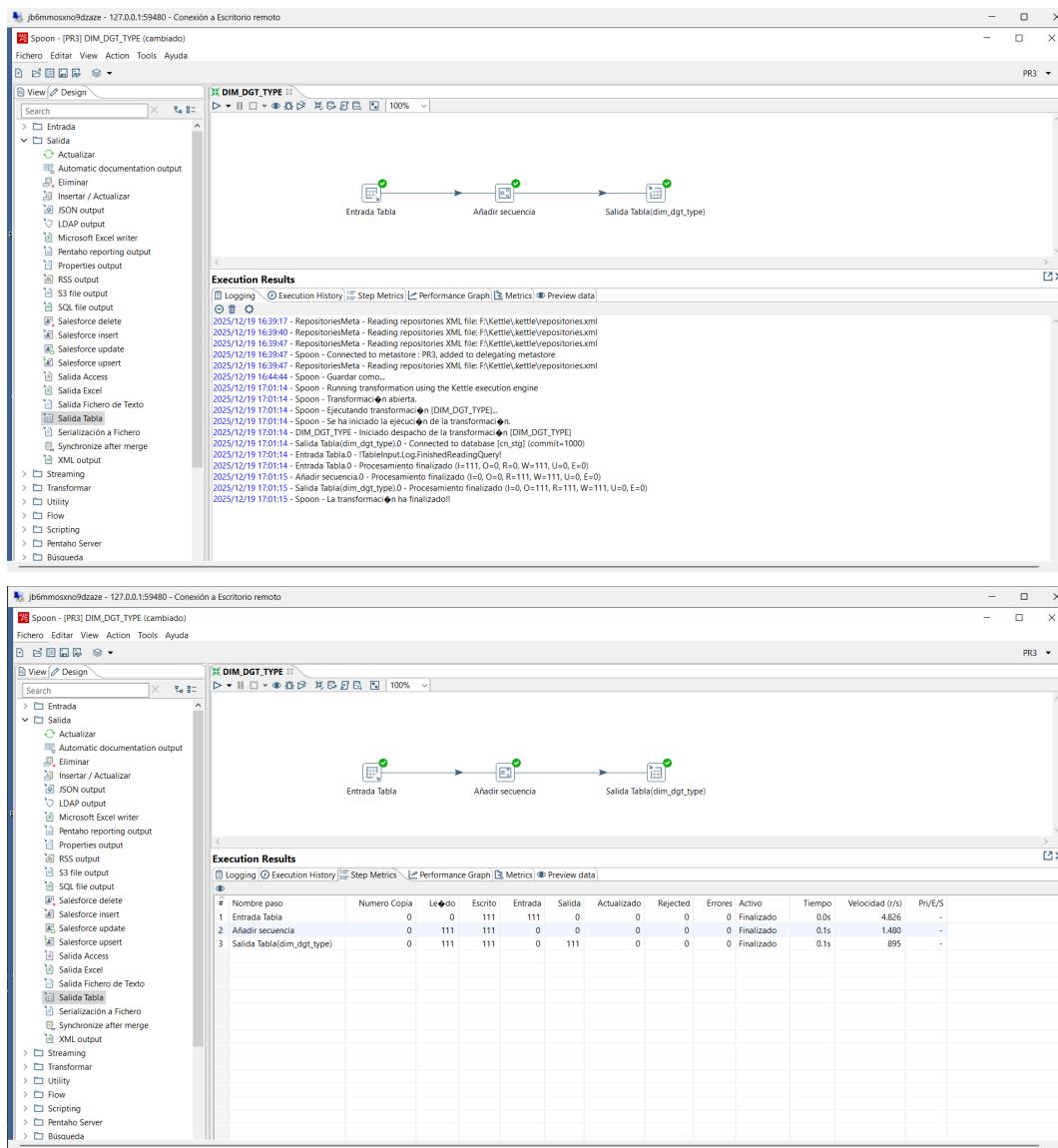
Finalmente configuré un **Table output** hacia `dbo.dim_dgt_type`, especificando el mapeo de campos y manteniendo **Truncate table** desactivado.



Evidencia (Figura D4): configuración del paso *Table output* con el mapeo visible.

3.1.4 Ejecución y métricas

Ejecuté la transformación y verifiqué que no existían errores y que la carga se realizó correctamente.



Evidencia (Figuras D5.1 y D5.2): métricas/Step Metrics de la ejecución mostrando 0 errores y filas insertadas.

3.1.5 Validación en PostgreSQL

Para validar la carga en base de datos comprobé:

- Volumetría total en `dbo.dim_dgt_type`.
- Correspondencia con el `DISTINCT` aplicado en staging con la misma lógica de normalización.
- Una muestra de registros ordenados por PK.

```

-- D6_1: volumetria cargada
SELECT COUNT(*) AS n_dim
FROM dbo.dim_dgt_type;

```

Total rows: 1 of 1 Query complete 00:00:00.129

Ln 3, Col 2:

Evidencia (Figura D6.1): COUNT(*) en dbo.dim_dgt_type.

```

-- D6_2: comprobar que coincide con el distinct de staging (misma lógica)
SELECT COUNT(*) AS n_stg_distinct
FROM (
    SELECT DISTINCT
        cod_type_dgt,
        des_type_dgt,
        UPPER(TRIM(cod_subtype_dgt)) AS cod_subtype_dgt,
        des_subtype_dgt
    FROM dbo.stg_dgt_type
) x;

```

Total rows: 1 of 1 Query complete 00:00:00.207

Ln 10, Col 1:

Evidencia (Figura D6.2): conteo del DISTINCT en staging con la misma lógica.

```

-- D6.3: muestra de datos
SELECT *
FROM dbo.dim_dgt_type
ORDER BY id_dgt_type
LIMIT 15;

```

| | id_dgt_type | cod_type_dgt | des_type_dgt | cod_subtype_dgt | des_subtype_dgt |
|----|-------------|--------------|------------------------|-----------------|--------------------------------|
| 1 | 1 | 2 | FURGONETAS | 23 | COCHE FUNEBRE |
| 2 | 2 | 1 | CAMIONES | 04 | CAMION BOTELLERO |
| 3 | 3 | 8 | TRACTORES INDUSTRIALES | 81 | TRACTOCAMIÓN |
| 4 | 4 | 2 | FURGONETAS | 22 | AMBULANCIA |
| 5 | 5 | 3 | AUTOBUSES | 33 | BIBILIBUS |
| 6 | 6 | 9 | R Y S | SF | SEMIRREMOLQUE CONTRA INCENDIOS |
| 7 | 7 | 1 | CAMIONES | 00 | CAMION BASURERO |
| 8 | 8 | 0 | OTROS vehiculos | 73 | CARRETILLA ELEVADORA |
| 9 | 9 | 5 | MOTOCICLETAS | 53 | AUTOMÓVIL DE 3 RUEDAS |
| 10 | 10 | 1 | CAMIONES | 0F | CAMÓN SILO |
| 11 | 11 | 0 | OTROS vehiculos | 71 | PALA CARGADORA |
| 12 | 12 | 1 | CAMIONES | 17 | CAMÓN ARTICulado FRIGORÍFICO |
| 13 | 13 | 0 | OTROS vehiculos | 75 | COMPACTADORA |
| 14 | 14 | 2 | FURGONETAS | 25 | TODO TERRENO |
| 15 | 15 | 1 | CAMIONES | 1D | CAMÓN ARTICulado VOLQUETE |

Total rows: 15 of 15 Query complete 00:00:00.144 ✓ Successfully run. Total query runtime: 144 msec. 15 rows affected. Ln 5, Col 1

Evidencia (Figura D6.3): muestra de datos (ORDER BY id_dgt_type LIMIT 15).

3.2 Transformación DIM_DATE

En esta transformación he cargado la dimensión temporal **DIM_DATE** mediante un único paso **Execute SQL script**, generando el calendario desde una fecha mínima hasta la fecha de ejecución, tal como plantea la guía.

3.2.1 Identificación del rango temporal en staging

Antes de generar la dimensión, obtuve el rango de fechas disponible en **dbo.stg_vehicles** (mínimo y máximo) para verificar coherencia con el periodo cubierto por la dimensión.

```

SELECT
    MIN(registration_date) AS min_date,
    MAX(registration_date) AS max_date
FROM dbo.stg_vehicles;

```

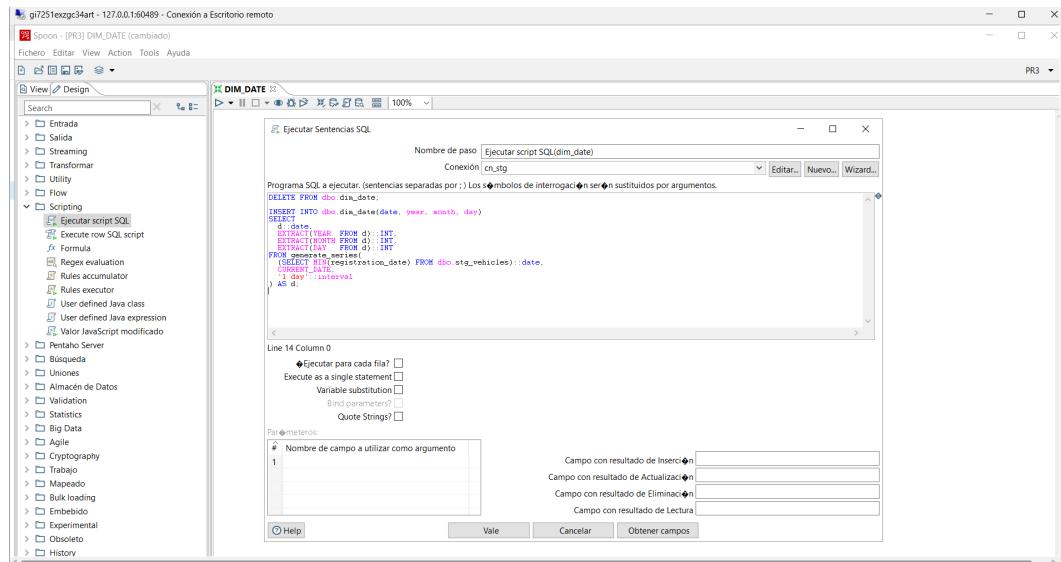
| | min_date | max_date |
|---|------------|------------|
| 1 | 2020-01-01 | 2024-12-11 |

Total rows: 1 of 1 Query complete 00:00:00.309 ✓ Successfully run. Total query runtime: 309 msec. 1 rows affected. Ln 5, Col 1

Evidencia (Figura D7.1): consulta `MIN(registration_date)` y `MAX(registration_date)` sobre `dbo.stg_vehicles`.

