www.sqlbi.com

sqlbi

Loader

Microsoft Partner
Gold Business Intelligence
Gold Data Platform

SSAS
MAESTRO
by Microsoft

MVP Microsoft Most Valuable Professional

Different techniques to handle relationships in DAX

Advanced Relationships in DAX

MASTERING DAX Workshop

sqlbi Loader

Who We Are

- BI Experts and Consultants
- Founders of www.sqlbi.com
  - Problem Solving
  - Complex Project Assistance
  - Data Warehouse Assessments and Development
  - Courses, Trainings and Workshops
- Book Writers
- Microsoft Gold Business Intelligence Partners
- SSAS Maestros – MVP – MCP

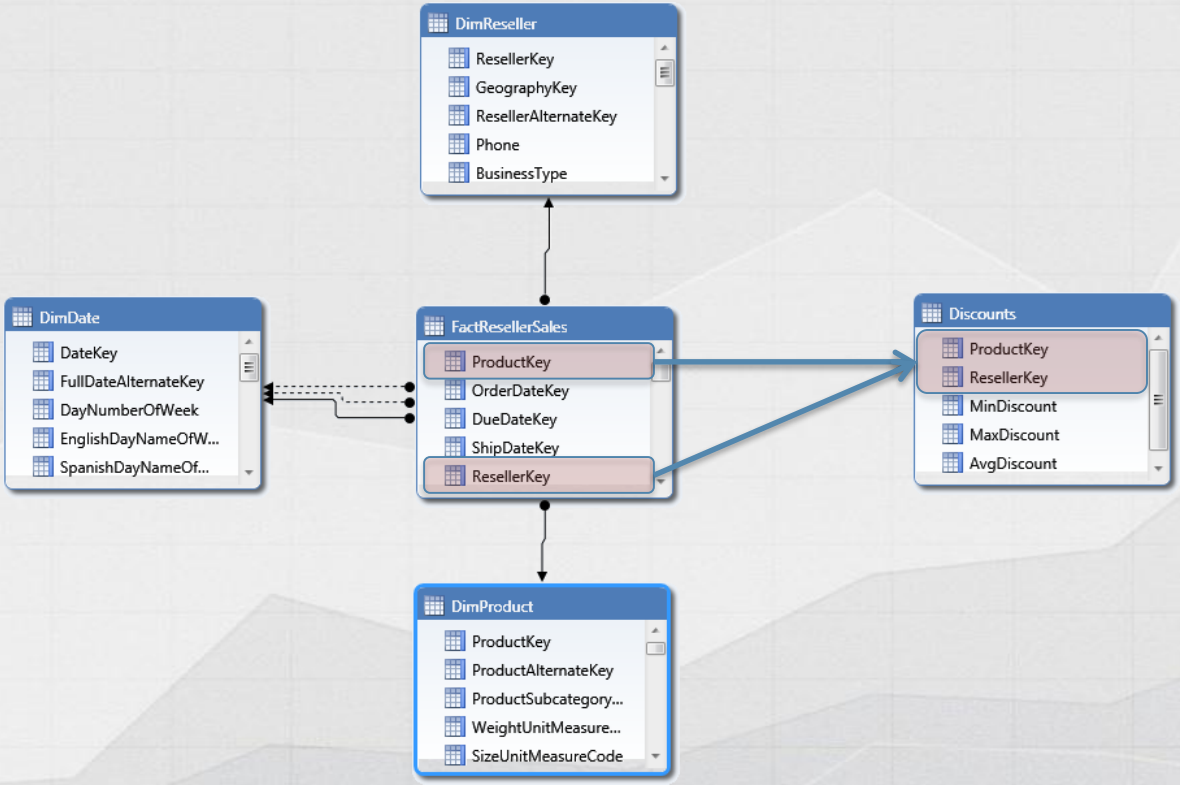
Introduction

- Tabular handles only one-to-many relationships
- The key to advanced relationships is DAX
  - Calculated Relationships
  - Virtual Relationships
  - Bidirectional Filtering
  - Many-to-many relationships
- Beware of performance, relationships come at a cost
- Let us see some examples

Multi-Column Relationships

- Tabular supports standard 1:N relationships
- Sometimes you need relationships than span over more than a single column

