stackfuel

DAX (Data Analysis Expressions)

DAX ist die Formelsprache von Power BI. Sie beinhaltet viele verschiedene Operatoren und Funktionen, mit denen du Formeln und Ausdrücke für selbst berechnete Spalten oder Measures erstellen kannst. Diese Übersicht gibt dir einen Überblick über die DAX-Syntax und die im Training verwendeten DAX-Operatoren und DAX-Funktionen. Weitere DAX-Operatoren und DAX-Funktionen sowie Updates und ganz neue DAX-Funktionen findest du in der Microsoft-Dokumentation beschrieben: https://docs.microsoft.com/de-de/dax/

Berechnete Spalte oder Measure?

Bei jeder Berechnung stellt sich die Frage: Spalte oder Measure? Diese Tabelle stellt ihre Eigenschaften gegenüber und soll dir so die Entscheidung erleichtern.

Eigenschaft	Berechnete Spalte	Measure
Berechnung	Definitionszeitpunkt, Auswertung pro Zeile	Live, Auswertung im jeweiligen Kontext
Granularität	Zeile	je nach Kontext
Datenquelle	• Rohdaten • Berechnete Spalten	• Rohdaten • Berechnete Spalten • Andere Measures
Ressourcenbedarf	Speicherplatz im Datenmodell	Rechenleistung bei Nutzung bzw. Veränderung des Kontextes
Verwendung	• Vorberechnung komplexer Werte • Verwendung als Datenschnitt, seiten- und berichtsweiter Filter, Visual-Achse	• Aggregation, insbesondere laufende Werte, Verhältnisse • Kontextabhängige Berechnungen, z. B. Verkäufe für ausgewählte Region

DAX-Syntax

Die DAX-Syntax beschreibt den allgemeinen Aufbau von DAX-Ausdrücken, z.B. eines Measures. Jeder Ausdruck beginnt mit einem Gleichheitszeichen (`=`). Davor steht der Name des Measures (`sum impressions`). Dahinter steht der Ausdruck, der aus einem oder mehreren verknüpften Elementen verknüpft bestehen kann. Das Ergebnis der Berechnung wird an das Measure übergeben. Im Ausdruck werden Tabellen mit ihrem Namen referenziert (`fact_posts`) und Spalten über den Tabellennamen und Spaltennamen in eckigen Klammern (`fact_posts[impressions]`). Tabellen oder Spalten können über Funktionen oder Operatoren verändert oder miteinander verknüpft werden. Hinter dem Funktionsnamen stehen runde Klammern, die die Funktionsargumente einklammern (`SUM(fact_posts[impressions])`). Was hinter doppelten Schrägstrichen (`//`) folgt, dient nur der Dokumentation mit Kommentaren und geht nicht in den DAX-Ausdruck mit ein. Kommentare können auch über mehrere Zeilen gehen (`/*…*/`).

```
sum impressions = SUM(fact_posts[impressions])

// comment
/* multiline
comment */
```

stackfuel

DAX-Operatoren

In der DAX-Sprache werden Operatoren dazu genutzt, um Texte zu verbinden, Werte zu vergleichen, mathematische Berechnungen durchzuführen oder logische Ausdrücke zu erstellen.

`Textoperator: &` verbindet zwei Texte miteinander
Beispiel: `Bewegungen[Erst.Dat] & " " & Bewegungen[Erst.Zt]`

Vergleichsoperatoren vergleichen zwei Werte miteinander:

Operator	Bedeutung	Beispiel
>, >=	Größer als, größer oder gleich	Follower[date_id] >= 15
<, <=	Kleiner als, kleiner oder gleich	Follower[Facebook total followers] > 50
=, <>	Gleich, ungleich	dim_account[platform] = "Facebook"

Mathematische Operatoren verbinden Werte in mathematischen Rechnungen miteinander:

Operator	Bedeutung	Beispiel
+, -	Addition, Subtraktion (bzw. Vorzeichen)	SUM(Follower[LinkedIn monthly follower difference]) + 841
*, /	Multiplikation, Division	SUM(fact_posts[impressions]) * 2
^	Potenz	SUM(table_exp[base]) ^ 2

