En el lenguaje M de Power Query, las declaraciones each, if...then...else y los if anidados son componentes clave para escribir y entender las transformaciones de datos. Vamos a desglosarlos:

**1. Declaración each**

* **each** es una palabra clave en M que se usa para simplificar la escritura de funciones anónimas (funciones sin nombre) que se aplican a cada fila de una tabla o a cada elemento de una lista.
* **Contexto:** Cuando usas each, estás indicando que la expresión que sigue se aplica a cada uno de los elementos de la tabla o lista en el contexto en el que estás trabajando.

Ejemplo 1: Uso de each para agregar una columna calculada: Vamos a crear una tabla de ventas y agregar una nueva columna que muestre el precio total (Cantidad \* Precio).

**2. Declaración if...then...else**

* **if...then...else** es una estructura condicional que te permite realizar operaciones lógicas en función de ciertas condiciones.

**3. if Anidados**

if anidados se utilizan cuando necesitas evaluar múltiples condiciones de manera secuencial. Es básicamente un if...then...else dentro de otro if...then...else.

let

Source = Table.FromRecords({

[Producto = "Laptop", Cantidad = 5, Precio = 1000],

[Producto = "Mouse", Cantidad = 20, Precio = 300],

[Producto = "Teclado", Cantidad = 10, Precio = 50],

[Producto = "Audifonos", Cantidad = 70, Precio = 300],

[Producto = "Monitor", Cantidad = 15, Precio = 900],

[Producto = "CPU", Cantidad = 5, Precio = 3000],

[Producto = "Escritorio", Cantidad = 10, Precio = 20]

}),

AddTotalPriceColumn = Table.AddColumn(Source, "Precio Total", each [Cantidad] \* [Precio]),

AddPriceCategoryColumn = Table.AddColumn(Source, "Categoría Precio", each if [Precio] > 500 then "Alto" else "Bajo"),

AddPriceRangeColumn = Table.AddColumn(AddPriceCategoryColumn, "Rango de Precio", each if [Precio] > 500 then "Alto" else if [Precio] > 100 then "Medio" else "Bajo")

in

AddPriceRangeColumn

TRY OTHERWHISE

En el lenguaje M de Power Query, las declaraciones try...otherwise se utilizan para manejar errores de manera controlada. Esto te permite capturar errores en las expresiones y proporcionar un valor alternativo o una lógica diferente cuando ocurre un error.

let

Source = Table.FromRecords({

[Valor = 2],

[Valor = 0],

[Valor = 5]

}),

AddInverseColumn = Table.AddColumn(Source, "Inverso", each try 1 / [Valor] otherwise null)

in

AddInverseColumn

------------------------------------------------------------------------------------------------------------

**1. #duration**

* **Descripción:** Crea un valor de duración a partir de cuatro componentes: días, horas, minutos y segundos. Este valor de duración puede usarse para sumar o restar tiempo a fechas y horas.

let

DurationExample = #duration(5, 3, 15, 36)

in

DurationExample

**Resultado:** Esta expresión genera una duración de 5 días, 3 horas, 15 minutos y 36 segundos.

#### 2. Duration.Days

* **Descripción:** Calcula el número de días en una duración. Es útil para extraer la cantidad de días de un valor de duración.

let

DurationExample = #duration(10, 4, 30, 0),

NumberOfDays = Duration.Days(DurationExample)

in

NumberOfDays

**Funciones de Fecha y Hora**

#### 3. DateTime.Date

* **Descripción:** Extrae la parte de la fecha (sin la hora) de un valor date, datetime, o datetimezone.

let

DateTimeExample = #datetime(2024, 8, 22, 14, 30, 0),

DatePart = DateTime.Date(DateTimeExample)

in

DatePart

**4. DateTime.LocalNow**

**Descripción:** Devuelve el valor datetime actual según la configuración local del sistema.

let

CurrentDateTime = DateTime.LocalNow()

in

CurrentDateTime

**5. Date.Year**

* **Descripción:** Devuelve el componente de año de una fecha.

let

YearPart = Date.Year(DateTime.LocalNow())

in

YearPart

**6. Date.Month**

* **Descripción:** Devuelve el componente de mes de una fecha.