Multi-Column Relationships



sqlbi

1st Solution: Create Relationship

If the relationship is needed in the model, then you need to create a calculated column to set the relationship

```
ProductAndReseller =

Discounts[ProductKey] & "-" & Discounts[ResellerKey]
```

ProductKey	ResellerKey	MinDiscount	MaxDiscount	AvgDiscount	ProductAndReseller
470	2	0.02	0.02	0.02	470-2
214	3	0.02	0.02	0.02	214-3
224	3	0.02	0.02	0.02	224-3
231	3	0.02	0.02	0.02	231-3
233	3	0.02	0.02	0.02	233-3
236	3	0.02	0.02	0.02	236-3
327	3	0.02	0.02	0.02	327-3
333	3	0.02	0.02	0.02	333-3

2nd solution: Calculated Column

Using LOOKUPVALUE you can avoid setting the relationship and you denormalize the attribute in the fact table

```
Discount =

LOOKUPVALUE (
    Discounts[MaxDiscount],
    Discounts[ProductKey], FactResellerSales[ProductKey],
    Discounts[ResellerKey], FactResellerSales[ResellerKey]
)
```

Static Segmentation

Segmenting the prices

- Price changes over time
  - Discounts
  - Price variations
- Continuous dimension
- High fragmentation
- Segmentation
  - From 0 to 100 USD
  - From 101 to 500

Row Labels	Reseller Order Quantity	Reseller Sales Amount
2.29	674	$925,21
4.99	2.571	$7.476,60
7.95	2.411	$11.188,37
8.6442	3.289	$16.779,84
8.99	6.284	$32.826,92
9.5	1.197	$6.573,39
19.99	1.130	$13.514,69
20.24	774	$9.377,71
23.5481	1.877	$26.419,61
24.49	3.621	$52.507,99
25	1.086	$16.225,22
27.12	448	$7.280,43
33.6442	6.692	$131.508,29
34.2	95	$1.949,40
34.99	6.409	$127.204,64
37.99	6.055	$128.847,58
39.14	618	$14.469,82
40.49	1.317	$31.995,20
40.4909	531	$12.900,38
44.54	547	$14.530,43
46.09	56	$1.548,62
48.0673	6.587	$187.952,11

Segmentation in SQL

Building a data driven model with SQL is very easy because you can leverage the BETWEEN JOIN, which is not available in Tabular

BandName	FromPrice	ToPrice
VERY LOW	0	5
LOW	5	30
MEDIUM	30	100
HIGH	100	500
VERY HIGH	500	9999

```
SELECT
    P.BandName,
    SUM ( S.ExtendedAmount )
FROM dbo.FactResellerSales S
    JOIN PriceBands P
    ON S.UnitPrice BETWEEN P.FromPrice AND P.ToPrice
GROUP BY
    P.BandName
```

Static segmentation in DAX

Leverage CALCULATE and set the relationship inside the formula, producing a materialized version of the calculated relationship

```
Segment Price =

CALCULATE (
    VALUES ( 'Price Segments'[Segment Name] ),
    FILTER (
        'Price Segments',
         AND (
           'Price Segments'[FromPrice] < Sales[Net Price],
           'Price Segments'[ToPrice] >= Sales[Net Price]
        )
    )
)
```

