Pentaho

ÍNDICE

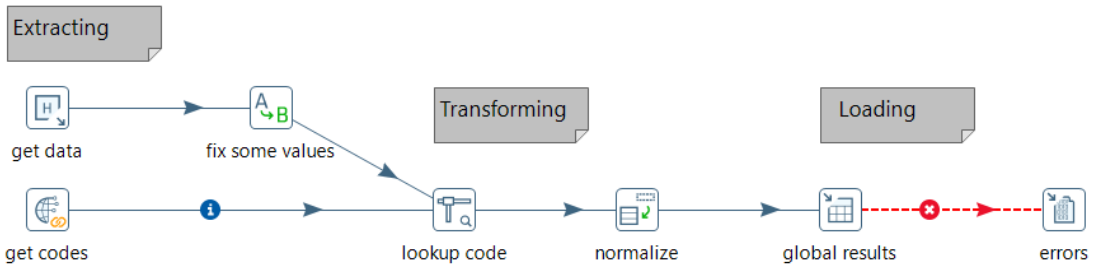
Introducción a Pentaho Data Integration (PDI).

¿Qué es Pentaho?

Pentaho es una **plataforma de integración y análisis de datos** desarrollada por Hitachi Vantara, ampliamente utilizada en entornos de **Business Intelligence (BI)**, **Big Data** y **Data Warehousing**.

Su principal objetivo es facilitar el diseño y la automatización de procesos de **extracción, transformación y carga (ETL)**, permitiendo a las organizaciones **unificar, limpiar y analizar sus datos** de forma eficiente.

La herramienta más representativa de Pentaho es **Pentaho Data Integration (PDI)**, también conocida como **Kettle**, que proporciona un entorno visual para crear flujos de datos sin necesidad de programar manualmente.



Pentaho - Ejemplo de flujo ETL

Kettle y sus componentes.

Kettle es el núcleo de PDI y está compuesto por varias aplicaciones que trabajan de manera conjunta:

Componente	Descripción	Extensión de archivos
Spoon	Entorno gráfico para diseñar transformaciones y trabajos ETL.	.ktr (transformaciones) / .kjb (trabajos)
Pan	Motor de ejecución de transformaciones (sin interfaz gráfica).	.ktr
Kitchen	Motor de ejecución de trabajos completos (orquesta transformaciones).	.kjb

En resumen:

- **Spoon** se usa para diseñar.
- **Pan** ejecuta transformaciones.
- **Kitchen** ejecuta procesos más complejos o secuenciales.

¿Qué se puede hacer con Spoon?

El entorno **Spoon** permite **crear procesos ETL** (Extracción, Transformación y Carga) **de manera visual** y muy intuitiva.

A través de un sistema de componentes arrastrar-y-soltar, se pueden construir flujos de datos sin escribir código.

Algunas de las operaciones más comunes son:

- **Conexión a fuentes de datos:** bases de datos, ficheros planos (CSV, Excel, XML, JSON), APIs, etc.
- **Transformaciones:** filtrado, limpieza, enriquecimiento, combinación, formateo y agregación de datos.
- **Aplicación de fórmulas y cálculos** personalizados.
- **Carga de datos transformados** en distintos formatos o destinos: bases de datos, almacenes de datos, archivos, o sistemas en la nube.
- **Diseño de modelos de Data Warehouse** con esquemas en **estrella (Hechos/Dimensiones)**, listos para análisis OLAP.

Todo esto se realiza **sin programar código**, lo que hace que PDI sea considerado una **herramienta de metadatos**:

trabaja a nivel de **definición del proceso (qué hacer)**, mientras que **el "cómo hacerlo"** queda gestionado por la propia herramienta de manera transparente.

Características principales de Pentaho PDI

Característica	Descripción
Multiplataforma	Funciona en Windows, Linux y macOS.
Open Source	Disponible en versión *Community Edition* (gratuita).
Versión comercial	Ofrece soporte técnico y funciones avanzadas mediante suscripción (*Enterprise Edition*).
Integración amplia	Compatible con múltiples motores de bases de datos (MySQL, PostgreSQL, Oracle, SQL Server, etc.) y servicios Big Data (Hadoop, Spark, etc.).
Escalabilidad	Los flujos ETL pueden ejecutarse de forma local, programada o en servidores distribuidos.

Descarga e instalación.

La **versión Community** puede descargarse desde el portal oficial de Hitachi Vantara:

GitHub - ambientelivre/legacy-pentaho-ce: Legacy versions - Pentaho CE

Pentaho Data Integration – Community Edition

- Descarga el archivo: **pme-ce-9.4.0.0-343.zip**
- Última versión estable (en este contexto): **9.4**, publicada en **noviembre de 2022**.
- También se puede trabajar con versiones anteriores (9.1 o 9.2) tanto en **Windows** como en **Ubuntu**.

Requisito previo

Pentaho necesita **Java 8** (JDK 1.8).

Es imprescindible tenerlo instalado antes de ejecutar Spoon.

Vídeo donde se explica como instalar PDI en W11

Instalación en Ubuntu paso a paso

1. **Instalar Java 8**

   ```
   sudo apt install openjdk-8-jdk
   ```

   ```
   sudo update-alternatives --config java
   ```

   Luego selecciona la versión 8 de la lista.

2. **Instalar dependencias gráficas**

   Pentaho requiere la librería libwebkitgtk, que puede no venir preinstalada.

   - Añadir el repositorio correspondiente:

     ```
     sudo nano /etc/apt/sources.list
     ```

     Y añadir al final:

     ```
     deb http://cz.archive.ubuntu.com/ubuntu bionic main universe
     ```

   - Actualizar repositorios:

     ```
     sudo apt-get update
     ```

   - Instalar el paquete:

     ```
     sudo apt-get install libwebkitgtk-1.0.0
     ```

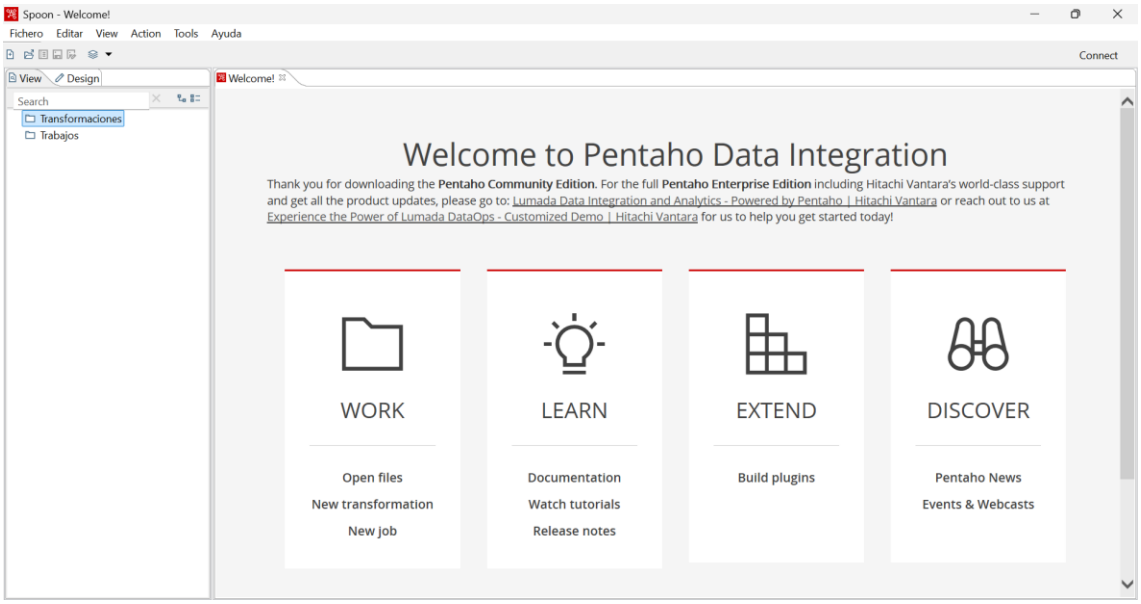
   Si aparece algún error de certificados, puede solucionarse instalando **Y PPA Manager**, según la guía oficial.

3. **Lanzar Spoon**

   Una vez descomprimido el archivo .zip, ejecuta:

   - En Windows → spoon.bat
   - En Linux → spoon.sh

   Aparecerá la **pantalla de inicio de Spoon**, donde se pueden diseñar transformaciones y trabajos.



Pantalla de inicio de Pentaho/Spoon

Dentro de Spoon

Una vez dentro del entorno gráfico, se pueden crear y guardar los flujos ETL:

- Los **archivos .ktr** representan **transformaciones de datos** (ejecutadas por Pan).
- Los **archivos .kjb** representan **trabajos o procesos** que combinan varias transformaciones (ejecutados por Kitchen).

Ejemplo:

Podemos diseñar un flujo que lea datos de un CSV, los limpie y cargue el resultado a una base de datos SQL, todo desde un lienzo visual de pasos conectados por flechas.

Conceptos clave de ETL (contexto)

Fase	Descripción	Ejemplo en Pentaho
E (Extract)	Obtiene los datos desde una o varias fuentes.	Conector de base de datos, lectura de CSV.
T_(Transform)	Limpia, valida y transforma los datos.	Filtros, reemplazos, cálculos, uniones.
L (Load)	Carga los datos procesados en el destino final.	Inserción en base de datos o exportación a Excel/JSON.

El enfoque de Pentaho es **visual**, lo que lo hace especialmente útil para:

- proyectos de integración rápida,
- docentes y analistas que no son programadores,
- y equipos que necesitan documentar fácilmente sus flujos de datos.

Casos de uso típicos de Pentaho PDI

- **Integración de múltiples fuentes** (bases SQL, APIs, Excel, JSON, etc.).
- **Migraciones de datos** entre sistemas.
- **Creación de Data Warehouses** y modelos multidimensionales.
- **Procesos de calidad de datos** (detección de duplicados, limpieza).
- **Automatización de flujos periódicos** con Kitchen o tareas programadas en servidores.

Conclusión

Pentaho Data Integration (Kettle) es una herramienta **potente, flexible y accesible** para cualquier profesional del dato.

Su enfoque **visual, multiplataforma y sin código** permite construir flujos ETL complejos de forma sencilla, transparente y rápida.

Gracias a sus distintos motores (Spoon, Pan y Kitchen), es posible **diseñar, probar y desplegar procesos de integración de datos a gran escala**.

Esto la convierte en una solución ideal tanto para proyectos educativos como para entornos empresariales de **Business Intelligence, Big Data y analítica avanzada**.

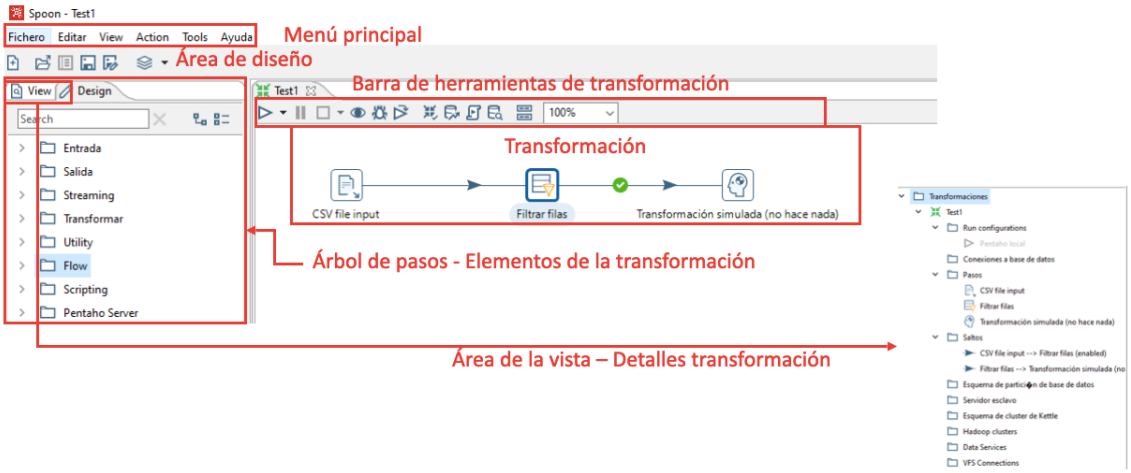
Elementos

En PDI hay dos tipos de elementos: *Transformations* y *Jobs*. (Transformaciones y Trabajos)

- Se definen *Transformations* para transformar los datos, describiendo flujos de datos para ETL como leer desde una fuente, transformar los datos y cargarlos en un nuevo destino.
- Se definen *Jobs* para organizar tareas y transformaciones estableciendo su orden y condiciones de ejecución (del tipo ¿existe el fichero X en origen? ¿existe la tabla X en mi base de datos?).

Tanto las transformaciones como las tareas, cuando se definen, se almacenan como archivos.

Los elementos del interfaz son:



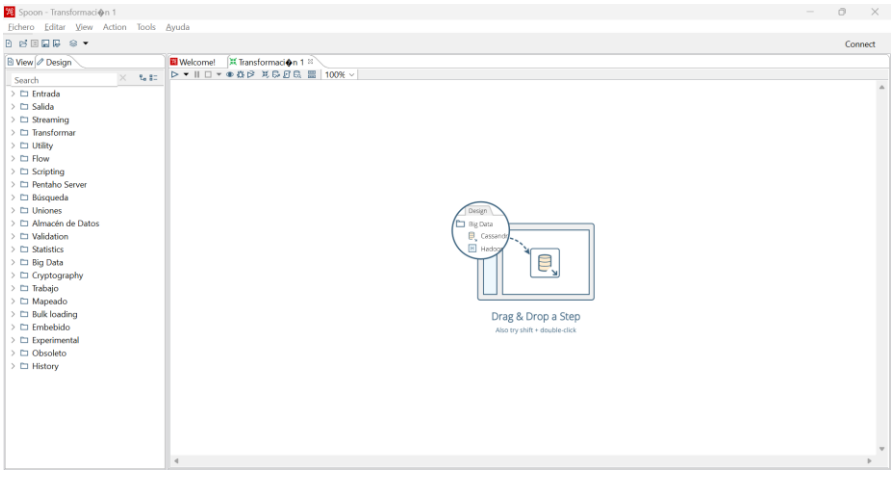
Interfaz de Spoon

Caso de Uso 0. Guía Paso a Paso: Transformación Básica para Obtener la Versión de PDI.

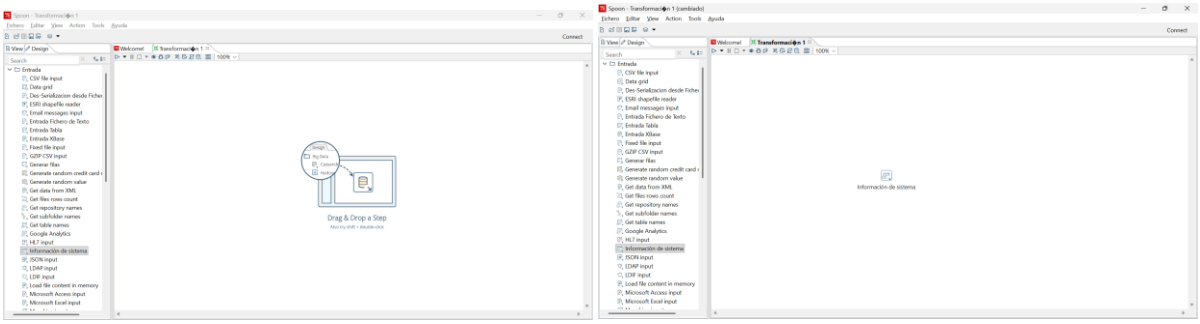
Este ejercicio introductorio tiene como objetivo familiarizarnos con el entorno de **Pentaho Data Integration (PDI)**, una herramienta clave en el procesamiento de **Big Data** (ETL/ELT). Crearemos una transformación simple para obtener y mostrar la versión de Kettle que se está utilizando.

1. Preparación de la Transformación

1. **Crear una Nueva Transformación:**
    - Vete a **File** -> **New Transformation** para abrir el área de trabajo.



2. **Añadir el Paso de Entrada (Input):**
    - En el **árbol de pasos** (la lista de categorías y pasos), navega hasta la categoría **Input** (Entrada).
    - Selecciona el paso **Get system info** (Información del sistema) y arrástralo a la zona de trabajo. Este paso es fundamental para obtener metadatos y variables del sistema.

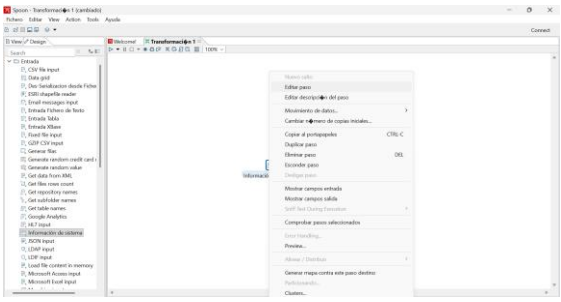


Menú emergente

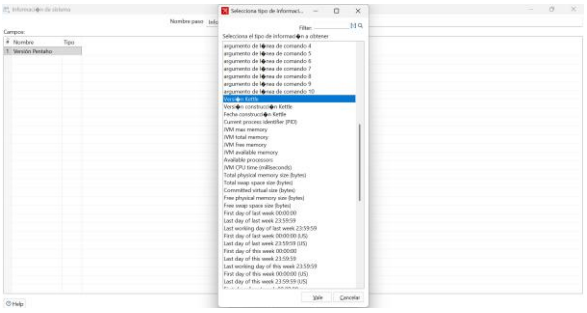
2. Configuración de los Pasos

A. Configurar "Get system info" (“Información de sistema”)

1. **Editar Propiedades:**
    - Haz doble clic en el paso **Get system info** (**Información de sistema**) o utiliza la opción de menú contextual (botón derecho) para abrir sus propiedades en “Editar paso”.

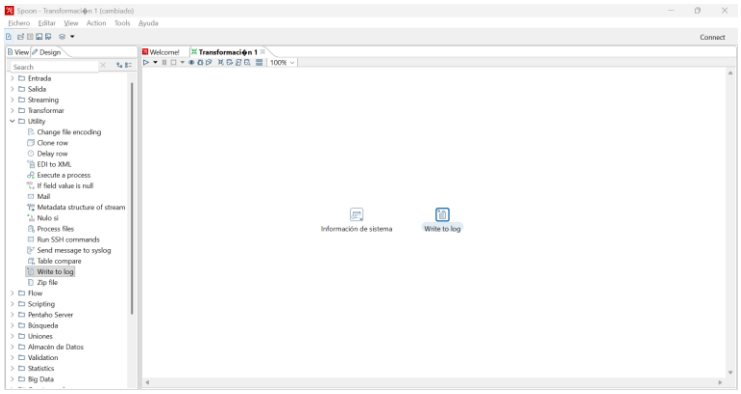


    - Configura la información que deseas obtener:
        - Crea una nueva propiedad con el nombre: **Versión Pentaho** (este será el nombre del campo de salida).
        - En el desplegable correspondiente a la fuente de información, selecciona la opción: **Kettle Version**.



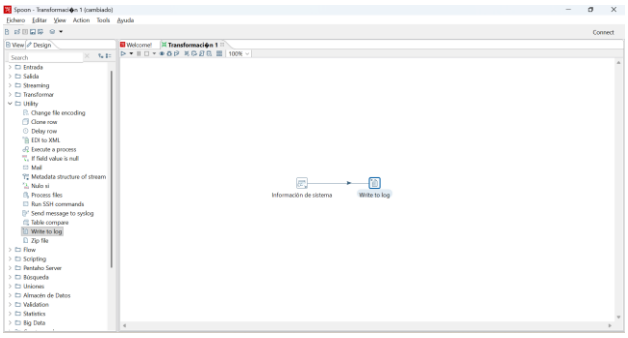
B. Añadir el Paso de Utilidad (Utility)

1. **Añadir "Write to Log":**
    - En el **árbol de pasos**, navegamos hasta la categoría **Utility**.
    - Selecciona y arrastra el paso **Write to Log** al área de trabajo. Este paso es ideal para depuración y para mostrar datos en el panel de resultados.

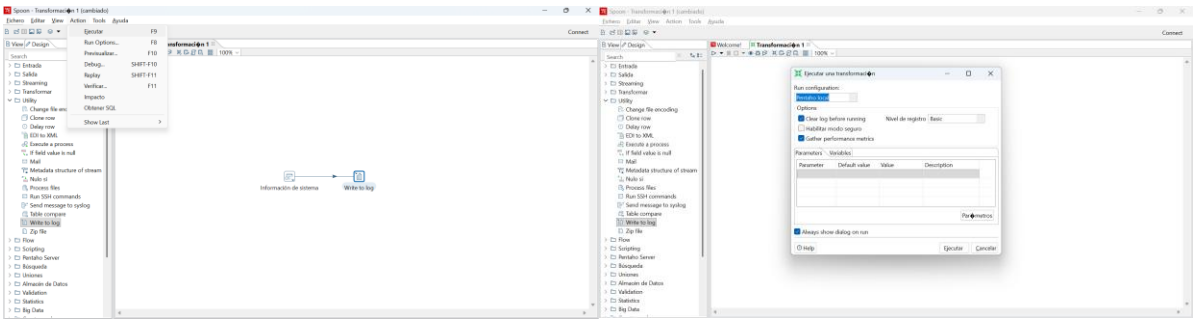


3. Conexión y Ejecución

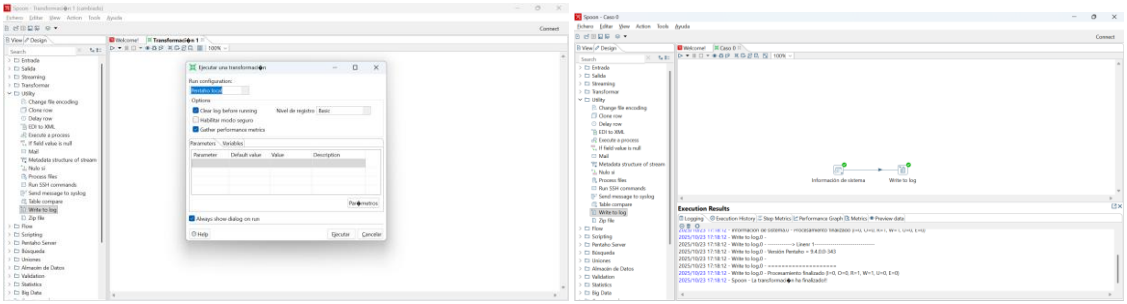
1. **Conectar los Pasos (Crear Flujo):**
    - Sitúa el ratón sobre el paso **Get system info** hasta que aparezca el **menú emergente**.
    - Selecciona la **4ª opción del menú emergente** (típicamente una flecha para **nueva salida/conexión**).
    - Haz clic en el paso **Write to Log**. Esto establece el flujo de datos: la información del sistema se pasará al registro.



