Power BI Modeling MCP Server Workshop

Deep Dive: AI-Powered Semantic Model Development

Duration: 2 Hours (Hands-On Lab Format)
Audience: Power BI Developers familiar with semantic models, new to MCP
Tools: VS Code + GitHub Copilot + Power BI Desktop
Sample Report: Sales & Returns Sample v201912.pbix

HOUR 1: Setup & Architecture (~60 minutes)

Title

Power BI Modeling MCP Server
Deep Dive Workshop

- 2 Hours | Hands-On Lab | VS Code + Copilot
- AI-Powered Semantic Model Development
- github.com/microsoft/powerbi-modeling-mcp

Workshop Agenda

Hour 1 — Setup & Architecture

- What is MCP? Protocol fundamentals
- Power BI Modeling MCP architecture
- Install VS Code extension
- Configure GitHub Copilot
- Clone sample report from Git
- Connect MCP to Power BI Desktop

Goal: Everyone connected and making their first MCP call

Hour 2 — Hands-On Lab

- **Lab 1:** Bulk rename columns & measures
- **Lab 2:** Add descriptions across model
- **Lab 3:** Create KPI Measures table with complex DAX
- **Lab 4:** Build Product Performance calculated table
- **Lab 5:** Add fiscal calendar columns & time intelligence
- **Lab 6:** Create Calculation Groups (Time & Currency)
- **Lab 7:** Implement Row-Level Security (static roles)
- **Lab 8:** Create Perspectives & generate documentation
- **Lab 9:** Git commit, push, and rollback

Goal: Complete real-world model changes with CI/CD safety net

---

What is MCP?

Model Context Protocol — The bridge between AI and your tools

MCP is an **open protocol** that standardizes how AI assistants connect to external tools, data sources, and services. Think of it as a **universal adapter** between LLMs and the real world.

Three Core Components:

Component	Description
TOOLS	Functions the AI can call to perform actions (create measure, modify table, execute DAX)
RESOURCES	Data and context the AI can read (model metadata, DAX patterns, documentation)

Component	Description
PROMPTS	Pre-built templates for common tasks (connect, query, analyze performance)

Key Insight

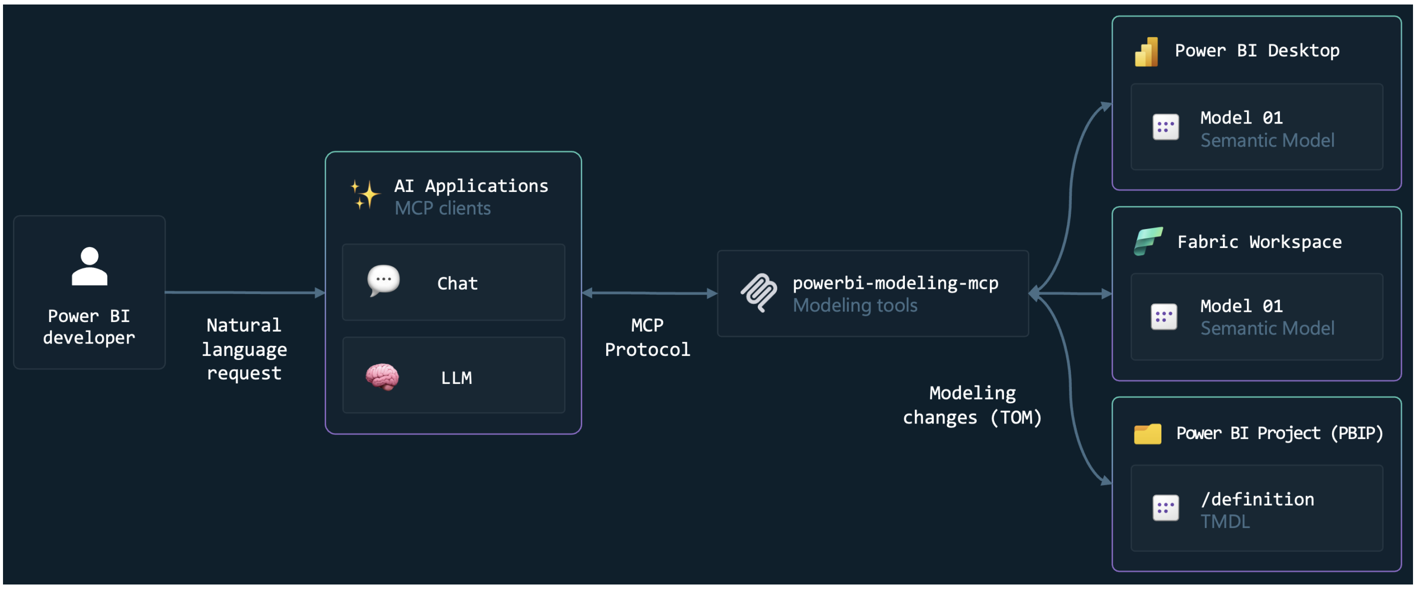
MCP servers run **locally** on your machine. Your data never leaves your environment — the AI sends commands, not your data.