3.2.2 Carga con Execute SQL script

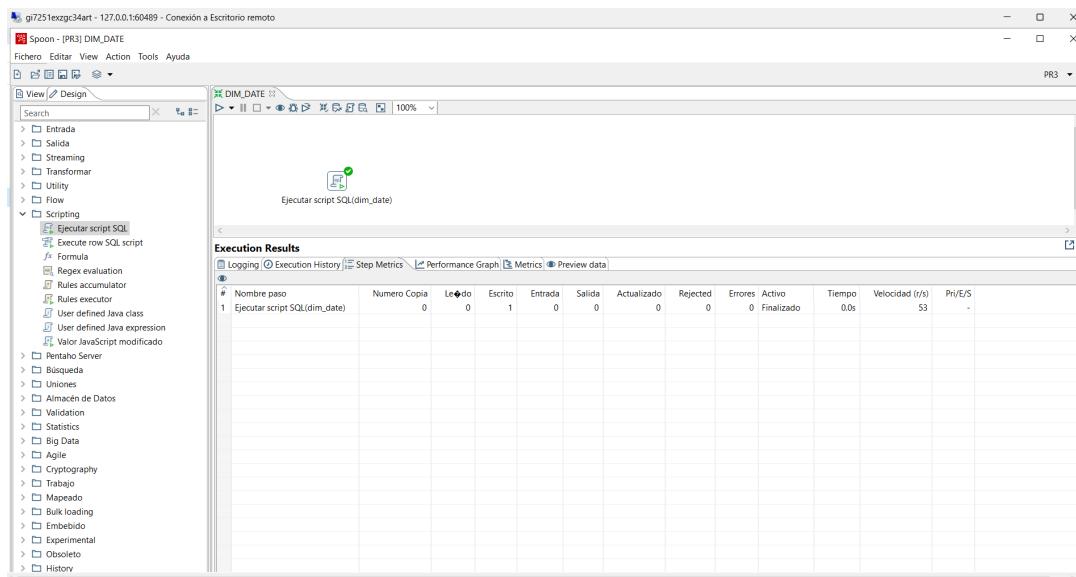
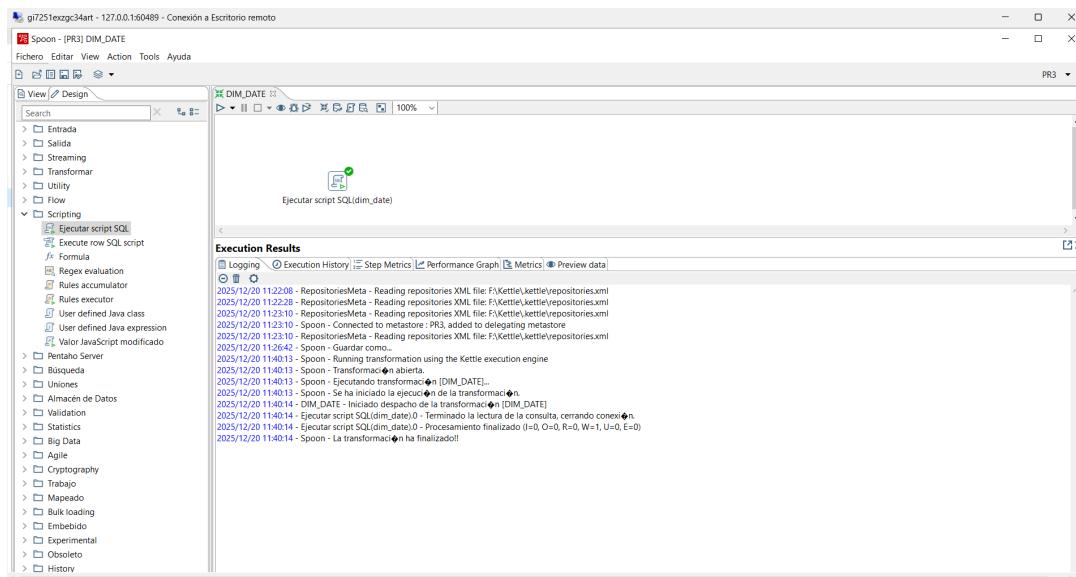
A continuación implementé la transformación usando el componente **Execute SQL script** para insertar los registros diarios en `dbo.dim_date` y asegurar que la dimensión queda actualizada para análisis por año/mes/día.



Evidencia (Figura D7.2): configuración del paso *Execute SQL script* con el script completo visible.

3.2.3 Ejecución y métricas

Ejecuté la transformación y confirmé que finalizaba correctamente, sin errores.



Evidencia (Figuras D7.3.1 y D7.3.2): métricas/log de ejecución correcta (0 errores).

3.2.4 Validación en PostgreSQL

Validé la dimensión comprobando:

- Volumetría total en `dbo.dim_date`.
- Rango mínimo y máximo en la dimensión.
- Muestra de fechas para verificar estructura y orden temporal.

```

1 SELECT COUNT(*) AS n_dim_date
2 FROM dbo.dim_date;
3

```

| n_dim_date |
|------------|
| 2181 |

Successfully run. Total query runtime: 230 msec. 1 rows affected.

Evidencia (Figura D7.4): COUNT(*) en dbo.dim_date.

```

1 SELECT MIN(date) AS min_date,
2          MAX(date) AS max_date
3 FROM dbo.dim_date;
4

```

| min_date | max_date |
|------------|------------|
| date | date |
| 2020-01-01 | 2025-12-20 |

Successfully run. Total query runtime: 98 msec. 1 rows affected.

Evidencia (Figura D7.5): MIN(date) y MAX(date) en dbo.dim_date.

The screenshot shows a PostgreSQL client interface with the following details:

- Object Explorer:** Shows databases: SOURCE_ejmenezcorra, SOURCE_emunozno, SOURCE_elpalomosa, SOURCE_eronquillo, SOURCE_ssamperp, SOURCE_stomasm, and SOURCE_suave.
- Query Editor:** A tab titled "SOURCE_suave/STUDENT_suave@PostgreSQL_PR1" contains the following SQL code:


```

1 SELECT *
2 FROM dbo.dim_date
3 ORDER BY date
4 LIMIT 12;
      
```
- Data Output:** A table titled "dbo.dim_date" displays 12 rows of data:

| date | [PK] date | year | month | day |
|------|------------|------|-------|-----|
| 1 | 2020-01-01 | 2020 | 1 | 1 |
| 2 | 2020-01-02 | 2020 | 1 | 2 |
| 3 | 2020-01-03 | 2020 | 1 | 3 |
| 4 | 2020-01-04 | 2020 | 1 | 4 |
| 5 | 2020-01-05 | 2020 | 1 | 5 |
| 6 | 2020-01-06 | 2020 | 1 | 6 |
| 7 | 2020-01-07 | 2020 | 1 | 7 |
| 8 | 2020-01-08 | 2020 | 1 | 8 |
| 9 | 2020-01-09 | 2020 | 1 | 9 |
| 10 | 2020-01-10 | 2020 | 1 | 10 |
| 11 | 2020-01-11 | 2020 | 1 | 11 |
| 12 | 2020-01-12 | 2020 | 1 | 12 |
- Messages:** A green message at the bottom right says "Successfully run. Total query runtime: 450 msec. 12 rows affected."

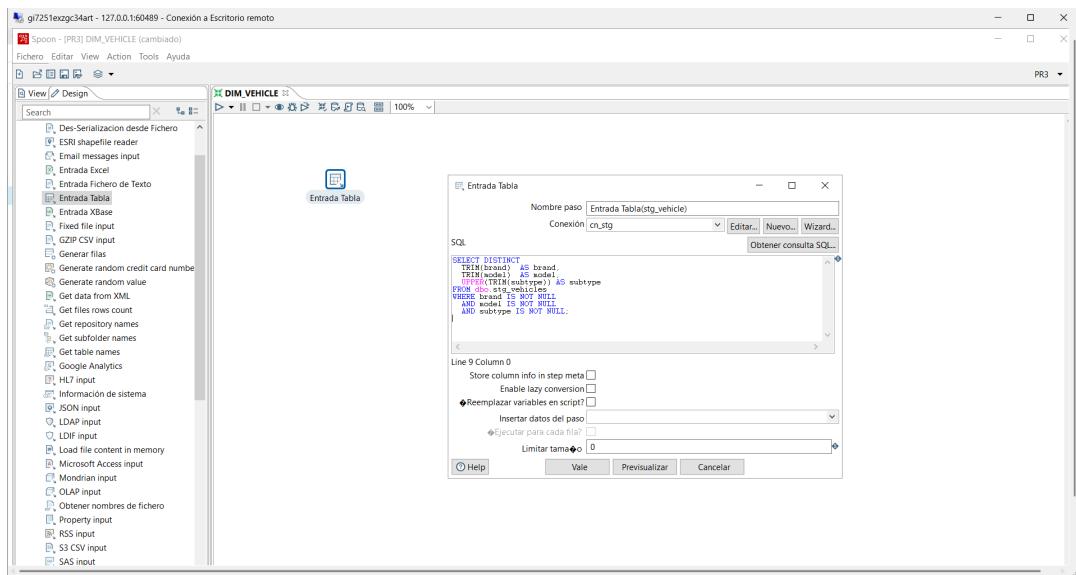
Evidencia (Figura D7.6): muestra (ORDER BY date LIMIT 12).

3.3 Transformación DIM_VEHICLE

En esta transformación he cargado la dimensión **DIM_VEHICLE** desde **dbo.stg_vehicles**, manteniendo únicamente los atributos dimensionales del vehículo (brand, model, subtype) y generando PK sustituta con **Add sequence**, siguiendo el patrón estándar de dimensiones.

3.3.1 Lectura desde stg_vehicles (Table input)

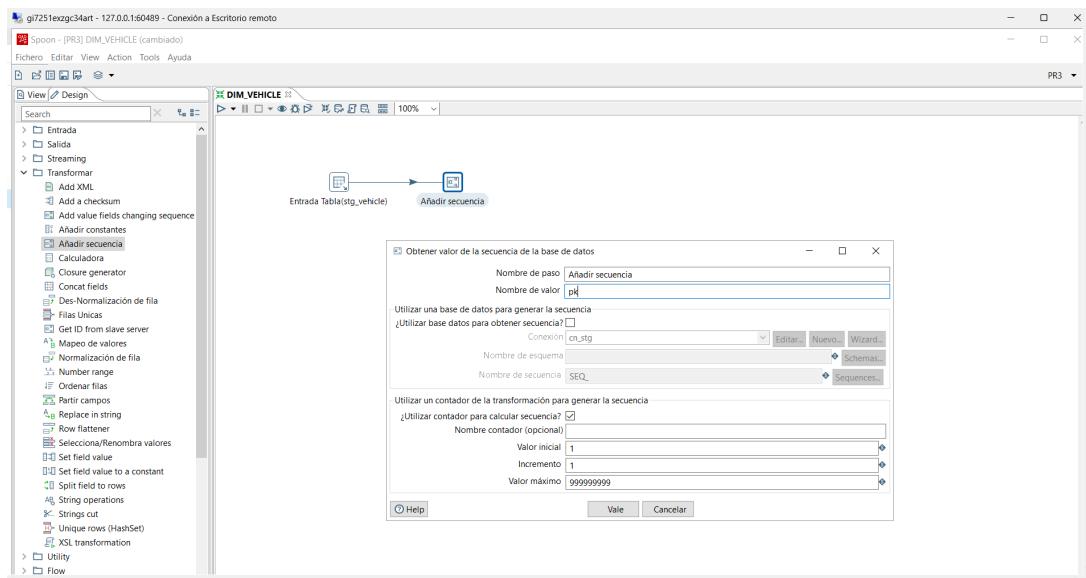
Configuré un **Table input** para extraer los datos necesarios desde **dbo.stg_vehicles** (y/o su **DISTINCT** correspondiente), aplicando las normalizaciones necesarias para garantizar consistencia posterior en los joins de la FACT.



Evidencia (Figura D8.1): configuración del paso *Table input* (SQL visible).

