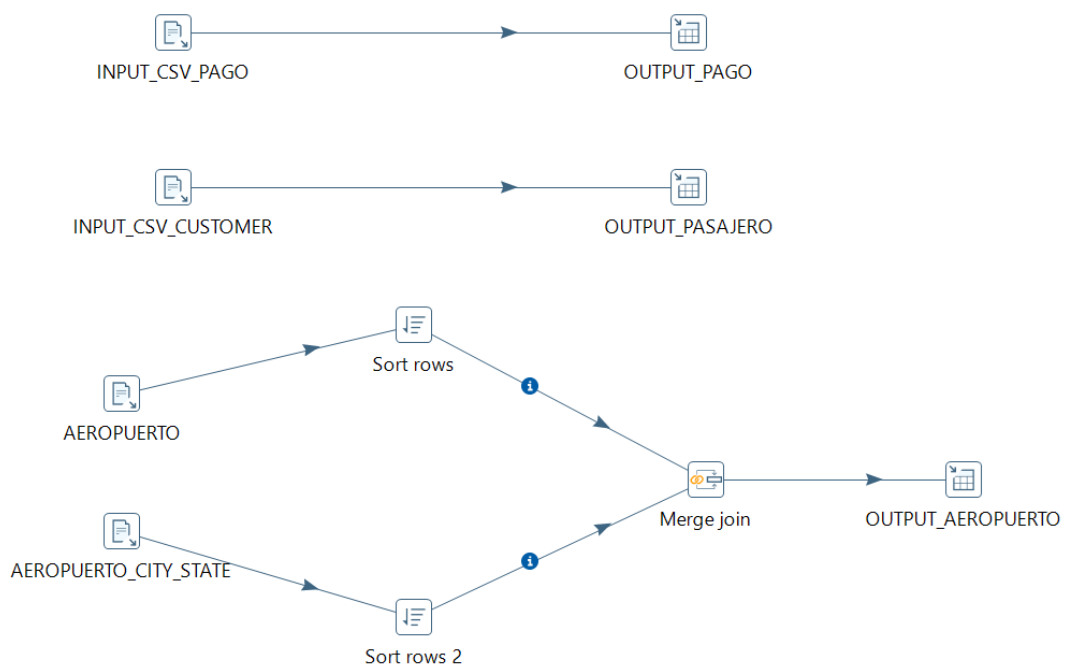


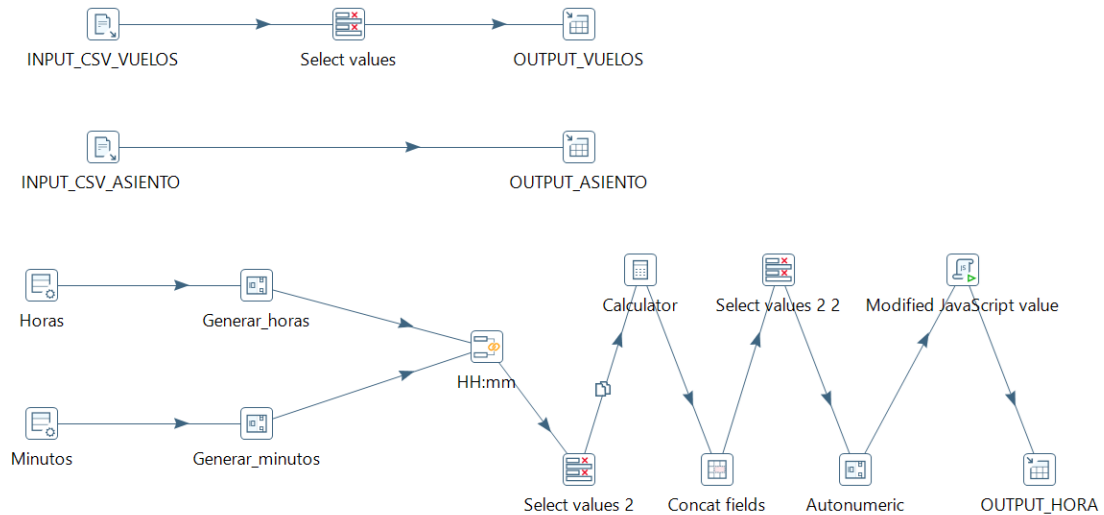
# Task 2: Data transformation

Cristian Torres Ortega, Pablo Gamarro Lozano

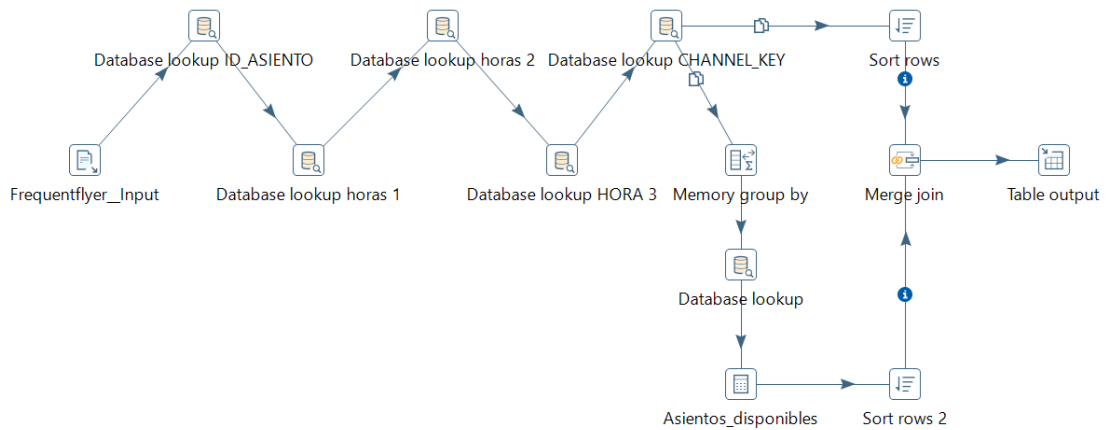
En esta parte final del conjunto de tareas destinadas a diseñar, implementar y cargar datos en un data warehouse, se va a tratar la parte final. Se procederá a la carga de datos en el data warehouse, sin embargo, antes de iniciar este procedimiento se debe de estar completamente seguro de que los datos están de manera correcta para su carga.

A continuación, se muestra el esquema general en la herramienta Pentaho de todas las transformaciones que se han llevado a cabo. El siguiente conjunto de transformaciones corresponden a las dimensiones:



