Contents

Guidance for Power BI

5/13/2020 • 2 minutes to read • Edit Online

Here you will find the guidance and recommended practices for Power BI. Guidance will continue to be updated and added to.

Data modeling

GUIDANCE	DESCRIPTION
Understand star schema and the importance for Power BI	Describes star schema design and its relevance to developing Power BI data models optimized for performance and usability.
Data reduction techniques for Import modeling	Describes different techniques to help reduce the data loaded into Import models.

DAX

GUIDANCE	DESCRIPTION
DAX: DIVIDE function vs divide operator (/)	Describes proper use of the DIVIDE function within DAX.

Dataflows

GUIDANCE	DESCRIPTION
Dataflows best practice	Describes best practices for designing dataflows in Power BI.

More questions? Try asking the Power BI Community

Optimization guide for Power BI

5/13/2020 • 6 minutes to read • Edit Online

This article provides guidance that enables developers and administrators to produce and maintain optimized Power BI solutions. You can optimize your solution at different architectural layers. Layers include:

- The data source(s)
- The data model
- Visualizations, including dashboards, Power BI reports, and Power BI paginated reports
- The environment, including capacities, data gateways, and the network

Optimizing the data model

The data model supports the entire visualization experience. Data models are either external-hosted or internal-hosted, and in Power BI they are referred to as *datasets*. It's important to understand your options, and to choose the appropriate dataset type for your solution. There are three dataset modes: Import, DirectQuery, and Composite. For more information, see Datasets in the Power BI service, and Dataset modes in the Power BI service.

For specific dataset mode guidance, see:

- Data reduction techniques for Import modeling
- DirectQuery model guidance in Power BI Desktop
- Composite model guidance in Power BI Desktop

Optimizing visualizations

Power BI visualizations can be dashboards, Power BI reports, or Power BI paginated reports. Each has different architectures, and so each has their own guidance.

Dashboards

It's important to understand that Power BI maintains a cache for your dashboard tiles—except live report tiles, and streaming tiles. For more information, see Data refresh in Power BI (Tile refresh). If your dataset enforces dynamic row-level security (RLS), be sure to understand performance implications as tiles will cache on a per-user basis.

When you pin live report tiles to a dashboard, they're not served from the query cache. Instead, they behave like reports, and make queries to back-end cores on the fly.

As the name suggests, retrieving the data from the cache provides better and more consistent performance than relying on the data source. One way to take advantage of this functionality is to have dashboards be the first landing page for your users. Pin often-used and highly requested visuals to the dashboards. In this way, dashboards become a valuable "first line of defense", which delivers consistent performance with less load on the capacity. Users can still click through to a report to analyze details.

For DirectQuery and live connection datasets, the cache is updated on a periodic basis by querying the data source. By default, it happens every hour, though you can configure a different frequency in the dataset settings. Each cache update will send queries to the underlying data source to update the cache. The number of queries that generate depends on the number of visuals pinned to dashboards that rely on the data source. Notice that if row-level security is enabled, queries are generated for each different security context. For example, consider there are two different roles that categorize your users, and they have two different views of the data. During query cache refresh, Power BI generates two sets of queries.

Power BI reports

There are several recommendations for optimizing Power BI report designs.

> **NOTE**
>
> When reports are based on a DirectQuery dataset, for additional report design optimizations, see DirectQuery model guidance in Power BI Desktop (Optimize report designs).

Apply the most restrictive filters

The more data that a visual needs to display, the slower that visual is to load. While this principle seems obvious, it's easy to forget. For example: suppose you have a large dataset. Atop of that dataset, you build a report with a table. End users use slicers on the page to get to the rows they want—typically, they're only interested in a few dozen rows.

A common mistake is to leave the default view of the table unfiltered—that is, all 100M+ rows. The data for these rows loads into memory and is uncompressed at every refresh. This processing creates huge demands for memory. The solution: use the "Top N" filter to reduce the max number of items that the table displays. You can set the max item to larger than what users would need, for example, 10,000. The result is the end-user experience doesn't change, but memory use drops greatly. And most importantly, performance improves.

A similar design approach to the above is suggested for every visual in your report. Ask yourself, is all the data in this visual needed? Are there ways to filter the amount of data shown in the visual with minimal impact to the end-user experience? Remember, tables in particular can be expensive.

Limit visuals on report pages

The above principle applies equally to the number of visuals added to a report page. It's highly recommended you limit the number of visuals on a particular report page to only what is necessary. Drillthrough pages and report page tooltips are great ways to provide additional details without jamming more visuals onto the page.

Evaluate custom visual performance

Be sure to put each custom visual through its paces to ensure high performance. Poorly optimized Power BI visuals can negatively affect the performance of the entire report.

Power BI paginated reports

Power BI paginated report designs can be optimized by applying best practice design to the report's data retrieval. For more information, see Data retrieval guidance for paginated reports.

Also, ensure your capacity has sufficient memory allocated to the paginated reports workload.

Optimizing the environment

You can optimize the Power BI environment by configuring capacity settings, sizing data gateways, and reducing network latency.

Capacity settings

When using dedicated capacities—available with Power BI Premium (P SKUs), or Power BI Embedded (A SKUs, A4-A6)—you can manage capacity settings. For more information, see Managing Premium capacities. For guidance on how to optimize your capacity, see Optimizing Premium capacities.

Gateway sizing

A gateway is required whenever Power BI must access data that isn't accessible directly over the Internet. You can install the on-premises data gateway on a server on-premises, or VM-hosted Infrastructure-as-a-Service (IaaS).

To understand gateway workloads and sizing recommendations, see On-premises data gateway sizing.

Network latency

Network latency can impact report performance by increasing the time required for requests to reach the Power BI

service, and for responses to be delivered. Tenants in Power BI are assigned to a specific region.

> **TIP**
>
> To determine where your tenant is located, see Where is my Power BI tenant located?

When users from a tenant access the Power BI service, their requests always route to this region. As requests reach the Power BI service, the service may then send additional requests—for example, to the underlying data source, or a data gateway—which are also subject to network latency.

Tools such as Azure Speed Test provide an indication of network latency between the client and the Azure region. In general, to minimize the impact of network latency, strive to keep data sources, gateways, and your Power BI cluster as close as possible. Preferably, they reside within the same region. If network latency is an issue, try locating gateways and data sources closer to your Power BI cluster by placing them inside cloud-hosted virtual machines.

Monitoring performance

You can monitor performance to identify bottlenecks. Slow queries—or report visuals—should be a focal point of continued optimization. Monitoring can be done at design time in Power BI Desktop, or on production workloads in Power BI Premium capacities. For more information, see Monitoring report performance in Power BI.

Next steps

For more information about this article, check out the following resources:

- Power BI guidance
- Monitoring report performance
- Whitepaper: Planning a Power BI Enterprise Deployment
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Query folding guidance in Power BI Desktop

5/11/2020 • 3 minutes to read • Edit Online

This article targets data modelers developing models in Power BI Desktop. It provides best practice guidance on when—and how—you can achieve Power Query query folding.

Query folding is the ability for a Power Query query to generate a single query statement that retrieves and transforms source data. For more information, see Power Query query folding.

Guidance

Query folding guidance differs based on the model mode.

For a **DirectQuery** or **Dual** storage mode table, the Power Query query must achieve query folding.

For an **Import** table, it may be possible to achieve query folding. When the query is based on a relational source—and if a single SELECT statement can be constructed—you achieve *best data refresh performance* by ensuring that query folding occurs. If the Power Query mashup engine is still required to process transformations, you should strive to minimize the work it needs to do, especially for large datasets.

The following bulleted-list provides specific guidance.

- **Delegate as much processing to the data source as possible**: When all steps of a Power Query query can't be folded, discover the step that prevents query folding. When possible, move later steps earlier in sequence so they may be factored into the query folding. Note the Power Query mashup engine may be smart enough to reorder your query steps when it generates the source query.

  For a relational data source, if the step that prevents query folding could be achieved in a single SELECT statement—or within the procedural logic of a stored procedure—consider using a native SQL query, as described next.

- **Use a native SQL query**: When a Power Query query retrieves data from a relational source, it's possible for some sources to use a native SQL query. The query can in fact be any valid statement, including a stored procedure execution. If the statement produces multiple result sets, only the first will be returned. Parameters can be declared in the statement, and we recommend that you use the Value.NativeQuery M function. This function was designed to safely and conveniently pass parameter values. It's important to understand that the Power Query mashup engine can't fold later query steps, and so you should include all—or as much—transformation logic in the native query statement.

  There are two important considerations you need to bear in mind when using native SQL queries:

  - For a DirectQuery model table, the query must be a SELECT statement, and it can't use Common Table Expressions (CTEs) or a stored procedure.
  - Incremental refresh can't use a native SQL query. So, it would force the Power Query mashup engine to retrieve all source rows, and then apply filters to determine incremental changes.

  > **IMPORTANT**
  > A native SQL query can potentially do more than retrieve data. Any valid statement can be executed (and possibly multiple times), including one that modifies or deletes data. It's important that you apply the principle of least privilege to ensure that the account used to access the database has only read permission on required data.