2. **Ejecutar la Transformación:**
    - Ejecuta la transformación haciendo clic en el icono del **triángulo verde** (el botón **Run**) en la barra de herramientas, o pulsando la tecla de acceso rápido **F9**.



3. **Visualizar el Resultado:**
    - El resultado, que contendrá la **Versión Pentaho** obtenida por el paso inicial, aparecerá en el **panel inferior** de la pantalla, en la pestaña de **Logging** (Registro).



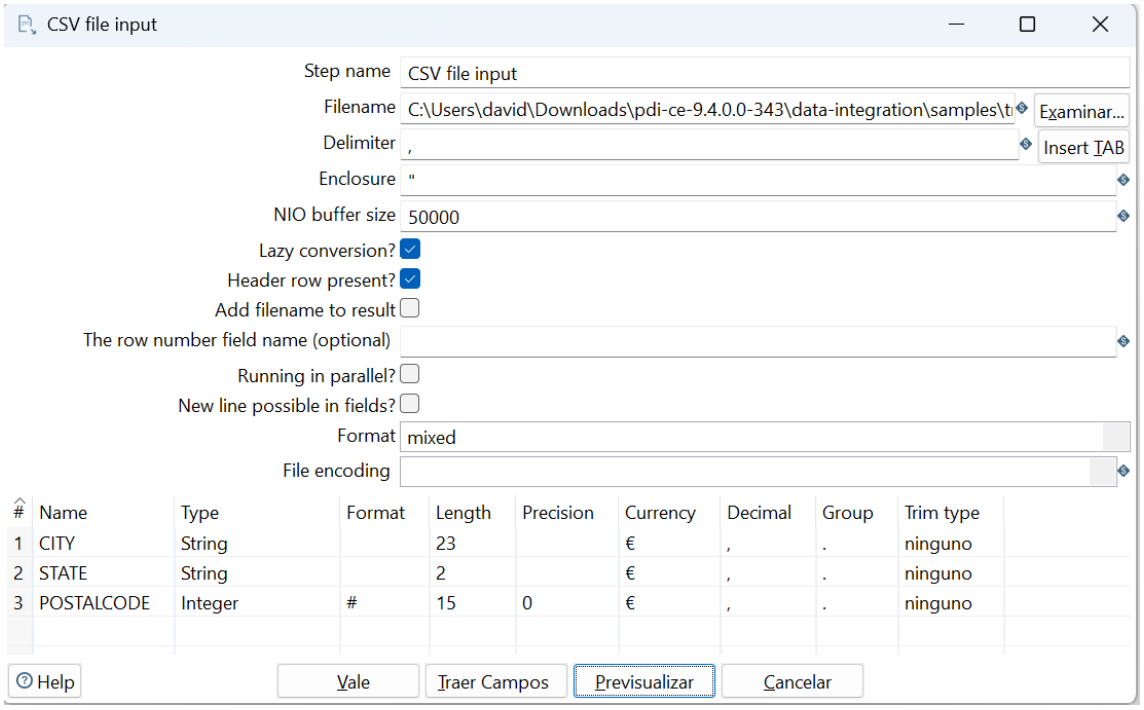
Caso de Uso 0 - Versión de PDI

Caso de Uso 1. Guía Paso a Paso: Lectura, Filtrado y Ordenación de Datos CSV.

Este caso de uso avanza en el manejo de PDI (Spoon) al enseñar cómo leer datos desde un archivo **CSV**, aplicar **filtros** y **ordenación**, **gestionar errores** y, finalmente, **ejecutar la transformación** de forma desatendida desde la línea de comandos (Terminal).

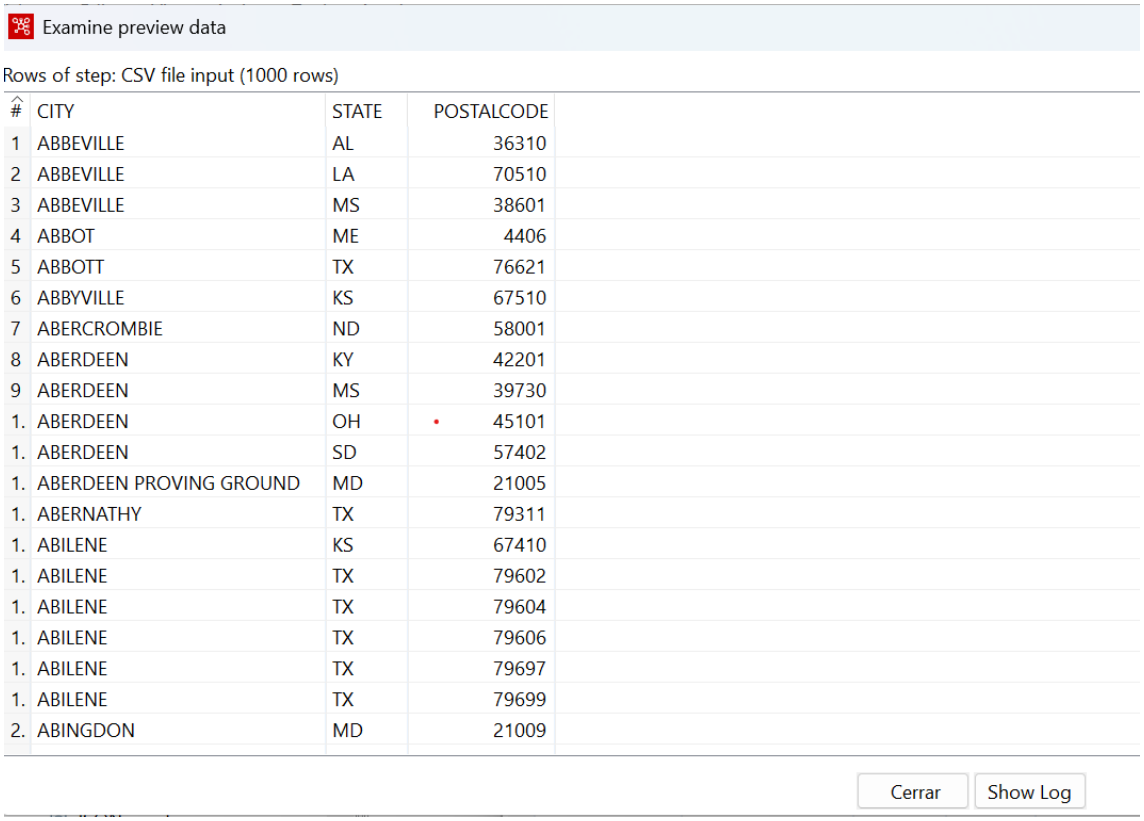
Lectura del Archivo de Entrada (CSV)

1. **Crear una Nueva Transformación:**
    - Cree una nueva transformación (**Ctrl + N**).
2. **Añadir el Paso de Lectura CSV:**
    - Desde la categoría **Input** (Entrada), arrastra el paso **CSV file input** (Entrada de archivo CSV) al área de trabajo.
3. **Configurar el Archivo:**
    - Haz doble clic en el paso para configurarlo.
    - Selecciona la ruta del archivo de ejemplo dentro de tu instalación de Pentaho: samples\transformations\files\Zipssortedbycitystate.csv.
4. **Cargar y Verificar Metadatos:**
    - Pulsa el botón **Get Fields** (Obtener Campos) para que PDI cargue automáticamente los nombres y tipos de datos de las columnas del CSV.
    - Verifica que los campos sean correctos.
    - **Traer campos y Previsualización:** Utiliza el botón **Traer Campos** y cuando se hayan cargado en caché utiliza el botón **Preview** (Previsualizar) para comprobar que la lectura de los datos se realiza sin problemas.



Caso de Uso 1 - Tras pulsar sobre Get Fields

Tras ello, mediante el botón *Preview* comprobaremos que los datos se leen correctamente.



#	CITY	STATE	POSTALCODE
1	ABBEVILLE	AL	36310
2	ABBEVILLE	LA	70510
3	ABBEVILLE	MS	38601
4	ABBOT	ME	4406
5	ABBOTT	TX	76621
6	ABBYVILLE	KS	67510
7	ABERCROMBIE	ND	58001
8	ABERDEEN	KY	42201
9	ABERDEEN	MS	39730
1.	ABERDEEN	OH	45101
1.	ABERDEEN	SD	57402
1.	ABERDEEN PROVING GROUND	MD	21005
1.	ABERNATHY	TX	79311
1.	ABILENE	KS	67410
1.	ABILENE	TX	79602
1.	ABILENE	TX	79604
1.	ABILENE	TX	79606
1.	ABILENE	TX	79697
1.	ABILENE	TX	79699
2.	ABINGDON	MD	21009

Caso de Uso 1 - Resultado de la opción Preview sobre Ciudades

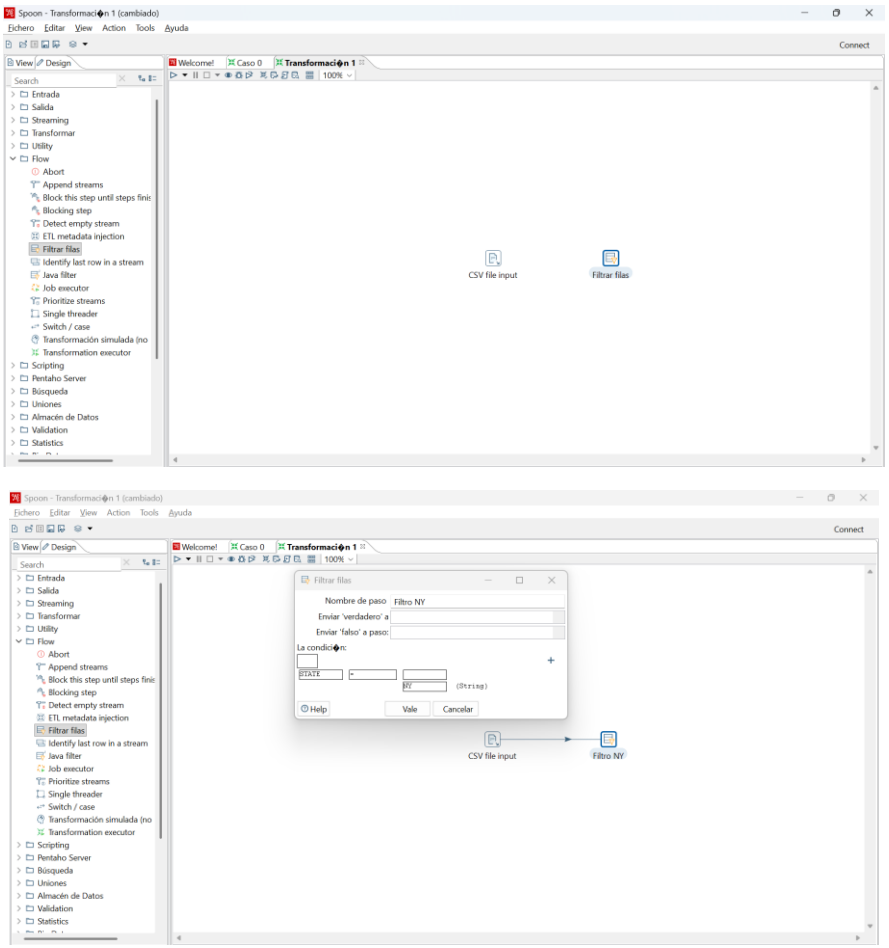
Filtrado de filas (datos)

Una vez leído, el siguiente paso es filtrar las filas. Para ello, desde la categoría de *Flow*, arrastramos el paso *Filter* (Filtrar filas), y las conectamos tal como hemos realizado en el caso anterior. Al soltar la flecha, nos mostrará dos opciones:

- *Main output of step*: define los pasos con un flujo principal, donde todo funciona bien

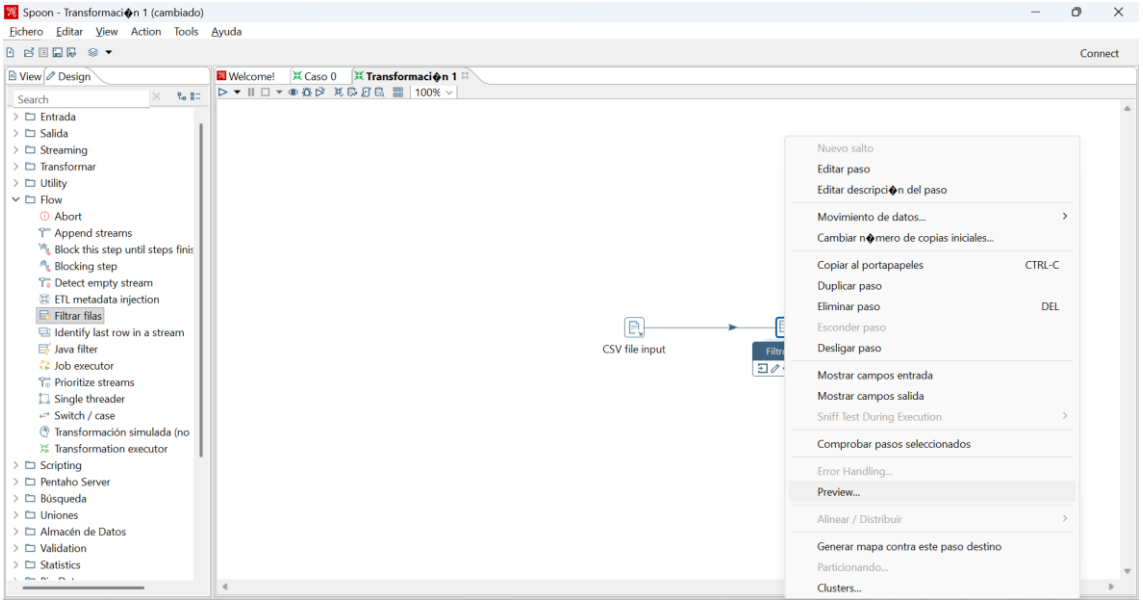
- *Error handling of step*: define los pasos a seguir en caso de encontrar un error

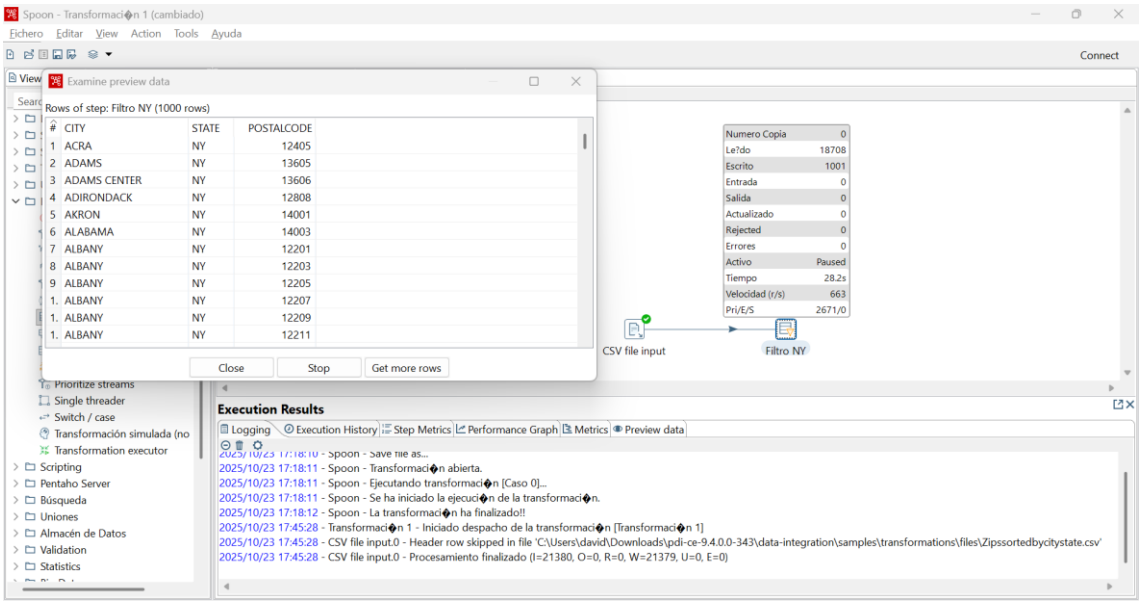
De momento elegimos la primera y configuramos el filtro para solo seleccionar aquellos datos cuyo estado sea NY (STATE = NY)



Caso de Uso 1 - Configuración del filtro

Para configurar el resultado, seleccionamos el paso del filtro, y bien pulsamos sobre el icono del ojo de la barra de herramientas, o sobre el paso, tras pulsar con el botón derecho, seleccionamos la opción *Preview*.





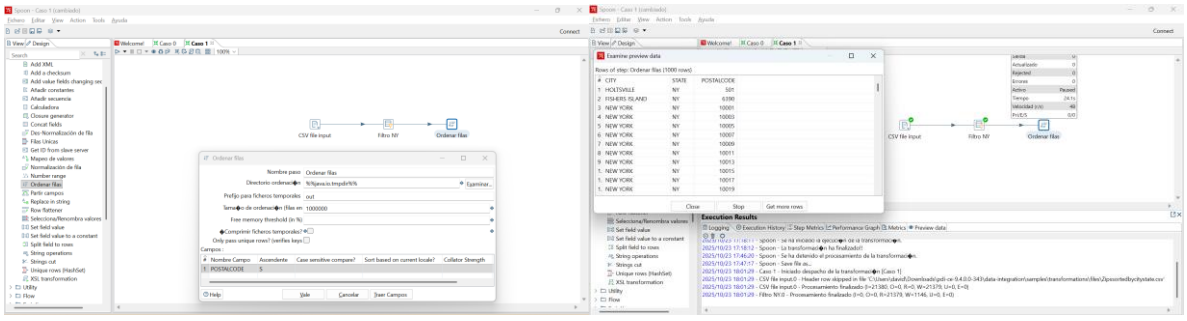
Caso de Uso 1 - Resultado de hacer Preview sobre Filtro NY

Por defecto se precargan 1000 filas. Tras comprobar el resultado, pulsamos sobre *Stop* para detener el proceso de previsualización. Las métricas que aparecen nos información del proceso y su rendimiento.

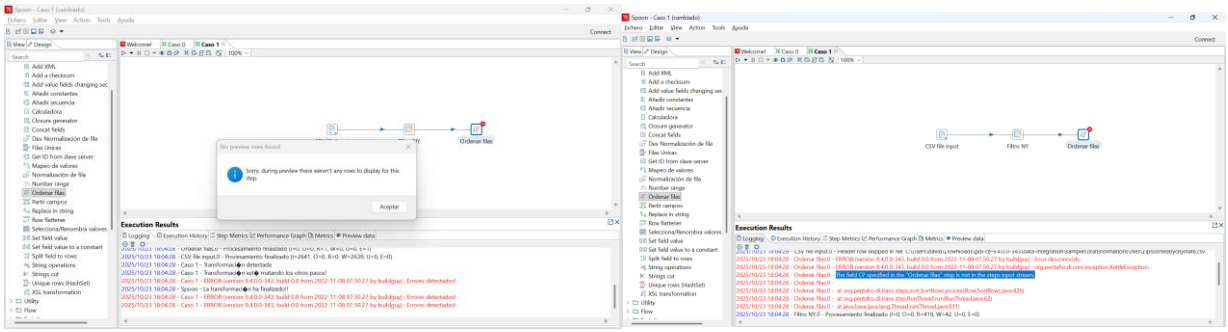
Ordenación y Gestión de Errores

El siguiente paso que vamos a realizar es ordenar los datos por su código *Postal Code*. Para ello:

1. **Añadir el Paso de Ordenación:**
   - Desde la categoría **Transform** (Transformar), arrastra el paso **Sort rows** (Ordenar filas) al área de trabajo.
   - Conecta la salida **principal** (**Main output of step**) de **Filter rows** al paso **Sort rows**.
2. **Configurar la Ordenación:**
   - Edita el paso **Sort rows**.
   - Indica que los datos deben ordenarse por el campo **POSTAL CODE**.



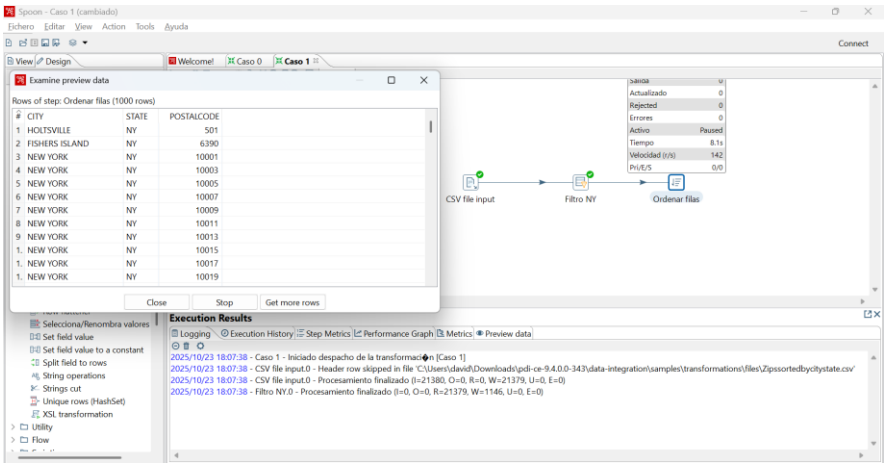
3. **Forzar y Gestionar un Error (Demostración):**
   - **Simulación de Error:** En la configuración de **Sort rows**, reemplaza temporalmente el nombre del campo correcto (**POSTAL CODE**) por uno incorrecto, por ejemplo, **CP**.
   - **Observar el Error en Spoon:** Al intentar previsualizar o ejecutar, verá un **símbolo de prohibido** en la esquina superior derecha del paso. El panel de *Log* y las *Métricas* indicarán el error de que el campo 'CP' no existe en el flujo de datos.



Caso de Uso 1 - Forzando un error

- **Corrección:** Vuelve a editar el paso **Sort rows** y escribe el nombre correcto del campo: **POSTAL CODE**.
- **Comprobación:** Previsualiza el paso nuevamente para confirmar que la ordenación funciona.