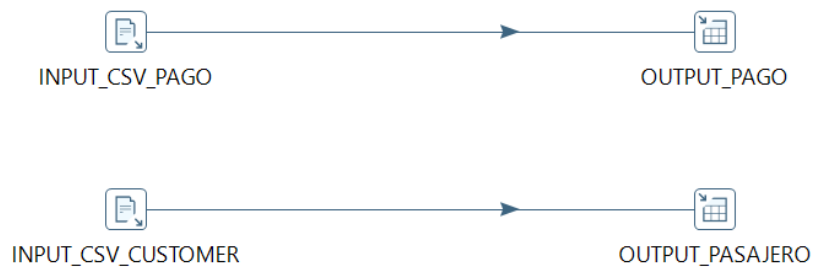


A continuación, se muestra el conjunto de transformaciones para el hecho:

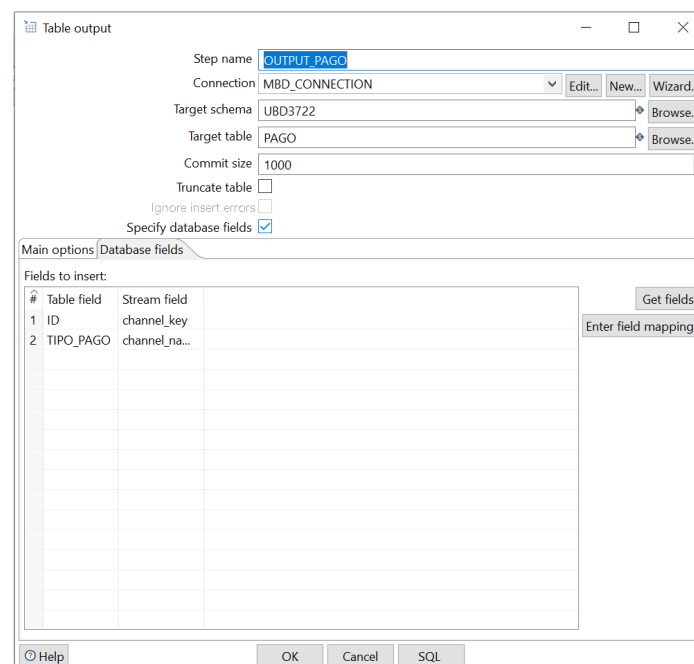


Una vez observado el esquema general de las transformaciones se realiza un análisis más individualizado de estas mismas, explicando el objetivo de cada una, así como la lógica que hay detrás. Como es lógico, todas estas transformaciones comparten un mismo objetivo, que es la carga de unos datos de entrada en unas tablas de salidas que se encuentran en una base de datos. Para cumplir con este objetivo, se llevarán a cabo diferentes etapas dentro de cada transformación.