Dynamic Segmentation

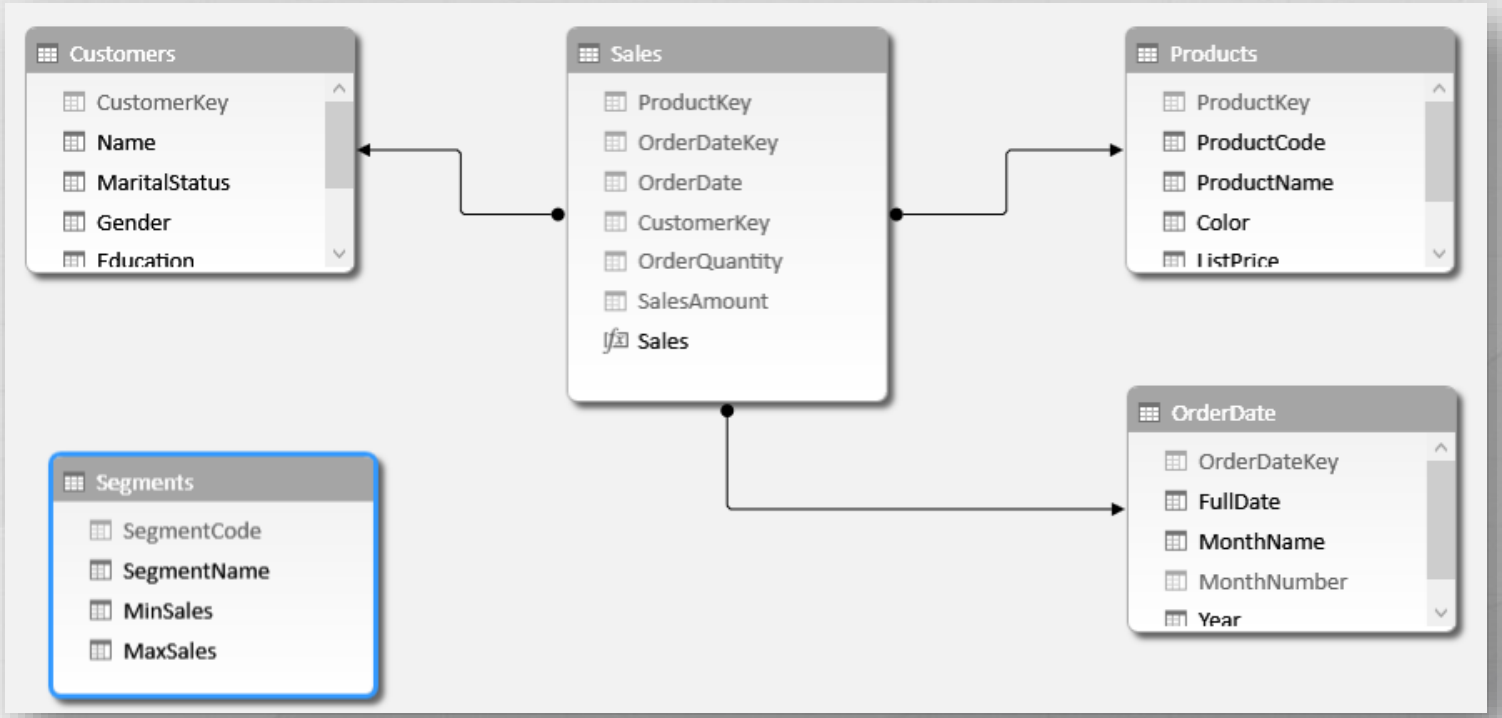
Dynamic Segmentation

SegmentCode	SegmentName	MinSales	MaxSales
1	Very Low	0	100
2	Low	100	1000
3	Medium	1000	5000
4	High	5000	10000
6	Very High	10000	9999999

CustInSegment	Column Labels				
Row Labels	2005	2006	2007	2008	Grand Total
Very Low			3810	5040	7676
Low	113	401	1805	2539	3112
Medium	900	2276	3639	3792	5964
High			54	6	1703
Very High			1		29
Grand Total	**1013**	**2677**	**9309**	**11377**	**18484**

sqlbi

The Data Model



sqlbi

Dynamic Segmentation

This time, the relationship is “virtual”.
It does not exists outside of the formula, no materialization

```
NumOfCustomersInSegment =

CALCULATE (
    COUNTROWS ( Customer ),
    FILTER (
        Customer,
        AND (
            [Sales Amount] > MIN ( 'Sales Segments'[MinSales] ),
            [Sales Amount] <= MAX ( 'Sales Segments'[MaxSales] )
        )
    )
)
```

Relationships at different granularities are a challenge

Working at different granularity

Different Granularity

- Sales recorded at levels of
  - Day
  - Product
- Budget stored in Excel
  - Year
  - Brand
  - Country

CountryRegion	Brand	Budget
China	A. Datum	2,000,000.00
China	Adventure Works	14,600,000.00
China	Contoso	37,800,000.00
China	Fabrikam	36,320,000.00
China	Litware	26,000,000.00
China	Northwind Traders	4,800,000.00
China	Proseware	8,000,000.00
China	Southridge Video	9,000,000.00
China	Tailspin Toys	3,200,000.00
China	The Phone Company	12,000,000.00
China	Wide World Importers	18,000,000.00
Germany	A. Datum	5,400,000.00
Germany	Adventure Works	6,000,000.00
Germany	Contoso	20,000,000.00
Germany	Fabrikam	18,000,000.00
Germany	Litware	5,000,000.00