Logische Operatoren verbinden logische Ausdrücke miteinander und geben einen logischen Wert zurück:

Operator	Bedeutung	Beispiel
&&	UND: zwei wahre Aussagen mit UND verknüpft geben eine wahre Aussage zurück (**TRUE**), sonst eine falsche (**FALSE**)	RELATED(dim_account[platform]) = "Facebook" && fact_posts[date_id].[Jahr] = 2022
||	ODER: wenn mindestens eine Aussage wahr ist, gibt die ODER-Verknüpfung eine wahre Aussage zurück (**TRUE**); zwei falsche Aussagen mit ODER verknüpft geben eine falsche Aussage zurück (**FALSE**)	act_posts[date_id].[Jahr] = 2021 || fact_posts[date_id].[Jahr] = 2022
IN	IN: gibt eine wahre Aussage zurück (**TRUE**), wenn ein Wert in einer Liste von Werten enthalten ist; sonst wird eine falsche Aussage zurückgegeben (**FALSE**)	fact_posts[date_id].[Jahr] IN {2021,2021}

stackfuel

Aggregationsfunktionen

Alle Zeilen einer Spalte oder Tabelle werden über Aggregationsfunktionen zu einem Wert zusammengefasst.

`SUM`: Addition
Beispiel: `SUM(fact_posts[impressions])`

`MIN`: Minimum; MAX: Maximum
Beispiel: `MIN(Follower[date_id])`

`COUNT`: Anzahl der Zeilen einer Spalte, die nicht leer sind
Beispiel: `COUNT(fact_posts_monthly[Post Count Target Reached])`

`DISTINCTCOUNT`: Anzahl an unterschiedlichen Werten in einer Spalte
Beispiel: `DISTINCTCOUNT(Follower[date_id])`

`COUNTROWS`: Anzahl an Zeilen in einer Tabelle
Beispiel: `COUNTROWS(churn)`

Mathematische Funktionen

Für mathematische Berechnungen werden neben DAX-Operatoren (siehe oben) auch DAX-Funktionen verwendet.

`DIVIDE:` Führt eine Division durch, bei einer Division durch 0 wird `BLANK()` zurückgegeben

Syntax: `DIVIDE(<numerator>, <denominator> [,<alternateresult>])`
Beispiel:

```
DIVIDE(
        SUM(fact_posts_monthly[Post Count Target Reached]),
        COUNT(fact_posts_monthly[Post Count Target Reached])
    )
```

`ROUNDUP:` Rundet eine Zahl auf

Syntax: `ROUNDUP(<number>, <num_digits>)`
Beispiel: `ROUNDUP(__FOLLOWER_2021 * (1 + __FOLLOWER_2022_TARGET_GAIN_PERCENT * YEARFRAC(__END_2021, __CURRENT_DATE, 1)),0)`

stackfuel

Datums- und Zeitfunktionen

Für Berechnungen mit Datums- und Zeitspalten hat die DAX-Sprache eigene Funktionen.

> DATE: Gibt das angegeben Datum im datetime-Format zurück
>
> Syntax: `DATE(<year>, <month>, <day>)`
> Beispiel: `DATE(2021, 12, 31)`
>
> DATEADD: Verschiebt das Datum um ein angegebenes Intervall
>
> Syntax: `DATEADD(<dates>,<number_of_intervals>,<interval>)`
> Beispiel: `DATEADD('Follower'[date_id].[Date], -1, MONTH)`
>
> DATEDIFF: Gibt das Zeitintervall zwischen zwei Daten zurück
>
> Syntax: `DATEDIFF(<Date1>, <Date2>, <Interval>)`
> Beispiel: `DATEDIFF(Bewegungen[Erst],Bewegungen[Quit],HOUR`)
>
> WEEKNUM: Gibt die Kalenderwoche für ein Datum zurück
>
> Syntax: `WEEKNUM(<date>[, <return_type>])`
> Beispiel: `WEEKNUM(Bewegungen[Erst.Dat])`
>
> WEEKDAY: Gibt den Wochentag eines Datums zurück (Zahl zwischen 1 und 7; Bedeutung je nach angegebenem `return_type`).
>
> Syntax: `WEEKDAY(<date>, <return_type>)`
> Beispiel: `WEEKDAY(Bewegungen[Erst.Dat],2)`

Beziehungsfunktionen

Um in einer DAX-Formel Werte aus unterschiedlichen Tabellen zu nutzen, werden Beziehungsfunktionen verwendet. Sie verwalten die Beziehungen zwischen den Tabellen.