Volvemos a editar el paso, corregimos el nombre del campo (escribimos *POSTAL CODE*) y comprobamos que ahora sí que funciona correctamente



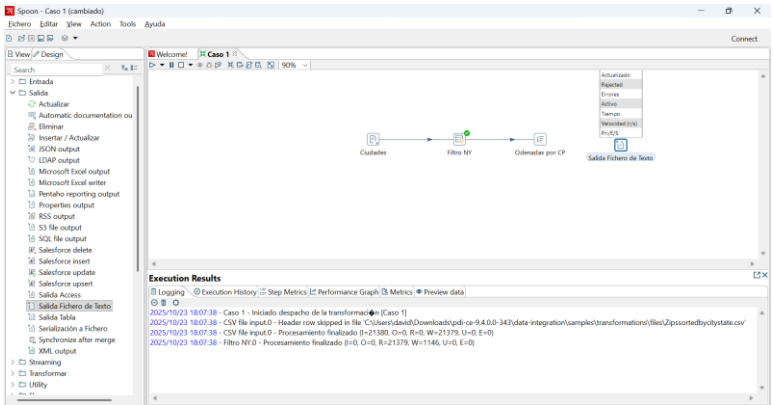
Caso de Uso 1 - Ordenando

Escritura y Ajuste del Resultado

Una vez realizados todos los pasos, sólo nos queda enviar el resultado a un fichero para persistir la transformación.

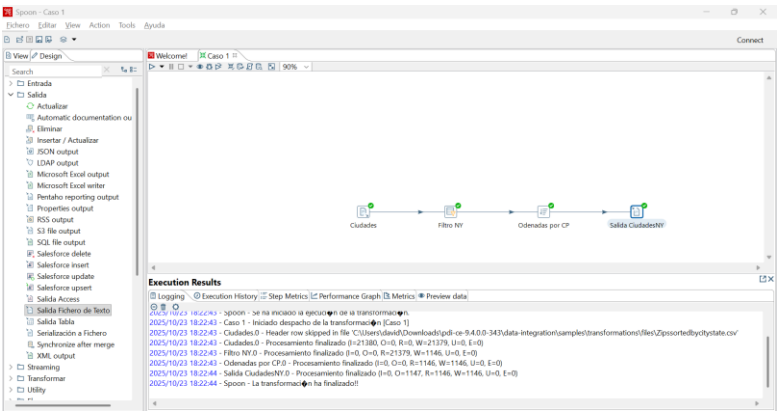
Para ello:

1. **Añadir el Paso de Salida:**
   - Desde la categoría **Output** (Salida), arrastra el paso **Text file output** (Salida a archivo de texto) al área de trabajo.

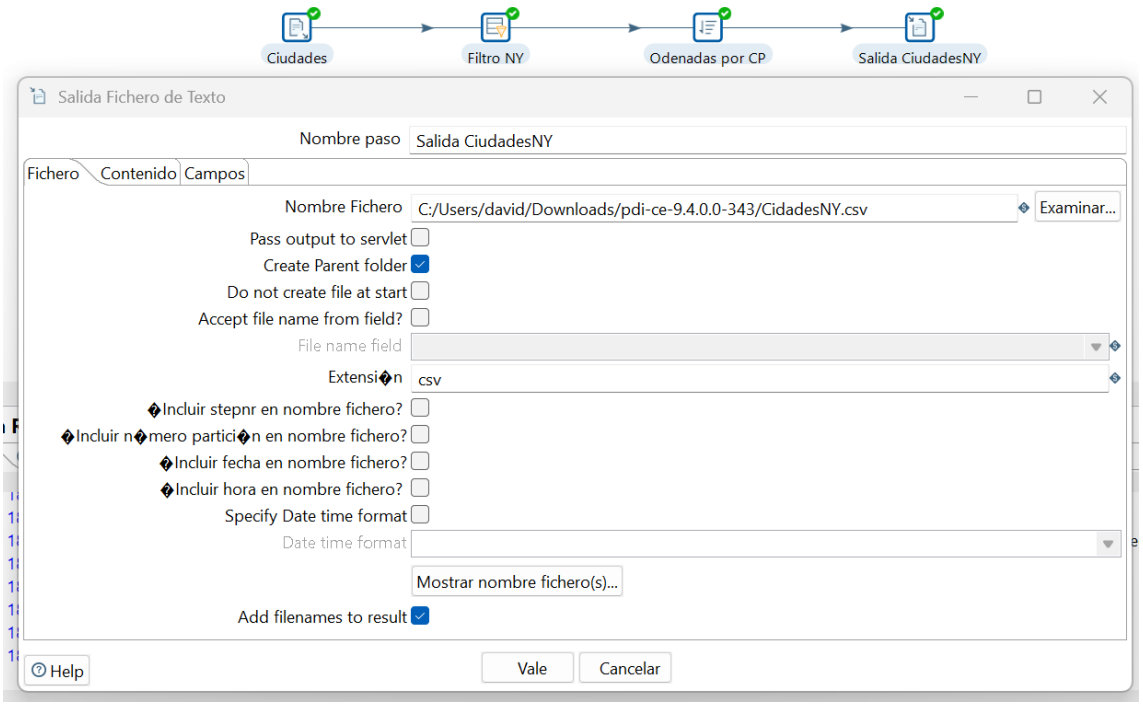


- Conecta la salida de **Sort rows** al paso **Text file output**.

2. **Configurar la Salida:**
   - Edita el paso e indique la ruta y el nombre del archivo donde se guardará el resultado (ej. CiudadesNY.csv).



3. **Ajustar el Ancho Mínimo de los Campos:**
   - **Problema:** Tras la primera ejecución, si revisas el archivo de salida, notarás que las columnas tienen **espacios en blanco** debido al tamaño de campo preconfigurado.
   - **Solución:** Vuelve a editar el paso **Text file output** y, en la pestaña **Fields** (Campos), presiona el botón **Minimal width** (Anchura mínima). Esto ajusta el ancho de las columnas a la longitud mínima necesaria para los datos.

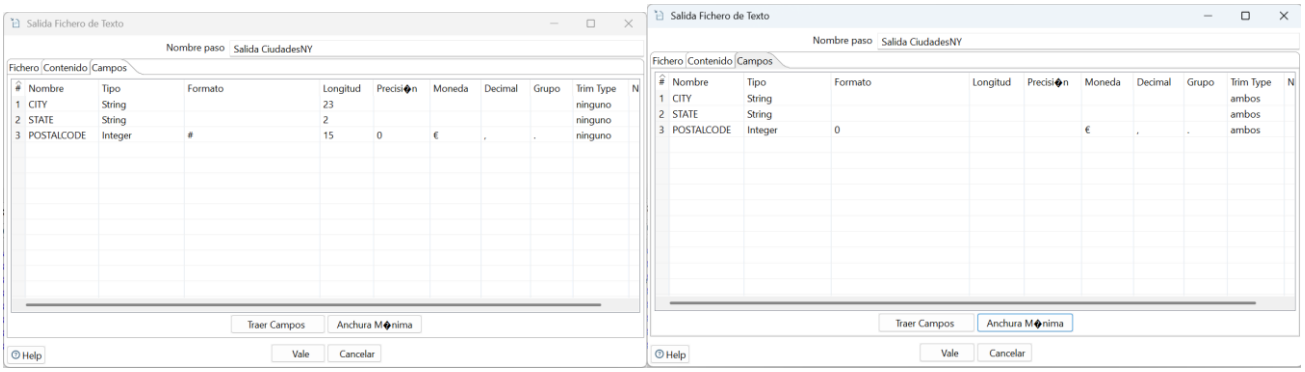


Caso de Uso 1 - Guardando el resultado

4. **Ejecución Final:**
   - Ejecuta la transformación (Icono **Run** o **F9**).
   - El archivo de salida mostrará ahora el formato correcto, sin espacios en blanco innecesarios:

```
CITY;STATE;POSTALCODE
HOLTSVILLE              ;NY;501
FISHERS ISLAND          ;NY;6390
NEW YORK                ;NY;10001
NEW YORK                ;NY;10003
NEW YORK                ;NY;10005
NEW YORK                ;NY;10007
NEW YORK                ;NY;10009
```

Al comprobar el fichero, vemos que se han quedado espacios en blanco a la derecha del nombre de las ciudades, ya que la columna tenía un tamaño configurado. Si volvemos a editar el último paso, en la pestaña de *Fields* (Campos) podemos indicar mediante el botón de *Minimal width* que reduzca su anchura al mínimo:



Caso de Uso 1 - Anchura mínima de los campos

Y tras volver a ejecutar la transformación, veremos que ahora sí que obtenemos los datos que esperábamos:

CITY;STATE;POSTALCODE
HOLTSVILLE;NY;501
FISHERS ISLAND;NY;6390
NEW YORK;NY;10001
NEW YORK;NY;10003
NEW YORK;NY;10005
NEW YORK;NY;10007
NEW YORK;NY;10009

Ejecución desde el Terminal (Uso de Pan)

Pan es una utilidad que permite ejecutar transformaciones de PDI sin la necesidad de abrir la interfaz gráfica (Spoon). Esto es esencial para la automatización de procesos ETL.

1. **Preparación:**
    - Guarda tu transformación (ej. caso1filtradoNY.ktr).
    - **Opcional:** Elimina el archivo de salida generado para verificar la nueva ejecución.
2. **Ejecutar Pan:**
    - Abre el terminal (Símbolo del sistema o Bash).
    - Navega al directorio de PDI y ejecuta el comando pan.bat (Windows) o pan.sh (Linux/Mac), pasándole la ruta del archivo de transformación mediante el parámetro /file:

      ```
      # Ejemplo de ejecución en Windows:

      C:\Users\david\Downloads\pdi-ce-9.4.0.0-343\data-integration\Pan.bat  /file="Caso 1.ktr"
      ```
3. **Verificación:**
    - El terminal mostrará un log de la ejecución, indicando el **inicio**, el **despacho** de los pasos y el **número de filas** procesadas en cada uno, confirmando el éxito del proceso:

```
C:\Users\david\Downloads\pdi-ce-9.4.0.0-343\data-integration\Pan.bat  /file="Caso 1.ktr"
DEBUG: Using PENTAHO_JAVA_HOME
DEBUG: _PENTAHO_JAVA_HOME=C:\Program Files\Java\jdk-16
DEBUG: _PENTAHO_JAVA=C:\Program Files\Java\jdk-16\bin\java.exe

C:\Users\david\Downloads\pdi-ce-9.4.0.0-343\data-integration>"C:\Program Files\Java\jdk-16\bin\java.exe" --add-opens java.base/java.net=ALL-UNNAMED --add-
opens java.base/java.lang=ALL-UNNAMED --add-opens java.base/sun.net.www.protocol.jar=ALL-UNNAMED  "-Xms1024m" "-Xmx2048m" "-
Djava.library.path=libswt\win64;/bin"  -Djava.locale.providers=COMPAT,SPI "-DKETTLE_HOME=" "-DKETTLE_REPOSITORY=" "-DKETTLE_USER=" "-DKETTLE_PASSWORD=" "-
DKETTLE_PLUGIN_PACKAGES=" "-DKETTLE_LOG_SIZE_LIMIT=" "-DKETTLE_JNDI_ROOT=" -jar launcher\launcher.jar -lib ..\libswt\win64  -main org.pentaho.di.pan.Pan -
initialDir "C:\Users\david"\ "/file=Caso 1.ktr"
2025/10/23 18:49:46 - Pan - Start of run.
2025/10/23 18:49:46 - Caso 1 - Iniciado despacho de la transformaci?n [Caso 1]
2025/10/23 18:49:47 - Ciudades.0 - Header row skipped in file 'C:\Users\david\Downloads\pdi-ce-9.4.0.0-343\data-
integration\samples\transformations\files\Zipssortedbycitystate.csv'
2025/10/23 18:49:47 - Ciudades.0 - Procesamiento finalizado (I=21380, O=0, R=0, W=21379, U=0, E=0)
2025/10/23 18:49:47 - Filtro NY.0 - Procesamiento finalizado (I=0, O=0, R=21379, W=1146, U=0, E=0)
2025/10/23 18:49:47 - Odenadas por CP.0 - Procesamiento finalizado (I=0, O=0, R=1146, W=1146, U=0, E=0)
2025/10/23 18:49:47 - Salida CiudadesNY.0 - Procesamiento finalizado (I=0, O=1147, R=1146, W=1146, U=0, E=0)
2025/10/23 18:49:47 - Pan - Finished!
2025/10/23 18:49:47 - Pan - Start=2025/10/23 18:49:46.901, Stop=2025/10/23 18:49:47.771
2025/10/23 18:49:47 - Pan - Processing ended after 0 seconds.
2025/10/23 18:49:47 - Caso 1 -
2025/10/23 18:49:47 - Caso 1 - Los procesos Ciudades.0 han finalizado correctamente, se han procesado 21379 l?neas. ( - l?neas)
2025/10/23 18:49:47 - Caso 1 - Los procesos Filtro NY.0 han finalizado correctamente, se han procesado 21379 l?neas. ( - l?neas)
2025/10/23 18:49:47 - Caso 1 - Los procesos Odenadas por CP.0 han finalizado correctamente, se han procesado 1146 l?neas. ( - l?neas)
2025/10/23 18:49:47 - Caso 1 - Los procesos Salida CiudadesNY.0 han finalizado correctamente, se han procesado 1146 l?neas. ( - l?neas)
```

- Y si comprobamos el fichero, veremos que ha vuelto a aparecer.

Caso de Uso 2 - Uniendo datos

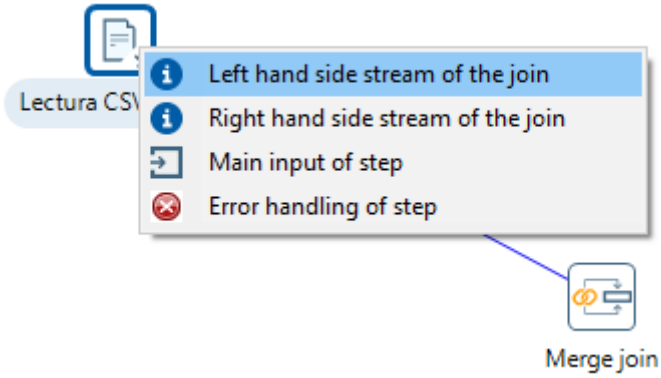
Caso de Uso 2 - Lecturas CSV

En este caso, vamos a leer datos de ventas y de productos, y vamos a unirlos de forma similar a un *join*, para posteriormente, tras agrupar los datos, crear un informe.

Merge join

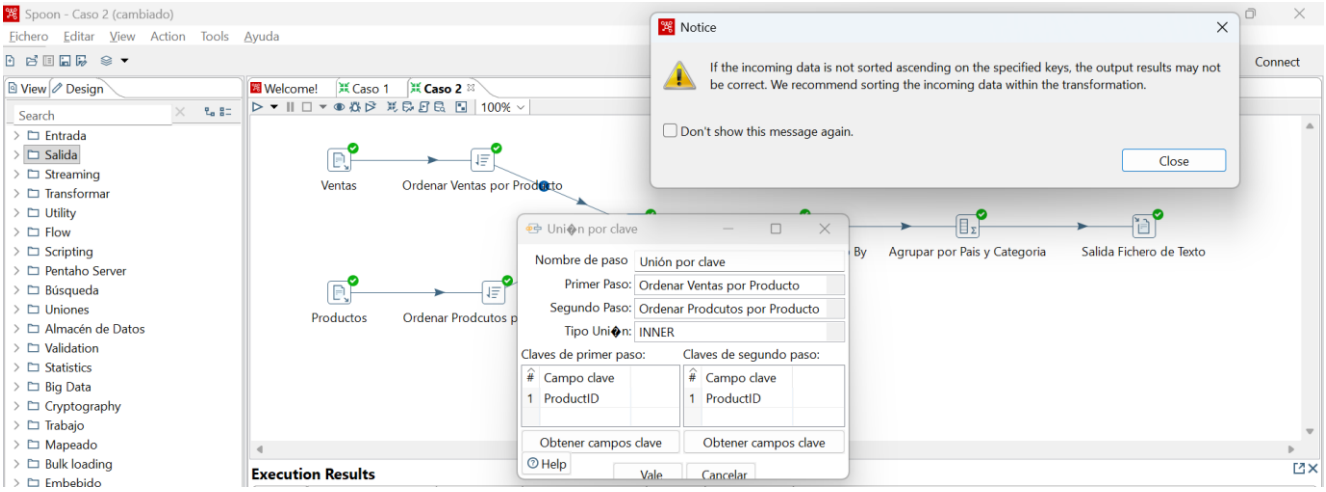
Para ello, primero vamos a leer de forma separada cada archivo mediante un paso de tipo *CSV file input*, utilizando el separador necesario en cada caso (; o ,). Así pues, tendremos dos pasos, tal como se puede observar en la imagen de la derecha. En el caso del CSV de *ventas*, al tener registros de diferentes países, deberemos cambiar el tipo del ZIP (el código postal) a String.

A continuación, para unir los datos, dentro de la categoría *Join/Unión* utilizamos el paso *Merge join/Unión por clave*. En este caso tras arrastrar el paso al área de trabajo, vamos a enlazar desde el merge hacia los dos orígenes, en un caso como *left hand side stream of the join* y el otro como *right hand side stream of the join*:



Caso de Uso 2 - Entradas de Merge

A continuación, editamos el *Merge* y le decimos que el campo de unión es *ProductID*. Cuando le damos a aceptar recibimos un mensaje de advertencia avisándonos que si los datos no están ordenados podemos obtener resultados incorrectos.



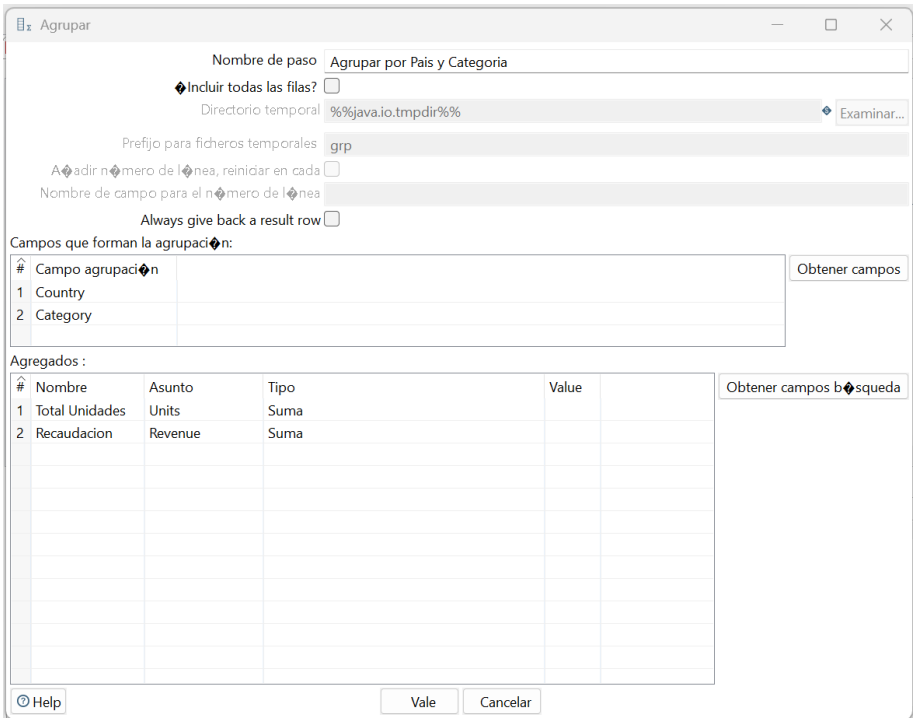
Caso de Uso 2 - Aviso Merge

Así pues, vamos a añadir previo al merge un paso de ordenación a cada entrada, ordenando los datos por ProductID de manera ascendente. Para comprobar su funcionamiento, podemos hacer un preview del merge.

Agrupando datos

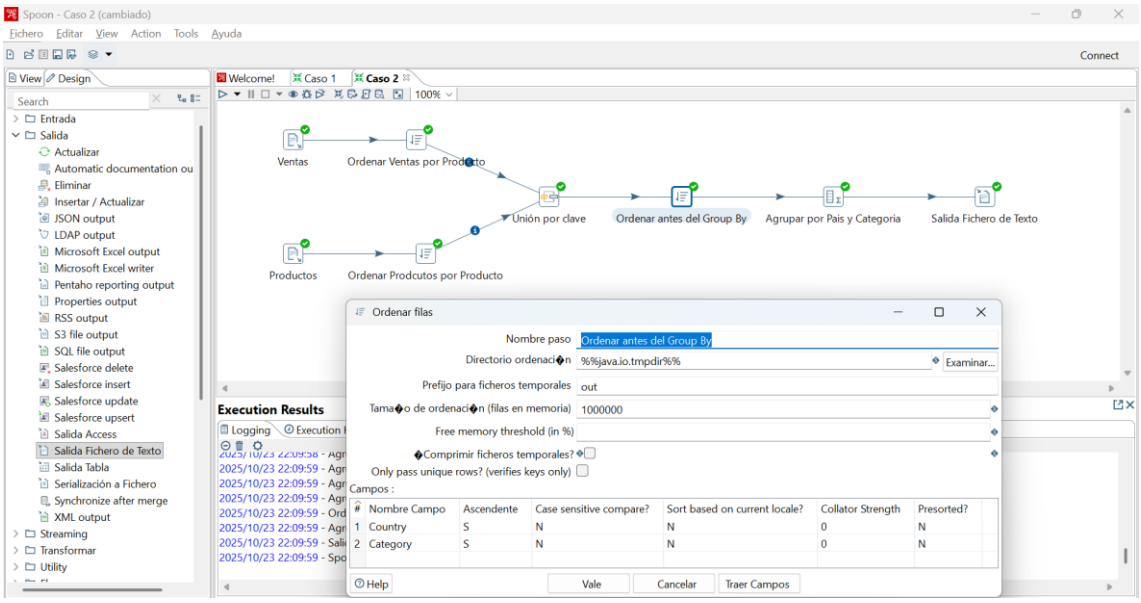
Como el resultado final es un informe de ventas con el total de unidades vendidas y la cantidad recaudada agrupada por país y categoría del producto, necesitamos agrupar los datos.