Options for granularity

- Change the data model
  - Create tables for the required granularity
  - Might create an unclear data model
- Force granularity with one element
  - For example, with dates, 1° of the month
  - Not an option for all the data models
- Rely on more complex DAX code
  - A bit harder to implement
  - Much more flexible
  - Might be slow on large data models

Granularity in DAX

By using set functions you can move the filter from one table to another one (from Product and Customer to Budget in the example)

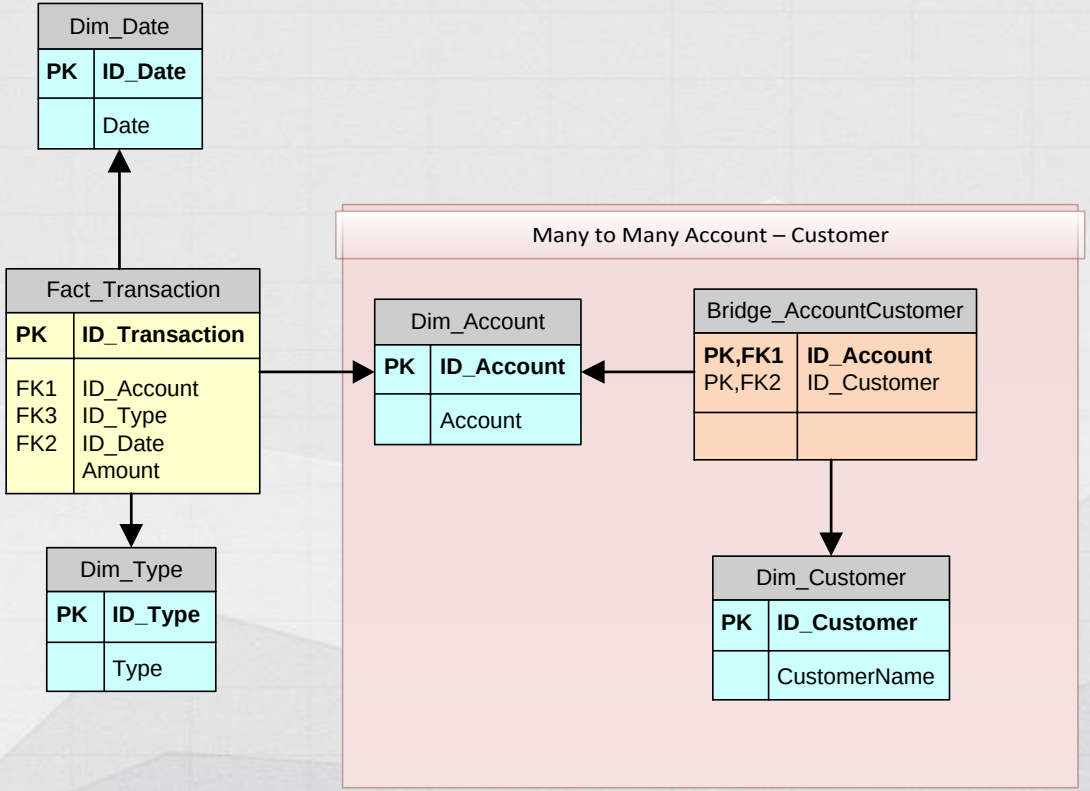
```
BudgetAmt =

CALCULATE (
    SUM ( Budget[Budget] ),
    INTERSECT (
        VALUES ( Budget[Brand] ),
        VALUES ( 'Product'[Brand] )
    ),
    INTERSECT(
        VALUES ( Budget[CountryRegion] ),
        VALUES ( Customer[CountryRegion] )
    )
)
```

