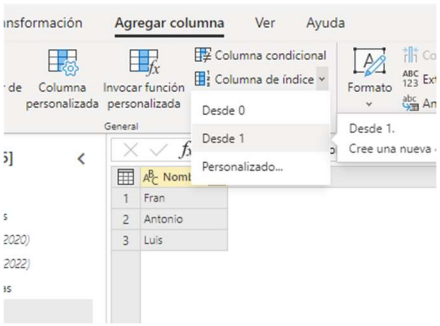
9. Creación y utilización de funciones personalizadas en flujos de datos

Power Query trabaja fundamentalmente con fórmulas o funciones. Cada vez que nosotros utilizamos un botón de la interfaz de Power Query, crea una línea de código M que es un lenguaje basado principalmente en funciones. Cada paso utiliza una o varias funciones nativas y el usuario final está programando código sin necesidad de conocerlo.

Por ejemplo, si añadimos a una tabla un índice con la interfaz:



Se va a generar la siguiente línea de código M:

```
Table.AddIndexColumn(Origen, "Índice", 1, 1, Int64.Type)
```

```
Table.AddIndexColumn es una función nativa que va a requerir una serie de parámetros:
```

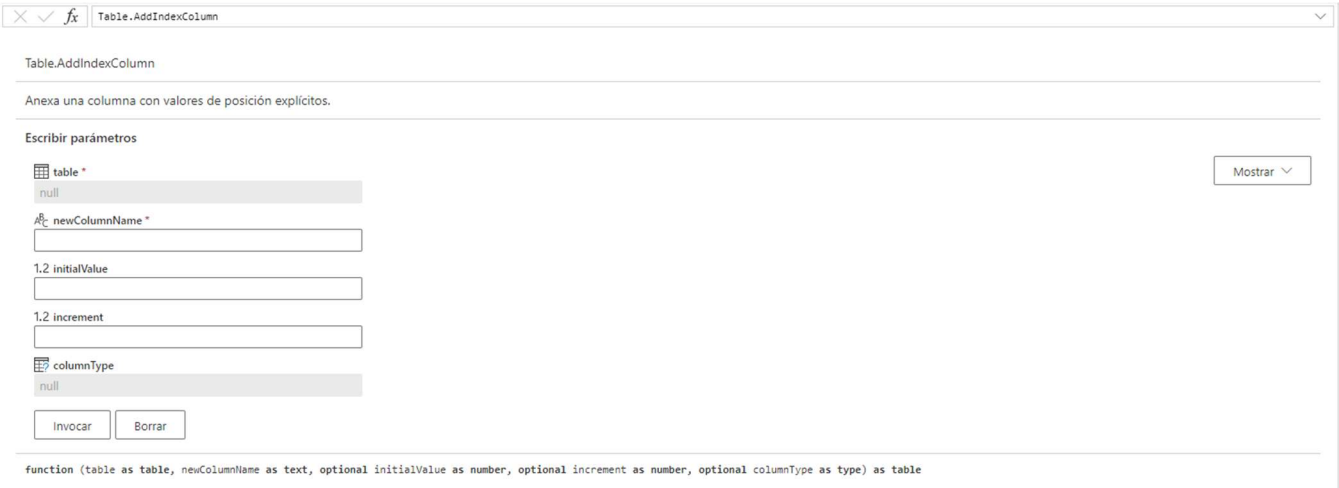


Table es el primer parámetro, que en nuestro caso es "Origen". Por lo general, cada vez que utilizamos una función de tabla, va a estar referenciada al nombre del paso anterior ya que el paso anterior en sí, es habitualmente una tabla

newColumnName es el nombre de la columna, que la interfaz pone por defecto como "Índice".

1 es el initial value y el segundo 1 es el increment y por último, Int64.Type es el columnType, el tipo de dato de la columna.

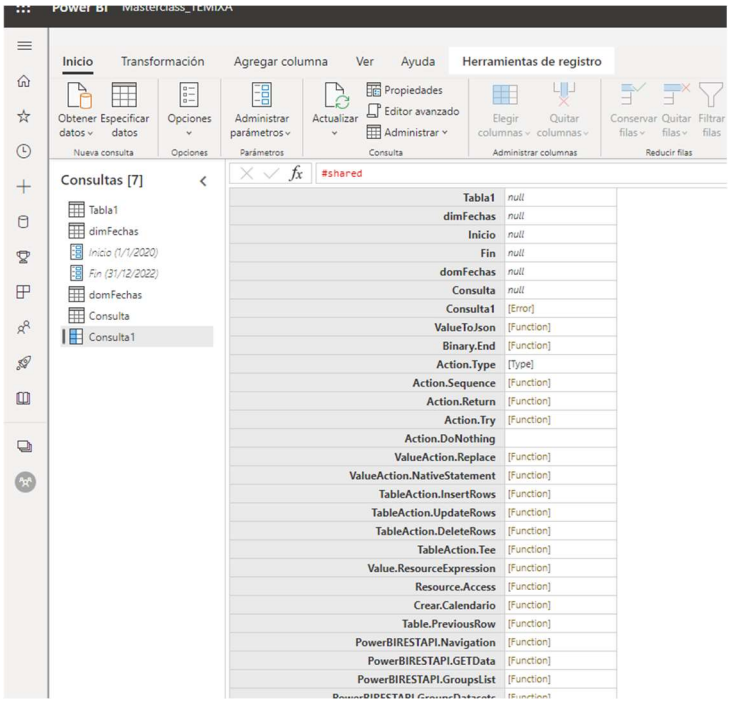
Como curiosidad, al igual que Power Query en el Desktop, los flujos de datos contienen la información de todas y cada una de las funciones nativas existentes. Hay dos vías para acceder a la documentación de esas funciones a través de los flujos de datos. En la imagen que he puesto anteriormente, en una Consulta en blanco en puesto la función sin más, `Table.AddIndexColumn` y me devuelve la información que el sistema tiene documentada de la propia función. (Ver imagen anterior)

```
let
  Origen = Table.AddIndexColumn
in
  Origen
```

Esta instrucción, por tanto, va a devolver la función y desde esa interfaz de la función podríamos invocarla y ver los datos que genera, aunque carece de sentido dado que tenemos el botón de la interfaz. Si nos sirve para ver su funcionamiento y los parámetros que necesita y que escribe la interfaz por nosotros al darle al botón.