RELATED: gibt den zugehörigen Wert aus einer anderen Tabelle zurück

> Beispiel: `FILTER(ALL(fact_posts_monthly), RELATED(dim_account[platform]) = "Facebook" && fact_posts_monthly[date_id].[Jahr] = 2021)`

stackfuel

Logische Funktionen und Informationsfunktionen

Um Informationen über Werte zu erhalten oder Bedingungen in einem Ausdruck zu stellen, benutzt du Informationsfunktionen und Logische Funktionen. Dabei wird ein Wert oder eine Bedingung überprüft und ein logischer Wert wird zurückgegeben. Dieser logische Wert gibt dann an, ob eine Bedingung stimmt oder ein Wert mit einer Erwartung übereinstimmt. In Power BI heißt der logische Wert „wahr“ `TRUE` und der logische Wert „falsch“ heißt `FALSE`.

`IF`: Die IF-Funktion wird für Fallunterscheidungen eingesetzt. Bei der Erstellung eines Measures möchte man vielleicht für verschiedene Fälle unterschiedliche Werte festlegen. Die IF-Funktion überprüft eine Bedingung bzw. mehrere über logische Operatoren verknüpfte Bedingungen. Ist die Bedingung wahr, wird der Wert `value_if_true` zurückgegeben. Sonst wird der Wert `value_if_false` zurückgegeben bzw. `BLANK()`, falls `value_if_false` nicht angegeben ist.

> Syntax: `IF(<logical_test>, <value_if_true>[, <value_if_false>])`
> Beispiel:
> ```
> IF(
>     DISTINCTCOUNT(Follower[date_id]) = 1
>     && MIN(Follower[date_id]) > __MIN_DATE
>     && MAX(Follower[date_id]) <= [Max_Date],
>     SUM('Follower'[Facebook total followers]) - __PREV_MONTH,
>     BLANK()
> )
> ```
>
> `NOT`: Verneinung ändert den logischen Wert in das Gegenteil: `TRUE` wird `FALSE`, `FALSE` wird `TRUE`
> Beispiel: `NOT(ISBLANK(MAX(Follower[date_id])))`
>
> `ISFILTERED`: Wenn eine Tabelle oder Spalte gefiltert sind, wird `TRUE` zurückgegeben, sonst `FALSE`
> Beispiel: `ISFILTERED('Follower'[date_id])`
>
> `ISBLANK`: Wenn ein Wert leer ist, wird `TRUE` zurückgegeben, sonst `FALSE`
> Beispiel: `ISBLANK(MAX(Follower[date_id]))`

stackfuel

Filterfunktionen

Viele Berechnungen von Spalten und Measures geschehen im Zeilenkontext. Das bedeutet, jede Berechnung bezieht sich auf Werte in einer Zeile und so enthält das Ergebnis auch einen Wert pro Zeile. Zusätzlich zum Zeilenkontext gibt es auch einen Filterkontext. Dabei werden z.B. in einer Visualisierung die zu berücksichtigten Werte weiter eingegrenzt. Um solche dynamischen Berechnungen zu erstellen, verwendest du Filterfunktionen. Du kannst sie nutzen, um neue Filter in Tabellen zu setzen oder zu entfernen oder Berechnungen in einem bestimmten Filterkontext auszuführen. Filterfunktionen sind sehr mächtig und können zu sehr komplexen Funktionen führen. Sie umfassen neue Syntaxelemente:

CALCULATE: Die `CALCULATE`-Funktion wertet einen Ausdruck in einem bestimmten Filterkontext aus. Durch Visualisierungen in Power BI wird für ein Measure ein Filterkontext gesetzt. In einer Tabelle kann man beispielsweise Werte für bestimmte Monate anzeigen und dabei automatisch nach den Monaten filtern ohne diese Werte vorher gesondert zu berechnen. Die `CALCULATE`-Funktion kann den Filterkontext, der von der Visualisierung kommt, vollständig ändern.

