Microsoft

Microsoft Power BI Data Analyst

Exam Ref PL-300

Daniil Maslyuk

FREE SAMPLE CHAPTER

Microsoft

Exam Ref PL-300 Microsoft Power BI Data Analyst

Daniil Maslyuk

Exam Ref PL-300 Microsoft Power BI Data Analyst

Published with the authorization of Microsoft Corporation by:
Pearson Education, Inc.

ISBN-13: 978-0-13-790123-4
ISBN-10: 0-13-790123-2

Library of Congress Control Number: 2022938757

ScoutAutomatedPrintCode

EDITOR-IN-CHIEF
Brett Bartow

EXECUTIVE EDITOR
Loretta Yates

SPONSORING EDITOR
Charvi Arora

DEVELOPMENT EDITOR
Songlin Qiu

MANAGING EDITOR
Sandra Schroeder

SENIOR PROJECT EDITOR
Tracey Croom

COPY EDITOR
Elizabeth Welch

INDEXER
Cheryl Lenser

PROOFREADER
Barbara Mack

TECHNICAL EDITOR
Owen Auger

EDITORIAL ASSISTANT
Cindy Teeters

COVER DESIGNER
Twist Creative, Seattle

COMPOSITOR
codeMantra

To Dasha, Leonard, and William, who served as a great source of motivation and support.

—Daniil Maslyuk

Contents at a glance

This page intentionally left blank

Contents

Introduction

Exam PL-300 focuses on using Microsoft Power BI for data analysis. About one-fourth of the exam covers data preparation, which includes getting data from different data sources, as well as cleaning, transforming, and loading the data. Approximately one-third of the questions are related to data modeling: designing, developing, and optimizing a data model. Another one-third of the book covers the skills necessary to visualize and analyze data, such as creating reports and dashboards, as well as performing advanced analysis. The remainder of the book discusses how to manage datasets and workspaces in the Power BI service.

This exam is intended for data analysts, business intelligence professionals, and report creators who are seeking to validate their skills and knowledge in analyzing data with Power BI. Candidates should be familiar with how to get, model, and visualize data in Power BI Desktop, as well as share reports with other people.

This book covers every major topic area found on the exam, but it does not cover every exam question. Only the Microsoft exam team has access to the exam questions, and Microsoft regularly adds new questions to the exam, making it impossible to cover specific questions. You should consider this book a supplement to your relevant real-world experience and other study materials. If you encounter a topic in this book that you do not feel completely comfortable with, use the "Need more review?" links you'll find in the text to find more information and take the time to research and study the topic. Great information is available on https://docs.microsoft.com

Organization of this book

This book is organized by the "Skills measured" list published for the exam. The "Skills measured" list is available for each exam on the Microsoft Learn website: http://microsoft.com learn. Each chapter in this book corresponds to a major topic area in the list, and the technical tasks in each topic area determine a chapter's organization. If an exam covers six major topic areas, for example, the book will contain six chapters.

Preparing for the exam

Microsoft certification exams are a great way to build your résumé and let the world know about your level of expertise. Certification exams validate your on-the-job experience and product knowledge. Although there is no substitute for on-the-job experience, preparation through study and hands-on practice can help you prepare for the exam. This book is *not* designed to teach you new skills.

We recommend that you augment your exam preparation plan by using a combination of available study materials and courses. For example, you might use the Exam Ref and another study guide for your ”at home” preparation and take a Microsoft Official Curriculum course for the classroom experience. Choose the combination that you think works best for you. Learn more about available classroom training and find free online courses and live events at *http://microsoft.com/learn*. Microsoft Official Practice Tests are available for many exams at *http://aka.ms/practicetests*.

Note that this Exam Ref is based on publicly available information about the exam and the author’s experience. To safeguard the integrity of the exam, authors do not have access to the live exam.

Microsoft certifications

Microsoft certifications distinguish you by proving your command of a broad set of skills and experience with current Microsoft products and technologies. The exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop, or implement and support, solutions with Microsoft products and technologies both on-premises and in the cloud. Certification brings a variety of benefits to the individual and to employers and organizations.

MORE INFO **ALL MICROSOFT CERTIFICATIONS**

For information about Microsoft certifications, including a full list of available certifications, go to http://www.microsoft.com/learn.

Check back often to see what is new!

Quick access to online references

Throughout this book are addresses to webpages that the author has recommended you visit for more information. Some of these links can be very long and painstaking to type, so we’ve shortened them for you to make them easier to visit. We’ve also compiled them into a single list that readers of the print edition can refer to while they read.

Download the list at *MicrosoftPressStore.com/ExamRefPL300/downloads*

The URLs are organized by chapter and heading. Every time you come across a URL in the book, find the hyperlink in the list to go directly to the webpage.

Errata, updates, & book support

We’ve made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

MicrosoftPressStore.com/ExamRefPL300/errata

If you discover an error that is not already listed, please submit it to us at the same page.

For additional book support and information, please visit *MicrosoftPressStore.com/Support*.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *http://support.microsoft.com*.

Stay in touch

Let’s keep the conversation going! We’re on Twitter: *http://twitter.com/MicrosoftPress*.

Acknowledgments

I would like to thank Loretta Yates for trusting me to write the official Power BI exam reference book, Charvi Arora for handling the project, the editing team for making this book a better read, and everyone else at Pearson who worked on this book to make it happen.

A few people have contributed to my becoming a fan of Power BI. Gabriel Polo Reyes was instrumental in my introduction to the world of Microsoft BI. Thomas van Vliet, my first client, hired me despite my having no prior commercial experience with Power BI and fed me many problems that led to my mastering Power BI.

About the author

DANIIL MASLYUK is an independent business intelligence consultant, trainer, and speaker who specializes in Microsoft Power BI. Daniil blogs at xxlbi.com and tweets as @DMaslyuk.

This page intentionally left blank

CHAPTER 2

Model the data

In the previous chapter, we reviewed the skills necessary to get and transform data by using Power Query Editor—the process also known as *data shaping*. In this chapter, we examine the skills needed to model data.

Although Power BI allows you to analyze your data to some degree right after you load it, a strong understanding of data modeling allows you to perform sophisticated analysis using rich data modeling capabilities, which includes creating relationships, hierarchies, and various calculations to bring out the true power of Power BI. Previously in the Power Query Editor we used the M language; after we load the data into the model, we use data analysis expressions, more commonly referred to as DAX—Power BI's native query language.

In this chapter, we review the skills necessary to design, develop, and optimize data models. Additionally, we look at DAX and how it can be used to enhance data models.

Skills covered in this chapter:

- Skill 2.1: Design a data model
- Skill 2.2: Develop a data model
- Skill 2.3: Create model calculations by using DAX
- Skill 2.4: Optimize model performance

Skill 2.1: Design a data model

A proper data model is the foundation of meaningful analysis. A Power BI data model is a collection of one or more tables and, optionally, relationships. A well-designed data model enables business users to understand and explore their data and derive insights from it. This step should be taken before you create any visuals by loading your data and defining the relationships between tables. Data modeling often occurs at the beginning phase of building a Power BI report so that you can create efficient measures that build upon your data model. In this section, we design a data model by focusing our attention on tables and their relationships.

This skill covers how to:

- Define the tables and design a data model that uses a star schema
- Configure table and column properties

- Design and implement role-playing dimensions
- Define a relationship's cardinality and cross-filter direction
- Create a common date table

Define the tables and design a data model that uses a star schema

Once a query is loaded, it becomes a table in a Power BI data model. Tables can then be organized into different data model types, also known as *schemas*. The three most common schemas in Power BI are:

- Flat (fully denormalized) schema
- Star schema
- Snowflake schema

There are other types of data models, though these three are the most common ones.

Flat schema

In the flat type of data model, all attributes are fully denormalized into a single table. Because there's only one table, there are no relationships, and in most cases there's no need for key.

In our Wide World Importers example, we have a single table that contains all columns from all tables, meaning that the Sale and Targets columns will be in the same table. Because the tables have different data granularity, you run into problems when comparing actuals and targets.

NOTE **DATA GRANULARITY**

We review the concept of data granularity later in this skill section.

From the performance point of view, flat schemas are very efficient, though there are downsides:

- A single table can be cumbersome and confusing to navigate.
- Columns and data can often be duplicated, leading to a comparatively large file size.
- Mixing facts of different grains results in more complex DAX formulas.

Flat schemas are often used when connecting to a single, simple source. However, for more complex data models, flat schemas should be avoided in Power BI as much as possible.

Star schema

When you use a star schema, tables are conceptually classified into two kinds:

- **Fact tables** These tables contain the metrics you want to aggregate. Fact tables have foreign keys, which are required in order to create relationships with dimensions, and columns that you can aggregate. In our Wide World Importers example, the Sale and Targets tables are fact tables. Fact tables are sometimes also known as *data tables*.

- **Dimension tables** These tables contain the descriptive attributes that help you slice and dice your fact tables. A dimension table has a unique identifier—a key column—and descriptive columns. In our Wide World Importers example, the City, Customer, Date, Employee, and Stock Item are dimension tables. Dimension tables are also sometimes known as *lookup tables*.

In a star schema, fact tables are surrounded by dimensions, as shown in Figure 2-1.

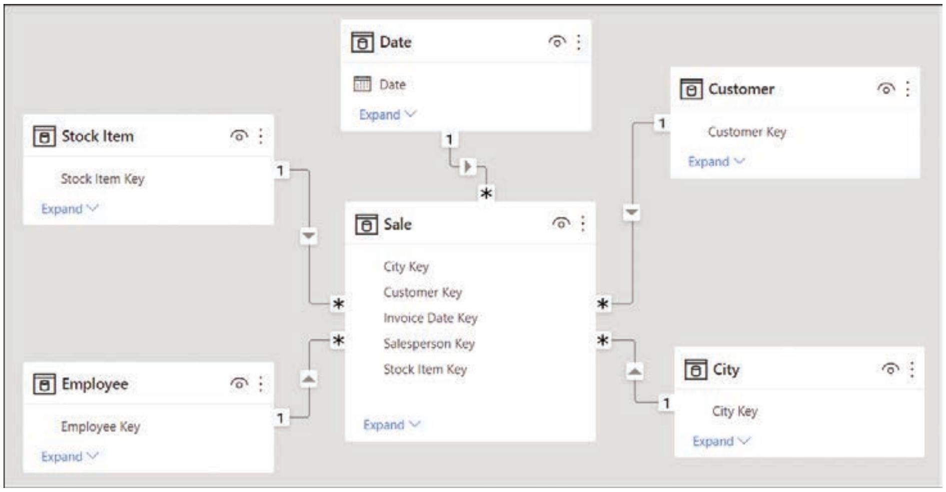


FIGURE 2-1 Star schema with Sale as the only fact table

The star schema has its name because it resembles a star, with the fact table in the center and dimension tables as the star points. It's possible to have more than one fact table in a star schema, and it will still be a star schema.

NOTE **RELATIONSHIPS**

The lines that connect tables in Figure 2-1 represent relationships. We cover the relationships in more detail later in this section.

In most cases, the star schema is the preferred data modeling approach in Power BI. It addresses the shortcomings of the flat schema:

- Fields are logically grouped, making the model easier to understand.
- There is less duplication of data, which results in more efficient storage.
- You don't need to write overly complex DAX formulas to work with fact tables that have a different grain.

Snowflake schema

The snowflake schema is similar to the star schema, except it can have some dimensions that "snowflake" from other dimensions. You can see an example in Figure 2-2.

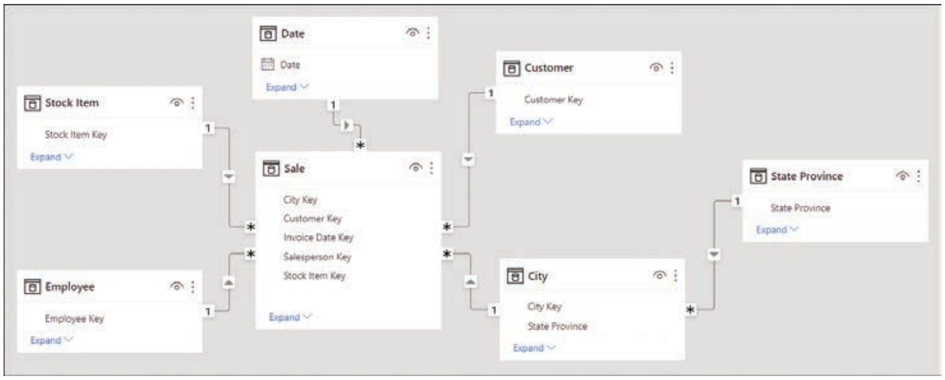


FIGURE 2-2 Snowflake schema with State Province snowflaking from the City table

In the Wide World Importers example, if we loaded the State Province query, the data model could be a snowflake schema. This is because the State Province table is related to the City dimension table, which in turn is related to the Sale fact table.

Snowflake schemas can be beneficial when there are fact tables that have different grains.

NEED MORE REVIEW? **DIMENSIONAL MODELING**

In addition to fact and dimension tables, there are other types of tables such as factless facts, junk, and degenerate dimensions. For more information, see "Understand star schema and the importance for Power BI" at *https://docs.microsoft.com/en-us/power-bi/guidance/star-schema*.

Configure table and column properties

Both tables and columns have various properties you can configure, and you can do it in the **Model** view. To see the properties of a column or a table, select an object, and you will see its properties in the **Properties** pane.

Table properties

For tables, depending on the storage mode, you can configure the following properties:

- **Name** Enter the table name.
- **Description** This property allows you to add a description of the table that will be stored in the model's metadata. It can be useful when building reports because you can see the description when you hover over the table in the Fields pane.
- **Synonyms** These are useful for the Q&A feature of Power BI, which we review in the next skill section. You can add synonyms so that the Q&A feature can understand that you're referring to a specific table even if you provide a different name for it.

- **Row label** This property is useful for both Q&A and featured tables, and it allows you to select a column whose values will serve as labels for each row. For example, if you ask Q&A to show “sales amount by product” and you select the Product Name column as the Row label of the Product table, then Q&A will show sales amount for each product name.
- **Key column** If your table has a column that has unique values for every row, you can set that column as the key column.
- **Is hidden** You can hide a table so that it disappears from the Fields pane.
- **Is featured table** This property allows you to make a table featured, which will allow it to be used in Excel in certain scenarios.
- **Storage mode** This property may be set to Import, DirectQuery, or Dual, as we covered in the previous chapter.

Column properties

For columns, depending on data type, you can configure the following properties:

- **Name** Enter the column name.
- **Description** As you can for tables, you can add a column description.
- **Synonyms** As you can for tables, you can add synonyms to make the column work better with Q&A.
- **Display folder** You can group columns from the same table into display folders.
- **Is hidden** Hiding a column keeps it in the data model and hides it in the Fields pane.
- **Data type** The available data types are different from those available in Power Query. For instance, Percentage, Date/Time/Timezone, and Duration are not available.
- **Format** Different data types will show different formatting properties. For example, for numeric columns, you'll see the following additional properties: Percentage format, Thousands separator, and Decimal places.
- **Sort by column** You can sort one column by another. For example, you can sort month names by month numbers to make them appear in the correct order.
- **Data category** This property can be useful for some visuals, and the default is Uncategorized. Depending on the data type, you can also select one of the following:
  - Address
  - Place
  - City
  - County
  - State or Province
  - Postal Code
  - Country
  - Continent
  - Latitude

    - Longitude
    - Web URL
    - Image URL
    - Barcode
- **Summarize by** This property determines how the column will be aggregated if you put it into a visual. The options you can choose depend on the data type. For most data types, in addition to Don't Summarize/None, you can choose Count and Count (Distinct)/Distinct Count, whereas for numeric columns, you can also choose Sum, Average, Minimum/Min, and Maximum/Max. While Power BI will try to automatically determine the appropriate summarization, it's not always accurate.
- **Is nullable**—You may disallow null values for a column; if during data refresh, a column is determined to get a null value, the refresh will fail.

EXAM TIP

You should know the difference between formatting a column and using the `FORMAT` function in DAX: the former retains the original data type, whereas the latter can be used to create a new column and always outputs text.