3.3.2 Añadir secuencia (Add sequence)

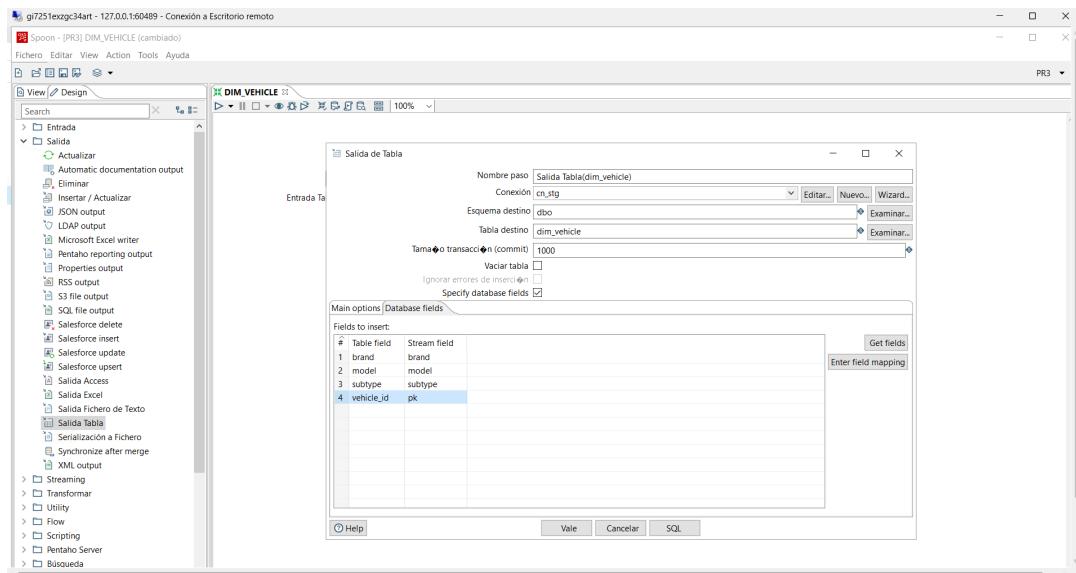
Generé la clave sustituta `vehicle_id` con **Add sequence**.



Evidencia (Figura D8.2): configuración del paso *Add sequence*.

3.3.3 Inserción en dim_vehicle (Table output)

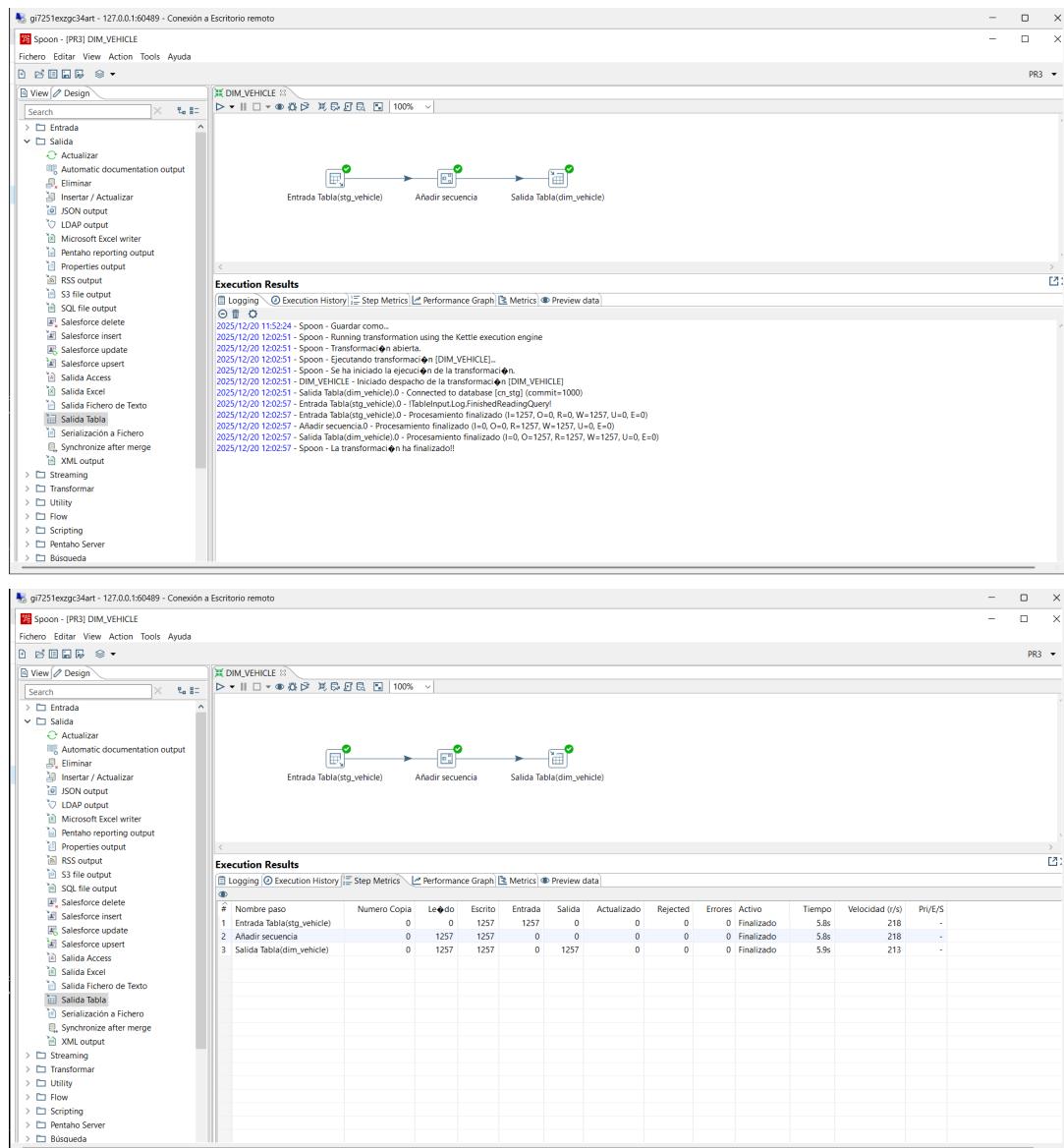
Configuré **Table output** hacia `dbo.dim_vehicle` con el mapeo correspondiente. Tal como indica la guía, en dimensiones **no activé Truncate table**, ya que el vaciado se gestiona en el job con control de integridad.



Evidencia (Figura D8.3): mapeo y configuración del *Table output*.

3.3.4 Ejecución y métricas

Ejecuté la transformación y comprobé ausencia de errores y filas insertadas.



Evidencia (Figuras D8.4.1 y D8.4.2): Step Metrics/metrics de ejecución (0 errores).

3.3.5 Validación en PostgreSQL

Validé:

- Volumetría total de `dbo.dim_vehicle`.
- Comparación con el `DISTINCT` en staging usando la misma lógica.
- Muestra de datos para revisión rápida.

```

g:\7251exzg34art - 127.0.0.1:60489 - Conexión a Escritorio remoto
Query History
1 SELECT COUNT(*) AS n_dim_vehicle
2 FROM dbo.dim_vehicle;
3

Data Output Messages Notifications
n_dim_vehicle
1 1257

Total rows: 1 of 1 Query complete 0:00:00.280

```

Evidencia (Figura D8.5): COUNT(*) en dbo.dim_vehicle.

```

g:\7251exzg34art - 127.0.0.1:60489 - Conexión a Escritorio remoto
Query History
1 SELECT COUNT(*) AS n_stg_distinct
2 FROM (
3     SELECT DISTINCT
4         TRIM(brand) AS brand,
5         TRIM(model) AS model,
6         UPPER(TRIM(subtype)) AS subtype
7     FROM dbo.stg_vehicles
8     WHERE brand IS NOT NULL
9     AND model IS NOT NULL
10    AND subtype IS NOT NULL
11 ) x;
12

Data Output Messages Notifications
n_stg_distinct
1 1257

Total rows: 1 of 1 Query complete 0:00:01.923

```

Evidencia (Figura D8.6): conteo DISTINCT equivalente en staging.

```

1 SELECT *
2 FROM dbo.dim_vehicle
3 ORDER BY vehicle_id
4 LIMIT 15;
5

```

| vehicle_id | brand | model | subtype |
|------------|---------------------|----------------|---------|
| 1 | ADRIA | ADORA 573 PT | RA |
| 2 | ADRIA | AVIVA 472 PK | RA |
| 3 | ADRIA | AVIVA 495 LX | RA |
| 4 | ADRIA | AVIVA 522 PT | RA |
| 5 | AGRIMAC | TW25-4 | 73 |
| 6 | AIXAM | S9 | 90 |
| 7 | AIXAM | S9 | 92 |
| 8 | ALFA ROMEO | ALFA GIULIETTA | 40 |
| 9 | ALFA ROMEO | GUILIA | 40 |
| 10 | ALFA ROMEO | STELVIO | 40 |
| 11 | ANGEL BRUGUERA FONT | AROCES 2546 L | 00 |
| 12 | APRILIA | RS 660 | 50 |
| 13 | APRILIA | SR 50 | 90 |
| 14 | APRILIA | SR GT 125 | 50 |
| 15 | AUDI | A1 | 40 |

Total rows: 15 of 15 Query complete 00:00:00.305 Ln 5, Col 1

Evidencia (Figura D8.7): muestra (LIMIT 15).

3.4 Transformación FACT_VEHICLE_REGISTRATION

En este apartado he cargado la tabla de hechos **FACT_VEHICLE_REGISTRATION** desde **dbo.stg_vehicles**, resolviendo previamente las claves foráneas hacia las dimensiones (date, vehicle, dgt_type) y generando un identificador **id_registration** mediante **Add sequence**. Además, la medida **quantity** se fija a **1** por registro.

3.4.1 Creación/estructura de la tabla fact_vehicle_registration en PostgreSQL

Antes de la carga, ejecuté el script de creación de **dbo.fact_vehicle_registration**.

```

1 DROP TABLE IF EXISTS dbo.fact_vehicle_registration;
2
3 CREATE TABLE dbo.fact_vehicle_registration (
4     id_registration integer NOT NULL,
5     registration_date date NOT NULL,
6     vehicle_id integer NOT NULL,
7     id_dgt_type integer NOT NULL,
8     municipalities_id bigint NOT NULL,
9     new_used boolean,
10    quantity integer NOT NULL DEFAULT 1,
11
12    CONSTRAINT pk_fact_vehicle_registration
13        PRIMARY KEY (id_registration),
14
15    CONSTRAINT fk_fact_date
16        FOREIGN KEY (registration_date) REFERENCES dbo.dim_date(date),
17
18    CONSTRAINT fk_fact_vehicle
19        FOREIGN KEY (vehicle_id) REFERENCES dbo.dim_vehicle(vehicle_id),
20
21    CONSTRAINT fk_fact_dgt_type
22        FOREIGN KEY (id_dgt_type) REFERENCES dbo.dim_dgt_type(id_dgt_type)
23)
24
25 CREATE TABLE

```

Query returned successfully in 81 msec.