Reference: modelcontextprotocol.io

Architecture Overview

Data Flow Diagram

```
 ┌─────────────────────────────────┐
│    YOU                          │
│    VS Code + GitHub Copilot     │
│    (Natural language prompts)   │
└───────────────┬─────────────────┘
                │
                │ MCP Protocol (stdio)
                ▼
┌─────────────────────────────────┐
│    LOCAL SERVER                 │
│    Power BI Modeling MCP        │
│    powerbi-modeling-mcp.exe     │
└───────────────┬─────────────────┘
                │
                │ TOM / AMO API
                ▼
┌───────────────────────────────────┐
│    SEMANTIC MODEL                 │
│    Power BI Desktop               │
│    localhost:port (auto-detected)│
└───────────────────────────────────┘
```



Key Points:

- ✓ Runs 100% local
- ✓ No data leaves your machine
- ✓ Uses existing PBI Desktop connection
- ✓ Auto-discovers running Desktop instances

Available MCP Tools (25+ Tools)

Core Operations

- `connection_operations` — Connect to PBI Desktop or Fabric
- `database_operations` — Manage semantic models, import/export TMDL
- `model_operations` — Get/create/update/refresh model
- `transaction_operations` — Begin, commit, rollback transactions

Model Structure

- `table_operations` — Create, update, delete, rename tables
- `column_operations` — Manage columns
- `relationship_operations` — Handle table relationships
- `partition_operations` — Manage table partitions

DAX & Calculations

- `measure_operations` — Create/update/delete measures
- `calculation_group_operations` — Calculation groups and items
- `function_operations` — DAX user-defined functions
- `dax_query_operations` — Execute, validate, generate DAX

Security & Access

- `security_role_operations` — Row-level security (RLS)
- `perspective_operations` — Filtered views for audiences
- `user_hierarchy_operations` — User-defined hierarchies

Metadata & i18n

- `culture_operations` — Multi-language support
- `object_translation_operations` — Translations for objects
- `named_expression_operations` — Power Query parameters
- `calendar_operations` — Time intelligence

Bulk Operations (Power Features!)

- `batch_table_operations` — Bulk table operations
- `batch_column_operations` — Bulk column operations
- `batch_measure_operations` — Bulk measure operations
- `batch_perspective_operations` — Bulk perspective management
- `batch_object_translation_operations` — Bulk translations

Power Features Highlight:

- Transaction support (commit/rollback)
- Bulk operations on 100s of objects simultaneously
- DAX query execution & validation
- TMDL import/export
- Deploy to Fabric workspace

Note: AI chooses the right tool based on your natural language request. No need to memorize tool names.

Requirements

Software

- **Power BI Desktop** (latest version)
- **Visual Studio Code**
- **Git** (for version control)

VS Code Extensions

- GitHub Copilot
- GitHub Copilot Chat
- **Power BI Modeling MCP** (the star of the show)

Accounts

- **GitHub account** with Copilot access
- **Microsoft Entra ID** (for authentication)

⚠ IMPORTANT

- Power BI Desktop must be **running** with a report open
- Use a **deep-reasoning model** (GPT-5 or Claude Sonnet 4.5) for best results
- We'll be working with **PBIP format** (not PBIX) for better Git integration

Today's Sample Report

Sales & Returns Sample v201912 (PBIP format)

Rich semantic model with:

- 18 tables
- 58 measures
- 9 relationships
- Time intelligence patterns

Repository: github.com/mdspinali/PowerBI-MCP-Workshop

Setup & Installation

HOUR 1

Setup & Installation

Get your environment ready for MCP development

~45 minutes

Step 1 of 5 — Install the MCP Extension

Instructions:

1. **Open VS Code**
    - Launch Visual Studio Code on your machine
2. **Go to Extensions** (Ctrl+Shift+X)
    - Or click the Extensions icon in the sidebar
3. **Search for "Power BI Modeling MCP"**
    - Published by Microsoft (analysis-services)
4. **Click Install**
    - Extension will download and activate automatically

