Charting outcomes in the match 2020 reddit

A paginated report has variable height that can be scrolled vertically on the screen or split across multiple pages when printed or saved to different output formats. Power BI has become the go-to business tool for modern reporting, but sometimes we need information displayed in the more traditional paginated format. A colleague asked if a Gantt chart could be created, and I thought this was an excellent example of where Paginated Reports & SSRS were an ideal choice for the purpose. A Gantt chart is a running list of activities with the duration for each displayed as a horizontal bar depicting the beginning and ending day along a horizontal scale. The challenge is that this is not a standard chart type in either Power BI or SSRS/Paginated Reports. Furthermore, project planners may prefer to see activities as rows in the format of a printed page, as you can see in this example: Let me just get this part out of the way before we proceed. I've been using SQL Server Reporting Services (now called "Paginated Reports" in the Power BI service) for most of my 20+ year career in BI and data analytics. It is a very flexible tool that can, at times, be tedious and time-consuming for a report developer to use beyond its basic features. I've used it, and seen it used in very creative ways, but some things take patience to work through the trial-and-error design process. Once you know what you are doing, it can be a wonderfully powerful and flexible design tool. Source Data The source for this report is a list of project tasks, each having a start date and an end date. For demonstration purposes, I used the Enter Data feature to populate a portable dataset, but you can use a database or any other supported data source. Since the data in my sample is stored as text, I do need to perform some extra data type conversion in some of the expressions. In some cases, this won't be necessary when using strongly-type data sources like SQL Server. A Gauge in a Table Most advanced Paginated reports are built by using a data region, like a table, matrix or rectangle; to repeat other report items in data groups. To achieve the Gantt chart effect, I use a table data region to repeat a linear gauge for each row in the dataset. The table is sorted by the Order column. You could also use the DateFrom column for sorting (again, my data is stored as text which doesn't naturally sort in date order). I've added two linear gauges to the table: one in the details row of the table with the numeric scale hidden, and one in the header with the range (or bar) removed, and the scale displayed. This image shows the report in design view after the following steps have been completed. You can look though the sample RDL file I've provided, which includes a completed report based on the steps outlined below. Make sure you have the Properties pane displayed in report Builder by checking the Properties box in the View ribbon. To set the "window" of dates shown in the chart, I've created two date-type parameters named ChartRangeStart and ChartRangeEnd, and assigned default values that include the records in the source data, 3 months apart. The gauge visual doesn't naturally work with dates, so we need a couple of calculated fields: StartNumber and EndNumber. These are created in the properties for the dataset using the following two expressions, respectively to define these two calculated fields (no line returns, the text wraps): =DATEDIFF("d", Parameters!ChartRangeStart.Value, IIF(Fields!DateFrom.Value="", Parameters!ChartRangeStart.Value, Fields!DateFrom.Value)) =DATEDIFF("d", Parameters!ChartRangeStart.Value, IIF(Fields!DateTo.Value="", Parameters!ChartRangeStart.Value, Fields!DateTo.Value)) For reference, this is the Dataset Properties dialog after adding these two new fields: Choose Gauge from the Insert ribbon. Select a Bullet Graph from the Select Gauge Type dialog and then drop it onto a blank space on the report canvas. When working with an intricate visual report item like a gauge, I make it a point to place it outside the data region, make it as big as possible to setup all the properties. Then, I cut and paste the gauge into the table cell. Mysteries of the Gauge Visual This is where things get tricky. The Gauge, Databar, Sparkline and Indicator visualizations were all developed by Dundas Software. They had their own charting product and Microsoft purchased the code base to integrate with Reporting Services back in 2010. There is a trade-off when using these components; they erred on the side of giving us lots and lots of wonderful options to build complex charting visuals, rather than making them uber simple to use. You can spend lifetimes digging through these object collections and properties. Believe me – I've spent much of mine doing just that! Anyway, speaking from experience, check your OCD and don't stay up too late at night trying to figure out every nuance because this can become a Pandora's Box of knobs and levers that control all the intricate features of these visualizations. To unravel the Gauge objects, start clicking-around to select objects in the gauge visual designer. You will see that objects are organized into a hierarchy of collections within the Gauge Panel. The items in bold are those used in this report: LinearGaugesScalesRangesPointersCustomLabelsRadialGaugesIndicatorsLabels Objects in the linear gauge and radial gauge have corresponding names, like LinearGaugePanel1 and LinearRange1. You can modify or remove these objects by selecting them in the designer, or you can add or remove then in the properties pane using the ellipsis button for the collection of child objects. For example, the Gauge Panel has a collection of Scales, and a Scale has a collection of Ranges and Pointers. Until you get the feel of things, you might want to make a second copy of the gauge. You can always use Ctrl-Z to undo a delete or property change if you get into trouble. Setup the Gauge Delete all the Pointers and all but the first Range (it's named LinearRange1). You can click to select each object in the designer. Confirm the selection by checking the object name in the top of the properties pane before you delete or modify anything. The following image shows the properties of the LinearScale1. Modified and non-default properties are shown in bold: This is the entire MaximumValue property expression: =DATEDIFF("d", Parameters!ChartRangeStart.Value, Parameters!ChartRangeEnd.Value) To choose the LinearRange1 object, either click to select the object in the designer or use the ellipsis (…) button in the properties panel to edit the Ranges collection, then update the properties shown here in bold: For properties that represent a collection of objects, you'll see an ellipsis button (a button with three dots) on the right side of the property value cell when that property is selected. Click the button to open a dialog window to add, remove and edit the objects in the collection. This is the entire ToolTip expression (text wraps so don't use line returns): =Fields!Task.Value & " (" & CSTR(DATEDIFF("d", Fields!DateFrom.Value, Fields!DateTo.Value)) & " days) " & vbCrLf & FORMAT(Fields!DateFrom.Value, "Short Date") & " - " & FORMAT(Fields!DateTo.Value, "Short Date") Using the Insert ribbon, drop-down the Table button and choose Insert Table. Drop the table in the body of the report. Expand the dataset and then drag and drop the Task field to the "Data" row in the first column of the table. Select the gauge and then use Ctrl-X to cut it to the clipboard. Select the second cell in the "Data" row of the table, and then use Ctrl-V to paste the gauge. Resize the second column and row height to adjust the gauge size and position. Select and copy the gauge to the clipboard, using Ctrl-C. Select the second cell in the "Header" row. Use Ctrl-V to paste a copy of the gauge into this cell. In the new copy of the gauge, you will delete the Range and show the Scale. To do this, select the gray range bar and make sure that LinearRange1 is displayed at the top of the properties panel. Once you confirm that the Range object is selected, press the Delete key to remove it. Select the LinearScale1 object in the new copy of the gauge. Change the Hidden property to False. Expand the LabelStyle group and change the UseFontPercent property to False. You can optionally expand the Font group and change the Font property of needed. This is one of the many properties you can tweak later if you want to adjust things to your liking. To change the sort order of the table, choose the table and use the SortOrder expression. The default table name is Tablix1. To select the entire table data region, click any cell in the table to show the gray row and column handles, and then click the gray box in the top-left corner, between the row and column handles. The rest of the report design is typical. I've added textboxes to show the title and parameters selections. I've also added the option to hide and show additional columns using a parameter. These should be apparent in the sample RDL file I've provided. With variations on the design, you can enhance this report to add more details, and perhaps deal with weekends and non-workdays, etc. If you make such enhancements, please share them for benefit of the community. Download Gantt Chart sample project here I started a series of blog posts back in 2020 about best-practice guidelines for planning and designing enterprise reporting solutions with Power BI. To make the topics covered in this series of posts easier to find and follow, they are listed on this page: Doing Power BI The Right Way – for Enterprise Reporting | Paul Turley's SQL Server BI Blog which you can access from the main menu on the blog. We have a few more topics to go so check back and subscribe for notifications. Technology evolves, the industry changes and the way businesses use technology changes. The road that has brought us to the current state of reporting capabilities in the Microsoft data platform has been long and winding. Ten years ago, we were using SQL Server Reporting Services (SSRS) to create interactive, dashboard-like reports scorecard reports. It wasn't easy and the report interactivity wasn't exactly "snappy" as the report server had to re-query data and re-render the report every time a link or chart series was clicked. But, because SSRS was the best tool that we had back in the day, the techniques we used for advanced report development were arcane by today's standards. Over the years, I've written a few articles and posts about how to create reports with interactive KPI gauges, charts and scorecards using SSRS. Today we have a better tool for that style of reporting and analytics. Reporting Services remains a powerful and flexible tool for a few different styles of reports. Now that SSRS has been integrated into the premium Power BI service as "Paginated Reports", this raises questions about how and when to use each tool – and how to use them together.Related: Gauges in SSRS reports | Graphical Report Design Power BI Power BI makes the chore of creating highly visual and interactive reports much easier and the user experience is for more modern and engaging. However, Power BI reports are purposefully intended to be viewed with each page in a single browser window, with all visuals in view. Power BI reports are not optimized to be printed or split across multiple pages. To a small degree, it is possible to export some data to Excel, PDF or PowerPoint; but this is not the design purpose of Power BI – and it probably never will be. By contrast, SSRS/Paginated Reports was purposefully designed to render reports as continuous, paper-sized pages, with breaks and totals. Paginated reports can be exported to a variety of formats in high fidelity. Can Should When should we use Power BI vs Paginated Reports for our reporting needs? You can use either tool to create different styles of reports for different purposes, but should you? Just because you CAN doesn't mean you SHOULD. Can we use Power BI to create operational or transactional style reports?Can we create scorecards and dashboard-style reports with SSRS/Paginated Reports?Should we use a Power BI dataset as a data source for a Paginated report? The following diagram depicts the relative capabilities and design purpose for these two reporting tools. See the overlap? On the left, you see that Power BI is optimized and best used to create interactive, visual reports, scorecards and dashboard-style reports. Contrast this with Paginated Reports on the right side, which is best used for "list-type" reports that continuously flow across multiple pages. SSRS/Paginated Reports was architected and works ideally with SQL queries, where filter values are passed into the query using parameters. Let's break-down a few of these important decision points. Data Model vs SQL Queries There are a few reporting use cases that would tip the scales in one direction. If the need is to run a query and dump the results into a long list of continuous report pages, that is a classic use case for Paginated Reports. Power BI relies on data organized and optimized in a semantic data model, with data shaped and purposefully structured to support interactivity and analytics. Although Power BI models can handle a lot of data if properly organized, handling transactional details are not a natural fit for Power BI reports, both due to the format and the way data must be modeled for reporting. Importing tables that are both tall and wide, with many columns, will tank Power BI performance and impede users' ability to explore and interact with analytic data. By contrast, Paginated Reports rocks at this sort of thing. Need to generate a running list of ten thousand order line items, split into formatted pages with breaks, headers and footers? This is a natural fit for a paginated report. Do you need to export a table-style report to Excel with each page in a separate worksheet? Paginated Reports is a good tool to use for this sort of thing. Under the covers, Power BI communicates with a tabular data model with the DAX query language. Although elegant and efficient, DAX queries can be very verbose and difficult to read. When SSRS was enhanced to work the option to query Power BI datasets, some effort was made to allow the Report Builder query designer to generate DAX query code. It is not robust nor very easy to maintain. If you need to enhance a DAX-query-based report with any degree of sophistication, you will likely not have a good experience. SSRS/Paginated Reports were developed to support, and are best used with, native SQL queries. There are many nuances to the choice between these two options. A semantic data model provides a layer of abstraction for measures that define calculations and business rules. A model supports self-service analysis and discovery. Row-level security rules ensure that users only see their own protected data. Running a SQL query just gets the job done for a finite set of requirements, and perhaps only for one report. Interactivity In a Paginated Report, if you want to give a user the ability to click a chart column or cell in a table or matrix and then navigate or filter items in the report using that value as a parameter; this kind of interaction is possible, but it might take some code and development effort to get there. Even though SSRS and Power BI Paginated Reports are both defined as RDL files, the implementation of Paginated Reports in the Power BI service doesn't yet support the same native report navigation and drill-though functionality as SSRS on-premises. Report navigation can be achieved using parameterized expressions to build clickable web URL links but this is a work-around solution at best. The product will evolve, and I think we can bet on seeing more and more parity between on-prem SSRS and Power BI Paginated Reports as the product continues to mature. Guidelines Can you create a dashboard-style report with SSRS or Paginated Reports? – Yes, but it is complex and time-consuming.Can a Paginated report use a published Power BI dataset or SSAS model as a data source, with filters and parameters? – Yes, but DAX queries generated with the query designer quickly get large, complicated and difficult to maintain. Even using a small number of parameters and fields can produce an unruly query.Can the page dimensions of a Power BI report page be resized or to accommodate more visuals than will fit on a standard browser page? – Yes, but several features won't work as intended, and performance will suffer with more visuals on a single page. Can a Power BI report page be formatted to print on a sheet of paper, or exported with formatted and high fidelity to Excel or a PDF file? – Maybe to a point, but perhaps not very effectively. The bottom line is that both of these tools can be used to create a variety of report styles and to support a range of features, but one is better suited than the other for different purposes. Before stretching the limits of any tool, make sure you are using the right reporting tool for the job and consider how you can use both Power BI interactive reports and Paginated Reports together, to meet your analytic and operational reporting needs. The December 2021 Power BI Desktop update introduced a long-awaited upgrade to the partitioning and Incremental Refresh feature set. The update introduces Hybrid Tables, a new Premium feature that combines the advantages of in-memory Import Mode storage with real-time DirectQuery data access; this is a big step forward for large model management and real-time analytic reporting. Partitions allow us to break large tables in a data model into smaller, manageable storage units for a variety of reasons. Until recently, each table in the data model could be stored only in Import storage mode or as a DirectQuery table, which stored no data in the model. Generally, Import mode is fast and full-featured but large tables take up memory, while DirectQuery uses no memory for storage but is typically slower and limits many reporting features. Incremental Refresh can automatically break a table down so that only partitions containing new or changed records can be refreshed. There are two scenarios where combining Import and DirectQuery mode partitions might be optimal: 1. providing real-time access to recently inserted records, or: 2. proving access to infrequently accessed old records that take up a lot of storage space. This demonstration addresses the first scenario. Hybrid tables, an extension to the Incremental Refresh feature set, can consist of both Import Mode and DirectQuery partitions, so that some data is cached in memory and some records are read directly from the data source as users interact with the data model. Setting up Incremental Refresh is the same as it was when introduced a few years ago, but the options have been enhanced. Let's take a look at the updated Incremental Refresh dialog for a fact table. I will demonstrate with three years of historic fact data so you can see the results. I've already setup the required query parameters and filter step needed to enable the feature, that I described in this post. Choosing the Incremental Refresh option from the ellipsis menu for the Online Sales table opens this dialog. There's a lot going on in this window but pay attention to the following options that I have called-out in the graphic: Enable automatic partitioning by switching on "Incrementally refresh this table"Archive data starting 3 Years before refresh dateIncrementally refresh data starting 12 Months before refresh dateEnable Get the latest data in real time with DirectQuery For simplicity, I have not set the Detect data changes option, which is something I typically would also do but we're just keeping this demonstration focused on the topic of how partitions are created. Note the graphic added to the bottom of the dialog window, which displays a graphical representation of the results. I will break this down into more detail using the following graphic. Let's see the effect of these settings The graphic depicts the partitioning scheme for the Online Sales fact table in this data model. Bases on the settings in this dialog, the Power BI service will generate partitions using the RangeStart and RangeEnd query parameters to inject dates into the WHERE clause of each partition query. A few query examples are shown for demonstration purposes. The dark-colored boxes represent the Import Mode partitions that cache a copy of data in the model, and the light-colored box represents a DirectQuery partition that runs in real time when the model is used for reporting. Note that I have abbreviated things for illustration purposes, and there will actually be more monthly partitions than those represented here. Referencing the earlier Incremental Refresh settings dialog, the Archive date starting option specifies three years, which generates three yearly partitions, each with a relative date range. The Incrementally refresh data starting option specifies 12 months, which generates 12 monthly partitions (only 3 are illustrated here). The example query shows a range of one month per partition query. Lastly, a DirectQuery partition uses a range of dates to include all records in the current month and possible future dates. These partitions are created by the Power BI service the first time the dataset is processed, after the data model is deployed to a Premium Capacity or Premium Per User workspace. Let's take a look at the actual partitions. Using the XMLA endpoint, I can use the workspace URI address to connect to the service using SQL Server Management Studio and show the partitions for the Online Sales table. With 15 partitions, it takes longer initially, to process the dataset than it would if it were not partitioned. This is because the database engine only processes a few queries at a time. However, subsequent refresh cycles should be much faster since only a few partitions will need to be processed with additions and changes. Connecting to the published dataset using Tabular Editor shows even more partition details. Here, you see the monthly partition for November 2021. The Start and End dates for this partition are 11/1/2021 and 12/1/2021 (US date format), respectively. The last partition, which is in DirectQuery mode has a Start date of 12/1/2021 and an End date several years in the future. This range is a catch-all for records that could potentially have future dates. What are the performance implications of a hybrid table?How can you add an historic partition that uses DirectQuery mode? I will address these questions in a later post. Please share your experience using the comments for this post. The documentation on this feature is very brief right now, but here are some additional resources: Hybrid tables | Microsoft Docs Power BI Hybrid Tables with Synapse Analytics Serverless SQL Pools – Serverless SQL Last year I wrote this post about the new composite model feature in Power BI that enables datasets to be chained together using a featured called "DirectQuery for PBI Datasets and AS". The prospect of creating a data model that virtually layers existing data models together without having to repeat the design, sounds like nothing less than Utopia. We can leverage existing datasets, with no duplicate models, no duplication of business logic, and no duplication of effort. The promise of this capability is that data models may be referenced from other data models without duplicating data. So, is this really possible? The feature we are discussing in this post is not for beginners and it is in Preview, with some important limitations that should be considered in the overall solution architecture. Make sure you understand these trade-offs before using this feature. When the new composite model feature (using DirectQuery for AS) was released to Preview last year, I gave it a thorough test as did others in the community. It was cool and I was able to whip up a few demos, but right from the gate, I could see that it wasn't ready for production use. The product engineering teams have been working on this dream feature for years, and there are several dependencies in the Power BI service (Gen2 architecture), the VertiPaq engine, and the Power BI Desktop model designer that needed to be orchestrated. It is complex and will be slow to mature. The most significant shortcomings, in my opinion, were that in the local model (where you add DirectQuery connections to existing, published data models), you couldn't add objects with duplicate names. If any measures with the same name exist in the remote data model, you couldn't add a reference to the remote model. If you could get past resolving the measure name conflicts, when connecting to a remote data model, you get every single table in the remote model; thus, compounding the duplicate object name issue. Cool technology with lots of promise but there were significant implementation blockers from the get go. The product teams were notably quiet over the past year, with very little news about enhancements to this feature. With all the fanfare about the Preview launch last year, I was looking for more news about critical feature updates. There have been a lot of organization changes and feature reassignments at Microsoft this year, so maybe the work on this feature just hasn't been visible externally. But alas, there is good news! As of the October Desktop update, the composite model design experience has improved significantly. Any duplicate measure names are now resolved by the designer by adding numbers to the end. You can now select only the tables you want to bring into the local model after connecting to a remote model. New Concepts and Terminology Gen2 composite models introduce some new concepts and terms that are important to understand. Before you can grasp these new composite model concepts, you must understand the fundamental data modeling architecture in the SSAS Tabular/VertiPaq modeling engine, which is utilized within Power BI. Direct Connect and DirectQuery The first thing is that when you create a new report connected to a published Power BI dataset, this uses "Live Connect" which is the default mode for Power BI reports connected to a published dataset. To enable a Gen2 composite model, this connection must be changed to use DirectQuery, using the XMLA endpoint, to connect to the remote published data model as an Analysis Services connection. Chaining A data model that references another data model is referred to as a chain of models. Currently, up to three levels of chaining are supported. This means that if you have a data model that imports or connects to data sources, that published dataset can be referenced by another data model. The second model can then be published as a dataset. The latter dataset may reference multiple published datasets but it cannot be used as a DirectQuery source for a third data model. The product documentation uses the following illustration to demonstrate the limits of a three-level chain. In this example, the Sales data model is an Azure Analysis Services data model referenced as a remote dataset by the Sales and Budget dataset. Note: The Sales AAS model in this example could just as well be a published Power BI dataset because the XMLA endpoint sees them as the same type of SSAS Tabular model connection. The Sales and Budget dataset, in turn, is referenced as a remote dataset by the Sales and Budget Europe dataset. As illustrated by the Budget Excel source, any of these data models can have a variety of connectors used to import or connect to various data sources. You can create relationships between the tables from different sources as long as they don't violate the rules for existing relationships, either in the local data model or relationships that exist in the underlying remote data model. Since the local model developer may not have visibility to the entire schema for the remote model, this is a complexity to consider when using composite models. This is an example of why composite models are not for beginners. Remote data model From the perspective of the "local" data model that is open in the Desktop designer, when a published dataset is referenced by another data model, it is referred to as a Remote dataset. Local data model Any published dataset (within the limits of this feature) can be used in a remote connection. The data model that connects to a remote dataset is referred to as the Local dataset. Relationships may be created between tables from different remote datasets. These relationships should generally behave like they do in standard data models but there are some limits and subtle behavioral differences. There are still features of Power BI and Azure Analysis Services features that are unsupported in Gen2 composite models, so use this with caution and consult the documentation for details: Using DirectQuery for datasets and Azure Analysis Services (preview) – Power BI | Microsoft Docs. Remote table relationships What are those funny icons on the relationship lines between local and remote tables? These denote that the relationship is "limited", otherwise known as a "weak" relationship. This means, in simple terms, that since the relationship is defined using columns from tables in different DirectQuery sources, the VertiPaq engine cannot guarantee that the usual cardinality and key uniqueness rules can be enforced. These rules are inferred by the modelling engine but could potentially be violated and produce inconsistent results. By defining a relationship, as the model author, you are essentially stating that you expect the data to conform to cardinality and key-matching rules without the ability to force the model to behave that way. Alberto Ferrari wrote an extensive article explaining these relationship types in detail. New Composite Model Features I'm not going to walk through an entire demo because this would repeat the same steps as my previous post from December 2020, but I will get started (again). If you are interested in