Die `CALCULATE`-Funktion ist also eine besondere Funktion und sie unterscheidet sich auch in der Syntax von anderen DAX-Funktionen. Normalerweise berechnen DAX-Funktionen von links nach rechts. Bei `CALCULATE` ist das aber anders. Wird das resultierende Measure in einer Visualisierung genutzt, wird erst der Filterkontext der Visualisierung wie bei jedem anderen Measure auch gesetzt. Anschließend wird zusätzlich der Filter angewendet, der der `CALCULATE`-Funktion übergeben wird (`<filter1> [, <filter2> [, …]]]`). Und erst danach wird der Ausdruck berechnet (`<expression>`).

> Syntax: `CALCULATE(<expression>[, <filter1> [, <filter2> [, …]]])`
> Beispiel 1:
>
> ```
> salesAM = CALCULATE(Sheet1[salesTotal],ALL(Sheet1[accountManager]))
> ```

`CALCULATE` kann mit anderen Funktionen kombiniert werden. Es kann zusammen mit der `FILTER`-Funktion oder anderer Funktionen, die komplexe Berechnungen machen, genutzt werden. Im folgenden Beispiel verwendet `CALCULATE` die `FILTER`- und `ALL`-Funktionen für den Filterkontext. Für die Berechnung des Ausdrucks werden Aggregations- und mathematische Funktionen verwendet. Beispiel 2:

```
CALCULATE(
    DIVIDE(
        SUM(fact_posts_monthly[Post Count Target Reached]),
        COUNT(fact_posts_monthly[Post Count Target Reached])
    ),
    FILTER(
        ALL(fact_posts_monthly),
        RELATED(dim_account[platform]) = "Facebook"
        && fact_posts_monthly[date_id].[Jahr] = 2021
    )
)
```

stackfuel

Verwendung von Variablen in DAX-Formeln

Die Verwendung von Variablen hilft dabei, komplexere und gleichzeitig effizientere Berechnungen zu schreiben. Variablen können sowohl die Leistung und Zuverlässigkeit verbessern als auch die Lesbarkeit vereinfachen.

Um eine Variable zu definieren, wird das Schlüsselwort `VAR` genutzt. Der Variablennamen darf keine Leer- oder Sonderzeichen enthalten und nicht mit einer Ziffer beginnen. Variablen können Konstanten oder ganze berechnete DAX-Ausdrücke enthalten. Sobald eine Variable definiert wurde, wird der Ausdruck berechnet und der Variable zugewiesen. Über das Schlüsselwort `RETURN` können die Variablen dann genutzt werden.
Das folgende Beispiel zeigt eine einfache Verwendung von Variablen. Variablen werden hier verwendet, um den DAX-Ausdruck übersichtlich und gut lesbar zu halten.

Beispiel 1:

```
salesPerAM =

VAR _salesAM = SUM(churn[sales])

VAR _salesTotal = CALCULATE(SUM(churn[sales]),ALL(churn[accountManager]))

RETURN DIVIDE(_salesAM,_salesTotal)
```

Außerdem wertet DAX Variablen im Kontext ihrer Definition aus; nicht in dem Kontext, in dem sie verwendet werden. Diese Auswertung geschieht im Moment der Definition einer Variablen. Der Wert der Variablen wird später nicht mehr verändert, selbst wenn sie in einem anderen Kontext verwendet wird. So kann im nächsten Beispiel der Vergleich von Follower`[date_id]` mit `__THIS_MONTH` funktionieren.

Beispiel 2:

```
LinkedIn total followers =

VAR __THIS_MONTH = Follower[date_id]
VAR __MIN_DATE =
    CALCULATE(
        MIN(Follower[date_id]),
        FILTER(
            ALL(Follower),
            NOT(ISBLANK([LinkedIn monthly follower difference]))
        )
    )

RETURN
    IF(
        __THIS_MONTH >= __MIN_DATE
        && __THIS_MONTH <= [Max_Date],
        CALCULATE(
            SUM(Follower[LinkedIn monthly follower difference]) + 841,
            FILTER(
                Follower,

                Follower[date_id] <= __THIS_MONTH
            )
        )
)
```