sqlbi

www.sqlbi.com

sqlbi

a brand of Loader

Microsoft Partner

SSAS MAESTRO

Marco Russo
marco@sqlbi.com
twitter @marcorus

DAX Patterns

sqlbi

Who's Speaking?

sqlbi

- BI Expert and Consultant
- Founder of www.sqlbi.com
  - Problem Solving
  - Complex Project Assistance
  - Data Warehouse Assessments and Development
  - Courses, Trainings and Workshops
- Book Writer
- Microsoft Business Intelligence Partner

a brand of Loader

Microsoft Partner

SSAS MAESTRO

sqlbi

DAX Patterns

- Improve DAX knowledge
  - Virtual Relationships
  - Limits of Time Intelligence functions
  - Alternatives to snapshot fact table
- www.daxpatterns.com
  - Examples and other patterns online

sqlbi

Agenda

- Cumulative Total
- Custom Calendar (also week-based)
- Compare and allocate budget
- Distinct Count on SCD 2 attributes

sqlbi

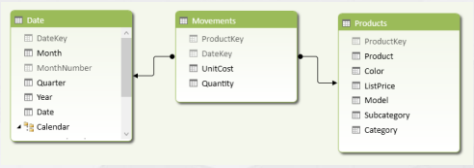
Running total, warehouse and accounts balance – no snapshot

http://www.daxpatterns.com/cumulative-total/

Cumulative Total

sqlbi

Cumulative Total: data model



Why avoid snapshot

- Avoid data duplicates by leveraging DAX
  - Reduce memory consumption
- Do not predefine data granularity
- Might not be better, evaluate case-by-case
  - Calculate transactions/snapshot rows ratio
  - Use snapshot only if it reduces rows

Cumulative Quantity

Sum all transactions filtered by date. Critical point is the number of rows in Date table, not the number of rows in Transactions table.

```
Cumulative Quantity :=
CALCULATE (
    SUM ( Transactions[Quantity] ),
    FILTER (
        ALL ( 'Date'[Date] ),
        'Date'[Date] <= MAX ( 'Date'[Date] )
    )
)
```

Inventory valuation

Calculate quantity per product by using cumulative quantity.

```
[UnitsInStock] := [Cumulative Quantity]

Value :=
SUMX (
    VALUES ( Products[ProductKey] ),
    [ProductPrice] * [UnitsInStock]
)
```

Valuation using last buy/sell price

Filter positive values for purchases, negative for sales.
AVERAGEX useful if more purchases in the same day (filtered by TOPN).

```
LastBuyPrice :=
IF (
    HASONEVALUE ( Products[ProductKey] ),
    AVERAGEX (
        CALCULATETABLE (
            TOPN ( 1, Movements, Movements[DateKey] ),
            Movements[Quantity] > 0,
            FILTER (
                ALL ( 'Date'[Date] ),
                'Date'[Date] <= MAX ( 'Date'[Date] )
            )
        ),
        Movements[UnitCost]
    )
)
```

Valuation using last buy/sell price

Filter positive values for purchases, negative for sales.
AVERAGEX useful if more sales in the same day (filtered by TOPN).

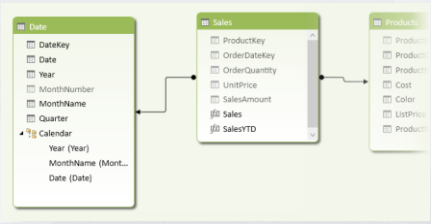
```
LastSellPrice :=
IF (
    HASONEVALUE ( Products[ProductKey] ),
    AVERAGEX (
        CALCULATETABLE (
            TOPN ( 1, Movements, Movements[DateKey] ),
            Movements[Quantity] < 0,
            FILTER (
                ALL ( 'Date'[Date] ),
                'Date'[Date] <= MAX ( 'Date'[Date] )
            )
        ),
        Movements[UnitCost]
    )
)
```

Implement Time Intelligence in DAX for non-standard calendars

http://www.daxpatterns.com/time-patterns/

Custom Calendars

Custom Calendars: data model



Rewrite Time Intelligence in DAX

- Support custom calendars
  - ISO 8601, 4-4-5 and similar ones
- Support DirectQuery in SSAS Tabular
  - Time Intelligence DAX functions do not support DirectQuery

Year-to-Date

Same technique used in Cumulative Total.

```
[YTD] :=
CALCULATE (
    [OriginalMeasure],
    FILTER (
        ALL ( 'Date' ),
        'Date'[Year] = MAX ( 'Date'[Year] )
            && 'Date'[Date] <= MAX ( 'Date'[Date] )
    )
)
```

Intelligence in Data

- DAX does not know the calendar
- Calculated column supports calculation

Day	YearDayNumber	QuarterDayNumber	MonthDayNumber	YearMonthNumber	MonthDays	YearQuarterNumber	QuarterDays
day	1	1	1	7	31	3	90
ay	2	2	2	7	31	3	90
day	3	3	3	7	31	3	90
day	4	4	4	7	31	3	90
nesday	5	5	5	7	31	3	90
rday	6	6	6	7	31	3	90
y	7	7	7	7	31	3	90
day	8	8	8	7	31	3	90
ay	9	9	9	7	31	3	90
day	10	10	10	7	31	3	90

Moving Annual Total (last 12 m.)