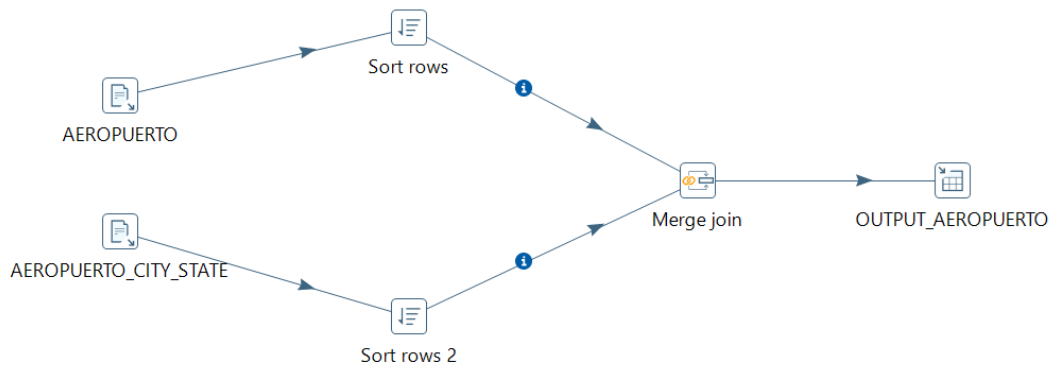
En primer lugar, se observa las siguientes dos transformaciones:



Estas transformaciones tienen como meta la carga directa de los datos en las tablas objetivos de la base de datos de Oracle que se creó anteriormente, en los ejercicios previos. Se trata de un proceso sencillo, donde en primer lugar se han cargado como inputs los “csv” de las tablas pago y customer, y luego se han enlazado a sus correspondientes salidas de tablas de la base de datos Oracle, como se observa en el siguiente ejemplo:



Cabe indicar, que el proceso es tan simple y directo, porque los datos de los “csv” ya vienen estructurados de forma correcta para la carga. En contraste, no sucede lo mismo a la hora de cargar los datos en la tabla aeropuerto:



La problemática que se encuentra a la hora de realizar la carga en la tabla aeropuerto, es que el “csv” llamado “airport” contiene todas las columnas necesarias para la carga excepto la columna “state”, que se encuentra en el “csv” llamado “airport\_city\_state”, por ello, se debe realizar una etapa de unión denominada “merge join” para unir esta columna faltante a las demás, obteniendo una tabla con todas las columnas necesarias para la carga correcta de los datos:

#	Key field
1	leg_city

#	Key field
1	city

Como se observa, la unión en la etapa “merge join” se está realizando a través de una columna en común en ambas tablas, que es la columna con el nombre de la ciudad. Cabe añadir, que para que dicha unión sea correcta y se evite cualquier tipo de error en la carga, es altamente recomendable realizar a cada tabla una ordenación ascendente de la columna por la que se quiere unir ambas tablas, como se ha realizado en la transformación. A continuación, se muestra ambas ordenaciones:

Sort rows

Step name: Sort rows

Sort directory: %%java.io.tmpdir%% Browse...

TMP-file prefix: out

Sort size (rows in memory): 1000000

Free memory threshold (in %):

Compress TMP Files? ☐

Only pass unique rows? (verifies keys only) ☐

Fields:

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

Help OK Cancel Get Fields

Sort rows

Step name: Sort rows

Sort directory: %%java.io.tmpdir%% Browse...

TMP-file prefix: out

Sort size (rows in memory): 1000000

Free memory threshold (in %):

Compress TMP Files? ☐

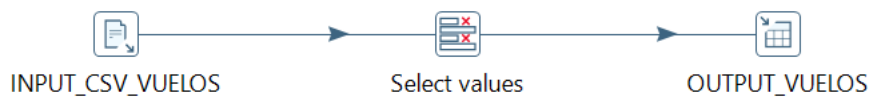
Only pass unique rows? (verifies keys only) ☐

Fields:

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

Help OK Cancel Get Fields

La siguiente transformación a tratar es:



En este caso, se podría hacer una carga directa de los datos, sin etapas intermedias, si no fuera porque en la tabla de destino en la base de datos, las horas de llegadas y salidas están en formato “timestamp” mientras que en el “csv” estas horas se encuentran en formato de “string”. Para solucionar esto, se ha introducido una etapa intermedia denominada “select values”, cuya función es la transformación del tipo de datos de estas celdas, es decir, ayudará a pasar de “string” a “timestamp”:

Select values

Step name: Select values

Select & Alter | Remove | Meta-data

Fields to alter the meta-data for:

#	Fieldname	Rename to	Type	Length	Precision	Binary to Normal?	Format	Date Format Lenient?	Date Locale
1	sched_depart		Timestamp			N	HH:mm	Y	
2	sched_arrival		Timestamp			N	HH:mm	Y	