let

MonthPart = Date.Month(DateTime.LocalNow())

in

MonthPart

**Funciones de Lista**

**7. List.Date**

* **Descripción:** Genera una lista de fechas comenzando desde una fecha de inicio, con un incremento especificado.

let

StartDate = #date(2011, 12, 31),

DateList = List.Dates(StartDate, 5, #duration(1, 0, 0, 0))

in

DateList

**Resultado:** DateList generará una lista de 5 fechas comenzando desde 2011-12-31, incrementando cada fecha por un día:

Resumen

* **#duration** crea un valor de duración a partir de días, horas, minutos y segundos.
* **Duration.Days** extrae el número de días de una duración.
* **DateTime.Date** extrae solo la fecha de un valor datetime.
* **DateTime.LocalNow** devuelve la fecha y hora actuales del sistema.
* **Date.Year** y **Date.Month** extraen el año y el mes de una fecha, respectivamente.
* **List.Date** genera una lista de fechas a partir de una fecha inicial y un paso determinado.

Estos ejemplos deberían darte una comprensión clara de cómo utilizar estas funciones en tus transformaciones de datos en Power Query/M.

**Creación y Uso de Parámetros**

**1. Creación de un Parámetro en Power Query**

Puedes crear un parámetro desde el entorno de Power Query siguiendo estos pasos:

Abre el editor de Power Query.

Ve a la pestaña **Inicio** y selecciona **Administrar parámetros** > **Nuevo parámetro**.

Configura tu parámetro:

**Nombre**: El nombre del parámetro.

**Tipo de datos**: Define el tipo de datos del parámetro (número, texto, fecha, etc.).

**Valor actual**: El valor que usará el parámetro cuando se evalúe la consulta.

**Valor sugerido**: (Opcional) Puedes definir valores predeterminados o una lista de opciones.

**Valor predeterminado**: (Opcional) Especifica un valor predeterminado para el parámetro.

2. **Uso de Parámetros en Consultas**

Una vez que has creado un parámetro, puedes usarlo en tus consultas. A continuación, te muestro algunos ejemplos de cómo podrías usar parámetros:

**Ejemplo 1: Filtro basado en un parámetro Umbral = 50000**

let

Source = Table.FromRecords({

[Producto = "Laptop", Ventas = 100000],

[Producto = "Mouse", Ventas = 15000],

[Producto = "Teclado", Ventas = 200000],

[Producto = "Monitor", Ventas = 30000]

}),

FilteredTable = Table.SelectRows(Source, each [Ventas] > Umbral)

in

FilteredTable

**Ejemplo 2: Uso de un parámetro de fecha**

Supongamos que tienes un parámetro llamado StartDate y quieres filtrar una tabla para mostrar solo las ventas a partir de esa fecha.

let

Source = Table.FromRecords({

[Producto = "Laptop", FechaVenta = #date(2024, 3, 15)],

[Producto = "Mouse", FechaVenta = #date(2023, 12, 25)],

[Producto = "Teclado", FechaVenta = #date(2024, 1, 10)]

}),

FilteredTable = Table.SelectRows(Source, each [FechaVenta] >= Fecha)

in

FilteredTable

CONEXIÓN BD

let

ServerName = "mi\_servidor\_sql",

DatabaseName = "mi\_base\_de\_datos",

Query = "SELECT \* FROM Ventas",

Source = Sql.Database(ServerName, DatabaseName, [Query = Query])

in

Source

**Modificar Parámetros en el Editor de Power Query**

* **Modificar Parámetros Existentes:** Si necesitas cambiar el valor de un parámetro, puedes hacerlo directamente en el editor de Power Query, en la sección de **Parámetros**.
* **Reutilización:** Los parámetros pueden ser referenciados en cualquier parte de la consulta, lo que te permite cambiar un valor clave en un solo lugar y que ese cambio se propague a todas las partes de la consulta que dependen de ese parámetro.

**Resumen**

* Los **parámetros** en Power Query/M son herramientas poderosas para hacer que tus consultas sean más flexibles y fáciles de mantener.
* Puedes definir parámetros para valores como umbrales, fechas, o detalles de conexión, y luego usarlos en tus consultas para filtrar, calcular o conectarte a datos.
* Al cambiar un parámetro, puedes actualizar múltiples partes de tu consulta de una sola vez, lo que facilita la gestión de cambios y la reutilización de código.