Total rows: 1000 of 1721704 Query complete 00:00:00.081 Ln 27, Col 1

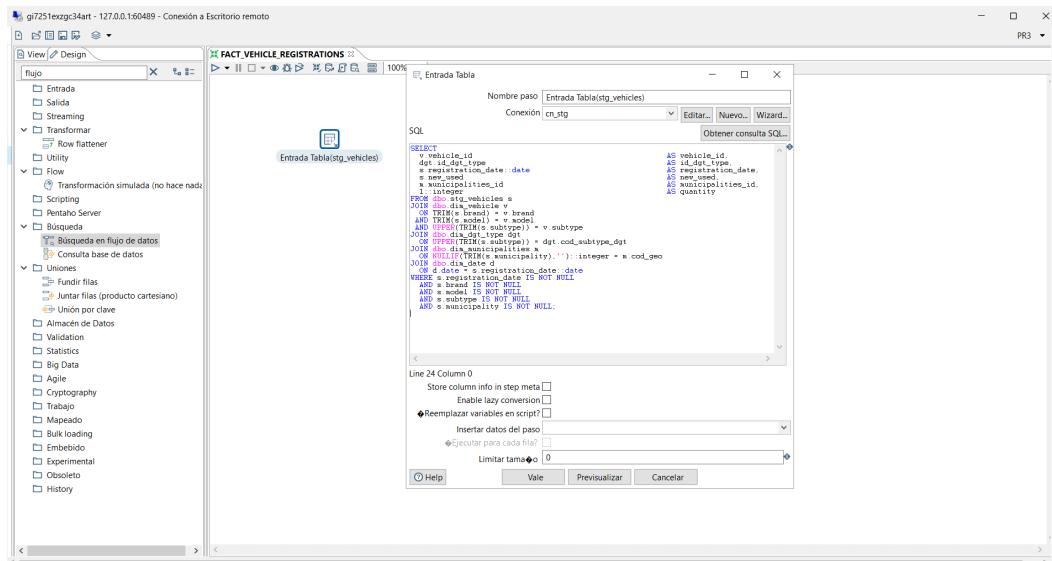
Evidencia (Figura D10.3): script ejecutado en PostgreSQL para la estructura de la FACT.

3.4.2 Lectura y resolución de FKs desde stg_vehicles (Table input con JOIN)

En el **Table input** preparé la extracción desde `dbo.stg_vehicles` resolviendo:

- `vehicle_id` vía join con `dbo.dim_vehicle`
- `id_dgt_type` vía join con `dbo.dim_dgt_type`
- `registration_date` validada contra `dbo.dim_date`
- y, tras la ampliación del modelo, `municipalities_id` vía join con `dbo.dim_municipalities`

Además incluí `quantity = 1`.

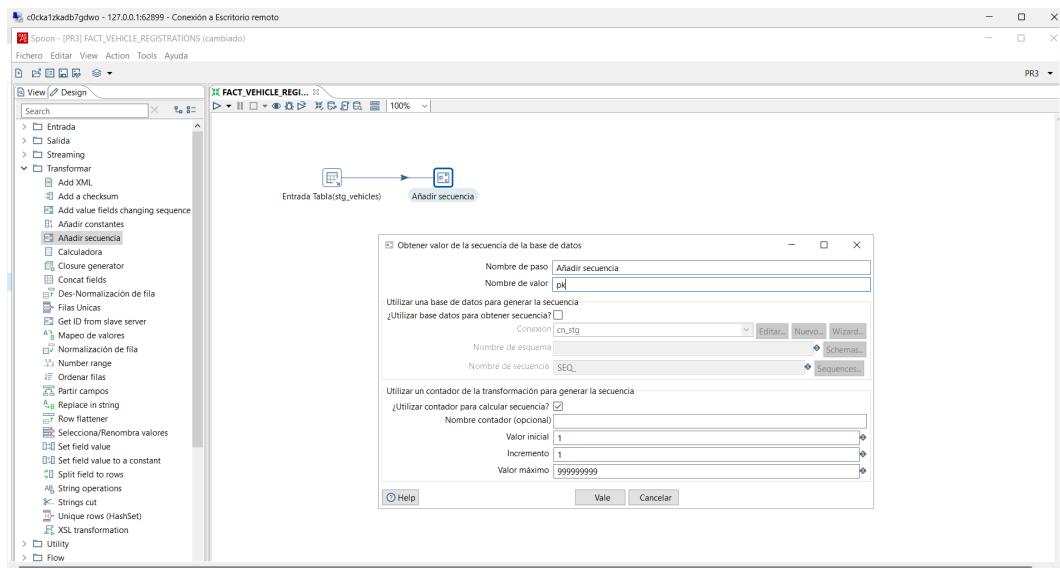


```
SELECT
    vehicle_id,
    dgt_id_dgt_type,
    registration_date,
    new_used,
    municipalities_id
    1 integer
FROM
    dim_vehicles v
JOIN
    dim_vehicle v
    ON TRIM(v.brand) = v.brand
    AND TRIM(v.model) = v.model
    AND TRIM(v.subtype) = v.subtype
JOIN
    dbo.dim_dgt_type dgt
    ON dgt.cod_dgt_type = v.id_dgt_type
JOIN
    dbo.dim_municipalities m
    ON m.cod_municipality = v.municipalities_id
JOIN
    dbo.dim_date d
    ON d.date = v.registration_date
WHERE
    v.registration_date IS NOT NULL
    AND v.brand IS NOT NULL
    AND v.model IS NOT NULL
    AND v.subtype IS NOT NULL
    AND v.municipality IS NOT NULL;
```

Evidencia (Figura D10.4): SQL del *Table input* con los JOINs y campos finales.

3.4.3 Añadir secuencia (Add sequence) para id_registration

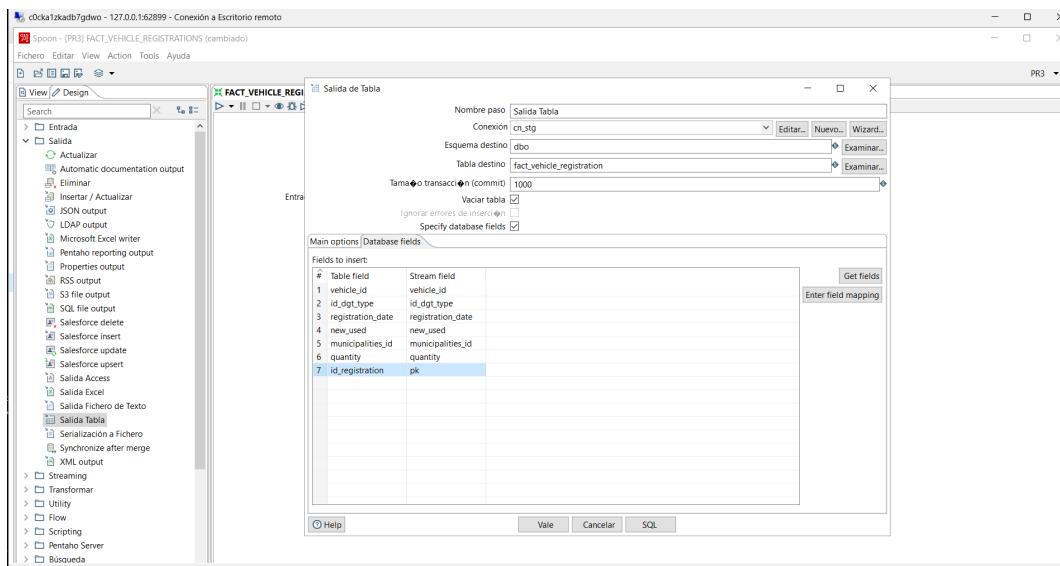
Apliqué **Add sequence** para generar `id_registration` como PK sustituta de la tabla de hechos.



Evidencia (Figura D10.5): configuración del *Add sequence*.

3.4.4 Inserción en fact_vehicle_registration (Table output) con Truncate

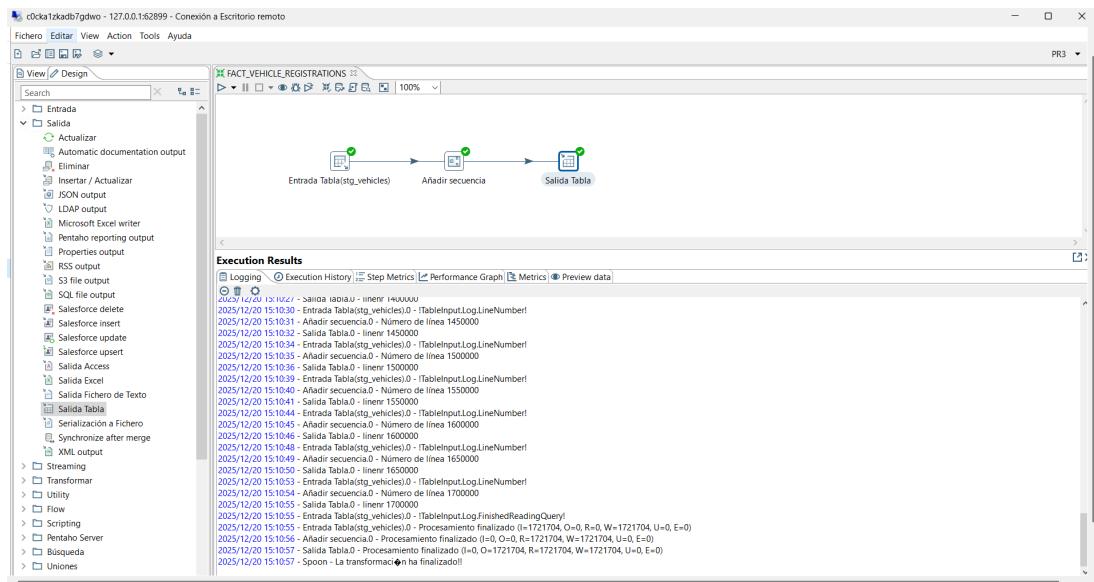
Configuré **Table output** hacia `dbo.fact_vehicle_registration` activando **Truncate table**, ya que se trata de una recarga completa del hecho, y definí el mapeo de campos conforme a la estructura final.



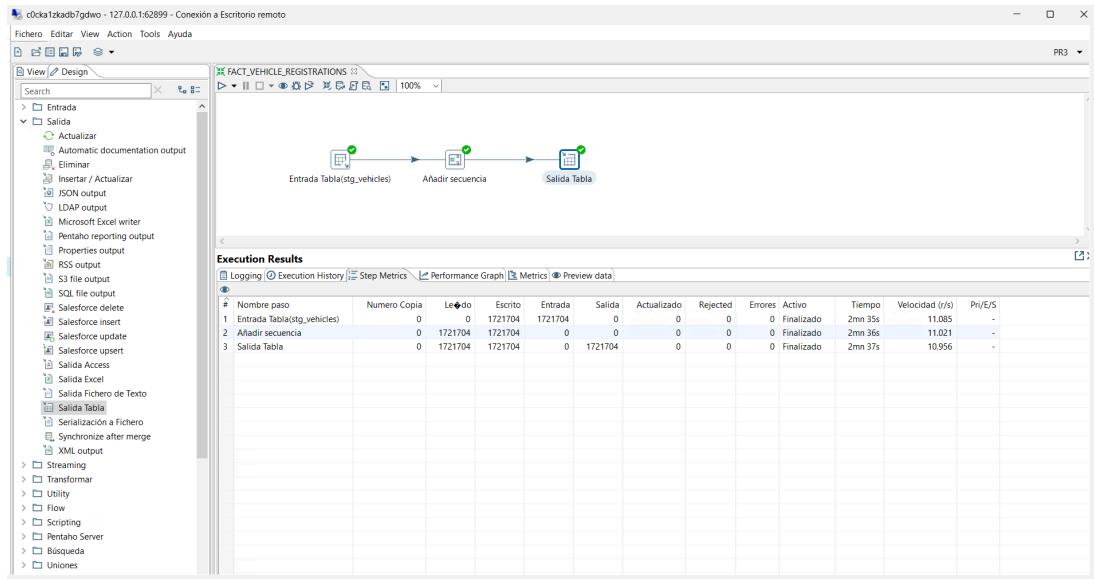
Evidencia (Figura D10.6): configuración del *Table output* con mapeo visible.