NOTE **MEASURE PROPERTIES**

You can also configure measure properties, many of which are the same as column properties. Notable exceptions include Sort by column, Summarize by, and Is nullable—these properties aren't available for measures. We review measures in more detail later in this chapter.

Design and implement role-playing dimensions

In some cases, there may be more than one way to filter a fact table by a dimension. In the Wide World Importers example, the Sale table has two date columns: Invoice Date Key and Delivery Date Key, both of which can be related to the Date column from the Date table. Therefore, it's possible to analyze sales by invoice date or delivery date, depending on the business requirements. In this situation, the Date dimension is a role-playing dimension.

NOTE **COMPANION FILE**

If you're interested in following along the examples in this chapter, you can start with the 2.0 Model.pbix file in the companion files folder. The completed examples are available in 2.1 Design.pbix.

While Power BI allows you to have multiple physical relationships between two tables, no more than one can be active at a time, and other relationships must be set as inactive. Active relationships, by default, propagate filters. The choice of which relationship should be set as active depends on the default way of looking at data by the business.

NEED MORE REVIEW? **ACTIVE AND INACTIVE RELATIONSHIPS**

For a more thorough explanation of when you would use active or inactive relationships, see "Active vs inactive relationship guidance" at *https://docs.microsoft.com/en-us/power-bi/guidance/relationships-active-inactive*.

To create a relationship between two tables, you can drag a key from one table on top of the corresponding key from the other table in the **Model** view.

NOTE **AUTOMATIC DETECTION OF RELATIONSHIPS**

By default, Power BI will try to detect relationships between tables automatically after you load data. In doing so, Power BI usually relies on identical column names, and the process is not always perfect. You can turn it off in Options > Current file > Data load if required.

In our Wide World Importers example, you can drag the **Date** column from the **Date** table on top of the **Invoice Date Key** column in the **Sale** table. This will create an active relationship, signified by the solid line. Next, you can drag the **Date** column from the **Date** table on top of the **Delivery Date Key** column from the **Sale** table. This will create an inactive relationship, signified by the dashed line. The result should look like Figure 2-3.

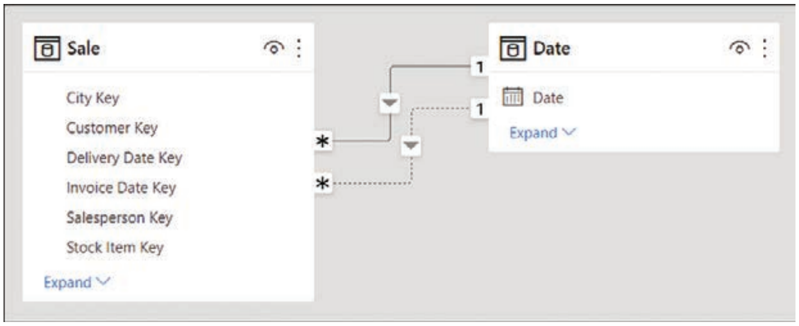


FIGURE 2-3 Relationships between Sale and Date

If you hover over a relationship line in the Model view, it'll highlight the fields that participate in the relationship.

NOTE **CARDINALITY AND CROSS-FILTER DIRECTION**

Note how each relationship line in Figure 2-3 has 1 and * at its ends, as well as an arrowhead in the middle. This represents the cardinality and cross-filter direction, respectively, and we review those concepts in the next section.

In our Wide World Importers model, you should also create the relationships listed in Table 2-1.

TABLE 2-1 Additional relationships in Wide World Importers

FROM: TABLE (COLUMN)	TO: TABLE (COLUMN)
Sale (City Key)	City (City Key)
Sale (Customer Key)	Customer (Customer Key)
Sale (Salesperson Key)	Employee (Employee Key)
Sale (Stock Item Key)	Stock Item (Stock Item Key)

Inactive relationships can be activated by using the `USERELATIONSHIP` function in DAX, which also deactivates the default active relationship, if any. The following is an example of a measure that uses `USERELATIONSHIP`:

```
Revenue by Delivery Date =
CALCULATE(
    [Revenue],
    USERELATIONSHIP(
        'Date'[Date],
        Sale[Delivery Date Key]
    )
)
```

To use `USERELATIONSHIP`, you need to define a relationship in the model first so that the function only works for existing relationships. This approach is useful for scenarios such as the Wide World Importers example, where we have multiple date columns within the same fact table.

NEED MORE REVIEW? `CALCULATE` **AND** `USERELATIONSHIP`

`CALCULATE` **is the most important function in DAX, and we review it in more detail later in this chapter. It's important to be aware of certain limitations of** `USERELATIONSHIP`**. For more information, see "USERELATIONSHIP" at** ***https://docs.microsoft.com/en-us/dax/userelationship-function-dax*****.**

If you have a number of measures that you want to analyze by using different relationships, this may result in your data model having many similar measures, cluttering your data model to a degree.

Another drawback of using `USERELATIONSHIP` is that you cannot analyze data by using two relationships at the same time. For instance, if you have a single Date table, it won't be possible to see which sales were invoiced last year and shipped this year.

An alternative to `USERELATIONSHIP` that addresses these drawbacks is to use separate dimensions for each role or relationship. In Wide World Importers, you would have Delivery Date and Invoice Date dimensions, which would make it possible to analyze sales by both delivery and invoice dates.

There are a few ways to create the new dimensions based on the existing Date table, one of which is to use calculated tables. For the Invoice Date table, the DAX formula would be as follows:

```
Invoice Date = 'Date'
```

The benefit of using calculated tables instead of referencing or duplicating queries in Power Query is that if you have calculated columns in your Date table, they will be copied in a calculated table, while you'll need to re-create the same columns if you use Power Query to create the copies of the dimension.

When you're creating separate dimensions, it's best to rename the columns to make it clear where fields are coming from. For example, instead of leaving the column called Date, it's best to rename it to **Invoice Date**. You can do so by right-clicking a field in the **Fields** pane and selecting **Rename** or by double-clicking a field. Alternatively, you can rename fields by using a more complex calculated table expression. For example, you could use the `SELECTCOLUMNS` function in DAX to rename columns.

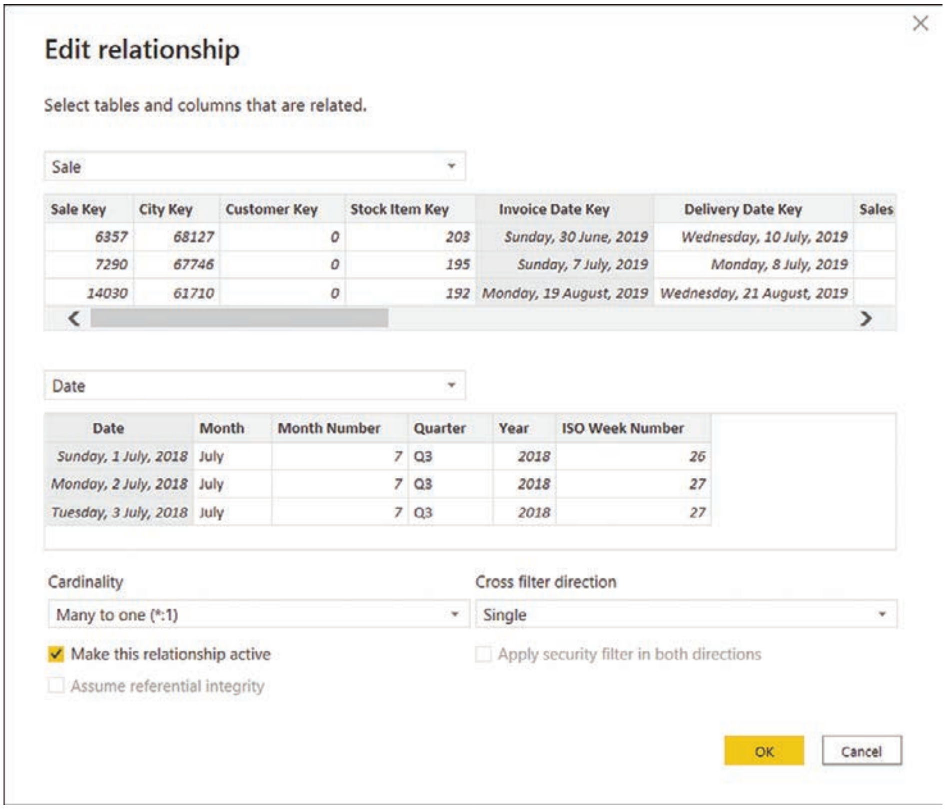
NOTE **CALCULATED TABLES**

DAX allows you to create far more sophisticated calculated tables than copies of existing tables. We review calculated tables in more detail in Skill 2.2: Develop a data model.

Define a relationship's cardinality and cross-filter direction

In the previous section, we looked at how you create relationships between tables. In this section, we review the concepts of cardinality and cross-filter direction of relationships.

You can edit a relationship by double-clicking it in the Model view. For example, in Figure 2-4 you can see the options for one of the relationships between the Sale and Date tables.


Edit relationship

Select tables and columns that are related.

Sale

Sale Key	City Key	Customer Key	Stock Item Key	Invoice Date Key	Delivery Date Key	Sales
6357	68127	0	203	Sunday, 30 June, 2019	Wednesday, 10 July, 2019	
7290	67746	0	195	Sunday, 7 July, 2019	Monday, 8 July, 2019	
14030	61710	0	192	Monday, 19 August, 2019	Wednesday, 21 August, 2019	

Date

Date	Month	Month Number	Quarter	Year	ISO Week Number
Sunday, 1 July, 2018	July	7	Q3	2018	26
Monday, 2 July, 2018	July	7	Q3	2018	27
Tuesday, 3 July, 2018	July	7	Q3	2018	27

Cardinality: Many to one (*:1)

Cross filter direction: Single

[✓] Make this relationship active

[ ] Apply security filter in both directions

[ ] Assume referential integrity

FIGURE 2-4 Relationship options

In the relationship options, you can select tables from drop-down lists. You get a preview of each table, from which you can select a column that will be part of a relationship. Unlike in the Merge operation in Power Query, only one column from each table can be part of a relationship.

The **Make this relationship active** check box determines whether the relationship is active. Between two tables, there can be no more than one active relationship.

When you're using DirectQuery, the **Assume referential integrity** option is available, and it can improve query performance in certain cases.

NOTE **ASSUME REFERENTIAL INTEGRITY**

There are some requirements that data must meet for the **Assume referential integrity** option to work properly. For advanced details on this feature, including the requirements and implications of not meeting the requirements with this option set, see "Apply the Assume Referential Integrity setting in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/connect-data/desktop-assume-referential-integrity*.

Two options are worth reviewing in more detail: **Cardinality** and **Cross filter direction**.

Cardinality

Depending on the selected tables and columns, you can select one of the following options:

- Many to one
- One to one
- One to many
- Many to many

Many to one and *One to many* are the same kind of relationship, and they differ only in the order in which the tables are listed. "Many" means that a key may appear more than once in the selected column, whereas "One" means a key value appears only once in the selected column. In our Wide World Importers example earlier, the Sale table was on the *many* side, whereas the Date table was on the *one* side; a single date appeared only once in the Date table, though there could be multiple sales on the same date in the Sale table.

One to one is a special kind of relationship where a key value appears only once on both sides of the relationship. This type of relationship may be useful for splitting a single dimension with many columns into separate tables. You should only use this if you are confident that no duplicates will appear in this table, because duplicates will cause immediate errors in your data model.

NEED MORE REVIEW? **ONE-TO-ONE RELATIONSHIPS**

One-to-one relationships are rarely encountered in real life. For advanced information on this type of relationships in Power BI, see "One-to-one relationship guidance" at *https://docs.microsoft.com/en-us/power-bi/guidance/relationships-one-to-one*.

Many-to-many relationships in this context refer to a direct relationship between two tables, neither of which is guaranteed to have unique keys. We review this type of relationship later in this chapter.

Cross filter direction

This option determines the direction in which filters flow. For many-to-one and one-to-many relationships, you can select Single or Both:

- If you select **Single**, then the filters from the table on the "one" side will filter through to the table on the "many" side. This setting is signified by a single arrowhead on the relationship line in the Model view.
- If you select **Both**, then filters from both tables will flow in both directions; such relationship are known as *bidirectional*. This setting is signified by two arrowheads on the relationship line in the Model view, facing in opposite directions. When this option is selected, you can also select **Apply security filter in both directions** to make row-level security filters flow in both directions too.

When editing table relationships, even if you set the relationship cross-filter direction to Both, by default the security filters are only applied in one direction. We noted that there's an option to control security filtering called **Apply security filter in both directions**. This means that role filtering applied to a table will also be passed to the filtered table. When this option is disabled, only the table with filtered applied will be affected. This option exists because applying security filters affects the performance of your data model, so in some cases applying it may be undesirable.

NOTE **SECURITY FILTERS**

Security filters refers to row-level security (RLS), a feature in Power BI that allows you to restrict access to data within a dataset based on a set of filters. We review row-level security in detail later in this chapter.

To illustrate how the cross-filter direction works, consider the data model shown in Figure 2-5.

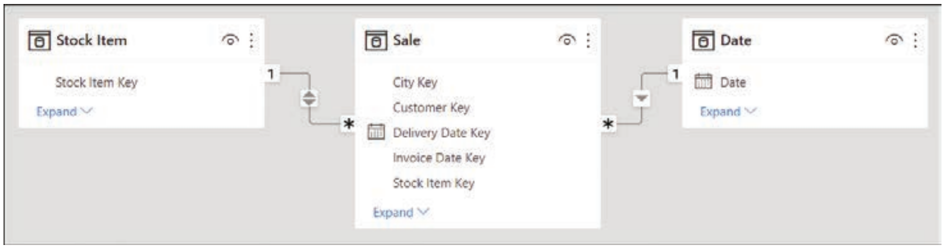


FIGURE 2-5 Sample data model

From this data model, you can create two table visuals as follows:

- Table 1: Distinct count of Stock Item by Year
- Table 2: Distinct count of Year by Stock Item

Both table visuals are shown in Figure 2-6. The first four rows are shown for Table 2 for illustrative purposes.

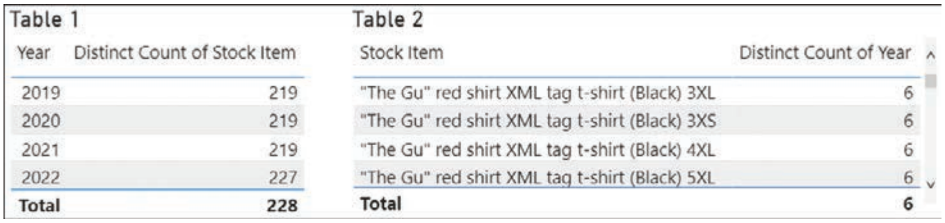


Table 1

Year	Distinct Count of Stock Item
2019	219
2020	219
2021	219
2022	227
Total	**228**

Table 2

Stock Item	Distinct Count of Year
"The Gu" red shirt XML tag t-shirt (Black) 3XL	6
"The Gu" red shirt XML tag t-shirt (Black) 3XS	6
"The Gu" red shirt XML tag t-shirt (Black) 4XL	6
"The Gu" red shirt XML tag t-shirt (Black) 5XL	6
Total	**6**

FIGURE 2-6 Table visuals

You can see that in Table 1, the numbers are different for different years and the total, whereas in Table 2, the Distinct Count of Year is showing 6 for all rows, including the Total.

The numbers are different in Table 1 because filters from the Date table can reach the Stock Item table through the Sale table; the Date table filters the Sale table because there is a one-to-many relationship; then the Sale table filters the Stock Item table because there is a bidirectional relationship. In 2019, 2020, and 2021, Wide World Importers coincidentally sold 219 stock items, whereas in 2022, they sold 227 stock items. At the total level you see 228, which is not the total sum of stock items sold across all years. Importantly, the total 228 is showing as the distinct count of stock items when filters from the Date table are not applied.

