[Modeling Data in Power BI](https://app.pluralsight.com/library/courses/power-bi-data-modeling)

# Course Overview

## Course Overview

Hello. I'm Stacia Varga, and welcome to the Modeling Data in Power BI course. I work with data all the time as a consultant, helping my clients use and understand their data and as an instructor and author, teaching people how to build data solutions and work with data effectively.

The whole point of using Power BI is to enable the exploration and visualization of data whether you're doing this for yourself or for others.

Because data in its raw form is not often suitable for this purpose, you need to create a data model that refines and enriches this data to make it more useful and understandable. There are a lot of steps involved in data modeling for Power BI, and Power BI is certainly packed with lots of features for data modeling. In this course, I'll show you how to start with a set of files and incrementally refine and enhance that data to produce a better data model. We'll start by loading data into a model and then shaping and cleansing the data, and we'll learn how the structure of the data model affects how the data displays and see how incremental changes improve the model structure, not only for reporting, but also to take advantage of special Power BI features for geospatial analysis and natural language queries.

We'll also learn how to enhance the model with various types of calculations that transform raw data into information.

By the end of this course, you'll know how to apply a variety of techniques in Power BI to model your own data. From here, you can continue learning more about Power BI by viewing other Pluralsight courses on Power Query, DAX, and several other courses covering different aspects of using Power BI. Although you don't need to know any programming or data query languages before you start this course, some familiarity with basic data analysis is helpful. Let's get started on this Power BI journey with the Modeling Data in Power BI course here, at Pluralsight.

# Using the Query Editor to Prepare a Power BI Data Model

## Overview

Hi. I'm Stacia Varga, and welcome to the Modeling Data in Power BI course. Whenever you need to build a new data model in Power BI, the very first step is to get your data and then figure out how to fix any problems that might exist in that data. That's our focus in this module, and I'm going to show you how to use the Query Editor to get started on a new model in Power BI.

The Power BI Query Editor is used for several important tasks in data modeling, two of which I'll describe throughout this module.

* First, it's the tool you use to load data into Power BI. In most cases, you're literally making a copy of the data and putting it into a specific structure in memory that Power BI uses to display data in a chart or a table or some other visualization in your report.
* In other cases, such as when you use the DirectQuery or Connect Live options for a data connection, you're telling Power BI where to find the data and how to query it when you want to refresh your report.

Throughout this course, I'll be using a specific set of data for demonstrations. So I'll first provide you with a big picture overview of the different types of data that we're going to use and then show you how to load all that data into Power BI. Whenever you load data into Power BI by using the option to copy the data, you usually need to do some data wrangling. In other words, you need to shape the data in some way. This task could be as simple as defining a filter to load only a subset of data into a model, or it can require a variety of transformations to change the data into a format that's better suited for reporting. In this module, we'll look at the more common types of transformations that you can use in the Query Editor to shape your data.

## Loading Data

The Query Editor is the entrance point for the data that you want to use in your Power BI model.   
The number of supported data sources is really quite extensive. And from time to time, you'll find that the latest version has added the even more data sources.

In general though, there are six categories of data sources that Power BI can access.

* Databases are a popular type of data source. Power BI can, of course, copy data from databases that are stored on-premises in your organization. You then need to refresh this copy of the data periodically. But in some cases, you can create a very specific type of connection to work with the data directly.
* Another option is to get data from files that are stored somewhere, such as on your computer or your company's network.
* You can also get data from external sources by using a web URL or an OData feed and other kinds of sources.
* And, of course, data that's stored in Azure is easily accessible.
* And Power BI can also connect to many different kinds of data available through Software as a Service options.
* And last, if you or someone in your organization has saved a Power BI dataset or dataflow to a workspace in the online Power BI service, you can access that dataset or dataflow in the Power BI desktop.

A dataset is the official name for the Power BI model when you publish it online, or it can even get created online directly.

By contrast, a dataflow is only created online, and it's basically a collection of steps to load, shape, and cleanse your data that gets created and saved in the Power BI service where it can be reused in different models by different people without having to recreate those steps in each model.

Exploring Power BI Data Sources

Here I have opened up the Power BI Desktop application, which you can download from the Microsoft site. I have the link available for you in the setup instructions that come along with the exercise files for this course. On the Help tab of the ribbon here, there's a lot of information available for learning more about Power BI. I want to show you that as I make this recording, I'm using the December 2018 version of Power BI Desktop. If you're using a different version that's earlier or later than December 2018, then you might see a few different things in Power BI Desktop. So normally when you open Power BI Desktop, you're looking at a blank report, and we have no data loaded in the model. The way that we get data in is to click Get Data. And when you do this, it displays a variety of data sources that Microsoft has determined to be the most common. If we want to see a complete list, we just click More, and now we get a dialog box that shows us all of the possibilities. In fact, the first thing we see is the entire list. So we could scroll through this whole list and pick something that is the source of our data, or we can zero in on a particular type, for example databases. We could use relational databases, like SQL Server, or IBM DB2, or multidimensional databases from Analysis Services, or Essbase. The SQL Server Analysis Services database connection here also supports tabular models. We can access big data sources using Impala or Google Big Query and lots of different kinds here. Now if you were to connect to SQL Server as an example, you need to provide a server name at minimum, and optionally you could provide a database. And notice here that you have the option to specify a data connectivity mode. This is not available for every type of data connection that you have, and I'll explain the difference later in this module. But after you provide a server and a database and click OK, then depending on what level of information you provided here, you would see another dialog where you would see either a list of databases if you had provided just the server name or a list of tables that are contained in the database that you specified. And then you could select then by selecting their checkboxes, and then Power BI copies that data into the model in your application. You could also expand Advanced options here and put in a specific SQL statement if you want to preprocess the data before bringing it into Power BI. Of course you have other options. You can use different kinds of file types, such as Excel or data in text or comma- separated files, XML documents, or JSON, or a SharePoint folder. Or you can even point to a folder to get metadata about the folder contents and then bring in groups of files. You can get data from a variety of sources online. Maybe you're using Dynamics or Salesforce among all kinds of other Software as a Service providers here. Or maybe you have a URL where you can download information, or you have access to SharePoint lists, maybe an OData feed. Lots of possibilities here. Then we have the Azure sources where we have Azure SQL Database and Azure SQL Data Warehouse and other types of data storage that Power BI supports and then also Power BI data flows and any datasets that have been published to the Power BI service. So that's how you can check to see what you have available in terms of data that you can use for your Power BI model.

Databases: DirectQuery and Analysis Services Connect Live

There's a data connectivity mode that allows you to use DirectQuery instead of import. This option only works with some databases, and you'll know which ones they are when you see that data connectivity mode option. And remember also as with anything I describe in this course, Power BI changes frequently. So even if a database does not support DirectQuery today, it's possible that could change in the future. I'll include a link at the end of the slides for this module so that you can check out the current list of supported sources. Now with DirectQuery, the data doesn't move out of the source database. Instead Power BI queries the database when you view the report, so the data is always current. In some cases, you will need to have Enterprise Gateway installed by your IT department to work with these sources if you're going to publish your reports to the Power BI service. Probably the biggest drawback of using DirectQuery is that your model can only use data from the same database. You are not going to be able to mix and match data sources like you can when you're importing data into the model. And you will not be able to add calculated tables to your model either. You'll learn more about those later when we work with dates later in this course. On the other hand, you can now add calculated columns to a model as long as the calculation operates only on data in the same row, like when you want to concatenate a first and a last name. Now despite these limitations, DirectQuery can be a great option when your dataset is really large or when you need near real-time data. I say near real time because you will have to refresh your dataset manually to get the latest data. It does not happen automatically. We have a different kind of data connectivity option for Analysis Services, and it's called a live connection. It's like DirectQuery because the model stores only the connection, and the data stays in the source. It doesn't get copied over into Power BI. If you're connecting to an Analysis Services model, then you can work with the data directly without making any changes to the model inside of Power BI. In fact, it won't let you. But just like DirectQuery, you are not going to be able to combine Analysis Services data with other sources when you're using this live connection mode. Using this connection, you'll be able to connect to tabular models, as well as multidimensional models. You do need to have an Enterprise Gateway installed by your IT department if you plan to publish reports to the Power BI service so that Power BI in the cloud can reach into your on-premises environment.

Demonstration Data Overview

Now that we've covered the kinds of data that we can use in Power BI data model, we need to get some data. Before I show you how to load that data, let me explain what kind of data I'm going to be using in the demonstrations throughout this course. I found a few different types of publically available datasets about movies. First, I have data that I downloaded from IMDB.com. That's a website for lots and lots of information about movies, even more information than they offer for download. I'll share the link that I used in the resources slide at the end of this module if you want the full dataset, but it's a pretty big dataset to work with. Even though Power BI can handle the volume, it really choked up my computer to load it all in and will probably do the same for yours. So I did some preprocessing to filter the data down to movies released since 1980. Notice these are TSV files. It's like a CSV file, which is comma separated, but it's tab separated instead, and that's how IMDB releases the data in its dataset. They also have a principals dataset, and that includes information about who participated in a movie, meaning the actors and actresses and directors and producers and cinematographers and all kinds of people. But I filtered it down a little bit to include only those principals related to the movies made since 1980. But that file doesn't include the names of those people. That's in another file, the names.tsv file. And that includes their names and some other basic information, such as when they were born and if they died. And I actually broke that out into a couple of smaller files just to give you an example of how we could combine multiple files together when we load data as long as they are similarly structured. So I went to look to see what other kinds of information I could get to combine with this IMDB data because that's really the power of Power BI, the ability to mix and match various data sources that have something in common, but they come from different places, and they're structured differently. So I found a dataset called MovieLens, and it's put together by an organization called GroupLens that is a bunch of researches at the University of Minnesota that prepared this dataset for machine learning. And I explained what I wanted to do with the data, and they gave me permission to use it in this Pluralsight course as long as you download it directly yourself if you want to follow along with the demonstrations. And again, I'll provide the link for this in the resources slide at the end of this module. In that case, we download one big file and then use an extraction tool, like 7-Zip for example, to pull out the individual CSV files that it contains. It has a movies.csv, which has the basic movie information, and then there's a links.csv, which gives me the IMDB identifier for each movie listed in the movies.csv file. Then there's a ratings.csv file that has ratings information that was assigned to the various movies by users of the MovieLens service. And there's also tags, and these are tags that were applied to movies by the MovieLens users. So basically this is a user-driver categorization of the movies that might be interesting to analyze. Now these two datasets give us plenty to explore, but I found one more dataset that I thought would be good to include. This data is based on material created by a Stanford University professor to teach students how to work with database technologies, but it's accessible online for anyone to use. I've added a link to the dataset documentation on the resources slide if you want to learn more about it. Now it's definitely not completely clean data, and it's not even as complete as the other dataset. But that's often what we run into with data in the real world, so let's see what we can do with it. In our data model, I'm only going to use the main movie information that's available on a single page as an XML document, and I'm going to parse that XML to keep just some of the movie data including its name and the date released and locations featured and also a code for awards won by the movie if any. And to translate those codes, I'll get the codes and award information from a table that's available on a different web page associated with this database. The common thread across these three sets of data is movies, each of which provide some slightly different information. But we can combine the datasets to get some insights that are not possible when looking at the datasets individually.

Loading the Demonstration Data: Single TSV Files and Groups of Files

Now we're ready to get some data and load it into our Power BI data model. We'll start with the IMDB data, which is a set of TSV files. So I'll go to Get Data, and I can choose Text/CSV even though it's a TSV file. So I've extracted the exercise files to my C drive, and I have a folder containing the IMDB folder and the MovieLens folder. So let's start with IMDB, and it looks like nothing's there, but that's because we have a filter here. I'm looking for just TXT, CSV, or PRN files. When I change this to All Files, we can see the various TSV files that I have. So the main movies file is titles.tsv, so we'll click Open. Power BI connects to the data, and then it shows you a preview of the data so you can confirm that this is indeed the data that you want to import into your data model. It does detect that it's tab delimited, and it shows you the various columns and the data. So we have the option to load the data or edit the data. And the difference is if you choose load, which is what I'm going to do now, it just copies the data into the Power BI data model, and you don't have to make any other changes; whereas maybe you want to change some column names or eliminate columns or eliminate rows, and that's when you would choose Edit. We'll do all those things a little bit later, but first I just want to get everything loaded. So I'm just going to choose the Load option. So now Power BI is literally copying the data from that TSV file, which stays on disk. And now it loads that data into memory. So any time we open our Power BI Desktop and open a particular file, that data that's contained in that data model is loaded into memory. And now we can see that that process is complete. It's loaded as a table into the data model, and then these are all the various columns, which are labeled as fields here in the data model. So that's one file. Let's go get the others. I'll get principals, load it in, and we'll take a closer look at this data when we go through the editing phase. Now for my last set of data from IMDB, I actually have two files, files that have the names that start with A through K and a file for names that start from L to Z just to demonstrate how we can pull in two different sets of files. Now if I use just the Get Data for the text CSV, I can only pull in one file at a time. So we have another option when we want to pull in multiple files. As long as they are structured in the same way, this technique will work. So this time I'm going to point to a folder and navigate to that folder. And here we can see a list of the files that are located in that location. So if the only files in this particular path were the ones that I wanted to combine, I could do a combine and load. But because I need to filter out the principals and titles, I'm going to choose the Combine and Edit option here. And Power BI says okay, you have multiple files here. I need to set up a structure for this particular query. And it defaults to the first file, but we can point to any of the files contained in that path. So the first one is fine. It's one of the names, so we're good with this. Click OK. And it's just using this as a template basically to say this is the structure that I'm expected, and it pulls that data into the model. But I need to go to the Query Editor. It's just given this query the name of the folder it was in, IMDB, and here we can see various data. And it pulled in some extra information here that we'll review a little bit later to explain what this means. But essentially what I need to do is modify this query so that we include only the files that are relating to the names. So if we look through the steps here, we have Source, which lists all of the files in that folder, and it filters out hidden files, but I don't know that there's any. And then it invokes this custom function. We'll talk about custom functions later. As we step through this process, we can see that various transformations are applied. So this is what we'll be learning about in this module. So the main thing that I want to do here to fix up this particular query is to say that I do not want these two CSV files. So I can apply a filter here, clear the Select All, choose the two that I want to keep, click OK. And when we go to the very last step here, we would see that there is data only from the two names files. So that's the limit of the editing that we'll do for this right now.

Loading the Demonstration Data: CSV Files

In the Query Editor, I can continue to bring in data sources. So my next set of files are, in fact, CSV files. This is coming from the MovieLens database, which you will need to download and extract separately. Instructions are found in the setup document in the exercise files that you can download for this course, and that was based on the permissions that I received from the University of Minnesota in order to use this data. I only want some of this data. The core file here is the movies.csv, and you can see it just has movie identifiers, title, and genre. And we'll pick up other files. I want links, and I want ratings, and I want tags. So each one of these files becomes a query in our Query Editor. So now I have data copied into memory, but actually technically it's just preview information, so it doesn't have all of the data yet. I have to do some additional steps to get all of the data, but I can see preview of the data so that I can start working with it and fine-tuning these queries to get it exactly the way that I want it.

Loading the Demonstration Data: XML and HTML Table

Next, we need to get the Stanford Movies database. This is available as an XML file, which we can access directly from a URL. So we'll use the web connector here, and the URL I'm going to paste in. This is available in the snippets file found in your exercise files for this module. But here's the URL that we're going to use, and we click OK. And Power BI detects some data out there labeled directorfilms. So this is XML structure where the top- level node is named directorfilms. And so it'll come up with a preview of this, and we can see that there's some tables embedded under these nodes, directors and films. That's as much as we can see. At this point, we'll just click OK and bring that into our model. So that's what XML data looks like. Not very useful at this point, but we're going to learn how to deal with this later in this module. One last step is to get some awards data related to these films, and that's accessible from a table on a web page. So let's take a quick peek at that web page. Here's the web page, and you can see that we have some awards information with codes on the left, organization, country, and so forth, various columns here. So Power BI is going to be able to read this page and turn it into data for us. So this time we'll do New Source, and again we'll use the web connector, paste in that URL, which, again, is available in the snippets if you want to copy and paste this. And we just want the Award Givers. Here we can see the data as turned into columns. We can see the Table View. We want to see the Web View. And now we click OK, and all of our data is now ready for us to start shaping it. So let's talk about shaping data next. Before we do that though, I'm going to go ahead and save this. And I'll choose the Apply later option because the Apply option is going to go out and read all of those data sources and load it into memory, and I'm not quite ready for that to happen yet. So I'll choose the Apply later, and I'll save this file.

Saving the Demonstration Data and Counting Loaded Rows

Now we have all of this data set up as individual queries. These queries become tables in our model. So what I'm going to do to get that data loaded into memory is to click the Apply button here. I could click Close and Apply, but I'm going to be moving back and forth, so I'm going to go ahead and just click Apply. And at that point, Power BI comments to each of those individual files or connects to the internet to get our web sources and reads that data into the model, which in turn is really loaded into memory on your computer. When you close the file that you're working with in the Power BI application, then that data is released from memory, and it's just held in the PBIX file that Power BI uses to store your model and all of the reports and things that become what you create inside this desktop application. So I'm going to speed up this process so you don't have to sit here and wait for Power BI to load in all of my data. It takes a while. And if you're following along, just hit Pause and resume when all of your data's loaded. And we can see all of our queries listed as tables now. So my first step is to filter because this file is going to be very large with all of these tables as is. Let's take a look at our filesystem, and we can see just exactly how big this is. So even though I have lots of data that was in the hundreds of megabytes from the source, Power BI does compress this down. And we can see that at this point the file size is 41 MB. That's pretty impressive compression. But in order to query the database and be able to hold things in memory adequately, I want to reduce the data footprint in the model as much as possible, and that's where filters come in. But to show you that we are, in fact, filtering because it's going to be harder to see in the editor, I'm just going to do some counts of each of these tables. So let's just set this up very quickly. For example titles, I know that the unique identifier here is the tconst column, and I'm just going to do a Count Distinct, and I'm going to make this larger. It's hard to see. So I'm going to change the text size. We click the Format button here. It looks like a paint roller. And if you go into the grid section, there's text size. And I'm going to change this to 20 so that we can actually see what we're looking at here. And I'll repeat this process for each item. Here I'm going to do, let's just do--- This one's harder because there aren't some unique rows here. So I'm not going to worry about these, but we can count the principals, which is using the nconst column, and we'll reformat this, and that's from the IMDB sources. Here we have movieId. I'm just clicking the white space here to make sure that when I click movieId that I'm starting a new table visualization. Interestingly, it didn't give me the count option here. It just gave me a pause for a moment because it's trying to chart this out. So I'm going to switch this to a table view. And once I do that, now I can do my count distinct, and we will again resize so we can see the IMDB has lots of movie titles in it, but my movies has much less links. We will do a count here by movieId and change this to a table. I can use this to Count Distinct, reformat. And I can rename these things so that I'm not confused. So for example, this particular table is coming from the movies. You see how this is highlighted here. So I can rename this to say Count of movies movieId, and I can rename this one to Count of links movieId. And then last we have the names. Of course, it's not particularly meaningful here because we just have column numbers here. So I'm going to take a guess. And I can see when I click on column 1, we have this nconst, which is the same as what we have in the principals. So right now I have correctly selected, and we'll change this to Count Distinct, and we can rename this to names nconst. So now we know how much data we have in each of our tables, and this was just to validate our work as we're doing some filtering just to prove it out.

Shaping Data: The Basics

We have a lot of data loaded into the model now, but it's not useable in its current state for reporting. Fortunately, we have lots of options for reshaping that data. And believe me, there are a lot of options available in the Query Editor. Some you may never use because they apply to special use cases, but there are definitely a handful that you're going to use regularly. One of the first things that you should consider when working on your Power BI model is whether you need to filter out rows because all the data in your model is held in memory, and you want to make sure that you only include the data that you really need. Power BI provides a lot of flexibility in how you identify which rows that you want to remove from the model. A similar consideration is the elimination of columns that you don't need. Although it's tempting to include every little bit of data just in case someone might need it, too much data puts a drain on memory and adds clutter to your model. You can always add rows or columns back later in you need to. You might need to combine two or more queries to work with results as if it were a single query. For example, you might have data that comes to you in a separate file for each month of the year and you want to group these files together as if they were one file. Or as you'll see later in a demonstration in this module, you might need to use two queries and a merge operation as a method for filtering rows or for combining related data from two tables at the same time. And fixing up metadata is a common task as well. This would include changing data types or column names. And this step is important for having a Power BI model that works effectively and applies naming conventions that users understand.

Filtering Rows in a Query

If I were to filter, maybe I had different title types, I could select specific items on this list or other options. When I have text, I can use one of these filter types. I can specify that I want something very specific by using equals or does not equals, or I can do sort of pattern matching if I want to say something begins with or does not begin with a set of characters or conversely ends with or even contains. So that gives me a lot of flexibility with my text filters. And then if I have things with numbers, here's movieId for example. Again, I can explicitly select items, or I can use number filters where I have equals, not equals, greater than, greater than or equal to, less than, less than or equal to, or even between. When I have text filters, let's say that I pick one, notice that I can specify up to two possible filters. So I can have a simple filter where I specify my filter range, and I can use and/or logic to have a more complex filter if that helps me fine-tune that filtering. And then if that's not enough, I can click Advanced and continue to add the and/or clauses to, again, fine-tune those filters. So let's go to the movies query, and we're going to do some filters later on. But just to give you an example of how the filters work and notice that I don't really see how many rows that are in my dataset when it exceeds 999, but we have 58, 098 movies in our movies query. I'm going to do a query where, let's say that I want to eliminate all of the movies from 1945. So we can say Text Filters, Contains, 1945, click OK. And in order to see that change appear in our report, I either need to do Apply here, or back in the desktop, I can apply the changes. And notice that when it refreshes the data for this table, it doesn't go through all of the other tables. It only changes the one that we just did. So it turns out there's 200 movies in 1945. However, that's going to be a problem when I try to combine the movies with the IMDB database, so I'm going to clear that filter. And I can do that by clicking on the Filter button again and clearing the filter and then Apply. We can see we're back to the 58, 098.