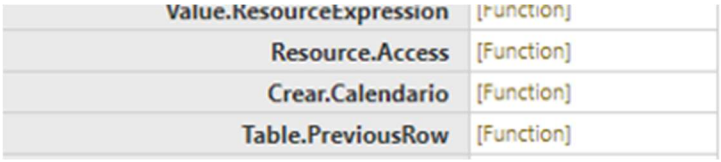
La segunda opción, que también funciona en los flujos de datos al igual que en el desktop es usar la instrucción #shared.

Eso nos va a devolver un listado con todas las funciones disponibles en los flujos de datos.



Por otra parte, los usuarios avanzados de power query son capaces de generar sus propias funciones personalizadas. Crear una función no es nada complicado, aprenderemos en la parte del libro dedicada a Power Query. En resumen, es generar un código que realice una serie de acciones más o menos complejas que se puedan parametrizar y reutilizar. La red está llena de ejemplos de funciones de M personalizadas y tenemos un magnífico repositorio explicado en español en la web https://powerquery.zone/ de los colombianos Miguel Caballero y Fabián Torres. La forma habitual de utilizar las funciones es crearlas en el editor e invocarlas generando el resultado deseado, y cuando se quiere utilizar en otro conjunto de datos, se copia y se pega el código que genera la función y se vuelve a invocar, pero Power BI Desktop nos permite de una forma particular que vamos a ver en este capítulo, generar una librería de funciones personalizadas que se quedan registradas en la aplicación para poder llamarlas directamente en nuestro editor de consultas como si de una función nativa se tratase.

El lector se preguntará por qué hablo de una característica propia de Power BI Desktop cuando el libro está centrado en los flujos de datos de Power BI. Pues bien, el motivo es que en el transcurso de la investigación que hice sobre le funcionamiento de las funciones personalizadas del Desktop descubrí la forma de trasladar esa librería de funciones personalizadas a los flujos de datos a través de la puerta de enlace. La ventaja, con respecto al Desktop es aun mayor, porque no va a depender de tener el archivo con la librería de funciones personalizadas en cada instancia de Power BI Desktop que utilices para desarrollar tus proyectos, sino que vas a disponer de ellas online a través de la puerta de enlace.



Value.ResourceExpression	[Function]
Resource.Access	[Function]
Crear.Calendario	[Function]
Table.PreviousRow	[Function]

Si nos fijamos en detalle en la imagen anterior, Crear.Calendario no es una función nativa de Power Query Online ni mucho menos. Es una función creada por mi mismo e integrada en la librería de funciones personalizadas y que me va a permitir crear automáticamente mi calendario con más de 30 atributos desde una fecha de inicio que yo determine hasta una fecha de fin, e incluso si lo requiero crear las columnas necesarias para tener un calendario fiscal determinando el mes de inicio de este.

La creación del calendario se verá exhaustivamente en la parte de Power Query, pero en este capítulo se adelanta como crear el fichero librería de funciones, donde se ubicará y como se trasladará a los flujos de datos a través de la puerta de enlace.

9.1 Creación de la librería de funciones personalizadas.

La creación de la librería es tremendamente sencilla. Se puede hace en el bloc de notas o con el Notepad++ o cualquier editor de código y la estructura es la siguiente:

```
section MyLibrary;

shared Nombre.Funcion=

Copiamos y pegamos la función

;

shared Nombre.Funcion2=

Copiamos y pegamos la función

;

shared Nombre.Funcion3=

Copiamos y pegamos la función

;
```

Así cuantas funciones personalizadas queramos integrar en nuestra librería. Importante siempre cerrar con ";" y el shared antes del nombre sin el que no funcionaría.

Una vez creado el archivo se guarda con el siguiente nombre y extensión MyLibrary.pqx

9.2Creación de la función personalizada y documentación

En uno de los capítulos dedicados a power query en la sección III haremos un estudio detenido de la tabla de calendario pero en este momento nos interesa traerla a colación ya que es un ejemplo claro de función que podemos personalizar.