In Table 2, the numbers are the same because filters from the Stock Item table don't reach the Date table as there is no bidirectional filter. Even though Wide World Importers only had sales in four years, you see 6 across all rows, which is the number of years in the Date table.

It's also possible to set the cross-filter direction by using the `CROSSFILTER` function in DAX, as you can see in this example:

```
Stock Items Sold =
CALCULATE(
    DISTINCTCOUNT('Stock Item'[Stock Item]),
    CROSSFILTER(
        Sale[Stock Item Key],
        'Stock Item'[Stock Item Key],
        BOTH
    )
)
```

The syntax of `CROSSFILTER` is similar to `USERELATIONSHIP`—the first two parameters are related columns. Additionally, there's the third parameter—direction—and it can be one of the following:

- **BOTH** This option corresponds to **Both** in the relationship cross-filter direction options.
- **NONE** This option deactivates the relationship, and it corresponds to the cleared **Make this relationship active** check box.
- **ONEWAY** This option corresponds to **Single** in the relationship cross-filter direction options.

Bidirectional filters are sometimes used in many-to-many relationships with bridge tables when direct many-to-many relationships are not desirable.

NEED MORE REVIEW? **BIDIRECTIONAL RELATIONSHIPS**

For more examples and information on bidirectional relationships, see "Bi-directional relationship guidance" at *https://docs.microsoft.com/en-us/power-bi/guidance/relationships-bidirectional-filtering*.

NEED MORE REVIEW? **RELATIONSHIPS TROUBLESHOOTING**

Relationships may not work as expected for numerous reasons. For a comprehensive troubleshooting guide, see "Relationship troubleshooting guidance" at *https://docs.microsoft.com/en-us/power-bi/guidance/relationships-troubleshoot*.

Create a common date table

By default, Power BI creates a calendar hierarchy for each date or date/time column from your data sources.

> ***NEED MORE REVIEW?*** **AUTO DATE/TIME HIERARCHIES**
>
> **For detailed considerations and limitations of the auto date/time feature, see "Auto date/time guidance in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/guidance/auto-date-time*.**

While these can be useful in some cases, it's best practice to create your own date table, which has several benefits:

- You can use a calendar other than Gregorian.
- You can have weeks in the calendar.
- You can filter multiple fact tables by using a single date dimension table.

If you don't have a date table you can import from a data source, you can create one yourself. It's possible to create a date table by using Power Query or DAX, and there's no difference in performance between the two methods.

Creating a calendar table in Power Query

In Power Query, you can use the M language `List.Dates` function, which returns a list of dates, and then convert the list to a table and add columns to it. The following query provides a sample calendar table that begins on January 1, 2018:

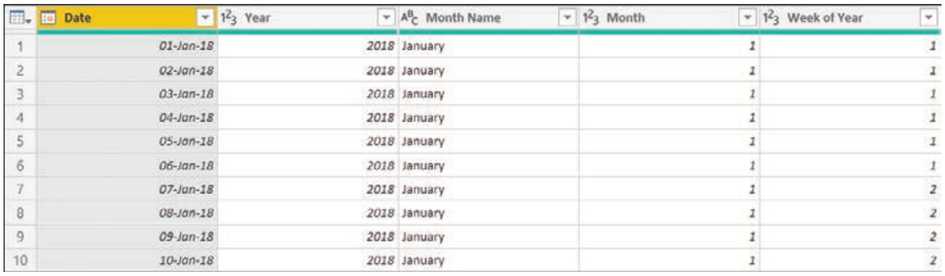
```
let
    Source = #date(2018, 1, 1),
    Dates = List.Dates(Source, Duration.TotalDays(Date.AddYears(Source, 6) - Source),
  #duration(1,0,0,0)),
    #"Converted to Table" = Table.FromList(Dates, Splitter.SplitByNothing(), type
  table [Date = date]),
    #"Inserted Year" = Table.AddColumn(#"Converted to Table", "Year", each Date.
  Year([Date]), Int64.Type),
    #"Inserted Month Name" = Table.AddColumn(#"Inserted Year", "Month Name", each Date.
  MonthName([Date]), type text),
    #"Inserted Month" = Table.AddColumn(#"Inserted Month Name", "Month", each Date.
  Month([Date]), Int64.Type),
    #"Inserted Week of Year" = Table.AddColumn(#"Inserted Month", "Week of Year", each
  Date.WeekOfYear([Date]), Int64.Type)
in
    #"Inserted Week of Year"
```

If you want to add it to your model, you'll need to start with a blank query:

1. In Power Query Editor, select **New Source** on the **Home** ribbon.
2. Select **Blank Query**.
3. With the new query selected, select **Query** > **Advanced Editor** on the **Home** ribbon.

4. Replace all existing code with the code above and select **Done**.
5. Give your query an appropriate name such as *Calendar* or *Date*.

The result should look like Figure 2-7, where the first few rows of the query are shown.



	Date	Year	Month Name	Month	Week of Year
1	01-Jan-18	2018	January	1	1
2	02-Jan-18	2018	January	1	1
3	03-Jan-18	2018	January	1	1
4	04-Jan-18	2018	January	1	1
5	05-Jan-18	2018	January	1	1
6	06-Jan-18	2018	January	1	1
7	07-Jan-18	2018	January	1	2
8	08-Jan-18	2018	January	1	2
9	09-Jan-18	2018	January	1	2
10	10-Jan-18	2018	January	1	2

FIGURE 2-7 Sample calendar table built by using Power Query

You may prefer having a table in Power Query when you intend to use it in some other queries, since it's not possible to reference calculated tables in Power Query.

Creating a calendar table in DAX

If you choose to create a date table in DAX, you can use the CALENDAR or CALENDARAUTO function, both of which return a table with a single Date column. You can then add calculated columns to the table, or you can create a calculated table that already has all the columns.

NOTE **CALCULATED TABLES**

We review the skills necessary to create calculated tables in Skill 2.2: Develop a data model.

The CALENDAR function requires you to provide the start and end dates, which you can hard-code for your business requirements or calculate dynamically:

```
Calendar Dynamic =
CALENDAR(
    MIN(Sale[Invoice Date Key]),
    MAX(Sale[Invoice Date Key])
)
```

The CALENDARAUTO function scans your data model for dates and returns an appropriate date range automatically.

To build a table similar to the Power Query table you built earlier, use the following calculated table formula in DAX:

```
Calendar =
ADDCOLUMNS(
    CALENDARAUTO(),
    "Year", YEAR([Date]),
    "Month Name", FORMAT([Date], "MMMM"),
    "Month", MONTH([Date]),
    "Week of Year", WEEKNUM([Date])
)
```

NEED MORE REVIEW? **CREATING DATE TABLES**

For more examples of how you can create a date table, see "Create date tables in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/guidance/model-date-tables*.

Skill 2.2: Develop a data model

Data model development refers to enhancements you add to your model after you've loaded your data and created relationships between tables. In this section, we review the skills you need to create calculated tables, calculated columns, and hierarchies, and we demonstrate how to configure row-level security for your report as well as set up the Q&A feature.

This skill covers how to:

- Create calculated tables
- Create hierarchies
- Create calculated columns
- Implement row-level security roles
- Use the Q&A feature

NOTE **COMPANION FILE**

The completed examples from this section are available in the 2.2 Develop.pbix file in the companion files folder.

Create calculated tables

Earlier in the chapter, you saw that one way to create a calendar table is to create a calculated table, which is an alternative to using Power Query. Calculated tables are defined by using DAX, and they're based on the data that is already loaded into the data model or new data generated by using DAX. You won't see calculated tables in Power Query Editor.

Calculated tables are especially useful when you want to:

- Clone tables, including calculated columns
- Create tables that are based on data from different data sources
- Precalculate measures to improve report performance

This list is not exhaustive—there are other cases when calculated tables are useful.

Cloning tables

You can use DAX to clone a table. To create a table called Invoice Date that's a clone of the Date table, perform the following steps:

1. Go to the **Data** view.
2. Select **New table** on the **Home** ribbon.
3. Enter a calculated table expression. For example, this formula creates a table called Invoice Date by copying the Date table:

   ```
   Invoice Date = 'Date'
   ```

4. Press **Enter**.

Creating tables that are based on data from different data sources

Sometimes—for example, when creating a bridge table—you may need to extract distinct values from more than one table because the distinct values may be different in different tables. In that case, you'd need to take distinct values from both tables, and if they come from different data sources or from different "islands," or both, the performance may be slow. You can solve this issue by using a calculated table.

For example, you could retrieve the distinct Buying Group values from both the Customer and Targets tables by using the following calculated table formula:

```
Buying Group =
DISTINCT(
    UNION(
        DISTINCT(Customer[Buying Group]),
        DISTINCT(Targets[Buying Group])
)
```

The `DISTINCT` function ensures there are no duplicates, and `UNION` combines values from two tables that come from different sources. `UNION` acts similarly to appending tables in Power Query, though they combine tables differently:

- `UNION` ignores column names and combines table columns based on their positions. The number of columns between tables must match.
- Appending tables in Power Query combines tables based on column names, and it's possible to combine tables that have a different number of columns.

In addition to `UNION`, other set functions available in DAX include `EXCEPT` and `INTERSECT`, which also require that tables have the same number of columns.

Since the data is already in memory, this process is usually much quicker compared to creating the same table by using Power Query.

Precalculating measures to improve report performance

If you have complex measures that perform poorly, depending on the type of calculation you may want to precalculate them in a calculated table, and then create new measures that aggregate the precalculated values. This approach may not work for some types of calculations, though it usually helps with additive measures.

Aggregations, which are outside the scope of the exam, are an example of calculated tables that precalculate measures and improve performance.

NEED MORE REVIEW? **USING CALCULATED TABLES**

For more details on how to create calculated tables, see "Create calculated tables in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/transform-model/desktop-calculated-tables*.

Create hierarchies

Power BI allows you to group columns into hierarchies, which you can then use in visuals.

In our Wide World Importers example, you can create a geographical hierarchy as follows:

1. Go to the **Model** view.
2. Right-click the **Sales Territory** column in the **City** table.
3. Select **Create hierarchy**.
4. Double-click the newly created hierarchy and rename it to **Geography**.
5. In the Fields pane, drag the **State Province** column on top of **Geography**.
6. Repeat the previous step for the **City** column.

Once created, the result should look like Figure 2-8.

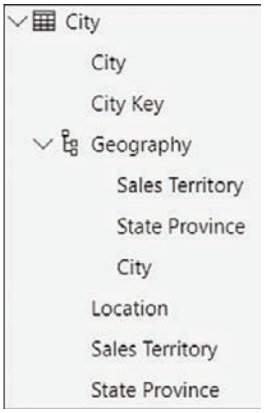


FIGURE 2-8 The Geography hierarchy

One column can be part of multiple hierarchies, and you can rename hierarchy items without affecting the original columns. At the same time, you don't need to sort a hierarchy element by another column, because it inherits this property from the original column. The original column can be hidden, if desired.

A hierarchy can be created using only existing columns, and the columns must be in the same table. To include columns from different tables in the same hierarchy, you have to bring the columns into one table. You can do that by using Power Query or the RELATED function in DAX, for example.

NEED MORE REVIEW? **RELATED IN DAX**

For more information on the RELATED function in DAX, including examples and the requirements for using the function, see "RELATED" at *https://docs.microsoft.com/en-us/dax/related-function-dax*.

Apart from the convenience of dragging multiple fields to a visual at once in the right order, a hierarchy does not provide any special benefits in Power BI compared to using columns individually, because you can use both hierarchies and multiple individual columns together in fields to achieve the same result.

Create calculated columns

Calculated columns are columns you create by using DAX. Similar to calculated tables, calculated columns can only use the data already loaded into the model or new data generated by DAX, and they don't appear in Power Query Editor because they are generated after the data has been loaded into the model. By nature, creating calculated columns widens your table, and they are calculated after all your data is loaded, so multiple calculated columns can contribute to slow performance of your data model.

If you're experienced in Excel, creating calculated columns in DAX may remind you of creating columns in Excel, because DAX resembles the Excel formula language, and there are many functions that appear in DAX and Excel. There are some important differences, however:

- In DAX, there is no concept of a cell. If you need to get a value from a table, you have to filter a specific column down to that value.
- DAX is strongly typed; it's not possible to mix values of different data types in the same column.

In general, calculated columns are especially useful when you are:

- Creating columns to be used as filters or categories in visuals
- Precalculating poorly performing measures

Here's one way to create a calculated column:

1. Go to the **Data** view.
2. In the **Fields** pane, right-click a table where you want to create a calculated column.
3. Select **New column**.
4. Enter a calculated column expression by using DAX.
5. Press **Enter**.

After you complete these steps, you'll be able to see the results immediately. The formula that you write is automatically applied to each row in the new column. You can reference another column from the same table in the following way:

```
'Table name'[Column name]
```

Though it's possible to reference a column within the same table by only using the column name, it's not considered a good practice and should be avoided.

For example, in Wide World Importers, you can calculate total cost in a calculated column in the Sale table by using the following expression:

```
Total Cost = Sale[Total Excluding Tax] - Sale[Profit]
```

If you want to reference a column from a related table that is on the one side of a relationship, you can use the `RELATED` function. For instance, in Wide World Importers, you can add a calculated column to the Sale table to calculate the price difference between the standard unit price and the price a product was sold for:

```
Unit Price Difference = RELATED('Stock Item'[Unit Price]) - Sale[Unit Price]
```

NOTE **`RELATED` AND INACTIVE RELATIONSHIPS**

By default, `RELATED` uses the active relationship. Though it's possible to make `RELATED` use an inactive relationship, it's much better to use the `LOOKUPVALUE` function for this. For more information on the function, see "LOOKUPVALUE" at *https://docs.microsoft.com/en-us/dax/lookupvalue-function-dax*.

`RELATED` works on the many side of a relationship. If you want to add a column to the one side of a relationship and reference the related rows, you can use the `RELATEDTABLE` function. For instance, you can add a calculated column to the Customer table to count the number of related rows in the Sale table for each customer:

```
Sales Rows = COUNTROWS(RELATEDTABLE(Sale))
```

EXAM TIP

Unless you want to use the values generated from the calculated column as filters or categories in visuals, you should be creating measures, which we cover in Skill 2.3: Create model calculations by using DAX.

NEED MORE REVIEW? **CREATE CALCULATED COLUMNS**

For more examples and a tutorial on how to create calculated columns, see "Tutorial: Create calculated columns in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/transform-model/desktop-tutorial-create-calculated-columns*.

Implement row-level security roles

A common business requirement is to secure data so that different users who view the same report can see different subsets of data. In Power BI, this can be accomplished with the feature called row-level security (RLS).

Row-level security restricts data by filtering it at the row level, depending on the rules defined for each user. To configure RLS, you first need to create and define each role in Power BI Desktop, and then assign individual users or Active Directory security groups to the roles in the Power BI service.

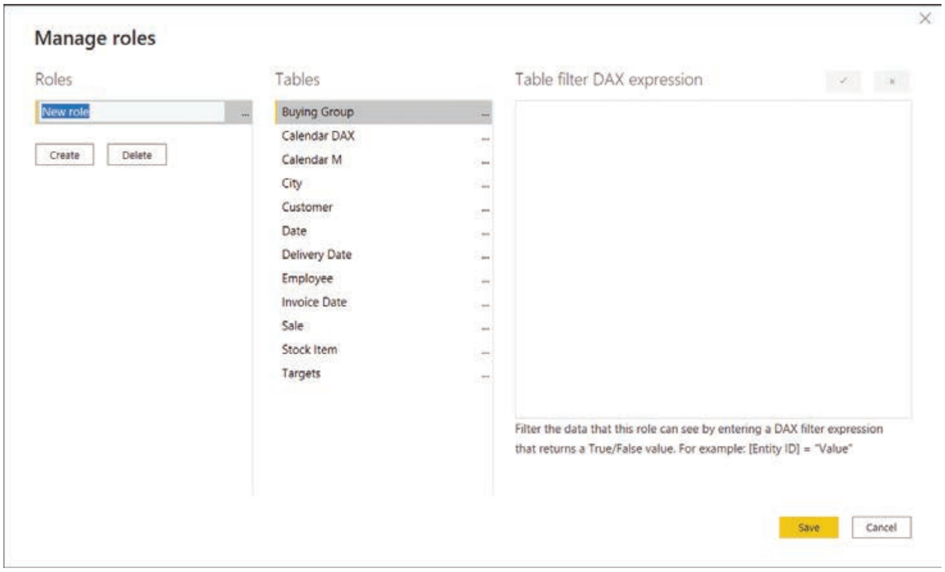
NOTE **ROW-LEVEL SECURITY AND LIVE CONNECTIONS**

Defining roles in Power BI only works for imported data and DirectQuery. When you connect live to a Power BI dataset or an Analysis Services data model, Power BI will rely on row-level security configured in the source, and you cannot override it by creating roles in Power BI Desktop.