Para ello, dentro de la categoría *Statistics*, utilizaremos el paso *Group by/Agrupar por*. En nuestro caso queremos agrupar por país (*Country*) y categoría (*Category*), y los datos que vamos a agregar son la suma de unidades (*Units*) y la suma recaudada (*Revenue*). Así pues, nuestra agrupación quedaría así:



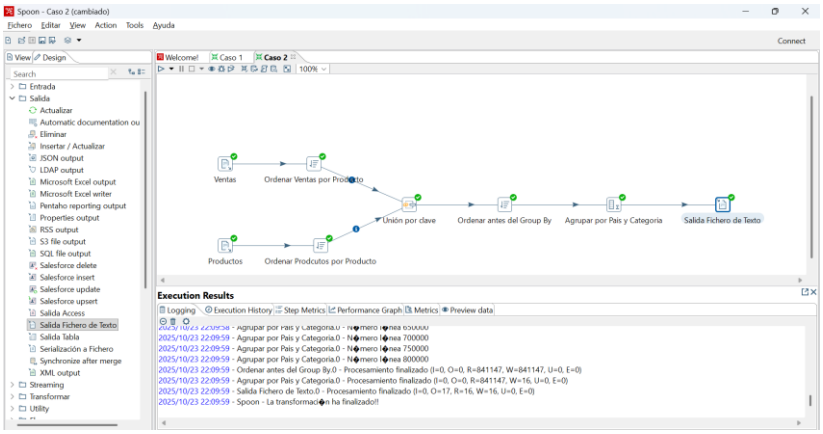
Caso de Uso 2 - Agrupamos por país y categoría

En este caso, sucede lo mismo que antes, que este paso necesita los datos ordenados. Así pues, vamos a ordenar las salidas del merge por país y categoría.



Caso de Uso 2 - Ordenamos antes de agrupar

El último paso que nos queda es exportar los datos a un fichero de texto mediante el paso *Text file output*.



Caso de Uso 2 – ETL final

Caso de Uso 3 - Cuestionarios Airbnb

Para el siguiente caso de uso, vamos a utilizar datos de los cuestionarios de AirBnb que se pueden descargar desde Get the Data | Inside Airbnb.

En concreto, nos vamos a centrar en los datos de Madrid que podemos descargar desde https://data.insideairbnb.com/spain/comunidad-de-madrid/madrid/2025-06-12/visualisations/listings.csv

Basado en las primeras líneas del archivo adjunto, esta es la estructura de las columnas más relevantes, sus tipos de datos esperados en PDI y una breve explicación:

Campo (Original)	**Tipo PDI Sugerido**	**Explicación en Castellano**
id	Integer / Number	Identificador único de la habitación/propiedad. (Necesario renombrar a room_id).

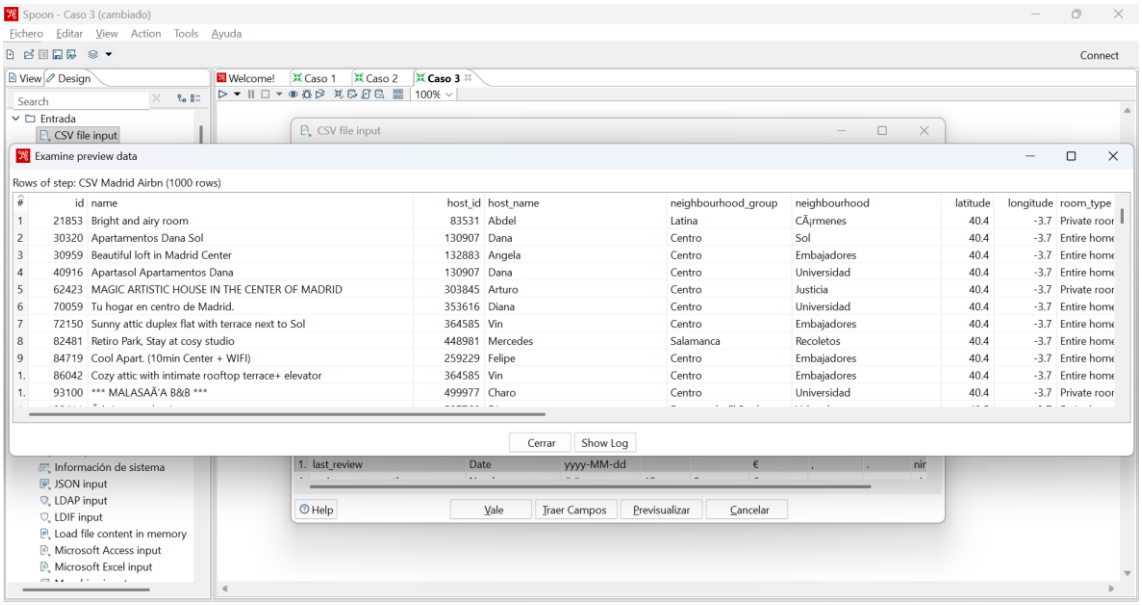
Campo (Original)	Tipo PDI Sugerido	Explicación en Castellano
name	String	Título del anuncio de la propiedad.
host_id	Integer	Identificador del anfitrión.
host_name	String	Nombre del anfitrión.
neighbourhood_group	String	Agrupación superior del barrio (a menudo vacío o igual al nombre de la ciudad).
neighbourhood	String	**Barrio** donde se encuentra el alojamiento.
latitude	Number	Coordenada de latitud.
longitude	Number	Coordenada de longitud.
room_type	String	**Tipo de habitación** (Ej: *Entire home/apt*, *Private room*).
price	Number	**Precio** nocturno de la habitación/propiedad en euros/moneda local.
minimum_nights	Integer	**Estancia mínima** requerida para reservar. (Necesario renombrar a minstay).
number_of_reviews	Integer	Número total de reseñas.
last_review	Date	Fecha de la última reseña.
reviews_per_month	Number	Promedio de reseñas por mes.
calculated_host_listings_count	Integer	Número de propiedades que tiene el anfitrión.
availability_365	Integer	Días disponibles para reservar en el año.
number_of_reviews_ltm	Integer	Número de reseñas en los últimos 12 meses.
license	String	Número de licencia turística.

Una vez descargados los datos y descomprimidos, vamos a cargar el fichero:

Carga del Archivo de Entrada

1. **Crear Nueva Transformación.**
2. **Añadir Paso de Entrada:**
    - Arrastra el paso **CSV file input** (Entrada de archivo CSV) de la categoría **Input**. (Usaremos CSV Input ya que es un único archivo y es más rápido que Text File Input para esta tarea).
3. **Configurar Archivo Único:**
    - Indica la ruta de su archivo **listings.csv**.

4. **Configurar Contenido y Campos:**
    - **Separator:** Asegúrate de que sea la coma (,).
    - Pulse **Get Fields** para cargar la estructura.
    - **Ajuste de Tipos:** Verifica que price y latitude/longitude sean tipo **Number** (para permitir decimales) y id y minimum_nights sean **Integer**.

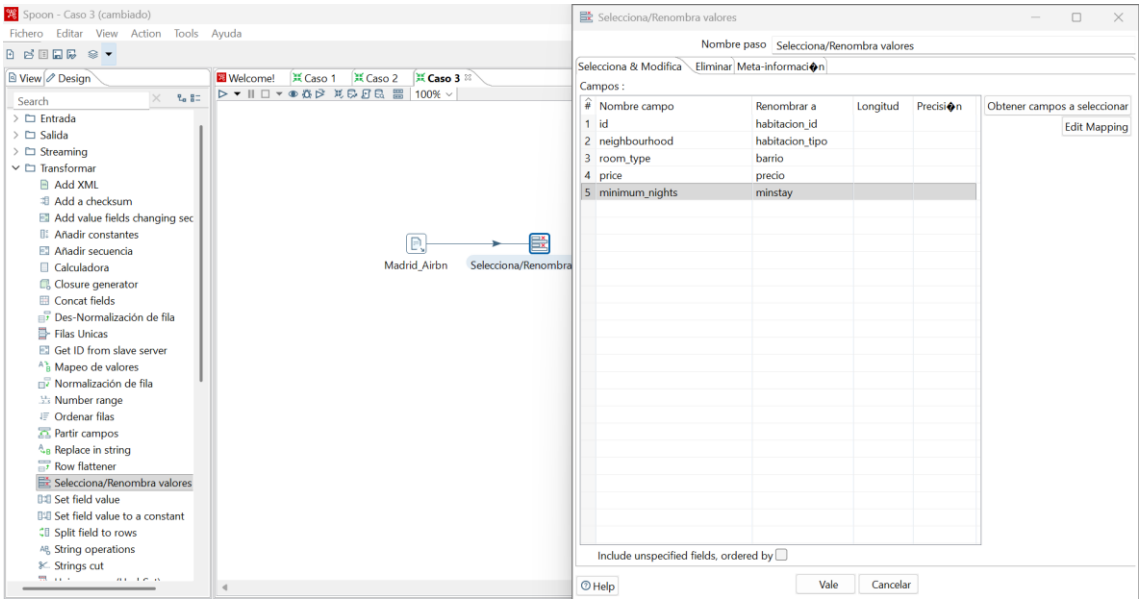


Selección y Renombre de Columnas (Select values)

Nos vamos a quedar con un subconjunto de las columnas y las vamos a renombrar. Para ello:

1. **Añadir Paso de Transformación:**
    - Arrastra **Select values** (Selecciona/Renombra valores) de la categoría **Transform** y conéctelo al paso anterior.
2. **Configurar Renombre y Selección:**
    - Mantenemos solo los campos necesarios para el análisis y la salida JSON, renombrándolos:

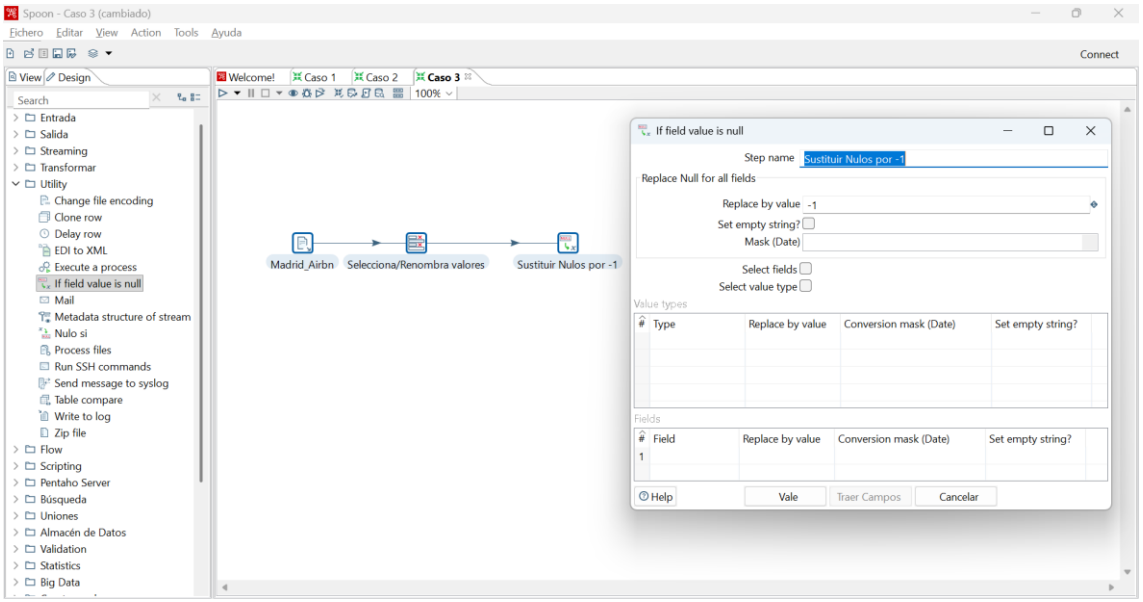
Campo Original	Campo de Salida
id	**habitacion_id**
room_type	**habitacion_tipo**
neighbourhood	**barrio**
price	**precio**
minimum_nights	**minstay**



Gestión de Valores Nulos (Limpieza de Datos)

Aunque el price no suele ser nulo, es buena práctica manejar los posibles nulos en campos numéricos clave como minstay (que usaremos en el filtro).

1. **Añadir Paso de Utilidad/Utility:**
    - Arrastra el paso **If value is null** (Si el valor es nulo) de la categoría **Utility**.
2. **Configurar Sustitución:**
    - Especifica que los valores nulos para los campos clave (precio y minstay) se reemplacen por **-1**.



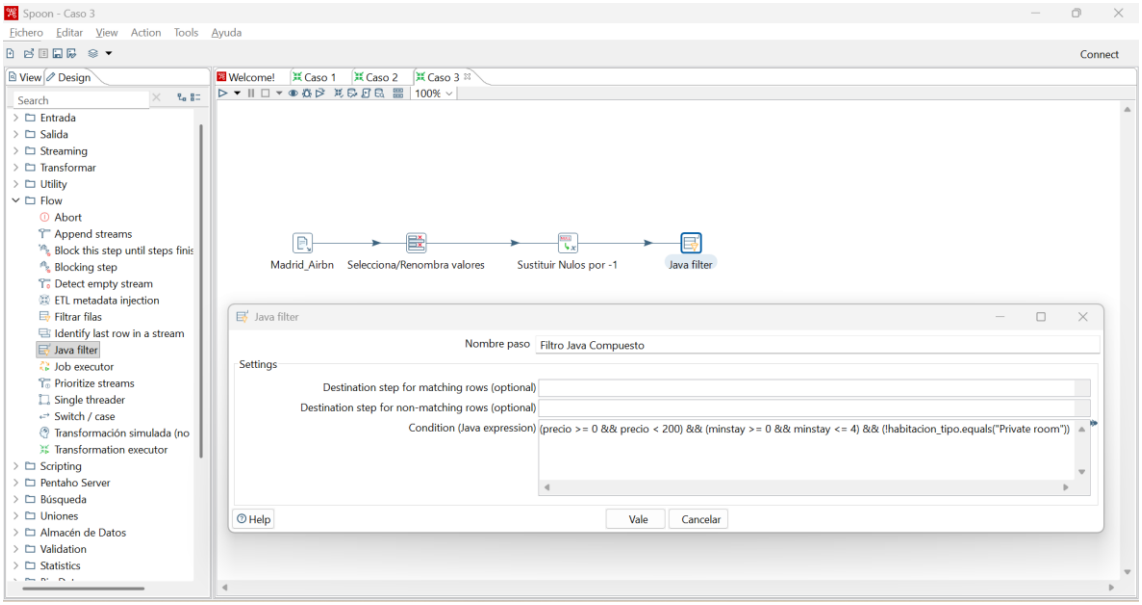
Filtrado Avanzado (Java Filter)

Ahora, adaptamos el filtro compuesto para usar los campos disponibles.

Nuevo Objetivo de Filtrado: Quedarse con los alojamientos cuyo **precio sea menor a 200€** Y la **estancia mínima sea de 4 noches o menos** Y el **tipo de habitación no sea un Private room**.

1. **Añadir Paso de Flujo:**
    - Arrastra el paso **Java Filter** de la categoría **Flow** y conéctalo al paso anterior.

2. **Configurar la Condición (Sintaxis Java):**
    - **Condición Ajustada:**
    - **Explicación:**
        - `(precio >= 0 && precio < 200)`: Asegura un precio válido y menor a 200 (descartando nulos/-1).
        - `(minstay >= 0 && minstay <= 4)`: Estancia mínima válida y corta (4 noches o menos).
        - `!habitacion_tipo.equals("Private room")`: Excluye las habitaciones privadas.



Generación de Salida JSON

1. **Añadir Paso de Salida:**
    - Arrastra el paso **JSON Output** (Salida JSON) de la categoría **Output** y conéctalo a la salida exitosa del **Java Filter**.
2. **Configurar el JSON Output:**
    - **Pestaña File:**
        - **Filename:** resultado_airbnb_filtrado_madrid.json
        - **Json bloc name:** datos
        - **Nr rows in a bloc:** 0
    - **Pestaña Fields:**
        - Pulsa **Get Fields** (Obtener Campos) para asegurar que todos los campos seleccionados (habitacion_id, habitacion_tipo, barrio, precio, minstay) se incluyan en el JSON final.

Ejecución y Verificación

1. **Ejecutar la Transformación** (Botón Run o **F9**).

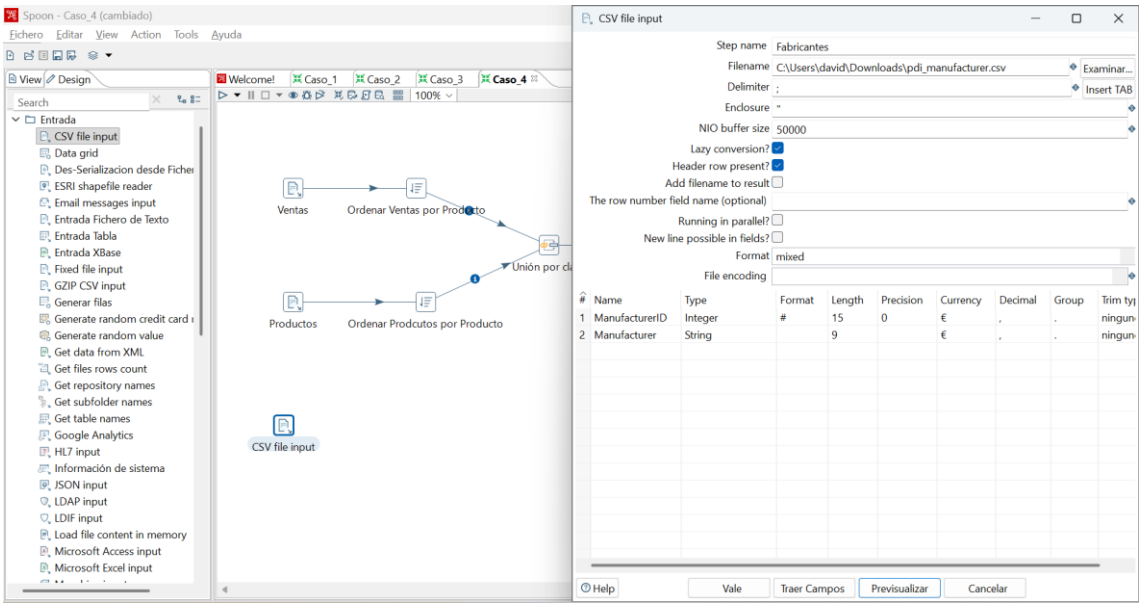
2. Verifica que el archivo resultado_airbnb_filtrado_madrid.json para confirmar que contiene solo los alojamientos que cumplen las condiciones de precio, estancia mínima y tipo de habitación.

Caso de Uso 4 - Informe fabricantes en S3

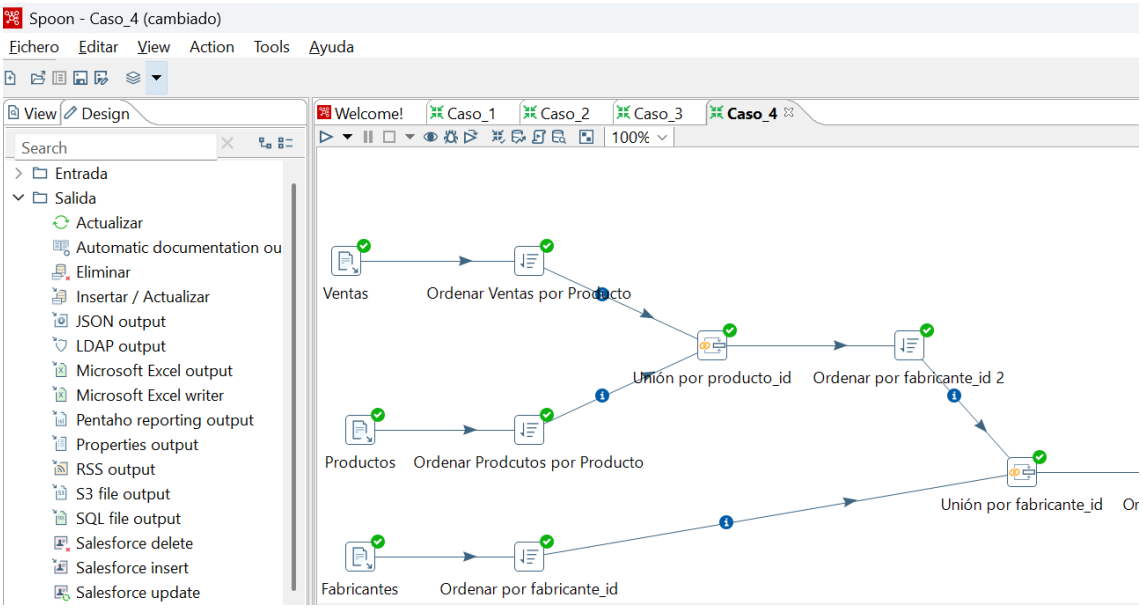
A partir del caso 2 que hemos realizado previamente, vamos a generar un informe sobre ventas de los fabricantes, y a persistir el informe en S3.

Unir fabricantes

Así pues, vamos a abrir el caso de uso 2, y a partir de ahí, vamos a fusionar los datos con los de fabricantes para obtener el nombre de éstos.