stepping through the process, I suggest following the instructions in that post. There are a few significant improvements from the previous design experience: Duplicate measure names are automatically resolvedIndividual tables can be selected when adding a new remote dataset connectionTables can be added and removed after the initial designObjects can be renamed after connecting To start using this feature, make sure you have the DirectQuery for PBI datasets and AS switched on in the Power BI Desktop Options dialog. Restart Desktop to apply any changes. The Power BI tenant must have support for XMLA endpoints enabled. This is managed in the Admin Portal under Tenant settings > Integration settings. After all datasets that you intend to use in remote connections have been published and access is granted from the respective workspaces, open or create a new PBIX file in Power BI Desktop. Here are the two published datasets I'll use in my new composite model: Use Get Data and connect to a published Power BI dataset. After the live connection is made to the remote dataset, the message shown below will appear in the status bar. This is subtle and easy to miss. Click the link titled Make changes to this model to update the data model and convert the connection to use DirectQuery for AS. You will be prompted to confirm the change. It is a good idea to make a backup of the PBIX before using a preview feature but there is little risk since this is a brand-new file for experimentation. However, keep this in mind for complex projects. After making this change, using Get Data to connect additional datasets will result in DirectQuery connections. For each, you are prompted to select only the tables you want to include in the local data model. If any objects in the remote model have the same name as those in the existing local model, those names will have a number appended to ensure uniqueness. This is true for tables and measures. In the model diagram view, tables from the same remote model are displayed with a unique color in the heading. The default colors for the first two remote datasets are blue and purple, shown here. These colors are similar so watch carefully. Just like a like a standard Import model, you can create relationships between any of the tables by dragging and dropping the key fields. Always double-check the relationship type and cardinality. Note that the relationship I just created between the Store and Geography 2 tables is incorrect (1:1 and bidirectional) which I will correct by editing the relationship, changing it to be one-to-many and a single-direction filter. You cannot create a relationship between two tables (whether local or remote), if it conflicts with an existing relationship. This is bound to become a challenge in more complex composite models, but it is a reality in dimensional model design. Objects can be added, removed and renamed This is a huge improvement over the previous design experience. Tables can be removed one-by-one from the data model and additional tables from the same source can be added later by repeating the steps using the Get Data options. Tables, fields and measures can also be renamed. In short, I think we're in a good place now, and this is a big improvement. What About Performance? There are many factors that can adversely affect the performance of reports, calculations and user interactions with a data model. Categorically, composite data models are bound to be less performant than a simple Import data model having all data residing within the same memory space. That's just a fact. Experience has shown that DirectQuery has performance impacts just due to the nature of query mechanics, and interactions between different databases, and network latency. In many cases, performance will be a trade-off for the convenience of data model consolidation using DirectQuery and composite models in general. Optimization techniques like aggregations will help. Time will tell with this new capability. With this incredibly powerful new tool there will be many opportunities to implement poor design choices and to take Power BI to – and beyond – its limits. There are several methods to tune and optimize models to run faster and perform well. I'm personally anxious to start using production-scale data with Gen2 composite models to test the water beyond small demo datasets. Please share your experience and I'll continue to share mine. You may not know that I have been travelling the country interviewing community leaders from the Microsoft data platform, on a new video blog: Data On The Road (dataontheroad.blog). Earlier this year, we travelled across the US in our motorhome, "The Data Bus" and stopped in several cities to chat with friends and colleagues who speak at conferences, write books, lead user groups and teach people about thriving in the data field. I had a chance to chat with folks like Bob Ward, Buck Woody, Adam Saxton, Kelly Kaye, Chris Finlan, Matthew Roche, Kellyn Pot'Vin Gorman, Leslie Andrews, Oz Du Soliel and many others. We also talk about how to work effectively from home and remotely, the importance of community and how these intense data experts unplug to balance work and life. Please check-out DataOnTheRoad.blog, and subscribe to see new videos, interviews and posts coming out in the near future, on this page. Seven days and counting… For the past eighteen years, thousands of data tech professionals would travel to the PASS Summit conference held somewhere in the US, paying for airfare, hotel and conference admission. This year, the PASS organization (Professional Association for SQL Server) is rebooted as the PASS Data Community, led by Red-Gate Software, and the virtual summit is free to attend online. Hundreds of speakers will deliver training sessions and informative lectures on a variety of important data tools and industry skills. This year, I am speaking about Paginated Reports in the Power BI service. I will reflect on the progression of SQL Server Reporting Services (SSRS) through past PASS presentations and then talk about how Paginated Reports integrates with the Power BI platform. Register to attend the PASS Community summit Two of my colleagues from 3Cloud are also presenting at the PASS Community Summit next week. There is just no good reason not to attend the summit his year, so register and make time to attend selected sessions. To reach the global audience, some of the keynotes and sessions are presented live. Others have recorded content followed by a live Q&A chat with the presenter. May I remind you again that the online summit is FREE, so participate, volunteer and attend! There is so much to learn from world-renowned experts. PASS Community Summit speakers are the subject matter experts who write the books, teach the classes, provide expert-level consulting and actually develop many of these products and Azure services. Data On The Road As we are traveling in the Data Bus, I have had the opportunity to interview server data community leaders and conference speakers, including those highlighted in this post. Here, Leslie and Andie talk about their topics and passion for data and community. Watch all of the interviews and follow us as we visit conferences and data community leaders in the Data Bus. Learn Azure Data Factory from Leslie Andrews Becoming a Data Engineer with Azure Databricks, from Andie Letourneau Please subscribe to Data On The Road blog to see the community leader interviews as we travel across the country. I just wanted to give a big big Thank You to Adam Saxton and Patrick Leblanc at GuyInACube.com for inviting me to participate in the Power BI Livestream yesterday. It was a blast and an honor to hang with you and the Saxton clan (even the sheep and the great dane!). You can watch the replay for the Saturday morning YouTube livestream here: I'm trying to minimize excessive cross-posting between the my long-time blog, SQLServerBI.blog and the new blog: DataOnTheRoad.blog, but we are still getting started and just wanted to make sure you all got the word. Please subscribe to DataOnTheRoad.blog if you are interested in following the data community leader interviews along our journey in the Data Bus. Regular technical blog posts and videos will continue to be here on SqlServerBI.blog. Here are a few quick shots from yesterday. Interviews with Adam will follow in the next few days. After the livestream, Adam and I sat down and talked about the genesis and evolution of the Guy In A Cube effort. In our interview, he told me about how he got started in his 16 year career with Microsoft, and his love of data, community, gadgets, games and video. You can watch the interview in this post: An Interview from The Cube, with Adam Saxton – Data On The Road Only the most dedicated data professionals are willing to invest their personal time and spend any part of a weekend improving their skills. Having organized many SQL Saturday events, I know that there is a big difference between those who punch out on the proverbial time clock on Friday and those who love working with data so much that they will invest their personal time on a Saturday. One of the greatest examples of this kind of dedication is the group who participate in the weekly Power BI Live Stream from Guy In A Cube. When Adam invited me to participate in the weekly Guy In A Cube Power BI livestream on Oct 23, I was elated. I am honored to join. If you are not already familiar with the Saturday morning live stream, you can join by subscribing to the Guy In A Cube YouTube channel. The channel is member-supported. Patrick and Adam do all of this outside of their day jobs at Microsoft and then member donations help cover their costs. I encourage you to support the channel with a donation but it is not a requirement. Even without supporting the channel, you can attend the hour-long livestream. Everyone can chat and submit questions during the first half and then the second half of the session is open for members to submit their questions. So, why am I going to be "in the cube" rather than joining remotely? Well, Houston is one of the stops in our DataOnTheRoad journey in the Data Bus! I'm interviewing Adam and other data professionals as we tour the country, working on the road, on the way to the Orlando Live! conference in November. It occurred to me that we have put so much effort into promoting best practices and proper design that there is far less information about how to create bad reports and data models. In that light, the purpose of this article is to talk about what to do if you want things to go poorly and make sure your projects fail – if not immediately, then sometime in the future – and if not for you then for whoever inherits the work that you have done. I welcome your comments and thoughts about the worst practices you have seen in Power BI. Post your ideas in the comments below. What examples have you encountered of sure-fire ways to make projects go badly? Keep in mind that the rules of "good" and "bad" design might be different for small, informal projects vs larger-scale, formal projects where longevity and scale are more important. Arguably, bad practices for enterprise-scale, multi-user report projects might be just peachy keen for small, informal desktop reports for one user with limited data. This article is written from the perspective of formal, larger-scale projects. So, you've taught yourself how to use Power BI! You've Googled some questions and let YouTube show you how to cobble together a little data from different places, wire-up a few tables and drag-and-drop fields to make some attractive visual reports. It wasn't all that hard to do, right? You've demonstrated your new Power BI design chops and now all the managers in the company want more cool charts and reports. Since the first round of reports were so well received, the company bought licenses so everyone in the organization can use Power BI to run their business. Welcome to the big leagues, folks! Just import data from all the corporate systems, desktop Excel files and whatever web feeds you can find. If it's on the Internet it MUST be right! Munge all this data together and create dashboards to help the executives run the business! This guide demonstrates common examples of how enterprise Power BI projects are bound to fail using self-taught undisciplined techniques to build larger, enterprise solutions. Most, if not all of the examples in this and the posts that follow, are examples that my colleagues and I encounter when we go in the fix failed Power BI projects. Bad practice #1: Keep PBIX files on your local computer rather than shared storage Many Power BI projects evolve through stages of prototypes, so they might create multiple PBIX files. After some trial-and-error, solution developers often save-off multiple files with specially-encoded names including things like the date, their initials and other information each one with different features and techniques. One example might be: FinancialAccountingReport_20210615_BobR_TestDisconnectedSlicer and another one might be FinancialAccountingReport_20210618_BobR_FixedForecastMeasureButNowItRunsSuperSlow. Reminds me of when I was eight. We would write notes with cyphers that only the writer and recipient would understand. There is nothing wrong with making a few backup copies of your work in a convenient place where you will find it. someone has to keep track of it all and recover work when there is a staff change. Without a deliberate effort to exercise discipline, delete or archive the old files and then store the most current copies in a central storage location that gets backup and synced to the cloud; you are sure to have a mess on your hands. Worse than that, if the project ever needs to be turned over to someone else, they wouldn't have a clue about how to continue the work. I always ask myself what would happen if I had to hand-over my work to another developer tomorrow? Would they have what they need to continue and would they know where to pick things up? Some transition documentation is good to have in this case, but make sure the master files are in a safe location. Better yet, keep them in a source code repository like Azure DevOps or GitHub, or in OneDrive or SharePoint with automatic file versioning. Oh, wait… this article is about creating bad projects! Bad practice is to just leave files on your local drive. If you want things to go badly, that's what you should do. When you leave this mess for someone else, they will have no idea about what you did or how to continue the work. Bad practice #2: Keep the only copy of a report and dataset published in the online workspace to download when you need it Today, we're all about using the cloud to safely store files. This is a good practice. You can store all your PBIX and other important files in a OneDrive folder where they are backed up to the cloud and the files are even versioned so you can revert back to an earlier version if you do something silly and need to recover. However, relying on the option to store your one-and-only copy of a report or data model file in a Power BI workspace is optimistic at best. There are several conditions that will disable to ability to download a PBIX file. The thing is that enterprise-scale data models need to take advantage of features that are not supported in Power BI Desktop (and probably never will be). This is by design. Desktop is primarily a self-service tool and most of the feature we use in enterprise-scale projects require the model definition file to either be saved to a JSON model definition file (e.g. "Model.BIM") or to a PBIX file with an internal model version number unusable by Power BI Desktop. I've heard from numerous novice developers who were accustomed to publishing their latest PBIX file to the service, only to discover that after enabling some advanced feature; they could no longer download the file. If you want to shoot yourself in the foot, do not keep a master copy of your model and report files (which you are managing separately, right?) in OneDrive and roll the dice that you will be able to download a copy when you need to edit the model or report in Desktop. Bad practice #3: Import source data files from your local file system and then publish the report to the service If your source data files are on your local computer, after you publish the data model to the service as a dataset; you will not be able to refresh the dataset from the service unless you take a few more steps. There are a few options to remedy this situation, such as: Move the source file folder to a shared OneDrive folder or SharePoint library, and then use that connectors to refresh the dataset from the service.Move the source file folder to a network share and then setup the enterprise gateway to import and refresh the published dataset from the service.Install and setup a personal gateway to fresh files from your local computer. Option #3 is a sure fire way to run into problems down the road. This might work for a small project managed by a single developer. But, if you have source files on your laptop which happens to be asleep or shut off during a scheduled refresh event, that's not gunna work, and the refresh will fail. Another common gotcha is that if you import source files that are stored in a OneDrive folder, and use a local syched folder as the source; the connection is to the local file system (like "C:\Users\Paul\OneDrive\…") and not to the OneDrive service in the cloud. In this case, the data refresh will try to read from the local folder which is inaccessible from the Power BI service. Bad practice #4: Use hand-written SQL queries to import large database tables SQL is the tool of choice for most database professionals, so it would stand to reason that you should write a SQL query when connecting to a relational database (like SQL Server or Oracle.) Right? Sure, if you want things to be slow and inefficient. Best practice is to connect your queries in Power BI to read from a database table or view, and not to use SQL statements so that Power Query can employ a method called "query folding". This means that the Power Query processing steps, where possible, will be translated into SQL or the native query language for the data provider. There are advanced techniques to promote query folding over a SQL statement, but this is the exception to the rule. If you need to write SQL logic to join, filter, group or sort data from multiple tables before it hits Power Query, write a view and store it in the source database. Simple answer: If you want things to go badly, write SQL queries in your Power Query queries using Power BI Desktop. Bad practice #5: Use DirectQuery for "real-time" analytics Power BI includes an awesome capability called "DirectQuery" that enables certain data source engines to handle query operations as user interacts with report visual, rather than using the VertiPaq in-memory analytic engine, to process queries. DirectQuery doesn't use the VertiPaq storage engine, which is really the heart and soul of Power BI and SSAS Tabular data models. With DQ, records are read live from the source. The downside to DQ is huge. Most interactive report visuals can take several seconds to load, and some DAX functions simply can't be used due to resource overhead and performance cost. Database professionals often consider DQ to be a cure-all solution because it enables familiar database technologies. Everything is managed centrally, there is no record duplication in the data model and users see the current state of a record when they interact with reports. That's all fine and dandy but compared to Import mode, DirectQuery performance can be several orders of magnitude slower. Import mode is fast because because data sits in memory, compressed by column. DQ is slow because queries must be translated and pushed down to the data store, then results must be sent back. DirectQuery is a powerful feature when used strategically to address a few specific problems. It is a good solution for massive fact tables that would otherwise use a lot of memory. Just filter large tables and don't do a lot of grouping and aggregation over a large set of records. In a Composite data model, with most tables using Import mode, and one or a few detail tables using DirectQuery, you can a achieve "best-of-both-worlds" solution by allowing users to drill to details against a filtered set a records. If you want your visual, analytic reports to be really slow, use DirectQuery for all the tables. Bad practice #6: Combine source data and load one big table into the Power BI data model People who approach data problems in Power BI the same way they do in Excel, tend to combine source tables into a large spreadsheet-like table. That approach creates an seemingly easy-to-navigate grid of data with a lot of columns; but you can lose a lot of functionality and create a very inefficient model this way. The VertiPaq, Tabular modeling engine was architected around the premise of "star schema" dimensional models. It's not that Power BI won't work with flattened data, or that you "have to" build a star schema, with separate dimension tables for every attribute! The the natural rules of data behavior just cause Power BI to work better when data is modeled in related fact and dimension tables. There are several DAX functions that expect records to be filtered through a relationship. One of the most common examples is a date dimension table, used to filter a fact table to perform time-series calculations. But that same behavior can exist for Products, Customers, Geography and other filtering or grouping attributes that can be used to filter a related table of numeric business facts. If you want to struggle with DAX calculations and limit users' options to interact with your data, pound your data a large, flat table, rather than individual dimension or look-up tables. This question comes up every few years in SQL Server Reporting Services. Of course, in Power BI Paginated Reports, we have the same features. A couple of days ago, Karthik posted this question as a comment to my post titled Chapter 7 – Advanced Report Design: I am working on a SSRS report where the grouping is done to group the records in to multiple tabs/worksheets. When the report is exported to excel, the worksheets has the default name (Sheet1, Sheet2, Sheet3,…). Here I want to override the default worksheet name with (Tab1, Tab2, Tab3, …). The number of tabs/worksheets varies each time depending on the parameter selection by the user. How to address this? any suggestions please. There are a couple of requirements to set this up. First, you must drive report pagination using a Group and then set a Page Break on the Group. This figure shows that I am grouping my table data region on the Brand field and then setting a page break on that group. You can get to the Page Break settings by using the little arrow next to the group item in the Row Groups pane or you can use the properties window like I'm doing here. Next, you want to give the page a name under the PageBreak properties. Expand PageBreak and then use the Expression builder in the PageName property to use the grouped field name in an expression. In this case, I am referencing the Brand field. The result is shown here when I open the file resulting from exporting this report to Excel. You can see that the worksheet tabs are titled with each Brand value and each sheet contains that product information group by the corresponding brand. That's right! I'm starting a new blog site today. Just to be clear, it will not replace this one. I will continue to post here at SqlServerBi.blog about everything related to Business Intelligence best practices, Power BI, Paginated Reports, SQL and the Azure Data Platform. The new blog is called Data On The Road and it is dedicated to connecting with the professional community while on the road as a working technology professional. In May of 2021, my wife and I decided we needed an adventure so we transitioned our living space into a new Tiffin Class A motorhome. We had it outfitted with a private office and tech so we could stay connected while working fulltime from the road. We call it the "Data Bus". We're just getting started and taking short trips to practice and work-out the kinks – but it has been great so far. We hit the road for good in the Fall to go experience the US from the Southwest, the South, the East coast – and who knows where? We're still assembling the tech and working out the logistics to visit and interview friends and colleagues along the way. In the database and BI community, we're a dedicated and social bunch so let's talk about work and about fun! We might chat a little tech but the purpose of this blog – and this adventure – is to talk about work/life balance. In our jobs, we work hard and play hard – so we want to know what our community friends and leaders do when they need to get away. How do they balance their careers with other activities? Please follow both of my blogs but head on over to DataOnTheRoad.blog and follow us on this new adventure! Part of the the series: Doing Power BI the Right Way Later in this post, you will find a 27 minute framed video including a series of interviews and discussions with a few of my colleagues. They provide perspectives and recent experiences related to this topic. The best approach to design and maintain large data models depends on different factors. However, many years of experience have proven that there are best practices to embrace and common mistakes to avoid. I was just talking to a consulting client about the best approach to build a data model and he told me something very interesting about the way they were loading data into Power BI. He said "We don't use facts and dimensions, we load all of our data into one huge table." He said that their data model performs well and that it meets their reporting needs. It is a difficult point to argue, when something is working at the time although the design might not follow the accepted rules. Life is like that and there are plenty of analogies to make the point that a practice, even a real bad practice, might solve a problem for a period of time and under certain conditions. You can drive a car at excessive speed to get to your destination faster. You might not get caught by the police on that day and you might not crash but eventually, if you make it a habit, this practice will catch up to you. Data is like that. If you don't play by the rules, you limit your options. Bending the rules lets you move faster and sometimes with less hassle. But, as the project scope expands – and after adding enough data or other complexities to the solution, it will not endure. The data model won't perform well, won't load the correct data or it just won't be reliable. This post will explore the realities of best practice design for large data models; some important considerations and trade-off decisions when working with both "big data" and "large data". One of the great challenges with Power BI is that it is so easy to build simple data models and reports quickly. But, often when those projects need to grow-up, they aren't ready for enterprise use. This isn't a problem with the tool, it is a problem with the way it is often used. If someone says "Hey, Paul, I need you to create a report quickly. Can you import data from these three sources into Power BI and create a quick dashboard report?" I might say "sure, let's just do this:" Merge records from multiple files and queries into a single table in the model.Use in-line SQL queries in Power Query to join source tables together and apply transformation logic.In reports, summarize numeric columns from a table in the data model, rather than defining DAX measures.Reload the entire model when source data changes. There is nothing inherently wrong or bad about using these quick-and-easy techniques in a smallish project, but if it needs to grow, scale and be future-proof, you may need to follow a longer path to achieve a more durable result. The answer to a question we often ask at the beginning of a BI project: "what information do we need include in the data model?", is often the same. I think Freddie Mercury said it best: "I want it all and I want it now!" More specifically, stakeholders might say that they want: A lot of rows of detail or historical fact records, (lets say tens of millions or billions of rows.)Many columns from the source table(s) so they can slice, filter or group values on anything imaginable.Reports must perform well and return results quickly. …to these requests, we might say "pick any two but you can't have all three". Maybe you can have all three to a certain point but not to an extreme. Tall and Wide tables A "tall" table contains many rows or records. A million might be a substantial number of fact table rows in a desktop Power BI data model, and row counts in the billions are not out of reach in an enterprise-scale solution built in either Power BI or Analysis Services. The VertiPaq analytic data processing engine (the data modeling engine behind SSAS, AAS, Power Pivot and Power BI) is very good at handling very tall tables when the columns are conservative, numeric data types; especially when they contain repeating, non-unique values to aid columnar compression. In rare cases, Power BI data models can even read and process PetaBytes of data! The point is that tall tables are very achievable. Tables with many columns can be managed by the modeling engine but tall (many rows) and wide (many columns) are often not a good combination. The fact is that analytic reporting solutions shouldn't include the whole kitchen sink. The problem often stems from developers not building data models based on on business requirements but simply exposing all the available data just in case someone might find it valuable. This is just not the way BI works. If you don't need certain columns to meet the current report requirements, leave them out. Columns can usually be added later when they are needed. Additional tables