Help OK Cancel Get fields to change

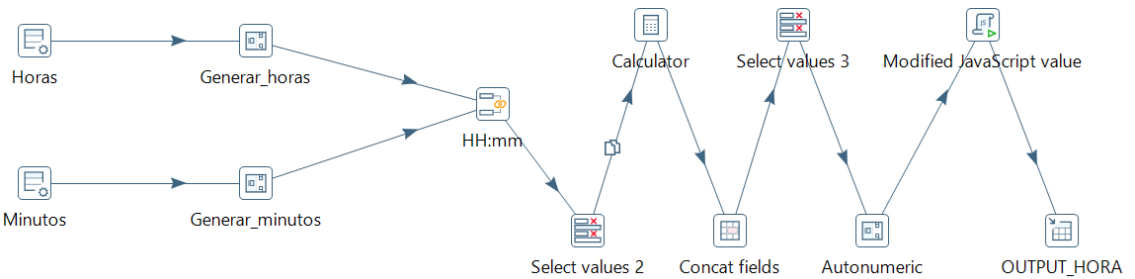
Como se observa, también se la ha introducido el formato que tienen las horas en el input. Además, se tiene habilitada la opción “Date Format Leniente”, esto posibilita que si algún dato

viene de forma diferente, Pentaho mediante determinadas herramientas internas, pueda procesarlo en el formato que se le ha indicado.

La siguiente transformación es:



En este caso, se trata de una transformación muy simple, cuya lógica es la misma que en las dos primeras transformaciones. Por lo tanto, se procede a explicar la siguiente transformación:



Como se muestra a simple vista en la imagen anterior, se trata de una transformación con múltiples etapas cuyo objetivo es poder cargar en la tabla “HORA” de la base de datos todas las horas en formato HH:mm que puede tener un día. Para ello, en primer lugar, se han utilizado dos etapas, “Horas” y “Minutos”, del tipo “Generate Rows”, con la finalidad de crear dos tablas con el número de filas necesarias para almacenar las horas y minutos generados en la etapa posterior:

Generate rows

Step name: Horas

Limit: 24

Never stop generating rows: ☐

Interval in ms (delay): 5000

Current row time field name: now

Previous row time field name: FiveSecondsAgo

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Value	Set empty string?
1										

Help OK Preview Cancel

Generate rows

Step name: Minutos

Limit: 60

Never stop generating rows: ☐

Interval in ms (delay): 5000

Current row time field name: now

Previous row time field name: FiveSecondsAgo

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Value	Set empty string?
1										

Help OK Preview Cancel

Como se observa, se ha colocado un límite de 24 filas en “Horas” y de 60 filas en “Minutos”. La siguiente etapa consiste en generar por un lado las horas, y por otro lado los minutos utilizando transformaciones del tipo “Add sequence”:

The 'Add sequence' dialog box for 'Generar\_horas' shows the following configuration:

- Step name: **Generar\_horas**
- Name of value: **horas**
- Use a database to generate the sequence: ☐
- Connection: **MBD\_CONNECTION**
- Schema name: (empty)
- Sequence name: **SEQ\_**
- Use a transformation counter to generate the sequence: ☒
- Use counter to calculate sequence?: ☒
- Counter name (optional): (empty)
- Start at value: **0**
- Increment by: **1**
- Maximum value: **23**

The 'Add sequence' dialog box for 'Generar\_minutos' shows the following configuration:

- Step name: **Generar\_minutos**
- Name of value: **minutos**
- Use a database to generate the sequence: ☐
- Connection: **MBD\_CONNECTION**
- Schema name: (empty)
- Sequence name: **SEQ\_**
- Use a transformation counter to generate the sequence: ☒
- Use counter to calculate sequence?: ☒
- Counter name (optional): (empty)
- Start at value: **0**
- Increment by: **1**
- Maximum value: **59**

Se ha utilizado en sistema horario de 24 horas. Por lo tanto se ha obtenido por un lado una columna con horas que van de los números 0 al 23, y otra columna con minutos que van de los números 1 al 59. La siguiente etapa, consiste en la combinación de estas dos columnas, utilizando la etapa de transformación de Pentaho denominada “Join rows (cartesian product)”. Con esta etapa se realizará un producto cartesiano entre ambas filas, lo cual generará todas las combinaciones posibles, que es justo lo que se quiere lograr:

The 'Join rows (cartesian product)' dialog box shows the following configuration:

- Step name: **HH:mm**
- Temp directory: **%%java.io.tmpdir%%**
- TMP-file prefix: **out**
- Max. cache size (in rows): **500**
- Main step to read from: (empty)
- The condition: (empty)

