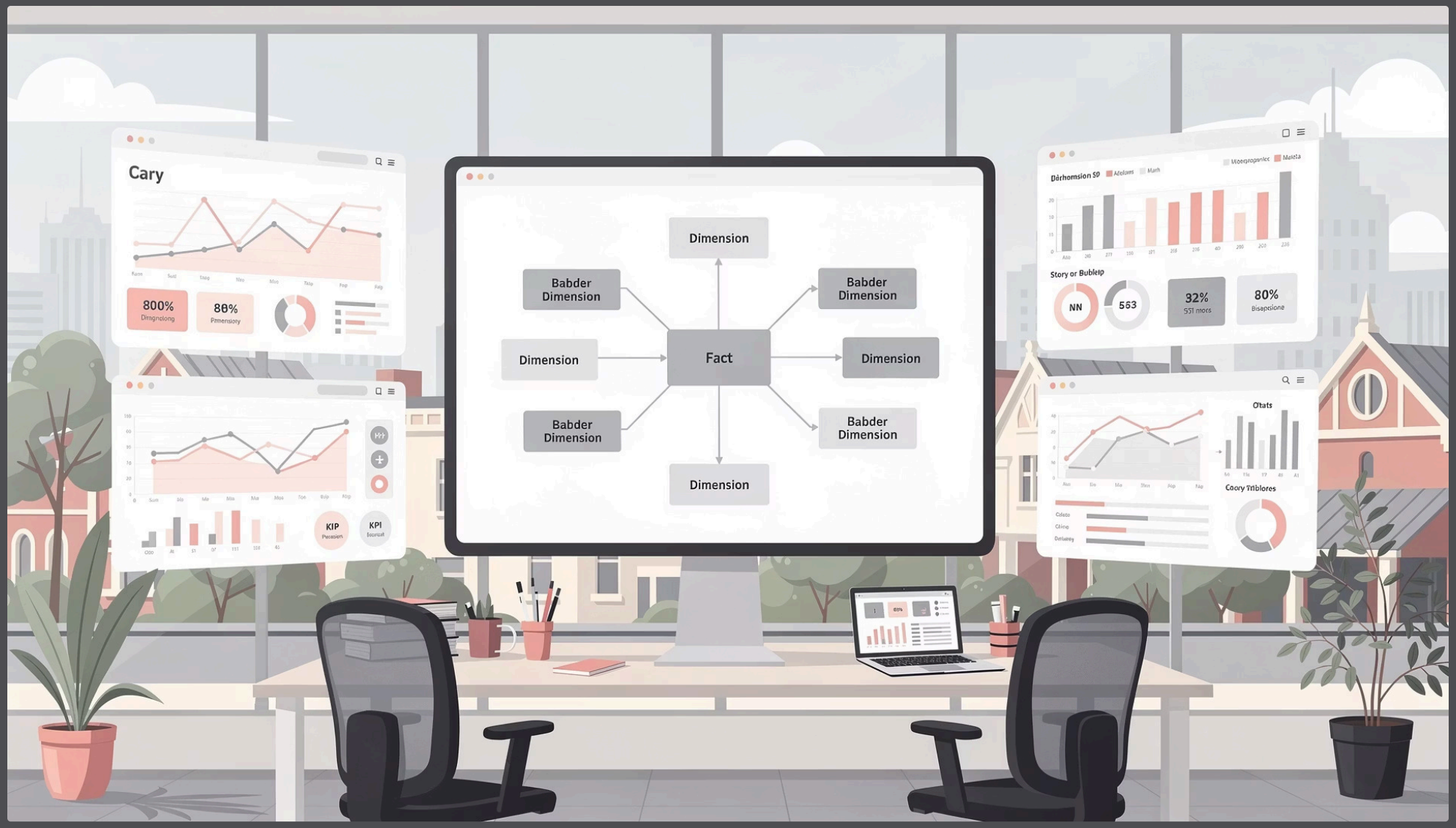


Complete Playbook - FreshMarket Performance Analysis

Duration: 35-40 minutes

Industry: Australian Grocery Retail

Focus: Star schema design, relationships, DAX measures, Australian FY time intelligence

Table of Contents

1. Project 2 Overview & Learning Outcomes

Duration: 35-40 minutes

What You'll Build

A **production-ready star schema data model** with:

- ✅ 1 fact table (FactSales) + 5 dimension tables
- ✅ Proper relationships (1:*, single direction)
- ✅ Australian FY date table (July 1 - June 30)
- ✅ 20+ explicit DAX measures
- ✅ Optimised for performance (<1 second query time)

Learning Outcomes

By the end of this project, you will be able to:

- ✅ Design and implement star schema data models
- ✅ Create relationships with correct cardinality and cross-filter direction
- ✅ Build custom date tables with Australian financial year logic
- ✅ Write 20+ DAX measures for business metrics (revenue, margins, time intelligence)
- ✅ Implement Australian FY time intelligence (YTD, QTD, YoY, QoQ)
- ✅ Calculate budget variance and performance metrics
- ✅ Use advanced DAX (TOPN, RANKX, CALCULATE, FILTER)
- ✅ Optimise model performance (calculated columns vs measures)
- ✅ Validate model accuracy with reconciliation

Why This Matters

According to Microsoft research, **poorly designed data models are the #1 cause of slow Power BI reports**. A well-designed star schema:

📊 Improves query performance

10-100x faster than normalised models

🎯 Simplifies DAX formulas

Relationships do the joining

🔍 Enables intuitive UX

Clear dimension hierarchies

🚀 Scales to millions of rows

Efficient storage and aggregation

📈 Supports complex analytics

Time intelligence, calculations

Enterprise Impact in Australia

Proper data modelling enables:

- ✅ **EOFY reporting accuracy** - June 30 year-end calculations
- ✅ **YTD/QTD comparisons** aligned to Australian FY (July-June)
- ✅ **Budget vs Actual variance** for financial planning
- ✅ **State-level performance analysis** (NSW, VIC, QLD)
- ✅ **GST reconciliation** - 10% tax validation across all transactions

2. Business Scenario Continuation

Where We Left Off (Video 1)

In Project 1, you built an ETL pipeline that:

- Connected 5 data sources (CSV, Excel, SQL Server, SharePoint, Web API)
- Cleaned 50,000+ transactions (removed duplicates, standardised states, calculated GST)
- Validated data quality (95%+ completeness, 100% GST accuracy)

Current state: Clean data in Power Query, ready to load to the model.

Executive Requirements (Project 2)

FreshMarket's CFO and Head of Operations need answers to:

Financial Questions:

- What is our **total revenue** and **gross profit** for FY2024?
- What is our **gross margin %** by state, category, and channel?
- How do we compare to **budget** (variance $ and %)?
- What is our **year-to-date (YTD) revenue** vs. prior year?
- What is our **quarter-to-date (QTD) performance** for Q3 FY2024?

Operational Questions:

- Which **top 5 products** drive the most revenue?
- Which **stores underperform** (bottom 10 by margin)?
- What is our **average order value (AOV)** by channel?
- How does **Click & Collect revenue** compare to In-Store and Online?
- What is our **30-day rolling revenue** trend?