- **Prepare and transformation data in the source**: When you identify that certain Power Query query

steps can't be folded, it may be possible to apply the transformations in the data source. The transformations could be achieved by writing a database view that logically transforms source data. Or, by physically preparing and materializing data, in advance of Power BI querying it. A relational data warehouse is an excellent example of prepared data, usually consisting of pre-integrated sources of organizational data.

Next steps

For more information about this article, check out the following resources:

- Power Query Query folding concept article
- Incremental refresh in Power BI Premium
- Questions? Try asking the Power BI Community

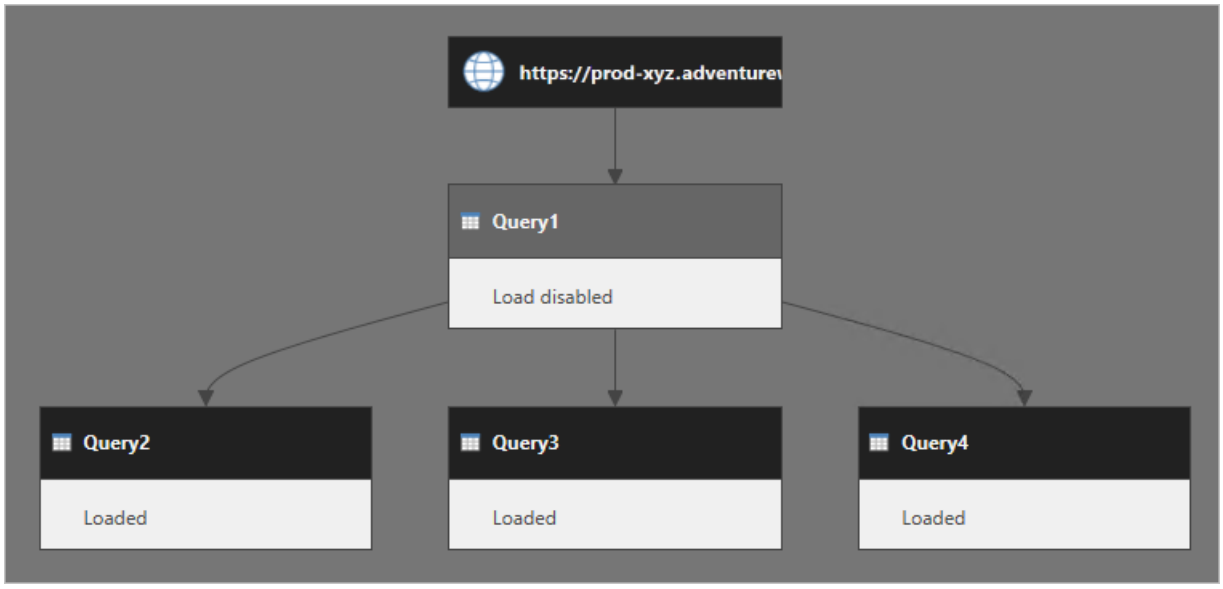
Referencing Power Query queries

5/13/2020 • 2 minutes to read • Edit Online

This article targets you as a data modeler working with Power BI Desktop. It provides you with guidance when defining Power Query queries that reference other queries.

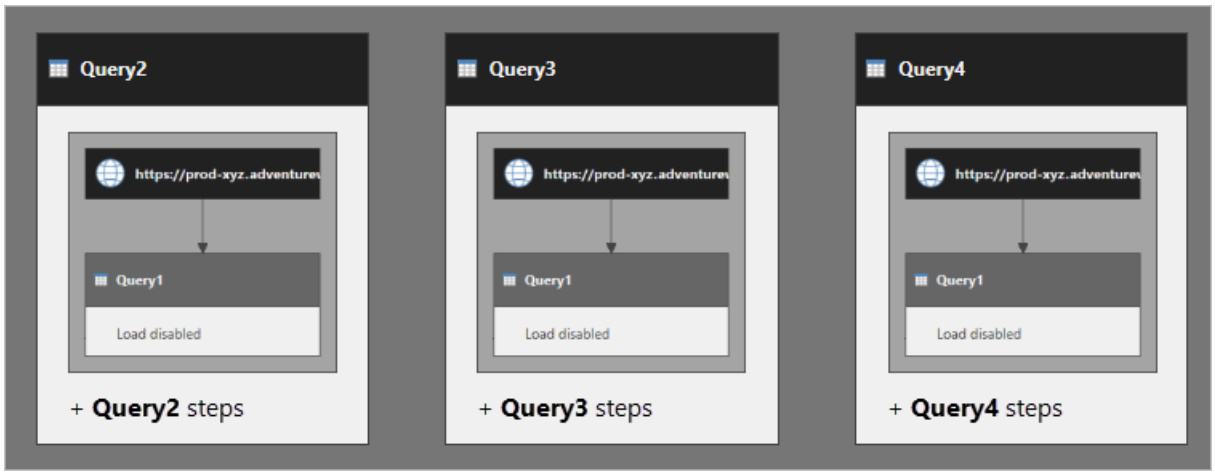
Let's be clear about what this means: *When a query references a second query, it's as though the steps in the second query are combined with, and run before, the steps in the first query.*

Consider several queries: **Query1** sources data from a web service, and its load is disabled. **Query2**, **Query3**, and **Query4** all reference **Query1**, and their outputs are loaded to the data model.



When the data model is refreshed, it's often assumed that Power Query retrieves the **Query1** result, and that it's reused by referenced queries. This thinking is incorrect. In fact, Power Query executes **Query2**, **Query3**, and **Query4** separately.

You can think that **Query2** has the **Query1** steps embedded into it. It's the case for **Query3** and **Query4**, too. The following diagram presents a clearer picture of how the queries are executed.



Query1 is executed three times. The multiple executions can result in slow data refresh, and negatively impact on the data source.

The use of the Table.Buffer function in **Query1** won't eliminate the additional data retrieval. This function buffers a

table to memory. And, the buffered table can only be used within the same query execution. So, in the example, if **Query1** is buffered when **Query2** is executed, the buffered data couldn't be used when **Query3** and **Query4** are executed. They'll themselves buffer the data twice more. (This result could in fact compound the negative performance, because the table will be buffered by each referencing query.)

> **NOTE**
> Power Query caching architecture is complex, and it's not the focus of this article. Power Query can cache data retrieved from a data source. However, when it executes a query, it may retrieve the data from the data source more than once.

Recommendations

Generally, we recommend you reference queries to avoid the duplication of logic across your queries. However, as described in this article, this design approach can contribute to slow data refreshes, and overburden data sources.

We recommend you create a dataflow instead. Using a dataflow can improve data refresh time, and reduce impact on your data sources.

You can design the dataflow to encapsulate the source data and transformations. As the dataflow is a persisted store of data in the Power BI service, its data retrieval is fast. So, even when referencing queries result in multiple requests for the dataflow, data refresh times can be improved.

In the example, if **Query1** is redesigned as a dataflow entity, **Query2**, **Query3**, and **Query4** can use it as a data source. With this design, the entity sourced by **Query1** will be evaluated only once.

Next steps

For more information related to this article, check out the following resources:

- Self-service data prep in Power BI
- Creating and using dataflows in Power BI
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Disable Power Query background refresh

1/2/2020 • 2 minutes to read • Edit Online

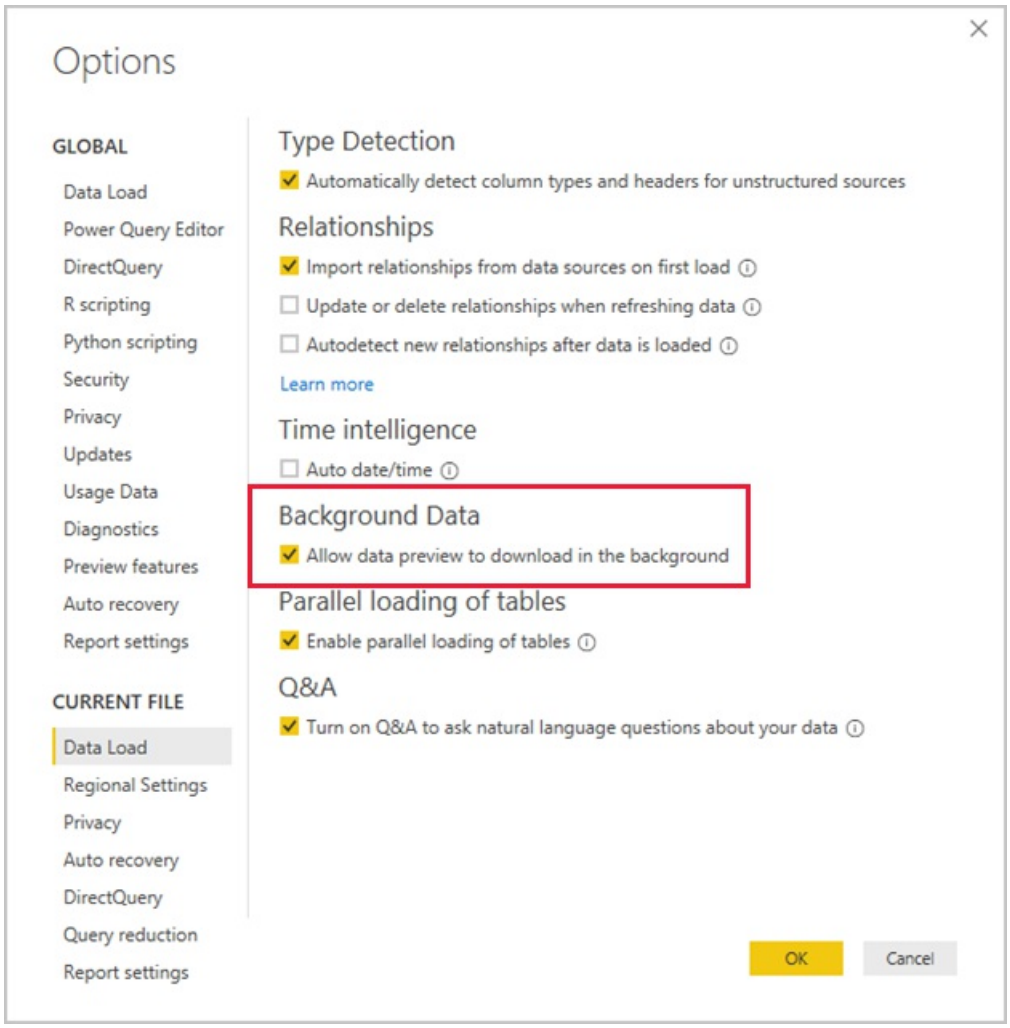
This article targets Import data modelers working with Power BI Desktop.