A continuación, se necesita determinar qué tipo de datos son estas horas y minutos, donde en primera instancia, serán tipos “string” para facilitar las siguientes etapas de transformación que se van a ver(concatenación). Para obtener las horas y minutos en tipo “string”, se utiliza la etapa “select values” de la misma forma que vimos en transformaciones anteriores:





Concat fields

Step name: Concat fields

Target Field Name: horas

Length of Target Field: 0

Separator: :

Enclosure:

Insert TAB

Fields Advanced

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Trim Type	Null
1	horas	None							both	
2	minutos	None							both	

Get Fields Minimal width

Help OK Cancel

Cabe señalar, que se ha utilizado como separador “:” para lograr el formato deseado, y que, “Trim Type” hace referencia a la eliminación de espacios en blanco, y en este caso, se ha seleccionado que se eliminen los espacios en blanco por ambos lados.

Después de esto, la siguiente etapa es pasar estas horas a formato “Timestamp”, utilizando una etapa de “select values”, como se ha realizado en pasos anteriores:

Select values

Step name: Select values 3

Select & Alter Remove Meta-data

Fields to alter the meta-data for :

#	Fieldname	Rename to	Type	Length	Precision	Binary to Normal?	Format	Date Format Lenient?
1	horas		Timestamp			N	HH:mm	Y

Get fields to change

Help OK Cancel

De esta forma se obtienen las horas generadas y en el formato requerido, sin embargo, para cargarlas se necesita de la generación de una clave primaria, la cual se generará mediante una etapa de “add sequence”:

**Add sequence**

Step name: Autonumeric

Name of value: id

Use a database to generate the sequence

Use DB to get sequence? ☐

Connection: MBD\_CONNECTION Edit... New... Wizard...

Schema name: Schemas...

Sequence name: SEQ\_ Sequences...

Use a transformation counter to generate the sequence

Use counter to calculate sequence? ☒

Counter name (optional):

Start at value: 1

Increment by: 1

Maximum value: 999999999

Help OK Cancel

La penúltima etapa que queda por explicar es la generación de la columna “momento\_dia” que se encuentra en la base de datos. Esta es la última columna a generar antes de proceder a la carga de los datos. Para realizar esta carga de datos, se utiliza una etapa de Pentaho denominada “Modified JavaScript value”:

**Modified JavaScript value**

Step name: Modified JavaScript value

Java script functions:

- Transform Scripts
- Transform Constants
- Transform Functions
- Input fields
  - horas
  - minutos
  - hora\_sola
  - horas\_1
  - id
- Output fields
  - Please use the 'Rep'

Script 1

```
//Script here
var momento_dia;
if(hora_sola < 6 || hora_sola >= 20){
    momento_dia = "Night";
}
else if(hora_sola >= 6 && hora_sola < 12){
    momento_dia = "Morning";
}
else{
    momento_dia = "Afternoon";
}
```

Linennr: 0

Compatibility mode? ☐ Optimization level: 9

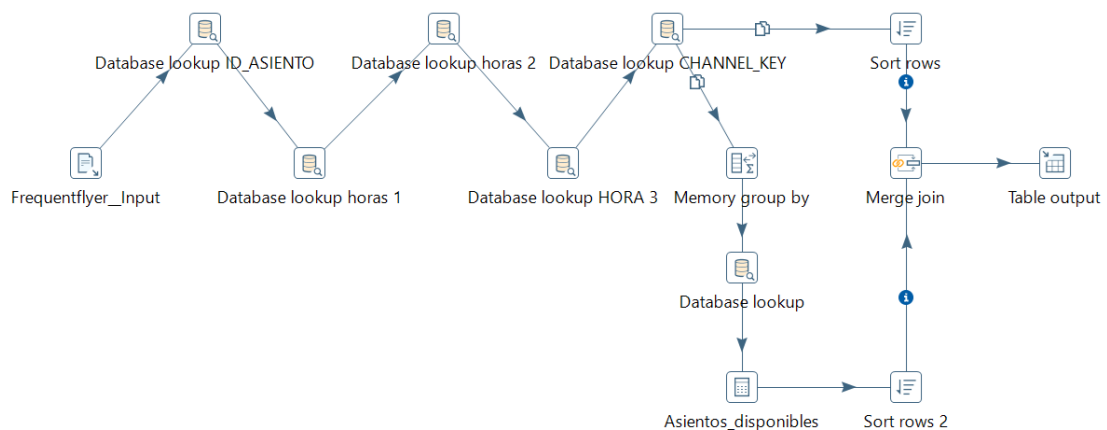
#	Fieldname	Rename to	Type	Length	Precision	Replace value 'Fieldname' or 'Rename to'
1	momento_dia		String			N

Help OK Cancel Get variables Test script

En esta etapa, se ejecuta un script que realiza transformaciones en el flujo de datos de entrada, dando lugar a un flujo de datos de salida. En este caso, como se observa en el script, se crea la variable “momento\_día”. Este campo es completado con los valores “Night”, “Morning” o “Afternoon”, según el rango de valores numéricos en el que se encuentre el campo “hora\_sola” que se había generado con anterioridad. Esta comprobación o cribado se realiza gracias al uso de condicionales. En la parte baja de esta ventana, se observa cómo se establece que esta etapa devuelva el campo creado “momento\_dia” en formato “String”.