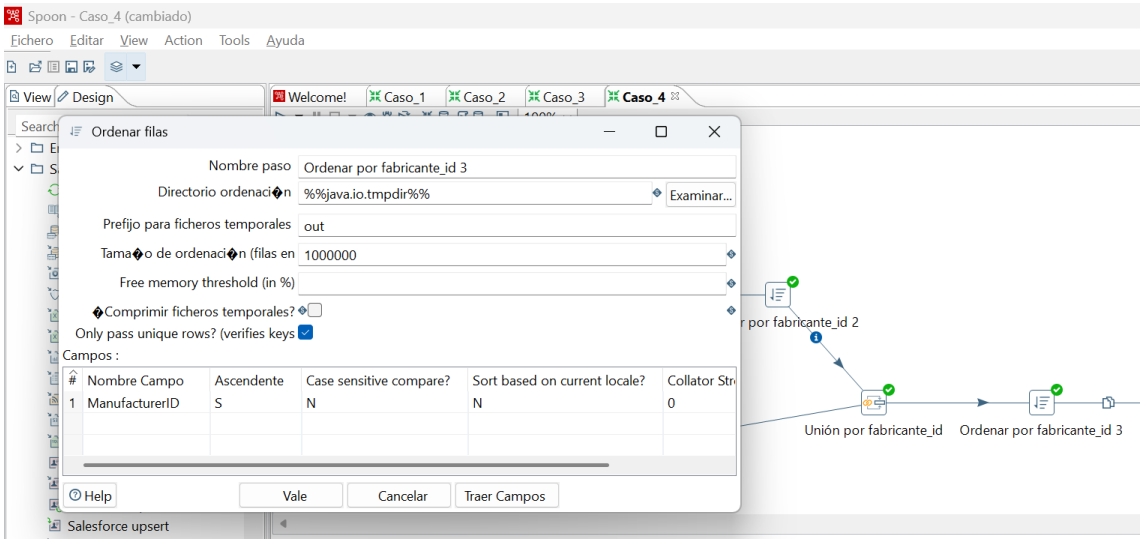
Para ello, creamos una nueva lectura de CSV, ordenación y posterior *Merge*. Cabe destacar que antes de hacer el segundo *join*, debemos ordenar ambas entradas por la clave de unión (ManufacturerID):



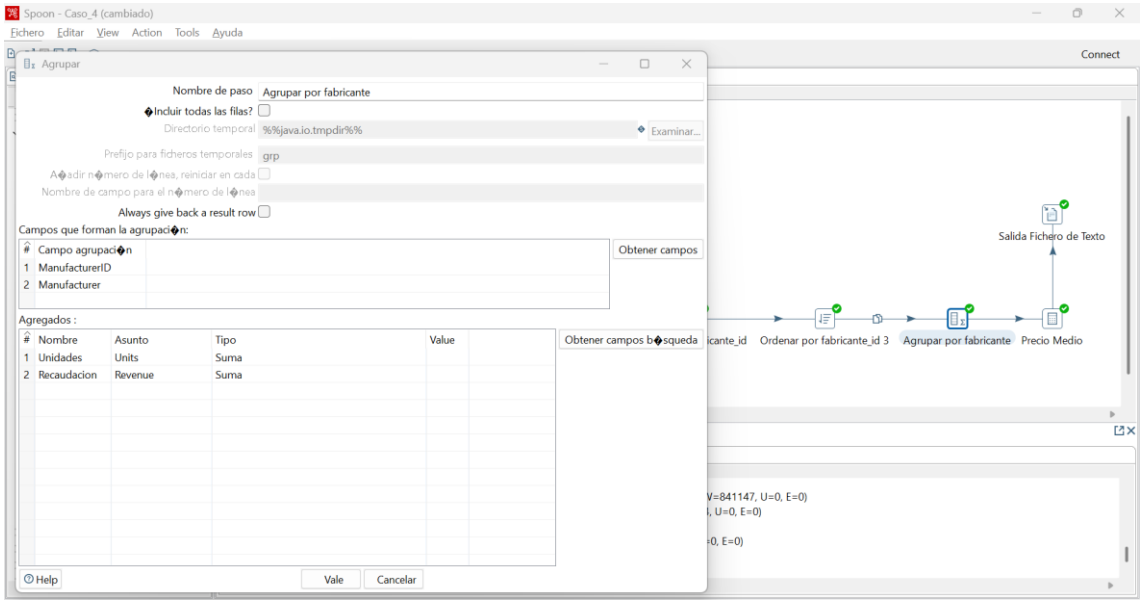
Caso de Uso 4 - Merge fabricantes

Aplicar fórmulas

Tras unir los datos de las ventas con los fabricantes, para poder preparar el informe mediante group-by, del mismo modo que antes, necesitamos ordenar los datos (en este caso por ManufacturerID).

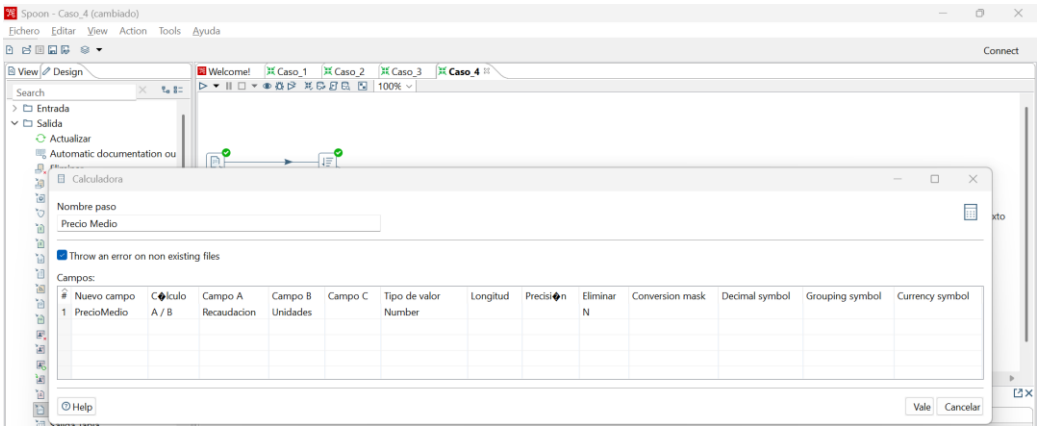


En el informe queremos mostrar para cada fabricante, además de su código y nombre (campos de agrupación), queremos obtener el total de unidades vendidad y la recaudación total de dicho fabricante. Para ello creamos una nueva agrupación.



Caso de Uso 4 - Agrupamos por fabricante

Además, queremos que nos muestre un nuevo campo que muestre el precio medio obtenido de dividir la recaudación obtenida entre el total de unidades. Para ello, dentro de la categoría *Transform*, elegimos el paso *Calculator*. Una vez abierto el diálogo para editar el paso, si desplegamos la columna *Calculation* puedes observar todas las posibles operaciones y transformaciones (tanto númericas, como de campos de texto e incluso fecha) que podemos realizar. En nuestro caso, sólo necesitamos la división entre *A* y *B*:



Caso de Uso 4 - Fórmula entre dos campos

Guardar en S3

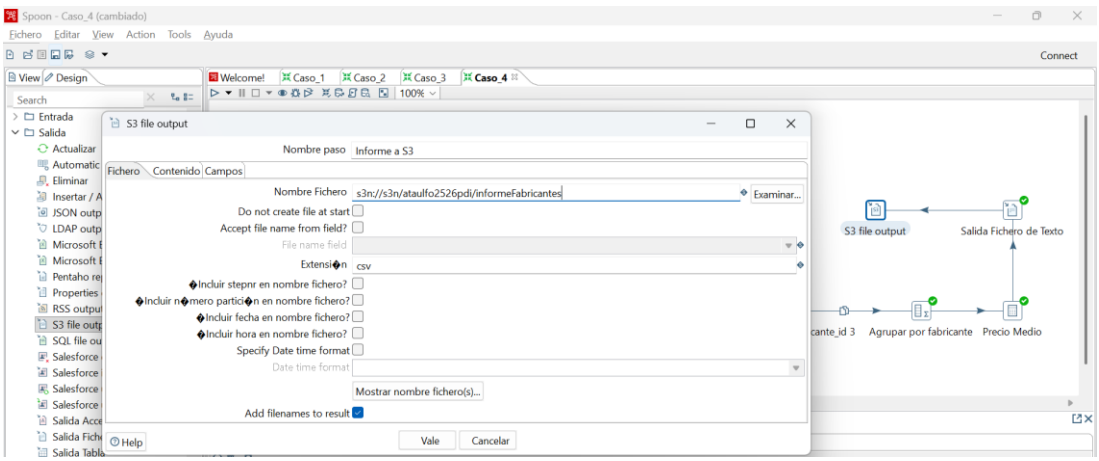
Para almacenar los datos en S3, primero crearemos un *bucket* público (en mi caso lo he denominado ataulfo2526pdi) y para facilitar el trabajo, vamos a crear una política que permita todas las operaciones:

```
{
    "Version": "2012-10-17",
    "Id": "Policy1635323698048",
    "Statement": [
        {
            "Sid": "Stmt1635323796449",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::ataulfo2526pdi"
        }
    ]
}
```

El siguiente paso es configurar las credenciales de acceso en nuestro sistema. Recuerda que lo haremos mediante las variables de entorno `(AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY y AWS_SESSION_TOKEN)` o almacenando las credenciales en el archivo `~/.aws/credentials`.

Finalmente, añadimos el paso *S3 file output* indicando:

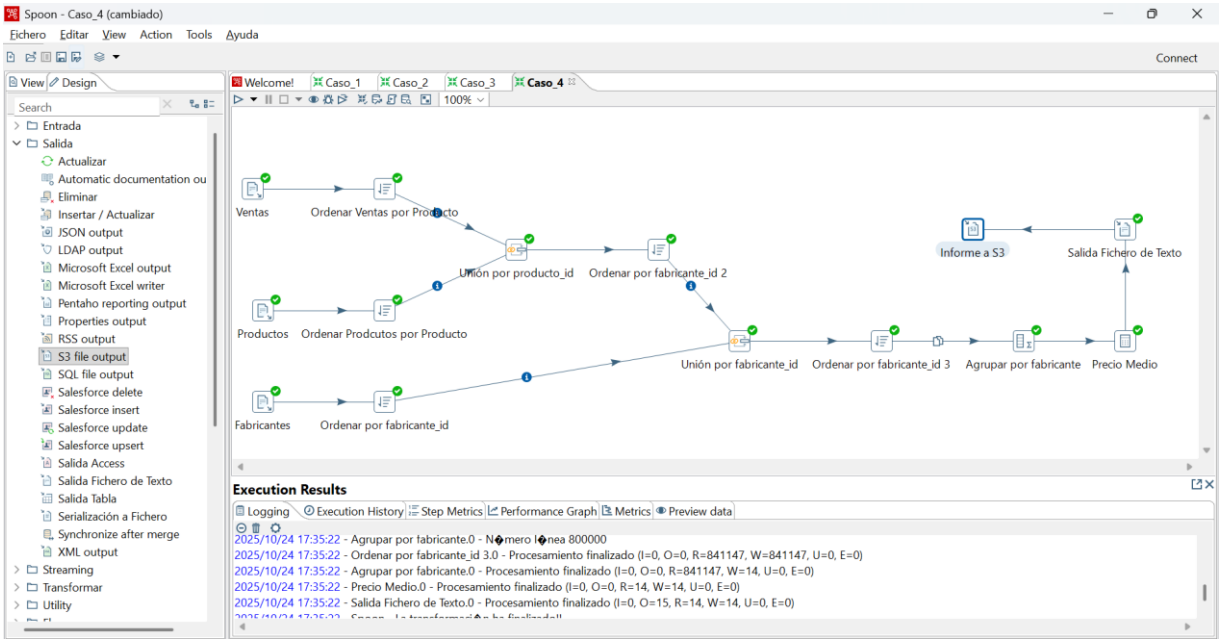
- La ruta del archivo, la cual se indica de forma similar a `s3n://s3n/bucket/nombreArchivo`. En nuestro caso, hemos utilizado el nombre `s3n://s3n/ataulfo2526pdi/informeFabricantes`.
- La extensión: nosotros hemos elegido el formato *csv*



Caso de Uso 4 - Salida a S3

Una vez todo unido, tras ejecutarlo podremos acceder a nuestra consola de AWS y comprobar cómo ha aparecido el fichero.

En resumen, en este caso de uso hemos generado la siguiente transformación:

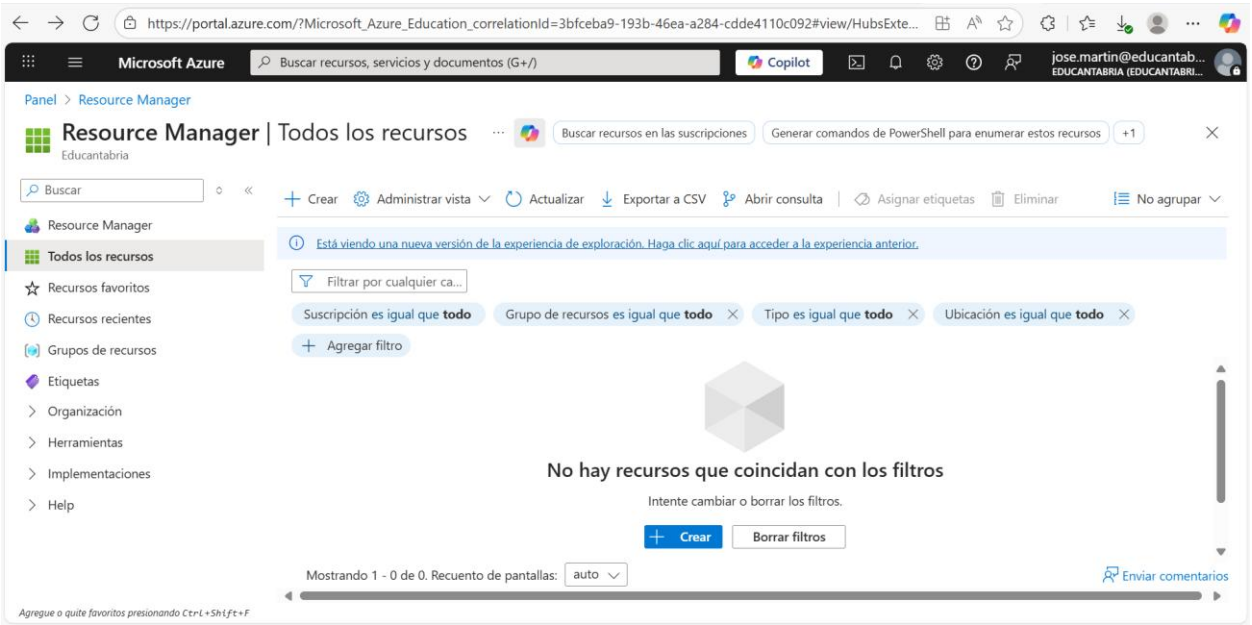


Caso de Uso 4 - Resultado final

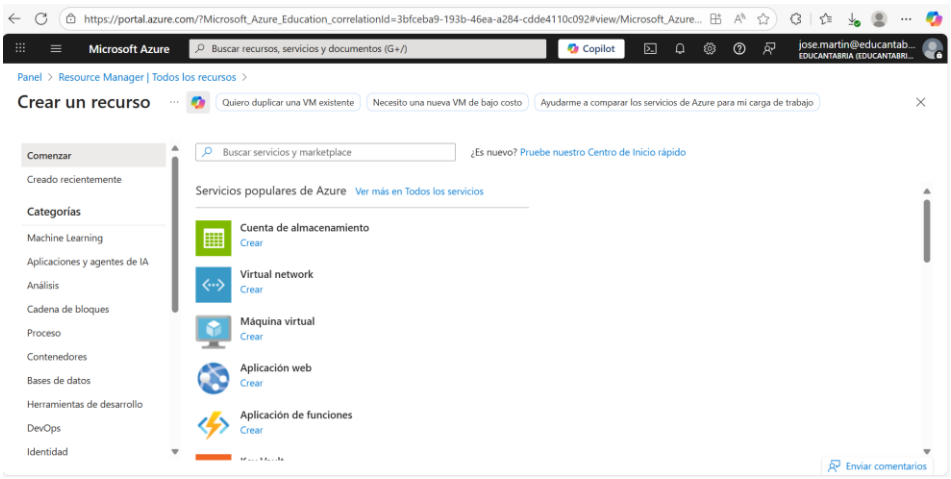
Guardar en Azure (Cuenta de almacenamiento/contenedor)

Dentro de nuestra cuenta de Azure Students, tenemos que crear una cuenta de almacenamiento.

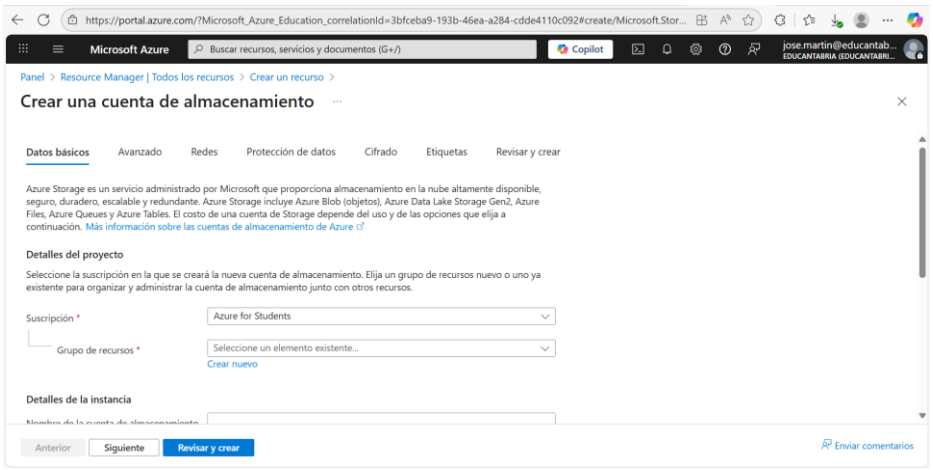
1º Vamos a la vista de Resource Maneger/Todos los Recursos



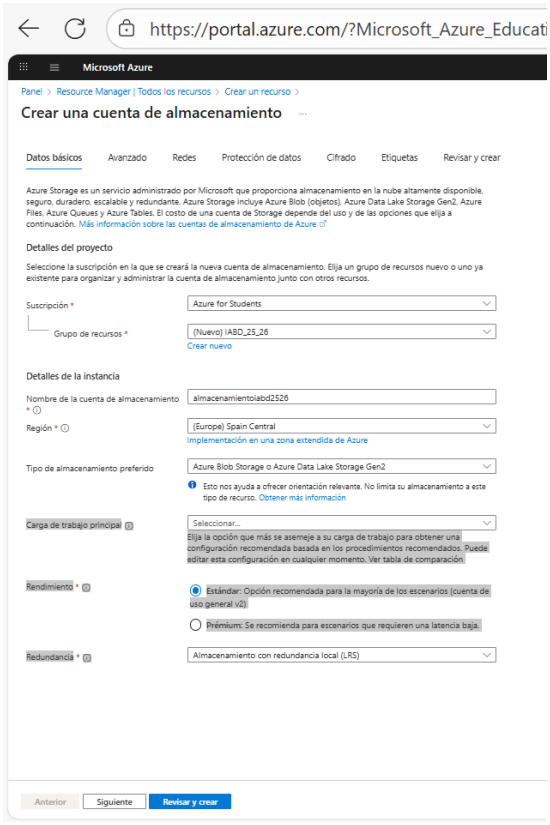
2º Hacemos click en “+ Crear”



3º Hacemos click en “Crear Cuenta de almacenamiento”:

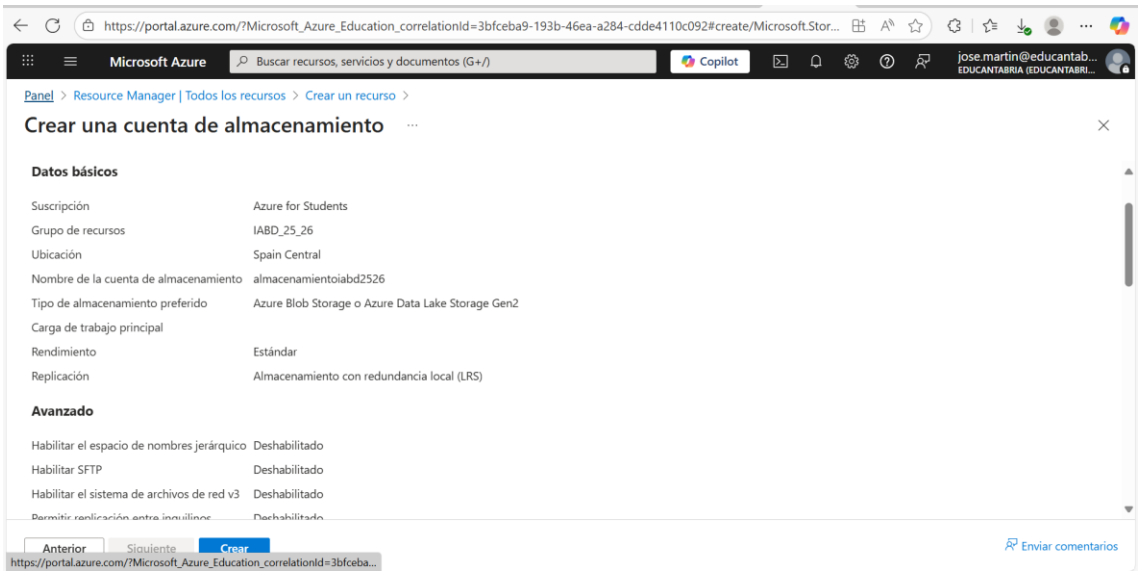


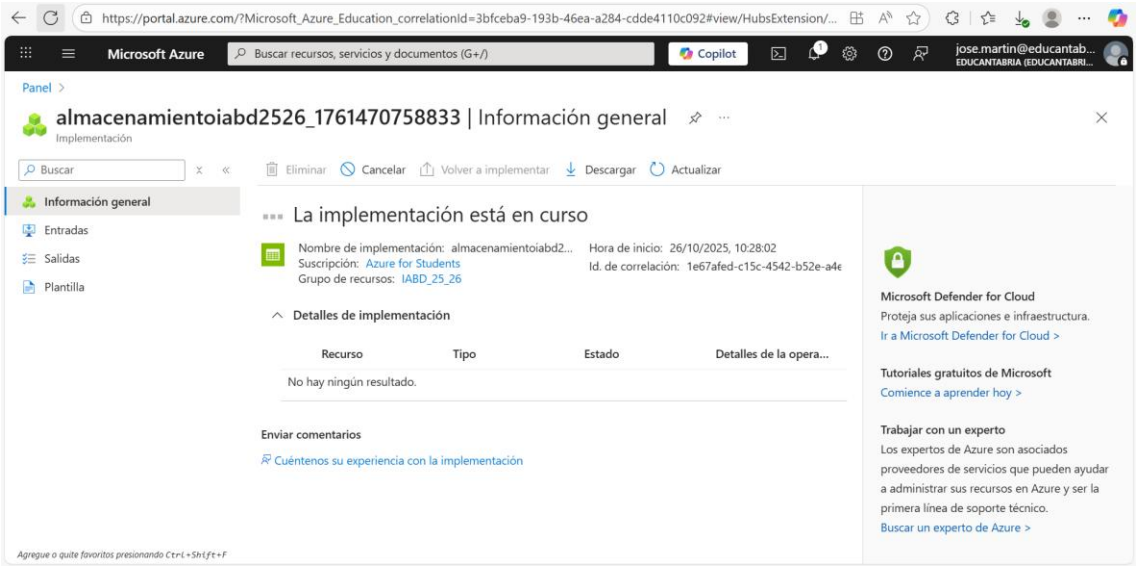
4º Rellenamos campos mínimos para crear la Cuenta de Almacenamiento

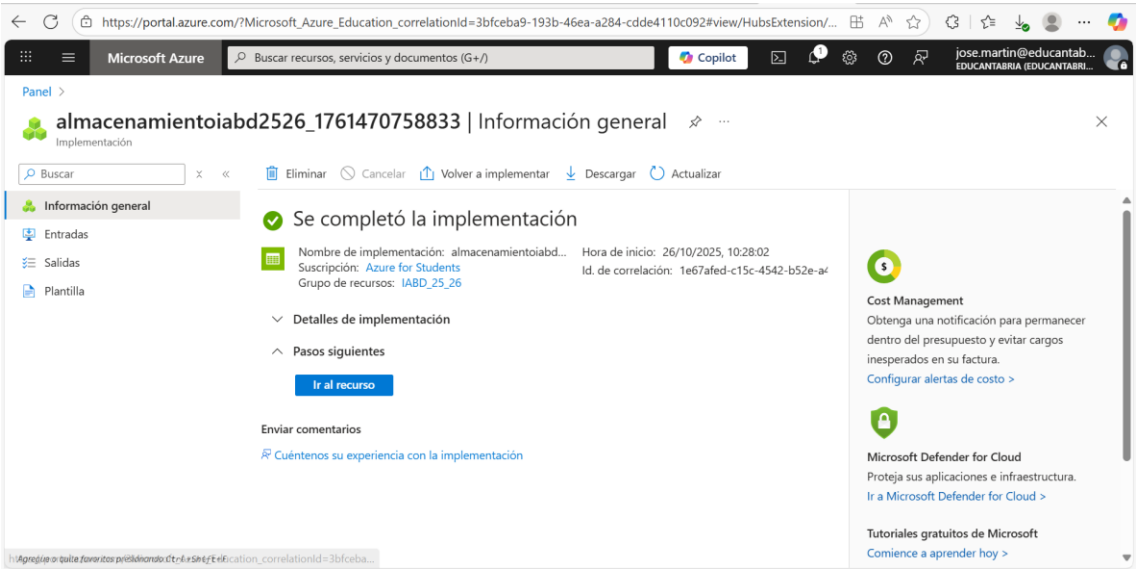


Observa que hemos tenido que crear un “Grupo de recursos”, donde alijar nuestra cuenta de almacenamiento. El resto de los campos a configurar lo dejamos por defecto.