By default, when Power Query imports data, it also caches up to 1000 rows of preview data for each query. Preview data helps to present you with a quick preview of source data, and of transformation results for each step of your queries. It's stored separately on-disk and not inside the Power BI Desktop file.

However, when your Power BI Desktop file contains many queries, retrieving and storing preview data can extend the time it takes to complete a refresh.

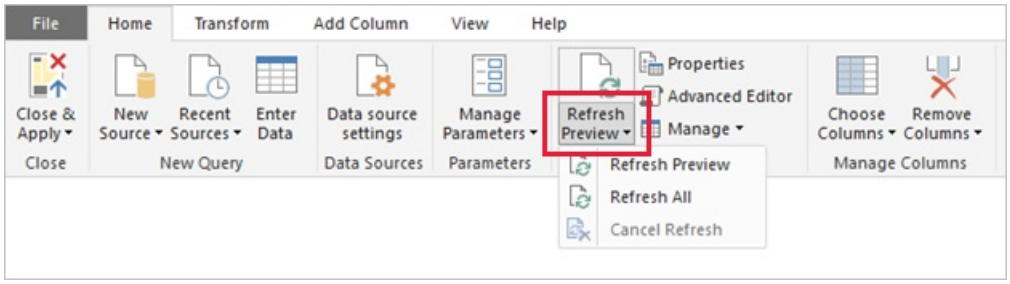
Recommendation

You'll achieve a faster refresh by setting the Power BI Desktop file to update the preview cache *in the background*. In Power BI Desktop, you enable it by selecting *File* > *Options and settings* > *Options*, and then selecting the *Data Load* page. You can then turn on the **Allow data preview to download in the background** option. Note this option can only be set for the current file.



Enabling background refresh can result in preview data becoming out of date. If it occurs, the Power Query Editor will notify you with the following warning:

It's always possible to update the preview cache. You can update it for a single query, or for all queries by using the **Refresh Preview** command. You'll find it on the **Home** ribbon of the Power Query Editor window.



Next steps

For more information related to this article, check out the following resources:

- Power Query Documentation
- Questions? Try asking the Power BI Community

Dataflows best practice

5/13/2020 • 3 minutes to read • Edit Online

This article provides best practices for designing dataflows in Power BI. You can use this guidance to learn how to create dataflows, apply these practices to your own dataflows.

> **NOTE**
> The recommendations made in this article are guidelines. For each best practice in this article, there may be legitimate reasons to deviate from this guidance.

Split ingestion and transformation to use the enhanced compute engine

When creating dataflows, you may be tempted to create a single dataflow with all entities, transformations, joins, and enhancements in one place. For smaller datasets a single dataflow may be effective. But when dealing with larger data volumes, performing joins or certain transformations can sometimes experience throttle or memory limits. To address those issues, an enhanced engine has been released for Power BI Premium users that scales to much larger data volumes. The enhanced compute engine works against linked or computed entities only, so creating a separate dataflow for ingestion and a linked dataflow to perform all the complex merges and transformations can benefit from the enhanced engine.

Splitting dataflows is also beneficial for diagnosing and debugging refresh issues, especially when working with sources that have throttling limits.

Perform actions that can use the enhanced compute engine

Ensure you make use of the enhanced compute engine by ensuring you perform joins and filter transformations first in a computed entity before performing other types of transformations.

Split dataflows when connecting to SharePoint

When working with SharePoint and connecting to multiple files, you may notice sporadic failures. SharePoint throttles requests to ensure it remains reliable and responsive. As a consequence, when connecting to Excel files from SharePoint you may be inclined to download all files in a single dataflow. If you're downloading many files, more than 20, create two dataflows or more to split the refresh load, and overcome SharePoint throttling.

Avoid scheduling refresh for linked entities inside the same workspace

If you're regularly being locked out of your dataflows that contain linked entities, it may be a result of corresponding, dependent dataflow inside the same workspace are locked during dataflow refresh. Such locking provides transactional accuracy and ensures both dataflows successfully refresh, but it can block you from editing.

If you set up a separate schedule for the linked dataflow, dataflows can refresh unnecessarily and block you from editing the dataflow. There are two recommendations to avoid this issue:

- Avoid setting a refresh schedule for a linked dataflow in the same workspace as the source dataflow
- If you want to configure a refresh schedule separately, and want to avoid the locking behavior, separate the dataflow in a separate workspace.

Create a separate workspace for ingestion, transformation

To provide access to underlying dataflow data for many users, without allowing access to the raw data of the underlying source system, create a separate workspace that has all the data you need to share, and assign permissions. Individual dataflow permissions are not currently supported.

Ensure capacity is in the same region

Dataflows do not currently support multi-geo regions. The Premium capacity must be in the same region as your Power BI tenant.

Separate on-premises sources from cloud sources

In addition to the previously described best practices, you can create a separate dataflow for each type of source, such as on-premises, cloud, SQL Server, Spark, Dynamics, and so on. It's strongly recommended to split your dataflow by data source type. Such separation by source type facilitates quick troubleshooting, and avoids internal limits when refreshing your dataflows.

Separate dataflows into a separate capacity

If you are experiencing throttling limits, or seeing regular failures for your dataflows due to memory limits on your capacity, consider separating your dataflows, or scaling up your Premium capacity for additional memory.

Next Steps

This article provided information about best practices for dataflows. For more information about dataflows, the following articles may be helpful:

- Self-service data prep with dataflows
- Create and use dataflows in Power BI
- Using computed entities on Power BI Premium
- Using dataflows with on-premises data sources

For information about CDM development and tutorial resources, see the following:

- Common Data Model - overview
- CDM folders
- CDM model file definition

For more information about Power Query and scheduled refresh, you can read these articles:

- Query overview in Power BI Desktop
- Configuring scheduled refresh

Understand star schema and the importance for Power BI

5/13/2020 • 17 minutes to read • Edit Online

This article targets Power BI Desktop data modelers. It describes star schema design and its relevance to developing Power BI data models optimized for performance and usability.

This article isn't intended to provide a complete discussion on star schema design. For more details, refer directly to published content, like **The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling** (3rd edition, 2013) by Ralph Kimball et al.