Por último, solo queda utilizar la etapa “Table output” de Pentaho para cargar todos los datos en la tabla de “hora” de la base de datos. Ahora sí, se procede a ejecutar las transformaciones en Pentaho para cargar todos los datos de las dimensiones en la base de datos, y así tener estas tablas preparadas para la posterior carga de los datos en la tabla de hecho tras las transformaciones pertinentes.

Hasta este punto, se tienen todas las transformaciones referentes a las dimensiones, a continuación, se va a proceder a explicar en detalle la transformación del hecho:



Como se observa, se parte de una etapa inicial, donde se carga el “csv” denominado “frequentflyer”. Luego se realiza una etapa “Database lookup”. Este tipo de etapas traen a la tabla de entrada los campos de la entrada de referencia donde se busca, extrayendo solamente los datos donde las columnas de valores elegidas de ambas tablas coincidan. En este caso, el lookup es el siguiente:

Database lookup

Step name: Database lookup ID\_ASIENTO

Connection: MBD\_CONNECTION

Lookup schema: UBD3722

Lookup table: ASIENTO

Enable cache? ☐

Cache size in rows (0=cache): 0

Load all data from table: ☐

The key(s) to look up the value(s):

#	Table field	Comparator	Field1	Field2
1	ID	=	fare_class_key	

Values to return from the lookup table :

#	Field	New name	Default	Type
1	ID	id_asiento		None

Do not pass the row if the lookup fails: ☒

Fail on multiple results? ☐

Order by:

Help OK Cancel Get Fields Get lookup fields

Aquí se observa, que se está buscando traer a la tabla de entrada aquellos valores de la tabla ASIENTO de la base de datos, donde los campos “ID” de la tabla ASIENTO y los campos “fare\_class\_key” de la tabla de entrada coincidan. El objetivo detrás de este movimiento es añadir a los datos de entrada la columna id\_asiento, el cual es el campo retornado por el “Database lookup” y necesario para cargar los datos en la tabla hecho de la base de datos.

Otras columnas que se necesitan añadir a los datos de entradas son “id\_hora\_salida” y “id\_hora\_llegada”. Para lograrlo, primero se añaden estas variables en formato “timestamp”, mediante un “Database lookup” a la tabla de referencia “VUELOS” de la base de datos, donde han de coincidir las columnas de ambas tablas que hacen referencia a las claves de vuelo:

Database lookup

Step name: Database lookup horas 1

Connection: MBD\_CONNECTION

Lookup schema: UBD3722

Lookup table: VUELOS

Enable cache? ☐

Cache size in rows (0=cache): 0

Load all data from table: ☐

The key(s) to look up the value(s):

#	Table field	Comparator	Field1	Field2
1	ID	=	flight_key	

Values to return from the lookup table :

#	Field	New name	Default	Type
1	HORASALIDA	HORA_SALIDA_TIMESTAMP		Timestamp
2	HORALLEGADA	HORA_LLEGADA_TIMESTAMP		Timestamp

Do not pass the row if the lookup fails: ☒

Fail on multiple results? ☐

Order by:

Help OK Cancel Get Fields Get lookup fields

Una vez se han retornado los dos campos desde la tabla de referencia, se deben realizar dos etapas más de “Database lookup”, una para cada campo, tomando como tabla de referencia en la base de datos la tabla “HORA”, y haciendo coincidir los campos de ambas tablas que contienen las horas en “timestamp”, retornando el valor de “ID” de la tabla “VUELOS”. De este modo, las dos etapas de “Database lookup” son las siguientes:

The screenshot shows the 'Database lookup' dialog box for 'Database lookup horas 2'. The 'Step name' is 'Database lookup horas 2'. The 'Connection' is 'MBD\_CONNECTION'. The 'Lookup schema' is 'UBD3722'. The 'Lookup table' is 'HORA'. The 'Enable cache?' checkbox is unchecked. The 'Cache size in rows (0=cache)' is '0'. The 'Load all data from table' checkbox is unchecked. The 'The key(s) to look up the value(s):' table has one row: #1, Table field: HORA, Comparator: =, Field1: HORA\_LLEGADA\_TIMESTAMP, Field2: (empty). The 'Values to return from the lookup table:' table has one row: #1, Field: ID, New name: id\_hora\_llegada, Default: (empty), Type: Number. The 'Do not pass the row if the lookup fails' checkbox is checked. The 'Fail on multiple results?' checkbox is unchecked. The 'Order by' field is empty. The buttons at the bottom are Help, OK, Cancel, Get Fields, and Get lookup fields.