3.4.5 Ejecución y métricas

Ejecuté la transformación y comprobé que finalizaba sin errores y con filas insertadas.



Evidencia (Figura D10.7.1): logging final de ejecución correcta.



Evidencia (Figura D10.7.2): Step Metrics con 0 errores y filas escritas.

3.4.6 Validación en PostgreSQL

Validé:

- Conteo total de registros insertados.
- Chequeo de nulos en claves foráneas (esperado 0).
- Muestra rápida para confirmar campos clave informados.

The screenshot shows the SSMS interface with a connection to 'cicka1zadb7gdwo - 127.0.0.1:62899'. The left pane displays the Object Explorer with several databases listed under 'SOURCE_suave'. The right pane shows a query window with the following code:

```

1 SELECT COUNT(*)::bigint AS n_fact_rows
2 FROM dbo.fact_vehicle_registration;

```

The results pane shows a single row with the value 1721704. A status bar at the bottom indicates 'Successfully run. Total query runtime: 283 msec. 1 rows affected.'

Evidencia (Figura D10.8.1): COUNT(*) total en la FACT.

The screenshot shows the SSMS interface with the same connection. The left pane shows the Object Explorer with 'SOURCE_suave' selected. The right pane shows a query window with the following code:

```

1 SELECT COUNT(*)::bigint AS n_fk_nulls
2 FROM dbo.fact_vehicle_registration
3 WHERE registration_date IS NULL
4 OR vehicle_id IS NULL
5 OR id_dgt_type IS NULL
6 OR municipalities_id IS NULL;

```

The results pane shows a single row with the value 0. A status bar at the bottom indicates 'Successfully run. Total query runtime: 239 msec. 1 rows affected.'

Evidencia (Figura D10.8.2): chequeo de FKs nulos (resultado esperado 0).

```

1 SELECT *
2 FROM dbo.fact_vehicle_registration
3 ORDER BY id_registration
4 LIMIT 15;

```

| | id_registration | registration_date | vehicle_id | id_dgt_type | municipalities_id | new_used | quantity |
|----|-----------------|-------------------|------------|-------------|-------------------|----------|----------|
| 1 | 1 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 2 | 2 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 3 | 3 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 4 | 4 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 5 | 5 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 6 | 6 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 7 | 7 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 8 | 8 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 9 | 9 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 10 | 10 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 11 | 11 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 12 | 12 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 13 | 13 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 14 | 14 | 2020-03-01 | 346 | 86 | 200 | true | 1 |
| 15 | 15 | 2020-03-01 | 346 | 86 | 200 | true | 1 |

Total rows: 15 of 15 Query complete 00:00:00.147

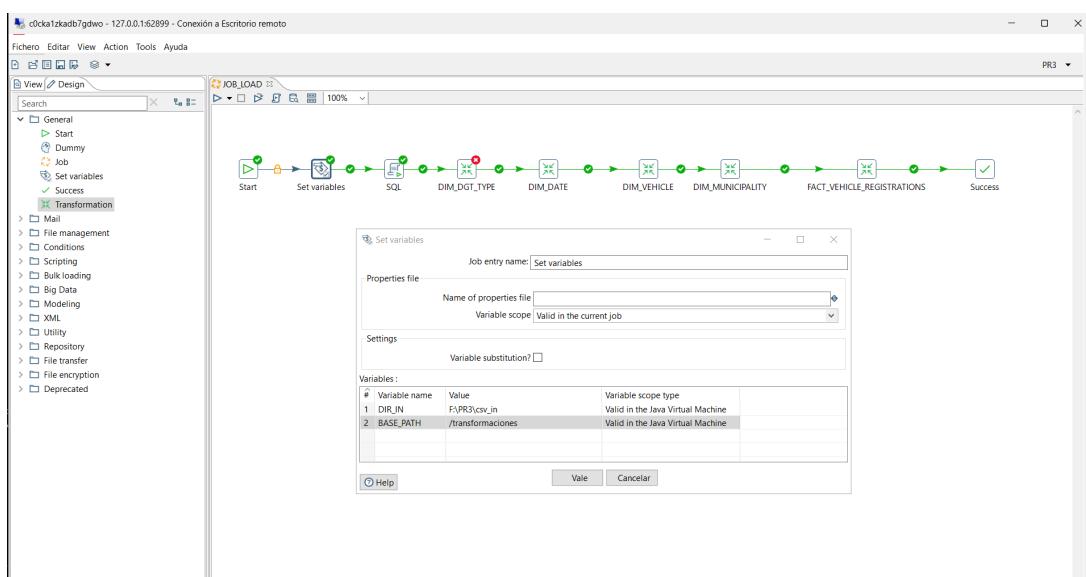
Evidencia (Figura D10.8.3): muestra de datos (verificación de campos clave).

3.5 Job JOB_LOAD

En esta sección he construido un **job** en Spoon para ejecutar de forma secuencial las transformaciones del proceso de carga, garantizando orden mediante hops “OK” (éxito) y aplicando limpieza controlada de tablas antes de recargar.

3.5.1 Parametrización de variables (Set variables)

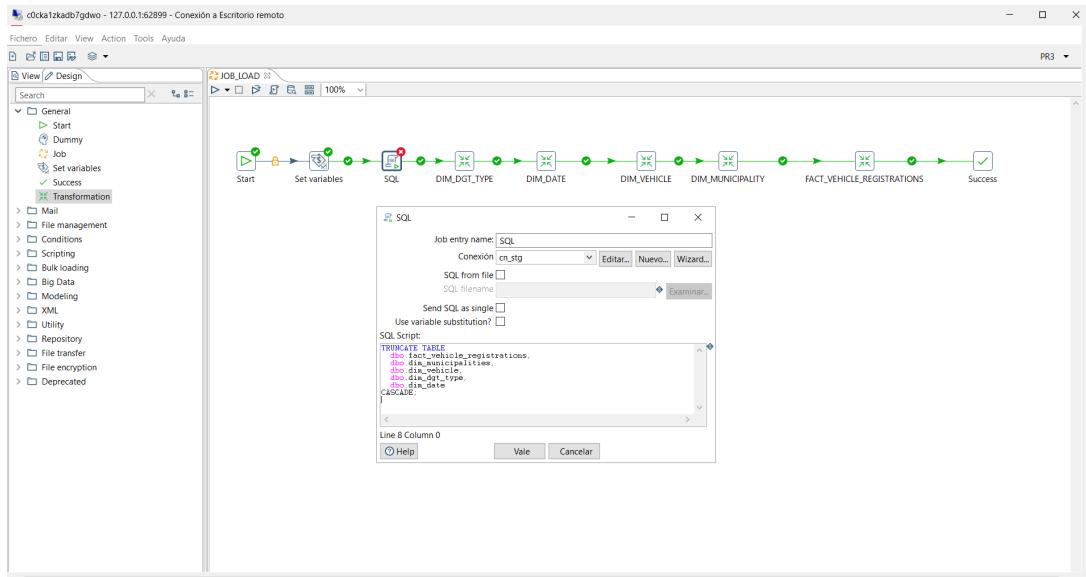
Definí variables para centralizar rutas (por ejemplo, **DIR_IN** y **BASE_PATH**) y poder referenciar los **.ktr** sin rutas hardcodeadas en cada paso.



Evidencia (Figura D11.2): configuración del paso Set variables con las rutas visibles.

3.5.2 Limpieza previa (SQL)

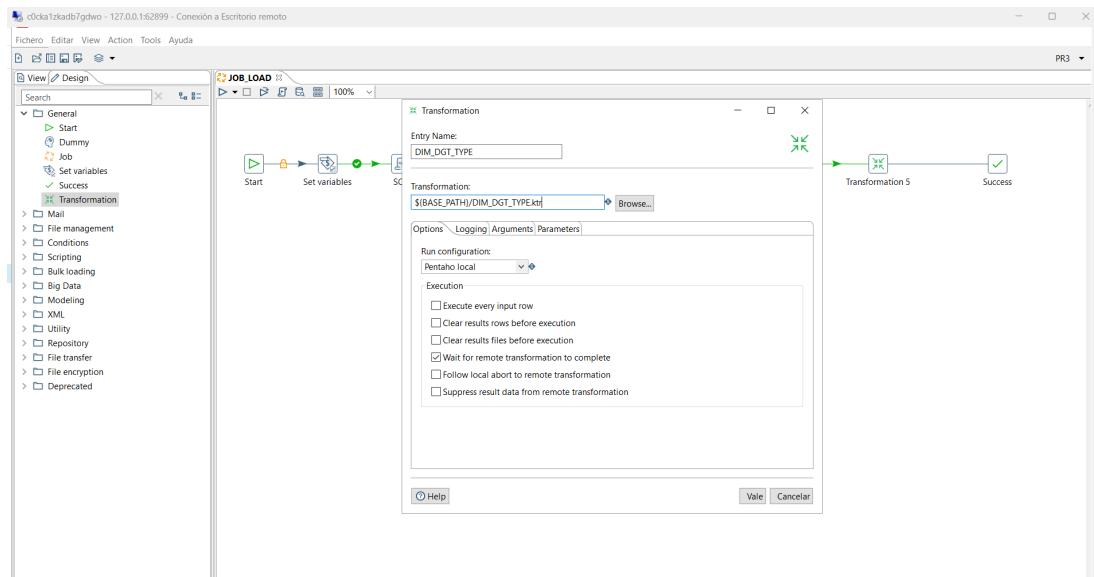
Añadí un paso **SQL** para limpiar el modelo antes de ejecutar las transformaciones. Dado que existen dependencias por claves foráneas, la limpieza se plantea en orden correcto (primero FACT y después DIMs) y con las opciones necesarias para no romper integridad.



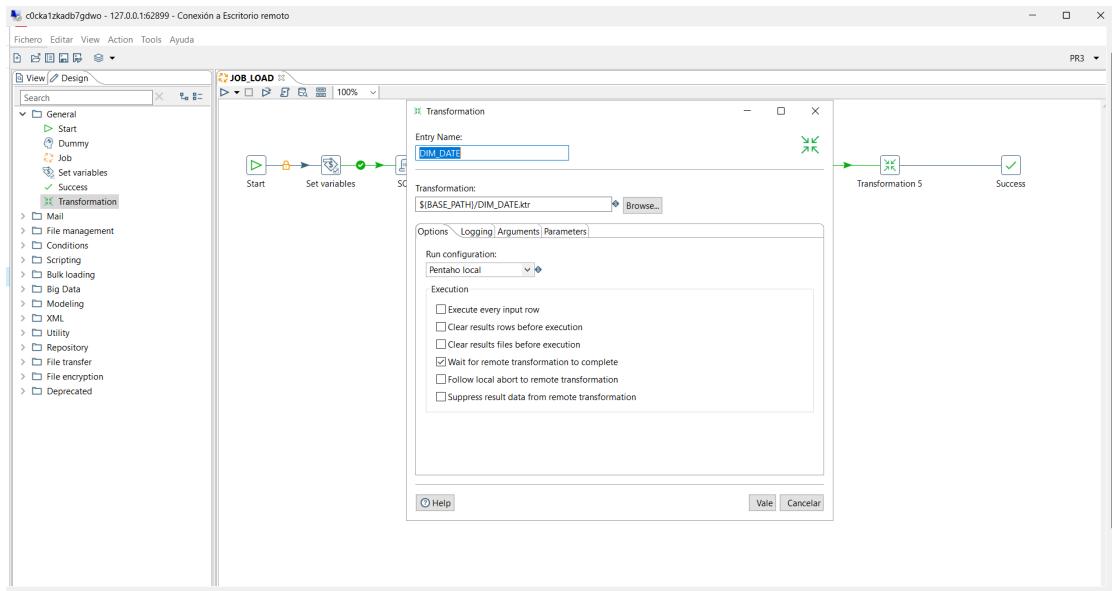
Evidencia (Figura D11.3): ventana del paso SQL con el script visible.