can be also added until the model exceeds its design goals. Herein lies the need for balance. The thing is that simple data models can be relatively easy to build and often don't take up a lot of space. It may be more cost-effective in the long run to create another data model with some duplicate design elements, than to spend all of your time and energy trying to craft an all-encompassing data model. Large Data and Big Data By "Large Data", we generally mean that tables contain a high number of rows. Conversely, "Big Data" refers to a set of technologies that were engineered to manage large volumes of data records using distributed storage and processing. Big Data In the first wave of cloud computing, we saw the introduction of "Big Data" which was a new concept for data professionals. Technologies like Hadoop and Apache Spark use distributed computing to process larges sets of data in parallel, using highly scaled-out architecture. The term "Big Data" doesn't just mean that there is a lot of data but it really means "distributed data processing" which can potentially be used to handle very large data sets. Like any architecture, there are strengths and weaknesses. Big Data implementations like Spark and Databricks can transform and process an enormous amount of information very quickly with portions of the data processed on distributed nodes. Conversely, the VertiPaq engine at the center of a Power BI and SSAS tabular data model, is not a distributed architecture. Because VertiPaq keeps all of the data in memory on a single computing node, it can perform calculations extremely fast. For this to work, data must reside in contiguous memory on a single computer. So, Power BI is not Big Data but it can be used in conjunction with Big Data sources when you use these respective technologies the way they were designed and architected to work. Complex Models Unique problems require unique data model designs. Sometimes a Power BI report and accompanying data model just need to address a few business reporting objectives. As time goes by, additional requirements are introduced and the model grows. Along the way, certain data challenges might require additional data massaging or new tables to bridge tables and address special needs. Over time, a model can get complicated. This often raises the question: should we create one big, complex data model to address many requirements or multiple, less complicated data models? There are trade-off decisions either way. It is often possible to engineer larger, high-value models. However, you have to weight the cost to maintain a complex model with the relative simplicity and cost savings of less-complicated models. Sometimes starting over is cheaper than layering more messiness onto a data model with a shaky foundation. Partitioning Once considered an essential activity in the design of any data model, partitioning breaks a table into multiple storage units that can be managed and loaded separately. Partitions containing unchanged records don't need to be reprocessed, and the smaller partitions containing new records or those that have changed, can be processed quickly and in parallel, so they don't block other operations. Video Interviews This video runs about 27 minutes and includes a series of interviews and discussions with some of my colleagues, with their thoughts and experiences on this topic. This is going to be a very busy week for presentations Iam presenting with five sessions scheduled on the topic of data modeling! Data modeling is the heart and foundation of a useful Power BI solution. If you can get the data model right, most everything else falls into place. Correct data modeling is relatively simple on the surface but can be challenging in practice. Evidence of the interest and importance of this topic is that I submit abstracts on a few different topics for these events and this is the one keeps getting selected. Silicon Valley Microsoft Data Platform MeetupTuesday, April 20 at 6:00 PM Pacific Time Power BI SummitApril 19-23 – multiple sessions and times The session will be repeated three times for attendees in different time zones. BI & Analytics community the Norwegian Computer AssociationApril 21 5:00 PM Central European Time Supporting materials are available on the Presentations page on my blog here: Presentations | Paul Turley's SQL Server BI Blog. In addition to to the material and demonstrations I present in these sessions, I wanted to share some additional thoughts which I have recorded in this short video: Navigation between reports is the hallmark of an interactive reporting solution, enabling the ability to drill-through and see relevant details and contextual filtered information in a target report. Power BI and Paginated Reports both offer navigation capabilities suited for different purposes. Having the ability to navigate between reports can provide a tremendous amount of power and flexibility. This is simple math: with two different report types, there are four different possible report navigation combinations. They are: Power BI report to Power BI reportPower BI report to Paginated reportPaginated report to Power BI reportPaginated report to paginated report Continuing this series of blog posts to highlight our recently published eBook about Paginated Report recipes, this post summarizes the technique detailed in recipe #4, "Drill-through from Power BI to Paginated Report On-premises". If you have worked with the native SSRS report drill-through using report actions, that capability is not yet baked into the product. It probably will be some day but likely not soon. That's OK because this technique, using URL navigation, works quite well. There is one other twist which is that the URL parameters are a little different when comparing on-premises Paginated (SSRS) reports to Paginated reports in the Power BI service. The technique in in this recipe uses an on-prem target report. You can apply the same pattern in the cloud with sight modifications and I will post an example in the near future. This image depicts the source Power BI report with links that navigate to the target paginated report, showing filtered details: The mechanism to enable navigation is a report URL produced with a DAX measure. The code emits a URL with parameters that preserve the row and filter context of the selected item or record. This URL measure is exposed as a link in a table or matrix visual in the Power BI report, optional with a URL "link" icon: To see the instructions in detail, follow this link to the recipe: 04-Drill-through from Power BI to Paginated Report On-premises | Paul Turley's SQL Server BI Blog First and most importantly, I want to acknowledge and thank these contributing authors who wrote report recipes and working samples included in this eBook: Robert BrucknerKirill PerianKathi KellenbergerStacia VargaTomaž KaštrunTom Martens As I am working on posting the initial report recipes, I wanted to share some comments about the first two recipes. These recipes are step-by-step instructions that lead through the start-to-finish process using provided sample database, and can be used with SQL Server Reporting Services (SSRS), Paginated Report Server or Paginated Reports published to a Power BI Premium capacity workspace. Reports may be authored with Power BI Report Builder, Report Builder or SQL Server Data Tools/Visual Studio. Additional recipes will follow. The eBook with all of the recipes will be accessible from the main menu here on the blog. Each recipe is published as a page rather than a post, so you don't have to search and try to figure out when each one was originally posted. As of tonight, I have published eight of twelve recipes that have been completed thus far. I'll get others posted in the near future. Please enjoy these and watch for more paginated report recipes here: Paginated Report Recipes eBook | Paul Turley's SQL Server BI Blog The first two recipes in the Paginated Report Recipe eBook are about applying alternate row and group shading. Dating back to the early days of printed reports, this style of reports were called "greenbar reports" because they were literally printed on continuous, pin-fed paper that had alternating shaded green and white rows to make it easier for someone reading the report to follow-along. As the industry evolve from scrolling paper reports to laser-printed pages and then to on-screen reports, the greenbar effect was still useful. Recipe #1 introduces expression-based styling by changing the background color using the modulus (divisional remainder) of the row number in a table data region. If the row number is on odd number, set the background color to a shading color. Otherwise, set it to white or transparent. 01: Alternate Row Table "Green Bar" Report | Paul Turley's SQL Server BI Blog In recipe #2, contributing author Kirill Perian takes this technique to the next level by maintaining the alternate shading across column and row groups in a matrix data region. The method is a little more involved than just using the row number. He uses a hidden column to track the column group and then an expression to reference that value to manage the background shading for each row. 02:

Alternate Row Shading in Matrix (with Column Groups) | Paul Turley's SQL Server BI Blog The end results looks similar to the simple table report using in recipe #1 but the problem is a little more complex in a matrix report because of the column groups. The advantage of this technique is that it will work across row and column groups at multiple levels. At last, the second edition of "SQL Server Report Recipes" written by Paul Turley, Robert Bruckner and a host of contributors; is being released, a few recipes at a time. This time around, it will be a free book published through my blog and perhaps other sources. I've posted the introductory chapter and six report design recipes. You may also download the sample SQL Server database used in all the recipe reports. I sincerely hope that you and others will use this as a free resource to serve the common interest of the greater community. Please spread the word and send those who might learn and benefit to this page. The work on this book really started about eighteen years ago… Late December back in 2003; what a very special time for me! SQL Server Reporting Services was due to be released to the market and I had been using the pre-released version to integrate reports into a custom web application. After using a variety of products like FoxPro, Access and Crystal Reports, and after mastering essential T-SQL queries and .NET programming, using SSRS was a dream. It was simple and elegant, programmable and powerful. I was so excited that I wanted to tell the world, so I recruited authors to write our first book about Reporting Services. Three editions later, I teamed up with Robert Bruckner, one of the lead developers on the SSRS product team at Microsoft, to write a different kind of book. Robert and I had both blogged quite a lot about report design techniques. Robert had done some crazy things in SSRS that no one knew was possible. He found very creative ways to develop games using SSRS such as tic-tac-toe, hangman and Battleship. Honestly, there are easier ways to do game programming, but he proved that almost anything was possible if you were willing to think outside the box and maybe even draw outside the lines a bit. Several contributing authors have worked tirelessly over the past difficult year to make this eBook a reality. Others, through no fault of their own, sign-up to contribute to the book but "the year that shall not be named" had other plans. At first we paused, and then we stopped and then we regrouped and decided to publish a shorter but comprehensive, modern kind of book. Please visit the new Welcome and Introductory page with the Table of Contents to the current and future recipes: Paginated Report Recipes: 2020-2021 | Paul Turley's SQL Server BI Blog Part of the the series: Doing Power BI the Right Way (link) Although my professional focus is building enterprise-scale BI solutions, I've created my share of informal Power BI reports that were put together quickly, with the goal to create something "good enough" rather then achieving perfection. This guide is about designing proper and formal solutions but these practices apply to any Power BI project that needs to survive future maintenance. When you need to load data into a data model, you have a few options and the right choice is going to depend on a few factors. This equation is usually a balance between quickly and conveniently generating a table or using a disciplined approach to get reliable data from a sustainable source of record. The following image shows that data can be transformed in the source (or before data is loaded into the source) or within Power Query using Power BI Desktop. Convenience or Scale? Life is full of choices and trade-off decisions. Let's say that you need to create a lookup table containing sales regions and this information doesn't exist in the source database. You could easily create a new table in Power Query using the "Enter Data" feature, and just manually enter the sales regions. This would solve the immediate problem with very little effort, but how will the table be maintained if sales regions are added or change in the future? We could keep the list of values in an Excel file stored in SharePoint for a business owner to maintain when information changes. You could also go all the back back to the beginning of the process and load a dimension table in the source database. IT professionals might take the hardline and say that all data must flow through the data warehouse regardless of the effort and cost. Which of these is the right choice? …it depends on the scope reporting project, and the long-term ownership of the solution. Sometimes quick and convenient are OK but particularly tables that need to scale and handle larger data volumes in the future, the following guidelines are critical. I'll get to the point and then explore the details afterward. Start with these fundamentals: A proper data model is the heart of a Power BI report solution.The purpose of Power Query is to shape and prepare each of the tables loaded into the data model.The tables, fields and measures in the data model should be uncomplicated and user-friendly; intuitive and easy to navigate. Recommended Practices Following are the general best-practices I religiously apply when creating queries. After briefly describing each recommended practice, I'll break it down and review some in detail. PracticeExplanationUse parameters for query paths & connectionsParameters are used to make the solution portable. Any connection information, like a file path or database server name, should be stored in a parameter so it can be changed without modifying query code.For large SQL tables, reference either tables or viewsPower Query has optimizations built-in to work with different data connectors. Several connectors support query folding, where Power Query translates query steps into the native query language.Starting with a SQL query rather than selecting a table or view from the list of database objects will ensure that query folding will not work. When possible, start with a table and if you need to use SQL to prepare data before loading it with Power Query, create a view.Limit large table volume using data range parameters, compatible with Incremental RefreshIncremental Refresh enables the Power BI service to partition large tables and only load data that changes rather then the entire table when the dataset is refreshed. This was once a Premium-only feature that now works with shared capacity licensing with datasets up to 1GB in size. Even if you don't intend to use the Incremental Refresh feature, using a pair of date range parameters allows you to filter large tables and keep the PBIX file size small. After publishing the file to the service, you can update the parameters and load more records.Create two date/time type parameters named RangeStart and RangeEnd, and then add a date range filter according to these instructions.Remove all unneeded columnsResist the urge to leave columns that you are not sure that you need for reporting. In each query, remove all unneeded columns early in the sequence applied steps. The easiest way to do this is to use the Choose Columns button on the Home ribbon and deselect columns. To change the selection later, click the gear icon next to the Remove Other Columns step.Rename using title case for all table names and column names that will be visible in the data modelAlthough it may seem trivial, it is absolutely necessary to apply friendly naming conventions to all tables and fields. Chris Webb wrote an excellent post and about object naming conventions. As a rule, rename all columns that will not be hidden in the data model, using friendly title names (with spaces and mixed case). There is no need to rename primary key, foreign key and other utility fields. After tables are added tot he data model, hide those fields to remove clutter and confusion for report developers (even if you are the report developer).Explicitly set data type for all columnsColumn renaming and changing the data types an be time-consuming work but be meticulous about check every column returned by the query.Consolidate multiple steps of the same type (such as change type and rename columns)As query design evolves, you will inevitably create inefficient queries with redundant steps. There will always be opportunities to improve the design, often by consolidating and reorganizing steps.Rename key steps to document purpose and aid future developmentRenaming query steps allows you to understand the function and purpose of each step in the Applied Steps list. This creates a self-documented series of operations that will be easier to maintain down the road.Add code comments in the M script and/or step descriptions to document queriesYou can further document query steps by changing the step Description in the Properties window or by adding code comments in the Advanced Query Editor.Use steps that promote query foldingMany query steps allow records to flow-through without blocking subsequent steps. There are also certain transformation steps that must read all the records into memory to apply the transformation.Move steps that break query folding as late as possibleSteps that support query folder, that can be translated into the native query language of the data provider, should be performed first. If non-foldable steps can't be avoided, they should be performed as late as possible in the sequence of query steps.Perform Pivot, Unpivot, Transpose transformations only on small result setsThese transformations must load all records into memory. They are powerful but have volume and performance limitations. Test them with production-scale source data.Create row-level calculations in Power Query instead of DAXRow-level calculations and derived column values can be performed either using Power Query or DAX, but M s a far more capable language for data transformation. Effective data preparation using Power Query will simplify and ease the burden of data model design. Regardless of the relative efficiency, if you perform all data preparation and transformation in one place, this can simplify maintenance in the future. There are less-common cases where DAX is the best choice to create calculated tables and calculated columns outside the scope of a single row.Avoid using DirectQuery when not warranted by latency and data volume requirementsDirectQuery has it's place in data model design but it should be an exception to the normal pattern of importing data into an in-memory data model. DirectQuery tables can work with simple aggregation calculations but don't perform well using many other DAX functions. Use DirectQuery to aggregate values over very large tables that won't otherwise fit into memory, or to support drill-through into non-aggregated detail records. These are advanced design patters that should be treated as rare and exceptional.Avoid using native SQL queries, especially with large source tablesNative SQL queries don't support query folding so they should be avoided when possible. This may be an acceptable exception to load low-volume query results but generally use a database table or view for query data sources. Use Date/Time values for dates (use Date or Time)Load column data only at the necessary level of grain so values in the data model will compress. In particular, reduce date/time values to date only. For time-level resolution, store date and time values in separate columns.Import or create Date/Time tables in Power Query rather than using auto-generated date hierarchiesAuto-generated calendar tables in Power BI Desktop are fine for self-service projects but purposely-built date tables will afford more flexibility.If available, use a date dimension table in your data warehouse or source database. Secondarily, generate a date table using Power Query.Date tables can effectively be created with DAX functions but if all tables are loaded with Power Query this gives you the convenience of managing all tables centrally. Breaking It Down Let's take a deeper look at some of these recommendations. Use parameters for query paths & connections For every file path, web address or server name in source connections; use a parameter. It is not hard to go back and edit source connection information using the Advanced Editor but the easiest way to build parameterized connections is to build them as you go. Start by enabling the feature "Always allow parametrization in data source and transformation dialogs" on the Power Query Editor page in the Option dialog. As you build each query connection, for most connection types, you will be promoted to select or create a new parameter. Here is an example of the parameters in a demonstration project. Without modifying any code or editing a query, any of these values can be changed easily. Here are two examples of parameterized connections. For the SQL Server connection, the server/instance name is passed as the first argument to the Sql.Database function. The second example concatenates the folder path (stored in the SourceFolderPath parameter) with the file name to create the fill folder and file path. For large SQL tables, reference either tables or views The best way to guarantee poor query performance with a relational data source is to start with a hand-written native query (like SQL) and then perform transformations on the results. If a query is based on a relational table or view, Power Query can generate the native SQL (and a few other supported languages) with a SQL statement rather than selecting a table or view. Use Query Diagnostics to Diagnose and Performance Tune The following image shows the results of Power Query Diagnostics which I describe here: Power BI Query Performance & Query Diagnostics | Paul Turley's SQL Server BI Blog. This post demonstrates how to capture timings for each step and overall queries. Part of the the series: Doing Power BI the Right Way (link) Power BI, more than any other Microsoft product in my recollection, offers more options and choices to architect and deliver a solution. Without compromise, Power BI can effectively be used to do anything from create a simple chart using an Excel spreadsheet, to enterprise reporting and analytics on a Fortune 100's massive data warehouse. At the end of this post, I will share a comprehensive list of resources and insights from Matthew Roche, a Program Manager on the Power BI Customer Advisory Team (CAT). To tease that series, I'll start with this quote from Matthews's blog: Succeeding with a tool like Power BI is easy – self-service BI tools let more users do more things with data more easily, and can help reduce the reporting burden on IT teams.Succeeding at scale with a tool like Power BI is not easy. It's very difficult, not because of the technology, but because of the context in which the technology is used. Organizations adopt self-service BI tools because their existing approaches to working with data are no longer successful – and because the cost and pain of change has become outweighed by the cost and pain of maintaining course.Matthew Roche, Building a data culture – BI Polar (ssbipolar.com) When should you use dataflows vs regular Power Query? I didn't jump on the dataflows bandwagon and struggled to adopt them at first. Frankly, Power Query is easier to use. The browser-based dataflows designer is quite impressive but it is not as responsive and convenient as the desktop app, so this is a bit of a trade-off. The power and value of dataflows becomes apparent when the business reaches a certain stage of data culture maturity. Before we can address the question of whether to use Power BI Dataflows, conventional Power BI queries, or any other approach to source and transform data; we need to briefly review different options for orchestrating a Business Intelligence solution in the Microsoft cloud ecosystem. Solution Architecture Choices On a scale of one to ten, with ten being the most formalized, strictly-governed and complex corporate reporting platform; the self-service Power BI option might range from one to four. For the self-service data analyst, working entirely in Power BI Desktop, data can be imported and transformed using Power Query. Tables are modeled, calculations are defined and data is visualized. This mode is simple and works well for small to moderate-scale solutions with less emphasis on data governance and centralized control. Even using this simple approach, data models can be developed separately from reports, certified and shared with multiple report developers and self-service report authors. So, to a point, business data can be managed and governed – but the queries in the Power BI solution read directly from source systems or files that are not curated for analytic reporting. Data Warehouses and Data Marts The "single version of the truth" or "golden record" repository, a data warehouse (or smaller-scale "data mart") is the ideal solution to store and manage reliable corporate information. The challenge with creating a central data warehouse to manage centrally-governed organizational data is that it is costly and time-consuming, however the trade-off is that self-service data models can be inaccurate and out of date. When business leaders need answers quickly, it is not always feasible to add more data sources to a data warehouse quickly. On the complexity scale of one to ten, versions of this option might be from seven to ten. A conventional DW/BI solution typically uses on-prem data transformation tools like SSIS to stage and transform source data into a central data warehouse built using a relational database product like SQL Server. Although viable for on-prem systems, this old-school architecture model doesn't embrace scalable and cost-effective cloud technologies. The first generation of the Microsoft cloud-based modern data warehouse can utilize several different Azure services. The components in following example are easily equated to the conventional data warehouse solution in the previous example. Azure Data Lake services as the staging environment typically using text files and structured file storage as an inexpensive landing area for ingested source data. Azure Data Factory is used to orchestrate and transform files and data streams into and out of the data lake – and the data warehouse. Depending on the need for scale and size, Azure SQL Database or Azure Data Warehouse (now called Azure Synapse) may be used for data warehouse storage. If your organization has a comprehensive data warehouse to serve-up all or most of the data needed for analytic reporting, this is probably the best fit for a Power BI solution in your business environment. Constructing an enterprise data warehouse solution is not a trivial endeavor, often involving as much effort to negotiate business process challenges as the technology development to implement the solution. The newer generation of the Azure modern data warehouse is a best-of-breed collection of tightly-integrated cloud services called Azure Synapse Analytics. Compared to the previous set of independent Azure services, Synapse Analytics provided a unified development and management interface. Apache Spark and other industry standard technologies designed for data science and platform-agnostic analytics provides the open source data prep engine. Azure Synapse is the evolution of Azure Data Warehouse, Microsoft's read-optimized, scalable massive parallel-processing (MPP) SQL-based database engine. Power BI Dataflows Dataflows can fill an important gap between purely self-service data prep and formal data warehouse solutions. If you don't have a comprehensive data warehouse to meet your analytic reporting requirements but need to provide more data quality control over standardized entities, incorporating dataflows might be the ticket. In its simplest form, dataflows provides reusable transformation logic (queries) that can be shared by multiple Power BI data models. Using dataflows deployed to a workspace can save data model developers from repeating the same transformation steps in multiple datasets. But these are more than just Power Query scripts stored in the cloud. A long list of capabilities are enabled by using dataflows. They can provide integrity and standard entity definitions stored in Dataverse (previously known as the Common Data Model) to enforce standard naming, data types and schema compliance among other features. In Premium Capacity, dataflow results may be persisted in Azure Data Lake Gen2 storage. This essentially allows you to use dataflows to create a moderate-scale data warehouse without a big investment. Entities may be linked to related entities which creates virtual joins and referential constraints. Other Premium features include DirectQuery, Computed entities and Incremental refresh – all managed in the dataflow rather than for each