

# Data Modeling & DAX Fundamentals

Building Star Schema & Australian FY  
Analytics

(Some things are intentionally wrong to  
increase your problem solving skills)



## Session Goals & Agenda

# What You'll Build & Learn

### What You'll Build

- Star schema: 1 fact table + 5 dimensions
- Proper relationships (1:\*, single direction)
- Australian FY date table (July 1 - June 30)
- 20+ DAX measures (revenue, margins, time intelligence)
- Optimised performance (<1 second queries)

### Learning Outcomes

- Design star schema data models
- Create relationships with correct cardinality
- Build custom date tables with Australian FY logic
- Write DAX measures for business metrics
- Implement time intelligence (YTD, YoY, QoQ)
- Calculate budget variance analysis
- Optimise model performance

## Agenda

01

### Star schema principles

5 min

02

### Building the model & relationships

10 min

03

### Australian FY date table

5 min

04

### Core DAX measures

8 min

05

### Time intelligence & budget variance

8 min

06

### Advanced DAX & optimisation

4 min

# Why Star Schema Wins for Analytics

## The Database Design Battle

Aspect	Normalised Database	Star Schema
Purpose	OLTP (transactions)	OLAP (analytics)
Joins	Many (3-7 tables)	Few (1 per dimension)
Performance	Slow for reporting	Fast (10-100x)
Storage	Minimal redundancy	Some denormalisation
User Experience	Complex queries required	Drag-and-drop friendly
Use Case	Banking transactions	Executive dashboards

## Star Schema Benefits

- ✓ **Fast queries:** 1 join per dimension vs. multiple joins through normalised tables
- ✓ **Simple DAX:** Relationships handle the complexity
- ✓ **Intuitive filtering:** Business users understand dimension hierarchies
- ✓ **Scalable:** Handles millions of rows efficiently
- ✓ **Optimised for BI:** Designed for aggregation, not row-level updates

## Example: "Revenue by Product Category"

### Normalised (3 joins):

Sales → Products → ProductCategories → Subcategories

### Star Schema (1 join):

FactSales → DimProduct (Category already in dimension)

# The 7 Golden Rules of Star Schema Design

DO:

1.  **Use surrogate keys** (ProductID, StoreID) for relationships
2.  **Store descriptive attributes** in dimensions (product names, store locations)
3.  **Store numeric measures** in fact tables (revenue, quantity, cost)
4.  **Use 1:\*** (one-to-many) relationships from dimension to fact
5.  **Use single-direction** cross-filter (dimension → fact)
6.  **Create a date dimension** with calendar hierarchies
7.  **Denormalise dimensions** (Category in DimProduct, not separate table)

DON'T:

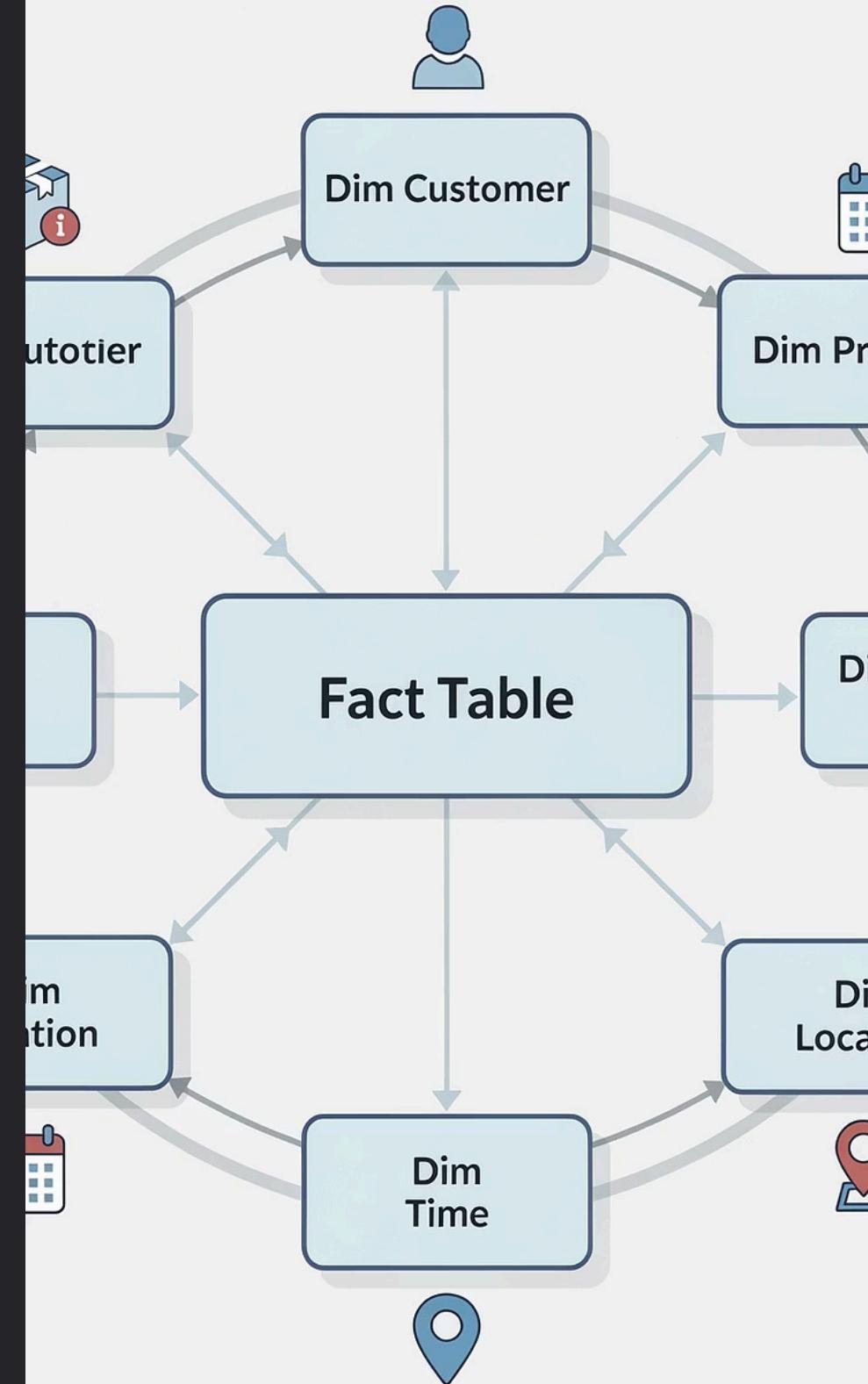
1.  **Many-to-many relationships** (causes ambiguity)
2.  **Bi-directional cross-filter** (performance killer)
3.  **Descriptive text in fact tables** (use dimension keys instead)
4.  **Snowflake schemas** (normalised dimensions) unless necessary
5.  **Natural keys that change** (use stable surrogate keys)
6.  **Mixed grain levels** in same fact table