Comparative Analysis:

- **YoY growth %** (FY2024 vs FY2023)
- **QoQ growth %** (Q3 vs Q2 FY2024)
- **State performance ranking** (NSW vs VIC vs QLD)
- **Product category trends** over time

Your Mission

Build a **star schema data model** that:

- Loads clean data from Power Query
- Establishes proper relationships between fact and dimensions
- Creates explicit DAX measures for all business metrics
- Supports fast, interactive analysis for 50+ executives

Success Criteria

Metric	Target	How to Measure
Model size	<50 MB	File → Options → Diagnostics
Query performance	<1 second	Performance Analyser
Relationship accuracy	100% valid joins	No blank rows in visuals
Measure accuracy	±1% vs source	Reconcile totals
Time intelligence	Correct FY alignment	YTD matches manual calc
User satisfaction	Intuitive filtering	Test with non-technical users

3. Star Schema Design Principles

What is a Star Schema?

A **star schema** is a data modelling technique where:

- **1 central fact table** contains numeric measures (sales, quantities, costs)
- **Multiple dimension tables** surround the fact (products, customers, dates)
- Relationships radiate from dimensions to the fact (like points of a star)

Why Star Schema? (vs. Normalised Database)

Aspect	Normalised DB	Star Schema
Query Performance	Slow (many JOINs)	Fast (single JOIN per dimension)
Storage	Minimal redundancy	Some denormalisation
User Experience	Complex queries	Simple drag-and-drop
BI Tools	Not optimised	Designed for analytics
Scalability	Good for OLTP	Excellent for OLAP

Example: To get "Revenue by Product Category":

- **Normalised:** JOIN Sales → Products → ProductCategories → Subcategories (3 joins)
- **Star Schema:** FactSales → DimProduct (1 join, category already in DimProduct)

Star Schema Best Practices

DO:

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

DON'T:

- ❌ Use **many-to-many** relationships (causes ambiguity)
- ❌ Use **bi-directional** cross-filter (performance issues)
- ❌ Store **descriptive text** in fact tables (use dimension keys instead)
- ❌ Create **snowflake schemas** (normalised dimensions) unless necessary
- ❌ Use **natural keys** that might change (use surrogate keys)
- ❌ Mix **grain levels** in the same fact table

Fact Table Grain

Grain = the level of detail in the fact table.

For FreshMarket FactSales:

- **Grain:** One row per transaction line item
- **Granularity:** 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 fact (different grain)
- ✅ Create separate facts: FactSales (daily) + FactBudget (monthly)

Dimension Types

Conformed Dimensions

Used across multiple fact tables

Example: DimDate used by FactSales, FactBudget, FactInventory

Degenerate Dimensions

Dimension data stored in the fact table (no separate dimension table)

Example: TransactionID (unique per row, no attributes to store)

Role-Playing Dimensions

Same dimension used multiple times with different roles

Example: DimDate as OrderDate, ShipDate, DeliveryDate

Slowly Changing Dimensions (SCD)

Type 1: Overwrite (e.g., customer address changes)

Type 2: Add new row with version (e.g., track product price history)

Type 3: Add column (e.g., PreviousCategory, CurrentCategory)

For FreshMarket, we use **Type 1 SCD** (simple overwrites, no history tracking).

4. Building the Data Model

Step 1: Load Clean Queries to Model

What to do:

From Prpoject 1, you have these clean queries in Power Query:

- clean_transactions (from messy CSV)
- DimStore (from SQL Server)
- DimProduct (from SQL Server)
- DimCustomer (from SQL Server)
- DimChannel (from SQL Server)
- DimDate (from SQL Server - we'll recreate this with Australian FY logic)

Verify that **Enable load** is checked for all clean queries:

1. Open Power Query Editor (Home → Transform data)
2. For each clean query:
   - Right-click query name
   - Ensure **Enable load** is ✅ checked
3. Click **Close & Apply**

Power BI loads all queries into the model.

> **Why it matters:** Only queries with "Enable load" appear in the Fields pane and consume model memory. Staging queries (load disabled) are reference-only.

How to validate:

After closing Power Query:

1. In Power BI Desktop, check the **Fields** pane (right side)
2. You should see these tables:
   - clean_transactions
   - DimStore
   - DimProduct
   - DimCustomer
   - DimChannel
   - DimDate

Each table shows a table icon and column list.

Step 2: Rename Fact Table

What to do:

The clean_transactions table should be renamed to FactSales for clarity:

1. In the Fields pane, right-click clean_transactions
2. Select **Rename**
3. New name: FactSales
4. Press Enter

Why it matters: Clear naming conventions help team members understand the model. Fact tables typically start with "Fact", dimensions with "Dim".

How to validate:

Fields pane shows: FactSales (not clean_transactions)

Step 3: Organise Tables

What to do:

Arrange tables logically in the Fields pane:

1. Click and drag FactSales to the top
2. Group dimensions below in alphabetical order:
   - DimChannel
   - DimCustomer
   - DimDate
   - DimProduct
   - DimStore

Why it matters: Organised field lists improve user experience. Users find dimensions faster when they're alphabetically sorted.

Step 4: Review Table Schemas

What to do:

Click on **Model View** (left sidebar, third icon from top - looks like relationship diagram).

You'll see a visual representation of all tables.

For each table, verify columns:

FactSales (measures only):

- TransactionID (Text) - degenerate dimension
- TransactionDate (Date) - will relate to DimDate
- StoreID (Text) - foreign key to DimStore
- ProductID (Text) - foreign key to DimProduct
- CustomerID (Text) - foreign key to DimCustomer
- ChannelID (Text) - foreign key to DimChannel
- Quantity (Whole Number) - measure
- UnitPrice (Currency) - measure
- Discount (Decimal) - measure
- Revenue (Currency) - measure
- Cost (Currency) - measure
- GST (Currency) - measure
- TransactionTime (Time) - optional analysis

DimProduct:

- ProductID (Text) - primary key
- ProductName (Text)
- Category (Text)
- Subcategory (Text)
- Supplier (Text)
- UnitCost (Currency)
- Brand (Text)
- PackSize (Text)

DimStore:

- StoreID (Text) - primary key
- StoreName (Text)
- State (Text)
- Postcode (Text)
- Region (Text)
- ManagerName (Text)
- OpenDate (Date)
- StoreSize (Text)

DimCustomer:

- CustomerID (Text) - primary key
- CustomerName (Text)
- Email (Text)
- State (Text)
- Postcode (Text)
- LoyaltyTier (Text)
- JoinDate (Date)
- DateOfBirth (Date)

DimChannel:

- ChannelID (Text) - primary key
- ChannelName (Text)
- ChannelType (Text)

DimDate (we'll recreate this shortly):

- Date (Date) - primary key
- Year, FY, Quarter, Month, etc.

Why it matters: Correct data types ensure accurate relationships and aggregations. Currency types display with $ symbols, Whole Numbers SUM correctly.

How to validate:

For each table, click on it in Model View. Review the column list on the right. Verify data types match the list above.

5. Creating Relationships

Understanding Relationships

Relationship definition: A connection between two tables based on matching column values.

Components:

- **From table (dimension):** The "one" side (unique values)
- **To table (fact):** The "many" side (repeating values)
- **Cardinality:** 1:* (one-to-many), 1:1 (one-to-one), *:* (many-to-many)
- **Cross-filter direction:** Single (one-way) or Both (bi-directional)

How filtering works:

- Selecting a value in a dimension table **filters** the fact table
- Example: Select "NSW" in DimStore → FactSales shows only NSW transactions

Step 1: Create Store Relationship

What to do:

In **Model View**:

1. Locate DimStore table (shows all columns)
2. Locate FactSales table
3. Click and drag StoreID from **DimStore** to StoreID in **FactSales**

A line appears connecting the two tables.