dataset. Integrations with Azure AI, Machine Learning and Cognitive Services allow you to utilize AI features without writing code. For example, in a recent project, we used AutoML on a dataflow containing high school student data to predict graduation outcomes. Dataflows Start with M Queries Dataflows begin with an M query, just like the queries in Power BI Desktop before adding the additional capabilities mentioned earlier. Queries are authored entirely in the browser but migrating from Power Query in Power BI Desktop is fairly easy. Start with a Power BI solution (PBIX file) in Desktop and open a query in the Advanced Query Editor. You can create a new dataflow in the browser and then copy and paste the existing query M code from Desktop to the dataflow designer. You do have to copy each query one at a time and there just a few compatibility differences but for the most part, it should be a one-to-one transfer. Building a Data Culture Matthew Roche from the Power BI Customer Advisory Team has an excellent 17-part blog series about Building a data culture. Dataflows sit at the crossroads between business process, data stewardship and technology. The industry has been throwing technology and software at data governance and quality problems for decades, with marginal success. It is much easier for data practitioners to acknowledge that these are multi-faceted business culture challenges than it is to formulate a plan to be successful. If anyone can effectively carry and delivery this message, it is Matthew. In this video series, he offers prescriptive guidance to enlist an executive sponsor, work with business stakeholders, and to navigate the landmines of a business landscape to a successful data culture transition. Honestly, I've only pursued this series in fragments over the past year and now that I've caught the vision, I plan to watch the entire series start-to-finish. It is that good. Think of it as Game Of Thrones with data. Resources Matthew also provides a comprehensive list of Power BI Dataflows resources here. Matthew recently presented to our 3Cloud Power BI and Analytics development team about using dataflows to promote a data culture. This presentation was an epiphany for me, that helped to better understand how dataflows fit into the BI solution puzzle – that's when the gauge metaphor popped into my head. I encourage you to watch and perhaps you will have a similar moment of reckoning. The Power BI Adoption Framework is a set of presentations from Microsoft that can serve as a checklist of important tasks and areas that should be covered in any Power BI implementation, large and small. These decks are also a great tool for adopting and sharing your organization's BI and Analytics strategy with business leaders and stakeholders. You can use them a s a picklist to assemble your own presentations. About three years ago when the data model development engineers from the Power BI product team told me they were working on the ability for tabular data models to share other published data models, that sounded amazing and almost too good to be true. They said it would be a lot of work to make this a reality. The Utopian dream of distributed datasets was publicly announced on the Wave 2 Roadmap last year and then the feature went to Preview in December with this announcement: Using DirectQuery for datasets and Azure Analysis Services (preview) – Power BI | Microsoft Docs. We've seen the new feature demonstrated and explained several times but now I'm exploring this exciting new feature for myself. For this test, I have two different data models in separate PBIX files. One published dataset contains Internet Sales information from the ContosoDW Azure SQL database and the other contains Store Sales information. These models contain many of the same tables and a few unique tables. Notably, each data model contains a different fact table. Here are the two published data models in the Contoso Sales workspace: After making sure that I had upgraded to the December 2020 version of Power BI Desktop, I open the Options dialog and enable the preview feature: DirectQuery for Power BI datasets and Analysis Services. After restarting Power BI Desktop, I use Get Data and choose Power BI datasets. After connecting to the workspace and selecting the ContosoDW Internet Sales published dataset, the local model is now connected using a live connection. After connecting to the remote model, all the tables are listed in the Fields list. At this point, I'm using a capability that has been available for a long time: one Power BI report file connected to a separate, published dataset. Previously I would not be able to add additional dataset connections or import from other sources into a live connected data model. The status bar shows a new message with a link to enable changes to the model. On the Model ribbon, a new button is also displayed. Using either of these options will modify the local model metadata to allow multiple live dataset connections. When I click the link Make changes to this model, a dialog box allows me to confirm that I want to switch the live connection to a DirectQuery connection and prepare the local model to allow additional DirectQuery connections. What happens if I start with an existing local model? The first important lesson I learned was this if I were to create a new PBIX report file containing existing tables and then connect to a published dataset, this does not enable the ability to make changes to the local model. With an imported table in the data model before using Get Data to select and connect to a published dataset, the status bar shows this message: I found this to be unintuitive: The text "(click to change)" is not a clickable link. At this point there is no way to update the local model with the necessary changes. I had to start with an empty PBIX to add the first live connection. After that, I could use Get Data to import additional tables from others sources, but not the other way around. Adding More Dataset Connections Next, I use Get Data to connect to the ContosoDW Store Sales published dataset. After the local model had been converted, there was nothing particularly unique about the experience. I could just connect to an additional dataset and all of the tables will simply show up in the data model diagram view. The tables from each model are displayed with a different colored heading, in this case they are blue and red. I had several duplicate table names after connecting the second data model. The second set of tables were renamed and postfixed with the number 2. I renamed the tables that I wanted to keep visible. The redundant tables that I don't need are displayed on the right side of this diagram layout. Tables in a remote model cannot be deleted from the local model. I used the Properties pane to set the unused tables to be hidden. If I were connecting to a large remote model that had a lot of tables just for the sake of bringing one table into the local model, it would be cumbersome to hide all of the the unneeded tables – so I hope there will be an easier way to manage this in the future. I opened Tabular Editor from the External Tools ribbon and was able to use it just as if I were designing a normal import model data model, to rename tables and set properties with multiple objects selected. Adding relationships between tables from each remote model was pretty much the same process as it would have been in a local model. I can use the Manage Relationships dialog and use the drag-and-drop method in the diagram view. The latter method opens the Manage Relationships dialog rather then simply creating the relationship. Finally, I added a new Combined Sales Qty measure to one of the remote tables in my local model that references an existing measure from each of the remote models. For demonstration, the table visual shown below includes the Order Date field which is related to each of the two fact tables (Online Sales and Store Sales), the Online Sales Qty measure from the Internet Sales remote model, the Store Sales Qty measure from the Store Sales remote model, and the new Combined Sales Qty measure which I added to the local model: After publishing the new composite data model and report, the Lineage View shows that the Combined Sales dataset has connections and dependencies on the two remote datasets that show up as Analysis Services DirectQuery connections, apparently using the XMLA endpoint in the Power BI service. Once I worked through the design process a couple of times, the data model behavior is seems to be exactly the same as always, and designing reports is no different at all. I'm just scratching the surface and I'm anxious to dig deeper and compare notes with others in the community. I have no doubt that behaviors, limits and issues will arise with this brand new and very compelling capability but I'm very encouraged so far. If you're like me right now, you need a distraction. I got started on a new blog and tweet about project management best practices and after yesterday's events, just had to put it on a shelf. So, a couple quick things… Power BI Top Features Melissa Coates, long-time BI community rock star, published this amazing infographic depicting all of the features and services in the Power BI ecosystem. Next time you hear someone incorrectly describe Power BI as "just a visualization tool", show them this: You can download a hi-res version and get more background here: Diagrams — Coates Data Strategies BI-Survey Searchable Maps I recent heard from Donald Farmer, one of the original architects of the Microsoft Business Intelligence platform. Donald went on the work for QlikTech and now runs an advisory practice called Treehive Strategy Services. Donald has been working with the BI-Survey, one of the leading analyst firms in Europe, to enable an interactive map of the Azure analytics ecosystem. Similar to Garner and Forrester research, BI Survey compare industry platforms and products but do so using survey data. I've been participating in the BI-Survey for about 12 years and find it to be a comprehensive and insightful resource to compare BI tools, among other products. They just released these interactive guide maps for Power BI and Azure Services, Partners and Success Stories. These are just a couple of examples: Each map has a bunch of filtering and search filters that allow you to visualize the heatmap based on pricing, features and a variety of categories. These are really useful tools. Part of the the series: Doing Power BI the Right Way (link) Data Modeling 101: part 1 (more advanced techniques in part 2) A data model is the foundation of analytic reporting. It provides structure and order over information that might otherwise be chaotic and untrustworthy. Loading data into a properly designed model ensures that it conforms to some essential rules that provide better performance, reliability and accuracy. Different Schemas for Different Needs One of the most important lessons I have learned about data modeling over the past 20+ years is that there isn't one model that fits all business needs. However, there are simple patterns we can follow to model data according to different business needs. Each pattern or schema has a different set of rules. At a very high level, we can think of these schemas in three different categories. When reporting requirements are simple and unsophisticated, sometimes the easiest way to model a set of data is to transform it into a single, flat table. There are several trade-offs and limitations when using this approach, though. This is typically the goal when bringing information into a spreadsheet so you can simply add-up a column of values, perhaps filtered by other columns. It might seem natural to use the same approach when making the transition from Excel to Power BI. Here is an example of retail order data organized into a flat table: Look over this sheet before moving to the next figure. If this data were stored in an Excel sheet, how could we use it for reporting? You could filter by the Order Date column and add up the Quantity, Unit Cost and Unit Price. You could also apply filters to the Company Name, Category, Subcategory or Product Name columns. For simple reporting, this could meet our needs. The master/detail pattern is typically used with transactional data. Master records, like orders or invoice headers, are related to detail records such as the order details or invoice line items. The master/detail pattern can be repeated with multiple tables to create a normalized database. Use this pattern sparingly to support drill-to-detail reports. Including unnecessary detail in an analytic data model uses a lot of memory and storage, and can impact report performance. When relational database professionals are introduced to Power BI, they will often see DirectQuery as a simple means to leverage their familiar and existing relational database or data warehouse, only to be disappointed by poor performance and other limitations. However, master/detail schemas are where DirectQuery can provide a best of both worlds solution so long as the analytic needs over detail data is light. If we need to provide the ability to drill-down to detail records from an interactive report without needing to perform calculations over the entire detail record set and avoid sucking billions of records into memory, DirectQuery might be the ticket. Dimensional Modeling 101 The rules of dimensional modeling are a lot like the laws of physics in that the rules of dimensional modeling were really discovered rather than invented. By applying dimensional design patterns, data will naturally behave and reports will perform well. Dimensional rules really help describe the way data behaves when stored in the most optimal form for reporting and analysis. The Kimball Group are credited as the pioneers of dimensional modeling and although the books and papers that Ralph Kimball and his colleagues authored are 20-30 years old, they are as relevant today as ever. I'll cover some of the basics but you can find a comprehensive reference guide to dimensional modeling patterns and techniques on the Kimball Group site here: Dimensional Modeling Techniques – Kimball Group. There have a been a few recent publications employing modern examples in books such as Star Schema – The Complete Reference and Agile Data Warehouse Design. Both are excellent books but unless you are undertaking a huge data warehouse project with complicated data problems, understanding the essential concepts is sufficient. By organizing information into fact and dimension tables, we store as little data as possible while maintaining consistency and data integrity. In data design, we often refer to "entities" and "attributes". An entity is something we keep track of (like a Customer or Product). Entities have attributes which describe specific properties. In the analytic data world, entities are usually materialized as dimension tables and each attribute is a column or field. A fact table contains the numeric columns used to summarize and aggregate measure values, along with the key columns that are related to the dimension tables. Dimensions contain the attributes used to group and filter the business facts. Fact records share the same level of grain across all dimensions. For example, if domestic sales orders and international sales orders each have the same level of detail for dimensions like Customers, Products and Order Date, these records could be stored in the same fact table. But, if Sales Targets are applied at the month level rather than the date, they must be stored in a separate fact table. The essence of a dimensional model is the star schema, simplified here to show one fact table related to a small number of dimensions. A real application of the star schema design is shown in the following image. The Online Sales fact table contains key columns that are used to relate this table to each of the dimensions. The fact table also contains numeric type columns used to define measures which aggregate and total numeric values such as Net Price, Quantity, Unit Cost, Unit Discount and Unit Price. You'll notice that the relationship from each of the dimension tables to the fact table is one-to-many and filters records in one direction, depicted by the arrow on the relationship line. For example, the relationship from the Customer table to Online Sales is based on the CustomerKey column in both tables. The relationship filter direction indicates that if the Customer table were filtered on a report, Online Sales would be filtered accordingly. However, if the Online Sales table were filtered in a report, this would not cause the Customer table to be filtered. According to the Kimball Method, there are four stages to define fact and dimension tables: Select the business processDeclare the grainIdentify the dimensionsIdentify the facts Seems pretty simple, right? It can be at a macro level until you need to get into the weeds. The trick is to maintain a high-level business perspective as you work through the the details to solve data problems. I find this to be a useful whiteboard exercise when starting a new project. Using only business tools and business language, diagram data entities at a high level using this approach. Next, break the process down using these steps: Load detailed atomic data into dimensional structuresStructure dimensional models around business processesEnsure that every fact table has an associated Date dimension tableEnsure that all facts in a single fact table are at the same grain or level of detailResolve many-to-many relationships in fact tablesResolve many-to-one relationships in dimension tablesStore report labels and filter domain values in dimension tablesMake certain that dimension tables use a surrogate keyCreate conformed dimensions to integrate data across the enterpriseContinuously balance requirements and realities to deliver a DW/BI solution that's accepted bybusiness users and that supports their decision-making Something to keep in mind is that these rules and steps apply equally to formal, IT-based model design and to self-service BI projects. However, the tools have been modernized over the past decade or so; along with the degree of strict adherence to the rules. Simply put, in the new age of self-service BI, everything is negotiable. But something else to keep in mind is that bending or breaking the rules nearly always comes at a cost. For example, step 8 prescribes creating surrogate keys to identify dimension records. You can use the original business keys throughout the model but run the risk of encountering key collisions or other data quality issues. In some cases, the lines that separate facts and dimensions can become blurred, which may need to be considered in your model design. For example, I recently developed a BI solution where we counted employee hiring events and a row in the Hires fact table was simply a record of the event when an employee was hired along with information like the hiring date and position. There were no numeric columns to sum-up so we simply needed to count these records. This is often referred to as a "factless fact". Another exception to the typical star pattern are cases where a fact table contains a value we could use for grouping or filtering, and it might be pointless to extract it into a separate dimension table. This is known as a "degenerate dimension". These are examples of cases where you could arguably apply the dimensional design rules strictly out of principle or err on the side of simplicity. The right answer for each case should be driven by the business reporting requirements while considering the the reliability and matainability of the overall solution. Let's apply the process to the flat table of sales orders. Here it is again with some of the dimensions and facts identified: In this example, there are three separate date columns that record the date an order was placed, the date it was due and the date that the order was delivered. Customer information includes the Company Name and a unique CustomerKey which could be used to lookup detailed customer information. There are three numeric columns that can be summarized or aggregated. These columns could also be used to perform more complex calculations like ratios, running totals and time-series comparisons. Dimensional Concepts Now, let's review some common practical dimensional modeling applications. Role Playing Dimensions Dimensional entities can play more than one role in reporting. In the flat table, the three date columns have completely different purposes but all store the same type of value; a date. But dates can be used to group and aggregate data in date part groups like month, quarter, year or fiscal periods. They can be used to perform time-series calculations like month-to-date or same period in the previous year. To do this, you absolutely must have a date dimension* table containing a contiguous set of date records over a period of time. For our purposes, we might need all dates for the past five years through the end of the current year. Date dimensions are the most common example of role-playing dimension but let's look at another scenario before looking at at more date examples. *Power BI Desktop includes features that will automatically generate a date dimension table, and there are DAX functions you can use to generate date tables in code. Although useful in certain cases, I prefer to manage date tables the old fashion way so I have one constant method for importing and managing all tables. By comparison to other data challenges, building date dimension tables is a relatively simple task. The following image shows a fact table named Flights, with two role-playing Airport dimensions. An airport is, well it's just an airport… it has a three-letter code and some information about the geographical location. But with respect to any given flight, an airport can serve two functions: to depart from or to arrive to. In this example, it is necessary to materialize the Airport table twice: the Departure Airport and the Arrival Airport. With two role-playing Airport dimensions, a report user could ask for all the flights from Japan to Australia on a given date. Back to the retail sales example, let's say that our Sales fact table has two date keys; OrderDate and ShipDate. A single Date dimension can only have one active relationship. The following image shows the result of the autodetected relationships between Sales and Date, where one of the relationships is inactive and effectively useless for most practical purposes. This image shows three separate instances of the Date dimension table imported using referenced queries in Power Query. The three tables: Order Date, Due Date and Delivery Date use up a tiny bit more memory than a single instance of the table but give report users more flexibility to answer business questions. A Table for Every Date Column? An attendee in a data modeling session recently asked this question. She said: "I have 14 date columns in my fact table used to track a variety of different events. Do I need to create 14 different Date dimensions?" My answer was "certainly not." This is an extreme example but a very good question. You only need separate date dimension tables when you need to have the flexibility to use time-series functions in DAX or to perform comparisons using date part fields (like Year, Quarter or Month). Operations like calculating the difference between two date type columns or counting rows with future dates can be performed using fields in the fact table. I'll show you an example in part 2 of this blog series. This is one of those "it depends" situations where you must make a judgement call depending on your business reporting requirements, balancing simplicity and maintainability with complexity and flexibility. Many-to-Many relationships and Bi-directional Filters File this under "don't do this unless you have to." Many data modeling decisions are a trade-off between performance and functionality; and using iterative design, you will often find a better way to solve problems as you go. There are a few different ways to design for many-to-many relationships. The conventional approach is to use a bridging table that contains all of the combinations of keys that relate two tables together. In the following example, the Customer and Product dimension tables normally would have a single-direction filter from the "one-side" to the "many-side" of the relationship. If reporting requirements call for filtering products based on selected customers who purchased them, we can use the Sales fact table as a bridge table and change the relationship between Product and Sales to use a bi-directional filter. Depending on the cardinality of the relationship, using a bi-directional filter could introduce a performance penalty. If we only have 100 products and less than a million Sales records, this is probably not a big deal. If we have 10,000 products and 100 million Sales records, this relationship will likely slow things down considerably. Here is another example: Given the requirement to show accounts and transactions for a selected customer, the following model doesn't work with out-of-the-box relationships. To understand why, follow the flow of filtered records through the relationships. From Customer to AccountCustomer, the arrow on the relationship line indicates that the filter flows in the right direction. Once the AccountCustomer table has been filtered, the Account table will not be filtered as we try to swim upstream because that relationships doesn't naturally flow from the many-side to the one-side. Changing the Account-AccountCustomer relationship filter to bi-directional fixes the problem and meets the reporting requirement. As I mentioned, the bi-directional filter introduces a potential performance penalty. However, if the cardinality is low and the memory footprint of the data model is low and lean (using efficient key values and removing unnecessary columns), this design may perform just fine and you are good-to-go. I'll demonstrate more creative techniques in part two of this two-part topic, as we explore more advanced data modeling design techniques. Continued in Part 2 My wife bought me a nice telescope for Christmas. I've wanted one since I was a kid. We opened gifts early because we're going to take the RV out in the dessert to watch the great convergence of Jupiter and Saturn next week. But, what am I doing now, after opening my gift? …updating Power BI Desktop to check-out the new Composite Model feature! This new feature of Power BI has been in the works for several years and we have been anxiously waiting for this announcement. It is truly the Utopian dream; the Holy Grail! It is hard to articulate the impact that this capability can have for businesses. The new composite model gen 2 feature is a monumental breakthrough that will significantly raise the bar for Power BI model development and could change the way we think about designing enterprise BI solutions – but customers will need guidance to use it correctly. If you haven't been sitting on the edge of your seat waiting for this announcement, read the announcement and understand how significant this is. Power BI December 2020 Feature Summary | Microsoft Power BI Blog | Microsoft Power BI There are been many attempts by Microsoft and other vendors to create a data modelling architecture that provides for fast access to cached data, direct access to live data and scaled-out connections to established data models. Remember ROLAP and HOLAP storage in multidimensional cubes? These were great concepts with significant trade-off limitations. No other vendor has anything like this. Way back in the day, Microsoft jumped on the Ralph Kimball bandwagon to promote the idea that a company should have a "one version of the truth" exposed through their data warehouse and cubes or semantic data models. They met customer demand and gave us a BI tool that, in order to bring data together from multiple sources, makes it easy to create a lot of data silos. Arguably, there are design patterns to minimize data duplication but to use governed datasets, self-service report designers are limited to connecting to large, central models that might only be authored and managed by IT. This new feature can restore balance to the force and bring us back to "one version of the truth" again. To put this into perspective, here's a quote from Alberto Ferrari's recent article: "It is not easy to find a name that accurately describes the massive step forward that is this feature in the evolution of Business Intelligence solutions. It would fully deserve a hyperbolic name, but the marketing departments already overused so many terms over the years: "universe" was already used in the past century; you can imagine what happened after that. We like "composite models" because it is a name that concisely describes the purpose of this new feature." Alberto has a great video emphasizing the magnitude of this release: New composite models in Power BI: A milestone in Business Intelligence – SQLBI Those of us who live a breathe BI will continue to have thoughts, opinions and eventually will have war stories and prescriptive guidance to offer about using this new capability most effectively. Upon initial review, it seems to work exactly as expected. In future posts, I will demonstrate how to design and implement a next-gen composite model and share my experience and thoughts as we move down this new road. One of the most common questions I hear from customers and students is "what are your recommended resources for learning and staying current with Power BI?". I find some people get stuck learning Power BI because they are simply overwhelmed with information and choices. Google search will often raise more questions than answers because there is a lot of noise on the Internet. Expert advice can be found only after dismissing the myriad of posts asking questions, sharing opinions, and sifting through outdated and