In this section, we review the skills necessary to implement row-level security roles in Power BI Desktop. We examine assignment of roles in the Power BI service in Chapter 4, "Deploy and maintain assets."

Creating roles in Power BI Desktop

To see the list of roles configured in a dataset in Power BI Desktop, select **Manage roles** from the **Modeling** ribbon in the **Report** view. To create a new role, select **Create** in the **Roles** section. You'll then be prompted to specify table filters, as shown in Figure 2-9.



Filter the data that this role can see by entering a DAX filter expression that returns a True/False value. For example: [Entity ID] = "Value"

FIGURE 2-9 Manage roles

When you create a role, you have the option to change the default name to a new one. It's important to give roles user-friendly names because you'll see them in Power BI service, and you need to be able to assign users to the correct roles. All roles are listed in the **Roles** section of the **Manage roles** window.

If you right-click on a role or select the ellipsis next to a role, you'll be presented with the following options:

- **Create** This option creates a new role and is an alternative to the Create button below the list of roles.
- **Duplicate** This option creates a copy of the currently selected role.
- **Rename** Use this option to rename the currently selected role; you can also rename a role by double-clicking on its name.
- **Delete** This option deletes the currently selected role; this action can also be performed by selecting Delete below the list of roles.

For each role, you can define a DAX expression to filter each table. When row-level security is configured, these expressions will be evaluated against each row of the relevant table, and only those rows for which the expressions are evaluated as *true* will be visible.

You can either enter a table filter DAX expression yourself or use the ellipsis menu next to each table to add an expression that you can then customize. You can also access the menu by right-clicking on a table and choosing from these options:

- **Add filter** This option lists all columns available in the table and lets you hide all rows.
- **Copy table filter from** This option copies a table filter DAX expression from another role that has a filter expression defined for the table.
- **Clear table filter** This option removes any table filter DAX expression from the table. It's a shortcut to erasing all text from the Table filter DAX expression area manually.

For example, in the Wide World Importers data model that we previously created, you can select the ellipsis next to **City** > **Add filter** > **[Sales Territory]** to insert an expression, as shown in the Table filter DAX expression area:

```
[Sales Territory] = "Value"
```

The placeholder expression depends on the data type of the column, and it helps you to write the correct filter expression.

After you modify the expression, you can validate it by selecting the **Verify DAX expression** (check mark) button above the Table filter DAX expression area. If the expression is invalid, you'll see a warning stating that the syntax is incorrect below the Table filter DAX expression area. Next to the check mark button is the **Revert changes** (cross) button, which reverts any changes that haven't been applied yet.

To hide all rows in a table, right-click on the table and click **Add filter** > **Hide all rows**. This will add the following table filter DAX expression:

```
false
```

Because *false* is never going to be *true* for any row, no rows will be shown in this case.

You can configure row-level security in the Wide World Importers data model. First, create two roles as follows:

1. Create a new role and call it **Southeast**.
2. For the **Southeast** role, in the **City** table, enter the following table filter DAX expression:

   ```
   [Sales Territory] = "Southeast"
   ```

3. Select the **Verify DAX expression** button above the **Table filter DAX expression** area.
4. Right-click the **Southeast** role and select **Duplicate**.
5. Rename the new role to **Plains**.
6. For the Plains role, update the table filter DAX expression in the City table as follows:

   ```
   [Sales Territory] = "Plains"
   ```

7. Select **Save**.

IMPORTANT **DUPLICATING ROLES**

If you duplicate a role before you verify the last added table filter, the table filter will not be copied to the duplicate role.

We can now test the roles in Power BI Desktop.

Viewing as roles in Power BI Desktop

In Power BI Desktop, you can check what the users with specific roles will see even before you publish your report to the Power BI service and assign users to roles. For this, once you have at least one role defined, select **View as** on the **Modeling** ribbon in the **Report** view. You'll then see the **View as roles** window shown in Figure 2-10.

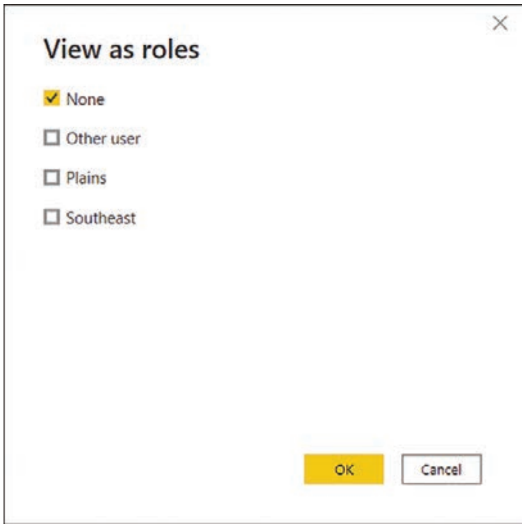


FIGURE 2-10 View as roles

Note that you can view as several roles simultaneously. This is because you can allocate a single user or a security group to multiple roles in the Power BI service; in this case, the security rules of the roles will complement each other. For example, if you select both the **Plains** and the **Southeast** roles, you'll see data for both territories. For this reason, you should always have clear names for your RLS roles.

EXAM TIP

You should know the effect of combining multiple security roles for a single user based on the description of security filters of each role.

When viewing data as roles, you'll see the bar at the top shown in Figure 2-11.

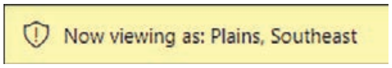

Now viewing as: Plains, Southeast

FIGURE 2-11 Now viewing report as

IMPORTANT **APPLICATION OF ROW-LEVEL SECURITY**

The filters applied by row-level security are applied only at query time and not at processing time. The implication of this is that the filters won't change the values of calculated columns and calculated tables.

Another option in the **View as roles** window is **Other user**. With this option, you can test dynamic row-level security, which is covered next.

Dynamic row-level security

The roles we've created so far have been static, which means that all users within a role will see the same data. If you have many rules that specify how you should secure your data, this approach may mean you have to create a number of roles as well as update the data model every time a new role should be introduced or an old one removed.

There is an alternative approach, called dynamic row-level security, which allows you to show different data to different users within the same role.

NOTE **DYNAMIC ROW-LEVEL SECURITY**

Because dynamic row-level security can use a single role, this approach is preferable in large-scale implementations of Power BI where there are many users who need to see different data.

For this approach, your data model must contain the usernames of people who should have access to the relevant rows of data. You'll also need to pass the active username as a filter condition. Power BI has two functions that allow you to get the username of the current user:

- **USERNAME** This function returns the domain and login of the user in the domain\login format.
- **USERPRINCIPALNAME** Depending on how the Active Directory was set up, this function usually returns the email address of the user.

NOTE **USING USERNAME AND USERPRINCIPALNAME**

If your computer is not part of an Active Directory domain, both functions will return the same result—domain\login. Once you publish your dataset to the Power BI service, both functions will return the email address of the user.

These functions can only be used in measures or table filter DAX expressions; if you try to use either of them in a calculated column or a calculated table, you'll get an error.

To see how dynamic row-level security works in our Wide World Importers data model, first create a new security role:

1. Select **Manage roles** on the **Modeling** ribbon.
2. Create a new security role and call it **Dynamic RLS**.
3. For the **Dynamic RLS** role, specify the following table filter DAX expression for the **Employee** table:

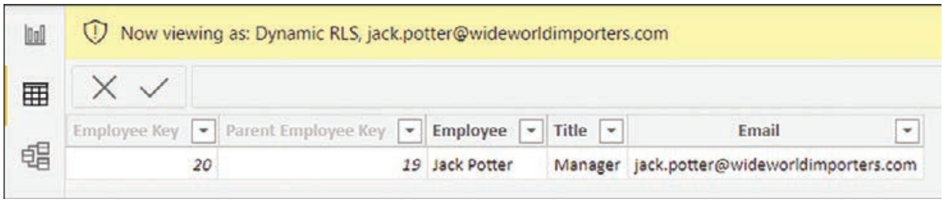
   ```
   [Email] = USERPRINCIPALNAME()
   ```

4. Select **Save**.

Now you can test the new role:

1. Select **View as** on the **Modeling** ribbon.
2. Select both **Other user** and **Dynamic RLS**.
3. Enter **jack.potter@wideworldimporters.com** in the **Other user** box.
4. Select **OK**.
5. Go to the **Data** view.
6. Select the **Employee** table.

Note that the Employee table is now filtered to just Jack Potter's row, as shown in Figure 2-12.



Now viewing as: Dynamic RLS, jack.potter@wideworldimporters.com

Employee Key	Parent Employee Key	Employee	Title	Email
20	19	Jack Potter	Manager	jack.potter@wideworldimporters.com

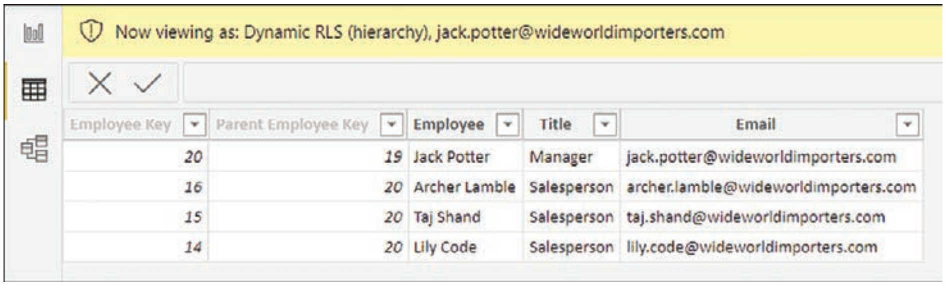
FIGURE 2-12 Employee table viewed as Jack Potter

Although this may be good enough for us in certain cases, it's a common requirement for managers to see the data of those who report to them. Since Jack is a manager, he should be able to see data of the salespersons who report to him. For that, we can create a new role called **Dynamic RLS (hierarchy)** with the following table filter DAX expression:

```
PATHCONTAINS(
    PATH(
        Employee[Employee Key],
        Employee[Parent Employee Key]
    ),
    LOOKUPVALUE(
        Employee[Employee Key],
        Employee[Email],
        USERPRINCIPALNAME()
    )
)
```

This table filter DAX expression keeps those rows where Jack is part of the hierarchy path, which relies on the Employee table having both the ID and parent ID columns.

After you make this change, the Employee table will show four rows: Jack's row and three rows of the salespersons who report to Jack, as seen in Figure 2-13.



Now viewing as: Dynamic RLS (hierarchy), jack.potter@wideworldimporters.com

Employee Key	Parent Employee Key	Employee	Title	Email
20	19	Jack Potter	Manager	jack.potter@wideworldimporters.com
16	20	Archer Lamble	Salesperson	archer.lamble@wideworldimporters.com
15	20	Taj Shand	Salesperson	taj.shand@wideworldimporters.com
14	20	Lily Code	Salesperson	lily.code@wideworldimporters.com

FIGURE 2-13 Employee table viewed as Jack Potter

So far, you've created the roles in Power BI Desktop. Once you publish the report, you'll have to assign users or security groups to roles in Power BI service separately. We review these skills in Chapter 4.

NEED MORE REVIEW? **ROW-LEVEL SECURITY**

For more examples of implementing row-level security in Power BI, see "Row-level security (RLS) guidance in Power BI Desktop" at *https://docs.microsoft.com/en-us/power-bi/guidance/rls-guidance*.

Use the Q&A feature

Both Power BI Desktop and the Power BI service allow you to create visualizations that provide answers to specific questions. Although this gives you great control over formatting, it won't work if you have RLS set up and users only have read access to content.

Another way to explore data in Power BI is to use the Q&A feature, also known as *natural language queries*. This feature enables you to get answers to your questions by typing them in natural language. Even users with read-only access can query datasets in a natural language.

To start using Q&A in Power BI Desktop, you need to be in the Report view. To insert the Q&A visual, double-click the empty space on the report canvas. Alternatively, you can select **Q&A** on the **Insert** ribbon. Either way, you'll see a visual, as shown in Figure 2-14.

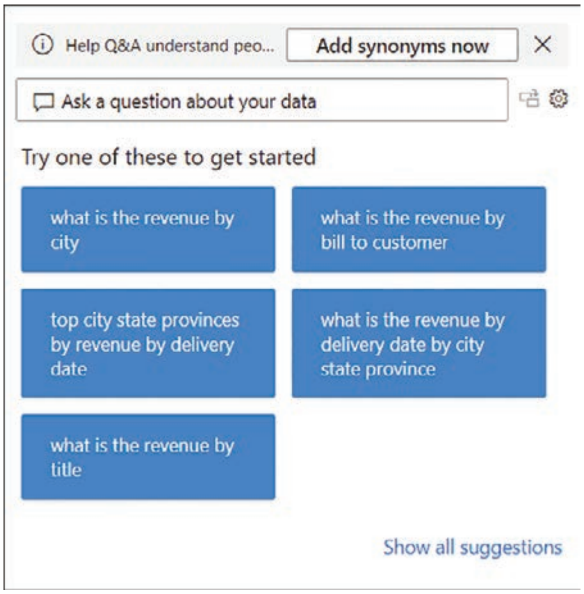

Help Q&A understand peo...
Add synonyms now
Ask a question about your data
Try one of these to get started
what is the revenue by city
what is the revenue by bill to customer
top city state provinces by revenue by delivery date
what is the revenue by delivery date by city state province
what is the revenue by title
Show all suggestions

FIGURE 2-14 Q&A visual

Although the suggestions may not be immediately useful, you can ask your own questions. For example, you can enter **profit by sales territory as column chart**, and the result will look like Figure 2-15.

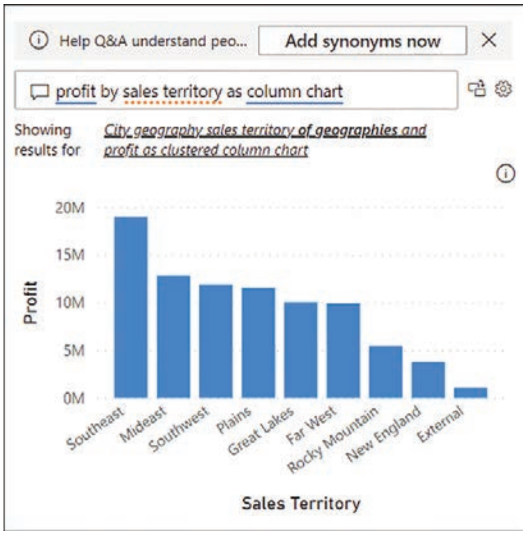
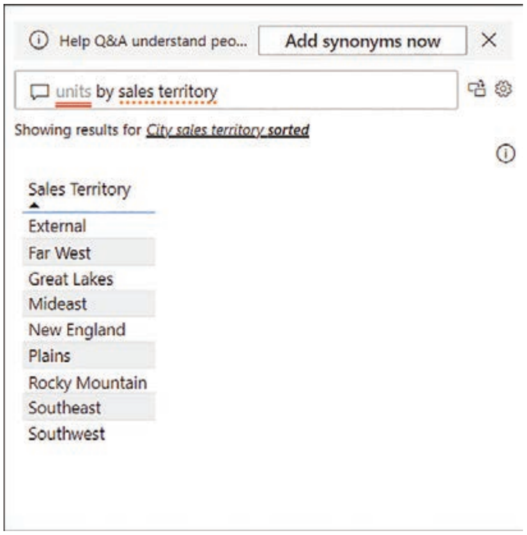

Help Q&A understand peo...
Add synonyms now
profit by sales territory as column chart
Showing results for City geography sales territory of geographies and profit as clustered column chart
20M
15M
10M
5M
0M
Profit
Southeast
Mideast
Southwest
Plains
Great Lakes
Far West
Rocky Mountain
New England
External
Sales Territory

FIGURE 2-15 Q&A showing profit by sales territory

Note that the Q&A visual updates its result as you type. Before we typed "as column chart," the Q&A visual was showing a bar chart.