El código completo de la tabla de calendario que crearemos, incluso con las columnas del calendario fiscal es el siguiente:

```
let
  // Listamos los números comprendidos entre la Fecha de Inicio y la Fecha de Fin
  Origen = {Number.From(FechaInicio)..Number.From(FechaFin)},
  // Convertimos nuestra lista en Tabla
  #"Convertido en tabla" = Table.FromList(Origen, Splitter.SplitByNothing(), {"Fecha"
}, null, ExtraValues.Error),
  // Cambiamos el tipo de número a Fecha
  #"Tipo de columna cambiado" = Table.TransformColumnTypes(#"Convertido en tabla", {{
"Fecha", type date}}),
  // Creamos la columna Año
  #"Año insertado" = Table.AddColumn(#"Tipo de columna cambiado", "Ejercicio", each D
ate.Year([Fecha]), Int64.Type),
  // Creamos las columna MesNro
  #"Mes insertado" = Table.AddColumn(#"Año insertado", "MesNro", each Date.Month([Fec
ha]), Int64.Type),
  // Creamos la Columna NombreMes
```

```
  #"Nombre del mes insertado" = Table.AddColumn(#"Mes insertado", "Mes", each Text.Up
per(Date.MonthName([Fecha])), type text),
  // Creamos la columna MesNombre en inglés
  #"Nombre del mes insertado1" = Table.AddColumn(#"Nombre del mes insertado", "Month"
, each Text.Upper(Date.MonthName([Fecha], "en-US")), type text),
  // Creamos la columna día
  #"Día insertado" = Table.AddColumn(#"Nombre del mes insertado1", "NroDia", each Dat
e.Day([Fecha]), Int64.Type),
  // Creamos la columna IdFecha
  #"Personalizado agregado" = Table.AddColumn(#"Día insertado", "IdFecha", each [Ejer
cicio]*10000 + [MesNro]*100 + [NroDia], Int64.Type),
  // Creamos la columna IdFechaEntero
  #"Personalizado agregado 1" = Table.AddColumn(#"Personalizado agregado", "IdFechaEn
tero", each Number.From([Fecha]), Int64.Type),
  // Creamos la columna Trimestre
  #"Trimestre insertado" = Table.AddColumn(#"Personalizado agregado 1", "Trimestre",
each "T" & Text.From(Date.QuarterOfYear([Fecha])), type text),
  // Creamos la columna Trimestre Inglés
  #"Trimestre insertado2" = Table.AddColumn(#"Trimestre insertado", "Quarter", each "
Q" & Text.From(Date.QuarterOfYear([Fecha])), type text),
  // Creamos la columna del numero de dia de la semana
  #"Día de la semana insertado" = Table.AddColumn(#"Trimestre insertado2", "DiaSemana
", each Date.DayOfWeek([Fecha], Day.Monday)+1, type number),
  // Creamos la columna numero de semana del año
  #"Semana del año insertada" = Table.AddColumn(#"Día de la semana insertado", "NroSe
mana", each Date.WeekOfYear([Fecha]), Int64.Type),
  // Creamos la columna Ejercicio-Trimestre
  #"Personalizado agregado 2" = Table.AddColumn(#"Semana del año insertada", "Ejercic
ioTrimestre", each Number.ToText([Ejercicio])&"-"&[Trimestre], type text),
  // Creamos las columnas mes corto y ShortMonth
  #"Personalizado agregado 3" = Table.AddColumn(#"Personalizado agregado 2", "MesCort
o", each Text.Start([Mes], 3), type text),
  #"Personalizado agregado 4" = Table.AddColumn(#"Personalizado agregado 3", "ShortMo
nth", each Text.Start([Month], 3), type text),
  // Creamos la columna DiasLaborables
  #"Columna condicional insertada" = Table.AddColumn(#"Personalizado agregado 4", "Es
Laborable", each if [DiaSemana] <= 5 then 1 else 0, Int64.Type),
  // Creamos la columna DiaEjercicio
  #"Día del año insertado" = Table.AddColumn(#"Columna condicional insertada", "DiaEj
ercicio", each Date.DayOfYear([Fecha]), Int64.Type),
  // Creamos la columna EjercicioMes
  #"Personalizado agregado 5" = Table.AddColumn(#"Día del año insertado", "EjercicioM
es", each Number.ToText([Ejercicio])&"-
"&Text.End("0"& Text.From([MesNro]),2), type text),
  // Creamos la columna Ejercicio Actual
  #"Personalizado agregado 6" = Table.AddColumn(#"Personalizado agregado 5", "Ejercic
io Actual", each Date.Year(DateTime.LocalNow()), Int64.Type),
  // Creamos la columna Desvio Ejercicio
  #"Personalizado agregado 7" = Table.AddColumn(#"Personalizado agregado 6", "DesvioE
jercicio", each [Ejercicio] - [Ejercicio Actual], Int64.Type),
  // Creamos la columna Mes Actual
  #"Personalizado agregado 8" = Table.AddColumn(#"Personalizado agregado 7", "MesActu
al", each Date.Month(DateTime.LocalNow()), Int64.Type),
  // Creamos la columna HoyEntero
  #"Personalizado agregado 9" = Table.AddColumn(#"Personalizado agregado 8", "HoyEnte
ro", each Number.RoundDown(Number.From(DateTime.LocalNow())), Int64.Type),
```

```
  // Creamos la columna Trimestre Actual
  #"Personalizado agregado 10" = Table.AddColumn(#"Personalizado agregado 9", "Trimes
treActual", each Date.QuarterOfYear(DateTime.LocalNow()), Int64.Type),
  // Creamos la columna Semestre
  #"Personalizado agregado 11" = Table.AddColumn(#"Personalizado agregado 10", "Semes
tre", each Number.RoundUp(Date.Month([Fecha])/6), Int64.Type),
  // Creamos la columna Desvio Mes
  #"Personalizado agregado 12" = Table.AddColumn(#"Personalizado agregado 11", "Desvi
oMes", each [MesNro]- [MesActual] + ([DesvioEjercicio]*12), Int64.Type),
  // Creamos la columna Desvio Dia
  #"Resta insertada" = Table.AddColumn(#"Personalizado agregado 12", "DesvioDia", eac
h [IdFechaEntero] - [HoyEntero], Int64.Type),
  // Creamos la columna HastaFecha
  #"Personalizado agregado 13" = Table.AddColumn(#"Resta insertada", "HastaFecha", ea
ch if [IdFechaEntero]<=[HoyEntero] then "PASADO"
else "FUTURO", type text),
  // Creamos la funcion TrimestreNro
  #"Trimestre insertado 1" = Table.AddColumn(#"Personalizado agregado 13", "Trimestre
Nro", each Date.QuarterOfYear([Fecha]), Int64.Type),
  // Creamos la columna DesvioTrimestre
  #"Personalizado agregado 14" = Table.AddColumn(#"Trimestre insertado 1", "DesvioTri
mestre", each [TrimestreNro]-[TrimestreActual]+([DesvioEjercicio]*4), Int64.Type),
  // Creamos la columna Nombredia en Español y en inglés
  #"Nombre del día insertado" = Table.AddColumn(#"Personalizado agregado 14", "Nombre
Dia", each Date.DayOfWeekName([Fecha]), type text),
  #"Nombre del día insertado 1" = Table.AddColumn(#"Nombre del día insertado", "NameD
ay", each Date.DayOfWeekName([Fecha], "en-US"), type text),
  // Creamos la columna Numero Ejercicio Fiscal
  #"Personalizado agregado 15" = Table.AddColumn(#"Nombre del día insertado 1", "NumE
jercicioFiscal", each if [MesNro]<PrimerMesFiscal then[Ejercicio]-
1 else [Ejercicio], Int64.Type),
  // Creamos la columna Numero Mes Fiscal
  #"Personalizado agregado 16" = Table.AddColumn(#"Personalizado agregado 15", "NumMe
sFiscal", each if [MesNro]<PrimerMesFiscal then
13 - PrimerMesFiscal + [MesNro]
else
[MesNro]-PrimerMesFiscal+1, Int64.Type),
  // Creamos la columna Numero trimestre fiscal
  #"Personalizado agregado 17" = Table.AddColumn(#"Personalizado agregado 16", "NumTr
imestreFiscal", each Number.RoundUp([NumMesFiscal]/3, 0), Int64.Type),
  // Creamos las etiquetas de las columnas fiscales
  #"Personalizado agregado 18" = Table.AddColumn(#"Personalizado agregado 17", "Ejerc
icio Fiscal", each "FY-" & Text.From([NumEjercicioFiscal]), type text),
  #"Personalizado agregado 19" = Table.AddColumn(#"Personalizado agregado 18", "Mes F
iscal", each "FM-" &Text.End("0"&Text.From([NumMesFiscal]),2), type text),
  #"Personalizado agregado 20" = Table.AddColumn(#"Personalizado agregado 19", "Trime
stre Fiscal", each "FQ-" & Text.From([NumTrimestreFiscal]), type text),
  #"Personalizado agregado 21" = Table.AddColumn(#"Personalizado agregado 20", "Ejerc
icioMesFIscal", each "FYM-" & Text.From ([NumEjercicioFiscal]) &
Text.End("0"& Text.From([NumMesFiscal]),2), type text),
  #"Personalizado agregado 22" = Table.AddColumn(#"Personalizado agregado 21", "Ejerc
icioTrimestreFiscal", each "FYQ-
" & Text.From([NumEjercicioFiscal]) & Text.From([NumTrimestreFiscal]), type text)
in
  #"Personalizado agregado 22"
```