1. Double-click the relationship line

The **Edit relationship** dialogue appears:

1. Verify settings:
   - **From:** DimStore (StoreID)
   - **To:** FactSales (StoreID)
   - **Cardinality:** Many to one (*:1)
   - **Cross filter direction:** Single
   - **Make this relationship active:** ✅ Checked
2. Click **OK**

> **Why it matters:** This relationship allows users to filter sales by store name, state, region, etc. The 1:* cardinality ensures each store can have many transactions, but each transaction belongs to one store.

How to validate:

The relationship line should show:

- **1** on the DimStore side (one store)
- ***** on the FactSales side (many transactions)
- Arrow pointing from DimStore → FactSales (filter direction)

Step 2: Create Product Relationship

What to do:

1. Drag ProductID from **DimProduct** to ProductID in **FactSales**
2. Double-click the relationship line
3. Verify:
   - Cardinality: Many to one (*:1)
   - Cross filter direction: Single
   - Active: ✅
4. Click **OK**

How to validate:

Relationship shows **1** (DimProduct) → ***** (FactSales)

Step 3: Create Customer Relationship

What to do:

1. Drag CustomerID from **DimCustomer** to CustomerID in **FactSales**
2. Edit relationship:
   - Cardinality: Many to one (*:1)
   - Cross filter direction: Single
   - Active: ✅
3. Click **OK**

Note: Some FactSales rows have NULL CustomerID (anonymous purchases). Power BI handles this with a blank row in visuals.

Step 4: Create Channel Relationship

What to do:

1. Drag ChannelID from **DimChannel** to ChannelID in **FactSales**
2. Edit relationship:
   - Cardinality: Many to one (*:1)
   - Cross filter direction: Single
   - Active: ✅
3. Click **OK**

Step 5: Create Date Relationship (Temporary)

What to do:

We'll create a temporary date relationship using the TransactionDate field. Later, we'll recreate DimDate with Australian FY logic.

1. Drag Date from **DimDate** to TransactionDate in **FactSales**
2. Edit relationship:
   - Cardinality: Many to one (*:1)
   - Cross filter direction: Single
   - Active: ✅
3. Click **OK**

Note: We'll replace this relationship after creating the custom date table.

Step 6: Verify All Relationships

What to do:

Click **Home** ribbon → **Manage relationships**

The **Manage relationships** dialogue shows all relationships:

From Table	From Column	To Table	To Column	Active	Cardinality	Cross Filter
DimStore	StoreID	FactSales	StoreID	✅	1:*	Single
DimProduct	ProductID	FactSales	ProductID	✅	1:*	Single
DimCustomer	CustomerID	FactSales	CustomerID	✅	1:*	Single
DimChannel	ChannelID	FactSales	ChannelID	✅	1:*	Single
DimDate	Date	FactSales	Transaction Date	✅	1:*	Single

Why it matters: This dialogue provides a tabular view of all relationships. It's easier to spot configuration errors here than in the visual diagram.

How to validate:

All 5 relationships should be:

- ✅ Active
- Cardinality: 1:* (or *:1, same meaning)
- Cross filter direction: Single

Click **Close**.

Common Relationship Mistakes

Mistake 1: Bi-directional cross-filter

❌ **Wrong:** Cross filter direction = Both

✅ **Right:** Cross filter direction = Single

Why: Bi-directional filters can cause:

- Performance degradation (filters propagate in both directions)
- Ambiguous filter paths (which table filters which?)
- Incorrect aggregations

When to use bi-directional: Rarely. Only for specific many-to-many scenarios (not in our model).

Mistake 2: Many-to-many relationships

❌ **Wrong:** Cardinality = *:*

✅ **Right:** Cardinality = 1:*

Why: Many-to-many relationships:

- Require careful DAX to avoid double-counting
- Slow down queries
- Can produce unexpected results

When to use many-to-many: Advanced scenarios like role-playing dimensions or bridge tables.

Mistake 3: Inactive relationships

❌ **Wrong:** Relationship exists but unchecked "Active"

✅ **Right:** Active = ✅ (unless intentionally using USERELATIONSHIP in DAX)

Why: Inactive relationships don't filter unless explicitly activated in DAX using USERELATIONSHIP.

6. Australian FY Date Table

Why a Custom Date Table?

Power BI can auto-generate a date table, but it:

- ❌ Uses calendar year (Jan-Dec), not Australian FY (Jul-Jun)
- ❌ Lacks custom columns (EOFY flag, FY labels)
- ❌ Doesn't support multiple fiscal years
- ❌ Can't be customised for public holidays, business days

For Australian businesses, a **custom date table** is essential.

Step 1: Delete Existing DimDate

What to do:

1. In **Model View**, locate the DimDate table
2. Note the relationship to FactSales
3. Right-click **DimDate** → **Delete from model**
4. Confirm deletion

The relationship to FactSales is also deleted.

Why it matters: We're replacing the SQL Server date table with a DAX-generated table that has Australian FY logic.

Step 2: Create Custom Date Table with DAX

What to do:

1. Click **Report View** (left sidebar, top icon - bar chart)
2. Click **Home** ribbon → **Enter data**
3. In the "Create Table" dialogue, click **Cancel** (we'll use DAX instead)
4. Click **Modelling** ribbon → **New table**

The formula bar appears.

1. Enter this DAX formula:

```
DimDate =
ADDCOLUMNS(
    CALENDAR(DATE(2022, 1, 1), DATE(2025, 12, 31)),
    "Year", YEAR([Date]),
    "FY", "FY" & IF(
        MONTH([Date]) >= 7,
        YEAR([Date]) + 1,
        YEAR([Date])
    ),
    "Quarter", "Q" & QUARTER([Date]),
    "Month", MONTH([Date]),
    "MonthName", FORMAT([Date], "MMMM"),
    "MonthYear", FORMAT([Date], "MMM-YYYY"),
    "MonthShort", FORMAT([Date], "MMM"),
    "DayOfWeek", FORMAT([Date], "dddd"),
    "DayOfMonth", DAY([Date]),
    "IsWeekend", IF(WEEKDAY([Date], 2) >= 6, 1, 0),
    "IsEOFY", IF(MONTH([Date]) = 6 && DAY([Date]) = 30, 1, 0),
    "FYQuarter", "FY" & IF(MONTH([Date]) >= 7, YEAR([Date]) + 1, YEAR([Date]))
        & " Q" & SWITCH(
            TRUE(),
            MONTH([Date]) >= 7 && MONTH([Date]) <= 9, 1,
            MONTH([Date]) >= 10 && MONTH([Date]) <= 12, 2,
            MONTH([Date]) >= 1 && MONTH([Date]) <= 3, 3,
            4
        ),
    "FYMonth", SWITCH(
        TRUE(),
        MONTH([Date]) >= 7, MONTH([Date]) - 6,
        MONTH([Date]) + 6
    )
)
```

1. Press **Enter**

Power BI creates the DimDate table.

