mentorBI
finanzas

17 fórmulas DAX imprescindibles

Una guía para dominar DAX

Las fórmulas que encontrarás en este documento son mucho más que simples ejemplos; son plantillas de cálculo probadas y optimizadas que puedes aplicar directamente en tus propios proyectos. Considéralas como soluciones listas para usar que puedes copiar, pegar y adaptar a tu modelo de datos con solo cambiar los nombres de las tablas y las columnas por los de tu propio modelo.

Dominar estos 17 patrones te dará una base sólida y versátil. De hecho, con este conjunto de plantillas, estarás en disposición de resolver entre un 60 % y un 80 % de los cálculos y medidas que puedas necesitar en tu día a día con Power BI, acelerando tu trabajo y asegurando resultados precisos y eficientes.

Es importante destacar que los ejemplos se basan en un modelo de datos común, con tablas de ventas, clientes y productos, y columnas como 'unidades' o 'precio por unidad'. El verdadero poder de estas plantillas reside en tu capacidad para extrapolarlas a tu realidad: simplemente, adapta los nombres de las tablas y columnas a los que utilizas en tu propio modelo de datos para que los cálculos funcionen en tu contexto específico.

#01 Ud vendidas

```
Ud vendidas =
SUM( factPedidos[Ud] )
```

Explicación:

El objetivo de esta medida es agregar los valores de la columna [Ud] de la tabla factPedidos. Toda expresión DAX se evalúa en un **contexto de evaluación**. La función SUM, al igual que otras funciones de agregación simples (como AVERAGE, MIN, MAX), opera dentro de un **contexto de filtro**. Esto significa que antes de realizar la suma, el motor de DAX examina todas las filas de la tabla factPedidos que son visibles bajo los filtros activos en el informe (procedentes de segmentaciones, filas de una matriz, otros gráficos, etc.). La agregación solo se aplica sobre los valores de la columna especificada en esas filas visibles.

#02 Total ventas €

```
Total ventas € =
SUMX(
   factPedidos,
   factPedidos[Ud] * factPedidos[precioVentaUd]
)
```

Explicación:

El objetivo es calcular los ingresos totales, que requieren una operación a nivel de fila antes de la agregación final. Aquí, SUM no es suficiente, ya que necesitamos multiplicar dos columnas fila por fila. Para ello, usamos SUMX, una **función iteradora**.

Las funciones iteradoras operan en dos contextos: **Contexto de Filtro** (su primer argumento, la tabla factPedidos, ya filtrada por el informe) y **Contexto de Fila** (creado para evaluar la expresión del segundo argumento, factPedidos[Ud] * factPedidos[precioVentaUd], para cada fila de esa tabla). Finalmente, SUMX suma los resultados de cada fila para devolver un único valor.

#03 Total peso de las ventas

```
Total peso de las ventas =
SUMX(
   factPedidos,
   factPedidos[ud] * RELATED( dimProductos[peso] )
)
```

Explicación:

Este cálculo requiere datos de tablas relacionadas dentro de una iteración. La función RELATED opera dentro de un contexto de fila (creado por SUMX). Le permite "salir" de la tabla que se está iterando (factPedidos) y, siguiendo una relación existente de "varios a uno", recuperar un valor de una tabla relacionada (dimProductos). En cada fila de factPedidos, RELATED busca el producto correspondiente en dimProductos y devuelve su peso, permitiendo la multiplicación.

#04 Resultado condicionado

```
Result. condicionado =
VAR _udVendidas =   SUM( factPedidos[Ud] )
VAR _result =       IF(
                        _udVendidas > 22,
                        _udVendidas,
                        "Inferior"
                    )
RETURN              _result
```

Explicación:

El uso de variables (VAR/RETURN) es fundamental por legibilidad, rendimiento (el cálculo se hace una vez) y depuración. La función IF evalúa una condición y devuelve un resultado u otro. Es importante saber que IF no modifica el contexto, solo dirige el flujo del cálculo. Para múltiples condiciones anidadas, es preferible y más legible usar la función SWITCH.

#05 N.º días con ventas

```
N.º días con ventas =
VAR _tabla =    ALL( factPedidos[FechaPedido] )
VAR _result =   COUNTROWS( _tabla )
RETURN          _result
```

Explicación:

El objetivo es contar los días distintos con ventas, sin importar los filtros de fecha del informe. La clave es la función ALL, que devuelve una tabla con todos los valores únicos de una columna, **ignorando cualquier filtro activo sobre ella**. Finalmente, COUNTROWS simplemente cuenta las filas de la tabla virtual generada por ALL, dándonos el recuento de días únicos.