Se ha subrayado en amarillo tres parámetros que son imprescindibles para su creación: FechaInicio, FechaFin y PrimerMesFiscal. Una función no es más que la aplicación de diversas instrucciones de código a unos parámetros que deben determinarse al invocar la función.

Véase como se crearía la función:

Se comienza como todas las instrucciones en Power query con un

```
let

Funcion=   //este es el nombre del paso

//Y se enumeran los parámetros entre paréntesis indicando su tipo de dato y separados por coma para finalizar con => que es el símbolo que crea la función

(FechaInicio as date, FechaFin as date, PrimerMesFiscal optional as number) =>

//tras la declaración de la función se introduce todo el código con el que se genera la tabla que ya se ha copiado más arriba

let
  // Listamos los números comprendidos entre la Fecha de Inicio y la Fecha de Fin
  Origen = {Number.From(FechaInicio)..Number.From(FechaFin)},

……

in
  #"Personalizado agregado 22"

in

Funcion // Aquí se estaría cerrando el let de la función.   Realmente el let … in de la función no sería necesario si no se integraran metadatos de documentación, pero vamos a aprender también a crear esa documentación
```

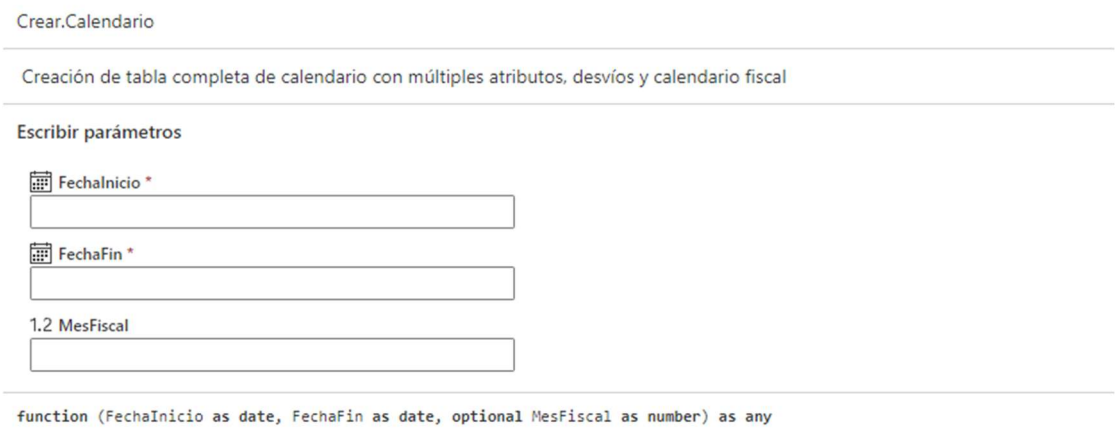
Véase ahora como integrar metadatos de documentación de la función.