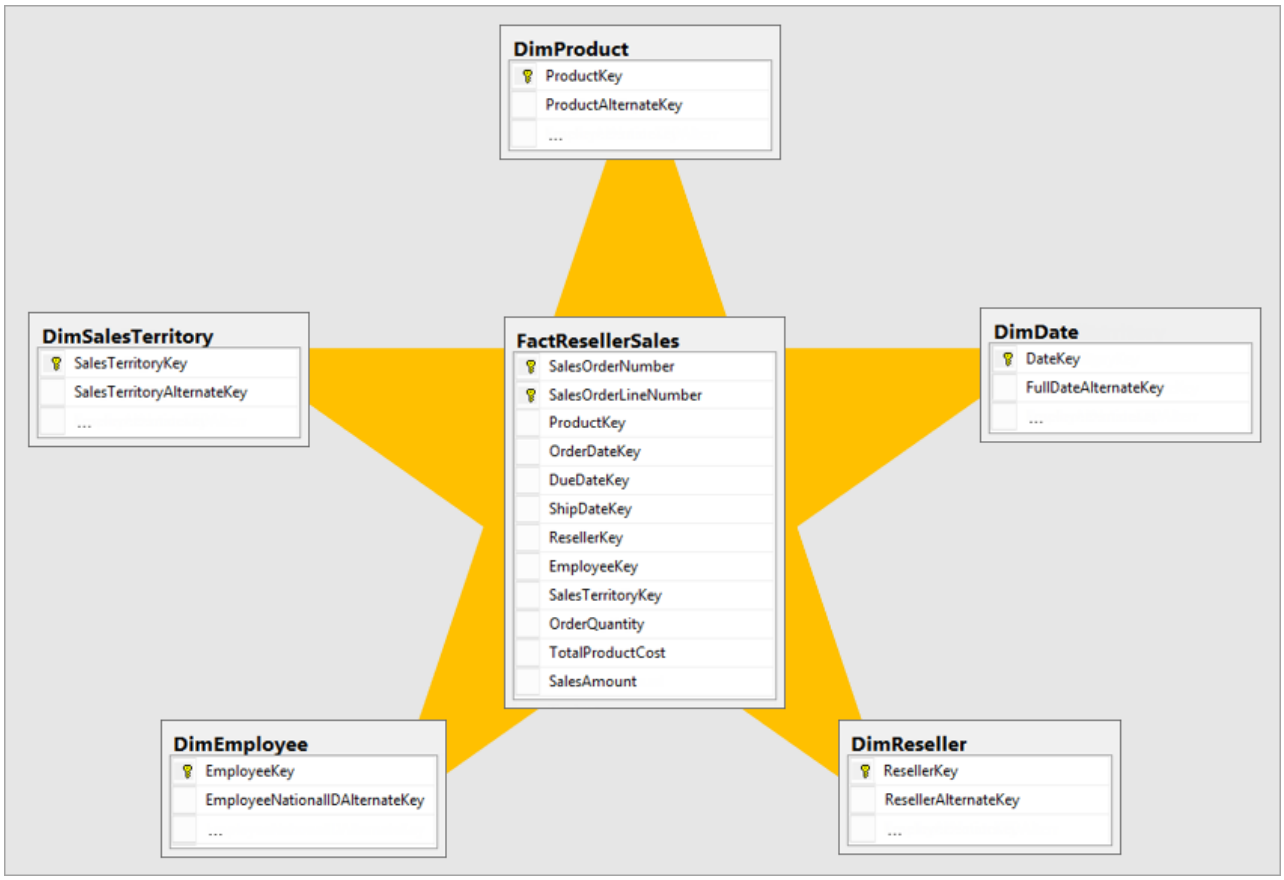
Star schema overview

Star schema is a mature modeling approach widely adopted by relational data warehouses. It requires modelers to classify their model tables as either *dimension* or *fact*.

Dimension tables describe business entities—the *things* you model. Entities can include products, people, places, and concepts including time itself. The most consistent table you'll find in a star schema is a date dimension table. A dimension table contains a key column (or columns) that acts as a unique identifier, and descriptive columns.

Fact tables store observations or events, and can be sales orders, stock balances, exchange rates, temperatures, etc. A fact table contains dimension key columns that relate to dimension tables, and numeric measure columns. The dimension key columns determine the *dimensionality* of a fact table, while the dimension key values determine the *granularity* of a fact table. For example, consider a fact table designed to store sale targets that has two dimension key columns **Date** and **ProductKey**. It's easy to understand that the table has two dimensions. The granularity, however, can't be determined without considering the dimension key values. In this example, consider that the values stored in the **Date** column are the first day of each month. In this case, the granularity is at month-product level.

Generally, dimension tables contain a relatively small number of rows. Fact tables, on the other hand, can contain a very large number of rows and continue to grow over time.



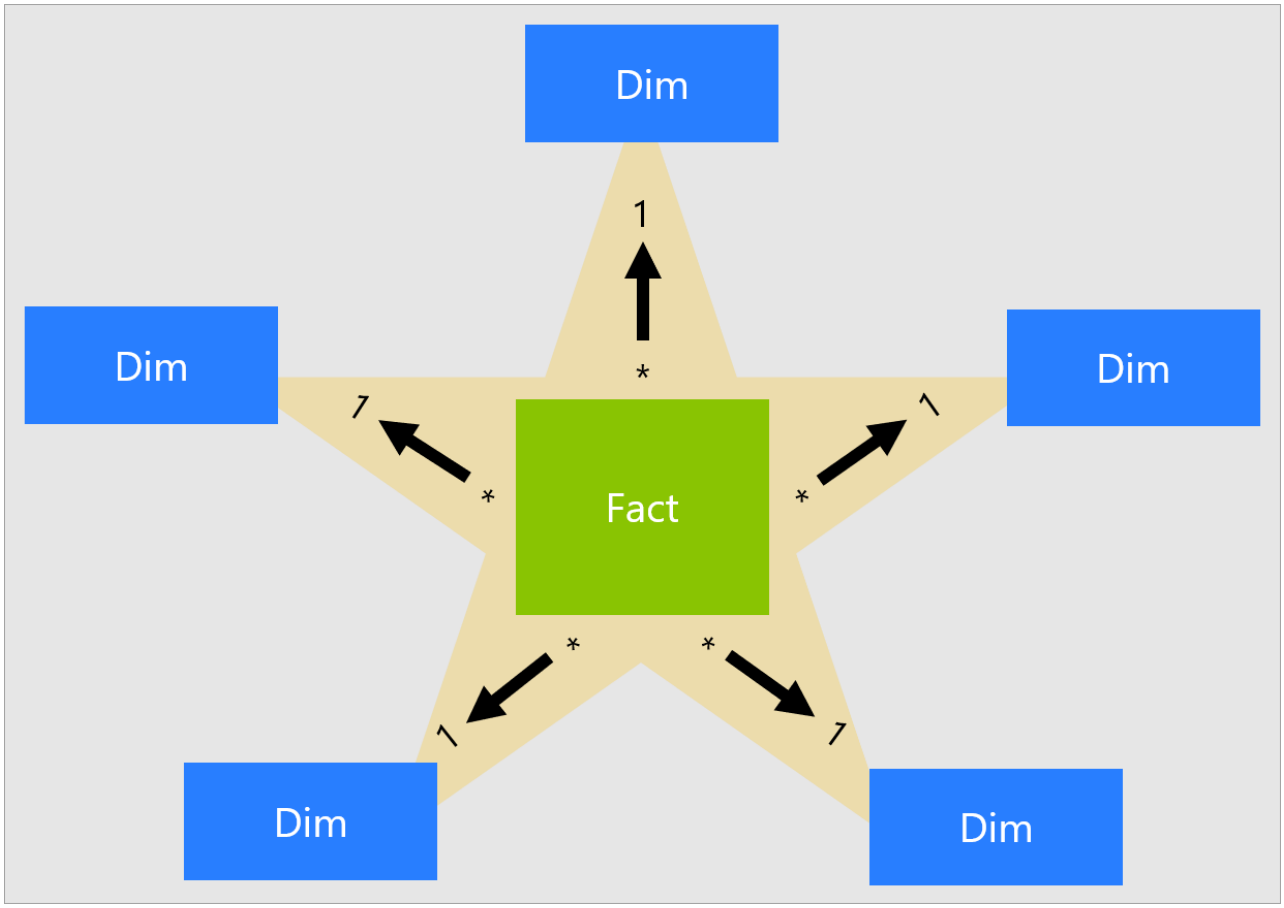
Star schema relevance to Power BI models

Star schema design and many related concepts introduced in this article are highly relevant to developing Power BI models that are optimized for performance and usability.

Consider that each Power BI report visual generates a query that is sent to the Power BI model (which the Power BI service calls a dataset). These queries are used to filter, group, and summarize model data. A well-designed model, then, is one that provides tables for filtering and grouping, and tables for summarizing. This design fits well with star schema principles:

- Dimension tables support *filtering* and *grouping*
- Fact tables support *summarization*

There's no table property that modelers set to configure the table type as dimension or fact. It's in fact determined by the model relationships. A model relationship establishes a filter propagation path between two tables, and it's the **Cardinality** property of the relationship that determines the table type. A common relationship cardinality is *one-to-many* or its inverse *many-to-one*. The "one" side is always a dimension-type table while the "many" side is always a fact-type table. For more information about relationships, see Model relationships in Power BI Desktop.



A well-structured model design should include tables that are either dimension-type tables or fact-type tables. Avoid mixing the two types together for a single table. We also recommend that you should strive to deliver the right number of tables with the right relationships in place. It's also important that fact-type tables always load data at a consistent grain.

Lastly, it's important to understand that optimal model design is part science and part art. Sometimes you can break with good guidance when it makes sense to do so.