If desired, you can turn the Q&A result into a standard visual by selecting the button between the question and the cog wheel in the upper-right corner of the Q&A visual.

The Q&A visual depends on the field names as they are defined in the data model. For example, entering **units by sales territory** in the Q&A visual won't provide any meaningful results, as seen in Figure 2-16.



Showing results for City sales territory sorted

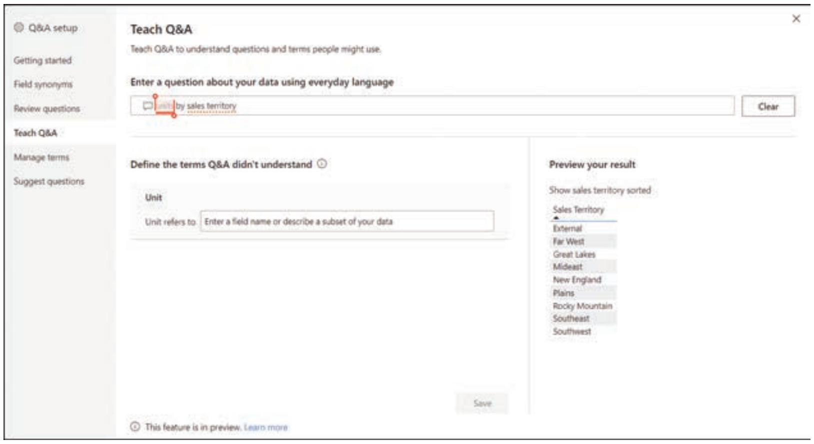
Sales Territory
External
Far West
Great Lakes
Mideast
New England
Plains
Rocky Mountain
Southeast
Southwest

FIGURE 2-16 Q&A visual showing units by sales territory

This issue can be fixed by *teaching* Q&A, as outlined next.

Teach Q&A

The Q&A visual didn't understand the term *units* because it doesn't appear in the Wide World Importers data model. The Q&A visual underlines in red the terms it doesn't understand. If you select *units* in the Q&A visual, you may be given suggestions to replace *units* with another term or to define the term. Selecting **define units** allows you to teach Q&A, as seen in Figure 2-17.



Q&A setup

Getting started

Field synonyms

Review questions

Teach Q&A

Manage terms

Suggest questions

Teach Q&A

Teach Q&A to understand questions and terms people might use.

Enter a question about your data using everyday language

units by sales territory

Clear

Define the terms Q&A didn't understand

Unit

Unit refers to: Enter a field name or describe a subset of your data

Save

Preview your result

Show sales territory sorted

Sales Territory

External

Far West

Great Lakes

Mideast

New England

Plains

Rocky Mountain

Southeast

Southwest

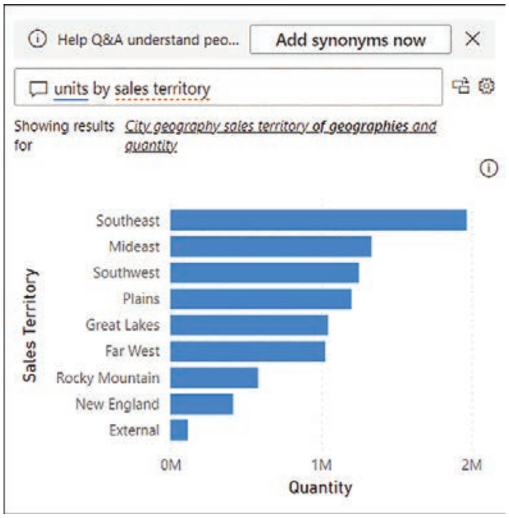
This feature is in preview. Learn more

FIGURE 2-17 The Teach Q&A window

In the **Define the terms Q&A didn't understand** section, you can teach Q&A that *units* refers to a certain field—for example, *quantity*—in the following way:

1. Enter **quantity** next to **Unit refers to**.
2. Select **Save**.
3. Close the **Q&A setup** window.

The Q&A visual now understands the term *units*, as you can see in Figure 2-18.



Help Q&A understand peo...

Add synonyms now

units by sales territory

Showing results for City geography sales territory of geographies and quantity

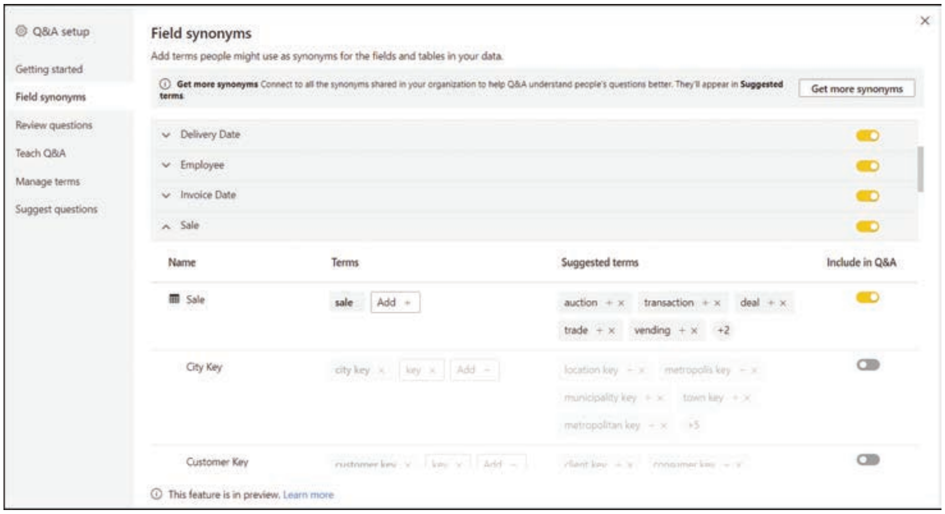
FIGURE 2-18 Q&A showing units by sales territory

Since teaching Q&A can be time-consuming, you can also add synonyms to your data model if you know them in advance, as covered next. This is another example where naming columns in Power Query Editor with friendly names will make this process easier.

Synonyms

Separately from teaching Q&A, you can introduce your own Q&A keywords and make Power BI recognize them. This is especially useful if your business users use acronyms or unique terminology such as substituting *margin* for *profit*. You can create a synonym for the Profit field, which will reduce confusion by your report users:

1. In the **Report** view, select **Q&A setup** on the **Modeling** ribbon.
2. Select **Field synonyms** on the left.
3. Expand the **Sale** section. You should see a list of fields in the Sale table, as shown in Figure 2-19.



Field synonyms

Add terms people might use as synonyms for the fields and tables in your data.

Get more synonyms Connect to all the synonyms shared in your organization to help Q&A understand people's questions better. They'll appear in **Suggested terms**.

Name	Terms	Suggested terms	Include in Q&A
Sale	sale, Add	auction, transaction, deal, trade, vending, +2	On
City Key	city key, key, Add	location key, metropolis key, municipality key, town key, metropolitan key, +5	Off
Customer Key	customer key, key, Add	client key, consumer key	Off

This feature is in preview. Learn more

FIGURE 2-19 Field synonyms for the Sale table

4. Next to **Profit**, select **Add**.
5. Enter **margin** and press **Enter**.
6. Close the **Q&A setup** window.

If you now enter **margin by color** in the Q&A visual, you'll see a bar chart showing Profit by Color, despite not using the term *profit* explicitly.

Additionally, in the **Field synonyms** section of **Q&A setup**, you can exclude specific tables and fields from Q&A if you don't want Q&A to use the table or field. Hidden objects are excluded by default. This is useful if you need to include staff data in your data model but don't want users to query this data.

NOTE **Q&A BEST PRACTICES**

Q&A, when set up correctly, can be a powerful tool that enables data exploration by report consumers. For Q&A best practices, see "Best practices to optimize Q&A in Power BI" at *https://docs.microsoft.com/en-us/power-bi/natural-language/q-and-a-best-practices*.

Skill 2.3: Create model calculations by using DAX

You used some DAX earlier in the chapter to create calculated tables and calculated columns as well as configure row-level security. In practice, DAX is most often used to create measures in Power BI.

Writing your own formulas is an important skill that allows you to perform much more sophisticated analysis based on your data compared to not using DAX.

In this section, we start by reviewing DAX fundamentals; then we look at CALCULATE, one of the most important functions in DAX, specifically in Time Intelligence or time-related calculations, which we review separately.

DAX can help you replace some columns with measures, allowing you to reduce the data model size. Not all DAX formulas need to be complex, and we review some basic statistical functions in this section as well.

This skill covers how to:

- Create basic measures by using DAX
- Use CALCULATE to manipulate filters
- Implement Time Intelligence using DAX
- Replace implicit measures with explicit measures
- Use basic statistical functions
- Create semi-additive measures
- Use quick measures

NOTE **COMPANION FILE**

The completed examples from this section are available in the 2.3 Create.pbix file in the companion files folder.

Create basic measures by using DAX

Although many things can be computed by using calculated columns, in most cases it's preferable to write measures, because they don't increase the model size. Additionally, some calculations are simply not possible with calculated columns. For example, to calculate a ratio dynamically, you need to write a measure.

As you saw earlier, quick measures already allow you to perform basic calculations without writing DAX yourself. In this section, you start using DAX to build complex measures.

It's important to understand that Power BI allows you to aggregate columns in visuals without using measures, a practice sometimes called *implicit measures*. These can be useful when you want to quickly test how a visual might look or to perform a quick analysis on a column. However, it's always best practice to create *explicit measures* by using DAX—even with trivial calculations such as `SUM`. Here are some reasons it's preferable to create measures yourself:

- Implicit measures may provide unexpected results in some cases due to the Summarize by column property. For example, if you have a column that contains product prices and Power BI sets the summarization to `SUM`, then dragging the column in a visual will not produce meaningful results. Although you can change the summarization in the visual, following this approach means that you need to pay attention to this property every time you use implicit measures.
- Explicit measures can be reused in other measures. This is beneficial because you can write less code, which saves time and improves the maintainability of your data model.
- Implicit measures cannot leverage inactive relationships.
- Implicit measures are not supported by calculation groups.

NEED MORE REVIEW? **CALCULATION GROUPS**

Calculation groups are outside the scope of the exam, but they can be extremely useful in practice. For more information, see "Calculation groups" at *https://docs.microsoft.com/en-us/analysis-services/tabular-models/calculation-groups*.

NOTE **LEARNING DAX**

Teaching DAX is not the purpose of this book. If you want to learn DAX, *The Definitive Guide to DAX* by Marco Russo and Alberto Ferrari (Pearson, 2019) is a great explanation of DAX and its use.

Measures are different from calculated columns in a few ways. The main difference is that you can see the results of a calculated column immediately after defining the calculation, whereas you can't see the results of a measure until you use it in a visual. This behavior allows measures to return different results depending on filters and where they're used.

Another difference between calculated columns and measures is that calculated column formulas apply to each row of a table, whereas measures work on columns and tables, not specific rows. Therefore, measures most often use aggregation functions in DAX.

There are a few ways to create a measure in Power BI Desktop. Here's one way:

1. Go to the **Report** view.
2. In the **Fields** pane, right-click a table in which you want to create a new measure.
3. Select **New measure**.
4. Enter the measure formula and press **Enter**.

You can also create a measure by selecting **New measure** on the **Home** ribbon, but you have to make sure you've got the right table selected in the **Fields** pane; otherwise, your measure may not be created in the correct table. If you do create a measure in the wrong table, instead of re-creating the measure you can move it by performing the following steps:

1. Go to the **Report** view.
2. In the **Fields** pane, select the measure you want to move.
3. On the **Measure tools** ribbon, select the table your measure should be stored in from the **Home table** dropdown list.

For example, to compute the total profit of Wide World Importers, use the following measure formula:

```
Total Profit = SUM(Sale[Profit])
```

You can compute total sales, excluding tax, by using the following measure formula:

```
Total Sales Excluding Tax = SUM(Sale[Total Excluding Tax])
```

If you want to compute the profit margin percentage, there are two ways of doing it. You could use this:

```
Profit % =
DIVIDE(
    SUM(Sale[Profit]),
    SUM(Sale[Total Excluding Tax])
)
```

NOTE **USING `DIVIDE`**

We're using `DIVIDE` in the formula to avoid division by 0. `DIVIDE` has an optional third parameter, which is the value to return in case you divide by 0.

However, this approach involves repeating your own code, which is undesirable because formulas become more difficult to maintain. You can avoid this issue if you reference the measures you created previously:

```
Profit % =
DIVIDE(
    [Total Profit],
    [Total Sales Excluding Tax]
)
```

NOTE **FORMATTING MEASURES**

Even though the Profit % measure has a percentage sign in its name, Power BI will format the measure as a decimal number by default. You can change the measure format on the **Measure tools** ribbon in the **Formatting** group. Formatting a measure after it's been created is a great habit to learn.

When you're referencing measures, it's best practice to not use table names in front of them. Unlike column names, measure names are unique; different tables may have the same column names, but it's not possible to have measures that share the same name.

Another feature of DAX that allows you to avoid repeating yourself is variables. Think of a variable as a calculation within a measure. For instance, if you want to avoid showing zeros in your visuals, you could write a measure as follows:

```
Total Dry Items Units Sold =
IF(
    SUM(Sale[Total Dry Items]) <> 0,
    SUM(Sale[Total Dry Items])
)
```

By using a variable, you can avoid calling `SUM` twice:

```
Total Dry Items Units Sold =
VAR TotalDryItems = SUM(Sale[Total Dry Items])
VAR Result =
    IF(
        TotalDryItems <> 0,
        TotalDryItems
    )
RETURN
    Result
```

Variables are especially useful when you want to store computationally expensive values, because variables are evaluated no more than once. As you'll see later in this chapter, you can use many variables within the same formula.

Use CALCULATE to manipulate filters

Earlier in this chapter, you saw that the `CALCULATE` function can be used to alter relationships when paired with other DAX measures. The `USERELATIONSHIP` function with `CALCULATE` can activate inactive relationships, and `CROSSFILTER` with `CALCULATE` can change the filter direction.

The `CALCULATE` function also allows you to alter the filter context under which measures are evaluated; you can add, remove, or update filters, or you can trigger context transition. We cover row context, filter context, and context transition in more detail later in this chapter.

`CALCULATE` accepts a scalar expression as its first parameter, and subsequent parameters are filter arguments. Using `CALCULATE` with no filter arguments is only useful for context transition.

Adding filters

`CALCULATE` allows you to add filters in several formats. To calculate profit for the New England sales territory, you can write a measure that you can read as "Calculate the Total Profit where the Sales Territory is New England":

```
New England Profit =
CALCULATE(
    [Total Profit],
    City[Sales Territory] = "New England"
)
```

Importantly, you're not limited to using one value per filter. You can calculate profit for New England, Far West, and Plains:

```
New England, Far West, and Plains Profit =
CALCULATE(
    [Total Profit],
    City[Sales Territory] IN {"New England", "Far West", "Plains"}
)
```

You can specify filters for different columns at once too, which are combined by using the AND DAX function. For example, you can calculate profit in New England in 2020 that reads as "Calculate the Total Profit where the Sales Territory is New England and the Year is 2020":

```
New England Profit 2020 =
CALCULATE(
    [Total Profit],
    City[Sales Territory] = "New England",
    'Date'[Year] = 2020
)
```

Removing filters

There are several DAX functions that you can use as CALCULATE modifiers to ignore filters, one of which is ALL. ALL can remove filters from:

- One or more columns from the same table
- An entire table
- The whole data model (when ALL is used with no parameters)

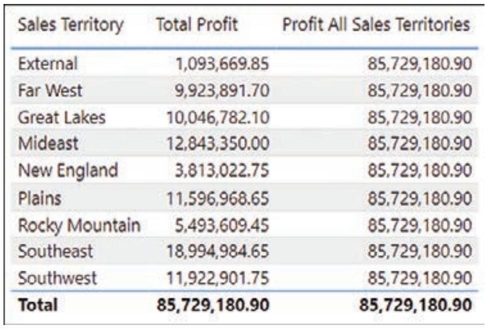
IMPORTANT **SORT BY COLUMN AND ALL**

If you're removing filters from a column that is sorted by another column, you should remove filters from both columns—otherwise, you may get unexpected results.