Eliminating Columns in a Query

Another way to reduce our data footprint inside the model is to eliminate columns. For the moment, we're good in the movies query, so let's go back to the IMDB. And as we look through, we don't really need title type. Everything's a movie. So we know that it's a movie. So we can just right-click on the column header. And then in the pop-up menu, we can select Remove. And that's how we can remove a single column. Or as we go through, we might decide that we want to eliminate multiple columns. So here we have originalTitle. And I'll hold down either the Ctrl key and click a second column, or I can do the Shift key if I have multiple columns that I want to select that are contiguous. But I can select multiple columns here and again right-click the column header of one of those and select Remove Columns. So now I have done two at a time. And so you can decide what's the best way to go through here. You can eliminate endYear as an example. So you can do things one by one by one. Or another option would be to identify which columns that you want to keep. So here I have in my steps all of the changes that I have made. So if I decide that oops, those are not the things that I wanted to eliminate, I can just click the X here to go back to my previous condition. And now I can say, to the opposite direction, I can say I want to keep this column with the tconst, and I can hold down my Ctrl key and select primaryTitle. And I want to keep startYear, runtimeMinutes, and genre. And so now I've identified what I want to keep. And when I right-click one of the column headers, I can choose the Remove Other Columns option. Either way, I have arrived at the same place where I have eliminated columns and keeping only what I need and thereby reducing the amount of data in my model, which reduces the amount of memory required on my computer. If I look at principals, I don't really need ordering. This was something to keep those rows unique in the original dataset in IMDB. I don't really care which file these came from. And I don't really care about what titles they're known for because we will be able to link each person to a movie title later on in our process. Movies, we have everything we need. Links, here's an ID for the movie database, which we're not using, so we can eliminate that column. So this just gives us a link between the movieId in the movies table and the imdbId, which is in the titles table, and we'll work on that later. Ratings, we want to keep all of this. Tags, we'll keep all of that. Directorfilms, this is our special case we'll come back to later. And then Award Givers, as we look through this data, maybe year, notes, and so on. This is all just kind of useless data, these last four columns. So I just click Year, hold down the Shift key, click column 8, right-click, and select Remove Columns. And that's how we go through and reduce our data footprint by eliminating columns.

Combining Queries

Now let's talk about combining queries. We already saw how to get two files combined as a single query when we use the Get Data option and point it to a folder. Another way that we could've done that would be to import a file separately, and then we could use the Append Queries option to point to one table and then select the second table and be able to combine them both that way. So again, this process works only when the tables contain similar structures. And you can even combine more than two tables just by selecting them in the list and adding them to the Tables to append list. This is an after-the-fact combining of multiple queries. Another option for combining queries is to use the merge capability, and we can use that in a number of different ways. We can use it to add columns to one query by getting data from a related query, or we could use it as a filtering mechanism, which is one thing I'm going to do with it right now. Ultimately, I'm going to need to combine the titles and the movies and the directorfilms from the movies database altogether, but we have a bit more work to do before we're ready to do that. But right now, there are some other things that I can do to take advantage of the merge capability. So let's, for example, look at movies. Up here we have a movie identifier, title, and genre, and then I have the links query where I have movieId and the imdbId. What I'd like to do is to get this imdbId into my movies query so that I'll have it available to ultimately merge with titles. So let's do that step first. We're going to select movies and then Merge Queries so it pulls in a preview of my movies, and then I tell it what I want it to merge with, which is links. And merge requires that I have at least one column, it could be multiple columns, that relate one table to another. So I would just select movieId in the movies table and select movieId in the links table. They both contain the same data and the same data type. And Power BI looks at the data and it says great, we have a match; 58, 098 rows out of this first table are found in this second table. Now what's going to happen is the matches are going to occur in certain ways. We have these different join types. So if you're familiar with relational databases and working with SQL queries, you can do left joins and right joins and full joins, and we can do all these sorts of things inside of our merge operation. So in this case, what I want to do is use a join where only the rows that match between table 1, movies, and table 2, links, and that's what's kept in the final results. So we should still have 58, 098 rows in our first table because it already tells us that it has found matches for all of those. So what happens is the merge brings in the entire links table as another table. So we can't really see the data. But if I were to click on this table link for this first row, I would see it expand out to show me that the links table has movieId1 with this imdbId of 114709, and so I can confirm that there's actually data there. But this is not the final result of what I want to see. So I'm going to eliminate that step where I expanded that single table. What I really want is to expand this table row by row by row by row. And the way that I do that is to click this icon in the right side of the column header. And this shows me the column names that come over from the links table, and I just tell it go ahead and expand. Now I don't need the movieId again because I already have it once in this particular query. So I can clear that selection and say just bring in the imdbId. And optionally I can include the original column name as a prefix. So in other words, the original column name is links, so I could have this be links.imdbId. Or I can clear this and just keep this name of imdbId. And the reason for this is in case I already have a column with that same name so I don't have collisions. So we're just going to expand this table out. And so what that does is it brings in that imdbId into my movies table. And so now links is not really useful to me anymore. I've used it to enhance another query. So just remember that for future when we clean up our model at the very end. So that's an introduction to a merge operation. We need to do a little more work with our queries here before we can continue to merge movies with titles. So hang on. Let's do some other work in the Query Editor first.

Fixing Metadata

So maybe your data is such that you don't really have to filter out rows or columns or combine queries by using the append or merge, but you probably will have to fix up some metadata here and there. So what that involves is going through and looking at each query and making sure that the data types are correct and that the column names are meaningful to you when you want to use reports. So let's look at the titles query for example. This tconst, what does that mean? Well, I would like to rename that column as a more meaningful name, which to me would be the imdbIdentifier, ID. And so we can just go through and rename things. So let's change this to Movie Name for example. And maybe I don't really want StartYear, but Year would be suitable. So we just go through and come up with names. Maybe the name is fine, but I just want to capitalize things and put in spaces, make it a little more user friendly. So it's quite simple to go through and rename the columns. So notice that even though I renamed every column in this query, I only have one Renamed Columns step in the process because Power BI combines all of those or consolidates all of those steps into a single definition. Now the other thing that I should do as I go along is check data types. And there's an indicator here in the top- left corner for our columns of the data types. So I can tell that this is text. But if you're not sure because you're still learning what these data type codes mean, you can just click on that icon and see what it is. In this case, it's text, and so that's perfectly fine. Movie Name is text. Year is represented as a whole number. So in many cases, Power BI can automatically detect the proper data type. Sometimes it doesn't. Here we see everything is text. We can rename this as imdbId to be consistent with titles. And then this is the Name ID. This will correspond to values that we have in this IMDB query. We could rename this to be Category and Job. Just capitalize things, I prefer for a nicer, more professional appearance in my reports. But this is purely up to you. Now here's a case where I have metadata where the column names were just generic names. For whatever reason, Power BI did not detect what those column names should be even though they do appear here in row 1. Fortunately we have a transformation for this, and that is in the transform group of our home tab, I have used first row as headers. I could also right-click this grid in the top-left corner to use first row as headers. Either way, it accomplishes the same thing. And those column names now have been removed from column 1 and appear as the names. So again, I can come through, rename things as I need. Make sure the data types are fine. Here we have movies, movieId. Notice we have genre twice, once in the titles, once in the movies. We'll deal with this later. And then imdbId. Data types are all fine. Links, we merged that with movies so we're good there. And in ratings, I'll capitalize all columns except timestamp. And in tags, I'll do the same. Going back to ratings, looking at the data types, notice that ratings have a different data type, and this is a decimal number, but I just wanted to point out that it is different. So everything looks good. So we have now fixed up our metadata.

Summary

We have a good start on our data model after working through the concepts that were the focus of this module. We started this module by looking at all the different types of data that you could possibly load into Power BI and some of the ways that we could also connect Power BI to live sources, whether that's by using DirectQuery for certain types of databases or Connect Live for Analysis Services, tabular, and multidimensional databases. And then we got into the specific data that we'll be using throughout the demonstrations in all the modules of this course. Specifically we learned how to work with single files and groups of files whether they're tab-delimited or comma-delimited files. And then we spent some time shaping the data to structure it better for exploration later. We learned how to apply filters to reduce the number of rows that are available in a query, as well as how to eliminate columns in the query to simplify our data model. We also looked at options for combining queries, and we spent some time fixing the metadata in our model by fixing data types and by renaming our columns. There's still some more work to do shaping our data, but we'll delve into those new topics in the next module. One more thing. Here's a link to the resource that I mentioned earlier. It's the MovieLens dataset that you do need to download if you want to follow along with the demonstrations in this course. You can also learn more about the GroupLens organization and the MovieLens dataset by exploring the links that are available on the same page that you can download the data from.

Exploring Additional Techniques in the Power BI Query Editor

Overview

Hi, I'm Stacia Varga. Welcome to another module in the Power BI Data Modeling course in which we build on the concepts introduced earlier in this course by exploring additional techniques in the Power BI Query Editor that further enhance the model to better support analysis. The process of building out a model for Power BI is very iterative and often requires the use of various techniques to continually refine and improve the model. Whereas there are many transformations that you're likely to use very often, such as the ones that I introduced in the previous module, there are steps to perform and transformations to use that you might need only occasionally. But you should be aware of what they are, the scenarios they support, and how to use them. Therefore, in this module we'll continue working in the Power BI Query Editor to enhance the queries that we added to the model in the previous module so that the data's more useful. For example, we're going to learn how to work with XML data in the Query Editor because in its raw form, that data isn't useable at all. We'll learn how XML data behaves in the Query Editor and how we need to add transformations to extract data from various nodes at different levels. We'll also learn how to take data from a single column and split it out into multiple columns. And we'll look at functions that we can use to manipulate data by changing data types and to convert timestamp data. Again, these tasks are probably not ones that you need to apply on all of your data on a regular basis. But when the time comes, you'll know what to do. Restructuring data is not something you always need to do. It all depends on what your data looks like to start with and how you wind up using it. You might not know what you need to do until you start trying to use the data. But the good news is that Power BI is incredibly flexible. It has a lot of transformations already built in to help you restructure data. So I'll show you different ways of copying queries so that you can pull out sections of tables and refine the queries separately. And another restructuring activity is unpivoting the data so that you move data found across multiple columns into separate rows in a single column. And you can always write custom functions if you need to do something more specialized. In this module, I'll show you how to use a custom function to accelerate the process of transforming XML data. In addition, we'll explore several transformations to cleans the data. That is we'll change the contents of the data where necessary where we find data quality issues that could adversely affect our data analysis, such as having to look at values or wrong values or removing nonsense characters from columns in the table to name just a few transformations. Again, whether you need to use any or all of these cleansing transformations on your own data just depends on what you have. But by the time you finish this module, you'll have a very good idea of the range of possibilities that Power BI can support, and you'll be ready to tackle all kinds of data transformations in the future.

Query Enhancements

Although there are a variety of ways that you can enhance your Power BI queries to prepare data for analysis, in this module, we'll focus on two tasks that you might need to perform. First, you might need to add in new columns to hold data that you can derive from data that you've brought into the model by using a query. In the demonstrations in this module, I'll show you a few different ways to add columns. For example, you can expand XML data held inside a single column so that each nested node becomes a separate column. Or you can parse a column to split it into multiple columns. The split operation could be based on the position of a delimiter, or it could be as simple as defining a specific position within the column. And you'll see how you can define a new column by combining strings from two separate columns or by applying a function to a value to convert it into a new value. Another way to enhance a query is to give it a friendlier or more meaningful name. This step is especially important if you're building a model for other people to use because they might not be as familiar with the data source. It's usually best to use naming conventions using terms that are familiar to the end users. That way they can jump right in and start using the data without having to spend time unnecessarily trying to decipher the model's contents. So let's return to our Power BI model to see how to enhance the queries using these techniques.

Adding Columns by Expanding XML

Now we're ready to enhance our data model by adding more columns. So yes, we took stuff away that we didn't need, but now we need stuff for other reasons. So let's talk about what those reasons are. Remember this directorfilms, this is our XML data, and it's not very useful in its current state. Remember when we did the merge to combine the movies and links queries, we had this Expand option. And that's what we're going to do here is expand films, see what's inside there. And inside films, we have a film table, so we get to expand again. And this contains all of our movie data. Now I'm going to remove this director column because I only really want information about the movies from this particular database. And so what happens when we have XML data is we have a lot of nested information. So with the various elements contained within an XML file, Power BI treats those elements as individual tables. And so we'd have to go through and expand and expand and expand and expand, and I have a trick for dealing with that that we'll cover a little bit later in this module. But for now, I have the film ID, the title. I'm going to keep year. We'll get rid of some of this other stuff. So I'll keep year, awards, and location, and then I will eliminate all the other columns. What I'm going to teach you later in this module is how we can automatically expand these out without having to do this column by column. And in fact, notice that year doesn't even give me the expand option even though it contains a table. We'll deal with that by using a custom function later in this module.

Adding Columns by Splitting by Delimiter

Now let's look at another way to enhance our model by adding columns. Let's look at the IMDB query here where I have names of individuals. Here the names are put in with first name and last name in a single column. And maybe I want to be able to reverse that so that I could see the last name first followed by the first name. Maybe I want to look for people by last name. I need to be able to isolate that last name. So the way we can do that is to use the split by delimiter function, which, in this case, is just simply a space. Now if somebody's got multiple names, such as a three-part name, that might not work all the time, but let's see what we get. So what I'm going to do here is right-click on the column, and we have a split column transformation. There's two options, By Delimiter or By Number of Characters. So if I knew that I wanted the left three characters or the right three characters of something, that would be one way to split a column up. But in this case, that space could be anywhere in the string. So I'm just going to say I have a delimiter, and Power BI is pretty clever. It looks at the data and says I bet that the delimiter is a space in this case. Sometimes it could be a tab or a semicolon or a comma, and it would suggest those things. And you could even use other symbols, such as a pipe symbol, or even a combination of characters to use as your delimiter. In this case, we're going to use space, and it would split them up each time it encounters a space. So if there's a possibility of having two spaces, we'd wind up with three columns. Or I can just say you know what, probably the right-most delimiter would be the last space in the name and would likely be the last name. Just take a peek into Advanced options. You can see that we could also handle quotes in here and split into columns or rows. So a lot of flexibility here. In this case, I just want to make another column. And so what happens is here we have Name.1 and Name.2. We have two columns that were created. So we replaced the initial column name with two separate columns based on the same prefix. And now I can just rename these as First Name and Last Name, and that's how we add additional columns by splitting up values in a single column.

Adding Custom Columns

The next task that we have for shaping our data is to add columns using formulas to manipulate the existing data into a structure that would be more useful for me. So for example, we have the movies that now includes the IMDB ID, but notice this is a number, 114709. Whereas if we look at the titles, we can see that the IMDB ID all began with tt and then some number. And if we look carefully, that number is one, two, three, four, five, six, seven digits. So when we look at movies, sometimes they don't have seven digits. We already have 114709. That's only six digits. So what we need to do is in order to set ourselves up to be able to merge the titles and the movies, ideally we would link IMDB ID from titles with IMDB ID in movies. But number 1, we're told they don't have the same data type. And even if I changed it, we wouldn't match because they're not structured the same way. So I need to change this IMDB ID in the movies query to match what it would look like in the other table. The way we can do this is to add a column, specifically a custom column. We'll give it a name, IMDB, and I'll do a dash just to make it different from the one with the space. And then we need to use some type of formula. And you have a link here that will take you out to a page to learn more about this. So in the exercise files, you have a snippets file that will have this code already for you to copy and paste in. We can't learn all about desktop formulas in this course, but I'll give you enough to get you started. And through your Power BI community and documentation, you can find more information about how this works. So in general, what you need to do is to put the tt in front of our number and combine them as a concatenation. So any time you work with strings in the Power BI formulas, you enclose them in double quotes. And then when you want to concatenate that string with another string, you use the ampersand. And then this section of the formula does a couple of things. Number 1, we have the IMDB ID column. Notice that it is enclosed in brackets. So this would be the IMDB ID. And if you were writing out this formula directly, you could just position your cursor and use the insert for whatever you happen to have highlighted, and it would put that in for you, which ensures that you type it correctly. So when we have this column, remember that it had a numeric data type. And in order to concatenate a number with string, I have to turn it into a string. So there's a formula for that called number to text, and we pass in the argument of that particular column value. Now we also had the requirement that that needed to be seven characters long. And so if it's only six, we need to put a zero in front of it or however many zeros are necessary in order to make sure that that is a seven-character string. So we have another formula that says Text.PadStart, and you give it the first argument here would be what is that text? Well that would be our IMDB converted to a text string. How many characters does it need to be, and what character do I use to pad that string until it reaches a length of seven characters? So that's how we get to the desired result. Click OK. And now we have a new step where we've added a custom column. You can see the original IMDB ID, and the new one now has the structure that we need. That means we can eliminate now that we've used it the IMDB ID. And even though we've removed it, that doesn't affect our formula because the Query Editor processes things step by step. So we can go through and see each step of the way what things look like. So just at the time that we created that custom column, the IMDB ID did exist. Therefore, it produces a value, but we can remove it from our final result so that we can save memory.

Converting an Epoch Timestamp

We can work with dates also in a custom column. Remember in ratings and tags, we have this timestamp column, which doesn't really mean anything to me when I look at that. So what I can do is to add a custom column, and we'll call this DateTime. And again, this formula is found in the snippets file, so you can just copy and paste it in directly. And here we have some functions that look slightly different than what we saw before. This is the datetime and duration functions, and notice that the pound sign is used in front of intrinsic functions, like datetime and duration, as well as certain keywords. And so there's just a requirement that we use that as a prefix for these date-related functions. So with datetime what we need to do is pass in some value. And the reason we're doing this is because this is what's known as an epic timestamp. In other words, this is the number of seconds that have elapsed since January 1 of 1970. So the way that we convert this into the actual date and time is we take a starting point of datetime where we pass in a year, a month, and a day, hours, minutes, and seconds, so that becomes our starting point. And then we convert this timestamp column found here, and we say that that is our number of seconds. So if you were to look at the documentation for duration, you would see that argument here represents seconds. So we're just adding seconds to our January 1 of 1970, and that will come out with a UTC datetime value. And there's other steps you could take to convert it to your local time zone. But for our purposes, we really don't care so much specifically about the time. I'm more interested in the dates actually. So in that case, all I need to do is change this to a date format. The times are dropped. And let me just change the name of the column since I'm not going to include the times. And then, of course, I can eliminate the timestamp column. So we can do the same things for the ratings query. And I'll name it Date this time. Paste in the same formula because the column's the same. Change the data type and remove the timestamp column. There we go. More refinements, little bit better model.

Renaming Queries

Other work that we want to consider doing for shaping our data would just be naming our queries. Some of these are going to go away later in this module. But for now what I'm going to do is rename some of these queries. For example, I'm going to rename movies to MovieLens and change titles to Movies. Hopefully that won't be confusing later, but I'll make it a capital Movies. And we could make capital Principals, and we could change IMDB to People and not just actors or directors. And we have Ratings, Tags, Award Givers, I'll leave it at this, and we could name this one the moviesdb. And I've left some of these lowercase. Those are going to be hidden a little bit later. But basically, think about how you want these queries to appear because once you rename them, that's how they're going to show up in our model in the reporting side. So before I apply, let's just double check. Here we have the old names, and I do Apply changes. And because I have touched a lot of queries, it is going to reread everything. So I will pause the video and resume when everything's reloaded. So now that that has completed loading, you can see that our tables listed in the model available for our report have been updated where I made those changes. But another side effect was one of the counts that I had here is now broken, and that's because I changed the name of a query and a column, and it doesn't know where to find what I had. So when I click to fix this, it just plain goes away. The other option here is I just go back and create Name ID, do a count on this, and I'm back to where I was again. We'll just make this bigger. And so there are all our basic changes that you should consider doing to any data model that you're working with.

Restructuring Data

After you use the basic transformations to shape your data, there may or may not be more to do, and that's really dependent on what data you're working with. In our movies data model, we still need to wrangle the data a bit more to make it useful. First, I want to reuse data from one query and shape it in a different way. There are two ways that I can do this, one is to duplicate a query, and in that case I literally make a copy of the query. And any further changes that I make to the first query are not reflected in this copy. Now as an alternative, I can reference a query. In that case, any changes that I make in the original query carry over to the reference copy. Another transformation I need is unpivot. This is useful when I have similar types of data spread out over multiple columns in a single row, and I want those values to be set up instead in separate rows with a copy of the original that is not unpivoted appearing in each of the new rows. That'll make more sense when we see the demonstration. And last, whenever I want to perform a series of steps over and over again, I can group those steps together as a function that I can call once to run all of those steps, and that simplifies my work. But it's an advanced feature that requires you to learn the m language, which is the secret sauce behind all the work that we've been doing in the Query Editor.

Duplicating vs. Referencing a Query

Earlier I had mentioned I want to deal with the genre. Back in the movielens query, I have a genre column where I have genres listed out with a pipe delimiter. And in the Movies query that comes from IMDB, I have genres also. But in this case, they're listed as comma delimited. Now I want to do a couple different things with genre. Ordinarily I would just keep one, but I want to demonstrate some features later on as we deal with modeling. So I'm going to leave the genre alone in the IMDB side, the comma-delimited version, and I'm going to illustrate how we can work with this genre in a different way. Ultimately what I want to do is just be able to identify for any given movie what are the separate individual genres that I can have. See right now if I were to create a table based off the movielens data, and I have Movie Name, and I list genres, after I make this larger of course so we can all read it, I can only see all of the genres to which our movie applies. Or if I wanted to create a slicer, the slicer means that I can select movies based on a particular item that I select in the slicer. But if I want all action movies, not just action adventure or action adventure animation, I don't have a way of finding all of my action movies. So we fix that by separating out the genres into a separate query. So what I'm going to do is right-click on the query. I have either the Duplicate option or the Reference option. Let me show you what happens when I use the Reference option. It creates a copy, and you see as the source, this is just another copy of the same query. And I can continue to make changes to it. For example, I could remove a column here, and my source remains unchanged. Or the other option is to duplicate. So here I have yet another copy, but you can see all of the individual steps. So if I wanted to eliminate certain steps and start from a prior version, I can manipulate the query in that way. Let's say I go back to movielens and the step where I removed the column, let's say I remove that removal, in other words I add back that column, then movielens where I reference the query also gets that column back; whereas the duplicate, you don't really see a change because I had already removed that column manually. But even if I had not, that step would still be in this query because it's a copy of the original steps. If I go back to movielens the original and remove that column, then in the reference it is also removed; whereas in the duplicate it just stays put because it is a completely separate query. So it all boils down to how you want to manage your changes. I'm going to get rid of one of these, and we'll rename this as Genre. And I only really need to keep one of these columns to link it back to the original movie. And because I'm going to keep everything consolidated where possible in this IMDB version, then I'm going to decide to eliminate the Movie ID and just keep the IMDB ID.