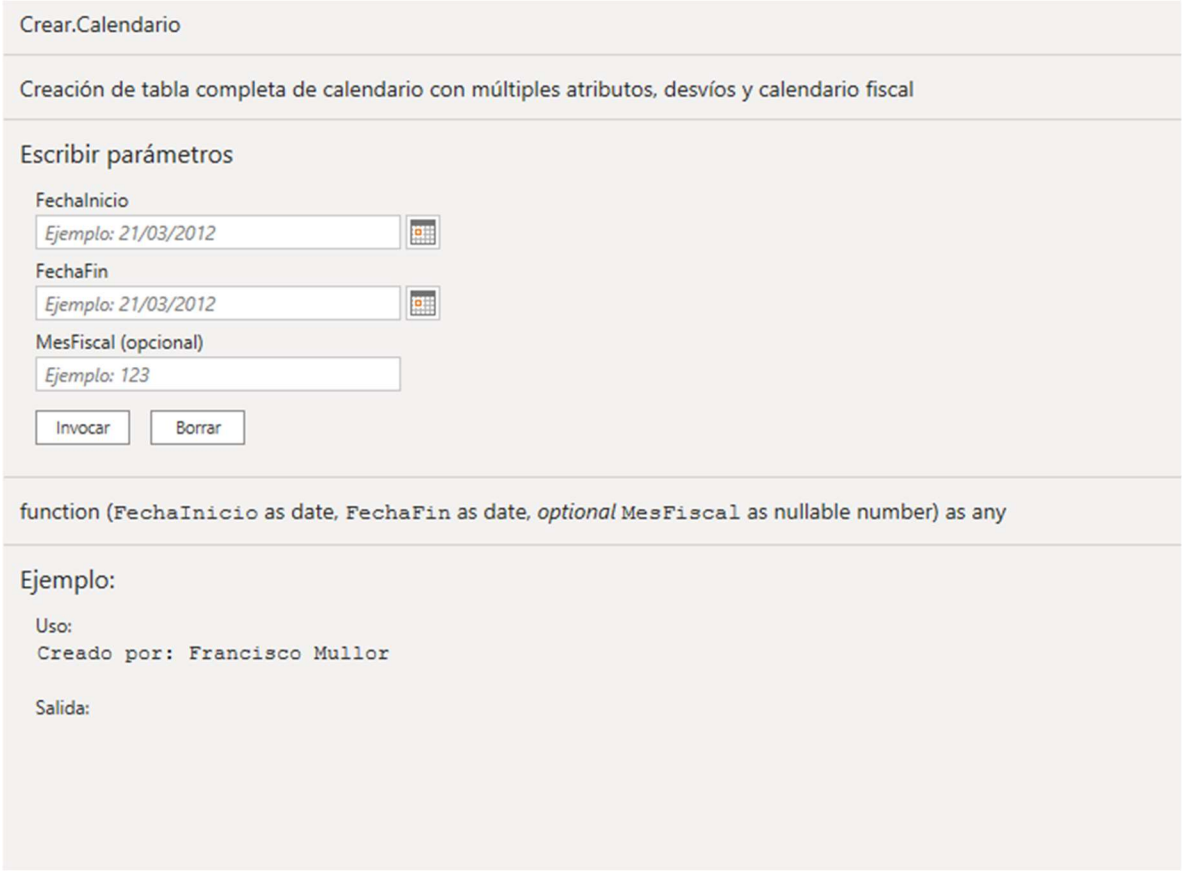
```
let

Funcion=
(FechaInicio as date, FechaFin as date, MesFiscal optional as number) =>
```

```
let

//Señalamos que pasa si el MesFiscal que es opcional no se inserta

PrimerMesFiscal=

if MesFiscal=null
then 1
else MesFiscal


// Listamos los números comprendidos entre la Fecha de Inicio y la Fecha de Fin

  Origen = {Number.From(FechaInicio)..Number.From(FechaFin)},

……

in
  #"Personalizado agregado 22", //Justo tras el código completo que vamos a invocar
con los parámetros de la función y tras una coma comenzaremos a crear los metadatos
de la documentación:

 documentation =
    [
        Documentation.Name =  //Nombre que vamos a dar a nuestra función
        " Crear.Calendario ",

        Documentation.Description = //Descripción

        " Creación de tabla completa de calendario con múltiples atributos, desvíos y
calendario fiscal ",

        Documentation.LongDescription = //Descripción larga, aunque he puesto lo
mismo que en Descripción
        " Creación de tabla completa de calendario con múltiples atributos, desvíos y
calendario fiscal ",

        Documentation.Category = " Table ", //Tipo de función

        Documentation.Source =  //Url donde podemos encontrarla
        " www.powerbisp.com ",

        Documentation.Version = " 1.0 ", //Version

        Documentation.Author = //Autor/-a
        " Francisco Mullor: www.powerbisp.com ",

        Documentation.Examples = //Ejemplos si los hubiera
        {
            [
                Description =  "  ",
                Code = "Creado por: Francisco Mullor ",
                Result = "  "
            ]
```

```
        }
    ]

in

// Y debemos modificar el in que antes habíamos referenciado a la función, para que
recoja la función y los metadatos de la siguiente manera:

Value.ReplaceType(
        Funcion,
        Value.ReplaceMetadata(
            Value.Type( Funcion ),
            documentation)
    ) ;  // Lo que hacemos en este paso es introducir los metadatos para que se vean
al nombrar la función como documentación
```

En todo caso, en la documentación completa solo aparece Nombre y descripción larga además de los parámetros en los flujos de datos de Power BI y en el Power BI Desktop también el apartado de ejemplos como se puede comparar en las dos imágenes;





Ya se ha visto como se genera el archivo MyLibrary.pqx y como se genera la función y su documentación quedando el texto contenido completo del archivo así:

```
section MyLibrary;

shared Crear.Calendario=

let

Funcion=
(FechaInicio as date, FechaFin as date, optional MesFiscal  as number) =>


let

//Señalamos que pasa si el MesFiscal que es opcional no se inserta

PrimerMesFiscal=

if MesFiscal=null
then 1
else MesFiscal,

  // Listamos los números comprendidos entre la Fecha de Inicio y la Fecha de Fin
  Origen = {Number.From(FechaInicio)..Number.From(FechaFin)},
  // Convertimos nuestra lista en Tabla
```