#06 N.º días laborables con ventas

```
N.º días laborables con ventas =
VAR _tabla =      FILTER(
                     ALL( factPedidos[FechaPedido] ),
                     WEEKDAY( factPedidos[FechaPedido], 2 ) <= 5
                  )
VAR _result =     COUNTROWS( _tabla )
RETURN            _result
```

Explicación:

Esta medida introduce FILTER, otra función iteradora. Itera sobre una tabla (el resultado de ALL(factPedidos[FechaPedido])) y aplica una condición fila por fila. Devuelve una nueva tabla solo con las filas que cumplen la condición. En este caso, filtra las fechas para quedarse solo con los días laborables (lunes a viernes). COUNTROWS cuenta las filas de esta nueva tabla filtrada.

#07 y #08 Ud vendidas 1er sem.

Sintaxis completa (explícita)

```
Ud vendidas 1er sem. =
CALCULATE(
    SUM( factPedidos[ud] ),
    FILTER(
        ALL( dimCalendario[semestre] ),
        dimCalendario[semestre] = "Semestre 1"
    )
)
```

Syntax Sugar (recomendada)

```
Ud vendidas 1er sem. (syntax sugar) =
CALCULATE(
   SUM( factPedidos[ud] ),
   dimCalendario[semestre] = "Semestre 1"
)
```

Explicación:

CALCULATE es la función más importante de DAX. Su propósito es **modificar el contexto de filtro** antes de evaluar una expresión. La segunda versión usa una sintaxis simplificada llamada **"Syntax Sugar"**. Cuando CALCULATE recibe una expresión booleana simple como filtro, la envuelve implícitamente en FILTER(ALL(...), ...), haciendo el código más limpio. Ambas medidas son funcionalmente idénticas y calculan las ventas del "Semestre 1" ignorando cualquier filtro externo sobre semestres.

#09 Ud vendidas mes anterior

```
Ud vendidas mes anterior =
CALCULATE(
   SUM( factPedidos[ud] ),
   DATEADD(
      dimCalendario[fecha],
      -1,
      MONTH
   )
)
```

Explicación:

Un patrón clásico de **Inteligencia de Tiempo**. Las funciones de inteligencia de tiempo como DATEADD devuelven una tabla de fechas modificada. DATEADD toma las fechas del contexto actual, se desplaza hacia atrás un intervalo (en este caso, 1 mes) y devuelve un nuevo conjunto de fechas. CALCULATE usa esa nueva tabla de fechas como filtro, calculando así el resultado para el mes anterior. Es imprescindible tener una tabla de calendario marcada como tal en el modelo.

#10 Denominador para %

```
Denominador para % =
CALCULATE(
    SUM( factPedidos[ud] ),
    ALLSELECTED( dimCalendario[Trimestre] )
)
```

Explicación:

ALLSELECTED es una de las funciones más sutiles. Su propósito es eliminar el filtro de una columna, pero con una excepción clave: respeta los filtros que provienen de fuera del objeto visual actual. Dicho de otro modo, elimina el filtro de la celda de la matriz (dimCalendario[Trimestre]), pero mantiene los filtros de los segmentadores. Esto lo hace la herramienta perfecta para crear denominadores para cálculos de "porcentaje del total visible".

#11 Porc. ud vendidas sem. / total

```
Porc. ud vendidas sem. / total =
DIVIDE(
    SUM( factPedidos[ud] ),
    CALCULATE(
        SUM( factPedidos[ud] ),
        ALLSELECTED(dimCalendario[Trimestre])
    )
)
```

Explicación:

Este es el patrón clásico de "ratio-to-parent". El numerador se calcula en el contexto de filtro natural de la celda. El denominador usa la lógica de la medida #10 para calcular el total para todos los trimestres visibles. El uso de DIVIDE es una buena práctica indispensable para manejar de forma segura los casos de división por cero.

#12 Máx suma ud vendidas/cliente

```
Máx suma ud vendidas/cliente =
MAXX(
   VALUES( dimClientes[ID_cliente] ),
   CALCULATE( SUM( factPedidos[ud] ) )
)
```

Explicación (Transición de Contextos):

Este es un ejemplo canónico de **transición de contextos**. La función iteradora MAXX crea un contexto de fila para cada cliente. La presencia de CALCULATE dentro de ese contexto de fila **desencadena la transición**: convierte el valor actual del contexto de fila (el ID_cliente de esa iteración) en un nuevo contexto de filtro equivalente. Por lo tanto, para cada cliente, se calcula la suma total de sus ventas. Finalmente, MAXX devuelve la suma más alta entre todos los clientes. Este concepto es clave para cálculos avanzados.