Using Unpivot

And then for my genre, here's where the unpivot comes in to play. So notice that for 114709, this has five different genres. So now what I need to is do a split by delimiter to create separate columns for each of these items. So we'll do a delimiter. In this case, it automatically detected the pipe delimiter, and we're going to split at each occurrence this time. And so now I get genres 1, 2, 3, 4, 5, and 6. Some of them are null because not every row has six genres represented. But somewhere in our data, there's a maximum of six. So notice that each column is not necessarily distinct. For example, here I have Comedy in Genre.1, but I also have Comedy in Genre.2. So what I'm going to do is consolidate all of the values across all six columns into a single column, and that's what the unpivot does for us. So I need to select all six columns and then right-click one of them and select the Unpivot Columns option. So now for the 114709, I can see I have five rows, one for each of the genres that had been in there. Where they had been separate values delimited by a pipe, I now have converted those into five different rows. And so now I can match my IDs to multiple genres. Then be careful here so that you don't remove both columns. Make sure you just have the attribute column selected and then remove that column so that we're now left with the combination of IDs along with the genre values, and we can rename this to genre. Now I'll keep this data separate from movies, but we'll see later in the next module how we can use this information in the genres table with the information in the movies table so that we can filter by any distinct value of a genre.

Introducing Custom Functions

We have an example of a custom function when we combined the People query coming from our two tables. There's actually a line in here where a custom function was invoked. And so this was built automatically by Power BI. And when you invoke a custom function, you tell it what column you want the results to land in and what query you're going to use. So if we look over here in our queries list, when you see this fx symbol, that is a custom function that takes a parameter optionally. And in this case what's happening is from a sample file parameter 1, the content, which is essentially what's in this column here, the actual file contents themselves, gets passed into the query. So this custom function, we can't see what it does unless we open the Advanced Editor. And because this is an automatically maintained function, there's a warning that says if you change this, you're going to break things. Make sure you want to continue. So we're just going to take a peek in here. But basically a function takes an argument, so here we have where the source of this function is going to be something called, this is a variable, Sample File Parameter 1, and it gets passed into this code. This is called M language, and there's a CSV document argument where we pass in whatever that variable was. So that variable is coming from Sample File Parameter 1, which is defined in our sample file. If we go work our way backwards, we can see that this is set up as a very specific file. Remember when we imported the two files together, we identified one as the first file. That becomes the sample for the custom function to work. So we're passing in that particular file name, and then this is the argument that it takes. And the output then is the structure that imports in these files. After that's executed, we'll see information about that, and then it further gets expanded. So basically, we're just passing in structural information so that other details can be placed in this transform file from IMDB as a table, which eventually gets expanded out. So that's one example of a custom function. And you don't have to write any of that code. Custom functions can do a lot of different things, but in general what you'll find is that they take a parameter, and then you call them or invoke them from some other query to produce some sort of result, which is what we'll do next.

Reviewing a Custom Function to Expand XML

So now let's learn about working with custom functions, and this will be useful for us to work with this moviesdb, which was the XML source. Remember earlier I pointed out that I have no way to expand year here. And I have awards, which I could expand and expand that out and expand it out and expand it out, and who knows many times I have to expand that out. Instead of manually doing this column by column, which could have 1 expansion or 10 expansions, I can consolidate some code and use a custom function for this. And the way that I'm going to do this is to create a blank query. So under the New Source button here, there's a Blank Query option. And in the snippets, I have the code that's necessary. So we'll open up the Advanced Editor, and I'm going to replace the default code with the code for my custom function. So this is an advanced topic. We're not going to get into details of this in this course, but I just want to introduce you to the concept that it's possible to do more advanced things, and an excellent resource for you to learn more about this is a book written by Chris Webb called Power Query for Power BI and Excel where he gets into great detail about working with M language. Here's how this works. We have the function takes arguments, and we define variables for those arguments. So I'm going to pass in the table that we saw in the XML, and we'll call it TableToExpand and also the ColumnName that table is found in. We'll pass one in as a table object and one in as text. And then this little symbol says pass these arguments into the function. Behind each of our queries, if you were to look at the Advanced Editor for any of our queries, you would see a let statement. And a let statement returns some variable. And we can set up variables in sequence to create a variable based on something that we do that then becomes the source for another variable, and we can continue to build on these things, transforming each variable with some type of function available in our M language and create these subsequent variables. And then the last one is the one that we return from the function and becomes whatever we see in our final query results. So on your own, you could actually open up the Advanced Editor for each of your queries and see what that looks like. But you don't need to understand how to write this to be able to use this. But in general what's happen is we pass in the table and the column, and then this becomes a new column in the query that we have where it says add in a column and give it the same name. So if the name was year, we would have year\_table. And then it looks in the ColumnName year and says is that a table or not? And this little underscore here is kind of a placeholder for looking at the individual row values. So row by row by row, this is going to say is that value in the column that we pass in, let's say year, is that a table? If it is, then assign that to this variable. Otherwise, make it null. Now if there's actually something in there, then we're going to create a new column called year\_table. This becomes the column. We'll pass that in as an object to a variable, and then we'll do some stuff to it. So basically we're going to do a transformation so that we'll have this whole big column row by row by row, one row for each row that's in our initial XML table, and we're going to pass in that column and look at each row and say hey, is that null? How is it null? Because if the original value is a table, it would be null. So in that case, we're going to create an empty value. Otherwise, we're going to give it whatever was in there. And basically we're just going to come up with a list of column names that are in there. And it will append the ColumnName that we originally passed in. It will append in the ColumnName that it found in that table that it was expanding. So in other words, the table contains things that say year released and something else. It'll come up with NewColumnNames and then actually expand out that column to give us a table. But let's use it and then backtrack and look at this so you can see how it works.

Naming and Using a Custom Function

We give this a name. I'm going to call this ExpandColumn. And you can see that it wants a table to expand, which is kind of hard for me to do from this interface, and a ColumnName in order to invoke it. But we're going to actually implement this inside of our moviesdb query. So in order to do that, I need to go into the Advanced Editor for the moviesdb, and I'm going to manually update this. So just a little quick walkthrough of what we have already. We have the source. We have Expanded films, Expanded film, Removed Columns. Those all have M language functions hiding behind the scenes here. So to use my new function, what I need to do is add a comma, each statement in here needs to be separated by commas, and then I can give this a statement. So I'll paste in my code from my snippets file. And this is how I use that function called ExpandColumn. I pass in the table to expand, which comes from the prior variable, which was a table. This is what this returns is a table. That's what we see when we're looking at the query results. So we pass in that table name in the variable, and then I tell it what column that I want to expand. So that returns another table called Expanded year. And so what I need to do is to use that as my result for the let statement. So the let statement includes a series of variables that perform some functions and stores it in a variable, and then my last step has to be in whatever variable I want to return as the result. So you can see when I click Done, we have Expanded year, and now I have several columns here, year.Element :Text, year.released, year.rereleased. So that will be as if I had expanded them out just like I have awards where I can expand and expand until I cannot expand any more. I used a single step in the Advanced Editor to achieve those same things. So you can see here with Expand awards, it uses a built-in function, ExpandTableColumn, to pass in the table that I had created, what column, and then it knew by looking at it what columns it was going to create. So we have two expand columns. So I just consolidated that and produced these new columns. So I can do the same thing with location. And you can see that gives me a column called site. How about we use our function again, so we'll add in a comma. And this time, we'll need to reference this Expanded location so that we use ExpandColumn using this previous variable, tell it what column we want to expand and then return the final result. And when it's finished, we have several new columns. Custom functions provide you with a lot of flexibility.

Consolidating the Data Model by Merging Queries

Now I have most of the pieces that I need here to consolidate the model and present it a little bit better in my Power BI file. What I want to do is get to that final stage of consolidating the movielens data with the IMDB data and also this movies database data with the Movies data. Now with Movies and movielens, I can merge those things together based on the IMDB ID. So let's do that now. I'll go back to Movies and the Home tab here, merge the queries, select my movielens, and we'll merge on IMDB-ID. And what I want to do here is to do an inner join. So in other words, I'm going to filter down the IMDB dataset so that it only keeps whatever it finds that matches in the movielens database. And then, just like we did the previous time we did a merge, we're going to expand this out. But the only thing that I really need to keep here is the Movie ID. And I'll need that so that I can link out to Ratings and to Tags later on, as well as to Genres. So I have one place for both the IMDB and the Movie IDs brought together into a single place. Now remember before the merge, we had 298626 items. Let me clear this thing. But now that I've done this merge, let's do the Apply change and watch how that number changes now that I've done the inner join merge between the two queries. So now we can see that instead of over 200, 000 items in IMDB, we now have 34, 000. So we filtered this down considerably. More things to do. Now that I have used movielens to filter the Movies query, I'm going to disable the load. In other words, right now the movielens shows up in my model, and I only need it to produce this final Movies query. So I'm not going to load it into the reports. And it's just letting me know hey, if you're using this anywhere, it's not going to be available now. So now that I take that away, of course I have to apply this, so it's gone. And we broke the one item that referenced it, and that's okay because I'm taking it away anyway. It is now italicized to let me know visually that this particular query is excluded from our reporting model. I'm going to do the same thing with links because it has served its purpose to get the IMDB into movielens, which we used for merging purposes. Now if I want information from the movies database to go into our Movies query, how am I going to do this? I have a film ID, I have a title, and nothing about IMDB in here. So we're going to take a chance and match on titles. So we'll go to Movies as the first item, do a merge query, select moviesdb. We'll take Movie name with title, and now it's saying hey, some of this data that you're using comes from someplace else, and make sure that it's okay to share that information. So we can either ignore that, or I can say you know what, this is public information anyway. It's fine. So we can save that. And what we're going to do is instead of an inner join to filter down Movies, we're going to say keep everything from the first query and only include data from the second query if you can find the match based on the names. So we'll see how well it matches up here. And it's done. You can see that it only matched 3, 583 records, so it's a good thing we didn't filter down Movies that much or we'd not have very much data at all. But we have something to work with here. So let's click OK. Just like other merges, we get a table that we need to expand, and then we can decide what we need to keep. So I don't want all of this information, but I do want the awards and maybe site place, name, class, and type. And when that's done, we can clean some things up. So here is our type. We're going to rename. Now that I look at the data, I don't need to keep this anymore. Let's remove this column. Then we can rename Site Place, Site Name, Site Class, Site Type. And we also need to clean up some data types here. So this is text, and we could actually do bulk assignments here. So we can do that by just selecting all of the columns, and you can right-click and pull up the Change Type pop-up menu to text, or you can click the icon here to change this to text. So multiple ways to accomplish the same goal. And then, of course, we can change the enable load. We'll disable this from our reporting side of the model. So now we have all of the queries set up the way that we need it, and the Movies query here has consolidated information from the other two movie-related tables.

Data Cleansing

Besides reshaping the data to get it into a structure that's better for exploring, we also likely have to deal with dirty data issues. This situation is much more likely to arise when working with text files rather than relational data because text files don't have rules enforced on them like relational data does. So we can wind up with data values that don't make sense or have the wrong data type or whatever. We need to go through our data and look to see do we need to replace values? Do we need to trim off any white space that's trailing at the end of text or maybe at the beginning of the text? Or do we need to create some custom column formula to transform some dirty data into something better or to create data that doesn't exist yet in our original source that might be more useful to us? Or maybe there's errors in the data from some step that we performed, and we want to replace the errors with just a null value as an example. Or maybe something has happened, and we have duplicate values. So these are all things among others that Power BI includes to enable us to clean up our data before we start reporting on it. Let's take a look.

Using Data Cleansing Transformations

So let's see what kinds of data cleansing we need to do inside of our data model using Query Editor. After we did the merge between the Movies query and the moviesdb query, notice some things that look a little odd. For example, here's two rows with the same ID, the same movie name. And then we have two more with the same name that have different years associated with them. And here we have another set of rows. And so this seems to be because of data that we got from this moviesdb. In fact, I can see duplicates in here. I don't want to have duplicates in my main Movies query here. So one of the things that I can do is fix this so that we remove duplicates. So we can see for example, here's three and two that were consolidated into a single value. So having fixed that problem, we'll go back to Movies, and we should see, as soon as this finishes resolving that our duplicates and triplicates and such have been fixed. Alright, that looks better. If we look at the Principals query, there's a couple things to clean up in here. For example, Job has a /N where the value is null. Same with Characters. We right-click and select Replace Values. So let's type in /N, and we'll replace it with nothing. Just leave that blank. We can do the same thing in Characters. Now the other thing that we can do with Characters is to take off these extra characters. So for example, let's remove the left bracket and double quote, and then we'll do the same thing with the right-hand side, double quote, right bracket. So sometimes we need to clean up characters that don't belong. In our People, we have a similar problem with Birth Year. And actually a shortcut would be just to right-click on that value, and it prepopulates, and we can just replace like so. Let's look at Award Givers, and I think we cleaned up most of the problems. Well here's one where we can replace a value with a null. So we can use Replace Values. Sometimes when you do the split of delimiters, depending on what your data looks like, you might need to do a trim. So there's a Trim command here that would get rid of any white space. I don't know that any exists there. But notice that there are other kinds of things that you could do to lowercase, UPPERCASE, or capitalize things. And, of course, you can always remove errors if some kind of transformation left behind something that could not be transformed, if it was just something maybe like you had data right here where there's some characters that didn't translate properly over. It's not technically an error. But let's say that you try to do some operation on this, and it turned this into an error, you would see the word error displaying here. You could replace those errors just like you can replace values. You can replace errors with something else. So we want to maybe infer the gender where we can where there's some gender-specific references, like actor and actress. But sometimes we have director who is also an actor, so we can't just look at whether a value exists in a particular column. So in that case, we've got the power query function that will help us out. So we have conditional logic where we could say something like if the primary profession equals actor. But a better approach, since we know that that primary profession may contain other data as well, we would say if it contains actor, then return male. Else we could return something else. We could say unknown. But I want to check for actress as well. I can put in another if condition on the tail end here. So I can say if Text.Contains Primary Profession actress then Female else Unknown. So I should get one of three possible values. So that's another way that we might want to cleanse data is to pull out information by using a custom column.

Summary

And now most of the work we need to do with our Power BI data model by using the Query Editor is complete. In this module, we continued to build on the data model started in the previous module, and we did quite a lot of work to get it into a useable condition. We started by performing many different types of query enhancements, such as working with the XML data to expand out the nested nodes into columns, and we used a transformation to create multiple columns from a single column by using the split by delimiter transformation. We were also able to enhance our raw data by using custom columns so that we can match up data between tables and convert a timestamp value to an actual date that people can understand. And we gave our queries better names as well. And that's not all. We began restructuring the data so that we could separate genre data out from individual movies. And to do that, we learned about duplicate versus reference queries. And another restructuring task was to use unpivot so that we could take genre values spread across multiple columns and set them up as unique values in multiple rows inside a single column. We also learned how to shortcut the process of expanding XML nodes by using a custom function and how to use the merge query transformation to consolidate our three separate Movies tables into one table that we're going to rely on for reporting. And last, we went through the data to clean up data quality issues. And by doing this work, we learned about several other Power BI transformations, which we used to replace bogus values with nulls or how to use conditional logic in a custom column to infer data when we don't have values that we need. All in all, we've done a lot of work to take our raw data from a variety of sources, bring it together into a cohesive whole and clean it up to make it useful, but we're not done yet. We're only done with our data modeling tasks in the Query Editor. In the next module, we'll learn about other types of data modeling tasks that do not involve the Query Editor. In that module, we'll be learning about adding in relationships between our tables and defining hierarchies, which are relationships between columns in a single table. Those additions to the model are just as important as all the work we've done so far. And I do have another resource for you. In this module, we were working with the XML data from the Movies database from a Stanford professor. And if you're curious about that data, here's a link that includes documentation about that database.

Enhancing a Power BI Model with Relationships and Hierarchies

Overview

Hi, I'm Stacia Varga. Welcome to another module in the Modeling Data for Power BI course. This time, we focus on enhancing a Power BI model with relationships and hierarchies. Both of these elements are important for successful exploration of the data, but in different ways. Once we have our data from different sources loaded, shaped, and cleansed one query at a time, we need to connect the separate queries together and refine the data even further so that it's ready for creating visualizations. Specifically, the connection between queries is known as a relationship. And I'll start by showing you what happens when your model has no relationships at all and how the addition of relationships is the key to successful data models that contain multiple queries. Under certain conditions, relationships might be detected and added automatically by Power BI. But sometimes they're not added correctly and often not at all. Therefore, you need to know how to review existing relationships and how to add new ones if necessary. Now relationships can be tricky, and I'm only qualified to talk about data relationships here folks. You need to be able to recognize an issue caused by relationships and know how to fix it. Whereas relationships are almost always required in a data model in order to display data correctly, another important element in a data model is a hierarchy. Now hierarchies are not required, but they can make it easier to explore the data in your model. So we'll spend some time adding and using a hierarchy also.

Understanding Relationships

Relationships between tables are the key to successful data models in Power BI. And incidentally you'll hear me switch back and forth between the terms tables and queries. They're really the same thing. Queries we work with in the Query Editor, but those turn into tables in the Report Editor. but regardless, right now in our data model, we have a lot of tables, and there are no relationships. Or are there? If we only concern ourselves with a single table, then we always have a relationship between columns in a single row. That is whatever is in the first column of row 1 is automatically related to the data in the second column of row 1, and both of those are related to the data in the third column of row 1. Otherwise, the data makes no sense, right? When we build a visualization using columns from the same table, we can count on the right data from each row automatically lining up, such as when we select Movie Name and Year from the Movies table for display in a table visualization. If a relationship exists between two tables, then we get similar behavior. Our relationship requires that each table has a column that contains values in common with values found in the other table. In at least one of these tables the values in the column must be unique. That is, each row has a value that's not found in any other row of the same table. Power BI acts as if the tables are merged into one. But the results depend upon the type of relationship. A one-to-one relationship exists when each table has only one row containing the value that they match on. So if hypothetically we had Movie Name in one table and the Year in another table, but we could match on the IMDB in both tables, then we'd get a result just as if both columns were in the same table. But sometimes we have two tables with a one-to-many relationship. In that case, one table has unique values in a column that match up to one or more rows in the second table. When that happens, the values in the table with unique rows gets repeated with each of the related values in the second table, the one on the many side of the relationship, like we can see here when we combine the Movie Name from the Movies table with the Characters column from the Principals table. In the Movies data model, this relationship between Movies and Principals was automatically detected. So there was no need for us to take any extra steps to make this work, and we'll see why in just a few minutes. When a relationship between two tables does not exist, there are clues. For example, when you try to create a visualization from movies by using the Movie Name and then try to add the Rating column from the Ratings table, we see the same rating value for each individual movie, and we see that value as the total. That's what happens when we use a numeric column in an unrelated table. The aggregation calculated by Power BI just simply repeats. If instead you try to use just columns containing text from two unrelated tables, like Movies and Genres, Power BI just says no. Let's look at all of these scenarios in our Movies data model.

Checking for Relationship Problems in the Model

Here you'll see a bunch of failed visualizations because of a lot of changes that I made to the model after adding these visualizations to the report. Don't worry about those. Those are not related to relationship issues. Those are just naming issues. For the moment, we're just interested in working with relationships. Now let's add another page to this report and delete that first page because it's not necessary anymore. We're going to look at what happens with our relationships. So we'll start with two columns from the same table. We'll do Movie Name and Year. I'll adjust the formatting here so that we get something we can read. And then I'm going to filter with a page level filter, so we can look at a few movies at a time. Make it a little easier to read. So I want this to start with A Beautiful, and apply this filter. So we just get a few items here with some movies and years. So everything looks good so far. That's because these come from the same table. So now I'm going to copy this table and paste another copy of it. This way I don't have to fiddle with the formatting each and every time. And so when I select this table, make sure that it's highlighted here so that we can get to the proper settings, I'm going to remove Year and add Characters from the Principals table. I'm going to expand this out and look at it. I'm going to look at it, and it looks good. And that's because a relationship was automatically detected by Power BI, and we'll look at that in the next demonstration later in this module. So that all looks fine. If we paste again, drag this version of the table. And in this version, we'll replace Year with the Rating. We can see, oops, there's a problem. It tells us that there's a relationship issue. Power BI recognizes Rating as a value, you see this sigma symbol here, and it uses a default aggregate function. Let me select this so we can see here. If we look at Rating, see the checkmark here says to Power BI go ahead and add up these values for each movie, meaning it should add up all the ratings for each movie individually. That's the expected behavior if the relationships were correct. But since there is no relationship, Power BI doesn't know what the total ratings are on a movie by a movie basis, so it says okay, I can give you a number, but it's going to be the grand total for each and every one of these. So when you see values repeat like this, and when you don't expect that behavior, then that's a clue that Power BI doesn't have the relationships necessary to compute the rating as it should. And another clue when a relationship problem might exist is when we have two columns, let's pull out here, and we're pulling data from two tables, but just the text values, then Power BI just gives up completely and says I can't display this visual. So we have relationships between columns in the same table, relationships between columns cross tables, no relationships as indicated by either repeating numeric values or by a failure to display a visualization.

Introduction to Managing Relationships

As we'll see in the next demonstration, there are a couple of ways to work with relationships. Regardless of which option we choose, we manage relationships in the Power BI model directly, not in the Query Editor like we did in the previous module. But in general, a great way to start is with the relationship diagram. It's a nice visualization of the relationship between two tables, and you can clearly see the relationship type after it's created, which, in this case, is one to many where you have the one indicator next to the Movies table and an asterisk indicator next to the Ratings table to represent many. What's less obvious in the diagram is, which are the columns that are related between the two tables. But you can double-click on the line to see the relationship definition that appears in the Relationship Editor. Here you can still see the tables involved with Ratings on top and Movies on the bottom. You can see the problems involved in the relationship on each side by looking at the cardinality, which here says Many to one. That means Ratings is the many side of the relationship, and Movies is the one side of the relationship. Or another way of thinking about this is we have one row in the Movies table that can correspond to 0, 1, 2, 3, or many different rows in the Ratings table. There are a couple of other settings here that will make more sense when we're inside Power BI, so let's go there.