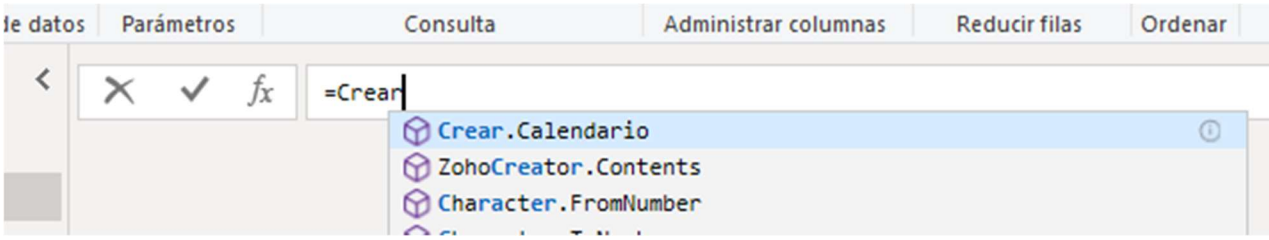
```
  #"Convertido en tabla" = Table.FromList(Origen, Splitter.SplitByNothing(), {"Fecha"
}, null, ExtraValues.Error),
  // Cambiamos el tipo de número a Fecha
  #"Tipo de columna cambiado" = Table.TransformColumnTypes(#"Convertido en tabla", {{
"Fecha", type date}}),
  // Creamos la columna Año
  #"Año insertado" = Table.AddColumn(#"Tipo de columna cambiado", "Ejercicio", each D
ate.Year([Fecha]), Int64.Type),
  // Creamos las columna MesNro
  #"Mes insertado" = Table.AddColumn(#"Año insertado", "MesNro", each Date.Month([Fec
ha]), Int64.Type),
  // Creamos la Columna NombreMes
  #"Nombre del mes insertado" = Table.AddColumn(#"Mes insertado", "Mes", each Text.Up
per(Date.MonthName([Fecha])), type text),
  // Creamos la columna MesNombre en inglés
  #"Nombre del mes insertado1" = Table.AddColumn(#"Nombre del mes insertado", "Month"
, each Text.Upper(Date.MonthName([Fecha], "en-US")), type text),
  // Creamos la columna día
  #"Día insertado" = Table.AddColumn(#"Nombre del mes insertado1", "NroDia", each Dat
e.Day([Fecha]), Int64.Type),
  // Creamos la columna IdFecha
  #"Personalizado agregado" = Table.AddColumn(#"Día insertado", "IdFecha", each [Ejer
cicio]*10000 + [MesNro]*100 + [NroDia], Int64.Type),
  // Creamos la columna IdFechaEntero
  #"Personalizado agregado 1" = Table.AddColumn(#"Personalizado agregado", "IdFechaEn
tero", each Number.From([Fecha]), Int64.Type),
  // Creamos la columna Trimestre
  #"Trimestre insertado" = Table.AddColumn(#"Personalizado agregado 1", "Trimestre",
each "T" & Text.From(Date.QuarterOfYear([Fecha])), type text),
  // Creamos la columna Trimestre Inglés
  #"Trimestre insertado2" = Table.AddColumn(#"Trimestre insertado", "Quarter", each "
Q" & Text.From(Date.QuarterOfYear([Fecha])), type text),
  // Creamos la columna del numero de dia de la semana
  #"Día de la semana insertado" = Table.AddColumn(#"Trimestre insertado2", "DiaSemana
", each Date.DayOfWeek([Fecha], Day.Monday)+1, type number),
  // Creamos la columna numero de semana del año
  #"Semana del año insertada" = Table.AddColumn(#"Día de la semana insertado", "NroSe
mana", each Date.WeekOfYear([Fecha]), Int64.Type),
  // Creamos la columna Ejercicio-Trimestre
  #"Personalizado agregado 2" = Table.AddColumn(#"Semana del año insertada", "Ejercic
ioTrimestre", each Number.ToText([Ejercicio])&"-"&[Trimestre], type text),
  // Creamos las columnas mes corto y ShortMonth
  #"Personalizado agregado 3" = Table.AddColumn(#"Personalizado agregado 2", "MesCort
o", each Text.Start([Mes], 3), type text),
  #"Personalizado agregado 4" = Table.AddColumn(#"Personalizado agregado 3", "ShortMo
nth", each Text.Start([Month], 3), type text),
  // Creamos la columna DiasLaborables
  #"Columna condicional insertada" = Table.AddColumn(#"Personalizado agregado 4", "Es
Laborable", each if [DiaSemana] <= 5 then 1 else 0, Int64.Type),
  // Creamos la columna DiaEjercicio
  #"Día del año insertado" = Table.AddColumn(#"Columna condicional insertada", "DiaEj
ercicio", each Date.DayOfYear([Fecha]), Int64.Type),
  // Creamos la columna EjercicioMes
  #"Personalizado agregado 5" = Table.AddColumn(#"Día del año insertado", "EjercicioM
es", each Number.ToText([Ejercicio])&"-
"&Text.End("0"& Text.From([MesNro]),2), type text),
```

```
  // Creamos la columna Ejercicio Actual
  #"Personalizado agregado 6" = Table.AddColumn(#"Personalizado agregado 5", "Ejercic
io Actual", each Date.Year(DateTime.LocalNow()), Int64.Type),
  // Creamos la columna Desvio Ejercicio
  #"Personalizado agregado 7" = Table.AddColumn(#"Personalizado agregado 6", "DesvioE
jercicio", each [Ejercicio] - [Ejercicio Actual], Int64.Type),
  // Creamos la columna Mes Actual
  #"Personalizado agregado 8" = Table.AddColumn(#"Personalizado agregado 7", "MesActu
al", each Date.Month(DateTime.LocalNow()), Int64.Type),
  // Creamos la columna HoyEntero
  #"Personalizado agregado 9" = Table.AddColumn(#"Personalizado agregado 8", "HoyEnte
ro", each Number.RoundDown(Number.From(DateTime.LocalNow())), Int64.Type),
  // Creamos la columna Trimestre Actual
  #"Personalizado agregado 10" = Table.AddColumn(#"Personalizado agregado 9", "Trimes
treActual", each Date.QuarterOfYear(DateTime.LocalNow()), Int64.Type),
  // Creamos la columna Semestre
  #"Personalizado agregado 11" = Table.AddColumn(#"Personalizado agregado 10", "Semes
tre", each Number.RoundUp(Date.Month([Fecha])/6), Int64.Type),
  // Creamos la columna Desvio Mes
  #"Personalizado agregado 12" = Table.AddColumn(#"Personalizado agregado 11", "Desvi
oMes", each [MesNro]- [MesActual] + ([DesvioEjercicio]*12), Int64.Type),
  // Creamos la columna Desvio Dia
  #"Resta insertada" = Table.AddColumn(#"Personalizado agregado 12", "DesvioDia", eac
h [IdFechaEntero] - [HoyEntero], Int64.Type),
  // Creamos la columna HastaFecha
  #"Personalizado agregado 13" = Table.AddColumn(#"Resta insertada", "HastaFecha", ea
ch if [IdFechaEntero]<=[HoyEntero] then "PASADO"
else "FUTURO", type text),
  // Creamos la funcion TrimestreNro
  #"Trimestre insertado 1" = Table.AddColumn(#"Personalizado agregado 13", "Trimestre
Nro", each Date.QuarterOfYear([Fecha]), Int64.Type),
  // Creamos la columna DesvioTrimestre
  #"Personalizado agregado 14" = Table.AddColumn(#"Trimestre insertado 1", "DesvioTri
mestre", each [TrimestreNro]-[TrimestreActual]+([DesvioEjercicio]*4), Int64.Type),
  // Creamos la columna Nombredia en Español y en inglés
  #"Nombre del día insertado" = Table.AddColumn(#"Personalizado agregado 14", "Nombre
Dia", each Date.DayOfWeekName([Fecha]), type text),
  #"Nombre del día insertado 1" = Table.AddColumn(#"Nombre del día insertado", "NameD
ay", each Date.DayOfWeekName([Fecha], "en-US"), type text),
  // Creamos la columna Numero Ejercicio Fiscal
  #"Personalizado agregado 15" = Table.AddColumn(#"Nombre del día insertado 1", "NumE
jercicioFiscal", each if [MesNro]<PrimerMesFiscal then[Ejercicio]-
1 else [Ejercicio], Int64.Type),
  // Creamos la columna Numero Mes Fiscal
  #"Personalizado agregado 16" = Table.AddColumn(#"Personalizado agregado 15", "NumMe
sFiscal", each if [MesNro]<PrimerMesFiscal then
13 - PrimerMesFiscal + [MesNro]
else
[MesNro]-PrimerMesFiscal+1, Int64.Type),
  // Creamos la columna Numero trimestre fiscal
  #"Personalizado agregado 17" = Table.AddColumn(#"Personalizado agregado 16", "NumTr
imestreFiscal", each Number.RoundUp([NumMesFiscal]/3, 0), Int64.Type),
  // Creamos las etiquetas de las columnas fiscales
  #"Personalizado agregado 18" = Table.AddColumn(#"Personalizado agregado 17", "Ejerc
icio Fiscal", each "FY-" & Text.From([NumEjercicioFiscal]), type text),
```