Also Required:

GitHub Copilot + GitHub Copilot Chat extensions (if not already installed)

Direct Link:

aka.ms/powerbi-modeling-mcp-vscode

~2 minutes

Step 2 of 5 — Clone the Sample Report

Terminal Commands:

```
 # Clone the workshop repository with PBIP format
git clone https://github.com/mdspinali/PowerBI-MCP-Workshop.git

# Navigate to the workshop folder
cd PowerBI-MCP-Workshop

# Create a working branch for our changes
git checkout -b my-workshop-changes
```

What You'll Find:

Power BI Project (.pbip) — Text-based format optimized for version control

This sample includes:

- 18 tables (Sales, Product, Store, Calendar, etc.)
- 58 measures
- 9 relationships
- Time intelligence patterns

Why PBIP Instead of PBIX?

Format	Best For	Git Friendly?
.pbix	Snapshots, downloads, sharing single files	Binary — no line-by-line diffs
.pbip	Development, version control, team collaboration	Text-based — see every change

PBIP Benefits:

- ✓ See exactly what changed in each commit (measures, columns, relationships)
- ✓ Merge changes from multiple developers
- ✓ Review code in pull requests
- ✓ Automatic conflict detection

Today's Workshop: We'll use PBIP exclusively to take advantage of Git's full power.

Note:

With PBIP, Git shows line-by-line diffs of every model change — measures, tables, relationships, all in readable TMDL format!

~3 minutes

Step 3 of 5 — Open in Power BI Desktop

Steps:

1. **Open the .pbip file**
   - Navigate to the cloned repository folder
   - Double-click `Sales & Returns Sample v201912 (1).pbip`
   - Power BI Desktop will launch automatically
2. **Wait for the report to fully load**
   - All visuals should render completely
3. **Keep Desktop running**
   - Don't close it — MCP connects to the running instance

How it Works:

Power BI Desktop runs a local Analysis Services instance on a dynamic port. The MCP server auto-discovers this port by matching the filename you provide.

```
 Power BI Desktop (.pbip)
    └── localhost:xxxxx (dynamic port auto-detected)
```

~2 minutes

Step 4 of 5 — Configure Copilot for MCP

1. Open Copilot Chat

Press `Ctrl+Shift+I` or click the Copilot icon

2. Verify MCP Tools

- Click the **Tools** button in the chat
- Look for "powerbi-modeling-mcp" in the list
- Ensure it shows as **ENABLED** (green indicator)

3. Select AI Model

Use the model selector dropdown:

- GPT-5 (recommended)
- Claude Sonnet 4.5 (recommended)

Deep-reasoning models produce significantly better results for complex modeling tasks.

~3 minutes

Step 5 of 5 — Connect MCP to Desktop

The Command:

Type this in Copilot Chat:

```
Connect to 'Sales & Returns Sample v201912' in Power BI Desktop
```

What Happens:

1. MCP scans for running PBI Desktop instances
2. Matches filename to find correct instance
3. Establishes TOM API connection
4. **Ready for modeling commands!**

Confirmation Prompt

MCP will ask for confirmation before making any changes to your model:

> "Do you want to allow this operation on [model name]?"

This happens once per session (can be disabled with `--skipconfirmation` flag)

Setup Complete!

You're now connected to your semantic model via MCP.

~2 minutes

Checkpoint — Test Your Connection

Try These Prompts:

```
"List all tables in my model"
```

```
"Show me all measures"
```

```
"What relationships exist between tables?"
```

```
"Run this DAX: EVALUATE Sales"
```

Expected Output: Model Overview

```
 Tables: 7
• Sales (fact table)
• Returns (fact table)
• Product (dimension)
• Store (dimension)
• Calendar (dimension)
• Region (dimension)
• Measures Table

Measures: 15+
Total Revenue, Total Cost, Gross Profit, Return Rate...
```

If you see your model structure, you're ready!