## Fact Table Grain Definition

**FreshMarket FactSales Grain:**

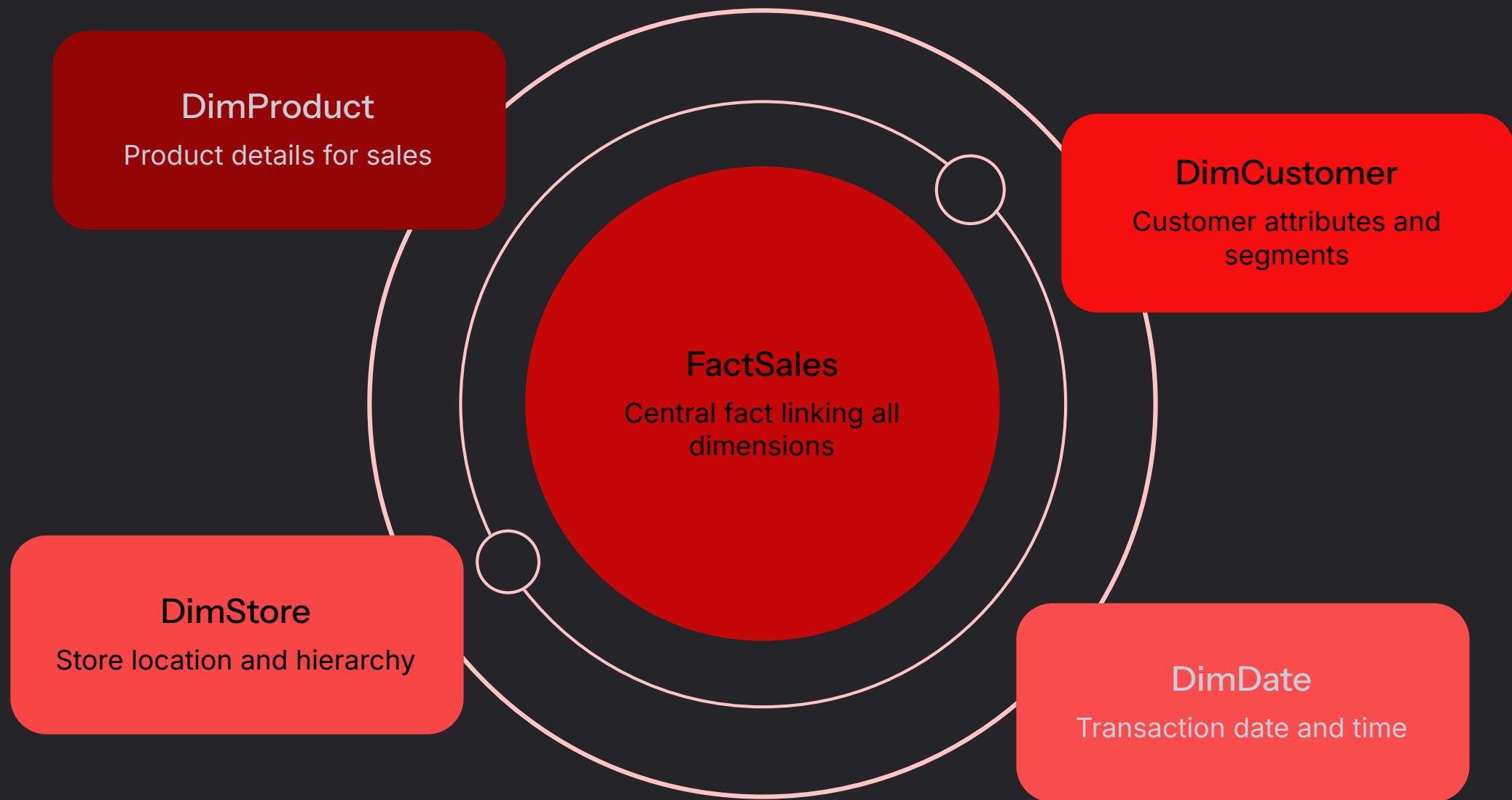
- One row per transaction line item
- Unique: TransactionID + ProductID + StoreID + Date + CustomerID + ChannelID
- Measures: Quantity, Revenue, Cost, GST, Discount

**Rule:** All facts must be at the same grain. Don't mix daily sales + monthly budgets in same table.



# Creating the Star Schema Relationships

## The 5 Relationships



## Relationship Configuration

From Table	From Column	To Table	To Column	Cardinality	Cross Filter	Active
DimStore	StoreID	FactSales	StoreID	1:*	Single	<input checked="" type="checkbox"/>
DimProduct	ProductID	FactSales	ProductID	1:*	Single	<input checked="" type="checkbox"/>
DimCustomer	CustomerID	FactSales	CustomerID	1:*	Single	<input checked="" type="checkbox"/>
DimChannel	ChannelID	FactSales	ChannelID	1:*	Single	<input checked="" type="checkbox"/>
DimDate	Date	FactSales	TransactionDate	1:*	Single	<input checked="" type="checkbox"/>

**Cardinality: 1:\*** (one-to-many)

- 1 store → many transactions
- 1 product → many sales
- Ensures proper aggregation

**Cross Filter Direction: Single**

- Filters flow: Dimension → Fact
- Prevents performance issues
- Avoids ambiguous filter paths

**Active:**

- Relationship is used by default
- Don't create inactive relationships unless using USERRELATIONSHIP

# Custom Date Table with Australian Financial Year Logic

## Why Custom Date Table?

### ✗ Power BI Auto-Date Tables:

- Use calendar year (Jan-Dec)
- Can't customise fiscal year
- No EOFY (End of Financial Year) flag
- Bloat model size

### ✓ Custom Date Table:

- Australian FY (July 1 - June 30)
- FY column: IF(MONTH ≥ 7, YEAR + 1, YEAR)
- EOFY flag: June 30 = 1
- Custom columns: FYQuarter, FYNMonth
- Enables time intelligence

## Key Date Columns

Column	Example	Purpose
Date	2024-06-30	Primary key (marked as date table)
FY	FY2024	Financial year label
IsEOFY	1	End of financial year flag (June 30)
FYQuarter	FY2024 Q4	Q1=Jul-Sep, Q2=Oct-Dec, Q3=Jan-Mar, Q4=Apr-Jun
FYNMonth	12	Month 1 = July, Month 12 = June