Reviewing and Creating Relationships in the Model

We have a relationships diagram, and there's an icon here on the left side of your screen that you click on, and that opens up the relationship diagram. Now it might be too large to see all of the tables and the relationships. So in the bottom right-hand corner, you can click this icon here to fit everything to your screen. And then if you want to adjust it to resize things, you can use the plus and minus symbol or the slider here to change things around. Now what we have is a set of tables that already have a relationship, and we can see that by these lines that exist. So we have the Movies table, and we can expand the size of this table to see all of the columns that it contains. And it has a line drawn to the Principals table. So that tells me that there's a relationship, but you can see that this arrow changes its location depending on how I position the tables and how I resize those tables. So it's not really a good indicator of what is related between those two tables. It just tells me that there's a one-to-many relationship here between those two tables. And if I want to see specifically the details about that relationship, I can double-click on this relationship line, and that opens up the editor where I can see. We have Principals, we have Movies. We can see that there is a relationship between IMDB ID in both of those tables. And I should point out that when we create relationships, the relationship is dependent upon a single column in both tables. So if we're not able to select a single column in each table for this relationship, then we need to use our Query Editor to make some type of change to the data so that we can create a column in each of the tables where they do have matching values. So in this case, IMDB ID is the single column that fits that criteria. We have a many-to-one relationship, meaning we could have many rows in the Principals table that correspond to exactly one row in the Movies table. We can see the preview of some of the data there, and we can see that there's a single cross filter direction, and we'll talk about this later. And we can see that the relationship is active. There are some capabilities in our expression language that can enable and disable relationships for the purposes of computing calculations. But for most purposes right now, we would leave this relationship active. So we currently have this relationship. I didn't create this. This was created automatically by Power BI. And the reason it did that is because it detected that both of these tables had a common column name with the same data type. And it was able to do a count on the values in both of those tables and determine the cardinality. So we could see that there were distinct values in the Movies table and non-distinct values in the Principals table for those columns, and therefore that relationship is likely to be one to many. Similarly, we have the People table where we have the Name ID column that exists in both of those tables, and that becomes the relationship that we see here in the Relationship Editor. Again, a many-to-one with many on the top, one on the bottom and then a single cross filter that we'll talk about later. Now there are no other relationships to the other tables. It wasn't smart enough to figure out the Movie IDs was the common element here. And sometimes Power BI just is not as clever as we would like it to be. That's what we need you for to do the data modeling. We need to create some relationships here. An easy way to do this is just to, we already know what we need to connect, is to go to the one side of a relationship, click on the column, and then drag it to the many side. And so I can just drag and drop Movie ID onto Movie ID in Ratings. And as long as the data types match, then it goes ahead and makes that relationship. We can, of course, confirm that that is set up correctly by opening up the Relationship Editor.

Using the Relationship Editor

The other option that we have for creating relationships is to click the Manage Relationships button, and that will be available to you whether you're on the report page or on the data page or on the relationship diagram page. All get you to the same place. Click that button, and here you can see what relationships already exist and which are active. And you can see the tables and column names as well. And if you want to add one through the editor as opposed to the diagram, then you just simply need to select the tables involved. Now it doesn't have to be selected in a way that the many is always on top, and the one side of the relationship's on the bottom. I could instead say I want to take Movies here and Tags here and then confirm the direction of the cardinality. So in this case, Movies is the one side, Tags is the many, so we have the one-to- many relationship defined here. And then I would just specify I have Movie ID in the Movies table, Movie ID in the Tags table, and that's all that I need to do to specify what column is involved in the relationship. There are other types of cardinality. We've seen the many to one, and this, of course, is the one to many. If we had situations where we had two tables that each just have a single row on that matching column, then we have one-to- one relationships, and it's possible to have many-to-many relationships, and we'll learn more about that later. So click OK when we're done, click Close, and we can see that to this relationship two tags has been correctly created. The last thing I'm going to do is take a Award Type and connect it to award in the Award Givers, and that should be a one-to- many relationship also. And in this case, we have a many-to-one relationship, the opposite of what we had in the other scenarios because in this case we have the Award Givers just list an award type once, and that award type can belong to one or more movies. So this is the reverse of what we've seen with the other relationships. So we can spread things out and make it look the way that we like here. And just to confirm, most of these have a relationship where the movies is the one leading out to a many with Principals to Tags to Ratings. We flip that with awards, and then we have Principals is the many side of two different relationships here. And we have one more relationship to create here between movies and Genre. We have the IMDB ID on the Movies side. Drag that to Genre. And so now we have a one-to-many between Movies and Genre.

Resolving Relationship Issues by Fixing Data Quality

So now let's take a look at the effect of setting up these relationships. If we go back to the report, we can see that the ratings are now calculating unique values for each movie, and we can see Genre has filled in where we previously couldn't see the visualizations here. Now let's check our other relationships. Let's start with clearing out Rating and adding in an award. And here we see a bunch of blanks. And so it's hard to say whether these movies didn't have awards or what we're looking at exactly here. So let's adjust this by clearing out the Movie filter. And so I can see I do have award values. And if I reverse the sort, there's blanks. And I reverse it again, I see values. So my assumption initially is that I have a bunch of blanks in here. Let me put my movie filter back on so I can check the other relationships. Here we have Principals with our Movies, and I'm going to add in People, and everything looks good here. And the other relationship that I want to check is Tags. So let me remove Genre, and then I'll add in tags. Basically, these tags look reasonable for this movie. So the only thing I need to fix here really is just the data quality in award. So let's go back to the query, and this is in the Movies table. And so there are null values in there, but those should be showing up as blank in the filter, and they're not. So I have some other data that's white space, maybe some unprintable characters. I'm not sure which. But I have transforms for either one. A trim and a clean trim says remove white space from the column. Clean says remove the non-printable characters. So I don't know which is my problem here, but I'm going to start with Trim. That's the most likely problem. And, of course, we'll apply those changes so that we can see the results in our report. And there are a couple of errors as a result of some other dirty data in our query, and you could look at those errors if you want. But there's only two of them, and so I'm not concerned about it for our purposes. But at any rate, they are there. But Power BI has loaded everything else into the data model. And now let's go back and check on this by clearing the movie filter, and then so we can still see blanks. But if we go to the visual filter and now clear out the blanks from our selection, and we can see awards by movie. So that confirms most of our relationships now.

Introducing Bridge Table Relationships

The next thing I want to show you is problems that can occur when we have relationships that look like this where we have a table that sits between two other tables and this middle table is the many side for both of those other tables. Back in the old days of data modeling, we used to call this a many-to-many relationship because we have many movies have many people work on them. Many people can work on many movies. And this Principals table is the table that is necessary to allow us to query both of those tables in a nice way. The problem with that terminology is that Power BI has this other reference to many to many. So let's look at this. And so there is support for many-to-many relationships, but that's a different kind of many to many. So I hesitate to use the terminology for this scenario here. So we'll just call this the use of a bridge table, rather than many to many. The difference would be in the Power BI many-to-many relationship, remember when I said that you have to have unique values in one column. So for example, Movies, each of these IMDB ID values is unique within this column. Whereas any given IMDB ID that's one row in movies could appear multiple times in the same column over in Principals. The difference with the many-to-many support would be if I had movies here where I had multiple rows per movie in this column. So maybe instead of having a separate table for genre, I had one row per different genre type per movie. The potential to have multiples of the same IMDB ID in the Movies table and I'd want to join it to multiples in the Principals tables, that would be the cardinality that this many-to-many cardinality is attempting to support. But we don't have that scenario. And what we're going to do instead is address what we used to call many to many with this bridge table. So I hope that clarifies some of that terminology.

Resolving Relationship Issues with a Bridge Table

So now let's see what happens when we have this bridge table involved. So I'm going to go back to my report. I've put all my filters back on to make this easier to look at. I'm going to copy this table here with movies, characters, and people in it and create a new page and paste this in. I'll move it over here. And of course we lose our filter because that was a page level filter. So I'll just add that back in as another page level filter. So here I have my filter. And we're going to pull out characters from this so that I just have people, which is fine. Everything looks good as long as we have text fields in common. Everything is filtering just fine. Where we start to see strange things happen is when we are working with counts. And we haven't learned how to build measures yet, but we're going to do a simple one. I always get myself in trouble if I try to use this new measure, but the measure always winds up in a place I don't expect. So make sure it's going to go where I want it to go. I'm going to select this ellipses button to the right of the Movies table name and select New measure. And here in the formula bar above our table, I'm going to say Movie Count is the name of the measure. And it's a simple measure. All I want to do is to count the rows in this table. And so as I type some function, it shows me all of the possibilities, and then it shows me the syntax that I need a table name in here. So I can find the table right here in Movies and then close parentheses and click the checkmark. And now I have a movie count that is something that I can add to my report. Notice when I add that, now I start getting all kinds of crazy things when first name and last name that just wasn't there before. And if I take first name and last name out, I get one movie count per movie with a grand total of 4. So that's what I expect. And if I were to add in the principal characters, then it's just being weird again. Whereas if I don't have the movie count, I see what I expect. For each movie, I see the characters, and I see more movies. So when I have this type of a relationship, as soon as I add a measure to this, weird things happen. And the reason is not particularly obvious. But what happens is the relationship on the other side of Principals is having some influence. See both of these tables are trying to filter the bridge table. So the People table is not filtered, and the Movie table is filtered by Movie Name. But the People lack of filter is overriding and saying I want to show all of those Principals. And so poor little Power BI says okay, that's fine, and it comes back to this measure for this particular combination. So there's still only a grand total of 4 movies in here, but Power BI says I don't know how to filter the characters when you're using this measure in there. So because of the People filter, I'm going to give you all of the characters. So normally what happens when we have these relationships, each one of these tables if we have a many-to-one relationship would individually apply their filters against the one table. Here we have two tables trying to filter a central bridge table, and that's where we're running into trouble. The way that we fix this is going to our relationships, we can say let's use a cross filter direction of Both. And we're not dealing with security in this course. But if you were to think about security when you need to publish this in your Power BI service, then you need to determine does security need to apply on both ends of this relationship. So now that I have modified this relationship and I go look at the report, now we're back to what we had originally seen with or without the movie count in here. So I can take movie count away. I have the same number of rows with or without movie count. And the unique thing here also to bear in mind is we're not counting the rows in the table, we're counting how many movies do we have. There's four. So the grand total is correct in this case. Similarly, we add a count to the people, people count, and add it to this. Notice that the people count measure has the effect of adding up all those characters. So this is our clue that there's a relationship problem. Remember when values repeat the grand total and individual values. So if we take Principals out, even doing this, we have the movies, but the People Count is incorrect. If I add in people, it's giving me every single person. So again, this is because of that relationship. The filter goes one way. It does not follow through. It does not push the filter all the way through over into People. Conversely, the People filter is simply not filtered at this point. So we change the relationship between People and Principals. Any filter from Movies flows from Principals into People. Any filter on People flows through Principals into Movies. And then at that point, we have correctly identified the movie count and the number of people. So that is something to pay attention to. It's kind of an odd type of data modeling situation that you might not run into in many of your data models. But on occasion, it does happen, and this is how you deal with it.

Working with Hierarchies (Ad Hoc)

So let's set up a new page here so that we can explore our situation with hierarchies. When we look through our data, a good possibility for hierarchies has to do with movies and sites. So I could say I want to look at movies, and let's make this bigger. I can tell already it's too small. And we'll make it nice and wide here. And we'll do Site Class, Site Name, Site Place. And I'm going to set the filter on Site Class so that it's not blank, so we have stuff to look at here. So again, we have Movie Name, Site Class, Site Name, Site Place, and we'll add in Movie Count so that we can see in our dataset 1, 466 of them have some sort of a site class. What kind of a place was this film taking place at? I can explore this data. This is a lot if I want to pick through and say well, where are all the movies that have cities? Of course I can sort by that sort of thing. But I can also organize my structure of my information here so that I can explore this in groups. And the easiest way to do that is just to change this over to a matrix. So I'll click on the matrix visualization in the tray here, and then I need to rearrange some things here. So let's say that I want Site Class to be first, and so that becomes the top level of this matrix. Now because I have multiple levels to find here, Power BI says aha, you have multiple levels, which means there's a hierarchy of some sort. And generally speaking, it's a good idea to arrange the fields and the rows here so that you go from few categories to more categories to more and more and more. So it's basically like a one-to-many relationship moving from the top levels down to the lower levels, which means Movie Name is really the most granular, most detailed part of this. It doesn't change our grand totals here. It just changes how we're going to be looking at things. We have this organization of movies grouped by place, grouped by name, grouped by class, and we can drill down on this or expand down to see the next level. So this one icon, which may display Tooltip on your screen, this icon says expand down to the next level. And so we can see the Site Class is now bold, and then the next level down is some information. And it looks to me, when I see this, that it might be something we need to rearrange. If I put place after Site Class, then I can see things that like country names, England, USA, so on. And so that looks better to me. So we can rearrange our hierarchy into something that's more logical if our first attempt doesn't make sense. And then again, I can use the same symbol to expand again, and that expansion just basically takes each level down to the next level, and some of these are empty. So they're not very nice to look at sometimes. This is dirty data. But if I have cities, then with no grouping, I can tell that there's a blank grouping here because this is a bold value and then some city names. Then I have a grouping by states or countries with the site name below that. And let's filter this down a bit more so that we cannot have to look at all these blank values. So I'll take Site Name, not Blank, Site Place, not Blank. And so this is a little nicer. And so we can again see this breakdown level by level by level. And then ultimately, if we go down to the bottommost level, we can see the actual movie that is associated with that particular grouping of things. So that's a hierarchy.

Adding a Permanent Hierarchy to the Model

So now I created this hierarchy in a report. That's not very useful for my model if I want to frequently use this type of hierarchy. So what I can do is create this in the model itself. And all I need to do is start with the top level and say New hierarchy, and so Site Class becomes part of that. And then I can right-click and add to the hierarchy, but I did it in the wrong order. So all I have to do is drag and drop to rearrange to match the hierarchy I have over here and then Movie Name, Add to hierarchy. So there's my hierarchy. If I want to create a visualization that uses that hierarchy, then all I need to do is I'll create a new visualization. Let's say that I have a matrix here. I'll put the hierarchy in there just for the single click. And so now I'm able to do what I had before, and the number's higher here just because I don't have filters applied like I do in the first matrix. But we can see that we get similar behavior here. So in one click, I have my hierarchy established. Now the other options for drillings with our hierarchies, and so let's actually remove this. Go back to the first one since I have all the filters applied. And to do that, I will just add this in, the site hierarchy, we'll add that in and remove all the non-hierarchical things, or the ad hoc hierarchy would be a way to refer to those. So now I have the official hierarchy and incidentally, you can remove any level if you prefer to jump from say Site Class to Site Place and directly into Movie Name. So even though you've established the hierarchy, it still gives you the ability to manipulate this. So let's go all the way up to the top. This up arrow collapses all of the levels. And other options allow you to drill down. So when I point to this down arrow once I'm at the very top, there's a tooltip up here that says Click to turn on Drill Down. So when I do that, what that means is I can double-click on an item, and then that takes me down to subsequent levels. So that's another way that hierarchies are helpful for exploring the data. Now the other thing that you might want to try, and it just depends on the structure of your data, is this icon with the two arrows, which says go to the next level in the hierarchy. So in other words, we're going to go from Site Class to Site Place, and we totally lose the Site Class. and so I have no idea what those things are. It's the equivalent of removing Site Class from the hierarchy except that it's still really there. I can drill up to return to that level. So just three different ways of working with your hierarchies. And this also works when you have charts as well. Another option that you might want to consider is working with hierarchies on columns. So I'll just make a generic type of hierarchy here. I'll just add IMDB to the columns just to illustrate the working with hierarchies on both rows and columns, one as a fixed hierarchy, one as an ad hoc hierarchy. But we get these drill symbols still. They just are superimposed on our column headers, which is kind of hard to read sometimes. But basically, they operate on whatever you have specified in this drop-down list. So we can work on columns and drill down, or we can operate on rows and drill down. So that is how we model our hierarchies so that we can explore data in different ways. The main rule here is that a hierarchy must include only columns from that table. You would not be able to add genre into this hierarchy over here in the Movies table.

Summary

There you have it. Some more enhancements for the data model that we make without the Query Editor after we've shaped and cleansed the data. Now as part of the modeling process, it's important to have a clear understanding of how the tables relate to one another, not only which columns are related, but also the direction and cardinality of the relationship. That way we can set up the relationships or correct the auto-detected relationships so that everything works in your data model correctly. We also reviewed how to recognize the kinds of problems that you can run into if the relationships are not set up correctly and how to fix them. And last, we added a hierarchy to the data model so that we have a convenient navigation path that moves us from summary to detail data in a visualization. In the next module, we'll continue making some incremental changes to our data model to keep improving it for better reporting.

Improving the Data Model for Reporting

Overview

Hi, I'm Stacia Varga, and welcome to another module in the Modeling Data in Power BI course. So far, we've done a fair amount of work in preparing our model for reporting, but we're not done yet. There's definitely still more to do. This time our emphasis is on how to improve the data model specifically for reporting. That means we're going to spend time on certain elements of the data model that affect the user's experience with reports, whether the user is a developer or a viewer of a report. First, there are several properties in the model that affect tables and objects and ultimately control the experience of the report development process in general, as well as specific Power BI behaviors or visualizations that viewers of your report will see, as well as specific Power BI behaviors with visualizations. Then for certain types of data, we can assign fields to data categories, and that affects how Power BI displays that data in a report. And then the last model enhancement specific to reporting relates to the Power BI Q and A feature, which is a way to create a visualization by asking a question rather than using drag and drop to add selected fields to your visualization.

Configuring Model Properties

Although a significant part of building out your Power BI data model takes place in the Query Editor as you learned in the previous module, there are some other modeling tasks that you perform outside of the editor. We'll call that the Model Editor in the Power BI Desktop application. Now in the last module, we looked at how relationships between tables and how hierarchies between fields in a table affect how data displays in a visualization. Now it's time to review the model properties that affect what you see in the Report Designer when you're building a report and how you can control what Power BI does with data that you add to that visualization. First, you can configure metadata for model objects, specifically tables or fields in a table. This could be as simple as renaming these objects, which, of course, you can do in either the Query Editor or the Model Editor. And you can also add descriptions to your fields to make it easier for people to understand what a particular field represents. Something you might consider is to hide tables or columns/fields, those terms I'll use interchangeably. But by hiding them, you can simplify the model overall and make it easier for yourself or others to find the objects that need to be added to a visualization. Some of the tables or fields in your model might only be necessary to set up relationships or maybe to build larger datasets or maybe to compute column values, but they're not useful as individual objects to add to a visualization. You can hide them. Sometimes you might need to adjust the sort order, especially when you have columns that you don't want to sort alphabetically, such as months in the year. January, February, that's not alphabetical. Or maybe sizes, like small, medium, and large. So you can set up one column that defines the sort order and then associate that column with the column to be sorted. Another common task is to fix up data types. Power BI takes a best guess at data types and categories for each field in a table, but it's not always accurate. Of course you can specify explicit data types in the Query Editor, but you can also do this in the Model Editor. To get a visualization to behave properly, you need to look at the data types individually over and over and over again and make sure nothing changed and comes as a surprise to you. Importantly, if you have numeric columns, which you likely do because that's what we're doing is summarizing and counting and so forth. And so for these numeric columns, you should fix the formatting so that you get commas or currency symbols when you need them. Or maybe you need to set the right number of decimal places. And then lastly, Power BI wants to sum up numeric values, and it sets that behavior by default. But sometimes that's not the way you want to summarize values. And sometimes even though a column contains numeric values, maybe it's not appropriate to summarize those values at all, such as when you have an identifier number, like the movie ID or the user ID in our movies data model.

Renaming and Hiding Columns and Reviewing "Sort by Column"

What are the things that we need to configure in this model to clean it up a bit better for reporting? Well the main things that I look for as I polish up my models is I want to make sure that the names of tables makes sense to myself or to the users. And depending upon your standards, maybe you want everything to be capitalized, or maybe everything should be lowercase. In my situation, I prefer to see the capitalization. And a lot of this I already took care of in the Query Editor when I set up this model, but you can see that I missed a few. And so just as an FYI here, we can just click the dot, dot, dot, the ellipses button to the right of each field name, and we could rename it right here inside of the Model Editor as well. Now the net effect is the same. As the end user, you can't really tell where that changed, whether it was here in the Model or whether it was in the Query Editor. But the benefit in my opinion of making these changes if you were to go into the Query Editor instead would be if I want to reuse any of these items. So we learned about the Advanced Editor in one of the previous modules. I could just copy and paste these steps into another Power BI file and essentially reproduce these steps without having to do them one by one by one, which is how I created these in the earlier module of this course. So in that case, my preference is to do all my renaming and capitalization here in the Query Editor. That way I've got it done once, and I don't have to go through and clean it up. But it is possible to rename things over here in the model side. Other things that I'll look at is what makes sense to expose for reporting usage versus hide things. So for example, I might want to look at the IMDB ID, consider that for hiding. So I just again click that little button, and I can click the hide, so now it goes away. It isn't removed from the model. It's just hidden from me so that number 1 at the end of the day here, I have a smaller list of fields to choose for adding to a visualization, and some things just aren't particularly meaningful to me, like Movie ID means absolutely nothing to me; whereas Movie Name is much more useful. So I'll go through and hide identifiers. Those are probably the most common things that we want to hide. We can not only hide fields, but we could also hide tables as well. Maybe we use a table just as something to contain some data that we're going to use in calculations. But in any event, we can clean up our model just by hiding things that we're not going to find useful. So for example, to eliminate confusions, I might hide the Genre column here or field as we might refer to it in the model. Again, that's an interchangeable term that you'll hear me use. So we clean things up and decide what we want to see and not see. I'm also, in addition to all the IDs, I'm going to hide Site Place here because I'm going to use some other things as we'll see later in some replacement actions that we'll do. So let's go ahead and hide that. Now if you hide things, they're not really gone as I mentioned. We switch over to the data view, we can see they're still there. They're just dimmed to indicate that they're hidden. I can always unhide everything. I can also just say view the hidden and toggle back and forth between whether I want to view the hidden elements or not. And actually even if something is hidden, I'm still able to add it into a report. So if I select this visualization here and then select Movie ID, then we can see it is useable. It's just a convenience factor to hide those things that I think aren't going to be useful to me in the report development process. The other activity that we could perform is to set a sort order on items. And I don't really have a good example right now in any of the columns. That will actually make more sense later when we start working with calculations and so forth. So we'll come back and revisit the sort order later. But in general, what you would do is select a column and then go to your Modeling tab of the ribbon, and then you would select the Sort by Column button in the ribbon and then identify some other column in the same table that you would be using for sorting purposes. But again, we don't have that situation. A very, very common example would be to set a sort order for months to sort them in order 1 through 12 for January through December so that months don't sort alphabetically if you were going to be displaying data in that way.