Many ways to handle many-to-many relationships

Many-to-many

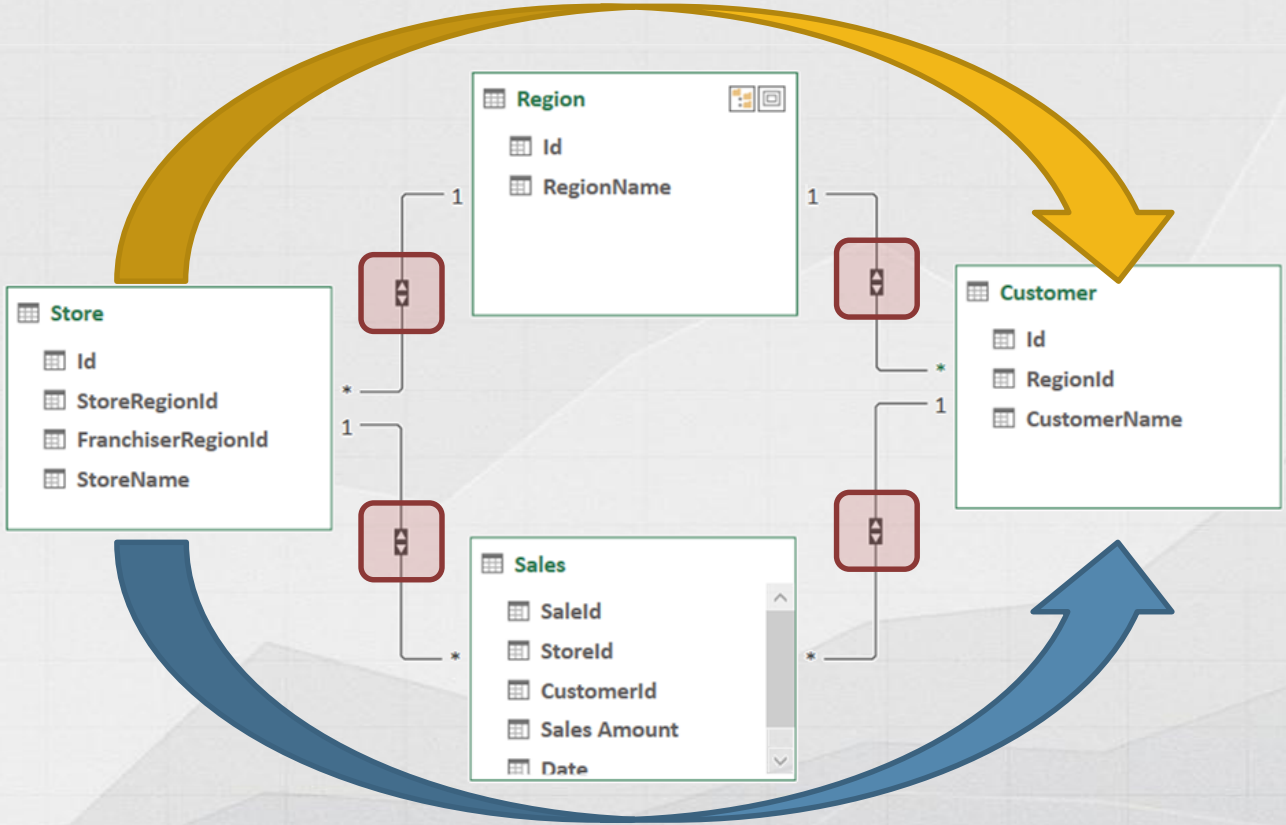
Many-to-many Relationships



Bidirectional Filtering

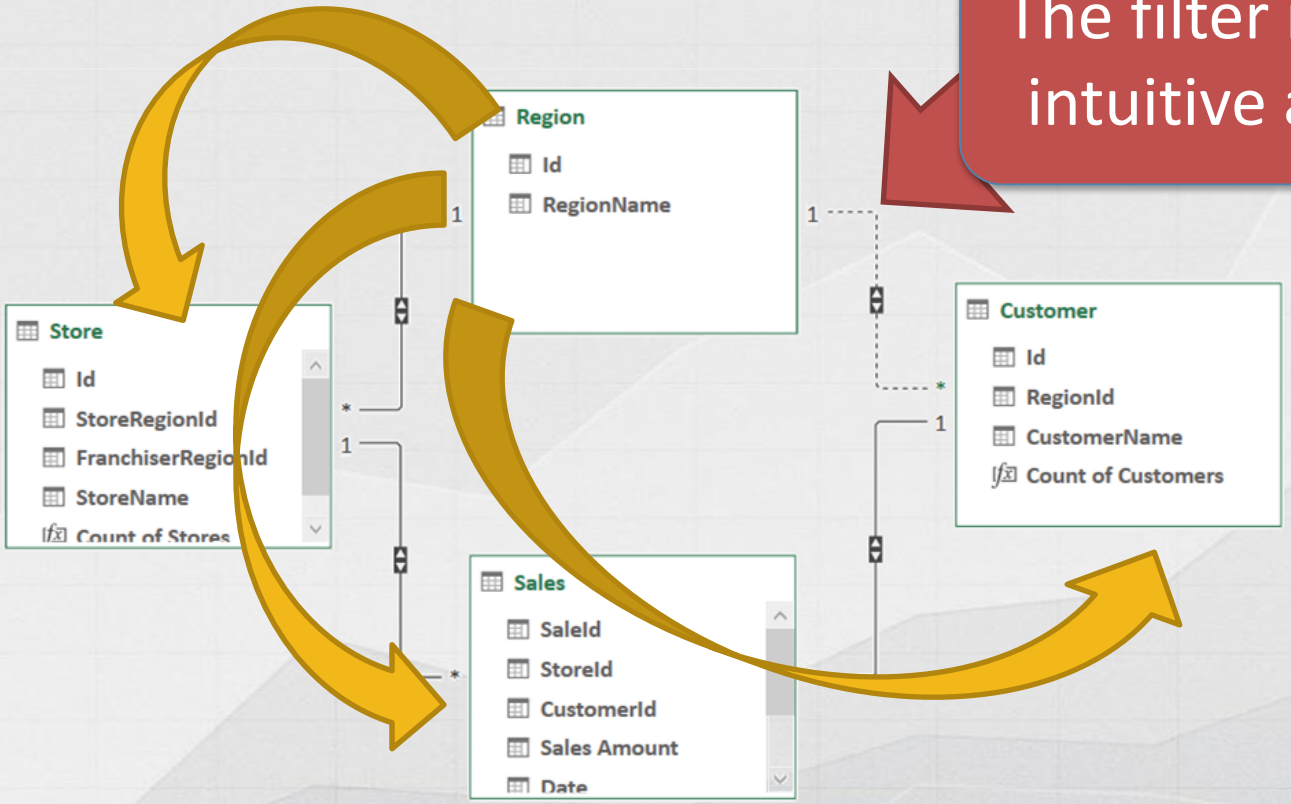
- Many-to-many require a bridge table
  - Enable bidirectional filtering
  - On both relationships
- Issues
  - Bridge table must be complete
  - Standard, plain, many-to-many
  - Bidirectional filtering might result in ambiguous models

What is ambiguity?