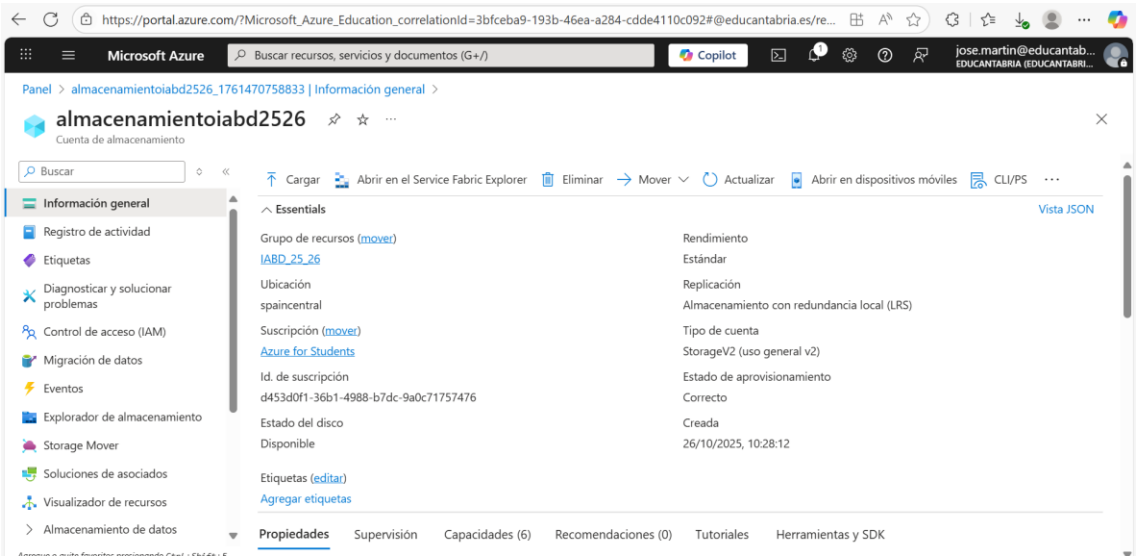
5º. Hacemos click en “Revisar y crear”.





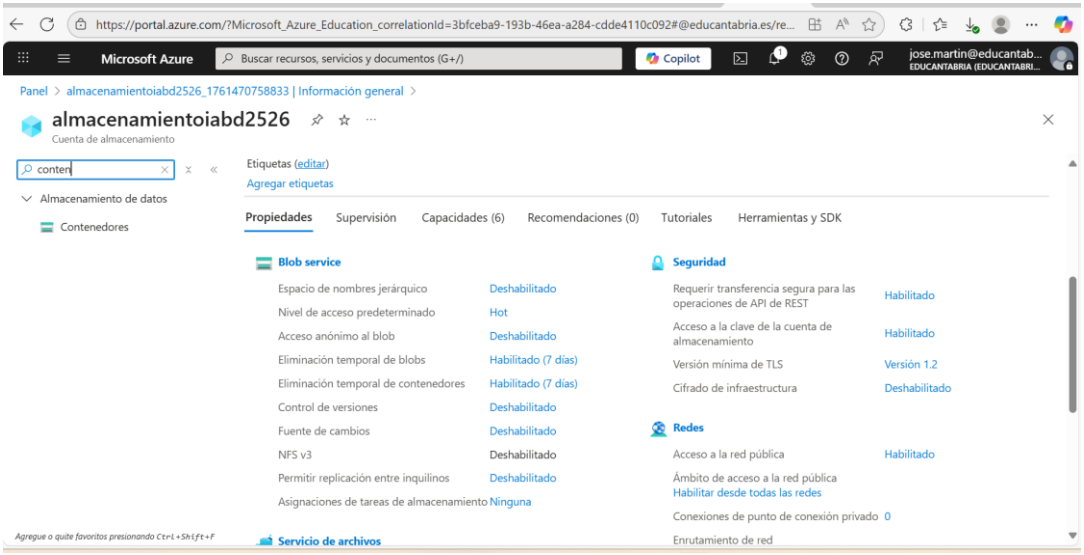


6º. Vamos al recurso recién creado.

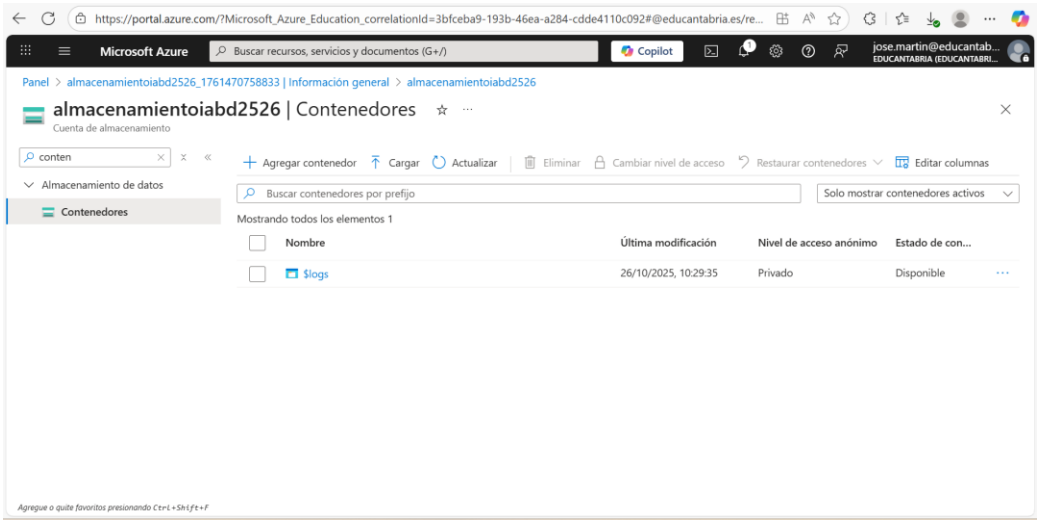


7º. Ahora que hemos creado la cuenta de almacenamiento: “almacenamientoiabd2526”, tenemos que crear un **contenedor**, dentro de dicha cuenta de almacenamiento.

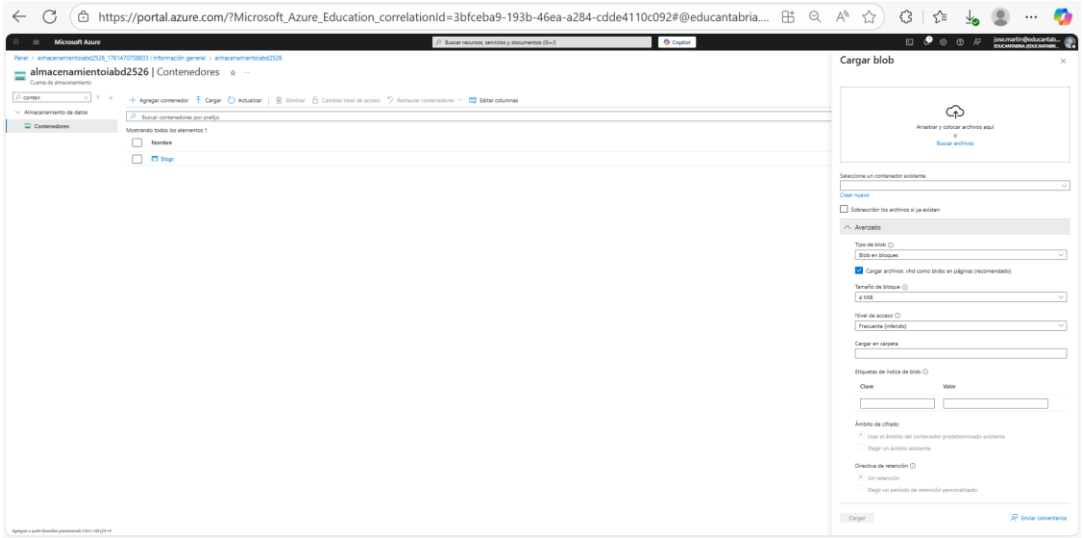
Para ello desde el buscador, introducimos el texto “contenedor”



Hacemos click en “Contenedores”



Desde esta ventana podemos Agregar un contenedor, o Cargar “elementos” desde nuestro equipo

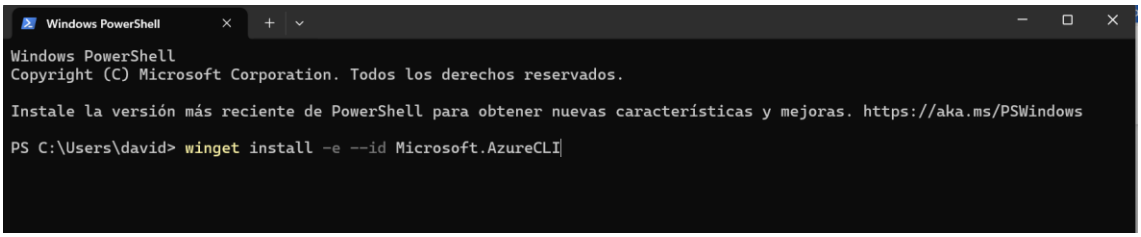


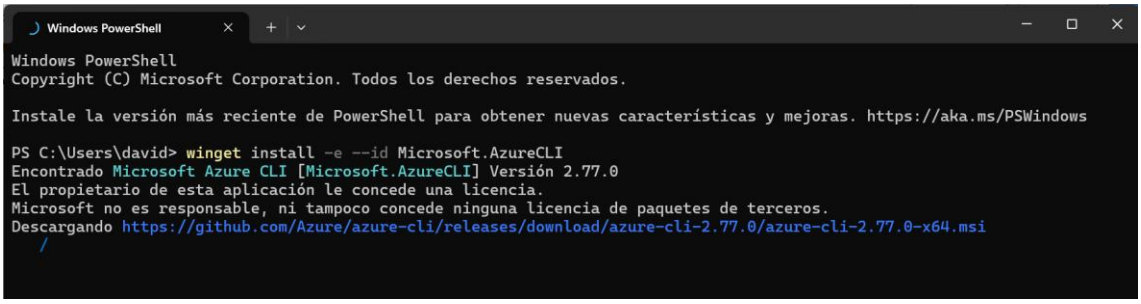
1) Instalar Azure CLI en Windows 11

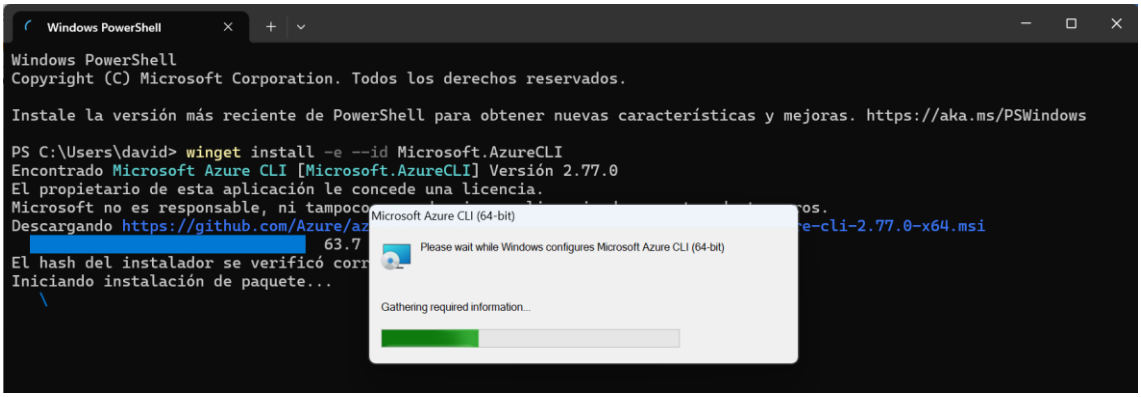
Opción A (recomendada): con winget

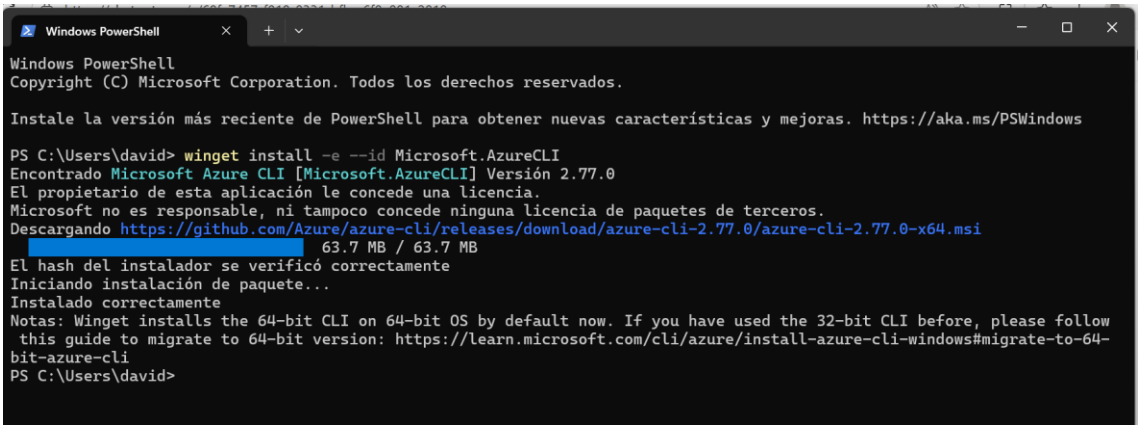
1. Abre **PowerShell** (mejor **como Administrador**).
2. Ejecuta:

```
winget install -e --id Microsoft.AzureCLI
```





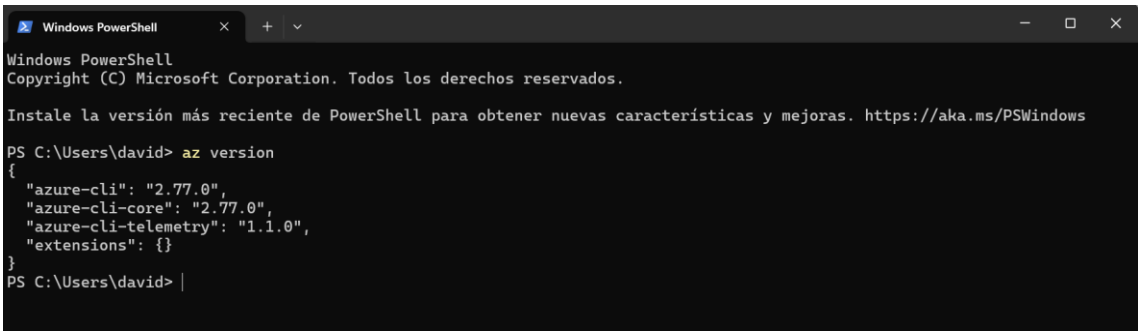




3. Cierra y **reabre** PowerShell (para refrescar el PATH).

4. Verifica:

```
az versión
```



Actualizar más adelante

```
winget upgrade -e --id Microsoft.AzureCLI
```

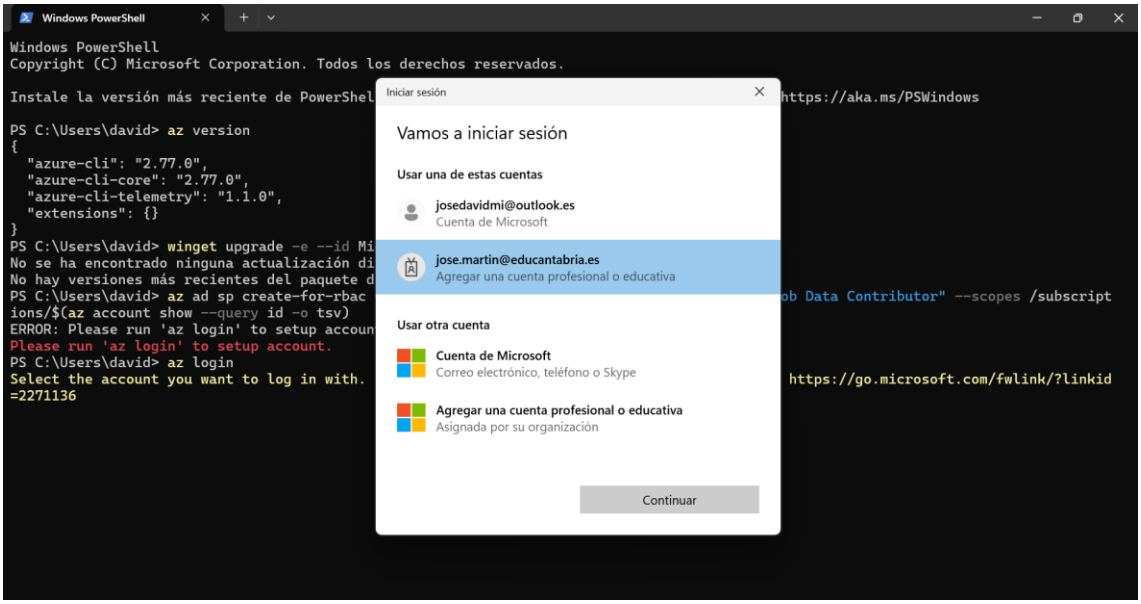
Si prefieres MSI: descarga desde la página oficial de Azure CLI para Windows, instálalo y reinicia PowerShell. (Con winget es más rápido.)

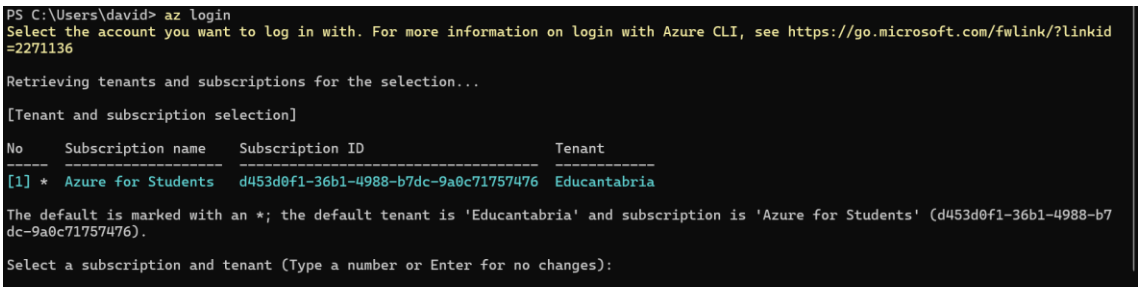
Para poder ejecutar un script en PowerShell o Bash, necesitamos estar “logeados” previamente por defecto en nuestro equipo local:

Opción Recomendada (Profesional): Service Principal + az login --service-principal

Sirve cuando vas a automatizar, ejecutar desde PowerShell, Bash, GitHub Actions, etc.