Configuring Column Data Types and Formatting

The other changes we would want to look at would be data types. Again, this is something that I might be more inclined to set in the Query Editor. One example is where we have Runtime Minutes is set up as text right now, but I want that to be numeric. And watch what happens if we change this. It says you're about to change the data type, which means your data is going to be stored differently, and you may lose some precision if you do that. So you can decide whether you want to say yes or not. I'm more inclined in this situation to just go ahead and go back to my query and adjust the data type for Runtime Minutes here to the whole number. And then, of course, we'll need to close and apply as soon as this finishes updating itself. Now we can do the close and apply. And as we've seen before, if you followed along through the previous modules, there is some dirty data in here so we'll just ignore those things for now. We have enough good data to do what we need to do in this module. My next step would be to look at formats of different types of data. So for example, let's go back to our report here, and I'm going to pull out things. We can see Movie Count for example is showing up fine. And then let's add a couple of other items, such as Rating or some dates. And notice what happens when I select a date even though if we look at the Ratings table, there's a date listed here. One of the behaviors of Power BI is to break this out into Year, Quarter, Month, and Day automatically. That data does not exist naturally in the data, but it's smart enough to pull this stuff all out. If you prefer to see the individual date, all you have to do is click this little down arrow and switch it over from Date Hierarchy to Date, and then it will show up in the format that is specified for that column. So I've got counts, and ratings don't want to show up. Not everything has a rating, but there must be some ratings in here because we can see a value. One quick way of seeing those would just be to flip this around, and we can see these values. So we can just sort to put the blanks on the bottom. So the whole point of this is to say we've got values that have formatting. So anything that has a numeric value, such as a rating as an example, might be something that on the data side, in other words go to the data page of your model and select the column that you're interested in, and you can see the data type that's associated with it. This right now has a decimal number that's set by the Query Editor, but we can't see the format here. It's set to General. So in the model, we now want to say this should be reflected as a decimal number. And maybe we only want one decimal place. We can configure that formatting just like you would in an application such as Excel to apply very specific formatting to this. Similarly with Date, Date right now is showing the day of the week, along with the month, day, and year number. So if I go to the Format drop-down list and select Date Time, I can see all of the possibilities for formatting things. So since I'm American, I like the month, day, year, year, year, year format. So whichever way you prefer to view the data, that's what you can set here. So I have Date in the Ratings. I also have Date in Tags. And here's an example of where I might want to differentiate between these two dates so that it's less confusing. Let me go ahead and change my format here. But I might want to rename this as Tag Date and Rating Date. And I forgot to hide the Movie ID over here and hide the Tag, Movie ID, and the User ID and Tag. Alright, so I have Rating specified. And then Movie Count, if I look at my counts, which were in People and in Movie, I can check that it's a whole number, and I might want to go ahead and apply a comma as a thousandth separator in that column as well for both people count and movie count so that if I have a large number, it just looks a little bit cleaner. But as a whole number, it will have no decimal places.

Configuring Default Summarization

And then the very last thing is summarization. So right now if we summarize Ratings, we can see that there are ratings that are very high. If there's a rating system of 1 through 5, why are we getting 422.5? And the answer to that is is right now the default summarization method that's being used for rating is a sum function, and we don't want that to happen. So what we can do is in the model, we can come to Rating, and we can change this default summarization to Average, that that might make more sense. So if we go back to look at the report, actually it didn't change it because it was already set to Sum of Rating when we added it. Now when we add the rating, sorted in descending order, we can see the average by date for certain values. And if I remove the date, which notice the new formatting, then I can see the average rating individual movies overall dates. So either way is a valid answer, but it's no longer summing up all the individual rating scores. It's averaging them. Similarly with Runtime Minutes, if I add this to the report, then what I would like to see is not a sum, but an average. And when I have an individual movie, then it's only summing up one row, so the answer is correct. And what we're seeing is a total of Runtime Minutes for all of these movies. But what if I was doing something like showing movies by year? Oh, which brings me to another point. Year is something that I don't want to summarize at all. So if we go to the model, and we go to year, we can also specify that a numeric value should not be summarized. And that would be true for year and also if we were going to be using things like User ID should not be summarized in the Ratings, as well as the Tags. So back to this idea of reporting by year and then doing Runtime Minutes, maybe I want to see an average instead of a sum of these individual values. Let's make this bigger. So it looks like each of these things came from a different movie so, or a different year, so that's not going to be a great example. But what happens is if we say we really want the Runtime Minutes to be displayed as an average as opposed to a sum, we can just set the default summarization. And then when we add this to a report, we can see the individual years still show up with the correct values. The difference is their grand total is the average as opposed to what we see with our individual movies showing up. So if I take Sum of Runtime Minutes out of there and add in Runtime Minutes, it will still be correct because we're averaging for one movie. If we had multiple movies per year in the second visualization, these values would change. But our overall total is going to reflect the average of the Runtime Minutes there. So those are all the different kinds of things that we want to do to continue to refine our model just by configuring various properties that we have available to us. And most of that takes place on the Data tab here by using the commands available on the Modeling tab of the ribbon.

Using Data Categories

Now as we've seen to some extent already, Power BI uses information in the data model to decide how to display data by default, such as aggregating numeric values and displaying the result in a chart. In some other specific cases, you can add some additional control over data behavior by assigning a field to a data category, and there are three types. The first is geographic location. Now Power BI is usually pretty good at detecting a city or a country, but sometimes it needs a little help. So by assigning a field to a type of geographic location, you can eliminate some ambiguity. Another type of data category is for URLs. There are URLs for web pages and URLs for images. And for both of these types, the locations and the URLs, I'm going to show you how to work with these in the next demonstration. The last category that we have is for barcodes. In that case, you need to put data into a column that relates to a barcode, and then Power BI uses this data as a filter, but this only works when you have the Power BI mobile app on a mobile device. And then you'd literally use the camera on your mobile device to scan a barcode on a product or maybe on a document to display your Power BI report for that scanned item based on its barcode. In other words, you're scanning to get a filter set in your report. Now I'm not going to show you how that works, but I have included a link on the resources page at the end of this module if you want to learn more about this feature.

Preparing Data for Geographic Data Categories

So now let's look at how to work with data categories in our model. So I'm going to start by showing you what happens now without any data categories assigned. We're going to look at geographic locations first. So if I were to do something like pick up Site Name, and we'll make this larger so you can see this. Basically, I get just a list of things. And because this was part of a hierarchy, it went ahead and pulled in a bunch of columns for me. So when I click on this visualization, I can clear those other times. Maybe I'll add in Movie Count so we can see. As I scroll through, I can decide what I want to look at. So let's say that Site Name is preferable here. So I have a bunch of names. And if I try to map this, I need to literally move Site Name from Legend up into Location, and then it says oh, okay, I think I know where to place each of these little bubbles. And these bubbles represent the movie count. So we have this odd thing called Aegean Island that's actually over there by Iceland, and we have Spitzbergen, which is up in Scandinavia here. Or I have Tacoma over in Washington in the United States. And there's just one movie listed for most of these it appears. And without going through every single little dot, most things probably line up where they need to be. But if you recall, when we look at the table, there's things that are all over the place. For example, I have rivers and islands and places like Acapulco and baseball fields or cities, and that's mixed in with things like colleges and so on. So there's not a very good separation of locations into streets versus cities and so forth. So one step that I would take here would be to clean up the data itself, do some data cleansing, which means we're going to need to go back to the Query Editor. And in this model what I'd like to do is add in some custom columns. So the first custom column I'm going to make is country. So let's take a quick look before I do that at Site Place. I can see in several places I have two- letter abbreviations, and those represent the states in the United States for the most part. And if we look at the data, we can see that's not always true, but mostly. I could say in all these cases, these look like, with maybe the exception of SA, they look like states in the United States. And anything that's longer than the two characters is something else, maybe a country. So we're going to make that assumption. It may not be a perfect assumption, but it's our starting point. So I'm going to create a custom column called Country. And I have in the snippets, so you can copy and paste this a formula that says look at the length of this column, Site Place. So Text.Length is the function that we use. And if that's equal to 2, then return the string USA. Otherwise, return the contents of Site Place, which most of the time is going to be a country. So we'll add in that new column, and we can see that we have several places where USA is listed and some things where it's not listed. And so it's probably not perfect, but it's good enough for our purposes of this demonstration. Now I'm going to add in another column, and this time I'm going to make a column called State and again copy this from the snippets file. So again, we'll look at Site Place, looking specifically at those strings that have two characters, and we're going to bring in those two characters. Otherwise, we're going to leave it blank. Now we can see the new columns show up, and, for the most part, it looks good, although we see here is a Great Britain that's not exactly correct. One thing I can do here is to say replace GB with Great Britain, and then this will fix eventually when we go to refresh the data. One more thing on one more column. This time I want City, and the formula I'm going to use is to look at the Site Class. And if it's city or if it's town, then we'll pick up the site name. Otherwise, we'll leave it blank. So I'm just basically trying to pick through these site places and site names to come up with my geographical structures by country, by state, and so forth. One of the things that might help is to combine city and country together. So we'll do one more custom column, and we'll do City, and we'll concatenate that with an ampersand and put in a comma and a space there and Country. And that will combine the two types of places so that we can have a single column to use for geographic data and eliminate ambiguity about where a city is located.

Assigning Fields to Geographic Data Categories

Now that we've added the custom column for City Country in the Query Editor, we're ready to use it. I'm going to take that City Country, and on the Modeling tab here, I'm going to assign that to simply a place. Then I'll use that in the visualization in lieu of city, and it doesn't look much different here. But one of the things that's not particularly obvious is that we have several combinations that are blank. In the visual level filter, I'm going to select the checkbox for Select All and then clear the Blank checkbox. And now we can see some changes in the bubble sizes. So for example, there's a very large bubble here for NYC, which stands for New York City. And if we focus in a particular country, like USA, we can see the variations in bubble sizes a little bit better here. If we change to a different country or combinations of countries, maybe Italy and Germany and France, we can see variations in sizes of our bubbles as well. And the bubble sizes are relative to what we see in the current level of filtering from the smallest to the largest.

Assigning Fields to the Web URL Data Category

Now let's take a look at working with the web URL data category. Let's just duplicate this page so that I can have a table that's already formatted here. And I'm going to remove everything except Movie Name. And I've unhidden the hidden fields here just by selecting the View hidden checkbox. And so now we can see the IMDB ID. We can create a custom column that concatenates this value with a constant string that represents the URL at the imdb.com website for these movies. So of course, we can go back to our query and add a column. We'll call this the IMDB URL. And then from our snippets file, we can copy this string, IMDB title, and then we can include the IMDB ID in here, click OK, and that will generate the URL for each individual movie. Be sure to select Close and Apply. And when this finished, click Close, and now we can add the IMDB URL to this particular table. But notice that it just shows up as text, and this is where the data category comes in handy. We can go to the IMDB URL, go to the Modeling tab, and set the data category to Web URL. When we go back to the report, now we have a clickable link. And as long as you're connected to the internet, then the link you selected will display. The URL for images works similarly. But instead of showing the link in your visualization, you will see the image rendered. Now if you want to save some real estate here, instead of having a whole big long URL, you can go to your formatting, and there's a switch here to display the URL icon. So rather than having the entire string displayed, you can just have this link image that is still your clickable link to the applicable movie. And that's how we work with data categories.

Enhancing the Model for Q&A

There's another feature in Power BI known as Q and A, and it's available both in the Power BI service, as well as in the Power BI Desktop application. And its purpose is to support natural language queries, such as which movie is longest as a really simple example. We'll look at some other examples in a moment. Now for this feature to work properly, the model has to meet certain requirements. So for example, it's really important for relationships to be defined correctly between tables, and other properties should be correct as well, such as the spelling of tables and fields and having data types set and having default summarization set and so on. Many of the properties that we covered at the beginning of this module become very important for Q and A to work properly. Also, it's important to normalize the model structure as much as possible. That means we need the information grouped into tables. Think of it like nouns, people, places, and things. Each major type of noun should be in its own table. So for example in theory, we could have one big giant table for movies that includes everything about the movies and the characters and the actors and the ratings and the tags, and it would just be one big, giant table, and Power BI can certainly find the distinct movies in this list and distinct characters and distinct ratings by user, and so on. But we have them separated out because Power BI behaves better that way, especially when it comes to the Q and A. So getting the data structure right is a big step in helping Power BI implement Q and A accurately, and it makes it more useful. But sometimes, Power BI cannot guess correctly. We have to help it along. We have to potentially enhance our model specifically to support Q and A. One way to do that is to add synonyms, which essentially expands the vocabulary about the model that Power BI understands. And if you really want to get in depth with enhancements, you can edit the linguistic schema that Power BI uses to translate your questions into targeted objects and structuring the queries. So let's take a look.

Adding a Synonym to the Model

Before we start changing our model to support Q and A better, let's see what happens without doing anything. I'm going to create a new page here and go to the Home tab of the ribbon where there's a button that says Ask a Question. Part of the trick to getting Q and A to work is just understanding how to phrase the questions. So if we do something simple like count movies, we can see a general number, overall count. And then I can say that I want to break that down by some category such as year. So I can type in by year. And you can see that here we have a line chart because I'm dealing with numeric values, and it understands year as time based that it sets up from 1980 to about 2020 and even 2025. Apparently, there's some movies out there with those dates in them. And we can see that the number of movies in this collection of data at any rate is increasing somewhat more rapidly from 2003 on up as compared to the earlier years. But you can see as I type, it starts to give me hints about what I might want to include with this particular question, and I can say not only count movies by year, but by genre. And so then I can compare. There's a quite a lot of genres here, but I could look at these individually. And it seems like the highest genre there is the genre that's not listed. So of course, I can apply filters here and continue to modify this particular visualization. Once I click out of the visualization, I no longer have access to the question. So asking a question is simply a starting point. And I could ask a question like show movie count. Again, it's starting to give me an answer to anticipate what my question is. Maybe this time I want to say by location. And here it doesn't know where location is. It starts looking inside of various columns to see if it can find the word location, and it found it inside of tag. And so it thinks it's being helpful to show me different tags with the counts associated with it, but that's not really what I was looking for. I could say show movie count by country, and that gives me a different answer, or by state. But if I just simply want to be able to say by location, I need to tell Power BI what represents a location. So that is where synonyms come into play where I can come up with different words for elements of my model so that it can be more useful. So let's go to the model, and lets' say that I want City Country to have a synonym. The access to synonyms seems like it should be in this Model Designer here, but it's actually over in the relationship diagram. If we go to the Modeling tab, there we have a button for synonyms. And then if I select the table, in this case movies, it will show me the various fields that I have in here. And I can go to City Country, click in that box, and it automatically includes a comma for me. So I can say location. Maybe I want to add in another synonym, so maybe site location. So I can put in all of the different ways that I might refer to this particular field, and then that's accessible to me through the Q and A. So now if I back up and say I want by location, then it knows that what I'm interested in is this site and country. If I hit Enter, then of course I can see what I had before. Let's resize this. And remember that we need to clear out the blank city countries in order to see the bubble relative sizes for the movie counts. So synonyms are a great way to enhance the Q and A experience just to use different terms. So maybe when dealing with movies, we might call them films. Or if we're dealing in a retail environment, we have sales or purchases and things like that that might reference the dollar amounts of our sales, so different ways of referencing existing fields. And that's the simplest way to handle improvements to your model specifically for Q and A.

Exploring Q&A Linguistic Issues in the Model

Now let's look at a more advanced feature. We'll add another page here, and let's ask another question. This time let's say I want to know the longest movies. And whenever Power BI says I don't really know what you want here, it just gives me a basic list of whatever the key object was, which, in this case, is movies. And so it just picked out some columns and says here you go. I don't know. And there's no real logic to what it produces. Or if I had really wanted to ask this question in a way that it could answer, I could say list the longest average runtime minutes by movie, and now I get an answer. But that was a sort of convoluted way to get to a result that would be useful to me. A simpler way to ask this question would be show long movies or show lengthiest movies. Or what if I want to see movies with the highest rating? So list movies with highest ratings. And it asks me well, which ratings are you talking about? And so here's a list, and that's fine. But what if I ask for list popular movies? And then Power BI says hmm, I don't really know what you're looking for, so this is my best guess. And popular isn't really a synonym for anything that we have in our dataset, and so this is where the notion of using synonyms to help enhance our model for Q and A is limited. I can solve these problems by working with what's called the linguistic schema.

Introducing the Power BI Linguistic Schema

So the first thing that I need to do is to go to my relationships page and go to Modeling. And then I can select the linguistic schema drop-down list and export it. And right now, the only language supported is English. So I need to export the linguistic schema and then save it. This is saved as a YAML file. That saves the definition to disk. And then what I need to do is to open another application, and that application is Visual Studio Code. I'll include a link to this on the resources slide at the end of this module. And after you install Visual Studio Code, you'll need to go to the extensions page, and then you can search for YAML. I already have it installed. But just so you know what you're looking for, you want to find the YAML Language Support by Red Hat. And you'll click on this, and it will give you a button to install. And after you do that installation, there will be another button here that says reload. And then once that's available, then you're ready to go. And then you can open up the file. YAML is similar in nature to JSON, but not exactly. And there's these object definitions inside the YAML file that are used by Power BI to match up words and phrases that are found in the Q and A box with tables and columns. So here we have a table called Movies, and it knows that the word movie relates to that particular table. And so the city refers to the Column City in the table movies and so on. And it recognizes that as a location. So if I say anything that's location related, it will try to serve up some information related to that. This is a fairly advanced topic, and so I'm not going to go into all of the details about how this works. But in order to answer those questions that were not answered properly, I'm going to go down to the bottom of this file. I'm going to add in a few things just so you can see what this looks like. So for example, I'm going to say for the popularity aspect, that has to do with the ratings. So in the snippets file, I have some code for you to paste into the bottom of this file. We have phrasing where we associate relationships and adjectives or verbs and subject object sorts of things into our schema here. And the structure looks sort of like this where I have a table called ratings that has a column called rating in it, and we'll say that rating has a rate. So rate, there is no rate in there, but what I'm essentially saying is I have an adjective. I have a high rate or a low rate. I have popularity or not popular, unpopular, worst, best, and so on. We just give this a phrase that relates this concept. And then notice that there's a warning here. It says unexpected property, and this is the thing that's not particularly obvious. But position matters when we add things to the YAML file. So you notice how these other objects in here are indented a bit. If I just add a couple of spaces in to align rating\_has\_rate with these other objects, then that warning goes away. So we give it a name that's unique inside of this file, and we're relating these two ideas, rating and rate. We're binding it to the ratings table, and we're saying that this is a type of phrasing, a very specific type of phrasing as an adjective. And there are rules that say when you define an adjective, you have to say what subject that adjective applies to. So this is a word that exists already that Power BI understands. It's been defined in the linguistic schema as something that belongs to the movies table and the movie column. And then we give it a list of one or more adjectives that will appear in a question as an adjective with this particular noun. And similarly, we can provide antonyms for those adjectives. So we're basically expanding what Power BI understands about these words in the context of movies. And then we tell it what this is measuring and so it knows if that's a value rating high versus low. Power BI knows that high is an increasing number and low is a decreasing number. And so by adding these things in in the structure that it needs, then Power BI can interpret our questions more effectively.

Editing the Linguistic Schema

Here's another example that I can give you is thinking about the movies that have awards. And so we can paste in another type of phrasing. This time this is a verb phrasing where we're saying that a movie can win an award from an awarding organization. So the term win is a verb in this case that applies to movies, and the object of that verb is an award. And there's a prepositional phrase here using the word from. So movies win awards from awarding organizations. Now my question doesn't have to use all of those elements, but it's this context of phrasing that's important to Power BI, and it's able to interpret specific questions using this information. Now I don't expect you to be able to go and write this linguistic schema on your own, but I will give you a link to a very long article from Microsoft that explains about the different kinds of phrasings. There's more than just these two, adjectives and verbs. So once you have things structured properly, and probably the biggest challenge that I found as I started working with this is just getting things lined up in the proper places. So if you don't have a particular element in the proper column, then you get errors like we see here. And one error in one definition makes it look like you have errors in other definitions. So there's a ripple effect here. So be careful with your spaces and look at other examples to get your spacing correct. So we've added in two kinds of phrasings here. I'm going to also update the movies phrasing for the runtime\_minutes. So if we look, we can do a find here. If we look for runtime\_minute and just see where it exists in here, we see movie\_has\_runtime\_minute. There's an attribute, phrasing, and just words that it's associated with. So there's really nothing here that talks about length. So if I want to be able to ask the question list longest movies or show the lengthiest movies, I need to add another adjective in here. Just to make sure, let's see if we have long. Long doesn't exist anywhere, so let's add in another definition here where movie has length. It's based on the runtime minute. We can say long or short. We could even say lengthy to have multiple adjectives. And when we're done, we just save this file. And then back in Power BI, we'll use the linguistic schema drop-down and import that schema. And it says everything's pretty successful, but we have a movie\_has\_runtime\_minute relationship, and we're going to merge that with your movie\_has\_length. So if we look for this movie\_has\_runtime\_minute, I don't have that in the existing YAML file, but Power BI inferred that from what I had created. And if we were to re-export, we'd see things merged together. So now let's see what happens when we start fresh. I'm going to remove this and ask a question. Let's start with list longest movies. And now we get an answer that matches the very verbose question that I had asked earlier, list movies by longest average runtime minute. So a much better way of asking that question, and that was facilitated by updating the linguistic schema. And let's try that question related to the awards, list movies that won the aa award in 1981. And if we expand this out so we can see it, we can see we have an answer. So by adding a verb phrase, we were able to extend how Power BI can interpret our question and answer it. And one more example is the list the most popular movies in 1992 in USA. And we can make this bigger by adjusting the text and stretching out our chart. So there you go. That's how we can enhance our model for reporting by using the Q and A feature.

Summary

And there you have it. Our data model just keeps getter better and better. In this module, we reviewed about all the different types of properties that you can configure for the data model that control the formatting and general appearance and behavior of data as you add fields to a visualization. And we explored the ways that data categories affect and improve visualizations. And we experimented with the Q and A feature and explored the effect of adding synonyms and adjusting the linguistic schema to get better results. And that pretty much covers everything we can do with a model without requiring the use of DAX, the formula language for Power BI, which is what we're going to cover in the next module. Meanwhile, here are some links to more information about working with barcodes. This feature is pretty straightforward, but you do need to work with specific barcode formats that are listed on this web page. And here's a link to learn more about editing the linguistic schema for Q and A. There's really quite a lot to this process that you need to study carefully in order to implement it effectively. And, of course, you will need to download and install Visual Studio Code, which you can find at this link, in order to work with linguistic schemas.