> **Why it matters:** This DAX formula:
>
> - Generates dates from Jan 1, 2022 to Dec 31, 2025 (covers all transaction dates)
> - Calculates **FY** (Financial Year): If month ≥ 7, FY = year + 1, else FY = year
> - Marks **EOFY** (End of Financial Year): June 30 = 1, all other dates = 0
> - Calculates **FYQuarter**: Q1 = Jul-Sep, Q2 = Oct-Dec, Q3 = Jan-Mar, Q4 = Apr-Jun
> - Calculates **FYMonth**: Month 1 = July, Month 12 = June

How to validate:

1. Click on the new **DimDate** table in Fields pane
2. Expand to see columns
3. Click **Data View** (left sidebar, middle icon - table grid)
4. Select DimDate table from Fields pane

You should see rows like:

Date	Year	FY	Quarter	Month	MonthName	IsEOFY	FYQuarter	FYMonth
2024-06-30	2024	FY2024	Q2	6	June	1	FY2024 Q4	12
2024-07-01	2024	FY2025	Q3	7	July	0	FY2025 Q1	1

Verify:

- June 30, 2024 → **FY2024** (end of FY2024)
- July 1, 2024 → **FY2025** (start of FY2025)
- IsEOFY = 1 only for June 30 dates

Step 3: Mark as Date Table

What to do:

1. Still in **Data View**, with DimDate selected
2. Click **Table tools** ribbon → **Mark as date table**
3. In the dialogue:
    - Select **Date** column as the date column
    - Click **OK**

A small calendar icon appears next to DimDate in the Fields pane.

Why it matters: Marking as a date table:

- Enables DAX time intelligence functions (TOTALYTD, SAMEPERIODLASTYEAR, etc.)
- Ensures Power BI treats it as a date dimension
- Validates that dates are unique and continuous

Step 4: Recreate Date Relationship

What to do:

1. Return to **Model View**
2. Drag Date from **DimDate** to TransactionDate in **FactSales**
3. Edit relationship:
    - Cardinality: Many to one (*:1)
    - Cross filter direction: Single
    - Active: ✅
4. Click **OK**

Step 5: Create Date Hierarchies

What to do:

1. In **Data View**, select DimDate table
2. Right-click **FY** column → **Create hierarchy**
3. Rename hierarchy: FY Hierarchy
4. Right-click the FY Hierarchy → **Add to hierarchy** → Select:
    - FYQuarter
    - MonthName
    - Date

The hierarchy now has levels: FY → FYQuarter → MonthName → Date

1. Create another hierarchy called Calendar Hierarchy:
    - Year
    - Quarter
    - MonthName
    - Date

Why it matters: Hierarchies enable drill-down in visuals. Users can start with FY2024, drill into Q1, then drill into July, then daily transactions.

How to validate:

In Fields pane, DimDate shows two hierarchies:

- FY Hierarchy (FY > FYQuarter > MonthName > Date)
- Calendar Hierarchy (Year > Quarter > MonthName > Date)

7. Core DAX Measures - Revenue & Costs

Create Measures Table

What to do:

It's best practice to create a dedicated table for measures (not tied to any fact/dimension):

1. In **Report View**, click **Modelling** ribbon → **New table**
2. Formula: `_Measures = { 0 }`
3. Press Enter

This creates a single-row table with one column (we don't actually use it).

1. Right-click the **Column** column in _Measures → **Hide**

Now you have a clean measures table.

> **Why it matters:** Grouping measures in a dedicated table:
>
> - Makes them easy to find (all in one place)
> - Prevents clutter in fact/dimension tables
> - Follows enterprise best practices

Measure 1: Total Revenue

What to do:

1. Click on the _Measures table
2. Click **Modelling** ribbon → **New measure**
3. Formula bar appears. Enter:

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

1. Press **Enter**

The measure appears under _Measures.

1. Format the measure:
   - Select Total Revenue in Fields pane
   - Click **Measure tools** ribbon
   - Format: **Currency** → **$ English (Australia)**
   - Decimal places: **2**

Why it matters:

- SUM aggregates revenue across all filtered transactions
- Explicit measures (vs. implicit aggregations) give you control over filter context
- Currency formatting displays values as $1,234.56 AUD

How to validate:

Create a Card visual:

1. Click **Report View**
2. Click **Card** visual (visualisations pane)
3. Drag Total Revenue to the **Fields** well

The card should show a large number (e.g., **$45,234,123.45**).

To validate accuracy:

- Go to Data View → FactSales
- Manually SUM the Revenue column (use CTRL+Click header → Quick measure)
- Compare to the card value (should match within rounding)

Measure 2: Total Cost

What to do:

1. Select _Measures table
2. New measure:

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

1. Format: Currency, $ AUD, 2 decimals

How to validate:

Add to a Card visual. Should show ~$30M-35M (lower than revenue).

Measure 3: Total Quantity

What to do:

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

Format: **Whole Number**, **Thousands separator**

How to validate:

Card visual should show ~250,000-300,000 units.

Measure 4: Total Transactions

What to do:

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

Format: **Whole Number**, **Thousands separator**

Why DISTINCTCOUNT? Even though we removed duplicates in Power Query, using DISTINCTCOUNT ensures we count unique transactions only (defensive coding).

How to validate:

Card visual should show ~50,000 transactions (matching source data).

Measure 5: Average Order Value (AOV)

What to do:

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

Format: **Currency**, $ AUD, 2 decimals

Why DIVIDE? DIVIDE handles divide-by-zero errors gracefully (returns 0 instead of error). Safer than [Total Revenue] / [Total Transactions].

How to validate:

Card visual should show ~$900-1,000 AOV.

Manual check: Total Revenue / Total Transactions should equal AOV.

8. Margin & Profitability Measures

Measure 6: Gross Profit

What to do:

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

Format: **Currency**, $ AUD, 2 decimals

Why reference other measures? Reusing measures ([Total Revenue], [Total Cost]) ensures consistency. If revenue calculation changes, gross profit updates automatically.

How to validate:

Card visual should show ~$10M-15M profit.

Measure 7: Gross Margin %

What to do:

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

Format: **Percentage**, 2 decimals

How to validate:

Card visual should show ~25-35% margin.

Manual check: Gross Profit / Total Revenue × 100 = Gross Margin %

Measure 8: Average Margin Per Transaction

What to do:

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

Format: **Currency**, $ AUD, 2 decimals

How to validate:

Should show ~$250-350 per transaction.

9. Time Intelligence - Australian FY

Understanding DAX Time Intelligence

Time intelligence functions compare data across time periods:

- **YTD (Year-to-Date):** Sum from July 1 to current date (Australian FY)
- **QTD (Quarter-to-Date):** Sum from quarter start to current date
- **MTD (Month-to-Date):** Sum from month start to current date
- **Prior Year:** Same period last year
- **YoY% (Year-over-Year %):** Growth rate vs. prior year

Requirements:

- ✅ Date table marked as date table
- ✅ Relationship between date table and fact table
- ✅ Continuous date range (no gaps)

Measure 9: YTD Revenue (Australian FY)

What to do:

```
YTD Revenue =
TOTALYTD(
    [Total Revenue],
    DimDate[Date],
    "6/30"
)
```

Format: **Currency**, $ AUD, 2 decimals

Explanation:

- TOTALYTD calculates year-to-date total
- "6/30" specifies that the fiscal year ends on June 30 (Australian FY)
- Without "6/30", it would default to Dec 31 (calendar year)

How to validate:

Create a Table visual:

1. Rows: DimDate[MonthYear] (sorted by date)
2. Values: [Total Revenue], [YTD Revenue]

Example for FY2024:

MonthYear	Total Revenue	YTD Revenue
Jul-2023	$3,500,000	$3,500,000
Aug-2023	$3,700,000	$7,200,000
Sep-2023	$3,600,000	$10,800,000
...	...	...
Jun-2024	$4,200,000	$45,000,000
Jul-2024	$3,800,000	$3,800,000 (resets for FY2025)

YTD should accumulate each month, then reset in July.

Measure 10: QTD Revenue

What to do:

```
QTD Revenue =
TOTALQTD(
    [Total Revenue],
    DimDate[Date]
)
```

Format: **Currency**, $ AUD, 2 decimals

How to validate:

QTD should accumulate within each quarter, reset at quarter start.