Troubleshooting:

Not working? Check:

- PBI Desktop is running
- Report is open
- MCP extension enabled
- Correct filename in connect command

HOUR 2: Hands-On Lab (~60+ minutes)

Hands-On Lab Overview

HOUR 2

Hands-On Lab

Real-world semantic model changes with MCP

Lab	Topic	What You'll Create
LAB 1	Bulk Rename	Consistent naming across model
LAB 2	Descriptions	Auto-generated documentation
LAB 3	KPI Measures	7+ new measures with complex DAX
LAB 4	Product Performance	Calculated table with performance analysis
LAB 5	Fiscal Calendar	6 new columns + time intelligence
LAB 6	Calc Groups	Time Comparison + Currency groups
LAB 7	RLS	Static security roles by type/category
LAB 8	Perspectives	3 views + full documentation
LAB 9	Git Rollback	Version control safety net

~60+ minutes

Lab 1 — Bulk Rename for Consistency

Scenario:

Your model uses inconsistent naming: some columns are CamelCase, others have underscores, some measures lack prefixes. Apply a consistent naming convention across the entire model.

Prompts to Try:

```
"Analyze my model's naming conventions and suggest renames to ensure consistency"
```

```
"Rename all columns to use spaces instead of underscores"
```

```
"Add prefix 'Total' to all measures that sum values"
```

```
"Analyze the naming convention of the 'Sales' table and apply the same pattern across the entire model"
```

```
"Rename tables to follow PascalCase convention"
```

Before → After Example:

Before	After
product_category	Product Category
ProductName	Product Name
store_id	Store ID
Revenue	Total Revenue
totalCost	Total Cost

Git Checkpoint:

After completing the renames, save your .pbix and run:

```
git add . && git commit -m "Lab 1: Applied consistent naming"
```

~10 minutes
Tools Used: batch_column_operations, batch_measure_operations

Lab 2 — Auto-Generate Descriptions

Scenario:

Your model has zero documentation. Report consumers don't know what measures mean. Use AI to generate meaningful descriptions for all objects.

Prompts to Try:

```
"Add descriptions to all measures, explaining what each one calculates and when to use it"
```

```
"Add descriptions to all columns explaining their purpose and data type"
```

```
"Add descriptions to all tables explaining their role in the model (fact vs dimension)"
```

```
"For each DAX measure, explain the logic behind the code in simple, business-friendly terms"
```

Example Output:

Measure: Total Revenue

> *"Calculates the sum of all sales revenue. Use this measure for high-level revenue reporting across any dimension."*

Measure: Return Rate

> *"Calculates the percentage of units returned versus units sold. Values above 5% may indicate product quality issues."*

Column: Product[Category]

> *"The high-level product grouping (e.g., Electronics, Clothing). Use for top-level product analysis."*

Git Checkpoint:

```
git add . && git commit -m "Lab 2: Added descriptions to all objects"
```

~10 minutes
Tools Used: batch_measure_operations, batch_column_operations, table_operations

Lab 3 — Create a KPI Measures Table

Scenario:

Your model lacks organized KPIs. Create a dedicated measures table with a full suite of business metrics including complex DAX patterns.

Prompts to Try:

```
"Create a new table called 'KPI Measures' to hold all calculated measures"
```

```
"Create these measures in the KPI Measures table:
- Gross Margin % = DIVIDE([Net Sales] - [Returns], [Net Sales])
- Average Transaction Value = DIVIDE([Net Sales], DISTINCTCOUNT(Sales[ID]))
- Total Units per Transaction = DIVIDE([Units Sold], DISTINCTCOUNT(Sales[ID]))
- Days Since Last Sale = DATEDIFF(MAX(Sales[Date]), TODAY(), DAY)"
```

```
"Create a measure called 'Net Sales Trend' that shows MoM growth percentage"
```

```
"Create a measure 'Sales Velocity' that calculates revenue per day for the selected period"
```

Expected Measures Created:

Measure	DAX Pattern	Complexity
Gross Margin %	DIVIDE with subtraction	Basic
Average Transaction Value	DIVIDE + DISTINCTCOUNT	Intermediate
Total Units per Transaction	DIVIDE + DISTINCTCOUNT	Basic
Days Since Last Sale	DATEDIFF + MAX	Basic
Net Sales Trend (MoM%)	Time intelligence + DIVIDE	Advanced
Sales Velocity	DIVIDE + date range calc	Intermediate
Return Rate Trend	Time comparison pattern	Intermediate

Advanced Measure Example:

```
Net Sales Trend =
VAR CurrentMonth = [Net Sales]
VAR PriorMonth = CALCULATE([Net Sales], DATEADD('Calendar'[Date], -1, MONTH))
RETURN DIVIDE(CurrentMonth - PriorMonth, PriorMonth)
```

Git Checkpoint:

```
git add . && git commit -m "Lab 3: Created KPI Measures table with 7 new measures"
```

~15 minutes
Tools Used: table_operations, measure_operations

Lab 4 — Build a Customer Segmentation Table

Scenario:

Marketing wants to analyze customers by segment. Create a calculated table that segments customers based on purchase behavior (RFM analysis).