confusing information. Most information found in discussion forums is outdated because the product changes so often. So, what are the most reliable sources for Power BI expert advice and best practices? I follow several industry experts and I own many great books. Chris Wagner (KratosBi.com | @KratosBi), Microsoft MVP, has compiled the most comprehensive go-to list of Microsoft data experts called the Data Gods (you can see the complete list here). When I was added to the collective of featured "Data Gods" last week, I was hesitant to brag about. I mean, there's no modest way to post this picture. The title of "Data God" is a lot to live up to – but it is truly an honor and I want to thank Chris and others for this recognition. More importantly, this is a concise list of reliable industry experts – which I have paired-down to those who blog and post regularly about their experiences and advice. I know and follow nearly everyone on this list but there are a few folks with whom I have not yet had the pleasure to meet. There are also others featured in the Data Gods collection who make significant contributions in other ways, aside from frequent blogs and articles. I've also added a few additional notable community contributors. Free Best Practice Resources Following is my selection of members of the Data Gods counsel with links to their free resources. Most of these are blog sites or web sites that include blogs and articles available for free without requiring a sign-in or subscription: Alberto Ferrari & Marco Russo – SQLBI.com | @marcorus | @FerrariAlberto For most of us who work day-to-day with Power BI and other Microsoft BI tools, SQLBI.com is the first stop for expert advice. Alberto and Marco have set the gold standard for Microsoft BI best practices and design patterns. The breadth and depth of their expertise is expansive and they are known for going deep into the nuts and bolts that make Power BI work. These guys are are literally an extension of the Microsoft product development teams by guiding Microsoft leadership and evangelizing the platform. Their books on DAX and Tabular model design are considered by the most serious Power BI Developers to be the most comprehensive reference books available. Patrick Leblanc & Adam Saxton – GuyInACube.com | @GuyInACube | @awsaxton | @patrickdba You either know these guys or you've never searched the web for Power BI help. Adam started the Guy In A Cube YouTube channel several years ago as a side project while doing BI technical support work for Microsoft. Patrick came from a BI and SQL Server consulting background before joining Microsoft and teaming with Adam. Today they share design techniques and feature highlights in weekly, high-quality short videos and a Saturday morning live streamed Q&A session. Adam and Patrick are members of the Customer Advisory Team (CAT) focused on supporting Power BI enterprise customers. Their weekly Power BI community Roundup video features the most current Power BI blog posts and release notes. Chris Webb – blog.crossjoin.co.uk | @cwebb_bi Chris is one of the most prolific, enduring and consistent bloggers in the Microsoft BI industry. His posts and articles go deeper then most, prepared with thought, effort and insight. Before joining the Microsoft Customer Advisory Team (CAT), he ran a successful training and consulting firm, specializing in the Microsoft BI platform. Over the years he was, and continues to be, the go-to expert on SQL Server Analysis Services, MDX, Power Query and DAX; and continues to share his knowledge and advice through his blog. Matt Allington – Excelerator Blog | @ExceleratorBI Matt's main focus is business-centered self-service BI and financial analysis. He is a tried-and-true expert in the Excel and Power BI Desktop space. His training, presentation and writing style is clear, authoritive and to-the-point. He has the best collection of intro and intermediate level books on Power BI and DAX; each is short, concise and easy-to-understand. His books, Supercharge Power BI and Learn to Write DAX are excellent way to get started. Seth Bauer, Mike Carlo & Steve Campbell – PowerBI.Tips | @PowerBITips | @Seth_C_Bauer | @Mike_R_Carlo | @PowerBISteve PowerBI.tips is an expansive collection of free resources and utilities for Power BI with focus on visual themes, layout and color selection. This dynamic trio covers the important bases: Seth on visual design, Mike on enterprise solutions and Steve focusing on application development. They have an array of external tools for things like connection management and project documentation. Melissa Coates – CoatesDataStrategies.com | @SqlChick Melissa is one of the deep thinkers in the industry, always asking the right questions and challenging the status quo. She's also one of the most pleasant people I've ever met. Her blog posts are thorough and insightful. Melissa has co-authored multiple whitepapers for Microsoft on data governance, enterprise deployment and migration. Leila Etaati & Reza Rad – Radacad.com | @leila_etaati | @Rad_Reza Leila and Reza are known as one of the power couples in the industry. With nearly limitless energy, they have traveled from New Zealand over the years to present at more conference events than I can count. Leila is one of the BI industry's thought leaders in the machine learning and AI space, with education and credentials behind her experience. Reza's expertise with Power Query and Power BI are impressive. They freely share their knowledge through their blogs, articles, training classes and have published numerous books. Kasper de Jonge – KasperOnBI.com | @Kjonge Kasper is a Principal Program Manager on the Power BI product team and based in The Netherlands. A regular conference session headliner, Kasper is very tuned into customers' real-world enterprise BI reporting problems. He has been blogging for years and has recently launched a YouTube channel. Alice Drummond – DiscoverEI.com | @AliceDiscoverEI With an emphasis on environmental data analysis, one of the things I appreciate about this Alice's active blog series is that with each post, she shares a comprehensive collection of blog sites and training resources to help readers find supportive information and expand their knowledge. David Eldersveld – DataVeld.com | @DataVeld If David can't figure it out, it probably can't be done. Over the years, he has pushed the limits, discovered and developed some very unique and interesting techniques to integrate things like graphics and geospatial data with Power BI. David is very active in the community and shares different ways to take Power BI beyond the out-of-the box features to do things that many of us had no idea was even possible. Brian Grant – CSGPro.com | @BrianGrantBI Brian has a unique intellect and a passion for problem-solving and sharing knowledge, particularly related to Power Query and DAX. I've had the pleasure of collaborating with Brian on multiple occasions in conference presentations and training events. He and his colleagues have developed an interesting method to understand the mechanics of the DAX language that he shares in a video series called the Elements of DAX. Reid Havens – HavensConsulting.net | @HavensBI In a few short years, Reid has risen to all-star status by tirelessly designing and sharing data visualization techniques. He is a master of spit-and-polish presentation and report design. Meagan Longoria – DataSavvy.me | @MMarie Meagan has broad expertise but her focus is designing reports for accessibility and accurate data presentation. Her posts and conference presentations are very insightful. She challenges the status quo and shares specific techniques to provide accessible report designs. Matthew Roche – BIPolar.com | @SQLAllFather Matthew is a member of the Microsoft Power BI Customer Advisory Team (CAT). He has considerable experience with several Microsoft development initiatives that have culminated in leading products and Azure services that are used to create enterprise data solutions. He is an industry thought leader who is very active in the online community. He shares his insights, leadership and expertise through his blog and industry conference events, most notably related to modern data governance and Power BI dataflows. Gil Raviv | DataChant.com | @GilRa Author of Collect, Combine & Transform Data in Excel using Power Query and Power BI, and excellent reference book. Gil maintains an active blog. SqlServerBi.com | @Paul_Turley I add my own blog to this resources list where I share my thoughts about best practice design and development approaches in this series: Doing Power BI The Right Way. Others Gilbert "Q" Quevauvilliers – FourMoo.com | @GilbertQue From has vast experience consulting with IT and business consulting customers, Gilbert shares relevant examples and lessons learned. He is an active blogger and community supporter. Imke Feldmann – TheBiAccountant.com | @TheBIccountant Deep, business-focused, creative problem-solving techniques using Power Query/M in Power BI and Excel. Ruth Pozuelo Martinez – Curbal.com | @CurbalEN Ruth puts tremendous energy and creativity into her video presentations which she posts frequently. She covers everything from intro to moderately advanced level Power BI design with complete and thorough examples, all presented in a stunning greenscreen format. Parker Stevens – BiElite.com | @PowerBIElite Parker's blog site is an extension of his "BI Elite" YouTube channel, with numerous short tutorials and excellent "how-to" examples. James Serra | JamesSerra.com Thorough and consistent, James shares announcements and guidance about the Microsoft data platform and data warehouse architecture. Books Beginner Intermediate/Advanced Part of the the series: Doing Power BI the Right Way (link) Back in the day before Power BI came along, when we created BI solutions, reports and data models were separate. If you created a cube or Tabular model with Analysis Services, it was developed with Visual Studio and deployed to a server. Reports could be authored and deployed to the report server, separately. Now, with Power BI Desktop, you author your data model and report in the same development space and then deploy the whole kit and kaboodle to the service at once. The Thick and Thin of Reports Power BI reports generally come in two varieties. If you open up Power BI Desktop, choose "Get Data", build a model and a report; all of that stuff ends up in one self-contained PBIX file. Conversely, a report using a live connection to a published dataset is often called a "thin report" because it contains no data or data model. Thin reports offer several advantages: not only separation of development effort but also centralized control and the ability to share published data models with multiple report authors. Even when a single PIBIX file contains everything, Power BI actually separates the report from the data model (called a dataset in the service) and gives them both the same name. This is very convenient for self-service projects because it is quick and easy to make changes to the data model, queries and measures if you need to make a report enhancement. This is all well and good for small, one developer projects but what about those larger scale solutions where the data model and reports are developed by different folks, or when multiple reports are connected to the same data model? At what point does it make sense to separate the data model and reports into separate files? I believe that managing reports and the dataset separately is a best practice in enterprise solutions. However, like a lot of "proper" practices in the IT data world, it is far less convenient to "do it right". In each project, we must exercise an appropriate degree of discipline. I've worked on projects that were so laise fair and lacking management structure that they produced a tangled web of garbage. I've also worked on projects managed with strict adherence to rules and restrictive application development methodologies; where report and BI developers had no freedom to be creative; and thus producing bland and useless reports. It is hard to prescribe a set of rules that always works for every project with the right dose of balance. A certain degree of freedom is necessary to learn-as-we-go and produce masterful analytic reports. At the same time, the process requires a well-defined plan, time-boxing and prioritizing of features. When I started writing this post, I was working on a consulting project that started out with the dataset and report in two separate files. Under that plan, one developer could work on the data model and publish it to the DEV/QA workspace and then the other developer could design reports as new modeling features become available. In theory, this is typically the right approach. The requirements for this project expanded and we found ourselves in a rapid iteration cycle, needing to make model, measures and report changes. As a result, I brought the report pages back into the dataset PBIX file. There is no elegant way to do this but it wasn't difficult to create new pages and then copy and paste the visuals from the separate report file to the combined dataset/report file. Bookmarks, drill-through and tooltip pages had to be rewired but it wasn't a difficult task. Kasper De Jonge walks through the process in this blog post from May, 2020. In cases where you have crisp requirements and a clear division of labor; you can start with two files to delineate the data model and the report. In cases where you are actively prototyping and working alone, developing a single PBIX file containing all the parts will allow you to move quickly. When it becomes more important to lock things down and manage releases than to keep iterating, move report pages to a separate PBIX file that is connected to the published dataset in the service. Hot Swap Live Connections I've stared using a new external tool that promises to make this process much easier. The Hot Swap tool is a utility developed by Steve Campbell and available from PowerBI.tips as part of the Business Ops externals tools package. This simple tool allows you to start a project with separate data model and report files and then to swap the live connection between copies of the model: either a dataset published to the Power BI Service or to a PBIX file on the local desktop using a local TCP port. Hot Swap is a community-developed external tool that is not officially supported by Microsoft but my experience in testing the tool thus far has been very good. I used an earlier solution from Steve in a project about a year ago that worked-out quite well. It was a set of PowerShell scripts that rewired connects in much the same way. Having this capability in a simple pop-up dialog is an even more convenient option. You can check out this demo of the Hot Swap tool in the PowerBI.tips YouTube channel. One of the tings I like about starting with two files is that we don't run the risk of design issues left-over after moving report pages and visuals to another file. The new approach is simple: begin your project with separate data model and report files. You can publish an early version of the data model file to a development workspace in the service and connect to establish a live-connected report. Then, run the Hot Swap tool and switch the connection to a local copy of the your data model to continue iterative development. Changes to the local data model PBIX file/Power BI Desktop instance are immediately available in the report PBIX. Publish changes whenever you need to and then you can use Hot Swap to remove the local connection. You don't actually use the tool to perform the reconnect because Power BI Desktop allows you to use the normal Get Data options after a live connection has been removed from the report file. Regardless, it is a simple process. Recommendations For small, informal projects developed by one person (or one developer at a time), keeping the data model and report in one file is convenient and easier to make changes as you go. If you later decide to share the dataset for others to create additional reports, or to promote or certify the dataset, you should separate the dataset file and create a "thin" report using a live connection to the published datset. For moderate-scale and formalized Power BI projects, begin with two separate PBIX files or make it a point to separate the data model from the report file before you deploy to the Power BI service. Consider using the Hot Swap tool to help manage local mode development. Part of the the series: Doing Power BI the Right Way (link) Part 1 is here Validation 301: Creating a Validation Report Solution, Part 2 We often have users of a business intelligence solution tell us that they have found a discrepancy between the numbers in a Power BI report and a report produced by their line-of-business (LOB) system, which they believe to be the correct information. Using the LOB reports as a data source for Power BI is usually not ideal because at best, we would only reproduce the same results in a different report. We typically connect to raw data sources and transform that detail data, along with other data sources with historical information to analyze trends, comparisons and ratios to produce more insightful reports. However, if the LOB reports are really the north star for data validation, these can provide an effective means to certify that a BI semantic model and analytic reports are correct and reliable. This is a pattern that I have applied for consulting customers. My current client refers to the main Power BI report and dataset as their "dashboard" solution. The data model is large and complex with sophisticated reports used monthly by hundreds of users. The dataset is setup with a scheduled refresh. The "dashboard" solution has been thoroughly tested and we are careful to only apply updates and enhancements infrequently under a rigorous application lifecycle management process. At the end of each month, a user exports their LOB report to an Excel file and drops it into a SharePoint library. I have one consulting client who use an automation tool to essentially open the LOB application, punch in the right keystrokes and export the report to a file at the end of every month. The format of these exported reports have proven to be fragile and we don't want to risk destabilizing the main Power BI "dashboard" solution. So, rather than importing the validation files directly into the main dataset, I have created a separate Power BI project with a dataset & report that has two sources. The first data source for the validation report connects to the main "dashboard" Power BI dataset using the Analysis Services connector and the XMLA endpoint. A DAX query is issued to return values to compare with the LOB report totals. The second data source for the validation report imports the validation files that were exported from the LOB report. If that import process were to fail for any reason, it doesn't affect the main dashboard solution. This diagram depicts the validation solution at a high level. The main Power BI "dashboard" solution get its data from the corporate EDW. That data is processed through rigorous ETL pipelines and contains data extracted from multiple systems used by the corporation. A lightweight DAX query imports summarized results from the main dataset into the validation dataset. Before we go any further, I just want to make the point that in most normal Power BI data models, I would never import data from another published Power BI or SSAS data model. That's just a bad idea and can make it very difficult to trace data lineage. Do not use DAX queries as a data source for your regular report models. This validation report is a rare and appropriate exception to the rule. The second data source for the validation solution is a set of files exported from reports produced by the LOB system. These reports only contain summary information but can be compared to totals from our Power BI solution. Values from the two sources are compared and the validation report shows whether they match or fail validation. Just a note about the decision to create a separate dataset. There were two reasons for this: 1. We could not control the format of the validation files. If the file format were to change, it would cause an error and stall our scheduled refresh cycle for a large mission-critical report used by several hundred high-level employees (whose job is it to tell the CEO why his dashboard doesn't work?)2. Today we do not have the ability to use a live connection to a Power BI dataset in a composite model along with other sources, but we will in the near future. This capability doesn't address problem #1 but will certainly change the way I think about architecting large, multi-source models in the future. Writing the DAX query to read the summary values from the main Power BI "dashboard" dataset was a great learning experience. Although I am now pretty good at writing DAX queries by hand after a couple of these projects, the easier way to get started is to allow Power BI Desktop to generate the query for you. Here are the steps using the ContosoDW dataset as an example: Add a table visual to the report canvas. Add the same field and measure used in the LOB report to be used for comparison.Turn off totals for the table visual.Add and set slicers or filters to equate the LOB report results.On the View menu in Power BI Desktop enable the Performance Analyzer.In the Performance Analyzer pane display to the right side of the report canvas, click the Refresh visuals link.Click the plus sign next to the table visuals to expand the performance counters for that visual.Click the Copy query link. The DAX query is now in the Windows clipboard and can be used to run the same query that the designer produces to populate the table visual. To test and modify the query, do the following. Make sure you have the latest version of DAX Studio installed. If not, install it from daxstudio.org. Save your work, close and reopen Power BI Desktop after installing DAX Studio.From the External Tools menu in Power BI Desktop, click DAX Studio. This opens DAX Studio with a query session connected to the local Power BI dataset.Paste the clipboard contents to the query window and run the query. Here's the query that was generated by Power BI: DEFINE VAR __DS0FilterTable = TREATAS({2009}, 'Order Date'[Year]) VAR __DS0FilterTable2 = TREATAS({"October"}, 'Order Date'[Month Name]) VAR __DS0Core = SUMMARIZECOLUMNS('Product'[Category Name], __DS0FilterTable, __DS0FilterTable2, "Online_Sales_Qty", '_Measures'[Online Sales Qty]) VAR __DS0PrimaryWindowed = TOPN(501, __DS0Core, 'Product'[Category Name], 1)EVALUATE __DS0PrimaryWindowedORDER BY 'Product'[Category Name] This code can be simplified for our needs. The essential code is the expression using the SUMMARIZECOLUMNS function. You can disregard everything after that. Make the following changes to simplify the DAX query code: Delete everything after the closing parenthesis for the SUMMARIZECOLUMNS function.Remove the variable line just before the SUMMARIZECOLUMNS function and replace it with "EVALUATE" The resulting query code should look like this: DEFINE VAR __DS0FilterTable = TREATAS({2009}, 'Order

Date'[Year]) VAR __DS0FilterTable2 = TREATAS({"October"}, 'Order Date'[Month Name])EVALUATE SUMMARIZECOLUMNS('Product'[Category Name], __DS0FilterTable, __DS0FilterTable2, "Online_Sales_Qty", '_Measures'[Online Sales Qty]) This is the query I would use to populate a new table in my validation dataset. Just use Get Data to open the Power Query editor, create a new SQL Server Analysis Services connection, paste the workspace path for a premium enabled workspace and connect to the published dataset for my main Power BI solution. Then finally, paste this query code into the query window. Next step is to import the validation file(s) that were exported from the LOB report. In our validation solution, I used the SharePoint connector and imported all files with a predefined file name prefix. After the validation files are imported, make sure that the column data types are compatible, load the table and wire-up a relationship between the two tables. In our last validation solution, I created a new measure that compares a measure from the summary table imported from the main "dashboard" solution dataset with a comparable measure from the validation file source table. Measures calculated the difference and the % difference between the two. I then used a KPI indicator icon in the table visual using conditional formatting to show if the differences were within an acceptable range. This screen capture from the validation page has the real data blurred out but you can see the grouping and comparisons resulting in either green or red indicators. Each month, I receive an email from our user who is responsible for the report validation. He tells me that at a quick glance he is able to verify that the totals from their line-of-business application and Power BI solution match, and that the executive leaders and financial analysts who use these reports can be certain that they are accurate and reliable. When an item doesn't match, they can pinpoint the account in question and usually are able to track it down to a data refresh timing issue or a mistake made in their source system that hasn't yet been reconciled. We are now setting up alerts using KPI tiles on a dashboard that will message key users when validation either succeeds or fails in key areas. As a result, the confidence in this solution has increased significantly and business leaders trust the information they get from these reports. Part of the the series: Doing Power BI the Right Way (link) Moving important business data into a data model for analytic reporting can often be a two-edge sword. Data retrieval is fast and can support all kinds of analytic trending and comparisons. But, data in the model may be one or two layers away from the original source data, making it more challenging to compare with familiar user reports. Often the first validation effort after transforming and loading data into the model and then visualizing the initial results is having a business user say "yeah, that looks about right." Then, sometime later after more development and extensive user testing, the feedback might be "hmm, that seems a bit off." …not exactly scientific. I have been doing a lot of data validation work lately – both formally and informally. Informally: Validating calculation results from a Power BI data model is just a normal part of the development process. Formally: After developing and delivering an entire business dashboard solution, a formal validation process is used to validate the ongoing results after future data refresh cycles and to certify reports so that business leaders know they can trust them. Particularly for database developers, making the leap to using reports built on a tabular data model instead of the source database is a mental transition. The developer has to stop thinking in SQL and start thinking in model mechanics. This can pose a challenge when needing to validate numbers in a report, which should be done using the data model and not only by running by a SQL query. Validation 101: Create & Use COUNT Measures When designing a new data model, this is typically the first thing I do… For every fact table and for each large dimension table, I create a measure that returns the record count for that table. Users normally think about their data in business terms (like sums, ratios and averages) and not about how many records there are in a table. Record counts are a convenient sanity check for record completeness and distribution across groups; and may also be a good indicator for model size and performance. Let's say I have a data model containing the tables: Sales, Orders, Customers, Products and Date. At minimum, I will create the following measures with code similar to this: Sales Row Count = COUNTROWS('Sales') Orders Row Count = COUNTROWS('Orders') Customer Count = COUNTROWS('Customers') These measures allow you to quickly troubleshoot problems that might otherwise go unnoticed – with model relationships, key matching, orphaned records and missing records. Just drag-and-drop the measure and a field to group the values. For example, if I drag the Sales Row Count measure and Year from the Date table onto the report canvas, I can immediately see sales records across years. If I do the same thing with the customer region field, I might be able to spot a gap in the region distribution of records over time. If every column in a column chart (created by default when dragging a numeric field) shows 100%, then I know that I either forget to create a relationship, have a mismatched data type or forgot the "Set as Date Table" setting for the Date table. Easy peasy. Can't I just count any field value rather that creating explicit measures? – Yes but don't. Coming from an SSAS multidimensional background, I learned to define measures for every useful numeric metric. If we needed to show the SUM or the COUNT of anything, we needed to define a measure for that. In Power BI and SSAS Tabular, you can simply use implicit measures – meaning you can just drag-and-drop any field and then indicate that you want to count each occurrence or sum up the numeric value of the field. For quick-and-dirty Power BI reports, this is a convenient way to get simple aggregate results but not a best practice in my opinion because of inconsistent behavior and other limitations. There is a growing consensus in the Power BI community that using explicit measures should be the norm and not the exception. If you always use explicit measures rather than aggregating numeric columns, users and report developers will always have a consistent experience. Validation 201: Measure Dependency Chains This is a fairly normal pattern: Create a simple base measure using a simple expression, let's say:[Employee Count]Create additional measures that reference and then perform some logic on the first measure. For