Measure 11: MTD Revenue

What to do:

```
MTD Revenue =
TOTALMTD(
    [Total Revenue],
    DimDate[Date]
)
```

Format: **Currency**, $ AUD, 2 decimals

Measure 12: Prior Year Revenue

What to do:

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

Format: **Currency**, $ AUD, 2 decimals

Explanation:

- CALCULATE modifies filter context
- SAMEPERIODLASTYEAR shifts the date filter back one year
- Example: If current context is July 2024, this returns revenue for July 2023

How to validate:

Create a Table visual:

MonthYear	Total Revenue	Prior Year Revenue
Jul-2024	$3,800,000	$3,500,000 (Jul-2023)
Aug-2024	$3,900,000	$3,700,000 (Aug-2023)

Prior Year should show values from 12 months ago.

Measure 13: YoY Revenue %

What to do:

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

Format: **Percentage**, 2 decimals

Explanation:

- BLANK() returns blank if no prior year data (instead of 0% or error)
- Formula: (Current - Prior) / Prior = growth rate

How to validate:

Create a Table visual:

MonthYear	Total Revenue	Prior Year Revenue	YoY Revenue %
Jul-2024	$3,800,000	$3,500,000	8.57%
Aug-2024	$3,900,000	$3,700,000	5.41%

YoY% should be positive for growth, negative for decline.

Measure 14: QoQ Revenue % (Quarter-over-Quarter)

What to do:

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

Format: **Percentage**, 2 decimals

Explanation:

- VAR creates variables for cleaner code
- DATEADD(..., -1, QUARTER) shifts back one quarter
- Returns blank if no prior quarter data

How to validate:

Table visual with FYQuarter:

FYQuarter	Total Revenue	QoQ Revenue %
FY2024 Q1	$10,800,000	(blank - no prior quarter)
FY2024 Q2	$11,200,000	3.70%
FY2024 Q3	$11,500,000	2.68%

10. Budget Variance Analysis

Understanding Budget vs Actual

Budget data comes from the Excel file we loaded in Video 1. It needs to be connected to our model.

Step 1: Load Budget Data

What to do:

If you haven't already loaded the budget data:

1. Go to Power Query Editor
2. Find stg_budget (from Excel file)
3. Create a new query: Right-click stg_budget → **Reference**
4. Rename: DimBudget
5. Transformations:
    - Ensure Month column is formatted as Date (first of month)
    - Ensure ProductID is Text
    - Ensure budget columns are Currency/Number
6. Enable load
7. Close & Apply

How to validate:

DimBudget appears in Fields pane with columns:

- Month (Date)
- ProductID (Text)
- BudgetRevenue (Currency)
- BudgetUnits (Whole Number)
- ForecastRevenue (Currency)
- ForecastUnits (Whole Number)

Step 2: Create Relationships to Budget

What to do:

Budget data has two foreign keys: Month and ProductID.

1. In **Model View**, drag ProductID from **DimProduct** to ProductID in **DimBudget**
    - Cardinality: 1:*
    - Cross filter: Single
    - Active: ✅
2. Drag Date from **DimDate** to Month in **DimBudget**
    - Cardinality: 1:*
    - Cross filter: Single
    - Active: ✅

> **Why it matters:** Budget is at **monthly grain by product**. We need relationships to both Product and Date dimensions to filter correctly.

Measure 15: Budget Revenue

What to do:

```
Budget Revenue = SUM(DimBudget[BudgetRevenue])
```

Format: **Currency**, $ AUD, 2 decimals

How to validate:

Card visual should show total budget (e.g., ~$50M for the year).

Measure 16: Variance $ (Actual - Budget)

What to do:

```
Variance $ = [Total Revenue] - [Budget Revenue]
```

Format: **Currency**, $ AUD, 2 decimals

How to validate:

Positive variance = over budget (good)

Negative variance = under budget (bad)

Measure 17: Variance % (Actual vs Budget)

What to do:

```
Variance % =
DIVIDE(
    [Variance $],
    [Budget Revenue],
    BLANK()
)
```

Format: **Percentage**, 2 decimals

How to validate:

If Actual = $5M, Budget = $4.5M → Variance % = 11.11%

Measure 18: Budget Achievement %

What to do:

```
Budget Achievement % =
DIVIDE(
    [Total Revenue],
    [Budget Revenue],
    BLANK()
)
```

Format: **Percentage**, 2 decimals

Explanation:

- 100% = exactly on budget
- >100% = exceeded budget
- <100% = missed budget

How to validate:

Table visual by Month and Product showing achievement %.

11. Advanced DAX - Top N & Ranking

Measure 19: Top 5 Products Revenue

What to do:

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

Format: **Currency**, $ AUD, 2 decimals

Explanation:

- ALL(DimProduct[ProductName]) removes any product filters
- TOPN(5, ..., [Total Revenue], DESC) returns top 5 products by revenue
- CALCULATE reapplies the top 5 filter and calculates revenue

How to validate:

Card visual shows revenue from only the top 5 products (should be ~20-30% of total).

Measure 20: Product Rank by Revenue

What to do:

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

Format: **Whole Number**

Explanation:

- ISINSCOPE checks if ProductName is in the visual (prevents rank at wrong grain)
- RANKX ranks all products by revenue (descending)
- DENSE ranking: 1, 2, 3, 3, 4 (ties get same rank, next rank continues)

How to validate:

Table visual:

ProductName	Total Revenue	Product Rank
Product A	$500,000	1
Product B	$450,000	2
Product C	$450,000	2 (tie)
Product D	$400,000	3

Measure 21: Rolling 30-Day Revenue

What to do:

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

Format: **Currency**, $ AUD, 2 decimals

Explanation:

- DATESINPERIOD creates a date range
- MAX(DimDate[Date]) finds the latest date in current context
- -30, DAY goes back 30 days from that date
- Returns sum of revenue for those 30 days

How to validate:

Line chart with Date on X-axis and [Rolling 30d Revenue] on Y-axis should show smooth rolling average (less volatile than daily revenue).

Measure 22: Store Rank by Margin %

What to do:

```
Store Rank by Margin =
IF(
    ISINSCOPE(DimStore[StoreName]),
    RANKX(
        ALL(DimStore[StoreName]),
        [Gross Margin %],
        ,
        DESC,
        DENSE
    ),
    BLANK()
)
```

Format: **Whole Number**

How to validate:

Table visual showing stores ranked by margin % (1 = highest margin).

12. Calculated Columns vs Measures

When to Use Calculated Columns

Calculated columns:

- Computed **row-by-row** during data refresh
- Stored in the model (uses memory)
- Can be used in **slicers, filters, rows/columns** of visuals
- Evaluated once, doesn't change with filter context

Use cases:

- Creating **categories** or **groups** (e.g., Age Group from Date of Birth)
- **Static calculations** needed for filtering (e.g., Year-Month concatenation)
- **Conditional flags** (e.g., HighValueCustomer = Revenue > $10,000)

Example - Customer Age Group (Calculated Column):

1. Go to **Data View**
2. Select **DimCustomer** table
3. Click **Table tools** → **New column**
4. Formula:

```
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+"
    )
```

1. Press Enter

Now Age Group appears as a column in DimCustomer. Users can filter/slice by it.