Prompts to Try:

```
"Create a calculated table called 'Product Performance' with columns: ProductID, Product, Category, TotalSales, TotalReturns, Retur
```

```
"Add a column to Product Performance that categorizes products as 'High Performer', 'Average', or 'Underperformer' based on TotalSal
```

```
"Add a calculated column 'ProfitabilityRank' that ranks products by (TotalSales - TotalReturns)"
```

```
"Create a relationship between Product Performance and the Sales table on ProductID"
```

Expected Table Structure:

```
Customer Segments (Calculated Table)
├── CustomerID (key)
├── TotalSpend (currency)
├── OrderCount (integer)
├── LastOrderDate (date)
├── DaysSinceLastOrder (integer)
├── ValueSegment (text: High/Medium/Low)
├── EngagementLevel (text: Active/At Risk/Churned)
└── RFMScore (integer: 1-5)
```

DAX for Calculated Table:

```
Product Performance =
ADDCOLUMNS(
    SUMMARIZE(Sales, Sales[ProductID], Product[Product], Product[Category]),
    "TotalSales", CALCULATE(SUM(Sales[Amount])),
    "TotalReturns", CALCULATE([Returns]),
    "UnitsSold", CALCULATE(SUM(Sales[Unit])),
    "ReturnRate", DIVIDE(CALCULATE([Units Returned]), CALCULATE(SUM(Sales[Unit]))),
    "PerformanceSegment",
        SWITCH(TRUE(),
            DIVIDE(CALCULATE([Units Returned]), CALCULATE(SUM(Sales[Unit]))) > 0.1, "Underperformer",
            CALCULATE(SUM(Sales[Amount])) > 50000, "High Performer",
            "Average"
        )
)
```

Create Supporting Measures:

```
"Create measures: High Performer Count, Underperformer Count, Average Sales by Performance Segment"
```

Git Checkpoint:

```
git add . && git commit -m "Lab 4: Built Customer Segmentation calculated table with relationships"
```

~15 minutes
Tools Used: table_operations, column_operations, relationship_operations, measure_operations

Lab 5 — Add Date Intelligence & Fiscal Calendar

Scenario:

The model has a basic calendar but lacks fiscal year support and advanced date intelligence. Extend the calendar and create time-based measures.

Prompts to Try:

```
"Add these columns to the Calendar table: FiscalYear (starting July), FiscalQuarter, FiscalMonth, WeekOfYear, IsWeekend, IsHoliday"
```

```
"Create a calculated column 'FiscalYearQuarter' that shows 'FY24-Q1' format"
```

```
"Create these time intelligence measures:
- Revenue YTD (fiscal year)
- Revenue QTD (fiscal quarter)
- Revenue Same Period Last Fiscal Year
- Fiscal YoY Growth %"
```

```
"Create a measure 'Rolling 12 Month Revenue' using DATESINPERIOD"
```

```
"Create a measure 'Revenue Moving Annual Total' (MAT)"
```

New Calendar Columns:

Column	Example	DAX
FiscalYear	FY2024	Based on July start
FiscalQuarter	Q1	July=Q1, Oct=Q2, Jan=Q3, Apr=Q4

Column	Example	DAX
FiscalMonth	1-12	July=1 through June=12
WeekOfYear	1-52	WEEKNUM()
IsWeekend	TRUE/FALSE	WEEKDAY check
FiscalYearQuarter	FY24-Q1	Concatenated

Time Intelligence Measures:

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

Revenue Same Period LFY =
CALCULATE(
    [Total Revenue],
    DATEADD('Calendar'[Date], -1, YEAR)
)

Rolling 12M Revenue =
CALCULATE(
    [Total Revenue],
    DATESINPERIOD('Calendar'[Date], MAX('Calendar'[Date]), -12, MONTH)
)
```

Git Checkpoint:

```
git add . && git commit -m "Lab 5: Extended calendar with fiscal year and time intelligence"
```

~15 minutes
Tools Used: column_operations, measure_operations, calendar_operations

Lab 6 — Create Calculation Groups

Scenario:

Instead of duplicating time intelligence patterns for every measure, create a Calculation Group that applies patterns dynamically to ANY measure.

Prompts to Try:

```
 "Create a calculation group called 'Time Comparison' with these calculation items:
- Actual (current value)
- Prior Year
- Year over Year Change
- Year over Year %
- Prior Period (month)
- Period over Period %"
```