Estos conceptos te permiten crear consultas dinámicas y escalables en Power Query, mejorando la eficiencia y la facilidad de mantenimiento en tus proyectos de análisis de datos.

EJERCICO 2.2

let

StartDate = #date(StartYear,1,1),

EndDate = #date(EndYear,12,31),

NumberOfDays = Duration.Days( EndDate - StartDate ),

Dates = List.Dates(StartDate, NumberOfDays+1, #duration(1,0,0,0)),

#"Convertida en tabla" = Table.FromList(Dates, Splitter.SplitByNothing(), null, null, ExtraValues.Error),

#"Columnas con nombre cambiado" = Table.RenameColumns(#"Convertida en tabla",{{"Column1", "FullDateAlternateKey"}}),

#"Tipo cambiado" = Table.TransformColumnTypes(#"Columnas con nombre cambiado",{{"FullDateAlternateKey", type date}}),

#"Año insertado" = Table.AddColumn(#"Tipo cambiado", "Año", each Date.Year([FullDateAlternateKey]), Int64.Type),

#"Mes insertado" = Table.AddColumn(#"Año insertado", "Mes", each Date.Month([FullDateAlternateKey]), Int64.Type),

#"Nombre mes insertado" = Table.AddColumn(#"Mes insertado", "Nombre de mes", each Date.MonthName([FullDateAlternateKey]), type text),

#"Trimestre insertado" = Table.AddColumn(#"Nombre mes insertado", "Trimestre", each Date.QuarterOfYear([FullDateAlternateKey]), Int64.Type),

#"Trimestre del año insertado" = Table.AddColumn( #"Trimestre insertado", "Trimestre del año", each Date.WeekOfYear([FullDateAlternateKey]), Int64.Type),

#"Semana del mes insertado" = Table.AddColumn(#"Trimestre del año insertado", "Semana del mes", each Date.WeekOfMonth([FullDateAlternateKey]), Int64.Type),

#"Dia insertado" = Table.AddColumn(#"Semana del mes insertado" , "Dia", each Date.Day([FullDateAlternateKey]), Int64.Type),

#"Dia de la semana insertado" = Table.AddColumn(#"Dia insertado", "Dia de la semana", each Date.DayOfWeek([FullDateAlternateKey]), Int64.Type),

#"Dia del año insertado" = Table.AddColumn(#"Dia de la semana insertado", "Dia del año", each Date.DayOfYear([FullDateAlternateKey]), Int64.Type),

#"Nombre de dia insertado" = Table.AddColumn(#"Dia del año insertado", "Nombre de dia", each Date.DayOfWeekName([FullDateAlternateKey]), type text)

in

#"Nombre de dia insertado"

DEFINIMOS PARÁMETROS AÑO INICIO 2000 A 2030

1. **Definir las fechas de inicio y fin**

let

StartDate = #date(StartYear, 1, 1),

EndDate = #date(EndYear, 12, 31),

* **StartDate**: Define una fecha de inicio usando el año StartYear y fija el 1 de enero como el día y mes.
* **EndDate**: Define una fecha de fin usando el año EndYear y fija el 31 de diciembre como el día y mes.

**Nota**: StartYear y EndYear serían parámetros que defines antes de este código o se pasan al código.

2. Calcular la cantidad de días entre las fechas

NumberOfDays = Duration.Days(EndDate - StartDate),

**NumberOfDays**: Calcula el número de días entre StartDate y EndDate. Se resta EndDate de StartDate y se usa la función Duration.Days para obtener el número total de días.

3. **Generar la lista de fechas**

Dates = List.Dates(StartDate, NumberOfDays+1, #duration(1, 0, 0, 0)),

 **Dates**: Genera una lista de fechas empezando desde StartDate. La lista tendrá NumberOfDays + 1 elementos (esto incluye el último día, por eso se suma 1).