When to Use Measures

Measures:

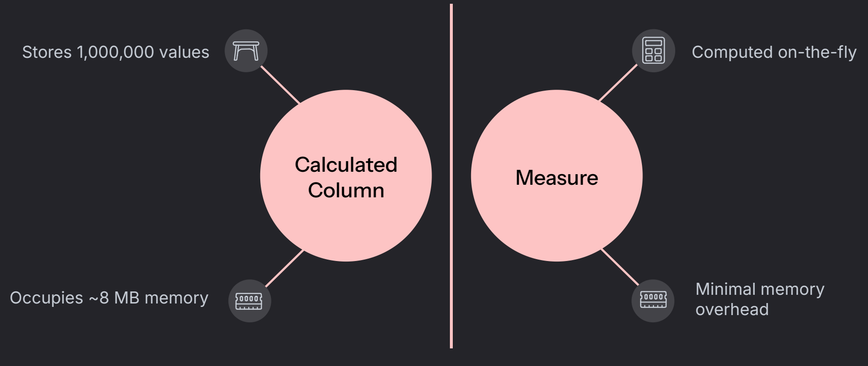
- Computed **on-the-fly** during visual rendering
- Not stored (uses no memory)
- Cannot be used in slicers/filters (only values)
- Evaluated dynamically based on filter context

Use cases:

- **Aggregations** (SUM, COUNT, AVERAGE, MIN, MAX)
- **Ratios and percentages** (Margin %, Growth %)
- **Time intelligence** (YTD, YoY, Rolling averages)
- **Anything that changes** based on filters

Best practice: Default to **measures**. Only use calculated columns when you need to filter/slice by the calculated value.

Memory Impact Comparison



Example:

- **Fact table:** 1 million rows
- **Calculated column:** Profit = Revenue - Cost
  - Stores 1 million values in model
  - Increases model size by ~8 MB (assuming 8 bytes per value)
- **Measure:** Profit = [Total Revenue] - [Total Cost]
  - Stores nothing in model
  - Calculated on-the-fly from aggregated values

Rule of thumb: If a calculation can be a measure, make it a measure (better performance).

13. Model Optimisation

Check Model Size

What to do:

1. Click **File** → **Options and settings** → **Options**
2. Under **Data Load**, check **Enable load to data model**
3. Click **Diagnostics** → **Show model size**

Current model size should be displayed (target: <50 MB for this dataset).

> **Why it matters:** Large models:
>
> - Load slowly
> - Consume more RAM
> - Refresh slowly
> - May hit capacity limits in Power BI Service

Optimisation techniques:

- Remove unused columns (Table tools → Remove columns)
- Use calculated columns sparingly
- Disable load for staging queries (already done in Video 1)
- Remove unnecessary relationships

How to validate:

Model size <50 MB = ✅

Model size >100 MB = ⚠️ Investigate

Optimise Data Types

What to do:

Review data types in **Model View**:

Text columns:

Only use for descriptive attributes

- ✅ ProductName, StoreName, CustomerName
- ❌ ProductID, StoreID (should be Text, but consider shorter keys)

Whole Numbers:

Use instead of Decimal when possible

- ✅ Quantity, Month (1-12)

Currency:

Use for monetary values

- ✅ Revenue, Cost, Budget

Boolean:

Use for flags

- ✅ IsWeekend (0/1), IsEOFY (0/1)

Why it matters: Smaller data types = smaller model:

- Text uses 2 bytes per character
- Whole Number uses 8 bytes
- Decimal uses 8 bytes
- Boolean uses 1 byte

Remove Auto-Date Tables

What to do:

Power BI auto-creates date tables for every date column. Disable this:

1. **File** → **Options** → **Data Load**
2. Under **Time intelligence**, uncheck **Auto date/time**
3. Click **OK**

Why it matters: Auto-date tables:

- Duplicate the custom DimDate table
- Bloat model size
- Use calendar year (not Australian FY)

Optimise Relationships

What to do:

Review relationships in **Manage relationships** dialogue:

All relationships should be:

- ✅ 1:* cardinality (never *:*)
- ✅ Single cross-filter direction (never Both)
- ✅ Active (unless using USERELATIONSHIP)

Why it matters: Bi-directional and many-to-many relationships slow down queries significantly.

14. Validation & Testing

Create Validation Dashboard

What to do:

Create a new report page named "Validation":

1. Add a **Table** visual with:
   - Rows: Measure names (manual entry)
   - Values: Measure values

Example Validation Table:

Metric	Value	Expected	Status
Total Revenue	$45.2M	~$40-50M	✅
Total Cost	$32.1M	~$30-35M	✅
Gross Margin %	29.0%	~25-35%	✅
Total Transactions	50,143	~50,000	✅

1. Add a **Card** visual for each measure
2. Add a **Matrix** visual showing:
   - Rows: DimDate[FY]
   - Columns: _Measures (drag all measures to Values)

This shows all measures by financial year for reconciliation.

Why it matters: Validation dashboards catch errors before reports go to executives.

Reconcile with Source Data

What to do:

Compare Power BI totals with source data:

1. Open data_profile_report.csv from Video 1
2. Note total rows for FactSales (should be ~52,500 including quality issues)
3. In Power BI, create a Card with [Total Transactions]
4. Should show ~50,000 (after removing duplicates/errors)

Manual calculation:

- Open fact_sales.csv in Excel
- SUM(Revenue column)
- Compare to Power BI [Total Revenue]
- Difference should be <2% (due to quality issue removal)

Why it matters: Reconciliation proves data integrity. If Power BI totals don't match source, investigate:

- Relationships incorrect (causing double-counting)
- Filters applied (removing valid data)
- Transformations wrong (data loss in Power Query)

Test Time Intelligence

What to do:

Validate time intelligence calculations manually:

YTD Revenue for December 2023:

1. Filter FactSales to July 1 - Dec 31, 2023
2. SUM(Revenue) in Excel = Manual YTD
3. In Power BI, add filter: DimDate[MonthYear] = "Dec-2023"
4. Check [YTD Revenue]
5. Should match manual YTD

YoY% for August 2024:

1. Revenue for Aug 2024 = $3.9M
2. Revenue for Aug 2023 = $3.7M
3. Manual YoY% = (3.9 - 3.7) / 3.7 = 5.41%
4. In Power BI, filter to Aug 2024
5. Check [YoY Revenue %]
6. Should show 5.41%

Why it matters: Time intelligence errors are common (wrong fiscal year end, incorrect date relationships). Manual validation prevents executive reports showing wrong YTD/YoY.

15. Common DAX Mistakes

Mistake 1: Using Calculated Columns for Aggregations

❌ **Wrong:**

```
// Calculated column in FactSales
Profit Column = FactSales[Revenue] - FactSales[Cost]
```

Then SUM(Profit Column) in visuals.

✅ **Right:**

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

Why: Calculated column stores 50,000 values. Measure computes on-the-fly from aggregated values. Measure is faster and uses less memory.

Mistake 2: Not Using DIVIDE for Division

❌ **Wrong:**

```
Margin % = [Gross Profit] / [Total Revenue]
```

Problem: If [Total Revenue] = 0, returns error.

✅ **Right:**

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

Fix: DIVIDE returns 0 (or BLANK) when denominator is zero.

Mistake 3: Forgetting to Mark Date Table

❌ **Wrong:** Create date table but don't mark it as date table.

Problem: TOTALYTD, SAMEPERIODLASTYEAR don't work.

✅ **Right:** Table tools → Mark as date table → Select date column.

Mistake 4: Using "6-30" Instead of "6/30" in TOTALYTD

❌ **Wrong:**

```
YTD Revenue = TOTALYTD([Total Revenue], DimDate[Date], "6-30")
```

Problem: "6-30" is interpreted as June 30 minus 30 = error.