```
 "Add a calculation item 'YTD' to the Time Comparison group"
```

```
 "Add a calculation item 'Rolling 3 Month Average'"
```

```
 "Create a second calculation group called 'Currency' with items for USD, EUR (rate 0.92), GBP (rate 0.79)"
```

Expected Structure:

```
Time Comparison (Calculation Group)
  ├── Actual           → SELECTEDMEASURE()
  ├── Prior Year       → CALCULATE(SELECTEDMEASURE(), SAMEPERIODLASTYEAR('Calendar'[Date]))
  ├── YoY Change       → SELECTEDMEASURE() - CALCULATE(SELECTEDMEASURE(), SAMEPERIODLASTYEAR(...))
  ├── YoY %            → DIVIDE([YoY Change], [Prior Year])
  ├── Prior Period     → CALCULATE(SELECTEDMEASURE(), PREVIOUSMONTH('Calendar'[Date]))
  ├── PoP %            → DIVIDE(SELECTEDMEASURE() - [Prior Period], [Prior Period])
  ├── YTD              → TOTALYTD(SELECTEDMEASURE(), 'Calendar'[Date])
  └── Rolling 3M Avg   → AVERAGEX(DATESINPERIOD(...), SELECTEDMEASURE())

Currency (Calculation Group)
  ├── USD              → SELECTEDMEASURE()
  ├── EUR              → SELECTEDMEASURE() * 0.92
  └── GBP              → SELECTEDMEASURE() * 0.79
```

Why This is Powerful:

- **Before:** 15 base measures × 8 time patterns = 120 measures to maintain
- **After:** 15 base measures + 1 calculation group = 16 objects to maintain
- Users select time comparison from slicer, applies to ALL measures instantly

Git Checkpoint:

```
git add . && git commit -m "Lab 6: Created Time Comparison and Currency calculation groups"
```

~15 minutes
Tools Used: calculation_group_operations

Lab 7 — Implement Row-Level Security

Scenario:

The organization needs to restrict data. Regional managers see only their region. Sales reps see only their accounts. Implement dynamic RLS.

Prompts to Try:

```
"Create a security role called 'Regional Manager' that filters the Store table where Region = 'North America'"
```

```
"Create security roles for each distinct region in the model"
```

```
"Create a dynamic RLS role called 'Sales Rep' that filters Sales where SalesRepEmail = USERPRINCIPALNAME()"
```

```
"Create a role 'Executive' with no row filters (full access)"
```

```
"List all security roles and their filter expressions"
```

RLS Architecture:

```
Security Roles
   │
   ├── Regional Manager - North America
   │   └── Store[Region] = "North America"
   │
   ├── Regional Manager - Europe
   │   └── Store[Region] = "Europe"
   │
   ├── Regional Manager - Asia Pacific
   │   └── Store[Region] = "Asia Pacific"
   │
   ├── Sales Rep (Dynamic)
   │   └── Sales[SalesRepEmail] = USERPRINCIPALNAME()
   │
   └── Executive
       └── (No filters)
```

Dynamic RLS Pattern:

```
// Filter expression for Sales Rep role on Sales table:
[SalesRepEmail] = USERPRINCIPALNAME()

// Or with a lookup table:
CONTAINS(
    FILTER(UserMapping, UserMapping[Email] = USERPRINCIPALNAME()),
    UserMapping[Region], Store[Region]
)
```

Test the Roles:

After creating, test in Power BI Desktop:

- Modeling → View as Roles → Select role to test

Git Checkpoint:

```
git add . && git commit -m "Lab 7: Implemented static and dynamic RLS"
```

~10 minutes
Tools Used: security_role_operations

Lab 8 — Create Perspectives & Generate Documentation

Scenario:

Different users need different views. Create perspectives for Sales, Finance, and Executive teams. Then generate complete model documentation.

Create Perspectives:

```
"Create a perspective called 'Sales View' that includes: Sales table, Product table, Store table, Product Performance table, and all
```

```
"Create a perspective called 'Finance View' that includes: Sales table, Calendar table, and all return/margin measures"
```

```
"Create a perspective called 'Executive Summary' that includes only the KPI Measures table and dimension tables"
```

Expected Perspectives:

Perspective	Tables	Measures

Perspective	Tables	Measures
Sales View	Sales, Product, Store, Product Performance	Net Sales, Units Sold, Transaction metrics
Finance View	Sales, Calendar	Returns, Margin, Trends
Executive Summary	All dimensions, KPI Measures	All KPIs

Generate Documentation:

```
"Generate a complete Markdown document documenting this semantic model including:
- Model overview and statistics
- All tables with columns and data types
- All measures with full DAX code
- Relationships diagram in Mermaid format
- Security roles and their filters
- Perspectives and their contents
- Calculation groups and items"
```

Example Documentation Output:

```
# Sales & Returns Semantic Model

## Model Statistics
- Tables: 9 (7 original + 2 created)
- Measures: 25+
- Relationships: 12
- Security Roles: 5
- Calculation Groups: 2

## New Objects Created in Workshop
- KPI Measures table (7 measures)
- Customer Segments calculated table
- Calendar fiscal columns (6 columns)
- Time Comparison calc group (8 items)
- Currency calc group (3 items)
...
```

Git Checkpoint:

```
git add . && git commit -m "Lab 8: Created perspectives and documentation"
```

~10 minutes
Tools Used: perspective_operations, batch_perspective_operations, model_operations

Lab 9 — Git Workflow & Rollback

Scenario:

The AI made a mistake! A bulk rename broke some DAX references. Use Git to review changes and rollback if needed.

View Changes Made:

```
# See all commits from workshop
git log --oneline

# Output:
# a1b2c3d Lab 5: Generated model documentation
# e4f5g6h Lab 4: Implemented regional RLS
# i7j8k9l Lab 3: Created Time Intelligence calculation group
# m0n1o2p Lab 2: Added descriptions to all objects
# q3r4s5t Lab 1: Applied consistent naming
# u6v7w8x Initial commit
```

Rollback Options:

Option 1: Undo last commit (keep changes staged)

```
git reset --soft HEAD~1
```

Option 2: Undo last commit (discard changes)

```
git reset --hard HEAD~1
```

Option 3: Rollback to specific commit

```
git checkout q3r4s5t -- "Sales & Returns Sample v201912.pbix"
```

Option 4: Create a new commit that undoes changes

```
git revert HEAD
```

Best Practice Workflow:

```
1. Make MCP changes
2. Save .pbix in Power BI Desktop (Ctrl+S)
3. Test changes work correctly
4. git add . && git commit -m "Description"
5. If something breaks → git reset or git revert
```

Pro Tip:

Commit frequently! Small commits = easier rollbacks.

~5 minutes

Advanced Scenarios (Bonus)

If Time Permits — Try These:

Model Translation (i18n)

```
"Generate a French translation for my model including tables, columns, and measures"
```

DAX Query Benchmarking

```
"Execute this DAX query and return the performance metrics: EVALUATE SUMMARIZE(Sales, Product[Category], 'Total Revenue')"
```

Refactor to Parameters

```
"Analyze the Power Query code for all tables, identify the data source configuration, and create semantic model parameters to enable
```

Bulk Add to Perspectives

```
"Create a perspective called 'Sales Analysis' that includes only the Sales table, Product table, and all revenue-related measures"
```

Model Statistics

```
"Give me statistics about my model: row counts, column counts, measure complexity"
```

MCP Settings & Options

Command Line Flags:

Flag	Default	Description
`--start`	—	Starts the MCP server (required)
`--readwrite`	Yes	Enables write operations with confirmation
`--readonly`	—	Safe mode, prevents any writes
`--skipconfirmation`	—	Auto-approves all operations (⚠ use carefully)
`--compatibility`	PowerBI	Set to `Full` for Analysis Services

Configuring in VS Code:

1. Open VS Code Settings (Ctrl+,)
2. Search for `@ext:Microsoft.powerbi-modeling-mcp`
3. Modify the `args` setting

Example JSON Config:

```
{
  "servers": {
    "powerbi-modeling-mcp": {
      "command": "powerbi-modeling-mcp.exe",
      "args": ["--start", "--skipconfirmation"],
      "type": "stdio"
    }
  }
}
```

Troubleshooting Common Issues

"Cannot connect to Power BI Desktop"

- Ensure PBI Desktop is running
- A report must be open (not just Desktop)
- Check exact filename spelling in connect command
- Only one instance of Desktop per report

"MCP tools not showing in Copilot"

- Restart VS Code after installing extension
- Check extension is enabled (not just installed)
- Click Tools button and verify "powerbi-modeling-mcp" listed

"Operation timed out"

- Large models may take longer
- Try smaller batch sizes

- Check Desktop isn't frozen

"Changes not appearing in Desktop"

- Changes apply immediately to the running model
- Refresh the model view in Desktop
- Check the specific table/measure

"Authentication error"

- Sign into VS Code with Microsoft account
- Entra ID must match your organization
- Check tenant allows MCP connections

Resources:

- Troubleshooting guide: github.com/microsoft/powerbi-modeling-mcp/TROUBLESHOOTING.md
- Open issues: github.com/microsoft/powerbi-modeling-mcp/issues

---

Security Considerations

Important Warnings from Microsoft:

1. **Use caution** when connecting an AI Agent to a semantic model. The underlying LLM may produce unexpected or inaccurate results.

2. **Always create a backup** of your model before performing any operations.

3. **LLMs might unintentionally expose sensitive information** from the semantic model in logs or responses. Exercise caution when sharing chat sessions.

4. The MCP server can only execute **modeling operations**. It cannot modify reports or diagram layouts.

Security Model:

- Credentials handled through official **Azure Identity SDK**
- Tokens are never stored or managed directly by MCP
- MCP clients invoke operations based on **user's Fabric RBAC permissions**

Best Practices:

- Enable Entra ID authentication
- Apply least-privilege RBAC roles
- Review changes before committing
- Use `--readonly` flag for exploration
- Don't share chat sessions containing sensitive data

What's Next?

After This Workshop:

1. **Practice on your own models**

    - Start with non-production models
    - Build confidence with read-only operations first

2. **Explore PBIP integration**

    - MCP supports Power BI Project files (TMDL)
    - Better Git diffs with text-based format

3. **Connect to Fabric workspaces**

    - Same MCP, different connection command
    - `Connect to semantic model '[Name]' in Fabric Workspace '[Workspace]'`

4. **Build custom workflows**

    - Combine MCP with other tools
    - Automate repetitive modeling tasks

Resources:

- **GitHub:** github.com/microsoft/powerbi-modeling-mcp
- **Demo Video:** aka.ms/power-modeling-mcp-demo

- **MCP Protocol:** modelcontextprotocol.io

Workshop Recap

What You Learned:

- **MCP Protocol** — How AI connects to tools via Tools, Resources, and Prompts
- **Architecture** — VS Code → MCP Server → Power BI Desktop (all local)
- **25+ Tools** — From single operations to bulk batch processing
- **Setup** — Extension install, Git clone, Copilot config, connection

What You Built:

- **KPI Measures Table** — 7 new measures with complex DAX patterns
- **Customer Segments** — Calculated table with RFM analysis
- **Fiscal Calendar** — 6 new columns + time intelligence measures
- **Calculation Groups** — Time Comparison (8 items) + Currency (3 items)
- **Row-Level Security** — Static regional + dynamic user-based roles
- **Perspectives** — Sales, Finance, Executive views
- **Documentation** — Auto-generated model documentation
- **Safety Net** — Git commits for easy rollback when AI makes mistakes

Key Takeaway:

> MCP transforms hours of repetitive modeling work into seconds of natural language conversation — with version control as your safety net.

Final — Go Build!

Now Go Accelerate Your Workflow

Remember:

- Backup before experimenting
- Commit often
- Start with read-only exploration
- Use deep-reasoning models for best results

Share with your team: github.com/microsoft/powerbi-modeling-mcp

Questions? [Your contact info / Q&A time]

Appendix: Quick Reference

Connection Commands

```
# Connect to Power BI Desktop
Connect to '[filename]' in Power BI Desktop

# Connect to Fabric
Connect to semantic model '[name]' in Fabric Workspace '[workspace]'

# Connect to PBIP
Open semantic model from PBIP folder '[path to definition/ TMDL folder]'
```

Common Prompt Patterns

```
# Exploration
"List all [tables/measures/columns/relationships]"
"Show me the DAX for measure [name]"
"What is the structure of table [name]"

# Modifications
"Rename [object] from [old] to [new]"
"Add description to [object]: [description]"
"Create measure [name] with DAX: [expression]"

# Bulk Operations
"Rename all [objects] to use [pattern]"
"Add descriptions to all [objects]"
"Delete all [objects] that match [criteria]"

# Analysis
"Analyze naming conventions"
"Find unused measures"
"Check for circular dependencies"
```

Git Commands Cheat Sheet

```
# Save progress
git add .
git commit -m "Description of changes"

# View history
git log --oneline

# Undo last commit (keep changes)
git reset --soft HEAD~1

# Undo last commit (discard changes)
git reset --hard HEAD~1

# Rollback specific file
git checkout [commit] -- [filename]

# Create undo commit
git revert HEAD
```