 **#duration(1, 0, 0, 0)**: Especifica un incremento de 1 día entre cada fecha en la lista.

4. **Convertir la lista en una table**

#"Convertida en tabla" = Table.FromList(Dates, Splitter.SplitByNothing(), null, null, ExtraValues.Error),

 **#"Convertida en tabla"**: Convierte la lista Dates en una tabla. Cada fecha se coloca en una columna llamada "Column1".

 **Splitter.SplitByNothing()**: No divide los valores (los deja tal como están).

 **ExtraValues.Error**: Genera un error si hay valores adicionales inesperados (no debería pasar en este caso).

5. **Cambiar el nombre de la columna**

#"Columnas con nombre cambiado" = Table.RenameColumns(#"Convertida en tabla", {{"Column1", "FullDateAlternateKey"}}),

**#"Columnas con nombre cambiado"**: Renombra la columna "Column1" a "FullDateAlternateKey".

**6. Cambiar el tipo de datos de la columna**

m

Copiar código

#"Tipo cambiado" = Table.TransformColumnTypes(#"Columnas con nombre cambiado", {{"FullDateAlternateKey", type date}}),

* **#"Tipo cambiado"**: Cambia el tipo de datos de la columna "FullDateAlternateKey" a tipo date.

**7. Agregar la columna del año**

m

Copiar código

#"Año insertado" = Table.AddColumn(#"Tipo cambiado", "Año", each Date.Year([FullDateAlternateKey]), Int64.Type),

* **#"Año insertado"**: Agrega una nueva columna "Año" que extrae el año de la fecha en "FullDateAlternateKey" usando Date.Year.
* **Int64.Type**: Especifica que el tipo de la nueva columna será Int64 (un número entero de 64 bits).

**8. Agregar la columna del mes**

m

Copiar código

#"Mes insertado" = Table.AddColumn(#"Año insertado", "Mes", each Date.Month([FullDateAlternateKey]), Int64.Type),

* **#"Mes insertado"**: Agrega una columna "Mes" que extrae el mes de "FullDateAlternateKey" usando Date.Month.

**9. Agregar la columna del nombre del mes**

m

Copiar código

#"Nombre mes insertado" = Table.AddColumn(#"Mes insertado", "Nombre de mes", each Date.MonthName([FullDateAlternateKey]), type text),

* **#"Nombre mes insertado"**: Agrega una columna "Nombre de mes" que devuelve el nombre del mes (por ejemplo, "January") usando Date.MonthName.
* **type text**: Especifica que el tipo de la nueva columna será text (texto).

**10. Agregar la columna del trimestre**

m

Copiar código

#"Trimestre insertado" = Table.AddColumn(#"Nombre mes insertado", "Trimestre", each Date.QuarterOfYear([FullDateAlternateKey]), Int64.Type),

* **#"Trimestre insertado"**: Agrega una columna "Trimestre" que devuelve el trimestre del año (1, 2, 3, o 4) usando Date.QuarterOfYear.

**11. Agregar la columna del trimestre del año**

m

Copiar código

#"Trimestre del año insertado" = Table.AddColumn(#"Trimestre insertado", "Trimestre del año", each Date.WeekOfYear([FullDateAlternateKey]), Int64.Type),

* **#"Trimestre del año insertado"**: Agrega una columna "Trimestre del año" que devuelve la semana del año en la fecha dada usando Date.WeekOfYear.

**12. Agregar la columna de la semana del mes**

m

Copiar código

#"Semana del mes insertado" = Table.AddColumn(#"Trimestre del año insertado", "Semana del mes", each Date.WeekOfMonth([FullDateAlternateKey]), Int64.Type),

* **#"Semana del mes insertado"**: Agrega una columna "Semana del mes" que devuelve la semana del mes en la fecha dada usando Date.WeekOfMonth.

**13. Agregar la columna del día**

m

Copiar código

#"Dia insertado" = Table.AddColumn(#"Semana del mes insertado", "Dia", each Date.Day([FullDateAlternateKey]), Int64.Type),

* **#"Dia insertado"**: Agrega una columna "Dia" que devuelve el día del mes en la fecha dada usando Date.Day.