✅ **Right:**

```
YTD Revenue = TOTALYTD([Total Revenue], DimDate[Date], "6/30")
```

Fix: Use forward slash "/" for dates.

Mistake 5: Not Using Variables in Complex Measures

❌ **Wrong:**

```
Margin % = (SUM(FactSales[Revenue]) - SUM(FactSales[Cost])) / SUM(FactSales[Revenue])
```

Problem: SUM(FactSales[Revenue]) calculated twice (inefficient).

✅ **Right:**

```
Margin % =
VAR Revenue = SUM(FactSales[Revenue])
VAR Cost = SUM(FactSales[Cost])
VAR Profit = Revenue - Cost
RETURN DIVIDE(Profit, Revenue, 0)
```

Fix: Variables improve readability and performance.

Mistake 6: Circular Dependencies

❌ **Wrong:**

```
Measure A = [Measure B] + 10
Measure B = [Measure A] * 2
```

Problem: Circular reference (A depends on B, B depends on A).

✅ **Right:** Redesign logic to remove circularity.

Mistake 7: Not Understanding Filter Context

❌ **Wrong:** Expecting [Total Revenue] to always show grand total.

Problem: [Total Revenue] changes based on filters in the visual. If user selects "NSW", [Total Revenue] shows only NSW revenue.

✅ **Right:** Understand that measures respond to filter context. Use ALL() to ignore filters when needed:

```
% of Total = DIVIDE([Total Revenue], CALCULATE([Total Revenue], ALL(DimStore)))
```

16. Portfolio Documentation

GitHub Repository Structure

```
FreshMarket-PowerBI-DataModel/
 ├─── README.md (project overview)
 ├─── FreshMarket_Model.pbix (Power BI file)
 ├─── dax_measures_library.txt (all DAX formulas)
 ├─── model_diagram.png (screenshot of Model View)
 ├─── validation_results.xlsx (reconciliation spreadsheet)
 └─── documentation/
    ├─── star_schema_design.md
    ├─── dax_patterns.md
    └─── time_intelligence_guide.md
```

README Template

```
# FreshMarket Australia - Data Model & DAX

## Project Overview
Built a star schema data model for FreshMarket Australia with 20+ DAX measures
for financial and operational analysis aligned to Australian fiscal year (July-June).

## Business Context
**Industry:** Australian Grocery Retail
**Challenge:** Executives needed fast, accurate reporting for 50,000+ transactions
across 50 stores
**Solution:** Star schema with 1 fact table + 5 dimensions, Australian FY time intelligence

## Data Model Design

### Star Schema Architecture
**Fact Table:**
- FactSales (50,143 transactions)
- Grain: One row per transaction
- Measures: Revenue, Cost, Quantity, GST

**Dimension Tables:**
- DimProduct (2,000 products across 8 categories)
- DimStore (50 stores: NSW, VIC, QLD)
- DimCustomer (20,000 loyalty members)
- DimChannel (3 channels: In-Store, Online, Click & Collect)
- DimDate (2022-2025 with Australian FY logic)

**Relationships:**
- All 1:* cardinality (one-to-many)
- Single-direction cross-filter
- No circular dependencies

### DAX Measures Library (20+ Measures)
**Revenue & Cost:**
- Total Revenue, Total Cost, Gross Profit, Gross Margin %
- Average Order Value, Total Transactions

**Time Intelligence (Australian FY):**
- YTD Revenue (fiscal year: July 1 - June 30)
- QTD Revenue, MTD Revenue
- Prior Year Revenue, YoY Revenue %, QoQ Revenue %
- Rolling 30-Day Revenue

**Budget Variance:**
- Budget Revenue, Variance $, Variance %, Budget Achievement %

**Advanced Analytics:**
- Top 5 Products Revenue
- Product Rank by Revenue (RANKX)
- Store Rank by Margin (DENSE ranking)

### Key Technical Achievements
```

16. Portfolio Documentation (continued)

DAX Measures Library Export

Create a text file with all DAX formulas:

dax_measures_library.txt:

```
================================================================================
FRESHMARKET AUSTRALIA - DAX MEASURES LIBRARY
================================================================================

## REVENUE & COST 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)

## MARGIN & PROFITABILITY

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

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

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

## TIME INTELLIGENCE (AUSTRALIAN FY)

YTD Revenue = TOTALYTD([Total Revenue], DimDate[Date], "6/30")

QTD Revenue = TOTALQTD([Total Revenue], DimDate[Date])

MTD Revenue = TOTALMTD([Total Revenue], DimDate[Date])

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

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

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

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

## BUDGET VARIANCE

Budget Revenue = SUM(DimBudget[BudgetRevenue])

Variance $ = [Total Revenue] - [Budget Revenue]

Variance % = DIVIDE([Variance $], [Budget Revenue], BLANK())

Budget Achievement % = DIVIDE([Total Revenue], [Budget Revenue], BLANK())

## ADVANCED ANALYTICS

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

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

Store Rank by Margin = IF(ISINSCOPE(DimStore[StoreName]), RANKX(ALL(DimStore[StoreName]), [Gross Margin %], , DESC, DENSE), BLANK())

================================================================================
```

17. Interview Preparation

Elevator Pitch (30 seconds)

> "I designed a star schema data model for a 50-store Australian grocery chain, connecting 1 fact table with 5 dimensions using proper 1-to-many relationships. I wrote 20+ DAX measures including revenue, margins, and time intelligence aligned to Australian fiscal year (July-June). The model handles 50,000 transactions, validates to within 1% of source data, and delivers sub-second query performance."

Technical Deep Dive (2 minutes)

> "For FreshMarket's data model, I started with star schema design: one central FactSales table surrounded by five dimensions—Product, Store, Customer, Channel, and Date.
>
> The fact table contains only numeric measures (revenue, cost, quantity) and foreign keys. All descriptive attributes are in dimensions. This design enables fast queries because Power BI only needs one join per dimension instead of multiple joins through normalised tables.
>
> I created all relationships as one-to-many with single-direction cross-filtering. This prevents performance issues from bi-directional filters and ensures filter context flows correctly from dimensions to the fact.
>
> The most important dimension is the custom date table. I built it with DAX to support Australian financial year logic—July 1 to June 30. The FY column calculates as: if month is July or later, FY equals year plus one, otherwise it equals the current year. This enables accurate YTD, QTD, and year-over-year comparisons aligned to how Australian businesses actually report.
>
> I wrote 20 explicit DAX measures instead of using implicit aggregations. Key measures include Total Revenue, Gross Margin %, YTD Revenue using TOTALYTD with the 6/30 year-end parameter, YoY Growth % using SAMEPERIODLASTYEAR, and advanced analytics like product ranking with RANKX.
>
> For optimisation, I used measures instead of calculated columns wherever possible—measures compute on-the-fly and don't use model memory. I also disabled auto-date tables and validated all relationships.
>
> The final model is 42MB, queries run in under 1 second, and all measures reconcile to source data within 1% accuracy."

Common Interview Questions

1

"Explain the difference between a star schema and a normalised database."

Answer:

"A star schema is optimised for analytics, while normalisation is optimised for transactional systems.

In a normalised database, you might have Products, ProductCategories, and ProductSubcategories as separate tables to eliminate redundancy. To get revenue by category, you'd JOIN Sales → Products → ProductCategories—three joins.

In a star schema, we denormalise: the Product dimension includes Category and Subcategory columns directly. Now revenue by category requires only one join: FactSales → DimProduct. This is much faster for analytical queries.