For example, you can show profit for all sales territories regardless of any filters on the City[Sales Territory] column:

```
Profit All Sales Territories =
CALCULATE(
    [Total Profit],
    ALL(City[Sales Territory])
)
```

If you create a table that shows the new measure alongside Total Profit by Sales Territory, you get the results shown in Figure 2-20.



Sales Territory	Total Profit	Profit All Sales Territories
External	1,093,669.85	85,729,180.90
Far West	9,923,891.70	85,729,180.90
Great Lakes	10,046,782.10	85,729,180.90
Mideast	12,843,350.00	85,729,180.90
New England	3,813,022.75	85,729,180.90
Plains	11,596,968.65	85,729,180.90
Rocky Mountain	5,493,609.45	85,729,180.90
Southeast	18,994,984.65	85,729,180.90
Southwest	11,922,901.75	85,729,180.90
Total	**85,729,180.90**	**85,729,180.90**

FIGURE 2-20 Total Profit and Profit All Sales Territories by Sales Territory

Note that the new measure displays the same value for any sales territory, which is the total of all sales territories combined regardless of sales territory.

NOTE **FILTER FUNCTIONS IN DAX**

In addition to `ALL`**, there are several other DAX functions that remove filters, such as** `ALLEXCEPT` **and** `ALLSELECTED`**. Full details of each function are outside the scope of this book. For an overview, see "Filter functions" at *https://docs.microsoft.com/en-us/dax/filter-functions-dax*.**

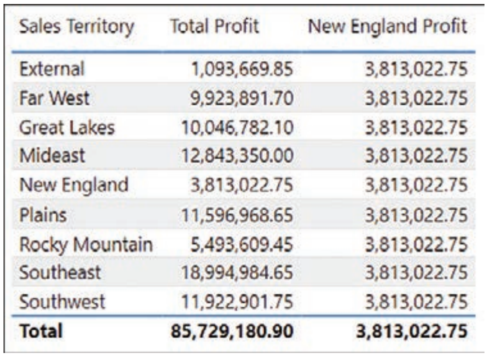
Updating filters

When you specify a filter such as `City[Sales Territory] = "New England"`, it's an abbreviated way that corresponds to the following filter:

```
FILTER(
    ALL(City[Sales Territory]),
    City[Sales Territory] = "New England"
)
```

By adding this filter, you are ignoring a filter by using `ALL`, and you're adding a filter at the same time. This allows you to filter for New England regardless of the selected sales territory.

If you create a table that shows Total Profit and New England Profit by Sales Territory, the result should look like Figure 2-21.



Sales Territory	Total Profit	New England Profit
External	1,093,669.85	3,813,022.75
Far West	9,923,891.70	3,813,022.75
Great Lakes	10,046,782.10	3,813,022.75
Mideast	12,843,350.00	3,813,022.75
New England	3,813,022.75	3,813,022.75
Plains	11,596,968.65	3,813,022.75
Rocky Mountain	5,493,609.45	3,813,022.75
Southeast	18,994,984.65	3,813,022.75
Southwest	11,922,901.75	3,813,022.75
Total	**85,729,180.90**	**3,813,022.75**

FIGURE 2-21 Total Profit and New England Profit by Sales Territory

When you have Sales Territory on rows, each row from the Total Profit column is filtered for a single sales territory and the Total row shows values for all sales territories. In contrast, by using the measure above in the New England Profit column, you are filtering regardless of the current sales territory, showing only the New England Profit.

Context transition

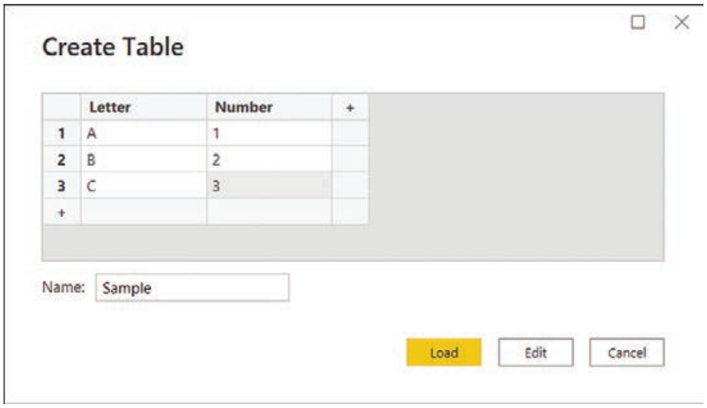
Another important function of `CALCULATE` is context transition, which refers to transitioning from row context to filter context.

In DAX, there are two evaluation contexts:

- **Row context** This context can be understood as "the current row." Row context is present in calculated columns and iterators. Iterators are functions that take a table and go row by row, evaluating an expression for each row. For example, `FILTER` is an iterator; it takes a table, and for each row, it evaluates a filter condition. Those rows that satisfy the condition are included in the result of `FILTER`.
- **Filter context** This context can be understood as "all applied filters." Filters can come from slicers, from the Filter pane, or by selecting a visual element. Filters can also be applied programmatically by using DAX.

To review context transition, let's create a sample table in the data model:

1. On the **Home** ribbon, select **Enter data**.
2. Enter **Sample** in the **Name** box.
3. Enter the data shown in Figure 2-22.


Create Table

	Letter	Number
1	A	1
2	B	2
3	C	3

Name: Sample

FIGURE 2-22 Entering data

4. Select **Load**.

Now that you have the table, you can add two calculated columns to it to see the effect of context transition:

1. Go to the **Data** view.
2. Select the **Sample** table in the **Fields** pane.

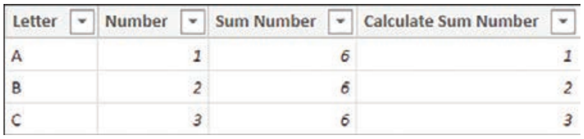
3. Create a calculated column with the following formula:

```
Sum Number = SUM('Sample'[Number])
```

4. Create another calculated column with the following formula:

```
Calculate Sum Number = CALCULATE(SUM('Sample'[Number]))
```

The result should look like Figure 2-23.



Letter	Number	Sum Number	Calculate Sum Number
A	1	6	1
B	2	6	2
C	3	6	3

FIGURE 2-23 Calculated columns in the Sample table

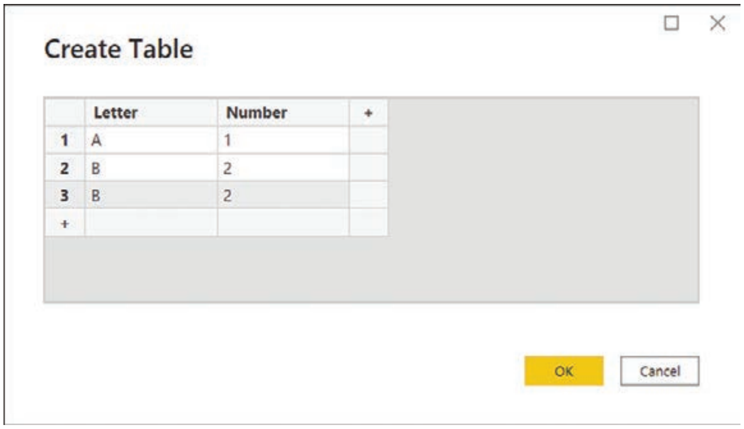
SUM, as an aggregation function, uses filter context. Because there are no filters in the data model—there are no visuals, and you're not adding any filters by using DAX—SUM aggregates the whole Number column, so the result in the Sum Number column is 6 regardless of the row.

On the other hand, the Calculate Sum Number column uses the same formula as Sum Number, but importantly has been wrapped in CALCULATE. CALCULATE automatically performs context transition, so the result is different from using the SUM function alone. Context transition takes all values from all other columns and uses them as filters. Therefore, for the first row, you aggregate the Number column, where:

- **Sample[Letter]** is **A**
- **Sample[Number]** is **1**
- **Sample[Sum Number]** is **6**

Where the sum of 1 is equal to 1, since there's only one such row that meets these filters, you get 1. Separately for row 2, the sum of 2 equals 2, and for row 3, the sum of 3 equals 3. Context transition can be made even clearer by modifying the Sample table slightly as follows:

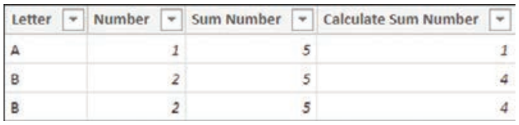
1. On the **Home** ribbon, select **Transform data**.
2. Select the **Sample** query.
3. Select the cog wheel in the **Source** step.
4. Change the third row to match the second row, as shown in Figure 2-24.


Create Table

	Letter	Number
1	A	1
2	B	2
3	B	2

FIGURE 2-24 Modified Sample table

5. Select **OK**.
6. On the **Home** ribbon of Power Query Editor, select **Close & Apply**.
7. If you now look at the Sample table in the **Data** view, the result will look like Figure 2-25.



Letter	Number	Sum Number	Calculate Sum Number
A	1	5	1
B	2	5	4
B	2	5	4

FIGURE 2-25 Sample table after update

Although the first row is calculated as you saw in the previous example, the second and third rows are now both showing 4. Intuitively, you could expect to see 2 and 2 in each row, though you're getting 4 and 4. This is because for each row, due to context transition triggered by CALCULATE, you're summing the Number column, where

- **Sample[Letter]** is **B**
- **Sample[Number]** is **2**
- **Sample[Sum Number]** is **5**

Because there are two such rows, you get 2 + 2 = 4 in both rows.

Implement Time Intelligence using DAX

It is common for business users to want to aggregate metrics—for example, revenue—across time, such as year-to-date revenue for a certain date, or prior-year revenue for the comparable period. Fortunately, DAX has a family of functions, referred to as Time Intelligence, that facilitate such calculations.

All Time Intelligence functions require a calendar table that has a date type column with unique values. If the date column is not part of a relationship, the calendar table must be marked as a date table, which can be done as follows:

1. Go to the **Report** or **Data** view.
2. Select the calendar table in the **Fields** pane.
3. On the **Table tools** ribbon, select **Mark as date table** > **Mark as date table**.
4. Select the date column from the **Date column** dropdown list.
5. Select **OK**.

NOTE **DIFFERENT CALENDARS**

The Time Intelligence functions in DAX only support the Gregorian calendar. If you use a different kind of calendar—such as a 4-4-5, which is common in retail, or a weekly calendar—then you'll need to use custom calculations. These types of calculations are out of the scope of this book.

Most Time Intelligence functions return tables that can be used as filters in CALCULATE. For example, you can use the DATESYTD function to calculate a year-to-date amount as follows:

```
Profit YTD =
CALCULATE(
    [Total Profit],
    DATESYTD('Date'[Date])
)
```

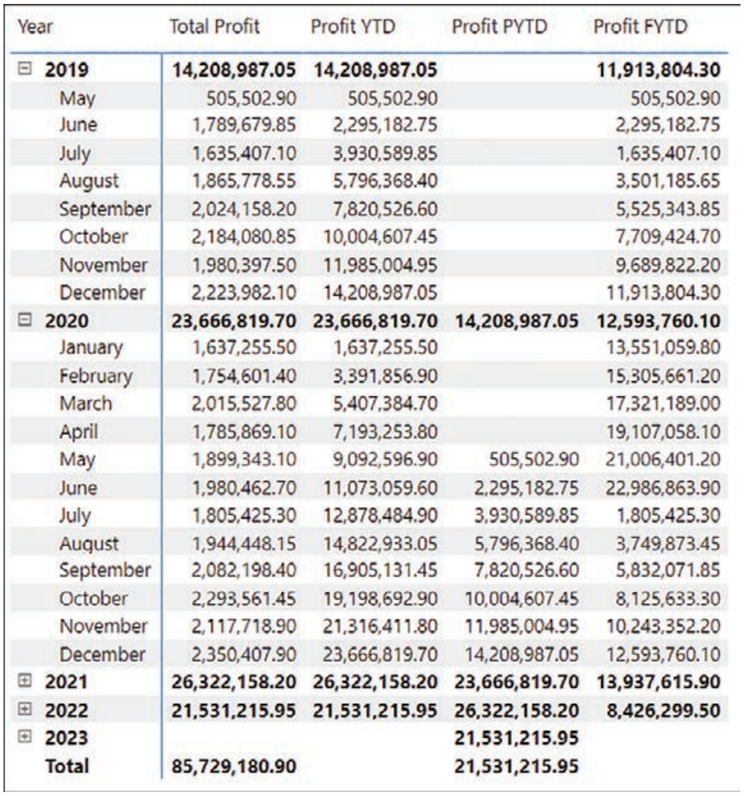
You can also combine Time Intelligence functions. For example, to calculate year-to-date profit for the previous year, use the following formula:

```
Profit PYTD =
CALCULATE(
    [Profit YTD],
    DATEADD('Date'[Date], -1, YEAR)
)
```

Some Time Intelligence functions, such as DATESYTD, can accommodate fiscal years. For example, if you had a fiscal year ending on June 30, you could calculate profit year-to-date for the fiscal year as follows:

```
Profit FYTD =
CALCULATE(
    [Total Profit],
    DATESYTD('Date'[Date], "30-6")
)
```

The Total Profit, Profit YTD, Profit PYTD, and Profit FYTD measures can be seen together in Figure 2-26.



Year	Total Profit	Profit YTD	Profit PYTD	Profit FYTD
⊟ **2019**	**14,208,987.05**	**14,208,987.05**		**11,913,804.30**
May	505,502.90	505,502.90		505,502.90
June	1,789,679.85	2,295,182.75		2,295,182.75
July	1,635,407.10	3,930,589.85		1,635,407.10
August	1,865,778.55	5,796,368.40		3,501,185.65
September	2,024,158.20	7,820,526.60		5,525,343.85
October	2,184,080.85	10,004,607.45		7,709,424.70
November	1,980,397.50	11,985,004.95		9,689,822.20
December	2,223,982.10	14,208,987.05		11,913,804.30
⊟ **2020**	**23,666,819.70**	**23,666,819.70**	**14,208,987.05**	**12,593,760.10**
January	1,637,255.50	1,637,255.50		13,551,059.80
February	1,754,601.40	3,391,856.90		15,305,661.20
March	2,015,527.80	5,407,384.70		17,321,189.00
April	1,785,869.10	7,193,253.80		19,107,058.10
May	1,899,343.10	9,092,596.90	505,502.90	21,006,401.20
June	1,980,462.70	11,073,059.60	2,295,182.75	22,986,863.90
July	1,805,425.30	12,878,484.90	3,930,589.85	1,805,425.30
August	1,944,448.15	14,822,933.05	5,796,368.40	3,749,873.45
September	2,082,198.40	16,905,131.45	7,820,526.60	5,832,071.85
October	2,293,561.45	19,198,692.90	10,004,607.45	8,125,633.30
November	2,117,718.90	21,316,411.80	11,985,004.95	10,243,352.20
December	2,350,407.90	23,666,819.70	14,208,987.05	12,593,760.10
⊞ **2021**	**26,322,158.20**	**26,322,158.20**	**23,666,819.70**	**13,937,615.90**
⊞ **2022**	**21,531,215.95**	**21,531,215.95**	**26,322,158.20**	**8,426,299.50**
⊞ **2023**			**21,531,215.95**	
Total	**85,729,180.90**		**21,531,215.95**	

FIGURE 2-26 Time Intelligence calculations

Notice how the Profit YTD measure shows the cumulative total profit within each year. The Profit PYTD measure shows the same values as Profit YTD one year before. Profit FYTD shows the cumulative total profit for fiscal years, resetting on July 1 of each year.