3.5.3 Entradas Transformation del job (llamadas a los .ktr)

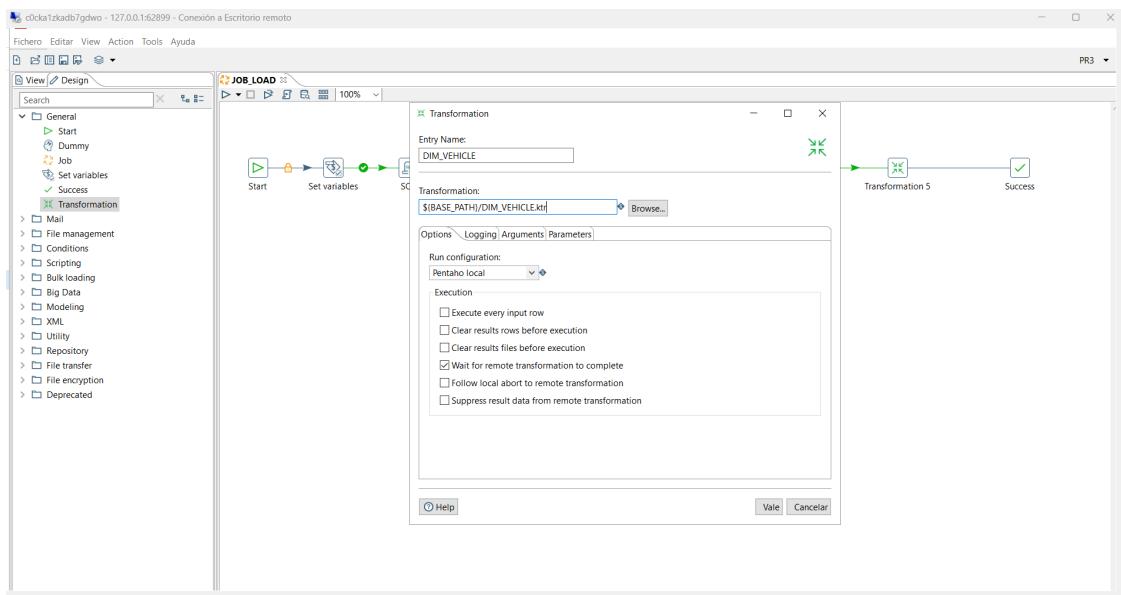
Incluí las entradas de tipo **Transformation** apuntando a los **.ktr** correspondientes, con ejecución secuencial.



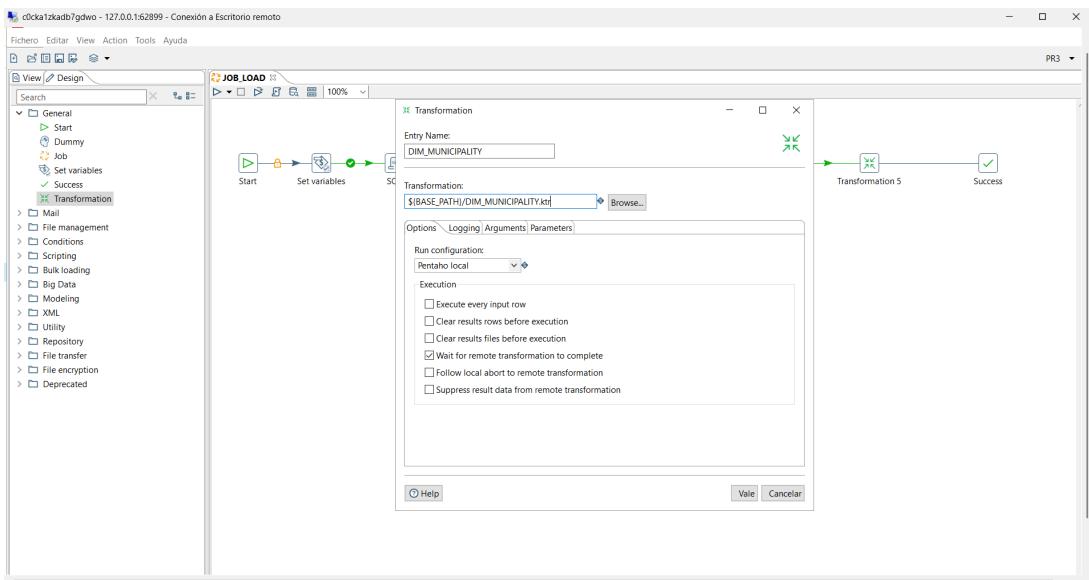
Evidencia (Figura D11.4.1): configuración entry DIM_DGT_TYPE.



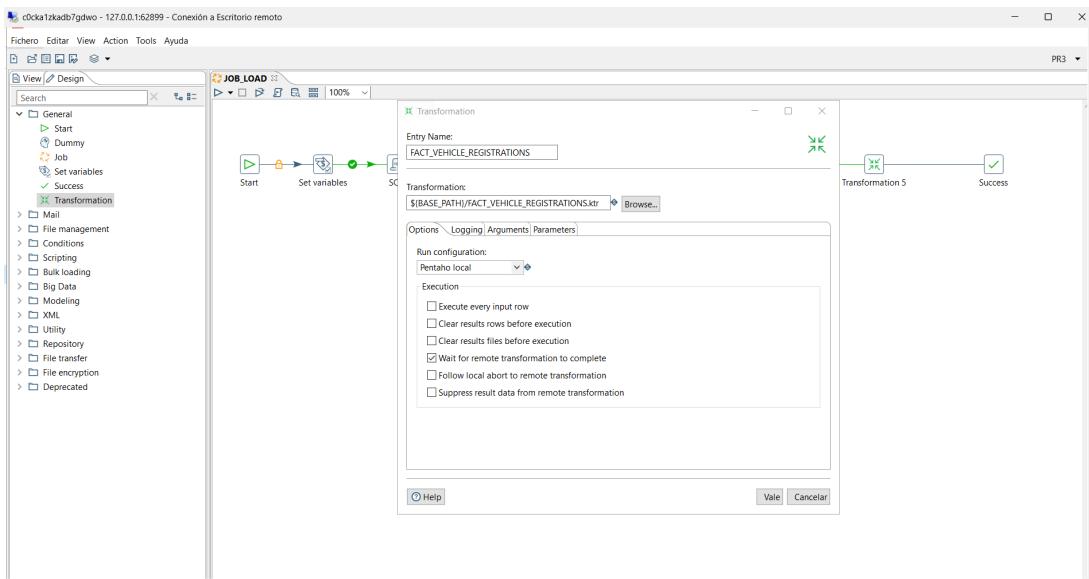
Evidencia (Figura D11.4.2): configuración entry DIM_DATE.



Evidencia (Figura D11.4.3): configuración entry DIM_VEHICLE.



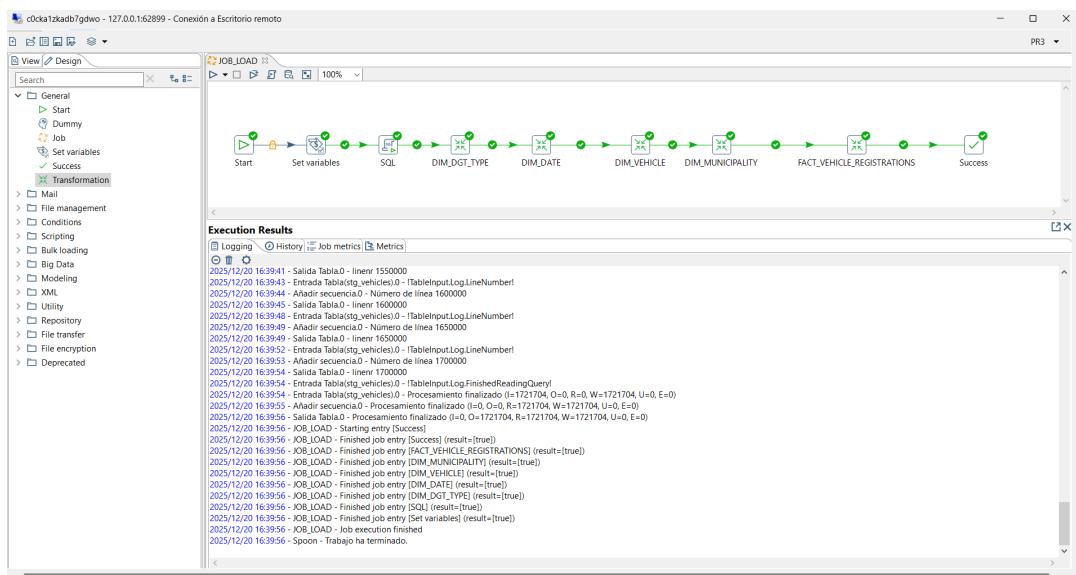
Evidencia (Figura D11.4.4): configuración entry DIM_MUNICIPALITY.



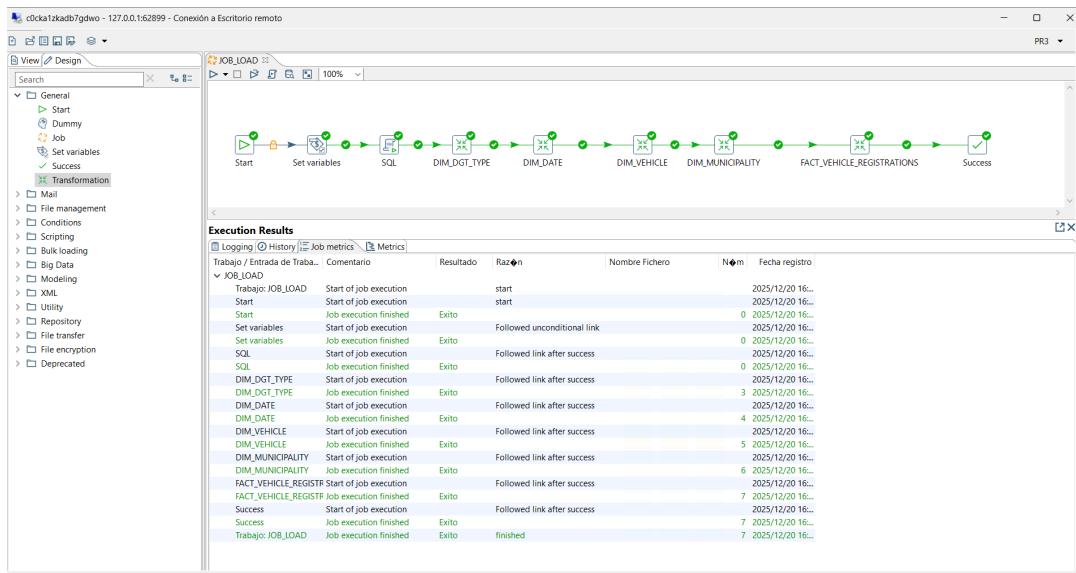
Evidencia (Figura D11.4.5): configuración entry FACT_VEHICLE_REGISTRATIONS.