#13 Acum fecha-mesAño

```
Acum fecha-mesAño =
VAR _fechaMax =     MAX( dimCalendario[fecha] )
VAR _resultado =    CALCULATE(
                        [#01 Ud vendidas],
                        FILTER(
                            ALLSELECTED(
                                dimCalendario[Fecha],
                                dimCalendario[Semestre]
                            ),
                            dimCalendario[fecha] <= _fechaMax
                        )
                    )
RETURN              _resultado
```

Explicación:

Este es el patrón canónico para un total acumulado. _fechaMax captura la última fecha visible en el contexto de filtro actual. FILTER, con ALLSELECTED, le da una vista de todas las fechas y semestres en la selección actual. La condición dimCalendario[fecha] <= _fechaMax crea el nuevo contexto de filtro. La inclusión de dimCalendario[Semestre] en ALLSELECTED hace que el acumulado se reinicie para cada semestre. Para un acumulado continuo, solo se usaría ALLSELECTED(dimCalendario[Fecha]).

#14 Variación porcentual

```
Variación porcentual =
VAR _act =      SUM( factPedidos[Ud] )
VAR _ant =      CALCULATE(
                    SUM( factPedidos[Ud] ),
                    DATEADD( dimCalendario[fecha], -1, MONTH )
                )
VAR _result =   SWITCH(
                    TRUE(),
                    _ant <> 0 && _act <> 0, DIVIDE( _act - _ant, ABS(_ant) ),
                    _act = _ant,            BLANK(),
                    _act - _ant > 0,        1,
                    -1
                )
RETURN          _result
```

Explicación:

Este es un patrón de variación porcentual robusto. La clave no está solo en la fórmula matemática, sino en el manejo de casos límite con SWITCH(TRUE(), ...): previene la división por cero y gestiona correctamente el crecimiento/decrecimiento desde/hacia cero devolviendo 1 (100%) o -1 (-100%). El uso de ABS en el denominador es el estándar profesional.

#15 Rotación inventario

```
Rotación inventario =
VAR _salidasProd =  CALCULATE(
                        SUM( factMovAlmacen[ud] ) * -1,
                        factMovAlmacen[ud] < 0
                    )
VAR _fechaMax =     MAX(dimCalendario[Fecha])
VAR _saldoMedio =   AVERAGEX(
                        VALUES( dimCalendario[Fecha] ),
                        CALCULATE(
                            SUM( factMovAlmacen[ud] ),
                            dimCalendario[Fecha] <= _fechaMax
                        )
                    )
VAR _result =       DIVIDE(
                        _salidasProd,
                        _saldoMedio
                    )
RETURN              _result
```

Explicación:

Esta versión calcula la rotación de inventario de forma precisa. El numerador representa las salidas. El denominador calcula el promedio diario real del inventario usando AVERAGEX para iterar sobre cada día y un CALCULATE con un patrón de total acumulado para obtener el saldo exacto en esa fecha. La división final proporciona una métrica de rendimiento fiable.

#16 Ud vendidas enviadas

```
Ud vendidas enviadas =
CALCULATE(
   SUM( factPedidos[Ud] ),
   USERELATIONSHIP(
      dimCalendario[Fecha],
      factPedidos[FechaEnvio]
   )
)
```

Explicación:

En un modelo solo puede haber una relación activa entre dos tablas. Si existen relaciones adicionales (ej. por fecha de pedido y fecha de envío), deben estar inactivas. USERELATIONSHIP es un modificador de CALCULATE que **activa una relación inactiva** para la duración del cálculo. En esta medida, le dice a CALCULATE que ignore la relación activa y use la inactiva con FechaEnvio para propagar los filtros desde el calendario.

#17 Prod. > 235gr en cada almacén

```
Prod. > 235 gr usados en cada almacén =
CALCULATE(
    CALCULATE(
        COUNTROWS(dimProductos),
        dimProductos[peso] > 235
    ),
    CROSSFILTER(
      dimProductos[ID_producto],
      factMovAlmacen[ID_producto],
      Both
   )
)
```

Explicación:

Por defecto, los filtros se propagan del lado "uno" al "varios" de una relación. CROSSFILTER es un modificador de CALCULATE que **cambia la dirección del filtro** durante la medida. Aquí, CROSSFILTER(..., Both) hace que la relación sea temporalmente bidireccional. Esto permite que un filtro en Almacén fluya a la tabla de hechos y luego "suba" a Productos para contar los correctos. Es la práctica recomendada para evitar los problemas de rendimiento de las relaciones bidireccionales permanentes.

Documento elaborado por Clara Vega y José Manuel Pomares

mentorBI finanzas