example:[Employee Count Females] and [Employee Count Males]Layer another new measure on the second layer of measures. Like:[Female to Male Employees Ratio]Create another measure to apply special business logic, let's say to conditionally apply row-level security and non-visual totals:[Female to Male Employees Ratio (RLS Adjusted)]..and so on This is a simple example but it is not unusual. Reusing measures helps to separate reusable calculation logic but it can also create deep dependency chains. At some point, you will inevitably discover a calculation error that you will need to track-down and debug. I can't remember my own name some mornings so I certainly cannot work through the mental gymnastics needed to keep track of every dependent measure. When I have a visual that is not displaying the expected results, I usually do the following: Duplicate the report page and prefix the page name with TEST: (something meaningful)Delete visuals not needed for testingConvert the visual to a table so I can just see the numbers grouped by a fieldDrag the referenced measures into the table to create multiple columns. Validate each dependent measure. This helps to break the problem down into manageable steps and components. If you know that the first measure is always behaving as expected, move on to the next measure that depends on it. If the problem isn't immediately apparent, recreate the same conditions with slicers. By isolating the problem, I can find and fix the issue every time. How to Track Object Dependencies Keeping track of dependencies between layers of measures can be challenging. I keep a scratchpad on my desk to make lists of things I need to remember. Sometimes I don't even save my notes but it just helps me organize my thoughts. I use a whiteboard for a similar purpose, diagramming dependencies between measures and other objects with boxes and lines. Just standing and diagramming on the whiteboard is a very effective problem-solving technique for me. Electronic tools like OneNote and Microsoft Whiteboard can be just as effective if you are in the habit of using them. The Object Dependencies feature in Tabular Editor is pure magic and a tool I am using more frequently to test, debug and to document my data models. Tabular Editor Object Dependencies dialog Reusing Measures On a different but related topic… Create separate measures for calculations that will be used in visuals and reused by other measures but having too many measures can be difficult to manage. Strike a balance between code reuse and a more succinct, easier-to-manage portfolio of measures. If you have to jump through seven hoops (e.g. trace multiple dependent layers of measures) every time you need to debug or validate a calculated value, you probably have too many dependent measures. Organize measures but don't over-organize them… I prefer to put all my measures in a central table or at least assign them to just a few subject area fact tables. Don't leave measures in different tables, scattered throughput the data model. Although I'm a fan of using display folders, it can be more work to assign measures to several different folders than it is just to use meaningful naming conventions. The implementation of display folders in Power BI Desktop in my opinion is an unfinished feature. If you want to use display folders to organize measures within container tables, use Tabular Editor. Name measures so they're predictable and easy to find in a sorted list. For example, if you will have a series of measures based on "Sales Amount", don't name the variations "MTD Sales" or "Ratio of Sales to Prior Year". Instead name them with the same prefix, like "Sales Amount", "Sales Amount MTD", "Sales Amount YTD", "Sales Amount Ratio to Prior Year" and "Sales Amount Region % of All Regions". This way, they all get sorted and grouped together in the field list. The theme here should be finding the right balance to organize your objects while keeping things simple. Validation 301: Creating a Validation Report Solution, Part 1 This is a big topic and I have a learned some great techniques which will be covered in a follow-up post. So, let's call this teaser. How do users and business leaders know with certainty that they can trust the information in a Power BI report? There are a variety of factors that can affect data reliability. If the data source is known to be correct, there could be issues in the extraction, staging, loading, transformation or the calculations used to present the information in an analytical report. Testing the report with data at a point in time is a good place to start but it doesn't absolutely guarantee accuracy down the road. If we had a report with a simple set of aggregate values read directly from the business source of truth alongside calculated values from the data model after it has been refreshed, this can provide a litmus test to check and instill congoing confidence. When (I said "when" not "if") a user tells us that report information isn't right, we can confidentially and easily prove that analytic report information from the data model is consistent with the source. This is what a validation report is all about. To set the stage for a longer post on this topic, think about this typical scenario:Every month we load fresh manufacturing production volume and forecast records into our data mart or data lake, and then into the Power BI data model. The results of the data model can be used as a data source for the validation report. Using the XMLA endpoint, you can run a DAX query against the published data model to get the values shown in a report visuals complete with filters and slicers. For comparison, you can load data from a report produced by the business system, load those values into a the validation data model and compare the monthly totals. We can ne flag the data that does and doesn't match, and calculate the differences. More on this later. Creating a Validation Report Solution – Part 2 Part of the the series: Doing Power BI the Right Way (link) It has been a few weeks since I last posted, and a while since I started the promised blog series "Doing Power BI the Right Way". I've received a lot of positive comments on the posts in the series this far, and people wondering why it's taking so long. Well, I am busy working on a book about Paginated Reports and SQL Server Reporting Services. This will be the second edition of a book I co-authored ten years ago and this edition is also a collaborative effort with multiple contributors. We have quite a lot of excellent material that is just starting to go through the process of editorial and technical review. Writing a technical book always takes more time and effort than imagined, and working on this one in my "spare time" is a laughable notion but this is going to be a great book. Writing this blog series while juggling other commitments has brought me to two realizations: 1. Posting articles in the series will be sporadic, and 2. The topics will be in fairly random order based on the projects I'm working on and the topics that are most top-of-mind at the time. This brings me to the subject of this post: Paginated and Analytic reports. Before we had Power BI, we had Reporting Services. When the focus of my career and consulting practice became Business Intelligence, starting about fifteen years ago, most of the "BI reports" I created were in SQL Server Reporting Services (SSRS). We could create a variety of charts and reports that allowed users to drill-down from summary to details. We could create reports that allowed a user to drill-through, from one record or report item to another report that presented more details. With a little creativity, we could create reports that allowed users to click items that would filter other items on the same report. It took some work and time to create these interactive "drill-through-to-self" reports. Today, after transforming data into the right format for a tabular data model, this type of interactive functionality just magically happens in Power BI with very little report design effort. But, Power BI is primarily a tool for analyst users to create their own reports and to get answers to business questions rather than for IT developers to create reports that print or export perfectly to a PDF or Excel. Although it is possible to create fairly advanced paginated reports, most bread-and-butter operational reports are really just what I call "lists of stuff"… the results of SQL queries presented as a table or matrix; perhaps with a few groupings and subtotals. Simple business processes and task often require simple reports to help manage them. Paginated reports, by definition, are reports that allow a lot of data to be presented on multiple pages. Paginated reports can be viewed online, printed or exported to a variety of file formats. Becasue there is so much overlap in the capabilities of SSRS/Paginated Reports and Power BI, the choice between these two report tools is not always clear. I can cite many cases when using Power BI Desktop is clearly the right choice to create an analytic report, and cases where it would make more sense to create an operational report using Paginated Report Builder. I think these are the easier decision points but I would like to explore those cases where we could argue either way. What are the pros and cons of using Power BI to create traditional reports? When would it make more sense to replace a static, paginated report with an interactive report created with Power BI? The integration of Paginated Reports into the Power BI cloud service continues to march forward at a rapid pace. Just yesterday, Chris Finlan teased a blog post announcing new Power Automate integrations that will allow report to be distributed and exported using flows. Listen up for some big announcements at Microsoft Ignite about Paginated Reports in the Power BI service. Whether you host operational reports on your own report server or in the cloud, Paginated Reports are an important part of most business reporting portfolios. When do you choose to use Paginated Reports or SSRS rather than Power BI? Do you design reports differently than you would have 5 years ago? Do your users prefer interactive, self-service reports now or do they still want made-to-order static reports? What are the deciding factors between creating a paginated report or an interactive report? Do you create paginated reports with a Power BI dataset or SSAS model as the data source or do you only use SQL queries for those reports? I'm interested in your ideas and questions. Please post them in the comments below this post and then I will extend this post with your input and my thoughts. Part of the the series: Doing Power BI the Right Way (link) In a business intelligence solution, data must be shaped and transformed. Your data is rarely, if ever, going to be in the right format for analytic reporting. It may need to be consolidated into related fact and dimension tables, summarized, grouped or just cleaned-up before tables can be imported into a data model for reporting. If source data is in a relational database, chances are that it is optimized for transactional processing or application development. The column names, data types and formats are likely not suitable nor user-friendly for simple reporting. If source data exists in files or spreadsheets extracted from business reports, without data preparation you are likely just to reproduce the existing reports without adding business value. This important topic raises one of the most critical questions to consider when designing a Power BI project: Where should I shape and transform data? Data can be shaped and transformed in at least three different layers in the business intelligence process: At the sourceIn Power QueryIn the BI data model Your options for transforming data at the source depend on the capabilities of the data source. In general, a query running closer to a data source will be more efficient than one that must transfer a set of data, cache and manipulate it downstream. If you are importing text files or Excel files, then your only choice is to import an entire file and then potentially throw away what you don't need – which is fine for small-ish data sets. The right side of the previous diagram depicts data transformed in the data model. Some data morphing can be performed in DAX using calculated columns or calculated tables. In rare cases, there may be good reasons to do this but if you can perform row-level calculations in a database view or in Power Query, you will be better off. It is also easier to maintain the solution if you can keep all that query logic in the same layer so you are only using one design tool. Data shaping at the Source If we break the first choice down a little further, we could prepare data in an Extract, Transform & Load (ETL) process and populate a new table in the source database with correctly-shaped and formatted data for BI reporting. This is the approach used in traditional data warehouse solutions, typically under the control of an IT-managed data warehouse development group. Data Warehouse, Data Mart or Data Lake Back in the day, the conventional approach was to move data from sources into a staging database, and then to move it again into a dimensional database that was structured to support analytic reporting. I say this in past-tense because that was the way we always did it 10-20 years ago. Now, in the age of self-service analytics, it has become more common to build reporting projects on-the-fly and to transform data without moving all the data from one location to another before it arrives in the BI data model. However, there are limits to just how much data we can transform and how much complex transformation can be done in an ad-hoc fashion. The water here is muddy and it is difficult to say that one approach is always preferable to another but one thing is very clear: a Data Warehouse is very necessary in cases where data governance, high volume and centralized management are required. If requirements are well defined and unlikely to change, building a data warehouse or dimensional data mart might be the ideal approach. However, data warehouse projects can get stuck in long cycles of over-planning and requirement gathering which has led many organizations to think about about BI as an extension of business rather than IT processes. Most organizations have come to grips with the reality that the vast majority of BI projects are conducted after databases and data warehouses have been completed. Over decades, we have seen hundreds of companies chasing the Utopian dream of the "perfect data warehouse" along with Master Data Management, data cataloging, cleansing, governance and stewardship. Each of these goals can be achievable but are potential rabbit holes. Experienced data practitioners have learned to apply their experience to time-boxed and resource constrained projects with the goal to create "good enough" solutions that don't completely drain IT budgets and resources. The data lake has become a viable option for staging and storing data before moving it into a structured model because relational storage can be expensive. Data lakes provide a low-cost means to store a lot of data in the cloud. We usually think of the "modern data warehouse" platform consisting of a data lake and dedicated data warehouse database. Flavors of the lake/warehouse approach could include using a lake AS a warehouse, using a standard relational database to store a small/medium scale warehouse or using a dedicated parallel processing DB platform (e.g. Synapse, Snowflake or Cosmos DB) to store a large-scale data warehouse. Power BI and the Data Warehouse Bringing to discussion back to focus on Power BI queries, data models and reports; let me put a bottom line on this multifaceted topic. I believe that there are three data transformation scenarios related to Power BI design: Business-Centric, Informal and Small Model/Report SolutionsPower BI makes it easier to transform data as you go. In small projects, using this approach saves time and money, allowing business professionals to meet your own reporting needs. End of story… win-win.Interim POC SolutionsWithout a formal data warehouse, Power BI can be used to munge-together data from multiple sources and discover requirements that a data warehouse solution would provide. Use Power BI to build a prototype or proof-of-concept solution that can eventually evolve into a formal solution. Accept the follies and trade-offs of ad hoc data models and use them to evolve toward the right solution using iterative design. Use Power Query to perform all necessary transformations and cleansing operations.Formal Data Model & Report SolutionsOnce properly governed and secured data warehouse/marts exist, rebuild Power BI data models and reports. The "rebuild" effort could encompass anything from switching data sources in an existing report project or starting from scratch. Either way, you are saving time by applying learnings and design elements. If additional query logic is needed, use database views to promote query folding in Power Query. The simple fact is the making such a transition is never "done". It is a journey that requires iterative thinking and management. Accepting and embracing this reality will save everyone a lot of angst. With data cleansing and transformation applied to ingested data warehouse data using tools like Azure Data Factory (which may utilize Power Query or SSIS), SSIS or Databricks, data sources for the Power BI project may simply be tables or views. Where and what is Power Query? Power Query is the data transformation technology built into Power BI. It is an extremely flexible and extensible data shaping platform that evolved from tools like SSIS, DTS, Excel, Microsoft Query and Access. When you choose Get Data or Transform Data from Power BI Desktop, this opens the Power Query editor. The Power BI service includes an online implementation of Power Query called Dataflows which allows queries and entities to be share between Analysts creating different data models. Power Query is also now integrated with Excel (when you select Get Data), the Visual Studio SSAS Tabular model designer and Azure Data Factory. Learn to leverage Power Query to perform data transformation efficiently to perform all row-level transformation and calculations. In general, Column naming and data type changes should be performed in Power Query rather than embedded SQL or DAX. Understand Query Folding This is a critical concept to understand about Power Query. For supported data sources and whenever possible, queries you create in the Power BI query editor will be converted to native queries to be executed at the source. This means that if you are using SQL Server, rather that transferring all the data from the database engine to the Power BI service before performing transformations, steps may be converted to an optimized TSQL query. This is an ideal scenario for high-volume fact tables. Query folding only works when Power Query can select records from a single database object like a database table, view or table-valued user-defined function. Hard pills to swallow For relational database professionals, there are a few realities about Power BI that can be a shock to their pattern for solving data problems. Without getting into all the details about why things are the way they are, it is very important to understand and embrace these points: SQL queries do not fold – If you hand-write a SQL query or copy and paste SQL script into a new query within Power BI, Power Query cannot fold – and therefor cannot optimize – that query. This doesn't mean that this is wrong or necessarily a bad practice. If you need to populate a relative small table in the data model that doesn't need to be filtered, grouped, sorted or otherwise transformed in ways that could otherwise be optimized; this is probably fine. Selectable database objects can be folded – If you can use a single database table or create a view in the database, particularly for high-volume fact tables, these queries will be faster and will consume fewer server resources. Stored procedures – Power Query doesn't include specific optimizations or support for stored procedures. I have successfully used stored procs as data sources, at the insistence of DBAs, but this is not a recommended design practice. Parameterized filters Queries for large tables should be designed with filters so you can easily control the records imported – both at design time and when you need to populate the data model with a full set of records. Even if you don't need to manage millions of records or load a table incrementally, this is just a good idea and will save a lot of redesign later on. I use parameters to manage anything that might change, like folder paths and server names. The example shown here conforms to the requirements needed to implement the incremental refresh feature if Power BI: The RangeStart and RangeEnd parameters are then used in a Filter Rows step in the query to limit the imported records within the selected date range. Incremental refresh One of the perks of using a date range filter is that it enables incremental refresh. This is one of the coolest features to come along in a long while and now it works with both Premium and share capacity licensing. If queries support query folding and use a set of date range filters to limit the records into a table in the data model, Power BI will automatically generate table partitions. For example, if I were to partition a table by month, each partition includes records for only one month and data in each monthly partition is only loaded when new records are available or when data has changed. I design my fact tables to support incremental refresh as a standard design pattern whether we have a requirement to implement it or not. This gives me the flexibility to only load a small set of data I need for development. Publishing a smaller PBIX file is faster and then more data can be loaded after the dataset has been published to the service. If you don't need it, throw it away The presence of every single column in your data model comes at a cost. It uses memory, storage space and can potentially impact performance. If you aren't absolutely certain that you need a column, just omit it from the table in Power Query. If you or you users change their minds, columns can easily be added. Before you ask, this is not a picture of my garage but according to my wife, it could be. Keeping all the columns in your data that you "might need someday" is a lot like all that crap I've been collecting in my garage and on my workbench that I tell myself might be useful some day. Error on the side of omitting columns that you don't need to meet immediate reporting requirements. The difference between the cluttered garage analogy and reality of Power BI design is that you immediately benefit by getting rid of unneeded data right now and it costs you little or nothing to go back and add one of those columns later, after your business stakeholders decide they need it. The easiest way to manage this is to use the Choose Columns button on the Home ribbon, and then just deselect the columns you don't need. In the future, use the gear icon next to the generated step to make changes to the column selection. Column naming Please read this aloud and commit it to memory: Rename all columns that will be visible in the data model using concise, self-describing, meaningful and user-friendly names. I can't emphasize how important this is. Not only is this good practice in BI solutions, but there is a psychology to the way business users perceive what they see in reports. When they see cryptic labels and field names, they become uncomfortable and may distrust the information. Your data model should be designed for users and not for developers to design reports and consume. Even if technical professionals are designing reports, field names are going to show up as labels and titles. Consider the following examples: Friendly Column NameTechnical NameCustomer NumberCustomer NbrCustomer #customerNumberCustomerNbrCUST_NBRProduct Brand NameproductBrandNameProdBrandNmPRD_BRAND_NM Technical naming conventions are deeply engrained into the culture of application and database design and for good reason. For developers and database professionals, certain naming patterns like Camel Case (camelCase) and capitalized word abbreviations separated by underscore characters are almost a right of passage; but object names in this format should not exist in a data model for analytic reporting. Spaces in column and measure names are welcome, as are punctuation and other special characters like #, $, %, etc. Table and fields should be named so they are familiar and easy to understand. When you or a user designing a self-service report drags and drops a field or measure into a report visual, these names are used as labels and titles that should also be meaningful and self-explanatory. Field names should also completely describe the object. Consider that if the Product and Customer tables both have "Name" columns, if you were to drag the Name field from the Customer table into one visual and the Name field from the Product table into another visual, each would only be labeled "Name". It would be more appropriate to name these fields "Customer Name" and "Product Name" in their respective tables. Naming & annotating steps Rename the steps in Power Query so that they are self-describing (like the following example). Your future self will thank you when you revisit the project in six months. This is done by right-clicking the step and choosing Rename from the menu. A new feature in Power Query displays a tooltip with annotations you add to the M script or the text you enter in the Description property of the step. The names of each step are applied as variable names in the Power Query "M" language script. Variables that contain spaces or special characters are proceeded with # and are encapsulated with double quotes. The comments in the code I added with // prefixes result in the tooltip annotations I mentioned in the previous paragraph. A recent suggestion from one of the blog readers is to use underscores rather than spaces in these step names. This allows you to double-click the variable name and copy it to the clipboard. Consolidating steps Building queries using linear steps helps break-down complicated logic but it can become inefficient as the project evolves. In the course of query design, you are likely to repeat steps like those in the second example below where I removed three different columns after using them in other steps. I also changed the data type each time I created a custom or duplicated column. If you select an existing step and then perform the same action on a different column or columns, Power Query will add the new transformation to the selected step. In the query on the left, I consolidated all these steps, removing columns and changing data types only once. The optimized query on the left runs in about one-third the time is the original query on the right. Power Query Best Practice Guidelines Here is a summary of best practice recommendations: Set EVERY column to the right data type. Watch for untyped columns with "ABC123" in the column heading.Rename all columns that won't be hidden in data model (keys and sort columns are OK).Consolidate redundant steps.Rename steps to document transformations.Perform row-level calculations in Power Query instead of DAX calculated columns.Use database views & avoid SQL statements so Power Query can implement query folding.Use parameters & range filters to reduce working set for development, reduce PBIX file size, speed up desktop refresh and reduce time to publish dataset.Use date range parameters in service to expand dataset size during data refresh & optionally implement incremental refresh. Part of the series: Doing Power BI the Right Way (link) When starting a Power BI project, you have many choices to make. Decisions like how to source your data, where and how to create queries to cleanse, transform and reshape the data; where and how to create calculations and the nuances of modeling are just the tip of the iceberg. Most Power BI solutions progress through a few stages in their lifespan, which include: Creating a simple prototype to explore ideas and to get feedback from users and business leaders Building a working proof-of-concept to frame the solutionDesigning a foundational data model to build uponCompleting a production-scale solutionIterative enhancements & releases The purpose of this post is to provide some guidance to help you design Power BI solutions that can survive the journey through these stages with as little "throw-away" design as possible. There will always be prototyping and redesign in any project but if you follow proven design patterns and lessons learned from prior experience, you'll have a much better chance of building a reporting solution that will endure. Oh, I should mention this… the difference between item #1 (the simple prototype) and #2 (working proof-of-concept) is that you need to throw away the first one – and anything else that doesn't lay a foundation that you can build on top of. This is a hard lesson but one that will pay off by helping our teams, sponsors and business stakeholders understand that we can proceed after laying down solid bricks (based on well-defined functional and technical requirements) to build the rest of the solution. Reality Check If you are using Power BI to create reporting solutions for business users, your circumstances are likely similar to one of these scenarios: You are a self-service report designer creating a small report solution that will grow and evolve over time.You work in a software development group in an IT organization with a formal application lifecycle management discipline.You work in a business unit or small company with an informal software process – and they need your guidance to be successful. If you are a