NEED MORE REVIEW? **TIME INTELLIGENCE IN DAX**

DAX includes over 30 Time Intelligence functions. Full details on all Time Intelligence functions are out of the scope of this book. For more details, see "Time intelligence functions" at *https://docs.microsoft.com/en-us/dax/time-intelligence-functions-dax*.

Replace implicit measures with explicit measures

It is sometimes possible to replace some numeric columns with measures, which can reduce the size of the data model. In our Wide World Importers example, there are several columns that could be replaced with measures.

For example, the Total Chiller Items and Total Dry Items columns in the Sale table show quantity of chiller and dry items, respectively. Essentially, these columns show filtered quantities depending on whether an item is a chiller or a dry item.

Before you replace the two columns with measures, create the following measure, which you'll reference and build upon later:

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

You can now create the following two measures and use them instead of columns:

```
Total Chiller Items (Measure) =
CALCULATE(
    [Total Quantity],
    'Stock Item'[Is Chiller Stock] = TRUE
)

Total Dry Items (Measure) =
CALCULATE(
    [Total Quantity],
    'Stock Item'[Is Chiller Stock] = FALSE
)
```

If you remove the Total Chiller Items and Total Dry Items columns from the model, you'll make it smaller and more efficient.

Another example of a column that can be replaced by a measure is Total Including Tax from the Sale table. Since Total Excluding Tax and Tax Amount added together equals Total Including Tax, you can use the following measure instead:

```
Total Including Tax (Measure) =
SUMX(
    Sale,
    Sale[Total Excluding Tax] + Sale[Tax Amount]
)
```

Again, once you have the measure, removing the Total Including Tax column would reduce the size of the data model.

Use basic statistical functions

As mentioned previously, it's best practice to create explicit measures even for basic calculations such as `SUM`, because you can build upon them to create more complex measures. You've already used `SUM` in our previous examples; here are several other basic statistical measures that are frequently used:

- `AVERAGE`
- `MEDIAN`
- `COUNT`
- `DISTINCTCOUNT`
- `MIN`
- `MAX`

All these functions take a column as a reference and produce a scalar value. In addition, every function except `DISTINCTCOUNT` has an equivalent iterator function with the X suffix—for instance, `SUMX` is the iterator counterpart of `SUM`. Iterators take two parameters: a table to iterate through, and an expression to evaluate for each row. The evaluated results are then aggregated according to the base function; for example, `SUMX` will sum the results. When you're learning the difference, it can be helpful to create sample tables similar to the examples shown earlier to visually compare the nuances of the different functions.

NEED MORE REVIEW? **STATISTICAL FUNCTIONS IN DAX**

There are over 60 statistical functions in DAX, and describing each one is out of the scope of this book. For an overview, see "Statistical functions" at *https://docs.microsoft.com/en-us/dax/statistical-functions-dax*.

Create semi-additive measures

In general, there are three kinds of measures:

- **Additive** These measures are aggregated by using the `SUM` function across any dimensions. A typical example is revenue, which can be added across different product categories, cities, and dates, as well as other dimensions. Revenue of all months within a year, when added together, equals the total year revenue.
- **Semi-additive** These measures can be added across some but not all dimensions. For example, inventory counts can be added across different product categories and cities, but not dates; if you had five units yesterday and two units today, that doesn't mean you'll have seven units tomorrow. On the other hand, if you have five units in Sydney and two units in Melbourne, this means you've got seven units in the two cities in total.
- **Non-additive** These measures cannot be added across any dimensions. For instance, you cannot add up the average price across any dimension, because the result would not make any practical sense. If the average sale price in Sydney was $4.50, and it was $3.50 in Melbourne, you cannot say that across both cities, the average price was $8.00 or even $4.00 because the number of units sold could be very different.

In this section, we focus on semi-additive measures. There are several ways to write a semi-additive measure, and the correct way for you depends on your business requirements. Let's say your business is interested in inventory counts, and you have the data model shown in Figure 2-27.

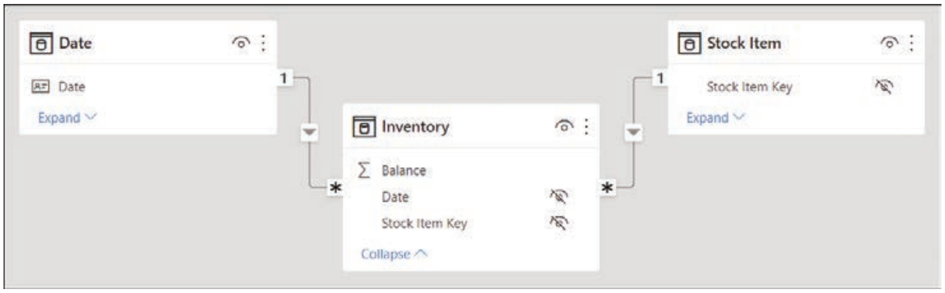


FIGURE 2-27 Inventory data model

If you have inventory figures for all dates of interest in your data, you can write the following measure:

```
Inventory Balance =
CALCULATE(
    SUM(Inventory[Balance]),
    LASTDATE('Date'[Date])
)
```

In addition to `LASTDATE` and its sister function `FIRSTDATE`, there are some DAX functions that can help you retrieve the opening or closing balance for different time periods:

- `OPENINGBALANCEMONTH`
- `OPENINGBALANCEQUARTER`
- `OPENINGBALANCEYEAR`
- `CLOSINGBALANCEMONTH`
- `CLOSINGBALANCEQUARTER`
- `CLOSINGBALANCEYEAR`

The functions that start with `CLOSING` evaluate an expression for the last date in the period, and the functions that start with `OPENING` evaluate an expression for one day before the first date in the period. This means that the opening balance for February 1 is the same as the closing balance for January 31.

For example, you can calculate the opening month balance for inventory as follows:

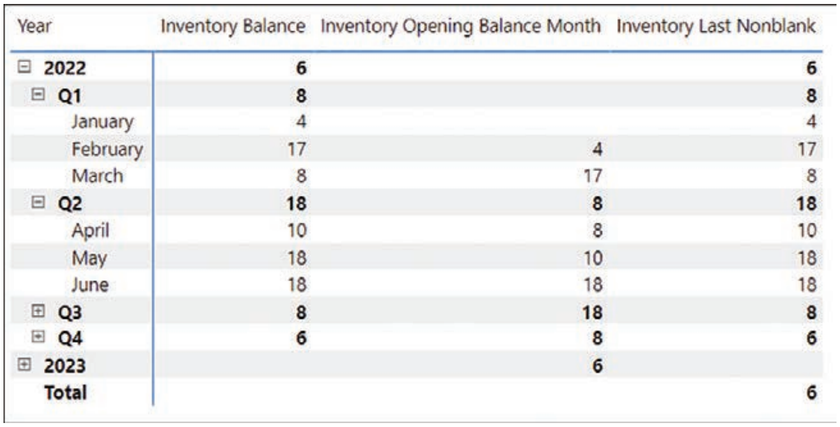
```
Inventory Opening Balance Month =
OPENINGBALANCEMONTH(
    SUM(Inventory[Balance]),
    'Date'[Date]
)
```

The date-based functions listed here only work if you have data for all dates of interest. For example, if you'd chosen to use `CLOSINGBALANCEMONTH` but your data ends on May 23, 2022, as is the case for sample data, you'd get a blank value for May 2022. For cases such as this, you can use `LASTNONBLANKVALUE` or `FIRSTNONBLANKVALUE` as shown here:

```
Inventory Last Nonblank =
LASTNONBLANKVALUE(
    'Date'[Date],
    SUM(Inventory[Balance])
)
```

This measure will show the latest available balance in the current context.

The Inventory Balance, Inventory Opening Balance Month, and Inventory Last Nonblank measures can be seen in Figure 2-28.



Year	Inventory Balance	Inventory Opening Balance Month	Inventory Last Nonblank
2022	**6**		**6**
Q1	**8**		**8**
January	4		4
February	17	4	17
March	8	17	8
Q2	**18**	**8**	**18**
April	10	8	10
May	18	10	18
June	18	18	18
Q3	**8**	**18**	**8**
Q4	**6**	**8**	**6**
2023		**6**	
Total			**6**

FIGURE 2-28 Inventory measures

Determining which calculation you should use depends on your business requirements—there is no single correct answer that applies to all scenarios. Missing data may mean there's no inventory, or it may mean that data isn't captured frequently enough, so the data modeler should understand the underlying data before writing the calculations to ensure the data isn't represented incorrectly.

Use quick measures

A measure in Power BI is a dynamic evaluation of a DAX query that will change in response to interactions with other visuals, enabling quick, meaningful exploration of your data. Creating efficient measures will be one of the smartest things you can do to build insightful reports. If you're new to DAX and writing measures, or you're wanting to perform quick analysis, you have the option of creating a quick measure. There are several ways to create a quick measure:

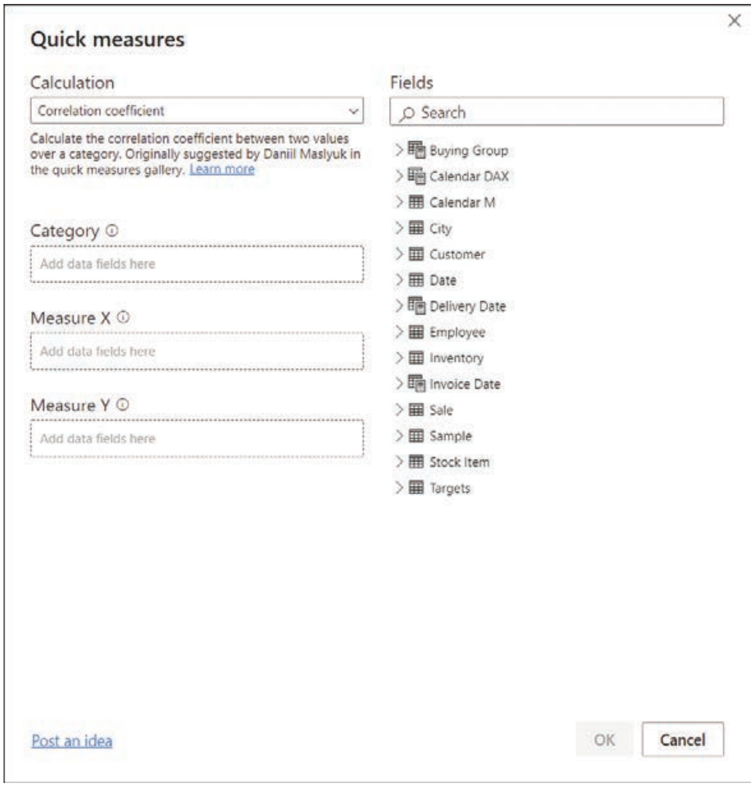
- Select **Quick measure** from the **Home** ribbon.
- Right-click or select the ellipsis next to a table or column in the **Fields** pane and select **New quick measure**. This method may prefill the quick measure form shown next.
- If you already use a field in a visual, select the dropdown arrow next to the field in the Values section and select **New quick measure**. This method also may prefill the quick measure form shown next. If possible, this will add the new quick measure to the existing visualization. You'll be able to use this measure in other visuals too.

The following calculations are available as quick measures:

- Aggregate per category
    - Average per category
    - Variance per category
    - Max per category

- Min per category
- Weighted average per category
- Filters
  - Filtered value
  - Difference from filtered value
  - Percentage difference from filtered value
  - Sales from new customers
- Time Intelligence
  - Year-to-date total
  - Quarter-to-date total
  - Month-to-date total
  - Year-over-year change
  - Quarter-over-quarter change
  - Month-over-month change
  - Rolling average
- Totals
  - Running total
  - Total for category (filters applied)
  - Total for category (filters not applied)
- Mathematical operations
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Percentage difference
  - Correlation coefficient
- Text
  - Star rating
  - Concatenated list of values

Each calculation has its own description and a list of field wells—you can see an example in Figure 2-29.



Calculate the correlation coefficient between two values over a category. Originally suggested by Daniil Maslyuk in the quick measures gallery. Learn more

FIGURE 2-29 Quick Measures dialog box

For example, by using quick measures, you can calculate average profit per employee for Wide World Importers as follows:

1. Ensure the **Sale** table is selected in the **Fields** pane.
2. Select **Quick measure** on the **Home** ribbon.
3. From the **Calculation** dropdown list, select **Average per category**.
4. Drag the **Profit** column from the **Sale** table to the **Base value** field well.
5. Drag the **Employee** column from the **Employee** table to the **Category** field well.
6. Select **OK**.

Once done, you can find the new measure called **Profit average per Employee** in the **Fields** pane.

NOTE **HOME TABLE**

Your new quick measure will be created in the last active table. If you're struggling to find the measure, you can use the search bar in the Fields pane.

To move a measure to a different table, select a measure in the **Fields** pane and select a new table in the **Home** table dropdown list on the **Measure Tools** ribbon.

If you select the new measure, you'll see its DAX formula:

```
Profit average per Employee =
AVERAGEX(
    KEEPFILTERS(VALUES('Employee'[Employee])),
    CALCULATE(SUM('Sale'[Profit]))
)
```

You can modify the formula, if needed. Reading the DAX can be a great way to learn how measures can be written.

NEED MORE REVIEW? **QUICK MEASURES**

For more information on quick measures, including limitations and considerations, see "Use quick measures for common calculations" at *https://docs.microsoft.com/en-us/power-bi/transform-model/desktop-quick-measures*.

Skill 2.4: Optimize model performance

Sometimes after you create the first version of your data model, you may realize that it doesn't perform well enough. Because of the way Power BI stores data, it may mean that your data model isn't performing as efficiently as it can. In this section, we review the skills necessary to optimize a model's performance and learn how you can identify measures, visuals, and relationships that are slow.

When working with imported data in Power BI, keep in mind that it's a *columnstore* database, which means that the number of distinct values in a column—also known as *cardinality*—usually plays a more important role than the number of rows. Therefore, one way to address poor performance is to reduce cardinality levels, which you can do by changing data types or summarizing data.

This skill covers how to:

- Remove unnecessary rows and columns
- Identify poorly performing measures, relationships, and visuals
- Reduce cardinality levels to improve performance

Remove unnecessary rows and columns

In Power BI, it's preferable to only load data that is necessary for reporting and then add more data later as required. In practice, you should disable loading of queries that aren't needed for reporting and filter the data to only the required rows and columns before loading into the model.

Remove unnecessary rows

Reducing the number of rows requires some filtering criteria, which can be based on attributes or dates.

For example, instead of loading all Wide World Importers data, you could load data for a specific sales territory if you're only interested in analyzing that specific sales territory. You can use parameters when filtering to make the process more manageable; this approach will also make it possible to change filters once the dataset is published to the Power BI service.

You can also filter by dates and only load some recent data in case you're not interested in historical data. In addition to parameters, you can apply relative date filters, such as "is in the previous 2 years."

Filtering rows after you create reports won't break any visuals in the existing reports.

Remove unnecessary columns

Columns in a data model usually serve at least one of two purposes: they could be used to support visuals or calculations, or both. It's preferable to not load columns that aren't used for any purpose, especially if they've got a high number of distinct values.

Some data warehouses include primary keys for fact tables. Although that may be useful for data audit purposes, you should remove primary keys from Power BI data models because they have a unique value for every row, and fact tables can be long. Primary keys of fact tables can occupy over 50% of data model size without bringing any benefit. In the Wide World Importers example, removing the Sale Key column from the Sale table reduces the file size by 43%.