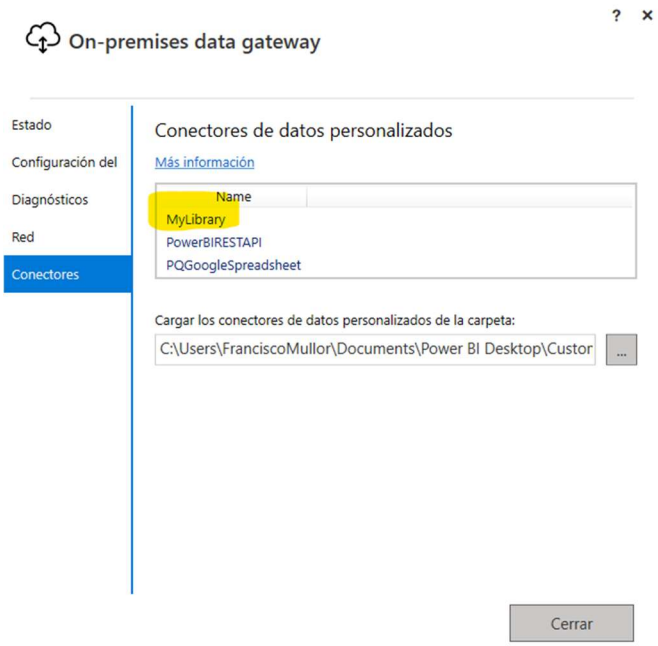
```
  #"Personalizado agregado 19" = Table.AddColumn(#"Personalizado agregado 18", "Mes F
iscal", each "FM-" &Text.End("0"&Text.From([NumMesFiscal]),2), type text),
  #"Personalizado agregado 20" = Table.AddColumn(#"Personalizado agregado 19", "Trime
stre Fiscal", each "FQ-" & Text.From([NumTrimestreFiscal]), type text),
  #"Personalizado agregado 21" = Table.AddColumn(#"Personalizado agregado 20", "Ejerc
icioMesFIscal", each "FYM-" & Text.From ([NumEjercicioFiscal]) &
Text.End("0"& Text.From([NumMesFiscal]),2), type text),
  #"Personalizado agregado 22" = Table.AddColumn(#"Personalizado agregado 21", "Ejerc
icioTrimestreFiscal", each "FYQ-
" & Text.From([NumEjercicioFiscal]) & Text.From([NumTrimestreFiscal]), type text)
in
  #"Personalizado agregado 22",
  documentation = [
        Documentation.Name =  //Nombre que vamos a dar a nuestra función
        " Crear.Calendario ",

        Documentation.Description = //Descripción

        " Creación de tabla completa de calendario con múltiples atributos, desvíos y
 calendario fiscal ",

        Documentation.LongDescription = //Descripción larga, aunque he puesto lo mism
o que en Descripción
        " Creación de tabla completa de calendario con múltiples atributos, desvíos y
 calendario fiscal ",

        Documentation.Category = " Table ", //Tipo de función

        Documentation.Source =  //Url donde podemos encontrarla
        " www.powerbisp.com ",

        Documentation.Version = " 1.0 ", //Version

        Documentation.Author = //Autor/-a
        " Francisco Mullor: www.powerbisp.com ",

        Documentation.Examples = //Ejemplos si los hubiera
        {
            [
                Description =  "  ",
                Code = "Creado por: Francisco Mullor ",
                Result = "  "
            ]
        }
    ]
in
  Value.ReplaceType( // Y debemos modificar el in que antes habíamos referenciado a l
a función, para que recoja la función y los metadatos de la siguiente manera:
        Funcion,
        Value.ReplaceMetadata(
            Value.Type( Funcion ),
            documentation)  // Lo que hacemos en este paso es introducir los metadato
s para que se vean al nombrar la función como documentación
    ) ;
```