## Date Table DAX Formula

```
DimDate =  
ADDCOLUMNS(  
    CALENDAR(DATE(2022,1,1), DATE(2025,12,31)),  
    "FY", "FY" & IF(MONTH([Date]) >= 7, YEAR([Date]) + 1, YEAR([Date])),  
    "IsEOFY", IF(MONTH([Date]) = 6 && DAY([Date]) = 30, 1, 0),  
    "FYQuarter", ...,  
    "FYNMonth", ...  
)
```

## Example Validation

Date	Year	FY	IsEOFY	FYQuarter
2024-06-30	2024	FY2024	1	FY2024 Q4
2024-07-01	2024	FY2025	0	FY2025 Q1

June 30, 2024 → End of FY2024

July 1, 2024 → Start of FY2025

# Building the Foundation: Revenue, Cost, Profitability

## The Measures Table Pattern

Create a dedicated \_Measures table (not tied to fact/dimension):

- Groups all measures in one place
- Easier to find and maintain
- Enterprise best practice

### Core Revenue Measures

Total Revenue =  
`SUM(FactSales[Revenue])`

Total Cost =  
`SUM(FactSales[Cost])`

Total Quantity =  
`SUM(FactSales[Quantity])`

Total Transactions =  
`DISTINCTCOUNT(FactSales[TransactionID])`

Average Order Value =  
`DIVIDE([Total Revenue], [Total Transactions], 0)`

### Profitability Measures

Gross Profit =  
`[Total Revenue] - [Total Cost]`

Gross Margin % =  
`DIVIDE([Gross Profit], [Total Revenue], 0)`

Avg Margin Per Transaction =  
`DIVIDE([Gross Profit], [Total Transactions], 0)`

#### ❑ Why DIVIDE() Instead of / ?

✗ Margin % = **[Gross Profit] / [Total Revenue]**

**Problem:** Error if denominator = 0

✓ Margin % = **DIVIDE([Gross Profit], [Total Revenue], 0)**

**Solution:** Returns 0 (or BLANK) if denominator = 0

## Measure Validation

Measure	Expected Range	Validation Method
Total Revenue	\$40M-50M	Compare to source SUM(Revenue)
Gross Margin %	25-35%	Manual: Profit / Revenue
Total Transactions	~50,000	Count distinct TransactionIDs

# YTD, YoY, QoQ Using Australian Fiscal Year

## Time Intelligence Requirements

1.  Date table marked as date table
  2.  Relationship between date table and fact table
  3.  Continuous date range (no gaps)
  4.  Fiscal year end parameter ("6/30")

Year-to-Date (YTD) - Australian FY

```
YTD Revenue =  
TOTALYTD(  
    [Total Revenue],  
    DimDate[Date],  
    "6/30" -- Fiscal year ends June 30  
)
```

**Resets on July 1 (start of new FY)**

## Prior Year & Year-over-Year

```
Prior Year Revenue =  
CALCULATE(  
    [Total Revenue],  
    SAMEPERIODLASTYEAR(DimDate[Date])  
)
```