There are many additional concepts related to star schema design that can be applied to a Power BI model. These concepts include:

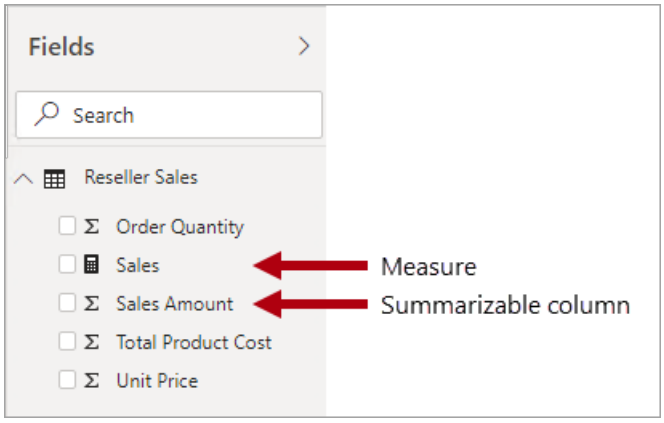
- Measures
- Surrogate keys
- Snowflake dimensions
- Role-playing dimensions
- Slowly changing dimensions
- Junk dimensions
- Degenerate dimensions
- Factless fact tables

Measures

In star schema design, a **measure** is a fact table column that stores values to be summarized.

In a Power BI model, a **measure** has a different—but similar—definition. It's a formula written in Data Analysis Expressions (DAX) that achieves summarization. Measure expressions often leverage DAX aggregation functions like SUM, MIN, MAX, AVERAGE, etc. to produce a scalar value result at query time (values are never stored in the model). Measure expression can range from simple column aggregations to more sophisticated formulas that override filter context and/or relationship propagation. For more information, read the DAX Basics in Power BI Desktop article.

It's important to understand that Power BI models support a second method for achieving summarization. Any column—and typically numeric columns—can be summarized by a report visual or Q&A. These columns are referred to as *implicit measures*. They offer a convenience for you as a model developer, as in many instances you do not need to create measures. For example, the Adventure Works reseller sales **Sales Amount** column could be summarized in numerous ways (sum, count, average, median, min, max, etc.), without the need to create a measure for each possible aggregation type.



However, there are three compelling reasons for you to create measures, even for simple column-level summarizations:

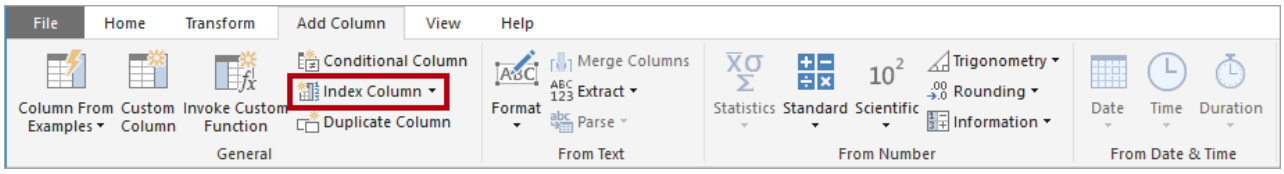
- When you know your report authors will query the model by using Multidimensional Expressions (MDX), the model must include *explicit measures*. Explicit measures are defined by using DAX. This design approach is highly relevant when a Power BI dataset is queried by using MDX, because MDX can't achieve summarization of column values. Notably, MDX will be used when performing Analyze in Excel, because PivotTables issue MDX queries.
- When you know your report authors will create Power BI paginated reports using the MDX query designer, the model must include explicit measures. Only the MDX query designer supports server aggregates. So, if report authors need to have measures evaluated by Power BI (instead of by the paginated report engine), they must use the MDX query designer.
- When you need to ensure that your report authors can only summarize columns in specific ways. For example, the reseller sales **Unit Price** column (which represents a per unit rate) can be summarized, but only by using specific aggregation functions. It should never be summed, but it's appropriate to summarize by using other aggregation functions like min, max, average, etc. In this instance, the modeler can hide the **Unit Price** column, and create measures for all appropriate aggregation functions.

This design approach works well for reports authored in the Power BI service and for Q&A. However, Power BI Desktop live connections allow report authors to show hidden fields in the **Fields** pane, which can result in circumventing this design approach.

Surrogate keys

A **surrogate key** is a unique identifier that you add to a table to support star schema modeling. By definition, it's not defined or stored in the source data. Commonly, surrogate keys are added to relational data warehouse dimension tables to provide a unique identifier for each dimension table row.

Power BI model relationships are based on a single unique column in one table, which propagates filters to a single column in a different table. When a dimension-type table in your model doesn't include a single unique column, you must add a unique identifier to become the "one" side of a relationship. In Power BI Desktop, you can easily achieve this requirement by creating a Power Query index column.

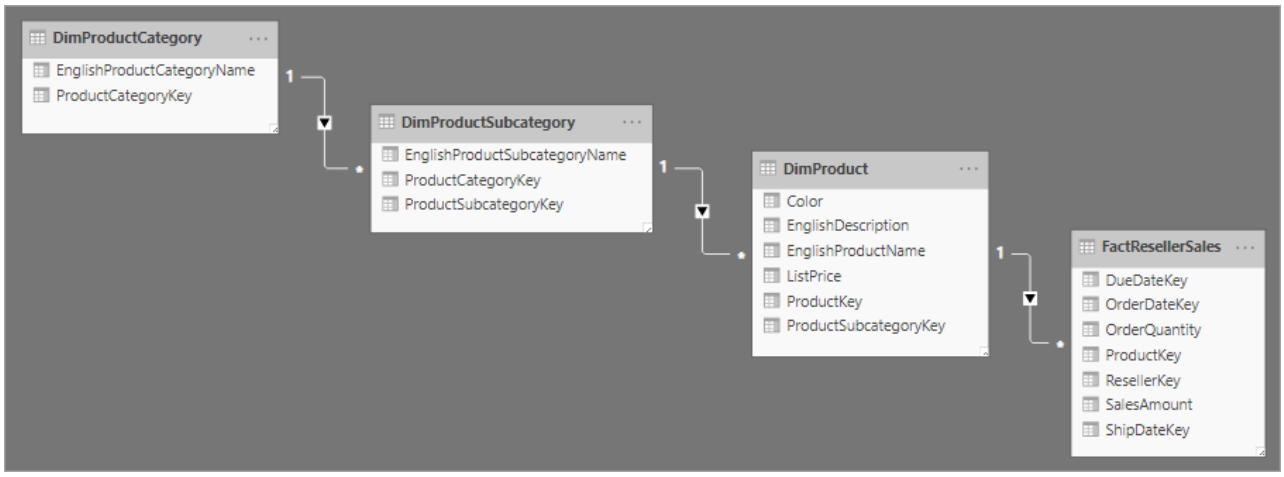


You must merge this query with the "many"-side query so that you can add the index column to it also. When you load these queries to the model, you can then create a one-to-many relationship between the model tables.

Snowflake dimensions

A **snowflake dimension** is a set of normalized tables for a single business entity. For example, Adventure Works classifies products by category and subcategory. Categories are assigned to subcategories, and products are in turn assigned to subcategories. In the Adventure Works relational data warehouse, the product dimension is normalized and stored in three related tables: **DimProductCategory**, **DimProductSubcategory**, and **DimProduct**.

If you use your imagination, you can picture the normalized tables positioned outwards from the fact table, forming a snowflake design.

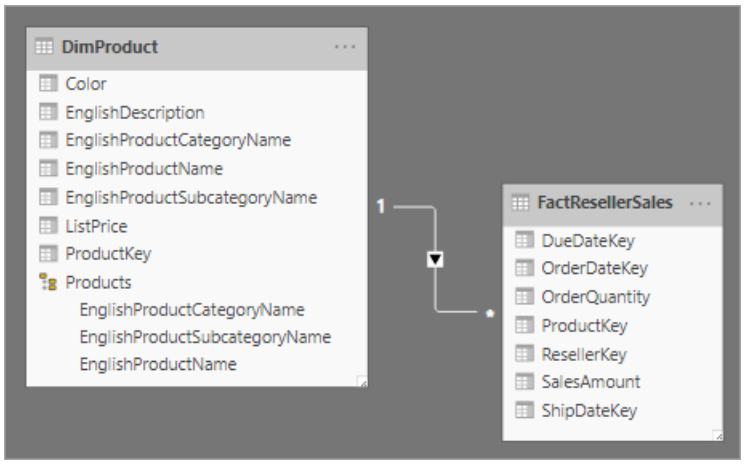


In Power BI Desktop, you can choose to mimic a snowflake dimension design (perhaps because your source data does) or integrate (denormalize) the source tables into a single model table. Generally, the benefits of a single model table outweigh the benefits of multiple model tables. The most optimal decision can depend on the volumes of data and the usability requirements for the model.

When you choose to mimic a snowflake dimension design:

- Power BI loads more tables, which is less efficient from storage and performance perspectives. These tables must include columns to support model relationships, and it can result in a larger model size.
- Longer relationship filter propagation chains will need to be traversed, which will likely be less efficient than filters applied to a single table.
- The **Fields** pane presents more model tables to report authors, which can result in a less intuitive experience, especially when snowflake dimension tables contain just one or two columns.
- It's not possible to create a hierarchy that spans the tables.

When you choose to integrate into a single model table, you can also define a hierarchy that encompasses the highest and lowest grain of the dimension. Possibly, the storage of redundant denormalized data can result in increased model storage size, particularly for very large dimension tables.