self-service report designer creating a small report solution that will grow and evolve over time, you likely have a data source that wasn't specifically curated for reporting and analysis. The data will need to be massaged and shaped before you can design insightful, interactive visuals. That's cool because we have great tools built into Power BI Desktop to do that sort of thing. You might have some ideas about how to present the data but you'll probably try different things and see where the journey takes you. These projects tend to have a lot of business impact right out of the gate. Business report users will be delighted with what you can do in a short time with Power BI and they will want more. You'll get excited and expand the project to include more data or more dimension to the data you have. It is almost inevitable that you will experience two common challenges: Scope creepData quality issues These are not necessarily "problems" per se but realities to acknowledge. Building a solid, foundational data model is challenging in under these circumstances. The good news is that Power BI let's you explore options quickly and allows you to try different approaches while you sort-out the formal business and technical requirements. The typical outcome is to start over with a new project where you will be well-prepared to use a cleansed data source and design to a well-defined set of requirements within the new project scope. An experienced BI practitioner will set this expectation with the business and either carefully evolve certain elements into the final project and toss the rest, or use the first attempt as a throw-away prototype and start over. Just know that this is a natural part of the process. If you work in an software development group in an IT organization with a formal application lifecycle management discipline, you are probably accustomed to using team development a code repository, version control and differencing tools to manage a shared code base. The main components of Power BI came from a long heritage of IT-centric development so at its core, objects (like tables, fields and measures) have properties and metadata descriptors that can be versioned, differenced, merged and scripted. But what has made Power BI so successful is that is primarily designed for self-service reporting. Power BI Desktop is a beautifully streamlined product that packages data transformation queries, model tables, data and reports into a tidy package but it is not an IT development tool and doesn't have these capabilities built in – nor will it. That's not what Power BI Desktop was designed for. The good news is that the community has rallied to develop tools to meet the needs of IT developers who work in teams to develop, test and deploy formal, enterprise-scale BI reporting solutions. Power BI and the rest of the Microsoft BI platform – can be scripted and automated to create custom deployments. Right-sizing solutions Before creating Power BI, earlier versions of Microsoft's Business Intelligence components existed as separate products. BI reporting projects took many months to design, develop and deploy. By contrast, today you can open-up Power BI Desktop and create a quick "report" (PBIX file). In this form, the file contains data transformation queries, an in-memory data model with measures and other calculations, and report pages that include the interactive visuals. This design typically meets the needs of small and intermediate sized projects. One of the first questions to consider is whether the solution you intend to create with Power BI is a quick informal project or a formal project – or may become a formal project in the future. Informal projects don't necessarily have to conform to all the design best practices. They are limited in scope and typically meet a very specific and short-term need. Formal projects are built on best practice design standards to endure a certain degree of expanding scope and are designed for longevity. They have a business sponsor or champion who drives the business requirements and at least one developer who is responsible for designing a solution that meets those requirements. Formal projects are typically testing, signed-off an then deployed in accordance with an established development discipline or application lifecycle process so they can be supported by the business. If your organization has a process for delivering and supporting company-sanctioned applications and reports, then you will need a roadmap to transition from an informal project to a solution that the business is prepared to support. A few common questions that come up in these discussions are: How is the source data procured and maintained? Who is responsible for maintaining that data?Who from the business defines the reporting requirements and signs-off on Power BI reports meeting those requirements?Who owns the development of the Power BI solution (data model and reports)?Who is the developer's backup if they become unavailable?Are the requirements and project assets sufficiently documented to support such a transition? What are the security requirements? Who and how will users be given access to reports or dataset(s) for ad hoc analysis? Are users assigned to report and dataset permissions directly or through group membership?Should users have conditional or restricted access to data within the Power BI dataset?What is the process for requesting new reporting features, adding features to the existing solution, testing and deploying updates? Personal BI projects, where a data analyst imports data, creates reports and then publishes a report for their own use serves an important need in business. These reports can easily be promoted to a departmental or business area solution by sharing the report with others. As the scope of the project continues to expand, there are several important considerations that define the threshold between an informal project, designed for quick development and convenience, and a formal project; intended for longevity and long-term business support. For each project component, I start with a typical informal project scenario and then provide recommended practices to to plan for futureproofing. Planning for separation Designing transformation queries, data model and reports in one PBIX file is convenient and offers quick troubleshooting, debugging and redesign. However, it also limits development to one person at a time and promotes a monolithic solution. Separating the data model from the report files helps to promote self-service and focused report design. The certified or promoted dataset is deployed, secured and managed as a separate object. Reports are connected to a Power BI dataset live connection and any number of reports can connect to a published dataset. This pattern supports both fully IT-managed projects with multiple developers and it supports self-service report design using certified and promoted datasets. After the data model design is stabilized and locked-down, report designers simply focus on report creation and data analysis by connecting to the secured dataset. Any tools and products capable of connecting to Power BI or SQL Server Analysis Services (SSAS/AAS) can be used for reporting and ad hoc analysis. Source data Spreadsheets and files extracted from reports, manually manipulated using Excel or other line-of-business applications have limited utility as reliable data sources. Manually preparing spreadsheets and source data files is only feasible for small sets of data and one-time or manual data loads. An analyst user can easily remove, rename or add a column or formula to a sheet that will break or negate the import process. Futureproofing: Data from source systems can be loaded into a managed database or data lake before using Power Query to load the data model. Data from files (if carefully protected and maintained) may be transformed, cleansed and validated before it is ready to be loaded. Using "brute force" transformation steps to replace values and cleanse records one step at a time that can be slow and difficult to manage as the complexity and volume increases. Reduce the number and complexity of query steps and apply the following recommendations for Power Query design optimization. Use parameters in Power Query to manage any variable information such as database servers, database names (e.g. DEV database, QA database & PROD database) and file paths. Planning for data volume In smaller, informal projects we typically designed for a limited set of data and fixed time-frame. If manageable, all production data can be loaded into the working PBIX file on a developer's desktop. Futureproofing: When working with large data volumes, a subset of data should loaded into the desktop dataset for development. Query parameters are used to expand the number of records (usually using a date range) loaded after the dataset is deployed for production use. Groups and ranges of data records, often partitioned by months or years, are defined and stored separately so a large volume of records don't need to be reloaded along with new records. Power Query design practices Using the robust Power Query design user interface (ribbon and menu options), you can build sophisticated and complex transformations that work well with moderate data volumes. With a small number of query steps, default names like "Rename Columns" and "Change Data Type" are fine but it can be difficult to trace the steps and remember where and why you performed each step. Futureproofing: Tables and fields should be meticulously renamed to use standard friendly names, categorized and optimized with appropriate data types. Unused columns are removed to save space and improve performance. Spend extra time to get the field names right before creating dependent steps. (see; Chris Webb: Names, Tables, Columns & Measures In Power BI) Consolidate redundant query steps – like those used to rename columns, remove columns and change data types – into a single step in the query.Attribute fields needed for grouping and filtering should be moved to separate dimension tables. Dimension unique keys are defined with corresponding fact table keys.All numeric columns used for aggregation (which share the same level of grain) should be extracted to fact tables, with corresponding dimension keys.Row-level calculated columns should be defined in either in database views or Power Query tables rather than DAX calculated columns.Promote query folding by using database tables, views or table-valued user-defined functions in the database rather than writing SQL statements in Power Query. Test and optimize queries to support query folding by checking to see in "View Native Query" is an available right-click menu option on each query step. Power Query also doesn't work well with stored procedures.For large fact tables, design them to be "tall" rather than "wide" by removing unneeded columns or moving them into dimension tables. Tall tables should only include columns that compress efficiently, such as numeric values and short, non-distinct text values. Do not store a single unique key column in a fact table.If large tables take too long to load or cause the PBIX file to exceed a reasonable size for development (no more than 500 MB), use parameters to limit the range of records. To manage the table size once deployed tot he service, do one of the following:Many modelling techniques that perform well when tested on small data sets will slow down with large tables. These include relationships on bridging tables, many-to-many relationships, bi-directional relationship filters and multi-measure layered calculations.Avoid using SWITCH statements over already complex DAX expressions or referenced measures. Avoid using iterator functions (like SUMX) with complex row-by-row predicate logic. Separating calculation logic into separate measures is a useful design pattern to segregate business logic but it can create performance bottlenecks. Identify slow measures and consider building streamlined measures using alternate patterns when bottlenecks are discovered.Import mode tables are generally faster and more flexible than DirectQuery mode tables. However, DirectQuery is a good option for very tall transactional tables where simple aggregations are used, rather than complex DAX measures. If Import and DirectQuery mode tables are both used in a composite data model, consider creating a summary table using Import model and a detail table using DQ, and then define aggregates to cache certain query results in memory (to compensate for the normal performance degradation resulting from the query translation in DirectQuery.)There is so much more to discuss at this point but I'll need to address additional topics in later posts in this series titled "Doing Power BI the Right Way". Topics include:Multi-tier and advanced self-service solutions can be architected by further separating the transformation queries from the data model using Power BI dataflows. These are server-hosted Power Query/M queries that are both designed and managed in the Power BI service.Dataflows expose several integrations with the Microsoft Azure cloud services, such as the Common Data Service and Azure Data Lake Gen2 Storage. Dataflows also have integrations with AutoML machine learning models.The advantages and disadvantages of dataflows should be carefully considered before jumping in head-first. Power Query in Power BI Desktop is a fast, right query development environment that provides nearly immediate feedback. By contrast, the browser-based dataflows designer is similar in function but slower and has a less-robust query design feature set. Some developers choose to develop Power Query in Desktop and then port their M query code to dataflows as needed. To wrap up this edition of the series, I will leave the rest of my outline points as teasers for subsequent posts. Please freely comment with your thoughts about what to include in these later editions.The rest of these are random thoughts left from the outline for the series. Feel free to comment.Know and understand the business audience Paul TurleyPower & Data Platform MVP, Principal Consultant, Pragmatic Works Preface: This post is set set of guidelines that address many aspects of new projects. Although I feel strongly that many of my recommendations are correct and applicable in most uses cases, other experienced practitioners have valuable ideas and I welcome their input. Please post your comments, links to other resources and ideas that may be different than mine. This will be a living document with ongoing revisions with the goal to provide a comprehensive, best practices guide as the platform continues to mature and as experts continue to use it. Revision: 8/25/2019 – cleaned-up & added checklistRevision: 9/7/2019 – added Security section & corrected Last Refresh Date/Time I find there there are so many things to remember when starting a project that a checklist is handy. I've been collecting the following as notes for some time. Bare with me as I work on consolidating this article into a concise checklist. The purpose of this article is to outline a set of guidelines and recommended practices for managing Microsoft Power BI projects. This guide is primarily focused on the work performed by the IT-managed BI Solution Developers for formally-managed BI projects. Why "Good" and "Best"? Power BI is a tool and platform that serves two masters: IT developers and business data users. As such, some practices are absolutely best-practice and should be followed with rare exception. Other recommendations are conditional guidelines, with possible exceptions. The flexibility of Power BI makes it possible for industry practitioners to developed their own solutions to apply design patterns in the absence of a specific product feature (like source control, for example). This guide is largely based on my experience, discussions with other members of the MVP community, clients and fellow consultants. Most of the information in this recommendation guide are based on the consensus of parties with the greatest experience with the Power BI platform. These parties include: Members of the Microsoft MVP Program (Data platform and Business Applications focus) who participate in daily interactions with the Power BI product development and leadership teams at Microsoft.Microsoft certified solution partners who offer consulting, training and value-added services to Microsoft product customers. Power BI has come a long way in a short time; since the Power BI service and Power BI Desktop were introduced in 2015. At that time, the emphasis was to enable self-service business users to analyze and visualize data in small-scale ad hoc reports. As Power BI grew into the enterprise space, it has been adopted by IT-managed data governance and solution development teams. Power BI is displacing – and being used in concert with – conventional IT-centric Business Intelligence product technologies that have been evolving for decades (such as SQL Server Analysis Services, Reporting Services and Integration Services). With this history and the rapid adoption of the platform, standard design practices will continue to evolve. Two discreet audiences and roles should be considered for Power BI: Business Data AnalystIT-managed BI Solution Developers Consider these roles, with respect to both the solution developer(s) and users, as you begin to plan a new project. Business Data Analyst Members of this role are part of a rapidly growing segment of the industry. They understand how to use software tools and understand data manipulation and reporting within the confines of area of business focus. Traditionally, data analysts were data consumers who use financial spreadsheets but now they use advanced data preparation tools to perform reporting and deeper data analytics. Business Data Analysts generally fall into two subcategories: Advanced Data Consumers These users explore data using spreadsheets, pivot tables and chart-style reporting tools that are served by existing multidimensional cubes and data models. They do not write code and work with IT/BI Developers, Data Modelers or Advanced Data Analysts to collect and model the data they use to create their reports.Data Modelers, Advanced Data Analysts and Data Scientists These advanced users perform their own data mashups, produce data models and write limited code to create solutions for their department or business area (using Power Query, DAX, R, Python, etc.) They do no architect and develop governed and certified datasets for enterprise-level business use. IT-managed BI Solution Developers Enterprise-scale BI solutions are typically created by a team of specialists who may include: Business Systems AnalystsBI Solution ArchitectsProject ManagersData Transformation (ETL) DevelopersEnterprise Data ModelersMeasure code experts (using DAX)Report DevelopersOperations & Deployment Engineers Power BI may be used alone or with other products to create BI data transformation queries, data models, reports and dashboards. IT-BI managed scenarios: Complete BI solution design, development and deployment, consumed by end users.Data models created and deployed to support user ad hoc report design.Support Advanced Data Analyst data transformation, data modeling & report designMigrate advanced user created data models to governed/certified datasets, transferring ownership to IT & operations Project Types – simplified For simplicity, this guide refers to Power BI projects as either "Formal" or "Informal" using the following definitions: Formal projects are scoped, funded, staffed and executed with the collaboration of a business champion and stakeholders; and IT Business Intelligence developers and data managers. These projects promote business and IT-governed datasets and certified reports.Informal projects are executed by business users and are considered ad hoc in nature. Datasets are generally not IT governed, and reports are typically not certified.Hybrid projects can be anything in-between. They might be a user-authored report using published, certified dataset used for self-service reporting. Informal, self-service datasets can be migrated to governed datasets in collaborative IT/business projects. Again, this guide is primarily focused on the work performed by the IT-managed BI Solution Developers for formally-managed BI projects. Topics included in this guide: Solution ArchitectureManaging Power BI Desktop FilesDatasets and ReportsVersion Control & Lifecycle ManagementWorkspace and App ManagementData Model and Power Query Design GuidelinesDimensional DesignQuery OptimizationManaging Dataset Size with ParametersImplicit and Explicit MeasuresIf Users Need Excel, Give them ExcelCertified & Shared DatasetsEnterprise Scale Options Solution Architecture All Business Intelligence projects involve the same essential components including: Source QueriesData transformation stepsSemantic data modelCalculations (typically measures)Data visualizations that reside on: reports report pages dashboards These components can be built using different products and development tools. Lengthy discussions can be had to debate the nuances and merits of different options; but a short summary of recommendations follows. To summarize these options in simple form, first consider the following options: Single PBIX File Build all queries, transformations, the data model, measures, reports and pages with Power BI Desktop in a single PBIX file and deploy to the Power BI cloud service; to be shared with users. Recommendation/Cost/Benefit:This option is most suitable for Business Data Analysts, erring on the side of simple design, typically optimal for small, informal self-service projects where a desktop business user needs to perform their own data mashups and quick, ad hoc analysis. Risks include unknown data integrity and the use of uncertified dataset results. Informal projects can effectively serve as prototypes for formally developed and managed dataset development. Single file projects can be migrated to dataset/report two file projects with some rework required for the reports, depending on the report complexity. Separate Dataset & Report PBIX In formal projects, use Power BI Desktop to author a separate PBIX "dataset" file containing the following: queries, transformations, the data model and measures. These are in turn deployed to the Power BI cloud service. Develop one or more separate PBIX "report" files that connects to the published dataset file in the service. All report pages and visuals reside in the "report" file. Some additional measures can be created in the report file. Recommendation/Cost/Benefit:This pattern is recommended for most projects where data governance is important and where new report features are likely to be requested in the future. By separating the dataset from report development; the data model design can be locked-down and managed by a data steward. This approach provides report designers freedom to add visuals and make report enhancements without risking the stability and integrity of the data model that may be serving data to other reports. Development and management of a two file project is a little more cumbersome because the dataset must be redeployed before changes are available in reports. Self-service report users who need to explore and visualize data, can use a governed and certified data model. Changes and enhancement should be coordinated between the report designer and the data steward or IT resource managing the data model. SSAS with Power BI Taking the "traditional route" in formal projects to build the data model in SSAS, hosted either on-prem or in Azure Analysis Services, and then creating reports and visuals with Power BI was a necessity under enterprise-level feature were added to the Power BI platform in 2017-2018. Although there are still some good reasons to use SSAS rather than Power BI to build the data model (called a "dataset" in Power BI), Power BI can be the more cost-effective choice today. Power BI and SSAS both offer unique capabilities which doesn't clearly make one option always better than the other. You can develop the source queries, transformations, data model and measures in an Analysis Services (SSAS) project using Visual Studio (also called SQL Server Data Tools or SSDT) and in turn deploy the SSAS database/semantic model to an on-premises SSAS instance or Azure Analysis Services (AAS) cloud service. Connect new Power BI reports to the deployed SSAS model using a direct connection. SSAS/AAS can handle data models with more than the 1 GB, 3 GB & 12 GB compressed data size limits of Power BI. The SSAS development environment can be more difficult to navigate and the model designer can be unstable. When the VS/SSDT designer crashes, developers can lose work even if they are routinely saving their work. Recommendation/Cost/Benefit:Favor using Power BI. It typically takes considerably longer to develop data models in Visual Studio/SSDT than using Power BI Desktop. This project architecture is no longer the primary recommendation for BI solution development. Microsoft is directing most of their development and support resources to Power BI rather than SSAS. Although SSAS still has some advanced capabilities not yet present in Power BI; Power BI can handle larger data volumes in certain cases. Microsoft and Microsoft partners recommend using Power BI is the tool for developing most new BI solutions. DirectQuery vs Import/In-memory Power BI desktop using DirectQuery mode, rather than import mode to reference tables in an underlying data mart or data warehouse relational database. No data is stored in the Power BI or SSAS data model; live SQL queries are generated when navigating report visuals. Recommendation/Cost/Benefit:Avoid this option, unless needed to satisfy unique requirements. Power BI and SSAS both are favored and are optimized to use: import mode, in-memory tabular data models for all but rare and exceptional scenarios. DirectQuery significantly limits the DAX Calculations supported in the data model. DirectQuery speed is typically many times slower than import, in-memory mode. The advantage of DirectQuery mode is that queries are executed directly against source tables in real-time. Consider building composite models with both import and DirectQuery mode to get good performance and drill-down to live transactional details. Composite models are typically complex and more time-consuming to develop, test and deploy. Managing Power BI Desktop Files Power BI Desktop files (PBIX and PBIT) should be stored in a centrally managed network-assessable folder, where they can be accessed by all members of a BI report development team. The storage folder should support automatic backup and recovery in the case of storage loss. Options depend on currently supported services and organization preferences within the following guidelines: File Size and Folder Synchronization Report and dataset developers must open files from the Windows file system and not from a web browser interface. Files must either reside in or be synchronized with the Windows file system.Files containing imported data typically range in size from 100 to 600 MB. Any shared folder synchronization or disaster recovery system should be designed to effectively handle multiple files of this size. Options: OneDrive For Business (shared by team, with folder synchronization).SharePoint or SharePoint Online (with folder synchronization).GitHub and/or VSTS with local repository & folder synchronization. If used, Git must be configured for large file storage (LFS) if PBIX files are to be stored in the repository. File Naming Give files an intuitive and friendly name, that will result in them not being confused with similar report projects. In default view, only the first 20 or so characters are visible for each report. Use self-evident names that can be used to identify datasets and reports, using the beginning of long names. Workspace and report names have limited space in the Power BI portal navigation bar. Use short and concise names when possible; but try to use names that differentiate like-named items in the first 20 characters. The full name is visible in the hover-over tooltip or by resizing the panel. Datasets and Reports Formal report projects with certified datasets should be developed with reports and datasets divided into separate files. By default, all Power BI objects, including reports and datasets, are stored in a single file. When the PBIX file is published from Power BI Desktop to a workspace; the PBIX file is separated into separate dataset and report objects. This default behavior is convenient for informal, small-scale, self-service analysis; but not optimal for managing formal solutions with governed datasets, managed by teams of developers or report authors. Separating the dataset and report into different PBIX files; allows them to be managed and versioned separately. This approach also promotes a division of labor and ownership, particularly for certified datasets that must undergo scrupulous validation. Like the way SSAS projects are used to develop multidimensional cubes and tabular models; deploying a separate Power BI dataset provides an uncomplicated black box solution for ad hoc report design and data exploration. Power BI can now be used to develop large data models containing 10s of gigabytes of compressed data. Using these development tools take far less time and possibly without additional licensing fees. Version Control & Lifecycle Management At the present time, Power BI has no integrated version control mechanism. In addition, Power BI is incompatible with most version control systems that perform branching and differencing operations on code and script files that are common in application development projects. Power BI (PBIX and PBIT) report and dataset files are stored as binary archive files with internal checksum metadata. Change and differencing operations performed by most code and version management systems will