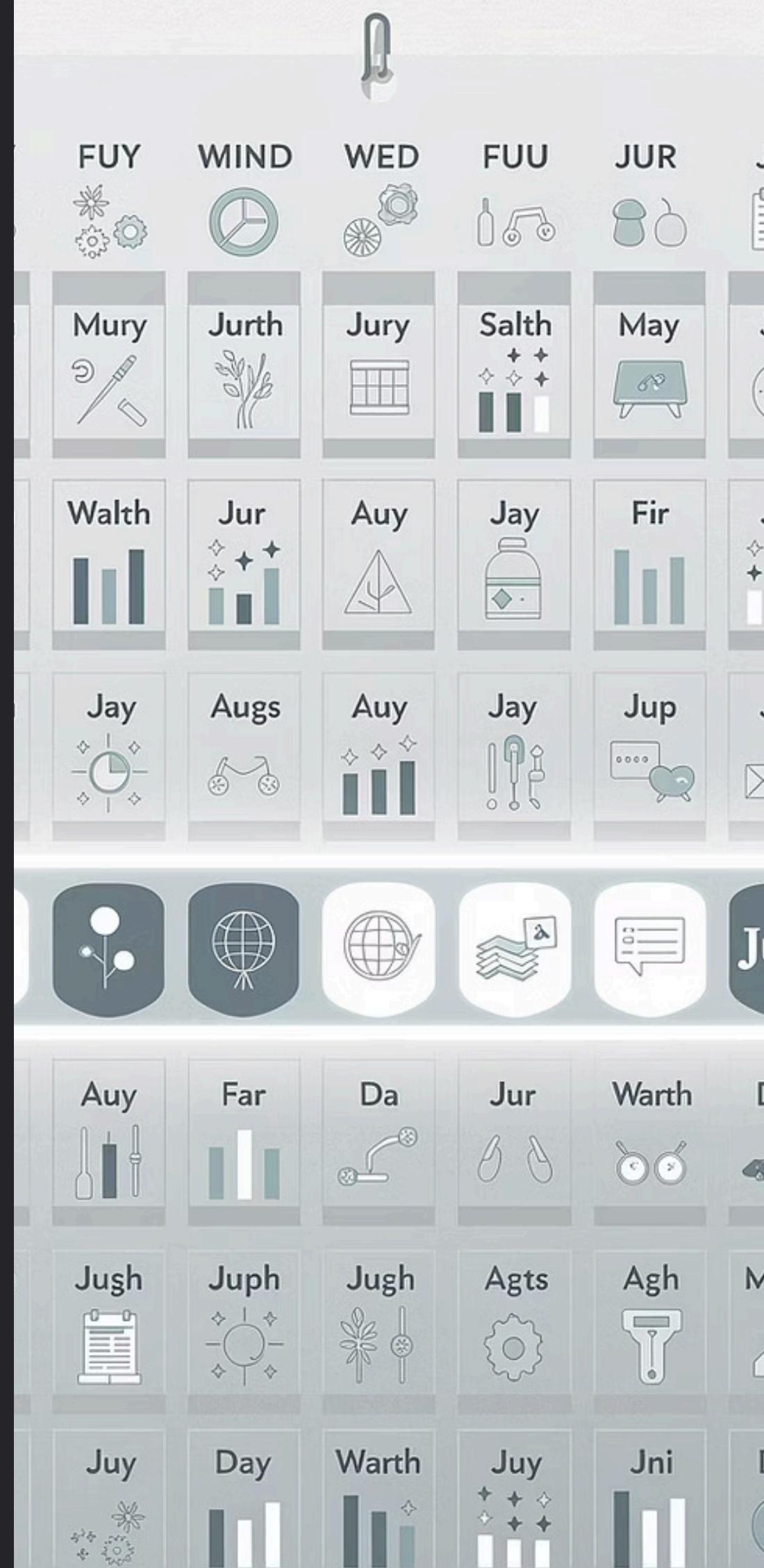
```
YoY Revenue % =  
DIVIDE(  
    [Total Revenue] - [Prior Year Revenue],  
    [Prior Year Revenue],  
    BLANK()  
)
```

## Quarter-over-Quarter

```
QoQ Revenue % =  
VAR CurrentQuarter = [Total Revenue]  
VAR PriorQuarter =  
    CALCULATE([Total Revenue],  
              DATEADD(DimDate[Date], -1, QUARTER))  
RETURN  
    DIVIDE(CurrentQuarter - PriorQuarter,  
           PriorQuarter, BLANK())
```

## Validation Example

MonthYear	Revenue	YTD Revenue	Prior Year	YoY %
Jul-2023	\$3.5M	\$3.5M	\$3.2M	9.4%
Aug-2023	\$3.7M	\$7.2M	\$3.5M	5.7%
...	...	...	...	...
Jun-2024	\$4.2M	\$45.0M	\$40.8M	10.3%
Jul-2024	\$3.8M	\$3.8M (resets)	\$3.5M	8.6%



# Tracking Performance vs Plan

## Budget Data Integration

DimBudget table (from Excel) with relationships:

- ProductID → DimProduct
- Month → DimDate

**Grain:** Monthly budget by product

## Budget Measures

Budget Revenue = SUM(DimBudget[BudgetRevenue])
Variance \$ = [Total Revenue] - [Budget Revenue]
Variance % = DIVIDE([Variance \$], [Budget Revenue], BLANK())
Budget Achievement % = DIVIDE([Total Revenue], [Budget Revenue], BLANK())

## Interpreting Variance

Achievement %	Variance \$	Meaning	Action
>100%	Positive	Over budget (good)	Investigate drivers
100%	\$0	Exactly on budget	Maintain performance
<100%	Negative	Under budget (bad)	Corrective action