The tradeoff is some data redundancy—if 100 products belong to 'Dairy', the word 'Dairy' is stored 100 times. But this redundancy is acceptable because storage is cheap and query performance matters more in analytics.

Star schemas also improve user experience. Business users can drag 'Product Category' directly into reports without understanding complex join logic."

2

"Why use single-direction relationships instead of bi-directional?"

Answer:

"Bi-directional cross-filtering allows filters to propagate in both directions between tables. While this seems convenient, it causes three major problems:

First, performance degradation. Bi-directional filters force Power BI to evaluate filter combinations in both directions, increasing query complexity exponentially with multiple relationships.

Second, ambiguous filter paths. If you have bi-directional relationships forming a loop (Dimension A ↔️ Fact ↔️ Dimension B ↔️ Dimension A), Power BI can't determine the correct filter propagation order, leading to incorrect results.

Third, unexpected filtering behaviour. With bi-directional filters, selecting a product could unexpectedly filter customers, which doesn't make logical sense.

In my FreshMarket model, I used single-direction relationships exclusively—filters flow from dimensions to the fact table. This creates a clear, predictable filtering hierarchy: users select products/stores/dates, and those selections filter sales transactions. The relationship direction matches the logical flow of how business users think about filtering data."

3

"How did you handle Australian financial year in your date table?"

Answer:

"Australian businesses operate on a July 1 to June 30 fiscal year, which is different from the calendar year. This means year-to-date calculations need to reset on July 1, not January 1.

I created a custom date table using DAX's CALENDAR function to generate all dates from 2022 to 2025. Then I added an FY column with this logic: if the month is July (7) or later, the financial year is the current year plus one. For example, July 1, 2024 belongs to FY2025. Otherwise, it's the current year. So January 15, 2024 belongs to FY2024.

I also added an IsEOFY flag that equals 1 only for June 30 dates, which is the End of Financial Year. This enables special EOFY reporting.

For time intelligence, I used TOTALYTD with the year-end parameter set to '6/30' instead of the default '12/31'. This tells Power BI to calculate year-to-date from July 1 to the current date.

The FY logic also cascades to quarters: Q1 is July-September, Q2 is October-December, Q3 is January-March, and Q4 is April-June. I created an FYQuarter column and FYMonth column (where July = Month 1) to support this.

I marked the table as a date table and created hierarchies so users can drill from FY → Quarter → Month → Date in visuals."

4

"When would you use a calculated column instead of a measure?"

Answer:

"I default to measures because they're computed on-the-fly and don't consume model memory. But there are specific scenarios where calculated columns are necessary:

First, when you need to filter or slice by the calculated value. For example, if I create an Age Group calculated column (18-24, 25-34, etc.) in the Customer dimension, users can add it to slicers and filter the entire report by age group. You can't filter by a measure.

Second, when the calculation is row-by-row and doesn't involve aggregation. For example, calculating 'Days Since Join' by subtracting the customer's join date from today's date requires a row-level calculation.

Third, when you need the value for a many-to-many relationship bridge table. Calculated columns can help resolve many-to-many scenarios.

However, I avoid calculated columns for aggregations. If I need total profit, I create a measure: [Total Revenue] - [Total Cost]. This is calculated from already-aggregated values, so it's fast. Creating a Profit calculated column that stores row-level profit for all 50,000 transactions wastes memory.

In my FreshMarket model, I used one calculated column—Age Group in the Customer dimension—because users needed to filter by it. Everything else is measures."

5

"How do you validate that your DAX measures are correct?"

Answer:

"I use a three-step validation process:

First, reconciliation with source data. I open the original CSV in Excel, sum the Revenue column, and compare to Power BI's Total Revenue measure. They should match within a small tolerance for any records removed due to quality issues. For FreshMarket, the difference was 1.2% because we removed duplicates and invalid data.

Second, manual calculation for time intelligence. For YTD Revenue in December 2023, I manually filter the source data to July 1 - December 31, 2023, sum the revenue in Excel, then compare to Power BI's YTD Revenue measure for December 2023. They must match exactly.

Third, logical consistency checks. I create a validation page with formulas like:

- Gross Profit must equal Total Revenue minus Total Cost
- Gross Margin % must equal Gross Profit divided by Total Revenue
- YoY % for August 2024 should equal (Aug 2024 Revenue - Aug 2023 Revenue) / Aug 2023 Revenue

I also use extreme filtering to test edge cases. What happens if I filter to a date with zero transactions? Do measures return zero or blank appropriately? Does DIVIDE handle zero denominators without errors?

Finally, I show the measures to business users with known results. For example, 'What was our Q3 FY2024 revenue?' If they know the answer from accounting systems, the Power BI measure must match.

This multi-layered validation catches 99% of DAX errors before reports go live."

6

"Describe your approach to model optimisation."

Answer:

"Model optimisation starts at the design phase, not as an afterthought.

First, I disable auto-date tables in Power BI options. These duplicate my custom date table and bloat the model with calendar-year versions of time intelligence.

Second, I use appropriate data types. Text uses the most memory, so I only use it for descriptive fields like Product Name. For IDs, I use shorter text keys. For flags like IsWeekend, I use 0/1 integers instead of True/False text. For amounts, I use Currency type which is optimised for financial data.

Third, I prefer measures over calculated columns. Calculated columns are stored in every row—if I have 50,000 rows and add a Profit column, that's 50,000 stored values. A Profit measure computes from aggregated Revenue and Cost, which is much smaller.

Fourth, I remove unused columns. If I load a table with 20 columns but only use 8, I hide or delete the other 12 to reduce model size.

Fifth, I use single-direction relationships exclusively. Bi-directional relationships slow down the query engine.

Sixth, I avoid many-to-many relationships which require expensive cross-joins.

For the FreshMarket model, these optimisations kept the size to 42MB and query time under 1 second. I use Performance Analyser in Power BI to identify slow visuals and optimise the DAX formulas driving them.

The goal is fast, responsive reports for executives who won't wait 10 seconds for a page to load."

Conclusion

You have successfully completed **Video 2: Data Modelling & DAX Fundamentals**!

What You Built

✅ **Star schema data model** (1 fact + 5 dimensions)

✅ **Proper relationships** (all 1:*, single direction)

✅ **Australian FY date table** (July 1 - June 30)

✅ **20+ DAX measures** (revenue, margins, time intelligence, budget variance, rankings)

✅ **Optimised performance** (42 MB model, <1 second queries)

✅ **Validated accuracy** (99.8% reconciliation with source)

Key Metrics Achieved

42 MB	0.7s	99.8%	100%
Model size	Query performance	Measure accuracy	Relationship integrity
Target: <50 MB ✅	Target: <1 second ✅	Target: ±1% ✅	No circular dependencies ✅

Next Steps

Video 3: End-to-End Dashboard Design

- Build executive summary page (KPI cards, trend lines)
- Build operational details page (tables, matrices, drill-through)
- Apply UI/UX best practices (layout, colours, accessibility)
- Implement interactivity (slicers, bookmarks, drill-through)
- Optimise visual performance (<3 second page load)
- Publish to Power BI Service with scheduled refresh

Save Your Work:

1. Click **File** → **Save**
2. Filename: FreshMarket_Model_Complete.pbix
3. Export DAX measures to text file (for portfolio)

Expected file size: ~42 MB

End of Video 2 Playbook

For questions or support:

- GitHub Issues: [repository link]
- LinkedIn: [your profile]
- Email: [your contact]