corrupt these files and render them inoperable. Reality: Integrating changes made to multiple PBIX files is difficult to manage and challenging to perform technically. Even with differencing tools and effective version-control, just avoid multi-developer work on Power BI dataset files. The Power BI ALM Toolkit (based on the older BISM Normalizer tool for SSAS Tabular) will make it easier to compare differences. Until (or if) specialized tooling becomes available, you should avoid replying in this method to manage project work. Power BI Desktop PBIX files that contain imported data can be large; typically, in the range of 100-600 MB. However, they can be checked into a version control repository for the purpose of backing up and restoring files. PBIX files can be reduced to small files by saving them to a Power BI Template (PBIT) file. This removes all data and may be an effective to make occasional backups for version archive purposes and disaster recovery. The template can then be used to generate a new PBIX file; much like using a template to generate a new Word, Excel or PowerPoint document. Saving a PBIX to a template for backup purposes takes minimal effort; but rehydrating a PBIX from a template is more labor-intensive and time-consuming. The reason for this is that a new file must be created and then queries must reprocessed to populate tables with new data. Recommendations: File versions Store the dataset and report PBIX files separately using a version number postfix for the file name in the following the format: Major.Minor.Revision. For example: Manufacturing Cost Analysis Dataset v1.5.3.PBIX Keep only the previous file versions that are useful for historical reference and recovery. Report files are small and don't take up much storage; but, dataset files can be large. If you need to archive older dataset files, resave them as Power BI Template (PBIT) files to reduce the file size. However, a developer must continue to work with the current PBIX file. Recovering a dataset from a template is time-consuming so developers cannot actively develop using only template files. Remove the version number from file names prior to production deployment. Dataset Version History By keeping version history information in an internally-generated table in the data mode; durable revision history notes and version information is self-contained and not dependent on external data sources. Add a Version History table to the data model by using the "Enter Data" feature in Power Query to create a table containing these columns: Version NumberRevision Date/TimeDeveloper NameRevision Notes Add a custom column named Refresh Date/Time that will be used to capture the date & time every time the queries & model is refreshed. This custom column could be added to any table but it is best to use a table with few rows because the value will be the same in each row. DateTime.LocalNow() Add the following measures to the model: Measure Name Definition Last Refresh Date/Time = MAX('Version History'[Refresh Date/Time]) Current Version = MAX('Version History'[Version Number]) Create a Dataset Information page in the dataset PBIX file and in any report that references the dataset. Every time a revision is made to the data model, edit the Enter Datastep and add a row to the table with a new version number. Use the following guidelines to increment components of the version number: Component Major releaseIncrement only when a newly scoped solution is released to PROD.Major releases have a specific set of planning features. A new major release is created only for a new set of planned features or significant enhancements as part of a new scope of work.Minor Version NumberAs a team, decide when changes are significant enough to warrant an incremental minor version release; typically, when changes are made to QA or PROD releases within a major release. Generally, for minor featureenhancements: For a dataset, minor version changes may break or affect a report's design.For a report, minor version changes add or change functionality aside from fixes.Fix or Revision NumberIncrement for every bug fix or design change, that will be deployed to the DEV environment or visible to other developers. Generally, for bugfixes but not for adding features. May be incremented by the developer. Resetto zero with a new minor version number. For example; the following version number represents Major Release Version 1, Minor Version 5 and Fix/Revision Number 3: v1.5.3 Datasets and reports should be versioned separately. The major version number for reports and datasets may be synchronized for major workspace app releases. Be mindful, that the purpose for version numbering is to provide at simple standard to keep track of changes. The process can easily be over-thought and over-engineered. Keep it simple, discuss changes as a team and make process adjustments to focus on priority tasks. Release Management, DevOps and Automation (topic draft – expect updates – please comment) This is an area where the Power BI platform currently lacks a formal process or mechanism. We know that release management tools are being developers but in the early stages. We may or may not see a release management too from Microsoft that blends with other tools that you might have in use, such as VSTS and AzureDevops. In my experience, automated Power BI release management isn't really being practiced in the industry yet. That said, there are techniques available. PowerShell script may be used to publish datasets and reports, and to change dataset bindings. It is possible to either publish to a production workspace or to effectively move assets from one workspace to another. This approach is discussed briefly in the Power BI Enterprise Deployment Guide. Other approaches are discussed here: Power BI release management PLEASE ADD REFERENCES Workspace and App Management To share reports to users who do not have a Power BI Pro license (each license has an additional cost) must be in a Premium capacity workspace and the workspace must be published as an app. Create a separate workspace to share with Free licensed users Create a separate workspace for formal projects that have a report, collection of reports and/or dashboard that should be distributed to multiple view-only users. A workspace must be converted to an app to be shared with users who do not have Power BI Pro licenses assigned to them. Once the workspace is published as an app; it can be shared with any group of "free" users or all users in the organization. The visible text for workspace names is limited; name workspaces so they be uniquely identified in about 20 characters or less. For a formal project, create the following workspaces: DEV Workspace Only development team members need Contributor access to this workspace. This workspace does not need to have Premium capacity; unless, developers need to unit test incremental refresh or other Premium features. QA Workspace All testers must have View access for testing and Contributor access for report authoring. Should be in Premium capacity to test

incremental refresh. PROD Workspace Omit the “PROD” designation in the name. This workspace will be thename of the published app that users will see in their Apps, Home and Favorite pages so use a name that is simple and sensible. Must have Premium capacity to share the app with non-Pro licensed users. Use a common workspace for informal reports not shared with Freelicensed users Multiple self-service and informal report files can share a common workspace without converting the workspace to an app. All users accessing any of those reports, must have a Power BI Pro license assigned to them. Informal or self-service Power BI projects typically don’t require separate Development or QA workspaces. Data Model and Power Query Design Guidelines The topics in this section are not an exhaustive guide to Power BI query and data model design. Following are common issues and recommendations often encountered in projects. Dimensional Design Always design data models for optimal performance, even when it “doesn’t matter” in simple projects. As data volumes grow and as data models are enhanced to address new requirements, increased complexity leads to performance challenges. There are many books on this subject dealing with the intricacies of best practice data model design. In brief: Build star schemas – wherever possible, reshape data into fact a dimension tables with single key, one-to-many relationships from dimensions to fact. Enforce dimension key uniqueness – Just because a key value “should” be unique, there is no guarantee that it will be unless enforced at the data source. Perform grouping and duplicate reduction in the data source views or Power Query queries to guarantee uniqueness. Duplicate record count checks and other mechanisms can be applied to audit source data for integrity but do not allow the data model to violate these rules. Avoid bi-directional filters and unnecessary bridging tables – These data modelling patterns adversely affect performance. Calculated columns – Should be applied in Power Query and not in DAX calculated columns wherever possible. This maintains a consistent design pattern for maintainability. Annotate code – Use in-line comments and annotations in all code including SQL, M and DAX; to explain calculation logic and provide author and revision information. Hide all fields not used directly by users. These include: primary and foreign key columns, numeric columns used to create measures, and columns used to specify the sort order of other fields. Set to Do Not Summarize – Any non-hidden numeric columns that are not intended to roll-up or summarize values should be set to “Do Not Summarize” in the Modeling ribbon in Power BI Desktop. Columns set to summarize are indicated with a Sigma icon. Support Query Folding with Views Data source queries in Power BI are authored using the Power Query editor; which generates queries in the Microsoft Data Mashup “M” universal query language. A critical optimization technique employed by Power Query is called “Query Folding”; where the “M” query definition is translated into a native query that can be executed by the source database server. If query folding works, SQL Server executes queries and returns results to the Power BI service through the on-premises data gateway. If query folding can’t be used, all raw data is streamed through the gateway, held in memory in the service and then processed inefficiently, row-by-row. Query folding works reliably when queries authored in Power Query reference relational database tables or views. Avoid using SQL statements in Power Query and avoid using stored procedures as query data source objects. This can be a difficult transition for database professionals who are accustomed to writing SQL queries but Power Query can optimize its queries only when referencing tables or views. In rare cases to avoid extremely long-running queries that might time-out during data refresh, it may be necessary to materialize source table results in a table stored in the EDW or data mart database. Using parameters with an incremental refresh policy can also reduce or eliminate query time-out issues. Managing Dataset Size with Parameters Use Power Query parameters to filter large tables (typically only fact tables). The incremental refresh feature of Power BI Premium capacity will automatically generate partitions and only process new or partial data to avoid reloading large tables. Even if you don’t intend to implement incremental refresh policies; data refresh parameters allow the working dataset size to be reduced in the development environment and to deploy a smaller dataset file that can then be refreshed with a full load of data in the service. Recommendation:Design large tables with a parameterized range filter that conforms to the requirements supporting incremental refresh policies. When a dataset PBIX file approaches 400 MB in size, parameters should be used to filter large tables and reduce the working set of records for development. Dataset approaching 1 GB in size should have fact tables configured with incremental refresh policies and must be in a Premium capacity workspace. First, create two Date/Time type parameters named RangeStart and RangeEnd. Next create a date range filter in the early steps of each fact table on a Date/Time type column in the table. Incremental refresh policy requirements are detailed in this article: Measures & Numeric Columns Format all DAX code for easy readability. not only does it look better but will help avoid mistakes and support debugging and maintenance. As a guide, use DAXFormatter.com to “beautify” measures and follow the same code formatting pattern going forward. Numeric Measures All measures should return the correct numeric type values and every measure should be correctly formatted. This seemingly trivial subject is important and an issue in most informal and ill-managed projects. Requirements should be defined for data types, precision and formatting. Common items include: Measures are designed to summarize and total over any filter, slicer or grouping condition. Apply the appropriate roll-up or conditional level logic if it is not Percent and ratio measures should be fractional decimals. For example, “45 percent” is represented as a decimal value .45. When formatted, it appears as 45%, or 45.00% with two decimals for better precision. All whole or large number measures should include a thousand separator. Use the DAX DIVIDE function instead of the forward slash operator to handle blank and zero division errors. Text Measures Do not conditionally return text values to handle exceptions (like “not applicable”) in a measure that should return a numeric value. Create a separate text-type measure to support titles & labels, if needed. Implicit and Explicit Measures For simple and informal projects, Power BI supports implicit measures. Numeric columns in each table are automatically set to summarize data for ad hoc reporting. This behavior is both convenient and confusing for ad hoc report users. In formal projects, avoid using implicit measures for the following reasons. Instead, create a measure for any numeric column values that should be aggregated for reporting, and then hide all redundant numeric columns. Implicit measures don’t work in Excel pivot table and expressionsLike when working with cubes and connected SSAS models, only explicit measures are available in Excel report. Two choices confuse usersSince all data models will include explicit measures, written using DAX. It is confusing for users to understand when to use measures and when to use numeric columns. In conventional Excel pivot tables (which connects to Power BIas i f it were a multidimensional cube); measures and calculation objects are contained in a measure groups and all columns (whether numeric or any other data type) are dimension attributes. The introduction of PowerPivot, SSAS tabular and Power BI confuses most users with inconsistencies. Measures are FlexibleInevitably, some implicit measure will eventually be replaced with explicit measures when they are required to support some conditional logic or alternate aggregation behavior. Starting with explicit measures, avoids retraining users and introducing inconsistent behavior. Promote Self-service Reporting The true value of the Power BI platform is when business users can use the interactive reporting features of Power BI to get answers to business questions. This can be as simple as starting with an existing report and changing a slicer or filter; or adding a visual to a report page to see a selected set of data. Novice report author/users can easily be taught to drag-and-drop fields into a new report page to group and slice data from an IT & certified dataset. Starting with IT/business certified datasets published to the Power BI service, users with a range of interests and skills can modify or author reports using their web browser without altering governed data or protected reports. For user training and support purposes, consider educating users with skills in the following roles: Novice Report Users Navigate to an existing published app or report Understand how to dashboard tiles, report pages and visuals Use filters and slicers Use visual interactions to select and slice data Use report visual drill-through navigation Use Q&A to ask and answer data questions Casual Data Analysts Edit a report Change a report visual to a different type of visual Add a report page Add visuals and fields to a report page Add and use slicers to filter and select data Use Analyze in Excel to create a connected pivot table Use Excel to add fields, groups, totals, filters & slicers to a pivot table Set alerts & create a report subscription User bookmarks to create custom filter sets and navigation Intermediate Report Authors/Analysts Using a published certified dataset, create a new report Add pages, visuals, slicers and filters Add custom measures to a report using DAX Create drill-through actions & navigation Create a custom dashboard by pinning tiles Use SSRS/Paginated Reports for Operational Reporting Power BI is not a replacement for paginated, operational reporting. For static, multi-page, printable reports; use SQL Server Reporting Services (SSRS) instead of Power BI. SSRS (now called “Paginated Reports”) is integrated into the Power BI service with Premium capacity licensing, and can be integrated with interactive Power BI reports and Power BI data datasets. To a limited degree, some operational reports can be reproduced using Power BI reports and SSRS can be used, some a limited degree, to create interactive reports. Recommendation:Use the right tool for the job and consider the strengths and limitations of Power BI, SSRS/Paginated Reports and Excel to author and deliver reports for different audiences and use cases. If Users Need Excel, Give them Excel Financial users often misunderstand that they must adapt to visual dashboard reporting when they really need spreadsheet reports with reliable, current data. Power BI was borne from Excel and from SQL Server Analysis Services (traditionally “cubes”). Therefore, it can meet the needs of both self-service users and IT-managed enterprise BI solutions. In many ways, Excel are a perfect pair. If users need spreadsheets, don’t give them reports that look like spreadsheets. Give them connected spreadsheets; then train them to use Excel from a Power BI report and certified dataset. Analyze in Excel is a gem that is often not promoted by IT-based report authors; typically, because report requirements are usually specified before users lean about this capability. It really is a best-of-both-worlds solution; because it keeps governed data in a published dataset which allows users to access it via a secure live connection from Excel. When selecting Analyze in Excel from a published report or dataset in the service, Excel opens with a new connection (using an OCD file) directed at the published dataset. The Analyze in Excel ribbon (installed to Excel when downloaded) can also be used to create a new connection from Excel.By default, a pivot table is generated in a new worksheet; but advanced Excel users can use cube functions to place live Power BI dataset data anywhere in a workbook. Power BI and Excel–the Dynamic Duo Recommendation:Rather than exporting data from Power BI (and creating off-line copies of data), promote the use of live, connected reporting with Excel. Enable Analyze in Excel in selected reports and datasets; then train a limited audience of advanced users to promote its use. Establish best practices for each business use case. Generally information about this feature is available here: Implementing Security There are many good references and tutorials about data and report security that I won’t cover here but will summarize the recommended approaches. Data level security and report/content level security are separate concepts that should be coordinated. Report/Content-Level Security There are multiple options for sharing report content with users. Users cannot view a dashboard or report unless they have been granted access to a workspace or have been invited to share a report. If you have a Premium capacity Power BI tenant, you can package the content of a workspace [dashboard(s) and report(s)] as an app which can be shared with any users in your organization who do not need individual Power BI Pro licenses to read and view content. These users will not be able to edit or author additional reports. Users with Pro licenses assigned through the Office 365 Portal can have a workspace or report shared with them, which they can (unless otherwise restricted) edit, save, copy or share with others. Data-Level Security Once a user can access a report, they will see the results of all data in the underlying dataset for that report. Users with report level access can read all dataset data unless role-based and/or row-level security has been implemented. Role/RLS security work essentially the same way in SSAS and with data models developed with Power BI Desktop. Role-based Security Role-based Security is active when at least one role is defined for the dataset, filter expressions are added to the role and them user or group members are added to the role. Conditional filtering is performed for each table in the data model using a DAX expression. Row-level Security RLS is an extension of role-based security. In addition to creating one or more roles with filter expressions and adding role members, filtering may be conditional based on the authenticated user. The typical approach is to populate and maintain a mapping table of users and key values for each table in the data model that should have restricted access. All related data will be filtered. For example, if a user can only see data for a particular sales territory, they should only see sales totals for the territory. The user/mapping tables must be maintained either manually or through automation if the user/object permission information is available in an existing data source. Don’t over-engineer this. The user/mapping information can be managed in a spreadsheet and securely stored in SharePoint or OneDrive, or driven by a PowerApp or custom form. *This requirement is often missed in large projects. Recommendation:Add users to Office or Azure Active Directory groups and then use groups to share report content. If you are using role-based or row-level security, use groups to manage role membership. If using Premium, use workspace apps to share and distribute reports with read-only users and reduce Pro license requirements. Create a strategy for managing RLS user/entity mapping tables. Either manage manually or automate if the data is available. Certified & Shared Datasets To promote reuse and data governance, datasets can be used in reports residing in different workspaces (including a personal workspace). Datasets can be endorsed to designate those that are certified and promoted by data owners and the organization. The Power BI service includes a platform and workflow to manage promoted and certified datasets. When enabled, a dataset author can enable a dataset to be marked for promotion and certification. Designated users can test and optionally certify specific datasets that are known to be reliable and conform to organizationalgovernance standards. Dataset promotional and certification markings require the coordinated effort of the solution developer, Power BI tenant admin, data steward or business stakeholders. The actual process is simple; but the certification process should be defined and managed by solution business owners and service administrators. The following image illustrates theaforementioned process. Information about this capability: Enterprise Scale Options In many ways, Power BI has now surpassed the capabilities of SQL Server Analysis Services. Microsoft are investing in the enterprise capabilities of the Power BI platform, by enhancing Power BI Premium Capacity, adding Paginated Report and features to support massive scale specialized use cases. Consider the present and planned capabilities of the Power BI platform; before, choosing another data modeling tool such as SSAS. Resources: nbsp; Data Model Options for Power BI Solutions Project Preparation Checklist [_] Categorize the solution by identifying the author & user roles related to the project: [_] Author role: Business Data Analyst [_] Author role: Skilled Data Modeler, Analyst, Data Scientist [_] Author role: IT BI Developer [_] Users’ role: Report/Dashboard Consumer [_] Users’ role: Self-service Report Author [_] Users’ role: Advanced Data Analyst [_] Develop & Document Support & training plan for users [_] Identify the Solution Type for the project. This will guide other project management designs: [_] Design single PBIX file for small geoup, departmental project authored by one developer for a limited group of users [_] Design & deploy a separate dataset PBIX file – from report file(s) – when the dataset should be branded as a Certified dataset [_] Design separate dataset and report PBIX fiels for formal projects with more than one dataset & report developer, to coordinate work [_] Use SSAS/AAS as a data modeling option when those databases exist or where IT operations insist o management development and maintenance through integrated source control (e.g. Visual Studio Team Services & Azure DevOps) [_] Identify the Project Type & related Soluton Architecture: [_] Project type: Formal project [_] Project type: Informal project [_] Project type: Hybrid project [_] Architectural approach: Single PBIX [_] Architectural approach: Separate dataset and report PBIX [_] Architectural approach: Report PBIX connected to SSAS or AAS [_] Understand DirectQuery model trade-offs and special use cases. Avoid if possible. [_] Create storage locations and folder structure for Development file management: [_] Development file storage [_] Team member collaboration environment & processes [_] Folder synchronization [_] Define File naming standards [_] Decide on dataset and report names [_] Define the Version Control & Lifecycle Management [_] Postfix files with 3-part version number [_] Remove ver number from published files in QA and PROD [_] Create Version History table in Power Query [_] Increment version numbers in data model [_] Backup PBIT files for archive [_] Create measures: Last Refresh Date/Time [_] Create measure: Current Version [_] Add data model info page to report [_] Define your Release Management, DevOps & Automation strategy (if any – Might be OK to deploy files manually) (to automate or not to automate) [_] Decide on Workspace and App Management, workspace & app name, etc. [_] Create PROD workspace (omit PRD from name), assign dedicated capacity [_] Create QA workspace (post-fix name with QA), assign dedicated capacity [_] Create DEV workspace (post-fixo name with DEV), dedicated capacity not required [_] Assign Pro licenses to all developers, admins and report author users (QA?) [_] Assign membership and access to workspaces [_] Create fact date range filter parameters: RangeStart & RangeEnd [_] Filter large fact tables with range filters, consider incremental refresh policies if slow and/or over 800 MB compressed. [_] Design source queries (T-SQL?) to reshape source data into conformed dimension & fact tables [_] Create views in database for each dimension and fact [_] Enforce key uniqueness to remove all duplicate keys from all dimension tables [_] Query Date dim/lookup table at source if it exists [_] If not available, generate Date dim/lookup table in Power Query [_] Avoid bi-directional relationship wherever possible [_] Include code annotations & comment blocks in: views, M queries,measures [_] Hide all key columns and other columns not used directly by users [_] Use parameters to filter and reduce dataset PBIX size to ~400 MB or less [_] Create incremental refresh policies for dataset larger than ~500 MB [_] Use Premium capacity workspaces for datasets approaching 900 MB [_] Use Premium capacity workspace when you need to share reports with non-Pro licensed user [_] For non-Pro licensed user, convert QA (and then PRD) workspace to app and distribute the app to users via AD group membership [_] Create explicit measures for all aggregate-able column values [_] Hide all measure base numeric columns [_] Set all non-aggregate-able numeric column default summarization to “Do Not Summarize” [_] Format add whole numbers with thousand separators [_] Format all currency & decimal measures to defined standard (perhaps 2 decimal, thousand separator) [_] Format all ratios and percentage with appropriate decimal positions [_] Use the DIVIDE function for all division operations to avoid div by zero [_] Name all measures and user-visible fields with mixed-case, short, [_] friendly sentenc-like terms [_] Assign measures to fact table where they logically belong – or to an empty measure group table. [_] Measures that don’t have an obvious home (fact) table should be assigned a measure group table. [_] Precede measure group table names with “_” so they sort to the top of the list. (display folders are not ready for use in Power BI Desktop)As a team decide: [_] When to increment the revision/fix number [_] When to increment the minor version number [_] Who owns the migration to QA process? How is it implemented? [_] Who own migration from QA to PROD? How is it implemented? [_] How is user/stakeholder sign-off obtained? [_] How are data source requirements defined, presented & documented? [_] Who owns & maintains the requirements log? [_] How are measure & KPI calculation requirements documented & maintained? [_] How are the report design, layout & visualization requirements documented & maintained? [_] What is the organization & project styling & branding standard? [_] Does the organization have a standard brand image? Where is it documented? How owns that standard & can answer questions? [_] Should report be designed for mobile consumption (with mobile layouts)?