Slowly changing dimensions

A **slowly changing dimension** (SCD) is one that appropriately manages change of dimension members over time. It applies when business entity values change over time, and in an ad hoc manner. A good example of a *slowly* changing dimension is a customer dimension, specifically its contact detail columns like email address and phone number. In contrast, some dimensions are considered to be *rapidly* changing when a dimension attribute changes often, like a stock's market price. The common design approach in these instances is to store rapidly changing attribute values in a fact table measure.

Star schema design theory refers to two common SCD types: Type 1 and Type 2. A dimension-type table could be Type 1 or Type 2, or support both types simultaneously for different columns.

Type 1 SCD

A **Type 1 SCD** always reflects the latest values, and when changes in source data are detected, the dimension table data is overwritten. This design approach is common for columns that store supplementary values, like the email address or phone number of a customer. When a customer email address or phone number changes, the dimension table updates the customer row with the new values. It's as if the customer always had this contact information.

A non-incremental refresh of a Power BI model dimension-type table achieves the result of a Type 1 SCD. It refreshes the table data to ensure the latest values are loaded.

Type 2 SCD

A **Type 2 SCD** supports versioning of dimension members. If the source system doesn't store versions, then it's usually the data warehouse load process that detects changes, and appropriately manages the change in a dimension table. In this case, the dimension table must use a surrogate key to provide a unique reference to a *version* of the dimension member. It also includes columns that define the date range validity of the version (for example, **StartDate** and **EndDate**) and possibly a flag column (for example, **IsCurrent**) to easily filter by current dimension members.

For example, Adventure Works assigns salespeople to a sales region. When a salesperson relocates region, a new version of the salesperson must be created to ensure that historical facts remain associated with the former region. To support accurate historic analysis of sales by salesperson, the dimension table must store versions of salespeople and their associated region(s). The table should also include start and end date values to define the time validity. Current versions may define an empty end date (or 12/31/9999), which indicates that the row is the current version. The table must also define a surrogate key because the business key (in this instance, employee ID) won't be unique.

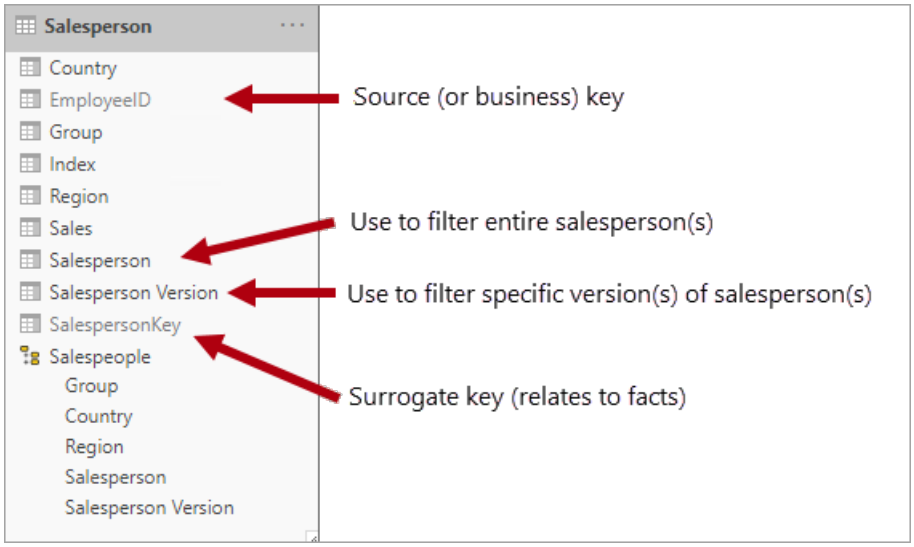
It's important to understand that when the source data doesn't store versions, you must use an intermediate system (like a data warehouse) to detect and store changes. The table load process must preserve existing data and detect changes. When a change is detected, the table load process must expire the current version. It records these changes by updating the **EndDate** value and inserting a new version with the **StartDate** value commencing from

the previous **EndDate** value. Also, related facts must use a time-based lookup to retrieve the dimension key value relevant to the fact date. A Power BI model using Power Query can't produce this result. It can, however, load data from a pre-loaded SCD Type 2 dimension table.

The Power BI model should support querying historical data for a member, regardless of change, and for a version of the member, which represents a particular state of the member in time. In the context of Adventure Works, this design enables you to query the salesperson regardless of assigned sales region, or for a particular version of the salesperson.

To achieve this requirement, the Power BI model dimension-type table must include a column for filtering the salesperson, and a different column for filtering a specific version of the salesperson. It's important that the version column provides a non-ambiguous description, like "Michael Blythe (12/15/2008-06/26/2019)" or "Michael Blythe (current)". It's also important to educate report authors and consumers about the basics of SCD Type 2, and how to achieve appropriate report designs by applying correct filters.

It's also a good design practice to include a hierarchy that allows visuals to drill down to the version level.



Group	Country	Region	Salesperson	Salesperson Version	Sales
North America	United States	Northeast	David Campbell	David Campbell (12/15/2008-06/26/2019)	$52,307.11
				David Campbell (06/27/2019-Current)	$119,807.56
				Total	**$172,114.67**
			Pamela Ansman-Wolfe	Pamela Ansman-Wolfe (03/03/2007-Current)	$222,087.01
				Total	**$222,087.01**
			Tete Mensa-Annan	Tete Mensa-Annan (01/09/2005-Current)	$23,098.4
				Total	**$23,098.4**
			Total		**$417,300.08**
		Total			**$417,300.08**
	Total				**$417,300.08**
Total					**$417,300.08**

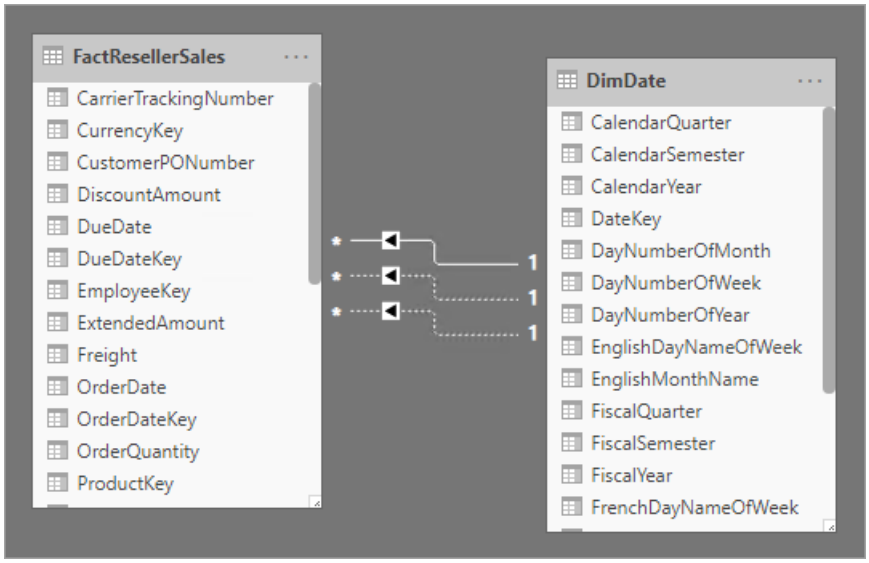
Role-playing dimensions

A **role-playing dimension** is a dimension that can filter related facts differently. For example, at Adventure Works, the date dimension table has three relationships to the reseller sales facts. The same dimension table can be used to filter the facts by order date, ship date, or delivery date.

In a data warehouse, the accepted design approach is to define a single date dimension table. At query time, the "role" of the date dimension is established by which fact column you use to join the tables. For example, when you analyze sales by order date, the table join relates to the reseller sales order date column.

In a Power BI model, this design can be imitated by creating multiple relationships between two tables. In the Adventure Works example, the date and reseller sales tables would have three relationships. While this design is possible, it's important to understand that there can only be one active relationship between two Power BI model tables. All remaining relationships must be set to inactive. Having a single active relationship means there is a