## Example Variance Report

Product Category	Actual	Budget	Variance \$	Variance %	Achievement %
Fresh Produce	\$12.5M	\$12.0M	+\$500K	+4.2%	104.2%
Dairy & Eggs	\$8.2M	\$8.5M	-\$300K	-3.5%	96.5%
Meat & Seafood	\$10.0M	\$9.5M	+\$500K	+5.3%	105.3%

### ☐ Key Insights

- Fresh Produce and Meat exceeded budget (investigate success factors)
- Dairy underperformed (review pricing, promotions, competition)

# RANKX, TOPN, and Context Awareness

## Top 5 Products by Revenue

```
Top 5 Products Revenue =  
CALCULATE(  
    [Total Revenue],  
    TOPN(  
        5,  
        ALL(DimProduct[ProductName]),  
        [Total Revenue],  
        DESC  
    )  
)
```

## Product Ranking

```
Product Rank =  
IF(  
    ISINSCOPE(DimProduct[ProductName]),  
    RANKX(  
        ALL(DimProduct[ProductName]),  
        [Total Revenue],  
        ,  
        DESC,  
        DENSE  
    ),  
    BLANK()  
)
```

### How it works:

1. ALL(DimProduct[ProductName]) removes product filters
2. TOPN(5, ..., [Total Revenue], DESC) returns top 5 by revenue
3. CALCULATE reapplys filter and calculates revenue

**ISINSCOPE:** Checks if ProductName is in the visual (prevents rank at wrong grain)

**DENSE:** Ties get same rank, next rank continues (1, 2, 2, 3 not 1, 2, 2, 4)

## Rolling 30-Day Revenue

```
Rolling 30d Revenue =  
CALCULATE(  
    [Total Revenue],  
    DATESINPERIOD(  
        DimDate[Date],  
        MAX(DimDate[Date]),  
        -30,  
        DAY  
    )  
)
```

## Example Output

ProductName	Revenue	Rank	% of Total
Product A	\$500K	1	12.5%
Product B	\$450K	2	11.3%
Product C	\$450K	2 (tie)	11.3%
Product D	\$400K	3	10.0%

# When to Use Each (and Why It Matters)

## Calculated Columns

**Definition:** Computed row-by-row during refresh, stored in model

### Use When:

- Need to filter/slice by the value
- Row-level calculation (not aggregation)
- Static value needed for relationships

### Example:

```
Age Group =  
VAR Age = DATEDIFF(DimCustomer[DateOfBirth],  
    TODAY(), YEAR)  
RETURN  
SWITCH(  
    TRUE(),  
    Age < 25, "18-24",  
    Age < 35, "25-34",  
    Age < 50, "35-49",  
    Age < 65, "50-64",  
    "65+"  
)
```

Now users can **filter/slice** by Age Group

## Memory Impact

Scenario	Calculated Column	Measure
1M row fact table	Stores 1M values (~8 MB)	Stores 0 values (0 MB)
Performance	Slower refresh	Faster refresh
Filtering	Can use in slicers <input checked="" type="checkbox"/>	Cannot use in slicers <input checked="" type="checkbox"/>
Context	Static (computed once)	Dynamic (changes with filters)

**Rule:** Default to measures. Only use calculated columns when you need to filter/slice.

# Performance Tuning for Sub-Second Queries

1

## Disable Auto-Date Tables

- File → Options → Data Load → Uncheck "Auto date/time"
- Saves 5-10 MB per date column
- Eliminates duplicate date logic

2

## Optimise Data Types

- Text: Only for descriptive attributes (ProductName, StoreName)
- Whole Number: Use instead of Decimal when possible (Quantity)
- Currency: For monetary values (Revenue, Cost)
- Boolean: For flags (0/1 instead of True/False text)

3

## Remove Unused Columns

- Table tools → Remove columns
- Only load columns used in model

4

## Use Measures (Not Calculated Columns)

- Calculated columns: Stored (uses memory)
- Measures: Computed on-the-fly (no memory)

5

## Optimise Relationships

- All 1:\* cardinality (never \*:\*)
- All single cross-filter (never Both)
- All active (unless USERELATIONSHIP)

6

## Disable Load for Staging Queries

- Only clean queries load to model
- Staging queries: reference only

## Performance Targets

Metric	Target	FreshMarket Actual
Model Size	<50 MB	42 MB <span style="color: green;">✓</span>
Query Time	<1 second	0.7 seconds <span style="color: green;">✓</span>
Refresh Time	<5 minutes	2.8 minutes <span style="color: green;">✓</span>
Visual Load	<3 seconds	1.2 seconds <span style="color: green;">✓</span>