**14. Agregar la columna del día de la semana**

m

Copiar código

#"Dia de la semana insertado" = Table.AddColumn(#"Dia insertado", "Dia de la semana", each Date.DayOfWeek([FullDateAlternateKey]), Int64.Type),

* **#"Dia de la semana insertado"**: Agrega una columna "Dia de la semana" que devuelve el día de la semana (0 para domingo, 1 para lunes, etc.) usando Date.DayOfWeek.

**15. Agregar la columna del día del año**

m

Copiar código

#"Dia del año insertado" = Table.AddColumn(#"Dia de la semana insertado", "Dia del año", each Date.DayOfYear([FullDateAlternateKey]), Int64.Type),

* **#"Dia del año insertado"**: Agrega una columna "Dia del año" que devuelve el número del día en el año usando Date.DayOfYear.

**16. Agregar la columna del nombre del día**

m

Copiar código

#"Nombre de dia insertado" = Table.AddColumn(#"Dia del año insertado", "Nombre de dia", each Date.DayOfWeekName([FullDateAlternateKey]), type text)

* **#"Nombre de dia insertado"**: Agrega una columna "Nombre de dia" que devuelve el nombre del día de la semana (por ejemplo, "Monday") usando Date.DayOfWeekName.

**17. Resultado final**

m

Copiar código

in

#"Nombre de dia insertado"

* **Resultado final:** Esta es la tabla completa con todas las columnas añadidas y transformaciones realizadas. Es el resultado final que devuelve la consulta.

**Resumen**

Este código crea una tabla de fechas que abarca desde una fecha de inicio (StartDate) hasta una fecha de fin (EndDate). Luego, la tabla se enriquece con columnas adicionales que detallan varios aspectos de cada fecha, como el año, mes, trimestre, nombre del mes, día, y más. Cada paso agrega o transforma la información de la tabla, lo que te proporciona una tabla completa con detalles sobre cada fecha en el rango especificado.

FUNCIONES PERSONALIZADAS:

En el lenguaje M de Power BI, las funciones personalizadas te permiten encapsular lógica o cálculos repetitivos en una función reutilizable, lo que puede hacer tu código más limpio, modular y fácil de mantener. Aquí te explico cómo funcionan y cómo puedes crear una función personalizada en M.

**1. Definición de una Función Personalizada**

Una función personalizada en M se define usando la sintaxis básica:

m

Copiar código

(nombre\_argumento1 as tipo1, nombre\_argumento2 as tipo2, ...) => expresión

* **nombre\_argumento**: Es el nombre del parámetro que la función aceptará.
* **tipo**: Es el tipo de dato que se espera para ese parámetro (puede ser opcional).
* **expresión**: Es el cuerpo de la función, lo que la función ejecutará cuando sea llamada.

**2. Ejemplo Sencillo**

Supongamos que quieres crear una función que sume dos números:

Se debe guardar la consulta con ese nombre Sumar

let

Sumar = (a as number, b as number) => a + b

in

Sumar

**Organizar el Uso de Funciones**

* **Mantén la función en una consulta separada**: Esto te permite reutilizar la función Sumar en cualquier parte de tu modelo de datos. Solo necesitas referenciarla por su nombre (FunciónSumar en este caso).
* **Invoca la función donde la necesites**: En cualquier otra consulta, simplemente invoca la función pasando los parámetros necesarios.

Ejecutamos

let

Origen = FuncionSumar(10, 20)

in

Origen

**Crear la Función Personalizada**

Primero, definiremos la función CalculaImpuesto que tomará un precio y una tasa de impuesto como parámetros.

let

TablaOriginal = Table.FromRecords({

[Producto = "Producto A", Precio = 100],

[Producto = "Producto B", Precio = 200],

[Producto = "Producto C", Precio = 300]

})

in

TablaOriginal

CREAR FUNCIÓN

let

CalculaImpuesto = (precio as number, tasa as number) => precio \* tasa

in

CalculaImpuesto

RESULTADO

let

TablaOriginal = Table.FromRecords({

[Producto = "Producto A", Precio = 100],

[Producto = "Producto B", Precio = 200],

[Producto = "Producto C", Precio = 300]

}),

TablaTransformada = Table.AddColumn(TablaOriginal, "Impuesto", each FunciónCalculaImpuesto([Precio], 0.18))

in

TablaTransformada