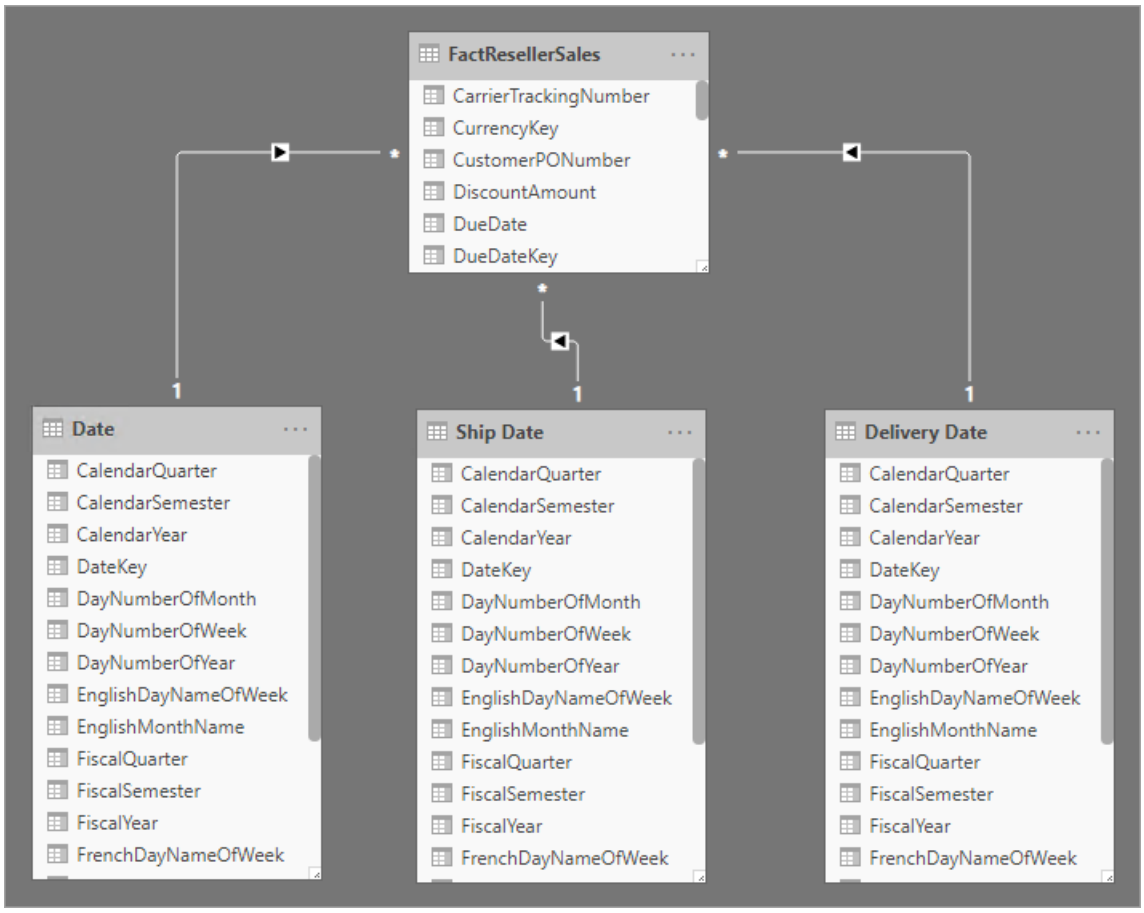
default filter propagation from date to reseller sales. In this instance, the active relationship is set to the most common filter that is used by reports, which at Adventure Works is the order date relationship.



The only way to use an inactive relationship is to define a DAX expression that uses the USERELATIONSHIP function. In our example, the model developer must create measures to enable analysis of reseller sales by ship date and delivery date. This work can be tedious, especially when the reseller table defines many measures. It also creates **Fields** pane clutter, with an overabundance of measures. There are other limitations, too:

- When report authors rely on summarizing columns, rather than defining measures, they can't achieve summarization for the inactive relationships without writing a report-level measure. Report-level measures can only be defined when authoring reports in Power BI Desktop.
- With only one active relationship path between date and reseller sales, it's not possible to simultaneously filter reseller sales by different types of dates. For example, you can't produce a visual that plots order date sales by shipped sales.

To overcome these limitations, a common Power BI modeling technique is to create a dimension-type table for each role-playing instance. You typically create the additional dimension tables as calculated tables, using DAX. Using calculated tables, the model can contain a **Date** table, a **Ship Date** table and a **Delivery Date** table, each with a single and active relationship to their respective reseller sales table columns.



This design approach doesn't require you to define multiple measures for different date roles, and it allows simultaneous filtering by different date roles. A minor price to pay, however, with this design approach is that there will be duplication of the date dimension table resulting in an increased model storage size. As dimension-type tables typically store fewer rows relative to fact-type tables, it is rarely a concern.

Observe the following good design practices when you create model dimension-type tables for each role:

- Ensure that the column names are self-describing. While it's possible to have a **Year** column in all date tables (column names are unique within their table), it's not self-describing by default visual titles. Consider renaming columns in each dimension role table, so that the **Ship Date** table has a year column named **Ship Year**, etc.
- When relevant, ensure that table descriptions provide feedback to report authors (through **Fields** pane tooltips) about how filter propagation is configured. This clarity is important when the model contains a generically named table, like **Date**, which is used to filter many fact-type tables. In the case that this table has, for example, an active relationship to the reseller sales order date column, consider providing a table description like "Filters reseller sales by order date".

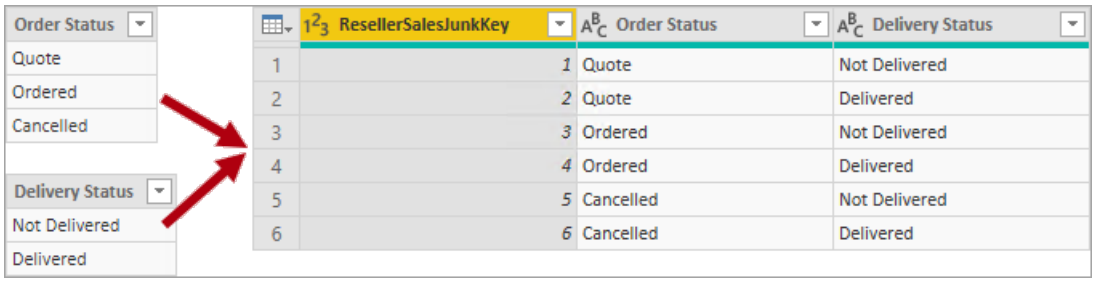
For more information, see Active vs inactive relationship guidance.

Junk dimensions

A **junk dimension** is useful when there are many dimensions, especially consisting of few attributes (perhaps one), and when these attributes have few values. Good candidates include order status columns, or customer demographic columns (gender, age group, etc.).

The design objective of a junk dimension is to consolidate many "small" dimensions into a single dimension to both reduce the model storage size and also reduce **Fields** pane clutter by surfacing fewer model tables.

A junk dimension table is typically the Cartesian product of all dimension attribute members, with a surrogate key column. The surrogate key provides a unique reference to each row in the table. You can build the dimension in a data warehouse, or by using Power Query to create a query that performs full outer query joins, then adds a surrogate key (index column).

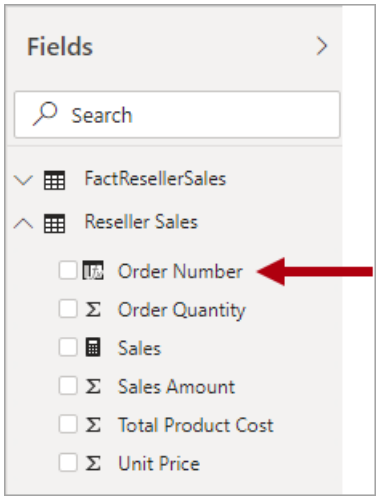


You load this query to the model as a dimension-type table. You also need to merge this query with the fact query, so the index column is loaded to the model to support the creation of a "one-to-many" model relationship.

Degenerate dimensions

A **degenerate dimension** refers to an attribute of the fact table that is required for filtering. At Adventure Works, the reseller sales order number is a good example. In this case, it doesn't make good model design sense to create an independent table consisting of just this one column, because it would increase the model storage size and result in **Fields** pane clutter.

In the Power BI model, it can be appropriate to add the sales order number column to the fact-type table to allow filtering or grouping by sales order number. It is an exception to the formerly introduced rule that you should not mix table types (generally, model tables should be either dimension-type or fact-type).



However, if the Adventure Works resellers sales table has order number *and* order line number columns, and they're required for filtering, a degenerate dimension table would be a good design. For more information, see One-to-one relationship guidance (Degenerate dimensions).

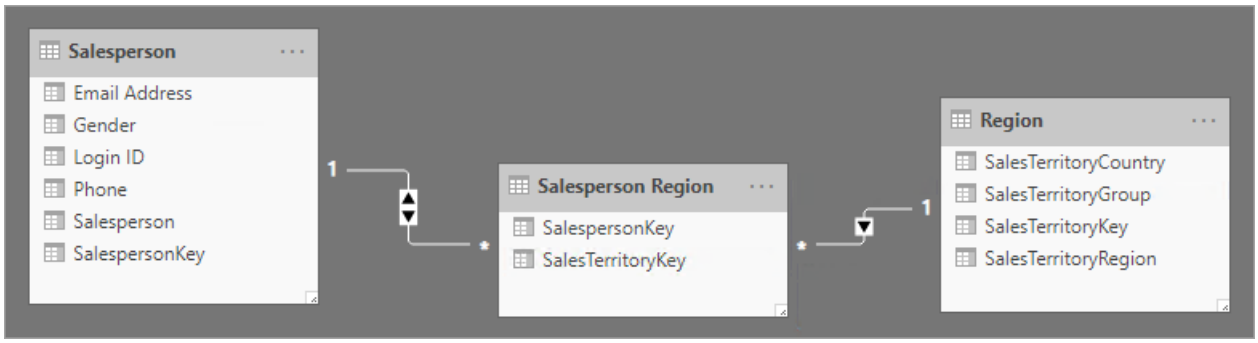
Factless fact tables

A **factless fact** table doesn't include any measure columns. It contains only dimension keys.