Using DAX to Enhance a Power BI Model

Overview

Hi, I'm Stacia Varga, and welcome to this module of the Data Modeling in Power BI course. Now that we've covered all the modeling options possible in the query editor and the model editor, we're ready to start using DAX to continue enhancing the Power BI model. If you don't know what DAX is, I'll start this module by providing you with a brief introduction to DAX, and then I'll show you how to use it to build out your model even further by defining new columns. There are many different reasons to add new columns to your model, but I'll cover the three most common in this module. First, concatenating columns enhances your reporting, say, for example, that you want to combine a first name column with a last name column to create just one column with both names together. That's a nice way to save some screen real estate. Another reason to define a new column is to translate a value. In other words, you might have a code that needs to be converted to a term or a phrase to make its usage more understandable to readers of a report. A simple example is to translating F to female and M to male, but you can certainly handle more complex translations and even use conditional logic. A third reason that I'll cover in this module is to perform a lookup to a related table. And the reason for doing this is less obvious, and it's not always necessary, but there are times when your model's simpler when you can combine columns into as few tables as possible. This is called denormalization if you're used to working with relational data. Now a major reason to use DAX in your model is to create measures that will be useful for analysis. Measures are the numeric values that represent sums or accounts or averages. These are the values that you want to report for a point in time or compare between different points in time or compare between categories, as just a few examples. DAX is also useful for some additional interactive features in your report, such as filtering or for manipulating values for hypothetical scenarios, so I'll show you how you can set up DAX expressions that use the values set for a parameter so that you can display the results in reports.

Introducing DAX

The ability to use DAX is an important skill to have for building great Power BI models. It stands for Data Analysis Expressions, and it's the key to analytical capabilities available in Power BI. You use it to add new columns to your tables inside the model editor and not at the query editor as we did at the beginning of this course. That language is known as EM. You can also use DAX to define measures, which are the numeric values that represent sums and counts and averages and so on. These are the values that show up as columns or bars or lines in your charts and are vital to insightful visualizations. If you have experience with using Excel functions, you might think you see some familiar functions when you start to learn DAX, but there are definitely some differences even though they look similar. The primary difference that you need to understand is that Excel functions operate on individual cells. Cells contain values, and you can reference those values by using a variable for the cell, which represents its coordinate, right? Cell B2 is the second column and the second row. So we can reference those cells in formulas that use operators like addition and multiplication, or we can reference them in functions. You can also operate on ranges of cells where you specify the beginning and the endpoints of the range. DAX functions, on the other hand, operate exclusively on tables or columns, and that's it. So, for example, the COUNTROWS function here looks at the entire table. Or DAX can operate on columns. So with this function, we identify both a table name and a column name. And that's it. The big learning curve with DAX is learning to think about applying operations to just tables or columns. There's no access to individual cells.

Defining New Columns: Concatenating Columns

You use concatenation when you need to set up data from separate columns into a better structure for reporting. That is, instead of creating a table that looks like this, you might want a table that looks like this. Now normally I try to do this type of work in the query editor, but you cannot do that if you're using DirectQuery to bring your data into the model. Or maybe you just find it easier to use DAX to do simple data transformations. It's really your choice. The net effect is the same. What we're going to do is just copy one of the existing pages so that I have a good starting point with formats of the tables. And I'm just going to clean this up a little bit. I'm going to remove one table here, and in the second table, I'm just going to remove several columns, and I'm going to add in people. We have First Name, and we have Last Name. Now that we have those in here, let's go ahead and remove the Movie Name. So we're just filtered down to a couple of movies so that we can see some people. And so we have this list, and it seems to be sorted alphabetically by first name here, and maybe just to save some real estate or make things list out more nicely, or maybe I want to be able to search for an entire name, if I set up a filter or slicer, any number of reasons why I might want to concatenate these values, I need to do that in the model. So we'll go to the Data page to open up the model editor. I'll select People here. And it's quite simple to add a new column. We just go to the Modeling tab of the ribbon, and I can click New Column here, and that opens up the formula editor here. And incidentally, we could click on the ellipses button over to the right of the table name and add new column, and it does the same thing by opening up the editor. So we need to give the column a name, so I'll call this Full Name. And then after the equal signs is where we need to put our DAX, and remember that DAX works on columns or tables. And in this case, what we're trying to do is to combine two different columns, the First Name column and the Last Name column. And the way that Power BI recognizes that we're about to provide a column name is that we use a bracket. So if I type a bracket, it now shows me a list of all of the columns that are available in the People table. So if I start with First Name, and then you might think some languages allow you to use a plus sign, so you might think that that's available here, so if I type First Name and then for strings, when I want to insert a space here, we use double quotes and then a plus sign, and then I'll do open bracket so that I can select Last Name. And using the drop-down list here for the column names helps ensure that you don't get any typos into your expressions. When you're finished, you click the checkmark here to commit that column, except that there's one little problem. If I don't have a correct expression in terms of syntax, it tells me what the problem is. So in other words, it's saying here, looking at the first row, that it has a value that it's trying to convert from a text data type to a numbered data type. And why is it trying to do that? Because I used a plus sign, and it assumes that when I'm using the plus sign that I'm trying to add two numbers together. So, oops, that won't work. When we're dealing with concatenation of strings, we need to use an ampersand. Now sometimes, when we have numbers that are contained in a text field and I use the plus sign, Power BI is smart enough to make that conversion. If it can convert the number 1 that's found as a string into a number, it'll just go ahead and do that. But if it truly is a string value and I use the plus sign, it says no way, can't do that, try again. So now that I have replaced my plus sign with ampersands, Power BI goes and calculates the results of this expression row by row by row by row, and we have now expanded the footprint of this model and memory. Because this data is instantiated into memory, we have effectively made another copy of data that's being held in memory for Power BI to use. Then when we go look at our report, we can add, make sure the table is selected, we can add in the full name, and notice that there's an icon to the left of Full Name to indicate that this is a calculated column as opposed to a column that's coming to us directly to us from the query. So when I check the box here, I can add it in, and if I remove First Name and Last Name, we can see our full name here. The default behavior of Power BI is to sort by this first letter, and if I want to sort by something else, then I can specify that. I had mentioned this in our previous module, but we didn't have a good example yet for using this feature. But if I wanted to sort by last name, what I do is point to the column that I want to configure this property for, there's a Sort By Column in the ribbon, and let's say that I would prefer to sort my full name values by the last name of individuals instead of the first name. We go back to the report, and now we can see that values have changed the sort order here automatically based on the last name.

Defining New Columns: Translating a Value

You can also use a calculated column to translate a value into something more meaningful in a report. The age of a person, whether it's a customer in a sales database or age of an actor in our movies database, might be easier to process and understand than trying to do the mental math to calculate an age from a birth date. So our DAX gets a little more complicated in this scenario. Here is the formula that we would use, and this is using conditional logic. In fact, it's using nested conditional logic here. So back to our report where I've renamed this tab to DAX Concatenate, but I'm actually going to update this one more time and say, & Age. And so we need to go back to our model and update this model by adding in another column for age. I'll go ahead and add a new column here. And the important thing to understand about calculated columns is that we're defining an expression once, and then it automatically applies to every single row in this table. And hopefully it works for every single row, but it's possible that something will fail on individual rows, in which case we'd have to fix that formula. I'm going to paste in this expression from the snippets file that's in the exercise files. And so you can see this is just one big long string, but sometimes when you have long expressions like this, you might find it easier to come in and lay it out with some white space in there. White space is permissible, but you do need to use the Shift+Enter key to start a new line. So I can do Shift+Enter and maybe add in some spaces, so that's my first argument, and then I can Shift+Enter a couple times to isolate my second argument and then Shift+Enter again for my third argument, which in fact is a nested IF condition. So again, Shift+Enter, Shift+Enter, and Shift+Enter. And sometimes I like to align my parentheses so I can keep track of where they belong so I can be certain that I have properly paired my parentheses. Our IF function basically takes three arguments, where the first argument is an expression that needs to resolve as true or false, the second argument is what we do if that expression evaluates as true, and then the third argument is what do we do when that expression evaluates as false. So in this case, what we're doing first is we're saying look at two columns, Birth Year and Death Year, and are those blank, or empty, or null, or however you prefer to look at those? So we're looking at both of those, and the double ampersand sign there says these are two expressions that both need to be true in order for this if condition to be true. So if both Birth Year and Death Year are blank, then the expression resolves as true, and the expression returns a blank value from our second argument here. If either Birth Year or Death Year has a value, then the third expression gets resolved, and there we look to see is the Death Year blank? Now, that may be incorrect data, we don't know with the condition of our data, but we'll make a decision here that if it is blank, we'll just use the current year as our starting point. So we have a couple of date-time functions here. We'll talk about date and time DAX functions in a bit more detail in a future module in this course, but the key pieces here are that we're using the NOW function to get whatever today's date is, and we'll use the YEAR function to extract just the year out of that date, and we'll use that as the value from which we are going to subtract birth year. Now if the death year does exist, we'll use that as the number from which we are going to subtract the birth year. So either way, we are going to derive, or translate, values from our Birth and Death Year columns into an age that we can display in a report. Another thing that you might find helpful, especially when you get into long expressions, is the ability to add in comments. So I can come in and just use a double slash here, and I can say this is my expression that must resolve as Boolean, and I can say for my second line, return this value when true, and we can say returns this value when false, and so on. So that can be helpful for understanding and keeping track of what you're doing in more complex kinds of expressions. So if we click the checkmark here, we get an error. What's going on this time? And the reason is because, I'm just going to copy this and hold it in the clipboard for a moment, the reason is that my Birth Year and Death Year are set as text data types. And I can try to change them to whole numbers here, and it will give me a warning that I'm about to change the data type, and I say yes, and then it says nope, can't do it. How am I going to fix that? I need to go to go back to edit the query and then go to the People table, and I can adjust these things here. So, a very irritative process when we build our models. It's not necessarily just use the query editor and you're done. You may find yourself going back and forth. That's a very normal development cycle. So now that we have this Age, our first set of people in here don't have an age. We can scroll way down and find people with ages to confirm that we do have ages and also look in the drop- down list for filtering. So all looks good. We can add this to our report, and now we can see ages for each of the actors in this set of movies.

Using Variables in a DAX Expression

Now I have a little tip for you. In working with DAX, we can use variables, and variables can be helpful when you have long expressions to make them simpler to read. Or maybe you're using a calculation over and over again in a very long expression, and if you set up that calculation in a variable, then Power BI can calculate faster because it only has to do that once. But this is a simple example of using variables here. So I'm going to do Shift+Enter here to make some space, and we define a variable by using the VAR key word. And just for readability, I'm going to pull out some of these expressions and make a variable. So I just give it a name and an expression, and then I can reference that variable in my expression, like so. Shift+Enter so I can say BlankBirthYear. So sometimes by using these variables in here, it just simplifies, fix my typo here, it just simplifies reading the longer expressions. And notice that, if you're a programmer, I don't have to specify any type of data type for my expression. I just set it up the way that I want. So there's lots of IntelliSense squiggles here acting like I have a problem, and that is because I need one more thing when I'm referencing variables. I need to set up those variables initially, and then the final expression that's going to be returned to my measure or calculated column needs to have the word RETURN and then the final expression where I'm using those variables. So this might be a little easier to read than having all these functions and parentheses and things that can trip you up if you leave out a parenthesis in the wrong place. So when we click Commit here, it does the syntax check, everything should be good, and that's how we use variables.

Defining New Columns: Performing a Lookup to a Related Table

Our third option for using DAX in calculated columns is to perform a lookup to another table. You're, in effect, borrowing values from a different table to store in a new column of your table, so you might want to do this if you're trying to simplify your model by moving related data into a common table rather than having users hunt through a whole bunch of tables to find the one thing that they want. So, for example, here I have the Award Givers table that had information about awards, and I have a Movies table that has an award type, and they're related on these highlighted column names, Award and Award Type. And maybe I want to bring in the name of the award into the Movies table because the Award Givers label on the other table implies that it's not the award itself, but rather the people or organizations giving those awards, so it might make more sense to store Award in the Movies table. And, by the way, another good reason to bring in a value based on a lookup is when you want to create a hierarchy in a particular table, but the columns that you want to include are in separate, but related, tables. We create a calculated column. In this case, I'm going to bring in a column from the Award Givers into Movies, and it's going to look something like this, where we define a column in our Movies table, we use the RELATED function, and point to another table, represented by Award Givers here, and the column that we want to get. Now notice how we reference tables versus columns. Columns always have the brackets around them; tables will have single quotes. Sometimes you might see tables without single quotes, but that's when they have single word names. If there's a space in the name of the table, then you must use the single quotes. So let's work on this task of moving data from Award Givers into our Movies table, which right now we have an Award Type, which contains various codes that correspond to award values over here in the Award Givers table. Now I'm going to do a couple of things. I want to do some translations first because I don't know what some of these codes mean, and sometimes what's found in the Colloquial column has what I want, but sometimes it doesn't, so I need to do some translation first. So I'm going to rename this column Award, and I'm going to call this Award Type to actually be more in sync with what we have in Movies. And that doesn't change anything in terms of our relationships. That name change preserves the relationship, and everything's just fine here. What I'm going to do next is I'm going to copy from my snippets file an expression. I'm going to add a new column here in the Award Givers, and let's open this up so we can read it better. So what I'm going to do in this particular expression is I'm using an IF function, and I'm looking at the Award Type column to see if it starts with an H because we have several here, some with asterisks, one, two, three, and four asterisks. So I just want to pick up the awarding organization in that case and concatenate that with the value in the Colloquial column or, and that's what this double pipe symbol stands for, or I'm going to look to see if Colloquial is blank, or I'm going to see if it is an empty string because sometimes it looks blank, and it really is just an empty string, or I'm going to see if that first character is a character that's numeric or a symbol or something that basically if I have certain conditions, I'm going to pull in the awarding organization; otherwise, I'm just going to use Colloquial. And in most of the cases, that gives me an Award column that gives me something that I like. Here's one where we don't really have a value in either of those columns. I'm going to just accept that for the time being. But you can see that now I have an Award column that, for the most part, has something a little bit more meaningful. And now that I have that defined, now that I can go to my Movies table. I can add a new column. This time I'm using the RELATED function, referencing that column that I just added to the Award Givers table. Click the checkmark. And not every movie has an award, but we can see that we did indeed populate this particular column with values. Incidentally, let's say that I had wanted to type this out. If I know the function name, if I start typing a few letters of that name, the drop-down list here gives me some clues as to the names of those functions that are available, and it will tell me what it does. So if I just use my arrow key to move up and down, I get some help defining that particular function. And if I double-click on that function, then it says, okay, I need to have a column here, and so it shows me what I might want to use, and then I can double-click on that column and then close the parentheses. And notice that it tries to match up parentheses by highlighting them so that I can visually confirm that I have all the pieces that I need. And so now that I have brought over that value into the Movies table, I can hide it from my Award Givers table. And I'm also going to hide these other columns, Award Type and Colloquial.

Creating Measures

The next thing that you need to do for your model is to add in measures. Now we already have a couple in our model, such as the Movie Count measure, which counts all the movies in the Movies table. Measures are created by using DAX to calculate and summarize values for your visualizations. Now we can create all kinds of measures for the current model, but let's just focus on a few to get a general idea of how DAX formulas work when we're creating measures. This one uses the COUNTROWS function, which requires a table as an argument, the Movies table. Power BI literally counts up every row of a table. You can combine multiple functions together by using operators, such as a subtraction operator here. So, in our model, not every movie has an award, so if I want to count how many movies do have an award, I can figure that out by using a combination of functions. First, I can use the function that counts the total number of movies, and then I can subtract out the rows that are missing an award value. For that, I use the COUNTBLANK function, which uses a column reference, Movies Award, instead of a table reference, Movies, so that I can specify which column to look at when determining whether a row is blank. Now notice I reference COUNTROWS('Movies') in the Movie Count measure and COUNTROWS('movies') in the Award Count measure, and one of them has single quotes, and one of them doesn't. In both cases, that's a valid syntax because there's no embedded spaces in the word Movies, so it's optional whether you want to use the single quote or not. Here's a third example, which is a common calculation in analysis where we compute percentages of a particular category. So here the example is to calculate what percentage of movies in our data model have awards. So we can use the measure that we already defined, Award Count, and we define it just by using brackets just as if it were a column in a table, although in this case, it is a measure. Measures differ from calculated columns in that they are only calculated when you include them in a visualization, such as a chart or a map. And although a measure can belong to a table, we are not required to list the table that it's found in when we use it in a calculation. So a measure does not add to the memory footprint required for our model, and it does not require specification of the table name. Now it doesn't save us any time whether we used the full calculation, the COUNTROWS('Movies') - COUNTBLANK Movies Award in our third measure here. That doesn't save us any time to reference it by name. It just makes it cleaner to read, but the calculation time would remain the same. So we have Award Count as the numerator of our calculation, and then as the denominator we have an expression that uses the CALCULATE function. The CALCULATE function takes an expression in the first argument, which happens to be another measure, Movie Count, and then the second argument is what we refer to as a filter that we're going to apply, or in this case, the ALL function I call the unfilter. And what this is saying is use the Movie Count, but remove any filters that might be affecting movies when you do this calculation. And that can be a little confusing initially. CALCULATE is a very powerful function, and it relies on filter context, which is something we talk about in the final module of this course, so just take it for granted at the time being that what we're doing is just saying calculate movies without applying any individual filter context in Movies, and we'll see a little bit of what I mean by this in the next demonstration.

Adding Measures to the Model

Let's add a few measures to our data model. I'm going to duplicate this page to give us a fresh start here, and we'll call this DAX Measures. And I'll pull Movies back into my table and pull out the names. And we already have this Movie Count. When I click on Movie Count here in the report designer, the formula editor opens up and shows me that expression used to represent this measure. So in this case, we're using countrows(Movies). And this is a sneak preview of filter context because it's not counting the rows in the entire table; it is counting the filtered table. We have Page level filters here that is limiting what we see to a specific set of movies. That becomes the new table that this particular expression applies to. So we have this expression. Now we're going to add in a new measure. I have a DAX measure that I want to create. We'll just paste this in from our snippets file, and we're going to call this Award Count, and here's where we combine two different calculations by using the subtraction symbol. So we count up all of our movies, and we subtract out any movies that do not have an award. We click the check button to validate this expression, and it works. And I can add it to this table, but none of these movies have awards. So, what I'm going to do just to set this up for easier viewing is I'm going to add a visual filter so that we're not looking at blanks, and I'm going to clear the movie filter. There's a grand total of 773 movies in here, all of which have awards because of the filter I set to explicitly exclude any movies that did not have an award. The Movie Count, even though the expression here says to count all rows in the Movies tables, the filter that I have for Award has been applied to that table so that we're only counting the results after the filter is applied. So let's make a copy of this table. So I'm going to take Movie Name out of this, and I'm going to remove our visual-level filter on this so we can see that overall we have 34, 889 movies in this data set, 773 of which have an award. So this Movie Count is filtered, this one is not. Now our next step is to create one more measure to do a percentage of Total. So really what I want to do is say what percentage is 773 as a percentage of all movies. And so when I create a measure to do this, the measure looks like this, where I have the Award Count, and I'm dividing by this Calculate with Movie Count, but this ALL(Movies) essentially removes any existing filters. So just for the moment, I'm going to remove the numerator here so we just see the denominator portion. So we're just going to calculate all movies without a filter applied. Even though the name still acts like it's a percentage, we'll just ignore that for a moment. So I'm going to go back to this original table, make sure that's selected, add in this percent, rearrange a few things here, and we can see that when I use that Calculate on Movie Count, but add the filter, which I refer to as an unfilter, I'm saying give me all movies, then this particular row that normally would have a filtered movie count of one and my grand total filtered of all movies in this filtered table at 773, that addition of the ALL(Movies) to my Calculate expression says ignore whatever filters we have so it can get the unfiltered version of Movie Count so that I can get a percent of Total. So that is the secret to percent of totals, is using this Calculate in there. So now I'll put the numerator back in and accept this. And the next thing I'm going to do is change this to a percentage format so that it displays properly in here, and we can decide how many decimal places we want to use. And so we can see that a very small percentage, 2.22 % of movies in our database, have won some type of award. So those are some examples of DAX measures to get you started with the types of things that you might commonly want to do as you start to explore data in your Power BI model.

Using DAX Parameters: Creating Query Parameters

For our final topic in this module, let's talk about using DAX in parameters. Now there are two types of parameters that we can use in a Power BI model. The first is a query parameter, which has an effect on the query editor only, and it essentially applies a filter and refreshes the data in the model. By using a DAX expression, you can add a title to a report page that displays the current query parameter value to make it more clear to the viewer how the data is filtered. Let's try out the query parameter. I've made a copy of our first page of this report and labeled it DAX Parameters, and I'm just going to pull out some of the data here, Runtime Minutes and Rating, and we're going to add in the year of the movie. So what I want to do is create a query parameter that filters our movies data set, in other words, the list of movies held within this file by year. So to do that, our query parameters are managed in the query editor, so let's go there. And in order to work with parameters, we have a couple of options. So when I click the Manage Parameters button and select New Parameter and give it a name, I need to give it a data type, and I need to tell it where the values are coming from to populate the parameter with a value. So, we have three choices. One is if the user can type in anything, just free-form text, we choose Any value, or if we want to type in hard-coded list here, and that's good when the list of values is pretty small. Or we can use a query. That means we have to have something available here in the query editor that can be used as a list from which the user selects a value. Now we've been working with tables so far, so we need to create a list, so let me show you how that's done. In my case, I just want a list of years, and to do this, I'm going to create a new source as a blank query. This gives me a query that I can name. We'll call this Years. So I'm going to open up the Advanced Editor here, and I'm going to replace the default blank query with my own query. The first step is to create a variable called Source, and we're going to define a set of values from 1980 to 2018. That's how we define a range of values. I'm going to take that value from Source in our second line and assign it to another variable called ConvertToList. So by enclosing source within the braces, we take that range of values and make a list out of them. That basically is going to be an array with a single column, and we're just basically converting data types in the EM language to come out with a final list of values. And this will be easier to see when we look at the finished query. And notice the icon down here is different to indicate that this is a list object. So if we look at the Source, where we initially specified our first step as a range of values, then the ConvertToList, you can see that the result when we put Source inside of a pair of braces that it returns a list here. If we were to click on that list, that creates this third step of actually translating that initial series of values into a list object that we can now reference in a query parameter. Now if we leave it as is, it's going to list from the earliest to the latest. I can decide to sort that if I want to and do a Sort descending and change the sort order. That's purely optional here. And then, to use this list, I'm going to go to my Movies table. I need to set up parameters in order to do this. We'll set up two parameters, one called Earliest Year, you can put a description in here, and then we will specify that this is a Decimal Number and that it's coming from a Query. In the drop-down list we select the query, and then we can put in a value so that there's some default available when the report opens. So there's my earliest year, and we can see the value. Click Manage Parameter to see it again if you need to make some kind of edits. I can click New from here and add in another parameter. Let's call this one the Latest Year and pretty much set it up the same way, except maybe we'll put 2018 as the current value. So now we have two parameters. Let's use them. We're going to go to the Year column in the Movies table, and here there's a Number Filters where we can specify a filter that uses a range, Between. So we can specify greater than or equal to some value, and rather than hard code the value in here, we can reference our parameter, and then it shows me which parameters are available. So I will choose my Earliest Year and Latest Year. So now we can add some flexibility to the contents of our data model driven by a query parameter. So this will take a moment to update. When it's finished, select Close & Apply. Click Close when it finishes. And in the next demonstration, I'll show you how to use these query parameters.