Filters SequentialDayNumber obtaining last 365 days.
Special case for leap year handled in SequentialDayNumber (process-time instead of query-time).

```
[MAT Sales] :=
CALCULATE (
    [Sales],
    FILTER (
        ALL ( 'Date' ),
        'Date'[SequentialDayNumber]
            > MAX ( 'Date'[SequentialDayNumber] ) - 365
            && 'Date'[SequentialDayNumber]
                <= MAX ( 'Date'[SequentialDayNumber] )
    )
)
```

Moving Annual Total (last 12 m.)

Handles leap year considering Feb 29 as it was Feb 28

```
SequentialDayNumber =
COUNTROWS (
    FILTER (
        ALL ( DATE ),
        'Date'[Date] <= EARLIER ( 'Date'[Date] )
            && NOT (
                MONTH ( 'Date'[Date] ) = 2
                    && DAY ( 'Date'[Date] ) = 29
            )
    )
)
```

Previous Month

Filters the column that uniquely identify the period with a sequential integer number, decreasing its value. Consider whole period, regardless of selection.

```
CALCULATE (
    [Sales],
    ALL ( 'Date' ),
    FILTER (
        ALL ( 'Date'[YearMonthNumber] ),
        'Date'[YearMonthNumber]
            = EARLIER ( 'Date'[YearMonthNumber] ) -
 1
    )
)
```

Compare arbitrary selection

As per previous period, adding a filter that repeats the selection of arbitrary periods selected (such as days).

```
CALCULATE (
    [Sales],
    ALL ( 'Date' ),
    CALCULATETABLE ( VALUES ( 'Date'[MonthDayNumber] ) ),
    FILTER (
        ALL ( 'Date'[YearMonthNumber] ),
        'Date'[YearMonthNumber]
            = EARLIER ( 'Date'[YearMonthNumber] ) - 1
    )
)
```

Previous Year

Repeat comparison month by month, applying an offset of 12 months instead of 1.

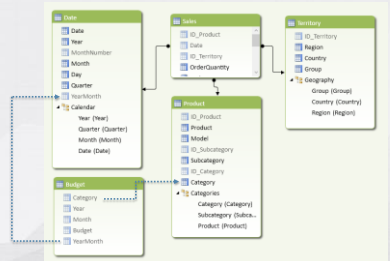
```
[PY Sales] :=
SUMX (
    VALUES ( 'Date'[YearMonthNumber] ),
    IF (
        < month fully selected >,
        < complete month 12 months before >,
        < selected days 12 months before >
    )
)
```

Allocate and compare budget defined at different granularity

http://www.daxpatterns.com/budget-patterns/

Budget Patterns

Budget: data model

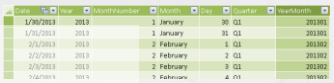


Relationships not available

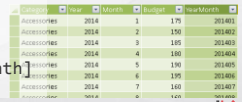
- Budget defined at different granularity
  - Month-Year instead of Day
  - Category instead of Product
- Define columns for "virtual" relationships

Virtual Relationships

Define single column on both tables if the relationship (even if virtual) is multi-column.



```
'Date'[YearMonth] =
'Date'[Year] * 100 +
'Date'[MonthNumber]
```



```
Budget[YearMonth] =
Budget[Year] * 100 + Budget[Month]
```

Budget Calculation

Two different formulas: with and without allocation, depending on the cardinality (filter context definition) of the current selection.

```
[Budget] :=
IF (
    [IsBudgetValid],
    [BudgetCalc],
    [AllocatedBudget]
)
```

Verify Cardinality

Compare number of rows (of fact table) forcing budget granularity. If the numbers are not identical, the budget must be allocated. This test is quick also on large tables.

```
[IsBudgetValid] :=
(
    COUNTROWS ( Sales )
        = CALCULATE (
            COUNTROWS ( Sales ),
            ALL ( Sales ),
            VALUES ( 'Date'[YearMonth] ),
            VALUES ( Product[Category] )
        )
)
```

Budget without Allocation

Applies filters on columns that defines virtual relationships in the budget table, selecting values active on the other side of the virtual relationship.

```
[BudgetCalc] :=
CALCULATE (
    SUM ( Budget[Budget] ),
    FILTER (
        ALL ( Budget[YearMonth] ),
        CONTAINS (
            VALUES ( 'Date'[YearMonth] ),
            'Date'[YearMonth], Budget[YearMonth]
        )
    ),
    FILTER (
        ALL ( Budget[Category] ),
        CONTAINS (
            VALUES ( Product[Category] ),
            Product[Category], Budget[Category]
        )
    )
)
```

```
// Filter argument per relazione virtuale
FILTER (
    ALL ( tab1[column] ),
    CONTAINS (
        VALUES ( tab2[column] ),
        tab2[column],
        tab1[column]
    )
)
```