3.5.4 Ejecución y métricas del job

Ejecuté el job y verifiqué en el log final y en métricas que la ejecución concluye correctamente.



Evidencia (Figura D11.6.1): logging final (“Job execution finished”).



Evidencia (Figura D11.6.2): métricas/resultados del job sin errores.

3.5.5 Validación post-job en PostgreSQL

Tras la ejecución del job verifiqué conteos y la ausencia de nulos en FKs de la FACT.

```

SELECT COUNT(*)::bigint AS n_dim_dgt_type
FROM dbo.dim_dgt_type;
SELECT COUNT(*)::bigint AS n_dim_date
FROM dbo.dim_date;
SELECT COUNT(*)::bigint AS n_dim_vehicle
FROM dbo.dim_vehicle;
SELECT COUNT(*)::bigint AS n_dim_municipalities
FROM dbo.dim_municipalities;
SELECT COUNT(*)::bigint AS n_fact_rows
FROM dbo.fact_vehicle_registration;

```

| n_fact_rows | bigint |
|-------------|---------|
| 1 | 1721704 |

Total rows: 1 of 1 Query complete 00:00:00.279

Successfully run. Total query runtime: 279 msec. 1 rows affected. Ln 6, Col 1

Evidencia (Figura D11.7.1): conteos de dimensiones y FACT tras el job.

```

SELECT COUNT(*)::bigint AS n_fk_nulls
FROM dbo.fact_vehicle_registration
WHERE registration_date IS NULL
    OR vehicle_id IS NULL
    OR id_dgt_type IS NULL
    OR municipalities_id IS NULL;

```

| n_fk_nulls | bigint |
|------------|--------|
| 1 | 0 |

Total rows: 1 of 1 Query complete 00:00:00.268

Successfully run. Total query runtime: 268 msec. 1 rows affected. Ln 6, Col 35

Evidencia (Figura D11.7.2): chequeo de FKs nulos en FACT (resultado esperado 0).

3.6 Ejercicios a realizar

En este apartado he aplicado literalmente los puntos solicitados:

3.6.1 Reproducir guía 3.1–3.3 (métricas DIM_DGT_TYPE, DIM_DATE y DIM_VEHICLE)

He incluido evidencias de configuración y métricas de ejecución correcta de las tres dimensiones:

- **DIM_DGT_TYPE**: Figuras D2, D3, D4, D5.1, D5.2, D6.1, D6.2, D6.3.
- **DIM_DATE**: Figuras D7.1, D7.2, D7.3.1, D7.3.2, D7.4, D7.5, D7.6.
- **DIM_VEHICLE**: Figuras D8.1, D8.2, D8.3, D8.4.1, D8.4.2, D8.5, D8.6, D8.7.

3.6.2 Reproducir guía 3.4 (métricas FACT_VEHICLE_REGISTRATION)

He incluido evidencias de configuración, ejecución y validación de la FACT (con la estructura ya adaptada para municipios):

- Figuras D10.3, D10.4, D10.5, D10.6, D10.7.1, D10.7.2, D10.8.1, D10.8.2, D10.8.3.

3.6.3 Reproducir guía 3.5 (métricas JOB_LOAD)

He incluido evidencias del job, su ejecución y validaciones posteriores:

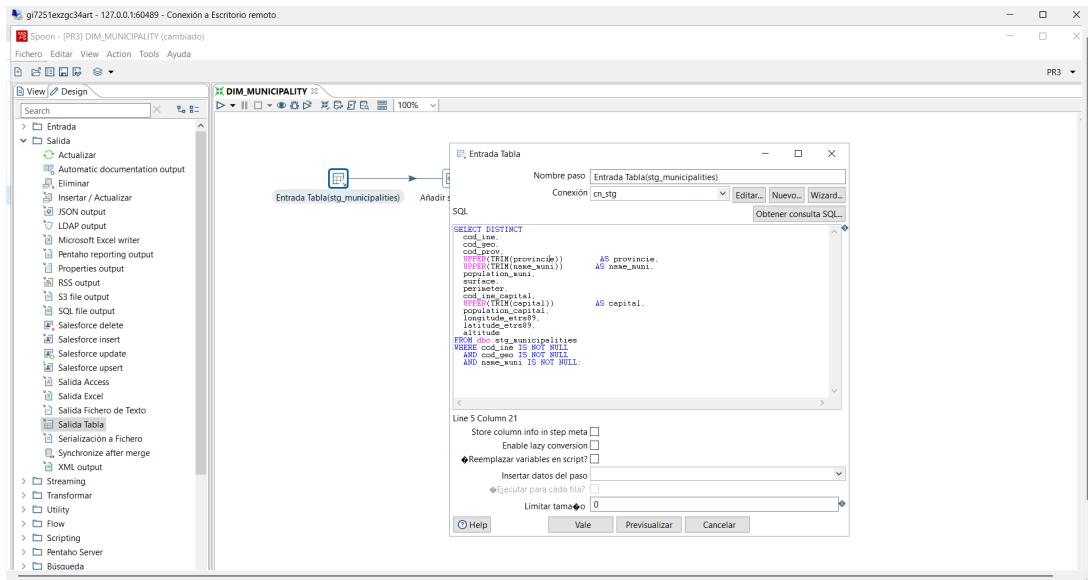
- Figuras D11.2, D11.3, D11.4.1–D11.4.5, D11.6.1, D11.6.2, D11.7.1, D11.7.2.

3.6.4 Diseñar y ejecutar DIM_MUNICIPALITY

En este apartado he diseñado y ejecutado la transformación **DIM_MUNICIPALITY** para incorporar al modelo estrella la dimensión geográfica a partir de **dbo.stg_municipalities**. El objetivo ha sido construir una dimensión limpia y consistente (normalización de textos) y con **PK sustituta** para poder vincularla posteriormente desde la tabla de hechos.

3.6.4.1 Lectura desde staging y normalización (Table input)

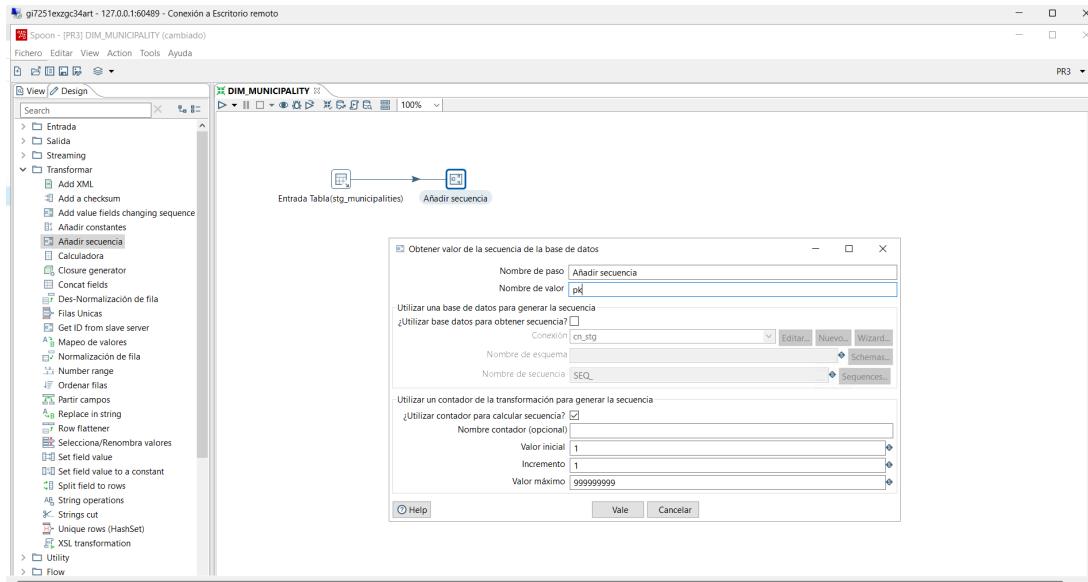
En primer lugar configuré un **Table input** leyendo desde **dbo.stg_municipalities** con **SELECT DISTINCT**, aplicando normalización sobre campos de texto mediante **UPPER(TRIM(...))** (por ejemplo, provincia, nombre de municipio y capital) y filtrando registros sin claves mínimas (por ejemplo, códigos nulos) para evitar filas dimensionales incompletas.



Evidencia (Figura D9.1): configuración del *Table input* con la SQL visible.

3.6.4.2 Generación de PK sustituta (Add sequence)

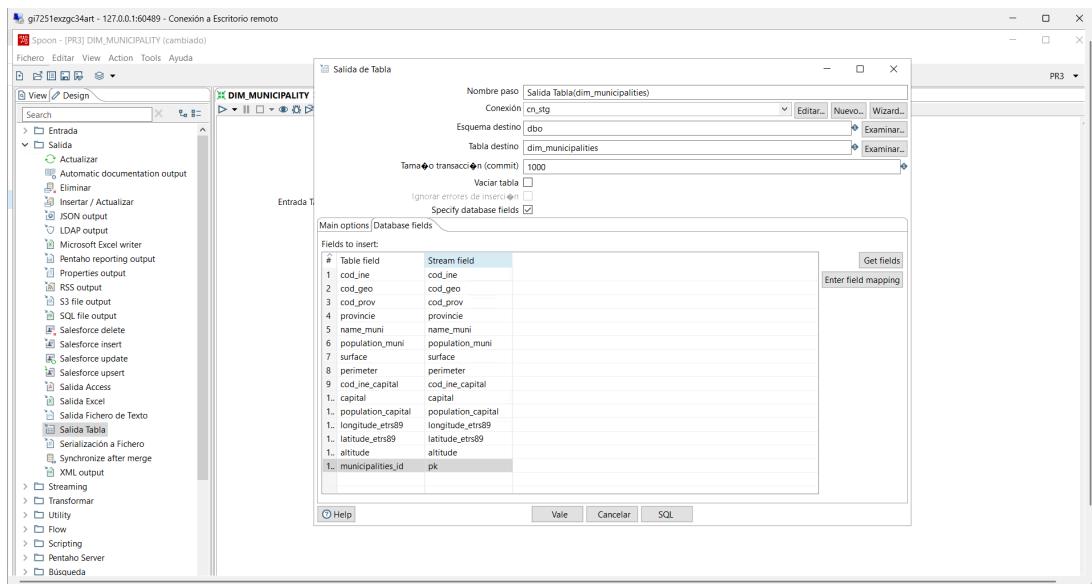
A continuación incorporé un paso **Add sequence** para generar una clave sustituta incremental (PK) que identifica únicamente cada municipio dentro de la dimensión.



Evidencia (Figura D9.2): configuración del Add sequence.