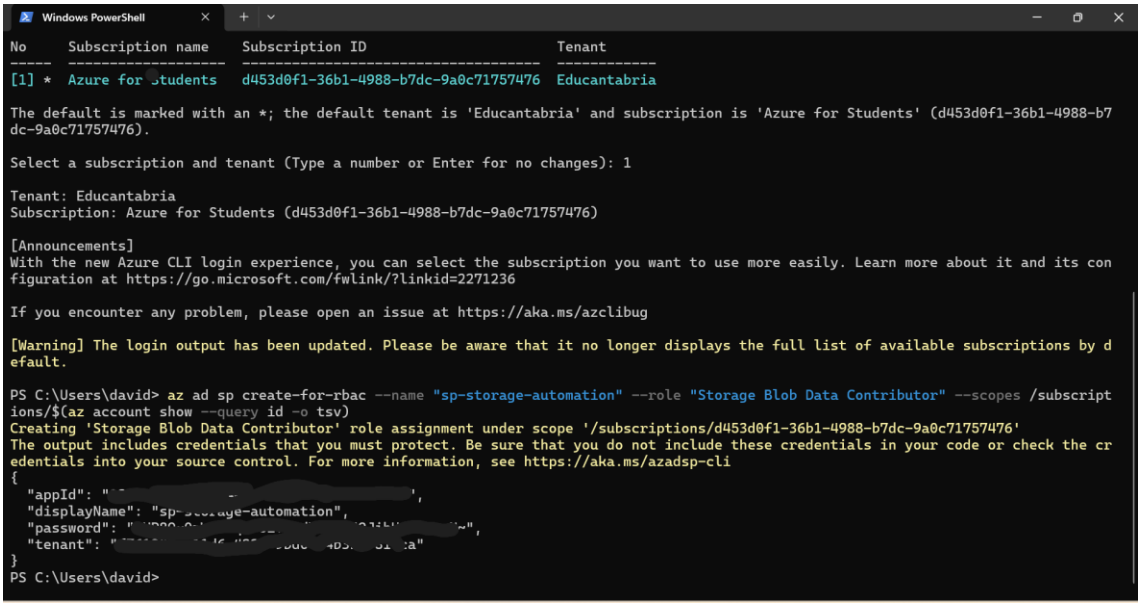
1) Crear el Service Principal (solo 1 vez)





Ejecuta esto **una sola vez** en PowerShell (sí requiere sesión interactiva la primera vez):

```
az ad sp create-for-rbac --name "sp-storage-automation" --role "Storage Blob Data Contributor" --
scopes /subscriptions/$(az account show --query id -o tsv)
```



Esto devolverá algo así:

```
{
  "appId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "displayName": "sp-storage-automation",
  "password": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "tenant": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
}
{
  "appId": "13a3374f-74f2-4e2d-96ef-e2bf38edc6b6",
  "displayName": "sp-storage-automation",
  "password": "wUP8Q~0rWa6mq096LT3zdTXuVK49JihUe8Z6-cN~",
  "tenant": "f761348e-11d6-433c-9bd0-84b3b8781bca"
}
```

GUARDA esos valores:

- appId → CLIENT_ID
- password → CLIENT_SECRET
- tenant → TENANT_ID

2) Logear automáticamente en tu script sin interacción

En tu script (.sh o PowerShell):

```
# Variables del Service Principal
CLIENT_ID="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
CLIENT_SECRET="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
TENANT_ID="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"

# Login no interactivo
az login --service-principal \
  --username $CLIENT_ID \
  --password $CLIENT_SECRET \
  --tenant $TENANT_ID
```

3) Crear contenedor y subir archivo en modo Shell (Linux/Mac/WSL/GitBash)

Script 1 · Bash (WSL/Git Bash/Ubuntu) — 100% no interactivo

Guárdalo como blob_upload.sh:

```
#!/usr/bin/env bash
set -euo pipefail

# ======= CONFIG =======
# Credenciales del Service Principal
CLIENT_ID="${AZ_CLIENT_ID:-}"      # o pega aquí el GUID si no usas variables de entorno
CLIENT_SECRET="${AZ_CLIENT_SECRET:-}"
TENANT_ID="${AZ_TENANT_ID:-}"

# Recurso
STORAGE="almacenamientoiabd2526"   # <-- cambia por tu storage
CONTAINER="practica1"              # <-- contenedor destino
FILE_LOCAL="./datos.csv"           # <-- ruta local
FILE_DEST="datos.csv"              # <-- nombre del blob

# ======= LOGIN NO INTERACTIVO =======
if [[ -z "$CLIENT_ID" || -z "$CLIENT_SECRET" || -z "$TENANT_ID" ]]; then
  echo "Faltan credenciales (AZ_CLIENT_ID / AZ_CLIENT_SECRET / AZ_TENANT_ID)."
  exit 1
fi

az login --service-principal \
  --username "$CLIENT_ID" \
  --password "$CLIENT_SECRET" \
  --tenant "$TENANT_ID" 1>/dev/null

# ======= CREAR CONTENEDOR (idempotente) =======
az storage container create \
  --account-name "$STORAGE" \
  --name "$CONTAINER" \
  --auth-mode login \
  --only-show-errors 1>/dev/null

echo "Contenedor '$CONTAINER' verificado/creado."

# ======= SUBIR ARCHIVO =======
az storage blob upload \
  --account-name "$STORAGE" \
  --container-name "$CONTAINER" \
  --file "$FILE_LOCAL" \
  --name "$FILE_DEST" \
  --overwrite true \
  --auth-mode login

echo "Archivo '$FILE_LOCAL' subido como '$FILE_DEST' en '$CONTAINER'."
```

Usa variables de entorno para no dejar secretos en claro:

```
export AZ_CLIENT_ID="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
export AZ_CLIENT_SECRET="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
export AZ_TENANT_ID="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"

chmod +x blob_upload.sh
./blob_upload.sh
```

Script 2 · PowerShell (Windows 11) — 100% no interactivo

Guárdalo como BlobUpload.ps1:

```
Param(
  [Parameter(Mandatory=$true)] [string]$ClientId,
  [Parameter(Mandatory=$true)] [string]$ClientSecret,
  [Parameter(Mandatory=$true)] [string]$TenantId,
  [Parameter(Mandatory=$true)] [string]$StorageAccount,
  [Parameter(Mandatory=$true)] [string]$ContainerName,
  [Parameter(Mandatory=$true)] [string]$LocalFile,
```

```
  [Parameter(Mandatory=$true)] [string]$BlobName
)

$ErrorActionPreference = "Stop"

# Login no interactivo con Service Principal
az login --service-principal `
  --username $ClientId `
  --password $ClientSecret `
  --tenant $TenantId | Out-Null

# Crear contenedor si no existe (idempotente)
az storage container create `
  --account-name $StorageAccount `
  --name $ContainerName `
  --auth-mode login `
  --only-show-errors | Out-Null

Write-Host "Contenedor '$ContainerName' verificado/creado."

# Subir archivo
az storage blob upload `
  --account-name $StorageAccount `
  --container-name $ContainerName `
  --file $LocalFile `
  --name $BlobName `
  --overwrite true `
  --auth-mode login

Write-Host "Archivo '$LocalFile' subido como '$BlobName' en '$ContainerName'."
```

Ejemplo de uso:

```
.\BlobUpload.ps1 `
  -ClientId "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" `
  -ClientSecret "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" `
  -TenantId "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" `
  -StorageAccount "almacenamientoiabd2526" `
  -ContainerName "practica1" `
  -LocalFile "C:\Users\TuUsuario\Downloads\datos.csv" `
  -BlobName "datos.csv"
```

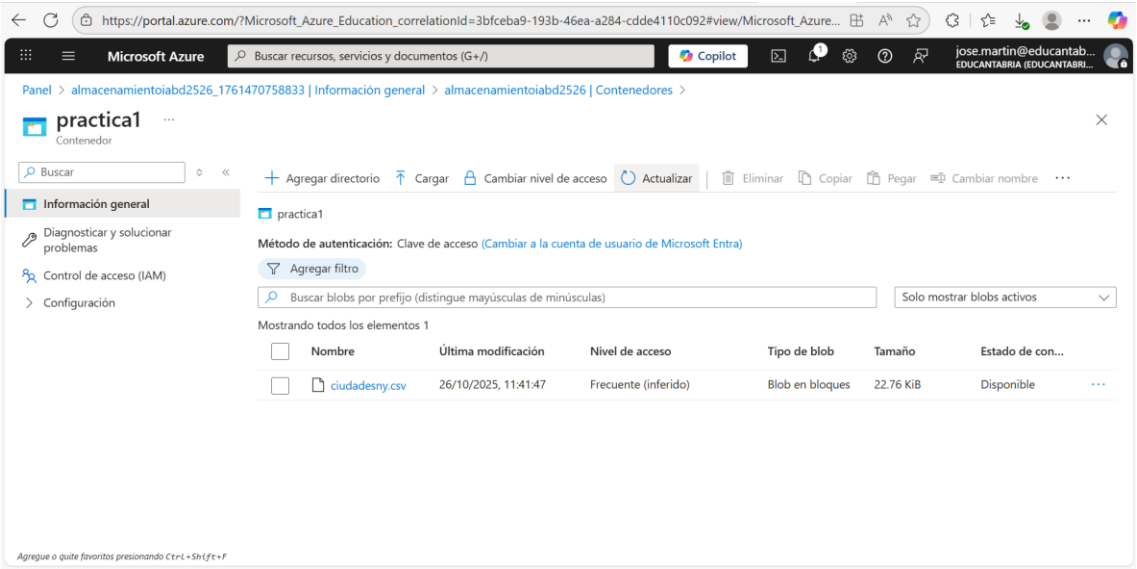
Alternativa sin SP: usar SAS (no requiere az login)

Si prefieres no crear un Service Principal:

```
SAS='?sv=2023-08-03&ss=b&srt=sco&sp=rlacup&se=2025-12-31T23:59Z&st=2025-01-
01T00:00Z&spr=https&sig=XXXX'

# Crear contenedor (necesita permisos 'c' en SAS si es a nivel de cuenta)
az storage container create \
  --account-name almacenamientoiabd2526 \
  --name practica1 \
  --sas-token "$SAS"

# Subir
az storage blob upload \
  --account-name almacenamientoiabd2526 \
  --container-name practica1 \
  --file ./datos.csv \
  --name datos.csv \
  --overwrite true \
  --sas-token "$SAS"
```



Notas importantes

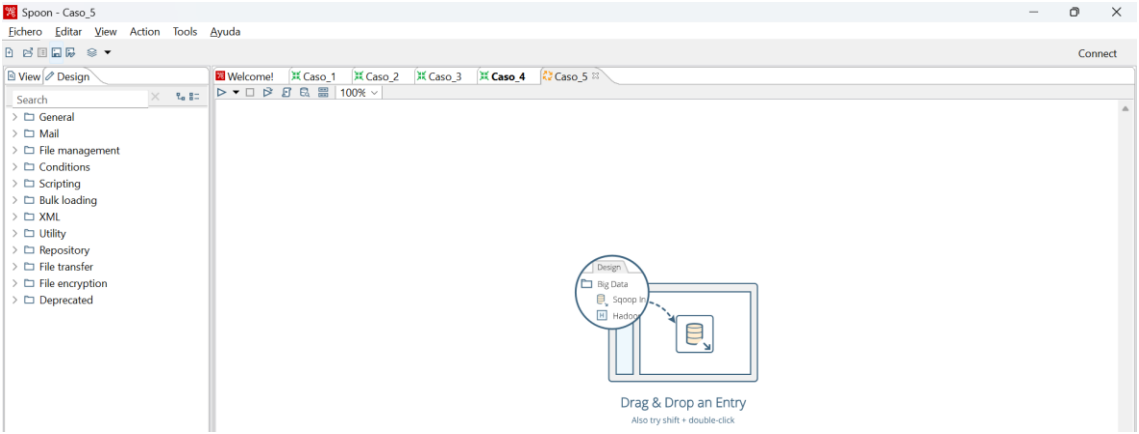
- **Permisos RBAC**: el Service Principal debe tener **Storage Blob Data Contributor** (o superior) **en la cuenta de almacenamiento** (o al menos en el contenedor).
- **Políticas/Regiones**: si tu suscripción tiene políticas de región, asegúrate de que el Storage y despliegues están en una región **permitida**.
- **Seguridad**: no subas ClientSecret a repositorios. Usa **variables de entorno** o **Azure Key Vault** en CI/CD.
- **Idempotencia**: az storage container create no falla si el contenedor ya existe.

Caso de Uso 5 - Jobs

En este caso crearemos un **Job/Trabajo** que coordine los **casos de uso 2 y 4** en un único proceso.

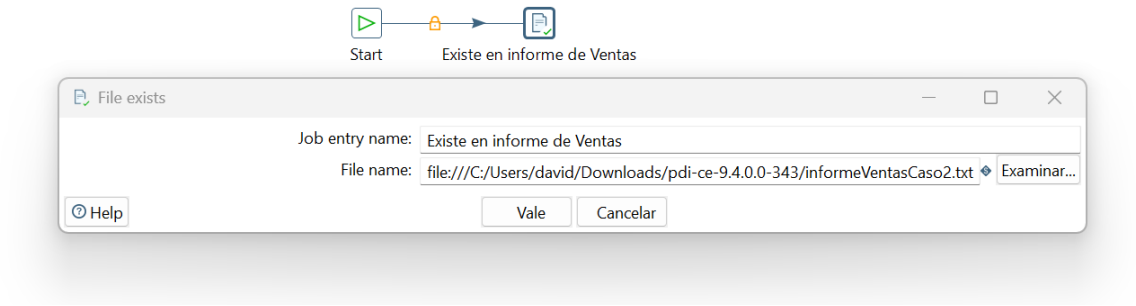
Crear el Job

1. Ve a **File → New → Job/Trabajo**.
2. En la pestaña **Design** verás los componentes disponibles para diseñar el Job.



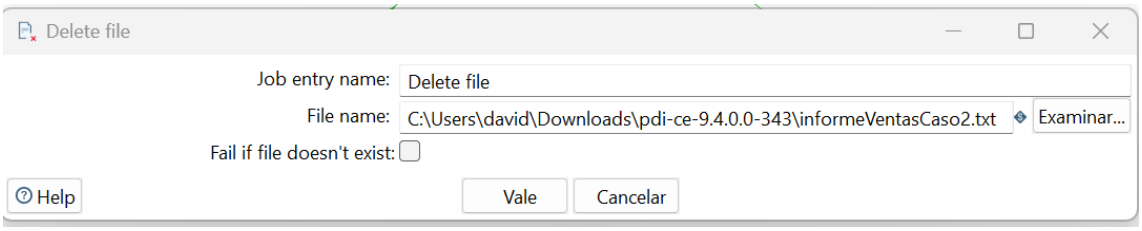
Comenzando el Job

1. **Start** (categoría *General*): inicia el flujo.
2. **File exists** (categoría *Conditions*): comprueba en el directorio donde guardamos el resultado del **caso de uso 2** (informe de ventas).

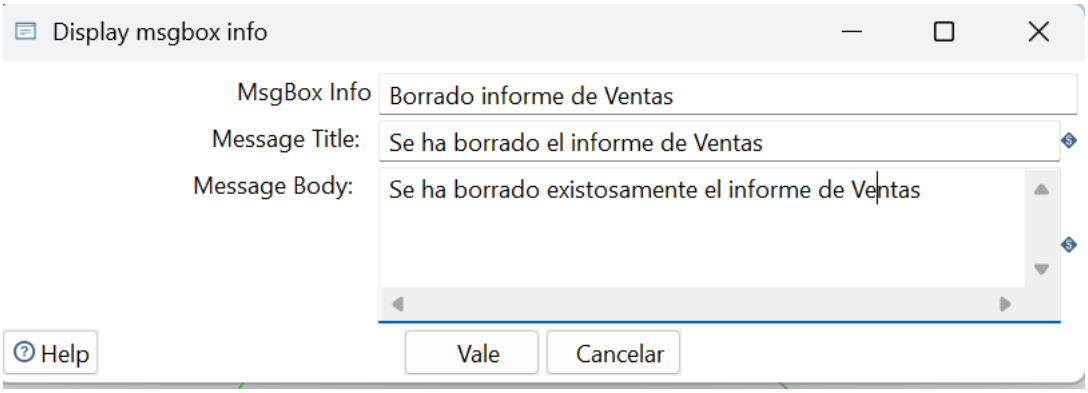


Caso de Uso 5 - Inicio

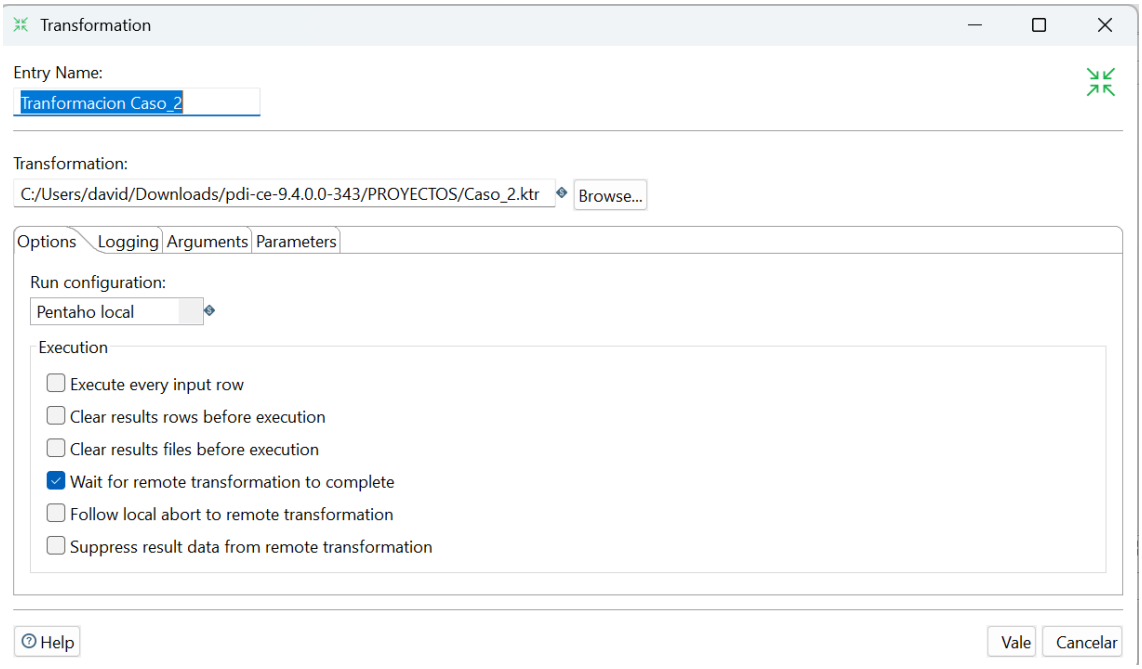
3. **Delete Files**: si el fichero existe, lo borra.



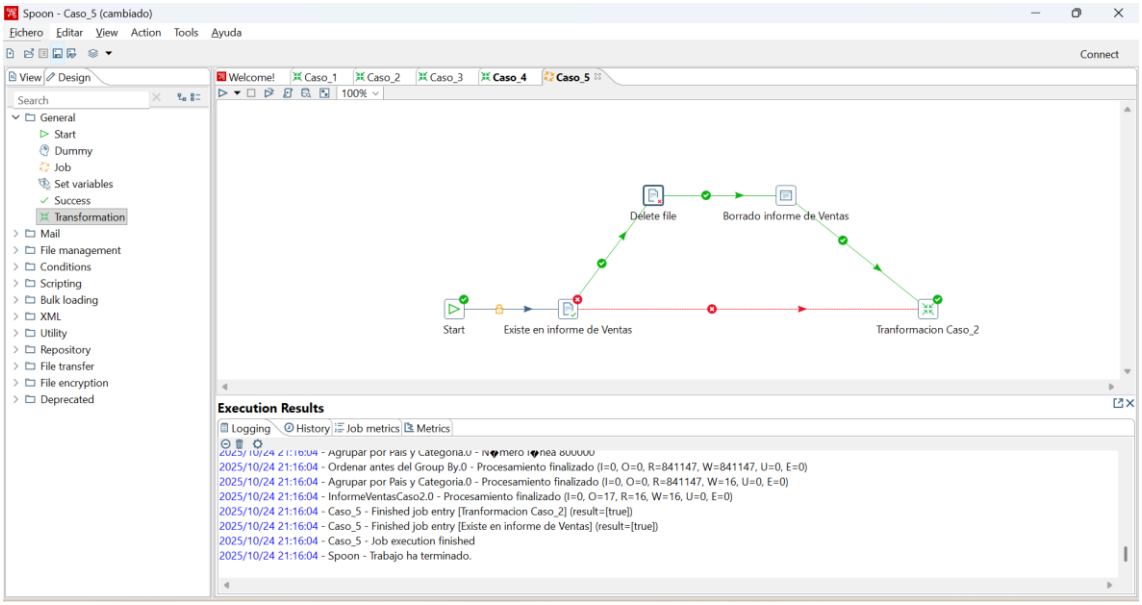
4. **Display msgbox info** (categoría *Utility*): muestra un aviso indicando que se ha borrado el fichero de ventas existente.



5. **Transformation**: incorpora la **transformación del caso de uso 2**.



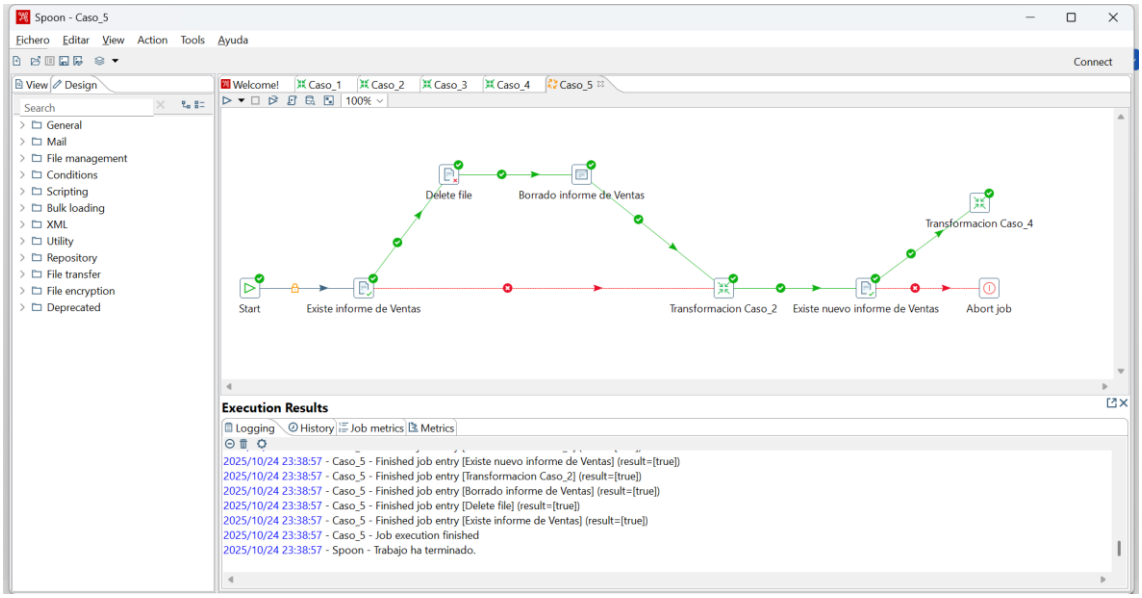
6. Conecta la salida de **Display msgbox info** y la de **File exists** con la **Transformation**.



Caso de Uso 5 - ejecución transformación caso de uso 2

Abortando un Job

Una vez generado el **informe de ventas**, queremos crear el **informe de fabricantes** (caso de uso 4). Antes, verificamos con **File exists** que la transformación anterior generó el fichero esperado. Si no existe, **Abort job** (categoría *Utility*) detiene el Job.