#	Table field	Comparator	Field1	Field2
1	HORA	=	HORA_LLEGADA_TIMESTAMP	

#	Field	New name	Default	Type
1	ID	id_hora_llegada		Number

The screenshot shows the 'Database lookup' dialog box for 'Database lookup horas 3'. The 'Step name' is 'Database lookup horas 3'. The 'Connection' is 'MBD\_CONNECTION'. The 'Lookup schema' is 'UBD3722'. The 'Lookup table' is 'HORA'. The 'Enable cache?' checkbox is unchecked. The 'Cache size in rows (0=cache)' is '0'. The 'Load all data from table' checkbox is unchecked. The 'The key(s) to look up the value(s):' table has one row: #1, Table field: HORA, Comparator: =, Field1: HORA\_SALIDA\_TIMESTAMP, Field2: (empty). The 'Values to return from the lookup table:' table has one row: #1, Field: ID, New name: id\_hora\_salida, Default: (empty), Type: None. The 'Do not pass the row if the lookup fails' checkbox is checked. The 'Fail on multiple results?' checkbox is unchecked. The 'Order by' field is empty. The buttons at the bottom are Help, OK, Cancel, Get Fields, and Get lookup fields.

#	Table field	Comparator	Field1	Field2
1	HORA	=	HORA_SALIDA_TIMESTAMP	

#	Field	New name	Default	Type
1	ID	id_hora_salida		None

La siguiente etapa es también de tipo “Database lookup”, donde se busca un filtrado de datos como también sucede en las etapas anteriores, donde al tener marcada la opción “Do not pass the row if the lookup fails” se eliminan aquellas filas cuyo campo se está comparando tiene valores que no se encuentran dentro de las restricciones establecidas. Normalmente, cuando existe un gran volumen de datos, se debe realizar un análisis exploratorio de datos para averiguar a que campos se debe de realizar este filtrado, pero en este caso como se tratan de pocos datos y fácilmente analizables a simple vista, se puede obviar este paso. Por lo tanto, este lookup se realiza sobre la tabla pago obtener los “id\_pago” correctamente filtrados, por lo que queda de la siguiente forma:

Database lookup

Step name: Database lookup CHANNEL KEY

Connection: MBD\_CONNECTION

Lookup schema: UBD3722

Lookup table: PAGO

Enable cache? ☐

Cache size in rows (0=cache): 0

Load all data from table ☐

The key(s) to look up the value(s):

#	Table field	Comparator	Field1	Field2
1	ID	=	channel_key	

Values to return from the lookup table:

#	Field	New name	Default	Type
1	ID	id_pago		Integer

Do not pass the row if the lookup fails ☒

Fail on multiple results? ☐

Order by:

Help OK Cancel Get Fields Get lookup fields

Desde esta etapa surgen dos caminos en paralelos, pero antes de su explicación, cabe indicar que cuando desde un input se ramifica en dos o más ramas diferentes como en este caso, Pentaho te da la opción de distribuir la información entre las ramas o copiar esta información de entrada en cada una de las ramas, en este caso se ha elegido la opción de copiar, para evitar pérdida de datos o posibles complicaciones. Bien, siguiendo con la explicación, como posteriormente se deben unir ambos caminos, se realiza una ordenación ascendente en la etapa llamada “sort rows”, esta ordenación se realiza tanto en la “flight\_key” como en la “flown key”:

Sort rows

Step name: Sort rows

Sort directory: %%java.io.tmpdir%%

TMP-file prefix: out

Sort size (rows in memory): 1000000

Free memory threshold (in %):

Compress TMP Files? ☐

Only pass unique rows? (verifies keys only) ☐

Fields:

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

Help OK Cancel Get Fields

A continuación, se procede a explicar el otro camino, donde la finalidad de este conjunto es crear la última columna que falta para poder insertar todos los datos en la tabla, esta columna es la de “asientos\_disponibles”, que se calcularán restando a la capacidad el número de pasajeros que viaja en cada vuelo. Por lo tanto, se debe de crear el campo “número\_pasajeros”, lo cual se realiza mediante una etapa “Memory group by”, donde se realiza la agrupación por “flight\_key” y “flown\_key” de los diferentes números de tickets, obteniendo la cantidad de tickets diferentes que hay en cada vuelo realizado, lo que vendría a ser el número de pasajeros:

Nota: como en los datasets no vienen datos de fechas, esta agrupación se realiza de la forma anteriormente explicada. Si hubiera información de fechas, se realizaría la agrupación por la “flight\_key”, “flown\_key”, “fecha” y “hora”, por ese orden.