Using DAX Parameters: Using Query Parameters

And I'm going to rearrange here to specify a place for a title that I want to add to this page. First, I need to create a new measure, so we'll add this to the Movies table. And I'll select the New Measure option, and I'll paste in the code that I pull in from a snippets file. Let's take a look at this. It says the name is Title, and we're concatenating some strings, starting with Movies Released Between, a space, and then we have two functions here, the FIRSTNONBLANK function and the LASTNONBLANK function. So basically, the Year column will be filtered based upon the query parameter selections, and some boundaries are going to be established. So when that filter occurs, Power BI is going to look to see what is the very first year, the earliest year in that column of that table. And similarly, we have the LASTNONBLANK. So these are nice ways to find the beginning and the end value to return into this measure. So this is a nontraditional measure in that we're using text, and we need to be sure that because it's text we're using functions like this to pick off a single value. So, I'm going to save this. And then to use it, I'm going to use a Card visualization. We'll add in the Title measure into this visualization, and now I can rearrange it and set this up just how I need right now. This particular page shows movies released between 2001 and 2017 because the page filter is filtering our movies list even though if I were to go to another page, if it didn't have a filter on it, we would see the 1980 to 2018. So this is correct; it just doesn't reflect the filter on the entire data set. Now if I were to want to change those parameters, what I need to do is to go to the Edit Queries button, select Edit Parameters, and now I can say maybe I want to see just the movies from 2016. Notice that the sort order goes from the highest to the lowest, and maybe I want the Latest Year to stay at 2018. So we'll click OK, and that doesn't apply the changes yet. I still have to click the Apply changes button. And now Power BI is going to refresh the data set based on the new data ranges that I've specified, and then any page of the report that I go to is going to be constrained to movies between 2016 and 2018, as well as any other additional filters that might be on those pages. And so now we see our list of four movies is now reduced to two, and our title is updated.

Using DAX Parameters: Working with What if Parameters

A second type of parameter is known as a what if parameter, and this type of parameter prompts the user for some type of numeric input, which can then be used in some calculation. So, as an example, let's say you have a sales data model. You can use a what if parameter to experiment with different discount rates to see what the effect on the gross profit margin would be. Now let's see how to work with what if parameters, and to do that, I'm going to go back to our DAX Measures page, and I want to copy one of these tables, and I'll go ahead and just paste another copy of it here. And in this case, what we're going to do is remove some of the measures, and I'm going to add in Ratings. Now normally, a what if parameter would apply when we have something that we would normally want to experiment with. Maybe if I'm doing budgeting or forecasting and I have sales amounts or expense amounts and I want to experiment with different values to multiply against maybe last year's values to come up with some proposed current year values, and I could see how the results roll up to grand totals, and I can filter and so forth. But this particular data set doesn't lend itself well to any kind of what if analysis, so we're just going to use our imaginations here a little bit and say what if I have a numeric value and I need to apply some other computation to it, just so I can give you an example of how the what if works. So, we're going to take Rating as our base value, and we're going to multiply it by some number. I have to set up a what if parameter, and that's available to us on the Modeling tab. So a what if parameter is strictly measure-based. It effects our data model calculations only. It does not go back to our query editor or restrict our data set in any way like the query parameter does. So to create one, I click the New Parameter button here, and let's just call this Multiplier. And it can be one of three different data types. We'll just use a whole number. And we can set a minimum and maximum value. This is fine. And then we can allow values between the 0 and 20 in increments of 1. That's fine. And I can even specify a default and keep the option selected to add a slider to this page. So if we click OK, then what happens is we get this slicer that had a slider on it. So I can adjust values, and as I slide this, it's very hard to see this number, but we can go to the formatting page, and we can adjust font sizes, and now you can see better what that value can be. And if you don't want to see the slider, you can just have this box be a place where you can type in new values. So maybe I want 10 as the value, which doesn't do anything yet, but what has happened in the background meanwhile, if you notice, over here in our fields list, let me just collapse a few things so we can focus on that, a new object here was added called Multiplier. And we have two things. One is we have a calculated column here that says GENERATESERIES(0, 20, 1). That was that range of possible values, so my minimum, my maximum, and the incremental values in between. So there was a list of possible values in here. And if we go and look at our data model, we can see there's actually a calculated column in here with those values listed one by one. Also, there was a measure created that looks at that particular column and uses the SELECTEDVALUE function. How do we know what the SELECTEDVALUE function does? Let's just remove this for a moment and let the IntelliSense pop up, and it says it wants a ColumnName and an AlternateResult. So whatever I change in the multiplier box will be the value returned by the expression and multiplier value, but if there is no selection, then we have the default, which is set as 10. So that got created automatically by that creation of the new parameter. Now, to use it, I can come up with a new measure. So I'm going to paste this in from my snippets file, and we're just going to take that column and use the AVERAGE function on it. But now we have an explicit measure that we can use in this table, and it just has some slightly different rounding there, so we can change this to one decimal if we prefer. And now I want to create another measure where I use the multiplier, so we'll just call this Adjusted Rating. And so we're going to multiply the Average Rating by the Multiplier value in the other table, and we can add this to our table here so we can see that the Average Rating has been multiplied by 10. And so the purpose of having the what if parameter is to allow for user input, either by using the slider that we removed or by typing in a new value. And as soon as I press Enter with a new value, we can see the result of that change immediately.

Summary

There's so much more to learn about DAX. My goal in this module was to get you acquainted with DAX and how it's different from Excel even though it might look the same. And I showed you how you can easily update the model by adding new columns with DAX. We added columns to get the first and last names of people into a single column, and we also did some calculations to get raw data for birth year and death year of people and translate those values into an age. Along the way, we learned how to use variables to make it easier to work with complex DAX expressions. And we also took advantage of the relationship between the Award Givers table and the Movies table to move the name of an award into the Movies table. In other words, we did a lookup to a related table. And then we used DAX to create a variety of measures using simple functions like COUNTROWS and COUNTBLANK to determine how many movies have awards overall. And we also learned how to do a percent of total calculation by using the CALCULATE function in the denominator of an expression so that we could override any filters and compute the grand total of a value. And we reviewed how parameters can be used to capture user input and then used in some way in a report. In this module, we used two query parameters to set upper and lower boundaries for the range of years for which we wanted to see movies in our report, and we then created a measure that's not really a measure because it contains text, but, nonetheless, it behaves like a measure and relies on the effect of the query parameter values to display values in a title or a report page using the FIRSTNONBLANK and LASTNONBLANK functions. And we wrapped up with a demonstration of a what if parameter, which creates both a calculated column and a measure that lets you get input from a user in a report page and then use that value in some other DAX measure that you can display in the report. The example in my demonstration was a bit contrived, but it showed it how it worked. And a more practical example would be to use a what if parameter in a budgeting or forecasting report. In this course, I can only show you a few relevant examples, but you can build on these basics by taking what you've learned and applying it to your own data. A good reference to continue your learning is available at this link. And, of course, DAX can be the subject of an entire course itself, and we have one here at Pluralsight. I encourage you to take a look at this course and build on your DAX skills.

Working with Dates and Time in Power BI

Overview

Hi. I'm Stacia Varga here with another module in the Data Modeling for Power BI course. This module continues on from the previous module, which introduced the DAX language, but this time we use DAX to work with dates and time in Power BI. DAX provides a lot of different functions related to dates and time. Analyzing information by data is probably one of the most common activities I see people need to do in Power BI, but a lot of times the datasets that they are using do not come prepackaged with a Date table. But that's okay because we have DAX to help us with that. DAX can generate tables including a calculated Date table. By having a single table as a source for dates, you'll find it easier to compare date-related measures that come from different tables in your model. Next, we explore a few of the DAX functions available for working with date and time in different ways. We can also use DAX to create calculated columns as I covered in the previous module, but this time we'll use DAX to create some columns for the calculated Date table to give us more options for analyzing data across time, and then I'll also show you how to add a calculated column to sort months correctly. And we wrap up this module by looking at how to use DAX to apply time intelligence to our data model. If you don't know what time intelligence even means, stay with me through this module, and I will explain. Time intelligence is useful for almost any form of analysis that you need to do, so it's an important concept to grasp, and DAX makes it easy to implement.

Creating a Calculated Date Table

If you need to do comparisons of your data over time, and you don't have a Date table from a data warehouse, for example, then you're going to need a Date table. DAX makes this easy for you because you can use the CALENDARAUTO function to generate a list that contains every date between the earliest and latest dates in your model, and then you can add calculated columns to your table such as Year and MonthYear. Now what I did after saving this version of the file is I also went back and changed the query parameters back to 1980 and 2018 here, and the reason why it shows 2001 and 2017 is because we currently have a filter set for this table. I'm going to keep using this table. And right now what I have is years. But they aren't set up as datetime objects, and that's what I want to get into this model because I have tags with dates, and I have ratings with dates, and maybe I'd like to do some analysis based on those dates. So in order to work with these dates, I need to create a new table. So I'm going to go to the data modeling editor, and I'll choose the Modeling tab on the ribbon. We've learned already how to work with measures and with columns, and now our third option is to create a table with DAX. And I can give it a name. I'll call this Date, and it's as simple as using a CALENDAR function. Now there are two of them. There's a CALENDAR in which I need to specify a start and an end date, or I can have Power BI look at my model and determine what the earliest date and latest date used anywhere in any of the tables are. And so that's what I'll choose to do because, in theory, this could change over time. So I will let the Date table also be dynamic depending upon the contents of my data model. And that's really all I have to do. I just click the checkmark here, and I get a column that says our earliest date is January 1, 1995, and if I wanted to see what the latest date in there, I can do a sort descending and see that it goes up to 12/31 of 2018. Now having this column full of dates is all well and good, but it also has time in there, and that's not very helpful for me. I don't really want to look at things by time, so I can just format this in a format that I would find appropriate, month/day/year for me. Maybe you'd like a different format. You can choose one that you find suitable for your own analysis. So that's step one, having a Date table that contains a Date column.

Using DAX Functions for Date and Time Operations

DAX includes a lot of functions for working with dates and time. Even if you don't need a Date table, sometimes you still need to manipulate dates. So, for example, you might have datepart references in your data, maybe a year here, a month number there, and a day of the month somewhere else, and you can bring those together to produce a datetime data type by using the DATE function. That way, you can build out a datetime data object that you can use with the time intelligence functions because referencing the individual columns in any other way just isn't going to work for time intelligence. Similarly, you might have a text string that looks like a date, but Power BI sees it as text, and you can't fool Power BI, but you can convert it into a date by using the DATEVALUE function like this. I'm American, so I use the month/day/year format like you see here, but if you're somewhere else in the world and your computer is set to a different format, use the format that you normally use on your computer, and Power BI will do the right conversion. The TIME function is like the DATE function but combining timeparts into a datetime object using our minute and second. And we have a TIMEVALUE function that corresponds to the DATEVALUE function, but in this case, it accepts values in hours, minutes, and seconds structure as a string. Maybe you need to convert values in the other direction moving from datetime objects to numbers or strings. So here we have, DATE, MONTH, and YEAR functions that get us the numeric values related to specific dateparts of a date. We can also get the WEEKDAY number from a date, but whether Sunday or Monday is the first day of the week depends on the second argument, so we have WEEKDAY, what's the date column, 2. Two stands for Monday as the beginning of the week, but if you want Sunday to be the first day of the week, use a 1. Either way, each number of the week progresses up to day 7, which would be Saturday if you're using Sunday as your day 1, or it would be Sunday if Monday is your day 1. Now you could also use a 3 as that final argument if you want Monday to be numbered 0 and then end the week on Sunday with number 6. Up to you. On the other hand, WEEKNUM only takes 1 or 2 in the second argument. So the WEEKNUM function returns the week number of the year. Is it 1st week or the 52nd week? And you just need to specify whether Sunday is the first day of the week with a 1 or Monday is the first day the week with a 2. And then we have our MINUTE and SECOND functions as well to pull out those respective parts from a datetime object that contains time. And there are more DAX functions for dates. So we can find out what today's current date is, whether you use your local time or UTC time. Whereas those previous functions focus only on dates, you can also get the date and time using the NOW or UTCNOW functions. You can even use relative date functions like EDATE to return a date that's some number of months before or after the date you specify, or you can use EOMONTH to find the last day of the month for some number of months before or after the specified date. There's a YEARFRAC function that calculates the percentage of the year that represents the range between the start and end dates that you supply. And we have the DATEDIFF function, which compares two dates and returns how much time has elapsed between those two dates. But you need to specify how you want that time resolved. Is it a year, a month, a day, an hour, and so on, whatever the valid options are for that function as shown here on the slide. Now if you have a Month column in your Date table, you might also need to add a column to your Date to handle sorting. You can do this by concatenating together the year and month, but for month, you need to make sure that the number appears as two digits so that you have 01 for January and have that sort correctly.

Creating Calculated Columns for a Date Table

Now I want to be able to create a hierarchical structure, so I want to have year and month that I can drill down into date within a hierarchy. So, for that, I need to add columns. So the first column I'm going to use will be for year, so I can set up a column called Year, and I use the YEAR function to pull out dates, and we'll talk about other kinds of datepart functions a little bit later in this module. But this is fairly straightforward and is common in many languages if you're used to doing any kind of programming or scripting. So when I click the checkmark, then this is populated row by row by row. Then I might add another column for months. And in this case, what I'm going to do rather than just pull out the month from that date, I'm actually going to do a combination deal. First, I'm going to use the FORMAT function with the format string MMM that converts the month digit into the month name but then uses only the first three characters of that name. So we return that value, and we concatenate it with a space and then reference the value that's in this Year column, so that produces both a Month and a Year designation, and that's useful when you're building date hierarchies because if you were to drill down and then lose context of what year you're in, if you just had the month of January, you wouldn't necessarily know what year January belonged to, and if you had Januarys from different years, all of a sudden they would be combined as if they were one January for all time, and it wouldn't necessarily make sense. So I like to have something where I would concatenate the month name with the year name. If we go back to our report, and let's place in Year, look what happens. Year wants to summarize, and that's not really what we want to have happen. So we need to change our default summarization. If you recall from an earlier module, we covered this property. So now when I go and use that value, I have to clear it from my table and add it in again. Let's pull out the first year that belonged to Movies. Now I can see that I have the years listed individually as objects as opposed to numeric values that need to be summed up. And then next to those, I can place MonthYear. And notice the sort order here. We have April, August, December, February, January, July, June. So we are sorting alphabetically here, which is not particularly helpful, whereas if I remove Year from this table, we can see all of the Aprils grouped together, and then we'll see all of the Augusts grouped together and so forth. And what I really want to see happen is a chronological sort. So that's when we have to add in a column explicitly for sort purposes. So we'll add in one more column into our Date table, and we'll call this MonthSort. In this case, we'll use that YEAR(Date) again, although we could've just referenced YEAR explicitly here. And we'll concatenate that with a dash, and then we'll take the MONTH portion of the date and basically pad it with a 0. So what will happen is if we take the month from 1 here, there will be 1 digit. So if we say give me the right most 2 digits, it's still just going to return a 1. But I will prefix that with a 0 to make it a 01, whereas if I had a month like, say, December, then the RIGHT function would have 012 as the string that it operated on, but it would drop that 0 to return only 2 characters, the 12. And so regardless of what month I'm in, I'm always going to get 2 characters for those months. So then I can go back to my MonthYear column, select it, and then use the Sort by Column option to specify MonthSort as the Sort by Column. And now when we go back to our report, we can see everything lines up with the proper chronological sorting.

Working with Date Hierarchies

Now that I have my Date table created, there's one more step that I'd like to do, and that is to create a Date Hierarchy. So I'll do that by clicking on the Year column as the starting point, and we'll say New hierarchy, and then we will add MonthYear to the Year Hierarchy. And, of course, you can rename that if you'd like, and we'll add Date to the hierarchy as well. I'm going to go ahead and hide my MonthSort because it's not useful for analytical purposes. And all looks well. I can use these columns individually or within the Year Hierarchy. Now to use this, let's set up another measure. Let's look at tags as an example, and let's say that I want to create a measure for tags where I am counting the number of tags by date. So we'll create this measure called Tag Count, and that will be a simple calculation based on counting the rows in the Tags table. So I'll select this table, add Tag Count to it, and we see that Tag Count is repeating. Now we talked about this type of problem earlier in this course. So we have a Date table, and we have a Tags table, but the two tables are not related. So I can bring Tags and Date next to each other so we can see them a little bit better, and I can take Tag Date, I go from the many table over to the one table. So drag and drop Tag Date onto Date, and now I have a proper relationship between them where I have the Tag Date related to the Date in a many-to-one relationship, and we'll leave it as a single direction. And, lo and behold, we have now correctly our tag count. The only other thing I might do with Tag Count is to set in a comma for the representation of things over a thousand. So there's our Tag Count. Now what about our hierarchy? So let's go back, make sure that table is selected, and clear out MonthYear, and add in this Year Hierarchy. Now when I add the Year Hierarchy, the default behavior is to add in all of those things, year, month, date, altogether. And we can click on that table, and I'm just going to move Tag Count down to the bottom part here. And also I am going to convert this table into a matrix, so this is how we can navigate our hierarchy so we can see at a year basis that we can then drill down into the months, and we can drill down even further into specific dates. Everything sorts properly, and we have now added the ability to analyze our tags by date. Now I could have just used Tag Date, and Power BI would recognize Tag Date, so if I have the datetime associated with that properly, which it is a date, but having Tag Date in one table and maybe I want to compare tags with ratings that also have its own date, I would not be able to line them up side by side because there's no relationship between tags and ratings, whereas if I add in a Ratings Count measure by using the COUNTROWS function, and then don't forget to add in a relationship between Ratings and Date once I do that, I can add in measures from two different tables, each of which is associated with dates, but I can combine them together and review the comparisons across dates.

Applying Time Intelligence

And all those stacks functions that we've covered so far are great. They all really lead up to the real power of working with dates in DAX, and that is to use time intelligence functions. These are the functions that are purposely built to support analysis that is very specific to dates such as cumulative value analysis. That's things like figuring out month-to-date, quarter-to-date, and year-to-date values. And then we have relative date sets, which is where we can list the actual dates used to determine month- and year-to-date. Or we can set a range of dates. So maybe we want to compute sales between a beginning and an end date. So relative date set functions allow us to define what specific dates that we want to analyze. And then we have relative date functions. These are useful for things like year-over-year calculations and other types of time period comparisons that are similar to this. And the last two types of time intelligence functions are related. We can look at values for the end of a period such as account balance for a customer at the end of the year, or conversely look at the opening balance at the beginning of the year or month or quarter. Both of these end of period values and balance forward values look specifically at month, quarter, and year. So let's go back to our Power BI model so we can see how to use all of these.

Using the TOTALYTD() Function

So we have our table setup looking at Tag Count and Ratings Count by year, month, and date. I'm going to shake this up a little bit. I'm going to remove Ratings Count so we can just focus on Tag Count for the moment. And I'm going to drill up so we can look just at months for now. And let's sort so that we can see things in chronological order as well. So a common type of analysis that we might need to do if we're measuring something over time, this happens a lot in retail, for example, where we want to see how things are changing and building up to some target value in a year. So we might do year-to-date calculations. So let's look at an example of how that would work with our Tag Count. So I'll create a new measure here, and I'm going to paste in my calculation, and we can see here this is a TOTALYTD calculation, and the first argument is the measure that I'm accumulating, and then the second argument is the Date table and the column in that table that contains the individual dates. And then here's that ALL function again. We've seen this before when we were looking at the CALCULATE function. And, basically, what this does is remove any date filters. Again, this gets into filter context, and we'll learn more about that in the next module. So just know that anytime you're doing year-to-date calculations, and you have your date values in the same visualization, you are going to need this ALL('Date') removed. So when I click the checkmark, and then I add this Count YTD, and, see, there's an example of how I messed this up, is I just created a new measure and then assigned it to the Award Givers table, and I don't really want it there. I want it in the Tags table, so I can simply move it over there. And let's give it a new name as well. I should be more specific and call this the Tag Count YTD. So, now that I have this in place, this is how a year-to-date accumulation works, and month-to-date and quarter-to-date are very similar. So here I have a year that begins in January and ends in December. So each month, there's a specific Tag Count. So our year-to-date value for January 2006 is 16. In February, we have 11. That 11 gets added to 16 to derive the value of 27. So at this point, the February is the year-to-date value, so everything from the beginning of the year, January 1, to the end of February, whatever date that was, the 28th or 29th, I don't remember without peeking, is the sum of those values, 27. And then we can just keep incrementing up through the year. And so we see here, for example, December has no values. So we have a repeat of November to December. So this is a year-to-date accumulation, and this works whether we're looking at the month level. Even if we drilled down at the day level, we can see the year-to-date works there as well. So here we have 4 counts up to January 18. Nothing happened on the 19th, so it remains 4. Then we had 6 tags on the 6th for a grand total of 10 on the 20th of January. So year-to-date calculations are real common, especially when you're trying to measure progress towards a goal. Maybe you have a key performance indicator where you want to hit a certain target of values. You want to see how you're progressing, and it reports that over the year. The month-to-date and quarter-to-date work similarly. It's just what I call the reset button. Where do we reset to 0 and start accumulating our values? Is it at the year level? If we go down to 2007, for example, we reset with a 2 here because the year started over. If I had a month-to-date calculation, then each month would reset our month-to-date value, or if I had total quarter-to-date, then each month that begins a quarter would reset the counter to 0.