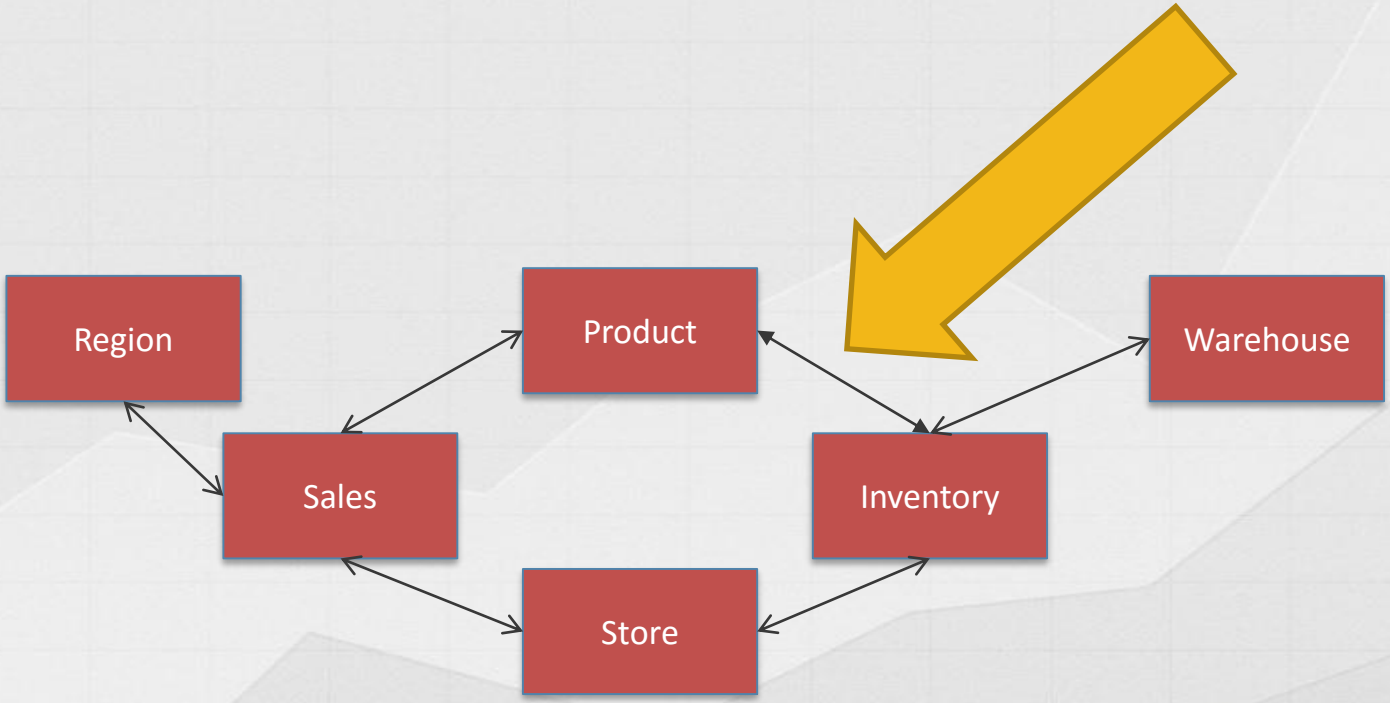
sqlbi

Solving ambiguity

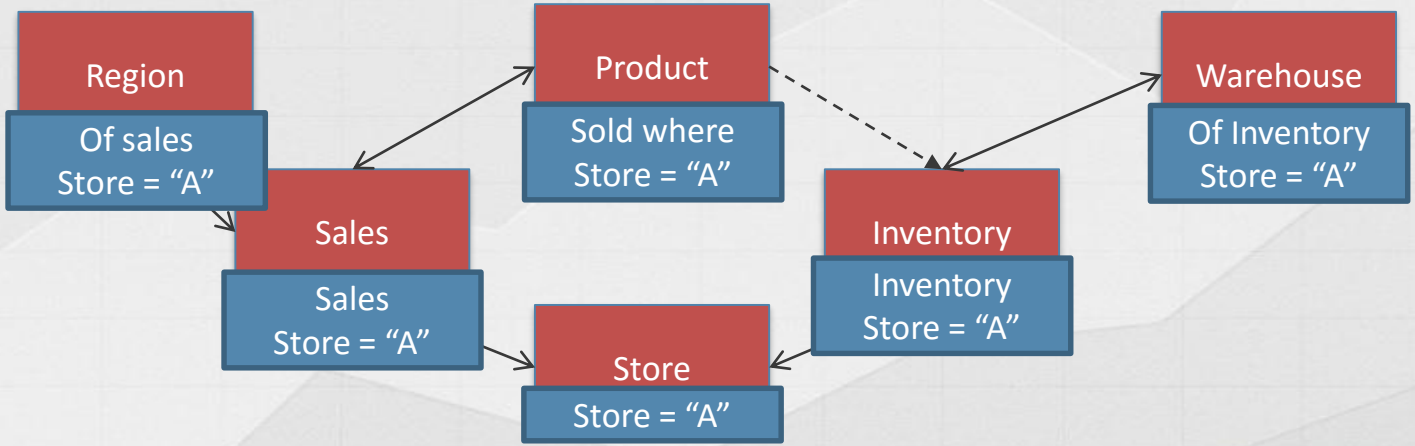


sqlbi

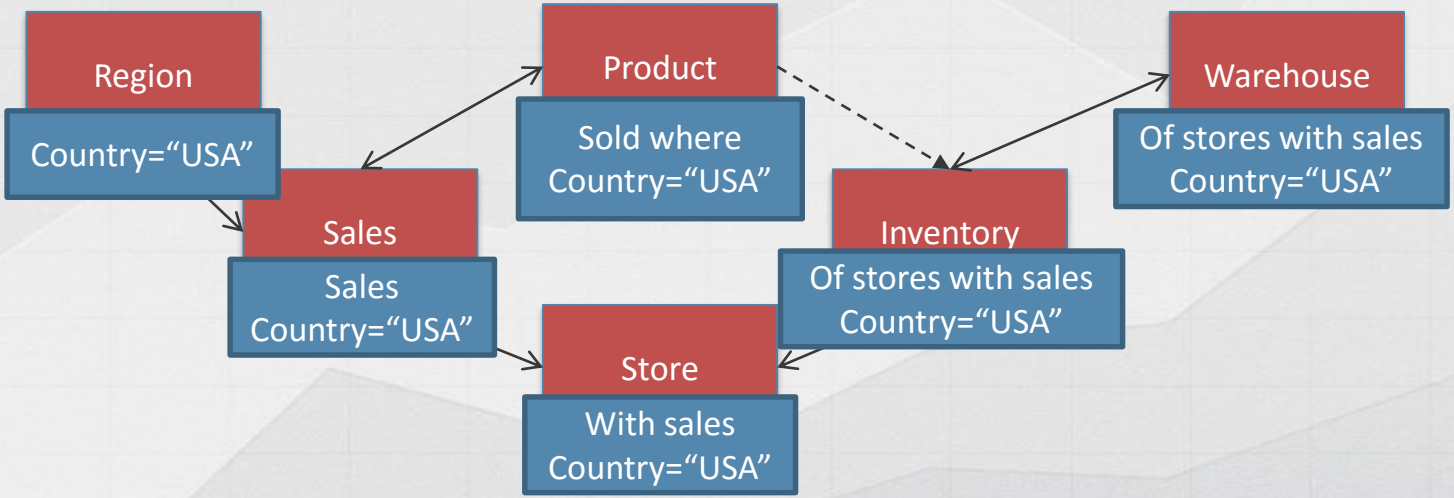
Ambiguous model



No ambiguity, still too complex



No ambiguity, still too complex



Relationship rules

- Single star schema
  - Enable bidirectional filtering
  - Beware of performance
- Any other model, including multiple star schemas
  - Keep all relationship unidirectional
  - Enable bidirectional when needed
  - **Only** when needed

Direct Table Filtering

Leveraging table expansion and filter context, you can obtain the same result, without enabling two-way relationships

```
AmountM2M :=

CALCULATE (
    SUM ( Transaction[Amount] ),
    AccountCustomer
)
```

MANY-TO-MANY with SUMMARIZE

Instead of using a table name, you can use any table expression that can filter the fact table.
In this example, we obtain the same result using SUMMARIZE

```
AmountM2M :=

CALCULATE (
    SUM ( Transaction[Amount] ),
    SUMMARIZE (
        AccountCustomer,
        Account[ID_Account]
    )
)
```

New Customers

- Many useful calculations
  - Customers
  - Buying customers
  - New Customers
  - Returning customers
- We will see two versions of the formula
  - The naïve one (created by me)
  - The fast one (learned while I was pretending to teach)

Computing new customers

The fast version is much better than the naïve one, because the first SUMMARIZE results in a pure SE query cached by the engine only once

```
NewCustomersFast =

COUNTROWS (
    FILTER (
        SUMMARIZE (
            CALCULATETABLE ( Sales, ALL ( 'Date' ) ),
            Sales[CustomerKey],
            "DateOfFirstBuy", MIN ( Sales[OrderDateKey] )
        ),
        CONTAINS (
            VALUES ( 'Date'[DateKey] ),
            'Date'[DateKey],
            [DateOfFirstBuy]
        )
    )
)
```

Conclusions

- Tabular handles only one-to-many relationships
- The key to advanced relationships is DAX
  - Calculated Relationships
  - Virtual Relationships
  - Bidirectional Filtering
  - Many-to-many relationships
- Beware of performance, relationships come at a cost
- Have fun with DAX!

Thank you!

Check our articles, whitepapers and courses on

www.sqlbi.com