Memory group by

Step name

Memory group by

Always give back a result row

The fields that make up the group:

#	Group field
1	flight_key
2	flown_key

Get Fields

Aggregates :

#	Name	Subject	Type
1	Número_Pasajeros	ticket_number	Number of Distinct Values (N)

Get lookup fields

Help

OK

Cancel

Luego, para calcular el campo de asientos disponibles, falta en los datos de entrada el campo “capacidad”, por lo que se debe de realizar una etapa “Database lookup” a la tabla vuelos para conseguir el campo “capacidad”:

Database lookup

Step name: Database lookup

Connection: MBD\_CONNECTION [Edit... New... Wizard...]

Lookup schema: UBD3722 [Browse...]

Lookup table: VUELOS [Browse...]

Enable cache? ☐

Cache size in rows (0=cache): 0

Load all data from table ☐

The key(s) to look up the value(s):

#	Table field	Comparator	Field1	Field2
1	ID	=	flight_key	

Values to return from the lookup table:

#	Field	New name	Default	Type
1	CAPACIDAD			Number

Do not pass the row if the lookup fails ☒

Fail on multiple results? ☐

Order by:

[Help] [OK] [Cancel] [Get Fields] [Get lookup fields]

Como ya se ha obtenido la capacidad y el número de pasajeros, se puede proceder a calcular los asientos disponibles con la utilización de la etapa “calculator”:

Calculator

Step name: Asientos\_disponibles [Calculator icon]

☒ Throw an error on non existing files

Fields:

#	New field	Calculation	Field A	Field B	Field C	Value type	Length	Precision	Remove
1	asientos_disponibles	A - B	CAPACIDAD	Número_Pasajeros		None			N

[Help] [OK] [Cancel]

Como se observa, los “asientos\_disponibles” se obtienen restando la columna “capacidad” a la columna “numero\_pasajeros”. Por último, solamente queda la etapa de ordenación ascendente de las mismas claves, para poder realizar la etapa posterior “Merge join” con garantía de éxitos:



Sort rows

Step name: Sort rows 2

Sort directory: %%java.io.tmpdir%% Browse...

TMP-file prefix: out

Sort size (rows in memory): 1000000

Free memory threshold (in %):

Compress TMP Files? ☐

Only pass unique rows? (verifies keys only) ☐

Fields:

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

Help OK Cancel Get Fields

Como penúltimo paso, se tiene la etapa “Merge join” para unificar todos los datos de ambas ramas, esta unión se realizará por los campos “flight\_key” y “flown\_key”:

Merge join

Step name: Merge join

First Step: Sort rows

Second Step: Sort rows 2

Join Type: INNER

Keys for 1st step:

#	Key field
1	flight_key
2	flown_key

Keys for 2nd step:

#	Key field
1	flight_key
2	flown_key

Get key fields Get key fields

Help OK Cancel

Ahora sí, el último paso es la etapa “table output”, gracias a la cual se cargarán los datos a la tabla del hecho (H\_TRAYECTO):

Table output

Step name: Table output

Connection: MBD\_CONNECTION [Edit... New... Wizard...]

Target schema: UBD3722 [Browse...]

Target table: H\_TRAYECTO [Browse...]

Commit size: 1000

Truncate table: ☐

Ignore insert errors: ☐

Specify database fields: ☒

Main options Database fields

Fields to insert:

#	Table field	Stream field
1	ID_AVION	flight_key
2	ID_PASAJERO	customer_key
3	ID_TRAYECTO_ORIGEN	leg_origin_k...
4	ID_TRAYECTO_DESTINO	leg_dest_key
5	ID_ITINERARIO_ORIGEN	trip_origin_k...
6	ID_ITINERARIO_DESTINO	trip_dest_key
7	ID_PAGO	id_pago
8	TARIFA_BASE	fare
9	MILLAS	miles
1.	RETRASO	minutes_late
1.	ID_NUMTICKET	ticket_num...
1.	ID_ASIENTO	id_asiento
1.	ID_HORA_LLEGADA	id_hora_lleg...
1.	ID_HORA_SALIDA	id_hora_salida
1.	ASIENTOS_DISPONIBLES	asientos_dis...

[Get fields]

[Enter field mapping]

[Help] [OK] [Cancel] [SQL]

Estos son los registros que contienen cada tabla de la base de datos:

	NOMBRE TABLA	Nº REGISTROS
1	AEROPUERTO	20
2	ASIENTO	4
3	HORA	1440
4	H_TRAYECTO	285
5	PAGO	4
6	PASAJERO	100
7	VUELOS	100