Caso de Uso 5 - Detención de un job

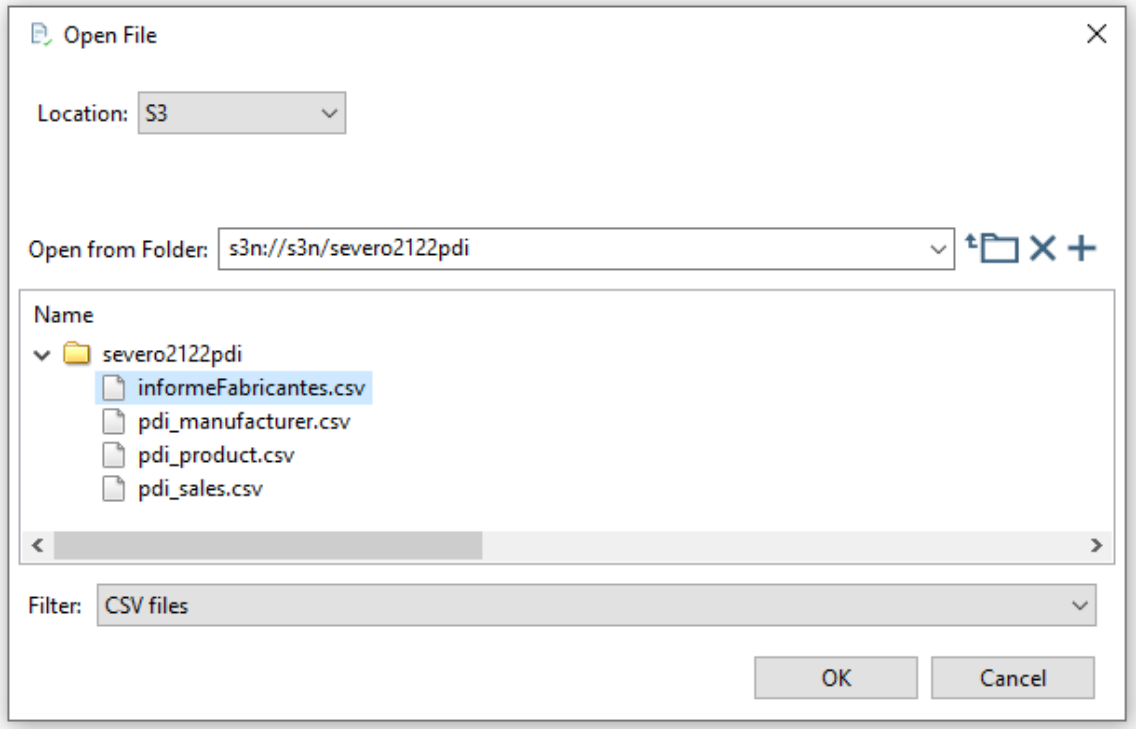
Comprobando S3

Repetimos los pasos iniciales, pero comprobando si el fichero existe en **S3**.

Si existe, lo **borramos** y **volvemos a ejecutar la transformación**.

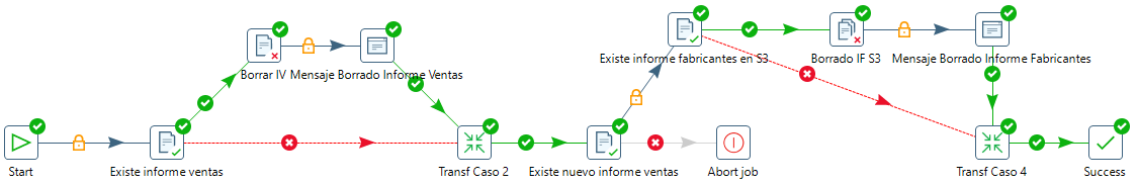
Para comprobar en S3 con el mismo componente:

- En la parte superior elige **S3**.
- Indica la **URL del bucket** donde están los archivos CSV.



Caso de Uso 5 - Comprobando S3

El resultado final debería ser similar al siguiente gráfico:



Caso de Uso 5 - Resultado final

Uso de Kitchen

Al igual que **Pan** ejecuta transformaciones desde terminal, **Kitchen** ejecuta Jobs.

Si guardaste el caso en caso5Job.kjb, se ejecuta así:

```
kitchen.sh /file=caso5Job.kjb
```

Tanto **Pan** como **Kitchen** aceptan parámetros:

- `level`: nivel de log, con la sintaxis `/level:<nivel>`

  Niveles posibles: `Nothing, Minimal, Error, Basic, Detailed, Debug, Rowlevel.`
- `param`: pasar parámetros al Job, uno por cada parámetro, con la sintaxis:

  `/param:"<nombre>=<valor>"`

Caso de Uso 6 — Interacción con bases de datos

Objetivo

Interactuar con una **base de datos relacional** (ejemplo con **Amazon RDS**; funciona igualmente en local).

1) Preparación del entorno

1.1 Crear la base de datos en RDS

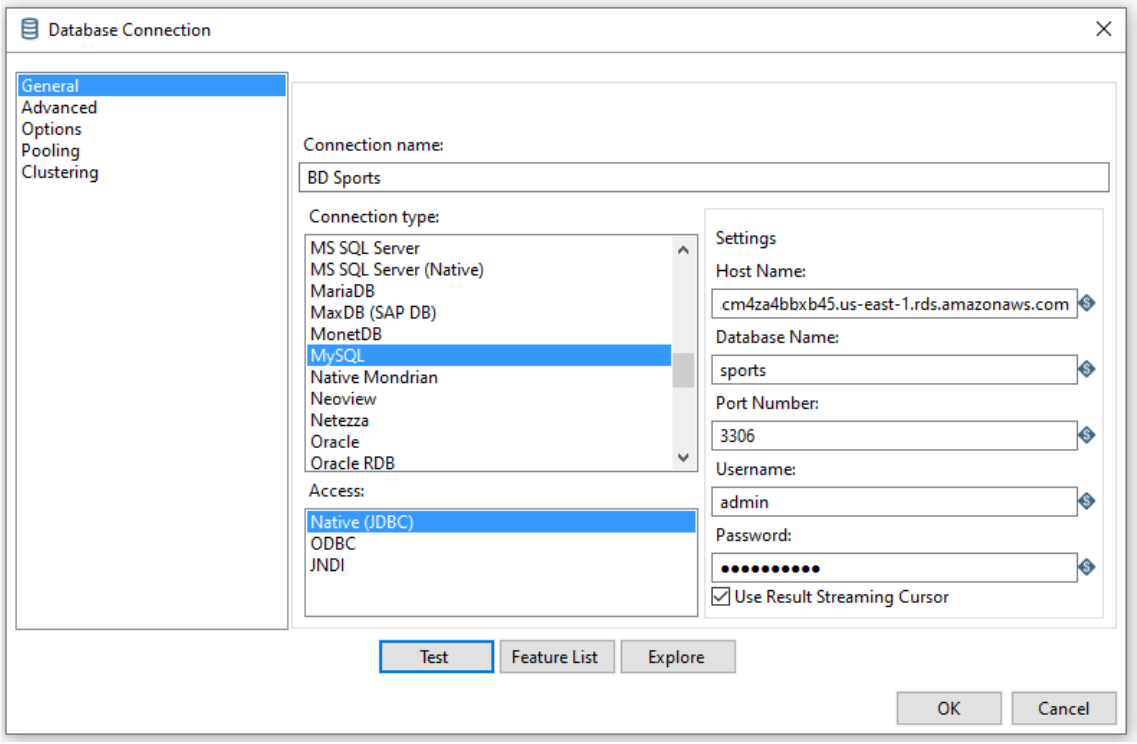
- Crear una base de datos llamada **sports** (puedes seguir el ejemplo de *Hola RDS*).
- **Descarga** los datos indicados para el ejemplo.

1.2 Instalar el driver JDBC de MySQL en Pentaho

- Por defecto solo viene el driver de PostgreSQL.
- **Descarga** el driver JDBC de MySQL (ojo: **la última versión del driver puede no funcionar**).
- Copia el **.jar** en la carpeta **lib** de tu instalación de Pentaho.

1.3 Configurar la conexión en Pentaho

- Menú **File → New → Database connection** y sigue el asistente para conectar con la BD creada en RDS.



Caso de Uso 6 - Conexión con RDS

2) Carga inicial de datos en la base de datos

Puedes cargar los datos con cualquiera de estas opciones (en el ejemplo se usa el usuario **admin/adminadmin**):

1. **Herramienta visual** (p. ej., DBeaver).
2. **Terminal**:

```
mysql -h sports.cm4za4bbxb45.us-east-1.rds.amazonaws.com -u admin -p sports < sportsdb_mysql.sql
```

4. **Instancia EC2** en la misma AZ que RDS (método más eficiente).
   Más info:
   https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/MySQL.Procedural.Importing.NonRDSRepl.html

3) Consultar datos (lectura con CSV + lookup en BD)

3.1 Archivo de entrada

Usaremos **injuries.txt** (CSV) con información de lesiones:

```
person_id;injury_type;injury_date
153;elbow;2007-07-09
186;fingers;2007-07-15
198;shoulder;2007-07-15
213;elbow;2007-07-16
```

- Leemos el fichero con un **CSV Input**.

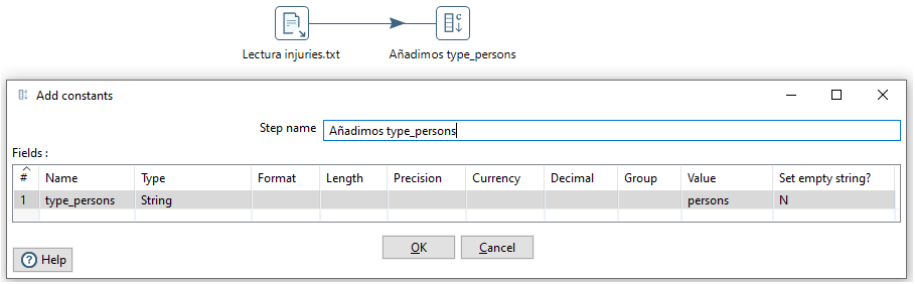
	id	language	entity_type	entity_id	full_name	first_name
145	145	en-US	teams	119	Dallas Stars	Dallas
146	146	en-US	teams	120	Los Angeles Kings	Los Angeles
147	147	en-US	teams	121	Phoenix Coyotes	Phoenix
148	148	en-US	teams	122	San Jose Sharks	San Jose
149	149	en-US	affiliations	27	Northeast	[NULL]
150	150	en-US	affiliations	28	American	[NULL]
151	151	en-US	persons	1	Grady Sizemore	Grady
152	152	en-US	persons	2	Casey Blake	Casey
153	153	en-US	persons	3	Victor Martinez	Victor
154	154	en-US	persons	4	Ryan Garko	Ryan
155	155	en-US	persons	5	Travis Hafner	Travis

Caso de Uso 6 - Datos de la tabla display_names

3.2 Preparar clave compuesta para la búsqueda

En la tabla **display_names** (BD) además del nombre existe el campo **entity_type** con valor **persons**. Para obtener el nombre al buscar en la tabla, añade al flujo la constante:

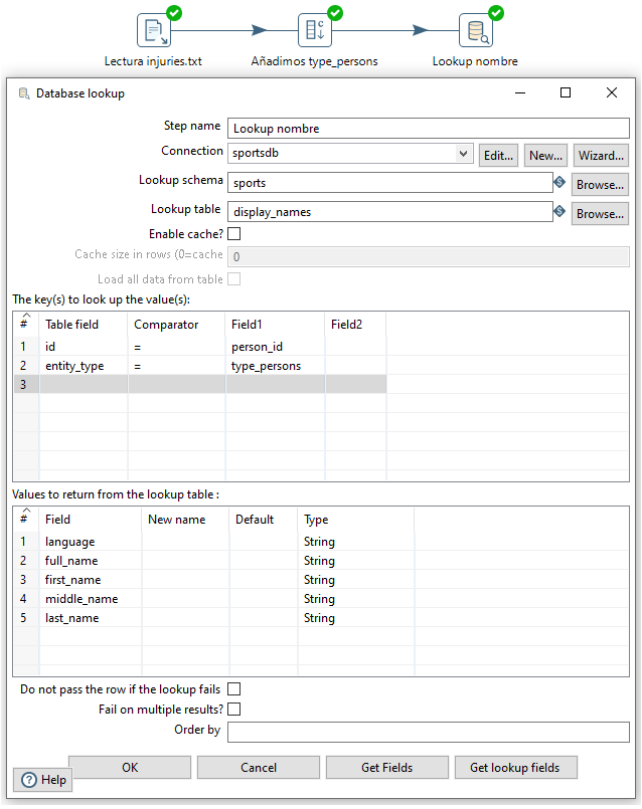
- Paso **Constant / Añadir Constante** → propiedad **type_persons** = **persons**.



Caso de Uso 6 - Añadimos constante persons

3.3 Búsqueda en la base de datos

- Paso **Database lookup / Búsqueda en base de datos** configurado para recuperar el **nombre** (y resto de datos) según el person_id y type_persons.



Con un **preview** verás ya los nombres de los deportistas con lesión.

3.4 Caso "usuario no encontrado"

Si un person_id del fichero no existe en la BD, los campos devueltos serán **nulos**.
Si no quieres este comportamiento, marca **"Do not pass the row if the lookup fails"**.

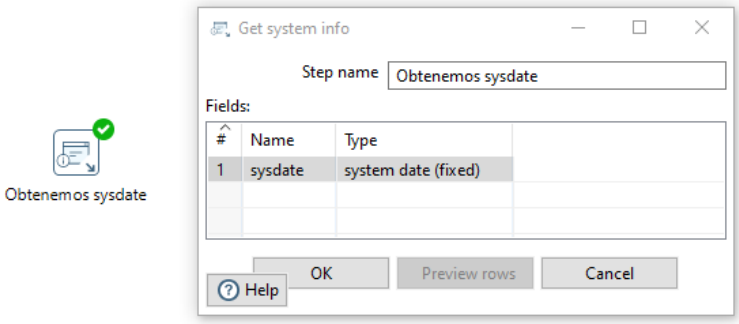
4) Insertar datos (Table Output)

Partimos del mismo fichero **injuries.txt** y vamos a insertar en **injury_phases**.

4.1 Añadir la fecha del sistema

Suponemos que las lesiones finalizan al ejecutar la transformación.

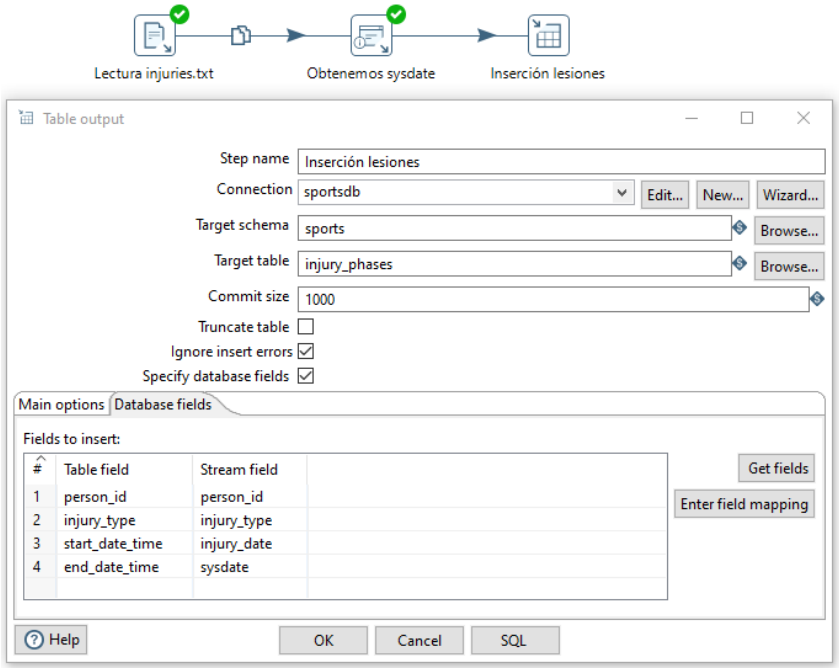
- Paso **Get System Info / Información de sistema** → crear campo **sysdate** con la **fecha del sistema**.



Caso de Uso 6 - Obteniendo la fecha del sistema

4.2 Insertar en la tabla

- Paso **Table output / Salida Tabla** (categoría *Output*): indicar la tabla y **mapear campos** (usa **Get fields**) con las columnas de **injury_phases**.



4.3 Verificación rápida (SQL)

```
SELECT *
FROM injury_phases
WHERE end_date_time > '2021-01-01';
```

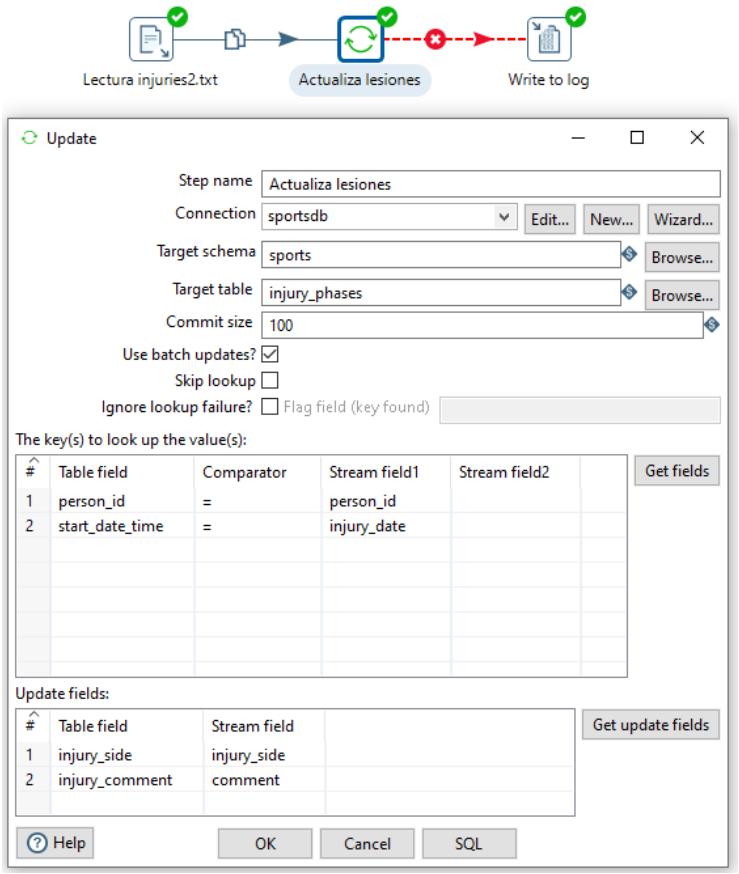
5) Modificar datos (Update + manejo de errores)

Usaremos **injuries2.txt** (CSV) con datos ampliados:

```
person_id;injury_type;injury_side;injury_date;comment
153;elbow;left;2007-07-09;temporada 2018
198;shoulder;left;2007-07-15;jugando al futbol
2523;knee;;2007-07-31;ligamento cruzado anterior
9999;other-excused;;2021-01-06;jugando a la consola
```

5.1 Flujo de actualización

1. Nueva transformación: leer el CSV con **CSV Input**.
2. Paso **Update** (categoría *Output*): mapear **campos de búsqueda** y **campos de actualización**.
3. Paso **Write to Log** enlazando los posibles **errores**.

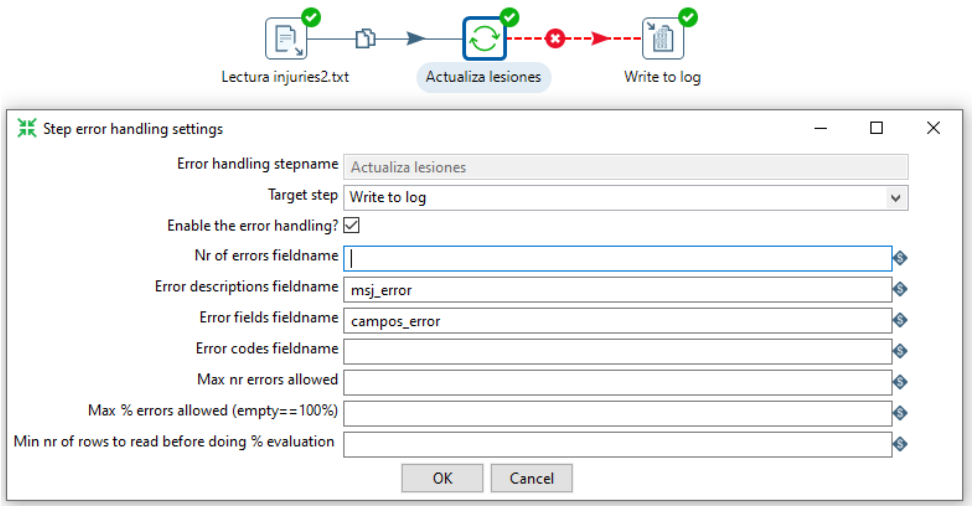


Caso de Uso 6 - Modificando datos

Al ejecutar, en el log puede aparecer un registro no procesado (ejemplo: E=1 en *Finished processing (I=4, O=0, R=4, W=3, U=0, E=1)*).

5.2 Mejorar el mensaje de error

- Clic derecho en **Actualiza lesiones** → **Error Handling...**
- Completar **Error description fieldname** y **Error fields fieldname**.



Caso de Uso 6 - Configurando mensajes de error

Al re-ejecutar, el log mostrará el **motivo del error**. Ejemplo:

```
2021/11/11 12:44:40 - Write to log.0 -
2021/11/11 12:44:40 - Write to log.0 - ------------> Linenr 1------------------------------
2021/11/11 12:44:40 - Write to log.0 - person_id = 9999
2021/11/11 12:44:40 - Write to log.0 - injury_type = other-excused
2021/11/11 12:44:40 - Write to log.0 - injury_side = null
2021/11/11 12:44:40 - Write to log.0 - injury_date = 2021-01-06
2021/11/11 12:44:40 - Write to log.0 - comment = jugando a la consola
2021/11/11 12:44:40 - Write to log.0 - cantidad_errores = 1
2021/11/11 12:44:40 - Write to log.0 - msj_error = Entry to update with following key could not be
found: [9999], [2021-01-06]
2021/11/11 12:44:40 - Write to log.0 - campos_error = person_id, injury_date
2021/11/11 12:44:40 - Write to log.0 -
2021/11/11 12:44:40 - Write to log.0 - ====================
2021/11/11 12:44:40 - Write to log.0 - Finished processing (I=0, O=0, R=1, W=1, U=0, E=0)
2021/11/11 12:44:40 - Actualiza lesiones.0 - Finished processing (I=4, O=0, R=4, W=3, U=0, E=1)
```

6) Upsert (insertar si no existe)

Si, en lugar de error al no encontrar el registro, quieres **insertarlo**, utiliza el paso **Insert/Update**.

Referencias

- Pentaho Data Integration. ETL mediante Spoon. - Inteligencia Artificial y Big Data de Aitor Medrano.
- Pentaho Data Integration Quick Start Guide de *María Carina Roldán*.
- Taller sobre Integración de datos (abiertos) / Uso de *Pentaho Data Integration* (vídeo y diapositivas), por *Jose Norberto Mazón* (Universidad de Alicante)
- Tutorial oficial sobre PDI
- Curso de Spoon en *youtube* del usuario *Learning-BI*.