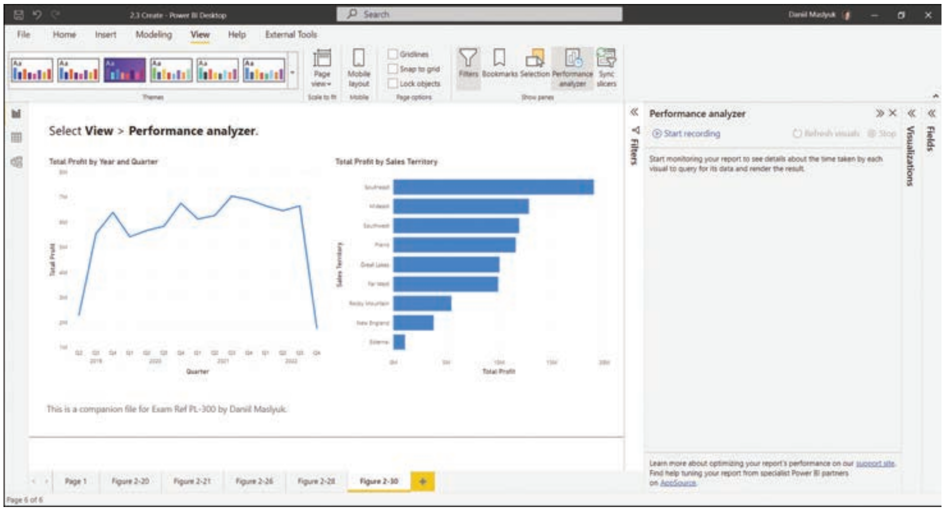
If you need to count the number of rows in a fact table, it's more efficient to use `COUNTROWS` than `DISTINCTCOUNT` of the primary key column.

Removing columns that are used in visuals or calculations is going to break existing reports or even the dataset. You can use the Remove Other Columns functionality in Power Query Editor to have a step to refer to if you need to add a column to your model later. This step will also prevent columns added to the dataset from being automatically brought into your model, such as a new column added to a SQL table by a database administrator.

Identify poorly performing measures, relationships, and visuals

Sometimes you may notice that the report performance is not optimal. Power BI Desktop has a feature called Performance Analyzer, which you can use to trace the slow-performing visuals and to see the DAX queries behind them.

To turn on Performance Analyzer, go to the **Report** view and select **View** > **Performance analyzer**. This opens the Performance Analyzer pane shown in Figure 2-30.



Select **View** > **Performance analyzer.**

Start monitoring your report to see details about the time taken by each visual to query for its data and render the result.

This is a companion file for Exam Ref PL-300 by Daniil Maslyuk.

Learn more about optimizing your report's performance on our support site. Find help tuning your report from specialist Power BI partners on AppSource.

FIGURE 2-30 Performance Analyzer

Performance Analyzer works by recording traces, and it then shows you how long each visual took to render. To start recording traces, select **Start recording**. After that, you need to perform some actions, such as applying filters, that will recalculate the visuals, or you can select **Refresh visuals** to refresh the visuals as they are. You'll then see the rendering duration for each visual.

To identify the slowest visuals, you can sort visuals in the Performance Analyzer pane by selecting the arrow next to **Duration (ms)**.

Each visual that contains data has a DAX query behind it, which you can copy by expanding the line of the visual in the Performance Analyzer pane and selecting **Copy query**. You can analyze the query further in DAX Studio, for example. It's also possible to export all traces by selecting **Export**.

To clear the Performance Analyzer pane, select **Clear**. Once you're done recording traces, select **Stop**.

Reduce cardinality levels to improve performance

Power BI employs several compression mechanisms to reduce the size of data, the details of which are outside the scope of this book. One way to decrease the data size, which we cover next, is by reducing the cardinality of columns by changing data types or the default summarization.

Changing data types

In Power BI, two data types can be used for decimal numbers:

- **Decimal number** Can store more than four decimal places
- **Fixed decimal number** Can only store up to four decimal places

If your data contains more than four decimal places for some values and you don't need that level of precision, you should change the data type to Fixed Decimal Number to save space.

Another way to change the cardinality levels is to split decimal number columns into pairs of whole numbers and decimal numbers, which should be done as close to the data source as possible. Whole numbers can be of any range, whereas decimal numbers should be between 0 and 1. These two columns can then be aggregated by using `SUMX` in the following fashion:

```
Full number =
SUMX(
    'Fact table',
    'Fact table'[Whole number] + 'Fact table'[Decimal number]
)
```

Although you'll get two columns instead of one, in many cases you'll see improvements in cardinality levels and, as a result, a decrease in the data model size. For the same reasons, in Power BI it's best practice to split Date/Time columns with Time components into two: Date and Time. This is because you are increasing the number of duplicates in each column, and therefore the column is more efficiently stored in memory.

Some text columns, such as invoice numbers that are stored as text, can also sometimes be reduced in size. For example, if your fact table contains a column with invoice numbers, which always have the INV prefix and eight numbers that follow it, such as INV01234567, you can remove the INV prefix and change the data type of numbers from Text to Whole Number. If the prefix is inconsistent, you can split it and move it to a different column. This is because storing whole numbers is usually more efficient than storing text.

EXAM TIP

You should be able to recognize models that would benefit from splitting columns and selecting different data types.

Summarizing data

If your source data provides a level of detail that's not required by reporting, then you may want to consider summarizing your data to reduce cardinality.

For example, if the source data contains daily sales information but you only report monthly values, you may want to summarize your sales data to be at the month level instead of the day level. This approach will reduce the size of your model dramatically, though it will make the reporting of daily data impossible.

It's preferable to summarize your data as close to the data source as possible. Power Query also allows you to summarize data by using the **Group By** functionality on the **Transform** ribbon.

Data summarization involves a trade-off between data model size and the available level of detail; whether you should summarize data depends on your business requirements.

NEED MORE REVIEW? **DATA REDUCTION TECHNIQUES**

All possible data reduction techniques are outside the scope of this book. For more examples of reducing your data when working with imported data, see "Data reduction techniques for Import modeling" at *https://docs.microsoft.com/en-us/power-bi/guidance/import-modeling-data-reduction*.

Chapter summary

- Power BI supports various types of schemas: flat (fully denormalized), star, and snowflake. The preferred schema for Power BI is the star schema.
- You can configure various column and table properties in the Model view.
- In some cases, it may be preferable to define role-playing dimensions, which allow you to use a single dimension to filter one fact table by using different keys in the table.
- Power BI supports the following three cardinality types for relationships: one-to-one, one-to-many, and many-to-many. For one-to-one relationships, the cross-filter direction is always Both (each table filters the other). One-to-many dimensions can have their cross-filter direction be set to either Single (the one side filters the many side) or Both. You choose the cross-filter direction of many-to-many relationships depending on your business requirements. Relationships whose cross-filter direction is set to Both are also known as bidirectional relationships.
- For bidirectional relationships, security filters won't flow in both directions automatically, though you can configure that behavior in the relationship properties.
- For best performance, look carefully at the storage mode of each table, the cardinality and cross-filter direction of relationships, and the cardinality of columns (the number of distinct values).
- Besides measures, you can use DAX to create calculated tables and calculated columns in Power BI.
- You can create a common table in Power BI by using Power Query or DAX, or you can load it from a data source.
- Power BI supports the creation of hierarchies, which can be useful to make models more user-friendly, though they have no technical advantages over several fields being used together in a visual without being combined in a hierarchy.
- You can secure your data model by using row-level security, which can use static DAX filters on one or more tables, or dynamic row-level security that considers which user is viewing the report.

- Power BI allows you to use natural language queries by using the Q&A visual. You can add synonyms to your data model to make Q&A work better.
- `CALCULATE` is one of the most important functions in DAX, and you can use it to manipulate filters. More specifically, you can add, ignore, and update filters. `CALCULATE` is also used for context transition.
- The Time Intelligence family of DAX functions allows you to aggregate values across time; for instance, you can use `DATESYTD` to calculate year-to-date values, or you can use `DATEADD` to calculate a value during the same period last year. There are also functions that allow you to create semi-additive measures, such as `OPENINGBALANCEMONTH`.
- Power BI has a feature called Quick Measures, which allows you to define calculations without writing any DAX code.
- In some cases, it may be preferable to replace numeric columns with measures to reduce the size of the data model.
- In general, you should only load data that is necessary for analysis by removing columns or filtering rows in Power Query, especially for primary keys of fact tables.
- Performance Analyzer in Power BI can be useful to identify performance bottlenecks.
- You can improve the cardinality of columns by selecting appropriate data types, as well as summarizing data.

Thought experiment

In this thought experiment, demonstrate your skills and knowledge of the topics covered in this chapter. You can find the answers in the section that follows.

You are a data analyst at Contoso responsible for creating Power BI reports.

Management has requested a report based on the historical data available. Based on background information and business requirements, answer the following questions:

1. A data model has a fact table that has over 15 million rows. There is a date/time column called DateTime, which contains both date and time. You need to reduce the size of the data model. Your solution must preserve as much of the original data as possible. Which solution should you implement?
   - A. Change the data type of the DateTime column to Text.
   - B. Clean the DateTime column.
   - C. Split the DateTime column into two separate columns: one column that contains dates, and one column that contains the time portion.
   - D. Change the data type of the DateTime column to Date.

2. You create a visual that is supposed to show revenue by year. You use the Year column from the Calendar table and the Revenue measure from the Sale table. The formula of the Revenue measure is as follows:

   ```
   Revenue = SUM(Sale[Total Including Tax])
   ```

   The result is shown in Figure 2-31. After checking data, you can see that in 2021, revenue was $60 million. How can you fix the visual? The solution must use the minimum amount of DAX and ensure that the Calendar table can be used with other fact tables. The solution must also take into account that you may be interested in analyzing other measures based on the Sale table.

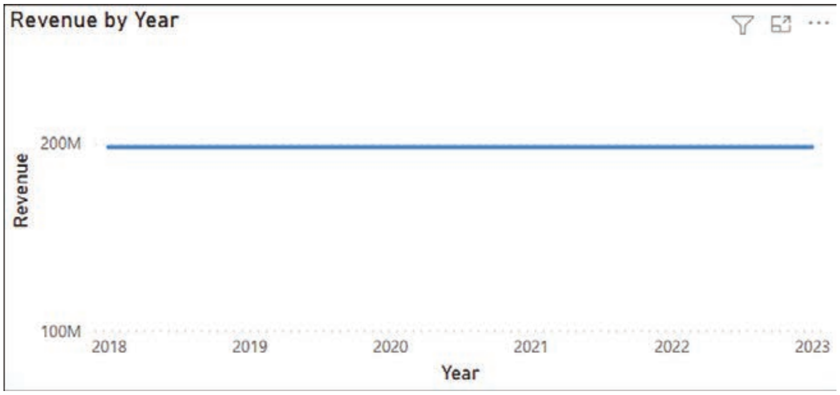


FIGURE 2-31 Revenue by Year

   A. Use the TREATAS function in DAX.

   B. Create an active relationship between the Calendar and the Sale tables.

   C. Merge the Sale and Calendar tables.

   D. Create a calculated table that calculates revenue for each year.

3. There are two roles in a data model: CentralRegion, which filters the Region table to only show the Central region, and AppliancesDepartment, which filters the Department table to only show the Appliances department. A user is a member of both roles. What will they see in a sales report?

   A. Sales from the Central region or the Appliances department; they will see all departments in the Central region and all regions in the Appliances department.

   B. Sales from the Central region and the Appliances department; they will only see the Appliances department within the Central region.

   C. Only one role will be applied, whichever was configured first.

   D. The user will see an error message.

4. Your Date table currently consists only of one column called Date, which contains dates. You need to add a column to the Date table that shows month and year in the MMMM YYYY format, e.g., May 2022. What should you do? Your solution must require the minimum amount of effort and storage, and the solution must ensure that the values are sorted chronologically.
   - A. Create a calculated column that uses the `FORMAT` function.
   - B. Create a calculated column that uses the `EOMONTH` function format as MMMM YYYY.
   - C. Duplicate the Date column and apply a custom format string.
   - D. Create a new calculated table called Date – MMMM YYYY and format as MMMM YYYY.
5. You need to write a measure that calculates the monthly balance. Which formula should you use?
   - A. `CALCULATE(SUM(Inventory[Balance]), ENDOFMONTH('Date'[Date]))`
   - B. `CALCULATE(SUM(Inventory[Balance]), MAX(Inventory[Date]))`
   - C. `CALCULATE(SUM(Inventory[Balance]), DATESMTD(Inventory[Date]))`
   - D. `CALCULATE(MAX(Inventory[Balance]), LASTDATE(Inventory[Date]))`
6. You inherit a Power BI data model that contains several tables, one of which has many calculated columns that all use the `RELATED` function. You would like to reduce the size of the model. What should you do?
   - A. Append tables.
   - B. Merge tables.
   - C. Separate tables into several data models.
   - D. Hide unused columns.
7. You created a sales report and enabled row-level security on it. There are multiple roles, each filtering the dataset to one department. Each role has a corresponding Active Directory group. The report is primarily used by sales managers, each of whom can view their department only. One sales manager has recently moved from one department to another. What should you do?
   - A. Update role membership in the Power BI service.
   - B. Change roles in Power BI Desktop.
   - C. Raise a request to remove the user from their old Active Directory group and add them to the new one.

Thought experiment answers

1. The answer is **C**. Splitting a date/time column into a date and a time column will keep the original data and reduces the number of distinct values in columns, resulting in a smaller data model. Changing the data type to text, as answer A suggests, won't change the number of distinct values, and therefore won't reduce the size of the file. Answer B, cleaning a column, removes nonprintable characters, which does not reduce the number of distinct values when applied to a column of type date/time. If you change the data type of the DateTime column to date in accordance to answer D, you'll see the reduction in the size of the model, and you'll lose the time portion, which goes against the requirements.
2. The answer is **B**. Creating an active physical relationship is the best solution because it requires no DAX, allows the Calendar table to be used with tables other than Sale, and you can use other measures from the Sale table together with fields from the Calendar table. While using the `TREATAS` function that answer A suggests may work, it requires using unnecessarily complex DAX, especially considering that you may be interested in analyzing measures other than Revenue. The merged table from answer C will either prevent the Calendar table from being used with other tables, or it'll duplicate data from the Calendar table unnecessarily. Answer D would fix the graph, but it won't solve the problem when other measures are analyzed by year.
3. The answer is **A**. Power BI supports multiple roles for a single user, and they are combined by using the union logic, so the user will see all departments within the Central region and all regions within the Appliances department.
4. The answer is **B**. If you use the `EOMONTH` function, you'll get a calculated column that contains the end-of-month dates, and you can then apply a custom format string to show the values in the desirable format. Since the values will still be of type date, they will be automatically sorted in the correct order. If we use the `FORMAT` function, you'll get the values in the format that you want, though they will be text values that require a sorting column—otherwise the values will be sorted alphabetically. A sorting column will use extra storage. If you apply a custom format string to a duplicated Date column, the values will look the way you want, though underneath they will still be dates, so there will be more than one value for each month-year combination. By creating a new calculated table as in answer D, you are increasing the data model size and adding unnecessary complexity.
5. The answer is **A**. `SUM` will correctly aggregate inventory balances for all dimensions except Date, since `ENDOFMONTH` will pick the last date of month to show the monthly balance. In answer B, MAX is used as a filter in `CALCULATE`, and it won't work because it returns a scalar value instead of a table. Answer C will provide incorrect values in cases where you have daily or weekly inventory balances. Answer D won't aggregate the balances correctly, since it will pick the maximum balance across the available values.
6. The answer is **B**. Using the `RELATED` function in a calculated column often means duplicating data. If columns need to be in the same table—for example, to build a

hierarchy—then it may be preferable to merge tables into one. Appending tables, as in answer A, would stack them vertically and wouldn't provide the desired output. Separating tables in several data models, as answer C suggests, will reduce the size of the model, though it won't allow you to have the same data. Hiding unused columns as suggested in answer D doesn't reduce the size of the model.

7. The answer is **C**. Since the security is managed by Active Directory groups, the user should be removed from their old department security group and added to their new department security group. If you add them to a new role in the Power BI service without affecting their group membership, as suggested in answer A, they'll see sales of both old and new departments. Changing roles in Power BI Desktop (answer B) won't help because role membership is managed outside of Power BI Desktop.

This page intentionally left blank

Index