Budget Allocation

Multiply budget by allocation factor. The CROSSJOIN increases the cardinality of the current selection to the budget cardinality level.

```
[AllocatedBudget] :=
SUMX (
    CROSSJOIN (
        VALUES ( 'Date'[YearMonth] ),
        VALUES ( Product[Category] )
    ),
    [AllocationFactor] * [BudgetCalc]
)
```

Allocation Factor

Allocation algorithm must be defined according to specific requirements. In this example, budget allocation uses quantities sold in previous year.

```
[AllocationFactor] :=
[QuantityPY]
    / CALCULATE (
        [QuantityPY],
        ALLEXCEPT (
            Product,
            Product[Category],
            Product[ID_Category]
        ),
        ALL ( 'Date' ),
        VALUES ( 'Date'[YearMonth] ),
        ALL ( Territory )
    )
```

```
[QuantityPY] :=
CALCULATE (
    [Quantity],
    SAMEPERIODLASTYEAR ( 'Date'[Date] )
)
```

sqlbi

Projection of Allocated Budget

- Budget is calculated also at different granularities than budget definition

Row Labels	2013 Quantity	2013 Budget	2013 Var%	2014 Quantity	2014 Budget	2014 Var%
Accessories	2,069.00			2,869.00	1,970.00	5.83%
Bottles and Cages	1,023.00			997.00	975.67	2.19%
Fenders	1,046.00			1,872.00	994.33	7.81%
Bikes	374.00			350.00	335.00	4.48%
Mountain Bikes	61.00			46.00	55.92	-17.74%
Mountain-100 Silver, 38	31.00			27.00	28.29	-4.54%
Mountain-100 Silver, 44	30.00			19.00	27.64	-31.25%
Road Bikes	313.00			304.00	279.08	8.93%
Road-150 Red, 44	140.00			138.00	126.36	9.21%
Road-150 Red, 62	170.00			166.00	152.72	8.70%
Grand Total	2,443.00			2,419.00	2,305.00	4.95%

Row Labels	2013 Quantity	2013 Budget	2013 Var%	2014 Quantity	2014 Budget	2014 Var%
Europe	540.00			584.00	514.80	13.44%
North America	1,476.00			1,409.00	1,294.92	5.85%
Canada	430.00			390.00	410.49	-4.99%
United States	1,046.00			1,019.00	984.43	3.51%
Central				1.00		
Northeast	2.00			1.00	1.83	-45.38%
Northwest	417.00			451.00	393.55	14.60%
Southeast	3.00			2.00	3.03	-34.08%
Southwest	624.00			564.00	586.01	-3.76%
Pacific	419.00			426.00	395.28	7.77%
Grand Total	2,443.00			2,419.00	2,305.00	4.95%

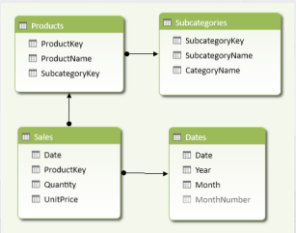
Related Distinct Count

Calculate distinct count on slowly changing dimension (SCD) Type II

http://www.daxpatterns.com/distinct-count/

sqlbi

Related Distinct Count: data model



sqlbi

No changes in data model

- Use same technique of many-to-many relationships
- Rows filtered in fact table becomes filter for dimension, for example:
  - Count unique categories of products sold in one month
  - Unique customers in SCD2 (distinct count on fact table scustomer key would produce an hight number)

sqlbi

Distinct Count for SCD Type 1

Use many-to-many relationships pattern to move the filter from fact table to dimension (also for snowflake).

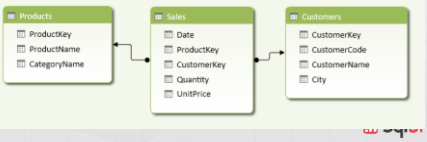
```
MonthsWithSales :=
CALCULATE ( DISTINCTCOUNT ( Dates[MonthNumber] ), Sales )

SoldCategories :=
CALCULATE (
    DISTINCTCOUNT ( Subcategories[CategoryName] ),
    Sales
)
```

sqlbi

Distinct Count for SCD Type 2

Use business key on dimension instead of surrogate key of SCD2 – same calculation as other attributes of the dimension.

```
UniqueCustomers :=
CALCULATE (
    DISTINCTCOUNT ( Customers[CustomerCode] ),
    Sales
)
```



Thank you!

Check our new articles on

www.sqlbi.com