3.6.4.3 Carga en la dimensión (Table output) y mapeo

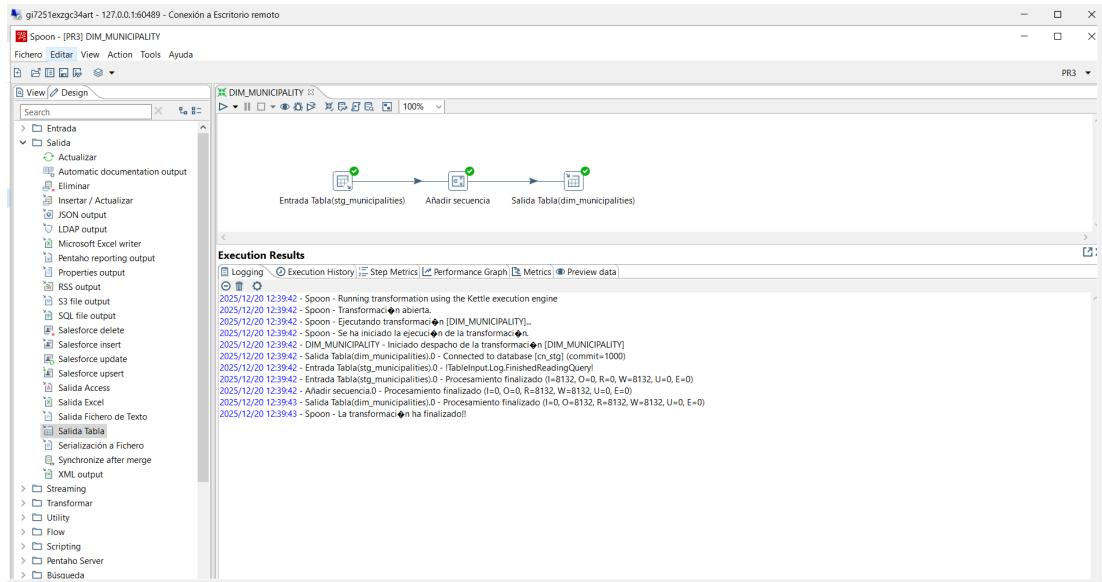
Finalmente configuré un **Table output** hacia la tabla destino de dimensión, especificando el mapeo de campos.



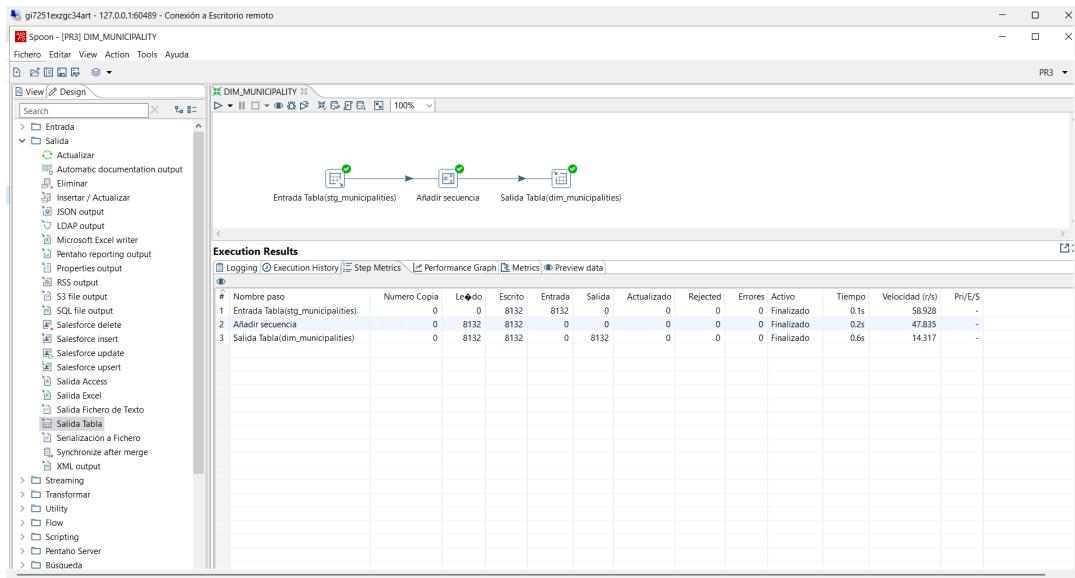
Evidencia (Figura D9.3): configuración del *Table output* con el mapeo visible.

3.6.4.4 Ejecución y verificación en Spoon

Ejecuté la transformación y verifiqué que finaliza correctamente.



Evidencia (Figura D9.4): logging / finalización correcta de la transformación.



Evidencia (Figura D9.5): Step Metrics con filas leídas/escritas y 0 errores.

3.6.4.5 Validación en PostgreSQL

Para validar la carga comprobé la volumetría final y la coherencia con el **DISTINCT** aplicado sobre staging utilizando la misma lógica de transformación, además de revisar una muestra de registros.

The screenshot shows the pgAdmin interface connected to 'PostgreSQL_PR1'. The Object Explorer on the left lists databases: SOURCE_sjimenezcorra, SOURCE_smunozno, SOURCE_spalomosa, SOURCE_aronquillo, SOURCE_ssamperp, SOURCE_stomash, and SOURCE_suave. The 'Schemas (2)' section under 'SOURCE_suave' includes 'dbo' and 'dgt_type'.

The main window displays a query result for the 'Query' tab:

```

SELECT COUNT(*) AS n_dim_municipalities
FROM dbo.dim_municipalities;

```

The result table shows:

| n_dim_municipalities | bigint |
|----------------------|--------|
| 1 | 8132 |

At the bottom, a message indicates: 'Successfully run. Total query runtime: 208 msec. 1 rows affected.'

Evidencia (Figura D9.6): COUNT(*) en la tabla dbo.dim_municipalities.

```

1 SELECT COUNT(*) AS n_stg_distinct
2 FROM (
3   SELECT DISTINCT
4     cod_ine,
5     cod_geo,
6     cod_prov,
7     UPPER(TRIM(provincia)) AS provincia,
8     UPPER(TRIM(name_muni)) AS name_muni,
9     population_muni,
10    surface,
11    perimeter,
12    cod_ine_capital,
13    UPPER(TRIM(capital)) AS capital,
14    population_capital,
15    longitude_etr89,
16    latitude_etr89,
17    altitude
18   FROM dbo.stg_municipalities
19  WHERE cod_ine IS NOT NULL
20    AND cod_geo IS NOT NULL
21    AND name_muni IS NOT NULL
22 ) x;
23

```

Total rows: 1 of 1 Query complete 00:00:00.280 ✓ Successfully run. Total query runtime: 280 msec. 1 rows affected. Ln 7, Col 4

```

1 SELECT *
2 FROM dbo.dim_municipalities
3 ORDER BY municipalities_id
4 LIMIT 15;
5

```

| | municipalities_id [PK] bigint | cod_ine bigint | cod_geo integer | cod_prov integer | provincia text | name_muni text | population_muni integer | surface numeric | perimeter numeric | cod_ine_capital bigint |
|----|-------------------------------|----------------|-----------------|------------------|-------------------|----------------------|-------------------------|-----------------|-------------------|------------------------|
| 1 | 1 | 31185000000 | 31555 | 31 | NAVARRA | OCHAGAVÍA/OTSAGABIA | 490 | 12916.0395 | 71600 | 3118500040 |
| 2 | 2 | 10152000000 | 10600 | 10 | CÁCERES | POZUELO DE ZARZÓN | 444 | 4742.5323 | 36940 | 1015200010 |
| 3 | 3 | 43084000000 | 43420 | 43 | TARRAGONA | MIRAVET | 690 | 3270.22 | 31526 | 4308400010 |
| 4 | 4 | 47161000000 | 47640 | 47 | VALLADOLID | SIMANCAS | 5493 | 4236.3539 | 33695 | 4716100010 |
| 5 | 5 | 44016000000 | 44060 | 44 | TERUEL | ALFAMBRA | 482 | 12244.3056 | 51091 | 4401600010 |
| 6 | 6 | 46099000000 | 46297 | 46 | VALÉNCIA/VALENCIA | CORTES DE PALLÁS | 753 | 23301.2831 | 77472 | 4609900010 |
| 7 | 7 | 50048000000 | 50144 | 50 | ZARAGOZA | BERRUENO | 33 | 1950.1226 | 18332 | 5004800010 |
| 8 | 8 | 44147000000 | 44508 | 44 | TERUEL | MAZALEÓN | 487 | 8624.9085 | 39780 | 4414700010 |
| 9 | 9 | 47085000000 | 47348 | 47 | VALLADOLID | MEDINA DEL CAMPO | 20990 | 15219.01 | 75561 | 4708500020 |
| 10 | 10 | 33010000000 | 33100 | 33 | ASTURIAS | CANDAMO | 1932 | 7197.2501 | 40614 | 3301004040 |
| 11 | 11 | 34112000000 | 34384 | 34 | PALENCIA | NOGAL DE LAS HUERTAS | 44 | 1380.1093 | 17846 | 3411200010 |
| 12 | 12 | 23904000000 | 23740 | 23 | JÁEN | SANTIAGO-PONTONES | 2729 | 68234.08 | 201604 | 2390400310 |
| 13 | 13 | 37116000000 | 37208 | 37 | SALAMANCA | DÓÑINOS DE LEDESMA | 67 | 4615.2749 | 47586 | 3711600010 |
| 14 | 14 | 19303000000 | 19801 | 19 | GUADALAJARA | VALEDECUBO | 38 | 1383.4118 | 17204 | 1930300010 |
| 15 | 15 | 46116000000 | 46348 | 46 | VALENCIA/VALENCIA | LELIANA | 19054 | 871.03 | 14365 | 4611600010 |

Total rows: 15 of 15 Query complete 00:00:00.118 ✓ Successfully run. Total query runtime: 118 msec. 15 rows affected. Ln 5, Col 4

Evidencia (Figuras D9.7 y D9.8): comprobación contra el **DISTINCT** de staging y/o muestra de datos.

3.6.5 Adaptación de FACT_VEHICLE_REGISTRATION + JOB_LOAD

He realizado las tareas solicitadas:

- (a) Modificar estructura de la FACT para incorporar la columna de vinculación con municipios (Figura D10.3).
- (b) Adaptar la transformación de carga para poblar la FK desde la dimensión (Figura D10.4 y mapeos D10.6).
- (c) Adaptar el JOB_LOAD para incorporar DIM_MUNICIPALITY y la FACT modificada (D11.4.4 y D11.4.5, además de ejecución D11.6.*).

3.6.6 Traslado al Data Mart en SQL Server (pendiente de ejecución por incidencia)

El enunciado requiere:

1. Crear tablas destino en SQL Server.
2. Crear transformaciones de copia **DM_DIM_*** y **DM_FACT_***.
3. Crear el job **JOB_DM_LOAD**.

En mi caso, actualmente **no he podido ejecutar esta fase** por problemas de conexión/autenticación a SQL Server en el entorno VDI, por lo que **no dispongo aún de capturas** de esta parte. En cuanto se resuelva el acceso, completaré:

- ejecución del script de creación de tablas en SQL Server,
- creación/ejecución de **DM_DIM_DATE**, **DM_DIM_VEHICLE**, **DM_DIM_MUNICIPALITY**, **DM_DIM_DGT_TYPE**, **DM_FACT_VEHICLE_REGISTRATION**,
- creación y ejecución de **JOB_DM_LOAD**,
- validaciones (conteos y nulos en FKs) directamente en SQL Server, incluyendo las capturas correspondientes en la entrega final.

3.7 Conclusión

He construido el **modelo estrella en PostgreSQL** mediante Spoon, validando la carga correcta de las dimensiones y la tabla de hechos con métricas de ejecución y comprobaciones en base de datos. Además, he incorporado la nueva dimensión **DIM_MUNICIPALITY**, he adaptado la **FACT** para vincularla con dicha dimensión y he actualizado el **JOB_LOAD** para ejecutar el proceso completo de forma secuencial y controlada.

La fase final de copia al **Data Mart en SQL Server** queda pendiente de ejecución por incidencia de acceso, y se completará en cuanto el entorno permita la conexión.