Using the CALCULATE() Function with the SAMEPERIODLASTYEAR() Function

I'm going to skip over working with relative date sets and jump into using a function for relative dates, specifically same period last year. So what we can do here is add in a new measure, and I'm going to choose the option off of my menu here rather than the button. I always get myself into trouble that way. I'm going to paste in my expression here, which this one's going to be Tag Count Last Year. So here we run into that CALCULATE function again, and if you recall how this works is the first argument is an expression, which in this case is saying look at the measure called Tag Count, and then the second argument is some type of filter. So the SAMEPERIODLASTYEAR function takes the Date column of the Date table as an argument and says whatever is in scope, and hold that thought for a moment, but whatever's in scope when we're trying to perform this calculation, then go back one year from that date and return the Tag Count value for that date. So let's go ahead and add it into the table so that we can see what that means. So here we have some data, and this, again, this is a little sneak preview of filter context, but let's look at just even the year. If we just go up one year for the year 2006, there's no other prior data available. So the Tag Count last year has no value, whereas with 2007, this is the context for this row 2007. So the filter SAMEPERIODLASTYEAR is looking at this date of 2007 or the underlying date set that belongs to that, which is January 1 through December 31 of 2007, and says, okay, go back one year, which is 2006, and return that value. So that is how Tag Count last year is calculating. Similarly for 2008, we're at the year level, so it's going to return the 2007 Tag Count on the row, that's 2008. And when we drill down, this same principle applies for the month. So here we have January 2006 with 16, so when we look at January 2007, we have 16 to represent January of 2006. And if we go forward to 2008, we'll see that it will have a value of 2 because that's what we had for January of 2007. So the SAMEPERIODLASTYEAR is relying on filter context to know which is our starting point that we're going to use to go back one year, whether that's at the year level, the month level, or even at the individual day level. If we need to find where we actually have dates means we need to get up into the 2007 timeframe. But there you go. We can compare values across time. And so that becomes useful when I want to do calculations like growth. How much growth did we experience from one year to the other? Or how much did growth go backwards? Either way, this is a very useful type of date calculation.

Using the OPENINGBALANCEMONTH() Function

And one more example. This time I'm going to show you the balance forward type of calculation. And this is very similar to the end of period values as well. We just have different functions. So here I'm using the OPENINGBALANCEMONTH, but there is also the CLOSINGBALANCEMONTH. We could have OPENINGBALANCEYEAR, OPENINGBALANCEQUARTER, or CLOSINGBALANCEYEAR, or CLOSINGBALANCEQUARTER. So those are our six options for working with either the beginning or the ending of a month. How would we use this? Well, it is going to, in this case, we're just going to use the year-to-date values just arbitrarily because it works for what we're trying to accomplish in this particular example. Your dataset might use a different measure. Maybe you have accounts receivable, and you are using transaction data, and so this would look a little differently in that situation. But the key elements here are we have our function that needs an expression as the first argument, and then like many of the other time intelligence functions, it needs to know which table and which column in that table actually contain individual dates. So when we apply this measure into our table, let's see how this behaves. We have---at least at the year level, everything looks pretty much the same as if we were doing a comparison with last year. So, in effect, we are saying at the beginning of 2008, for example here, what was the balance for 2007? So at least at the year level, we're saying if I'm at the 2008, go back to the year-to-date value, which would've been December of 2007, which would also be the year-to-date value, and use that as the starting point for calculating the Tag Count in 2008. And, indeed, when we go down to the next level, and let's look at 2007, we can see the year-to-date for December 2007 was, in fact, 22. So when we are at the month level, on the other hand, let's just look at 2008 as an example, then our opening balance for January 2008 is actually the year-to-date value for December 2007, but at the February, it's going to get January's value. March is going to use February's value, so on and so forth. So it's not exactly a good fit for the type of analysis we're doing here to use a balance forward. It doesn't really make sense. But the calculations I think you can see how that behavior is different from a year-to-date calculation where you can see the values here or a SAMEPERIODLASTYEAR type of calculation. So we have lots and lots of different options with our time intelligence functions in DAX.

Summary

And that concludes our whirlwind tour of using DAX for date and time calculations in Power BI. We explored a lot of capabilities beginning with the creation of a calculated Date table that we could use to compare measures over time. We then enhanced this calculated Date table by adding several calculated columns using date-specific functions such as YEAR and MONTH. And we learned how to use the FORMAT function to change how a month name is displayed. And we set up a calculated column to get months to sort in the correct order because, otherwise, we would see months list out as April followed by August, and so on, and that would be confusing, right? And we also looked at each type of time intelligence functions supported by DAX, such as TOTALYTD and SAMEPERIODLASTYEAR, among others, and we looked at several of these side by side to see the similarities and differences so that you could choose the right one for your own analytical requirements.

Applying Evaluation Context

Overview

Hi. I'm Stacia Varga, and welcome to the Data Modeling for Power BI course and the current module, Applying Evaluation Context. In other modules of this course, I have alluded a few times to the effect of filter context on the results of DAX expressions. Now's the time that I'm going to delve more into this topic, as well as other types of contexts that influence how Power BI returns a value for a calculation. Evaluation context is an umbrella term that describes the effect on query results that are created by row context as one effect, and I'll show you what that means with some concrete examples in this module. It makes sense when you see it, but just to hear a definition of row context isn't always helpful. Row context is a very important concept for DAX, but you needed to build up some experience working with DAX in general before turning your attention to it. And if you've successfully completed the other modules in this course, you're ready to tackle row context. Evaluation context also includes the effect of filter context. So we're going to spend some time exploring that concept further, as well, as it's very important to understand in order to get the right behavior from your calculations, especially because you need to factor in both filter and row context. Last, we'll apply these concepts in a slightly different way for a special type of data structure known as a parent-child hierarchy. And we'll learn how to work with functions that are built to handle this type of data.

Exploring Row Context

Let's start building a foundation for understanding evaluation context with row context. Row context is the simpler effect to understand. It defines which columns are related to an active row and in scope for a given calculation. You've seen this behavior when building out calculated columns, although I didn't call out that behavior specifically as row context. But when you add an expression to a calculated column, you can see a result populate in that column for every single row. Even an empty result or an error is still a result. It's possible to have some rows return valid values and others return invalid values, which proves the point that calculation is row based. The whole column doesn't fail because some or all rows fail when the expression is resolved. The bottom line is that row context defines which rows of any table happened to be in scope for a calculation, and you can take advantage of that and more complex calculations such as those that involve DAX iterator functions. In that case, rather than aggregate a group of values found in a column, an iterator function can get values from other columns based on row context, perform some type of operation, and then aggregate the results. That's easier to understand when you see it, so let's go to the Power BI model for some examples of row context.

Using the RELATED() Function to Explore Row Context

In this demonstration, we're going to take a look at row context. Now to work with row context, I'm just going to duplicate this page for the DAX Concatenate and Age, and we'll call this Row Context. This gives me a good starting point because it actually is a good demonstration of row context. So what we have here is just a table that includes the full name from the People table. So let's go look at the People table for a moment, and this Full Name calculated column is based upon the concatenation of two columns, First Name and Last Name. And if we sort this in the opposite direction, we can actually see some first names and last names appearing in here. So, literally, row by row by row, we can perform a calculation that is row context when we're working within the same table. If I want to work with columns across tables, then that's where we can use a RELATED function to apply row context across tables. So, for example, I have an age for my individual people that has been calculated where Birth Year exists, but let's say that I now want to know what the age of a person was when they were filming, and I can't put that into the People table because it all depends upon the individual movies. So in the Principals table, we are matching individuals to their movies so that we can calculate for any individual and a given movie what their age was at that time. So let's add a couple of columns. I'm going to add these incrementally just so you can see how this works. First, if I want to take the year that a movie was released, I'm just going to copy and paste this in from my snippets file, but we use this RELATED function to say look at the relationship between the current table, which is Principals, and the Movies table, and that's going to be based on this identifier here, and for each row, go and retrieve the year. So this, again, is taking advantage of row context that there's only one value associated with each and every row. And one thing that you might notice, I'm just going to remove this for a moment, is when I look at the columns that are available for use with a RELATED function, it's not showing me each and every possible table. It's only those for which we can get to by way of a relationship where the table that I'm in is the many side and the table that I'm using with the RELATED function is the one side. And the reason is we have to be able to return one row from that. Now we did have awards out there as a possible option, but that's because if we go to Movies, that's the one side, from that movie, we could go to one award type. So it will go across multiple tables as long as we're always going many to one, many to one. So that's how that RELATED function will work, and it will control what you can see in there. So I can use row context using the RELATED function. So here I have FilmYear, and then I'm going to take BirthYear and create a new column as well. So I'll just paste in BirthYear here. So if I want to determine age at filming, I'm going to use the technique much like the one that I used in an earlier demonstration where I calculated age in the People table. So I'm just going to check if FilmYear is blank. If it is, make the Age at Filming blank. Otherwise, look at BirthYear. If BirthYear is also blank, return a blank in this column. Otherwise, calculate the difference between the FilmYear and the BirthYear. And we can see where those values occur. So let's go look at our report now and add in Age at Filming. And so we can see for these particular movies, let's bring Movie Name back in to this report, we can see that we have different ages because these ages reflect current time as opposed to the age at filming.

Using the AVERAGEX() Iterator Function

What if I want to know for this set of movies or for individual movies what the average age of people working on this movie is? You might think that we could create a measure that just sums up the Age at Filming and divide it by the People Count, so let's try this. I'm going to call it Bad Avg Age. So if I said I want to do a sum of the Principals Age at Filming and divide that by People Count, and then let's add that to our report, it looks like a reasonable number. And if we look at Avg Age, of course, on an individual basis, we have values here that match up because there's only one individual, so dividing the Age at Filming by the one individual returns the same amount, but when we go to look at that average, it's not necessarily accurate because if I add in People Count, this People Count includes a number for the people that we don't have ages for. So this, in fact, isn't an accurate average. So one way to solve this would be to change my Age at Filming here to an average, and I can see the average age here. That is the correct value by the way. But what if I don't want to store this Age at Filming as a column in my Principals table? Maybe this is a very large table, and remember that calculated columns take up space, and I'd really rather just calculate at query time, in other words, when I view this value in a visualization. Let's add a new measure here, and let's say that I'm going to take that expression that I have for Age at Filming, and we'll create a new measure. I'm going to call this Bad Avg 2, and let's say that I want to try to use the AVERAGE function with this expression. And when I try to do this, the AVERAGE function comes back with an error and says I can only use a column reference as an argument, and I cannot use an entire expression like this as an argument. So that won't work. My solution for when I don't want to use this calculated column value, but I need to be able to evaluate row by row, I can instead create a measure using an iterator function. So let's call this the Avg Age at Filming. And interrater functions are like our aggregate functions except they have an X on the end. So this would be true for SUMX. We have COUNTX, MINX, MAXX. And the one that I'm going to use here will be the AVERAGEX. So in that case, we need to tell it what table we're going to be working with, which in this case is going to be Principals, and we use that same expression that we had in the calculated column, which we know calculates row by row. So in that way, we now can have an Avg Age at Filming, make sure to click the table, and add in that measure. Let's take out People Count and our Bad Avg, and we can see that the Avg Age at Filming from the measure is correctly calculating. So we can remove Age at Filming, and, in fact, we can remove it from our model entirely, which was the goal, so that is an example of row context both when we're dealing with calculated columns and when we need to use row context for measures.

Understanding Filter Context

Every time that you see data values display in a Power BI visualization, you see filter context at work. Even when you see the clues that a relationship problem exists when there are repeating values across rows of a table, that is still the effect of filter context. Now it's time to take a closer look at filter context conceptually, and then we'll explore its effect through demonstrations. Let's start thinking about a report visualization, a table. Filter context applies to all visualizations, but I think it's easier to think about it in terms of a table. After all, any visualization can be turned into a table. It's a collection of data that's just projected into the type of visualization that you select. So in a table, the data's organized into rows and columns, and as you make your selections of fields to add to the data visualization, Power BI has to decide whether to display a text string as a column header or a row label or how to calculate a value in a cell of that table when you add a measure. To calculate an individual cell, Power BI has to look at what's on rows. What are the field values influencing that row? And also what's on columns. What are the field values influencing that column? In other words, the row and column field values are used as filters on that cell. And, remember, in a matrix, you can actually have multiple rows and multiple columns influencing any given cell. Now, furthermore, any slicer that you have on that page and any filters defined for the visualization or for the page or for the report, these are all filters that Power BI must factor in when evaluating each and every cell that portrays a measure value. Collectively, these become the filter context for the cell. Power BI uses that filter context when it scans the data model. In other words, for any given cell, it filters the data model down and then aggregates the results of whatever cells are left over using sum or average or whatever aggregate function is defined for that measure, and it returns that result as the cell value. The amazing thing about Power BI is how fast it can do this for each and every measure cell in your table.

Using the CALCULATE() Function with ALL()

I'm going to go back to the DAX Measures here and duplicate this page, and we'll call this Filter Context. And I'll remove a few things here to make some room. But, basically, when we look at these two tables, one, this one up in the top right corner, is currently unfiltered. There are no visual filters here, no page level, no report filters. So this is an unfiltered value for Movie Count and Award Count. If we look over at this table to the left, Movie Count is filtered because, number one, we have Award is not (Blank). That means it has an award and, therefore, Movie Count and Award Count happen to match. This value is filtered. Movie Count to the right is unfiltered. Power BI looks at the context. This is what's on rows. On any individual row, we get a Movie Count value. So this grand total is computed by looking at the entire filtered table and summing up those individual cells. If I were to further filter in this table, I could add in Year as an example, and my Movie Count total continues to be revised. Now this percentage value has a calculation in it where we're looking at Award Count, and we're dividing by a grand total. In other words, this 34, 875 is the grand total. And so that Calculate function has the special feature in that it will give us the opportunity to override any existing filters. So, for example, if we were to move this out of the way for a moment and add in another column here, let's create another measure, I'm going to add it into the Movies, and let's just take that denominator from Pct of Movies, when we add this to our table, we can see the values repeat. Now this is not an indicator that I have a relationship problem, as I've pointed out in earlier examples, because this is purposeful. I need to have a denominator that stays constant regardless of what other filters are happening in this visualization. So my numerator will change in this Pct of Movies with Awards, but my denominator always stays constant. So we can change our filters, but the Unfiltered Movies always stays the same.

Using the CALCULATE() Function with KEEPFILTERS()

Now let's look at measures and filters from a slightly different perspective. I'm going to copy this table and paste it into another section here, and I'm going to remove all of the measures, and we'll add in People Count. Now I need to rename this Custom as Gender. Let's move this out of the way also, and make some room for ourselves. Let's look at just Oscars and clear the year. So what I want to do is turn this into a matrix and then show Gender as columns. And what we can see is that we have in many cases an unknown value for Gender. So let's say that I want a count of people for whom I do know what the genders are. So let's create another measure, call this Known Gender People Count, and what we can do is create a filter. We'll use the CALCULATE on the People Count, but the CALCULATE expression will take a second argument where we used ALL, the ALL function before to remove a filter. Now we're saying add in a filter and only include in our table that we're counting, only include the people with a gender that is not equal to unknown. So I'm going to add in my Known Gender People Count. So we have this balancing act of trying to show text large enough but still be able to see things. So I'll just rearrange slightly here. We have our new measure, and for any given movie, we can see that grand total represents the males plus the females, and this Known Gender People Count is a grand total and doesn't break down by individual gender. So the CALCULATE function here effectively said ignore any other filters that are applied at the moment through filter context and only use this filter. And if that's not the behavior that we want to have, there's a KEEPFILTERS function that says go ahead and keep whatever filters already exist, and then apply this filter as well. So now we can see that there's a blank value there, and for each individual gender, Female is properly applied and Male is properly applied. So we can pull out People Count and get to just the Known Gender People Count and have counts of people in each movie where we actually know the gender. So yet another way to manipulate filter context in our DAX Measures.

Working with Parent-child Functions

Another type of context that you might run into in some datasets has to do with parent-child structures. And DAX includes several functions that help you analyze this type of structure. A parent-child structure has data that is hierarchical, but the number of levels down any given path can vary. This is common in datasets that represent organizational charts or financial chart of accounts as a couple of examples. When you have this type of dataset, there's some additional processing that you need to do. Specifically, you need to create a calculated column for each level of the hierarchy, which means you need to have an idea of the maximum number of levels that will exist. Then for each calculated column, you use DAX functions to determine what value belongs in that level, and then add measures to the data model so that when we're navigating up and down the hierarchy, we can have Power BI ignore values for levels that don't exist, which is a lot easier to understand when you see it, so let's go to a demonstration.

Using the PATH() and PATHITEM() Functions

To work with parent-child hierarchies, we need to have some data. So let's go to our Query Editor, and I'm going to pull in a new data source from a CSV file. We have a CSV file containing genre in parent-child format. So the structure here is that we have one column for child, and one column for parent. We need to specify that we're using the first row as headers. And so effectively we have some top-level information, and then we have genres that are Action and Crimes, so those movies that would have this would roll up into Action, or Action, Drama would roll up into Action. These relationships where we have a child associated with a parent and one of these parents would be a child that could roll up to another parent, that's what we call a parent-child relationship. And, by the way, these correspond to the Genre column that we've hidden in the Movies table so that we can link these two things together. So I'm going to call this Genres PC for parent-child just to have a differentiator here. I also need to replace these NULLs. Those came through as text, and we just want to replace these with real null values. So I'm going to type this null with lowercase, and, incidentally, I don't have a perfect match for everything in our Movies table, but it's enough to demonstrate what we need to demonstrate for parent-child relationships. So let's copy one of these pages here. I'm going to go to the data page, and we'll go to this Genres PC, parent-child. So to work with parent-child data, you will have some kind of relationship defined in your data. Here I have one genre referencing a parent genre, and we can have what's called self-referencing relationships because the parent would have also a row in here as well. So everything that's in the parent column except when we get to null values would have to exist as a child. Then we need to create a new column, we'll call it Path, that takes the information from both columns and creates a delimited list using a pipe delimiter here. This is one of the DAX functions that we use with parent-child relationships. Then we need to define the levels. I happen to know that there are three levels in here, so I would create a column for each level. So let's add in another column here. I'll just call it Level1 and paste in my expression. So we'll use the PATHITEM to look at the Path column and pull out the first level, which would be the very topmost level as it traverses the relationships between parent-child, parent-child until it hits the very top. So we would repeat this for each of the three levels. The differences---each level, of course, is going to have a different name, and the second argument determines which level we're retrieving from the Path column. Now notice there are some cases where our level is empty when we hit the bottom most level.

Using the PATHLENGTH() and ISFILTERED() Functions

Let's go to our report, and let's change this to show us the individual levels, but what I'd really like to do is create a hierarchy. So I'm going to right-click Level1, add in Level2, and add in Level3. We can hide the original levels. And let's rename this hierarchy as the Genres Hierarchy. So now I'm going to place this into my matrix. I'm going to go ahead and remove the Movie Name and Gender, and we'll just do a Movie Count in here, our values, and we can see that we have repeating values when, for Movie Count, I don't expect it to do that. So that tells me I have a relationship issue. Remember that our Movies table has this Genres, which corresponds to a child in our Genres PC. Now when I go back, I can see that, indeed, I have genres. And this is a matrix with hierarchies, and so if I drill, I can see subsequent levels, except there are some cases where we don't have a third level. So what we can do is set up some measures so that we can evaluate where we are in a hierarchy and suppress a value so that the row does not show if there is no corresponding label for that level. So the first thing we need to do to make that work is to add a column to our table, and I'm going to call this Path Depth. And this uses yet another DAX function that looks at the path and determines how many levels does it contain. We can use that information with some other measures that we create. So I'm going to hide Path Depth and everything here except for what's in our hierarchy. Now I want to add new measures. One is going to say what is my Current Depth. So here we have another function that is called ISFILTERED, which returns a true or false, and it's going to look at each level, and if this is a filtered value, then it will return a 1, and if we're at Level2, it'll return 1 for both of these expressions and add them together. So we would have a 2. Or if we are at all 3 levels, we will have it filtered as 3. So let's go ahead and add this into our matrix just so you can see this working. So here we have Level1, Level2, Level3. This is for the Action where we have the top level with nothing to drill down from there versus we have Level3 items. We're going to create another measure that says what is the maximum depth we can be, and then we can see that there's a difference of the Current Depth, which is greater than the Max Depth, and that happens anywhere that we have these blanks so that what we can do is either adjust our current measure, but I'm going to create a new one just to have a differentiator here. I'm going to have Movie Count by Genres that says look at that Current Depth, and if it's greater than the Max Depth, return a blank. Otherwise, return the Movie Count. So when we add in this measure, we can see that Movie Count does not show up for our blanks, and so we'll remove Movie Count altogether. And so now we have the same number of movies that we started with, but now we can drill down up to three levels versus only one level. Now I might want to filter movies by genre, but when I'm working with parent-child hierarchies like this, if I were to add in Movie Name into this hierarchy, technically it works but only where we have Level3 labels. If I have Comedy, Drama, for example, I don't have a way to drill down on that. So if you'd like to be able to drill, then one solution would be to use the RELATED function and bring the Level1, 2, and 3 columns individually over into Movies, and then you could make a hierarchy from there. Or if you'd prefer to keep them separate, another option would be to use one visualization as a filter, and then you could see which movies belong to that genre.

Summary

The purpose of this module was to enhance your understanding of how Power BI evaluates measures, which has an effect on what you see in your reports. Using specific examples from the data model, I showed you how row context affects not only what values appear in a calculated column, but also how row context affects measures. Although row context applies no matter what when you add numeric values into a visualization, it is also a factor within certain types of functions, the iterator functions. Those are the ones that end in X like SUMX and COUNTX. In this module, we used the AVERAGEX function to determine how old a person was when a film they were involved with was released. And it happened on an individual basis. That would be where the row context comes into play. And then we averaged the results so we could see the average age of people associated with a specific movie as just one example. Then I provided examples of filter context to help you conceptualize this very important characteristic of Power BI behavior when calculating values for a visualization. In particular, the CALCULATE function lets us manipulate filter context, either by overwriting an existing filter or by adding a filter that's not already influencing a visualization. Filter context can be tricky to understand sometimes, but with lots of practice, you will come to appreciate how filter context gives you a lot of power for your analytical calculations. And I finished up this module by showing you how these row and filter context concepts come together in a different way when you have a parent-child dataset. We set up a parent-child structure, and then we learned how to use the PATH and PATHITEM functions to create calculated columns that become part of a hierarchy. And then we used PATHLENGTH and ISFILTERED functions to set up measures that change behavior based on the filter and row context of a hierarchy level that's represented in a visualization. And with these core concepts in mind, you now have a lot of tools at your disposal for building Power BI models with wide-ranging capabilities. Thanks for watching this course.