9.3 Integración de la función personalizada en los Flujos de datos de Power BI

Una vez que se tiene el archivo MyLibrary.pqx si se quiere utilizar la función personalizada en el Power BI Desktop debe estar situado en una ruta concreta. En Documentos/Power BI Desktop/Custom Connectors. Estando en esa carpeta, ese archivo registra las funciones personalizadas como si de funciones nativas se tratase y son accesibles desde el Desktop una vez que se reinicia como se puede observar en la imagen:



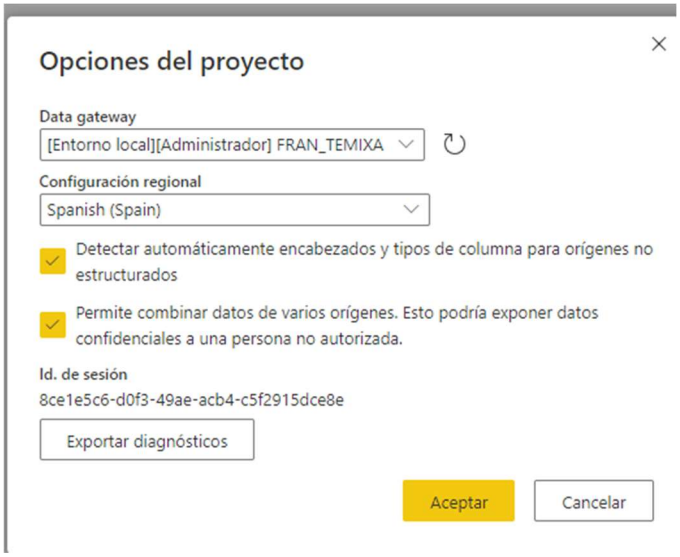
Para que funcione el mismo sistema en los flujos de datos y la función Crear.Calendario funcione correctamente, simplemente hay que hacer un pequeño ajuste en la puerta de enlace.



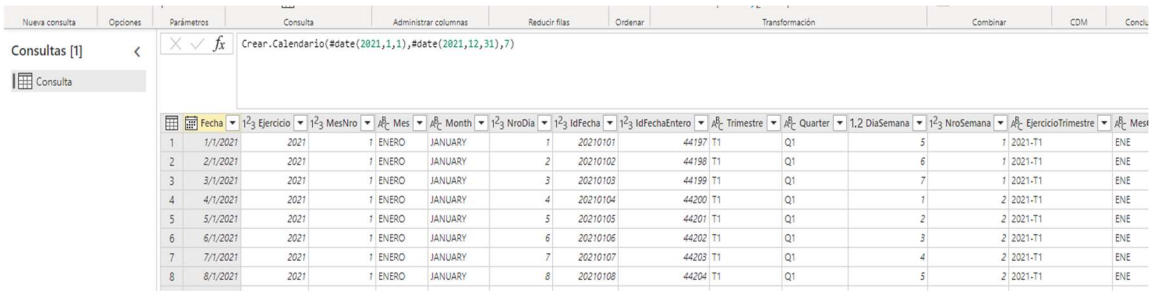
En la pestaña Conectores se debe indicar la ruta de donde cargará los conectores personalizados señalando la ruta donde tenemos nuestro archivo MyLibrary.pqx que en la imagen muestra la puerta de enlace como si de un conector más se tratase.

Con esto tan simple, se consigue que los conectores personalizados que integremos en ese fichero puedan utilizarse como si fueran funciones nativas desde los flujos de datos de Power BI de manera que seleccionando la puerta de enlace en nuestro proyecto:

EXTRACTO DEL LIBRO "POWER QUERY DATAFLOW" de
Francisco Mullor Cabrera
Fecha prevista publicación: 15/12/2021



Se podrá sin ningún problema utilizar la función que hemos creado e integrado en la librería:



	Fecha	Ejercicio	MesNro	Mes	Month	NroDia	IdFecha	IdFechaEntero	Trimestre	Quarter	DiaSemana	NroSemana	EjercicioTrimestre	Mes
1	1/1/2021	2021	1	ENERO	JANUARY	1	20210101	44197	T1	Q1	5	1	2021-T1	ENE
2	2/1/2021	2021	1	ENERO	JANUARY	2	20210102	44198	T1	Q1	6	1	2021-T1	ENE
3	3/1/2021	2021	1	ENERO	JANUARY	3	20210103	44199	T1	Q1	7	1	2021-T1	ENE
4	4/1/2021	2021	1	ENERO	JANUARY	4	20210104	44200	T1	Q1	1	2	2021-T1	ENE
5	5/1/2021	2021	1	ENERO	JANUARY	5	20210105	44201	T1	Q1	2	2	2021-T1	ENE
6	6/1/2021	2021	1	ENERO	JANUARY	6	20210106	44202	T1	Q1	3	2	2021-T1	ENE
7	7/1/2021	2021	1	ENERO	JANUARY	7	20210107	44203	T1	Q1	4	2	2021-T1	ENE
8	8/1/2021	2021	1	ENERO	JANUARY	8	20210108	44204	T1	Q1	5	2	2021-T1	ENE