A factless fact table could store observations defined by dimension keys. For example, at a particular date and time, a particular customer logged into your web site. You could define a measure to count the rows of the factless fact table to perform analysis of when and how many customers have logged in.

A more compelling use of a factless fact table is to store relationships between dimensions, and it's the Power BI model design approach we recommend defining many-to-many dimension relationships. In a many-to-many dimension relationship design, the factless fact table is referred to as a *bridging table*.

For example, consider that salespeople can be assigned to one *or more* sales regions. The bridging table would be designed as a factless fact table consisting of two columns: salesperson key and region key. Duplicate values can be stored in both columns.



This many-to-many design approach is well documented, and it can be achieved without a bridging table. However, the bridging table approach is considered the best practice when relating two dimensions. For more information, see Many-to-many relationship guidance (Relate two dimension-type tables).

Next steps

For more information about star schema design or Power BI model design, see the following articles:

- Dimensional modeling Wikipedia article
- Create and manage relationships in Power BI Desktop
- One-to-one relationship guidance
- Many-to-many relationship guidance
- Bi-directional relationship guidance
- Active vs inactive relationship guidance
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Data reduction techniques for Import modeling

5/18/2020 • 7 minutes to read • Edit Online

This article targets Power BI Desktop data modelers developing Import models. It describes different techniques to help reduce the data loaded into Import models.

Import models are loaded with data that is compressed and optimized and then stored to disk by the VertiPaq storage engine. When source data is loaded into memory, it is possible to see 10x compression, and so it is reasonable to expect that 10 GB of source data can compress to about 1 GB in size. Further, when persisted to disk an additional 20% reduction can be achieved.

Despite the efficiencies achieved by the VertiPaq storage engine, it is important that you strive to minimize the data that is to be loaded into your models. It is especially true for large models, or models that you anticipate will grow to become large over time. Four compelling reasons include:

- Larger model sizes may not be supported by your capacity. Shared capacity can host models up to 1 GB in size, while Premium capacities can host models up to 13 GB in size. For further information, read the Power BI Premium support for large datasets article.
- Smaller model sizes reduce contention for capacity resources, in particular memory. It allows more models to be concurrently loaded for longer periods of time, resulting in lower eviction rates. For more information, see Managing Premium capacities.
- Smaller models achieve faster data refresh, resulting in lower latency reporting, higher dataset refresh throughput, and less pressure on source system and capacity resources.
- Smaller table row counts can result in faster calculation evaluations, which can deliver better overall query performance.

There are eight different data reduction techniques covered in this article. These techniques include:

- Remove unnecessary columns
- Remove unnecessary rows
- Group by and summarize
- Optimize column data types
- Preference for custom columns
- Disable Power Query query load
- Disable auto date/time
- Switch to Mixed mode

Remove unnecessary columns

Model table columns serve two main purposes:

- **Reporting**, to achieve report designs that appropriate filter, group, and summarize model data
- **Model structure**, by supporting model relationships, model calculations, security roles, and even data color formatting

Columns that don't serve these purposes can probably be removed. Removing columns is referred to as *vertical filtering*.

We recommend that you design models with exactly the right number of columns based on the known reporting requirements. Your requirements may change over time, but bear in mind that it's easier to add columns later than it is to remove them later. Removing columns can break reports or the model structure.

Remove unnecessary rows

Model tables should be loaded with as few rows as possible. It can be achieved by loading filtered rowsets into model tables for two different reasons: to filter by entity or by time. Removing rows is referred to as *horizontal filtering*.

Filtering by entity involves loading a subset of source data into the model. For example, instead of loading sales facts for all sales regions, only load facts for a single region. This design approach will result in many smaller models, and it can also eliminate the need to define row-level security (but will require granting specific dataset permissions in the Power BI service, and creating "duplicate" reports that connect to each dataset). You can leverage the use of Power Query parameters and Power BI Template files to simplify management and publication. For further information, read the Deep Dive into Query Parameters and Power BI Templates blog entry.

Filtering by time involves limiting the amount of *data history* loaded into fact-type tables (and limiting the date rows loaded into the model date tables). We suggest you don't automatically load all available history, unless it is a known reporting requirement. It is helpful to understand that time-based Power Query filters can be parameterized, and even set to use relative time periods (relative to the refresh date, for example, the past five years). Also, bear in mind that retrospective changes to time filters will not break reports; it will just result in less (or more) data history available in reports.

Group by and summarize

Perhaps the most effective technique to reduce a model size is to load pre-summarized data. This technique can be used to raise the grain of fact-type tables. There is a distinct trade-off, however, resulting in loss of detail.

For example, a source sales fact table stores one row per order line. Significant data reduction could be achieved by summarizing all sales metrics, grouping by date, customer, and product. Consider, then, that an even more significant data reduction could be achieved by grouping by date *at month level*. It could achieve a possible 99% reduction in model size, but reporting at day level—or individual order level—is no longer possible. Deciding to summarize fact-type data always involves tradeoffs. Tradeoff could be mitigated by a Mixed model design, and this option is described in the Switch to Mixed mode technique.

Optimize column data types

The VertiPaq storage engine uses separate data structures for each column. By design, these data structures achieve the highest optimizations for numeric column data, which use value encoding. Text and other non-numeric data, however, uses hash encoding. It requires the storage engine to assign a numeric identifier to each unique text value contained in the column. It is the numeric identifier, then, that is then stored in the data structure, requiring a hash lookup during storage and querying.

In some specific instances, you can convert source text data into numeric values. For example, a sales order number may be consistently prefixed by a text value (e.g. "SO123456"). The prefix could be removed, and the order number value converted to whole number. For large tables, it can result in significant data reduction, especially when the column contains unique or high cardinality values.

In this example, we recommend that you set the column Default Summarization property to "Do Not Summarize". It will help to minimize the inappropriate summarization of the order number values.

Preference for custom columns

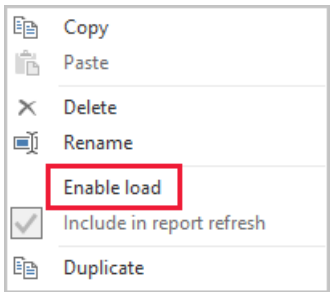
The VertiPaq storage engine stores model calculated columns (defined in DAX) just like regular Power Query-sourced columns. However, the data structures are stored slightly differently, and typically achieve less efficient compression. Also, they are built once all Power Query tables are loaded, which can result in extended data refresh times. It is therefore less efficient to add table columns as *calculated* columns than Power Query *computed* columns (defined in M).

Preference should be creating custom columns in Power Query. When the source is a database, you can achieve greater load efficiency in two ways. The calculation can be defined in the SQL statement (using the native query language of the provider), or it can be materialized as a column in the data source.

However, in some instances, model calculated columns may be the better choice. It can be the case when the formula involves evaluating measures, or it requires specific modeling functionality only supported in DAX functions. For information on one such example, refer to the Understanding functions for parent-child hierarchies in DAX article.

Disable Power Query query load

Power Query queries that are intended support data integration with other queries should not be loaded to the model. To avoid loading the query to the model, take care to ensure that you disable query load in these instances.



Disable auto date/time

Power BI Desktop includes an option called *Auto date/time*. When enabled, it creates a hidden auto date/time table for date columns to support report authors when configuring filters, grouping, and drill down for calendar time periods. The hidden tables are in fact calculated tables that will increase the size of the model. For guidance about using this option, refer to the Auto date/time guidance in Power BI Desktop article.

Switch to Mixed mode

In Power BI Desktop, a Mixed mode design produces a Composite model. Essentially, it allows you to determine storage mode *for each table*. Therefore, each table can have its Storage Mode property set as Import or DirectQuery (Dual is another option).

An effective technique to reduce the model size is to set the Storage Mode property for larger fact-type tables to DirectQuery. Consider that this design approach could work well in conjunction with the Group by and summarize technique introduced earlier. For example, summarized sales data could be used to achieve high performance "summary" reporting. A drill through page could display granular sales for specific (and narrow) filter context, displaying all in-context sales orders. In this example, the drill through page would include visuals based on a DirectQuery table to retrieve the sales order data.

There are, however, many security and performance implications related to Composite models. For further information, read the Use composite models in Power BI Desktop article.

Next steps

For more information about Power BI Import model design, see the following articles:

- Use composite models in Power BI Desktop
- Storage mode in Power BI Desktop
- Questions? Try asking the Power BI Community

Auto date/time guidance in Power BI Desktop

5/18/2020 • 3 minutes to read • Edit Online

This article targets data modelers developing Import or Composite models in Power BI Desktop. It provides guidance, recommendations, and considerations when using Power BI Desktop *Auto date/time* in specific situations. For an overview and general introduction to *Auto date/time*, see Auto date/time in Power BI Desktop.

The *Auto date/time* option delivers convenient, fast, and easy-to-use time intelligence. Reports authors can work with time intelligence when filtering, grouping, and drilling down through calendar time periods.

Considerations

The following bulleted list describes considerations—and possible limitations—related to the *Auto date/time* option.

- **Applies to all or none:** When the *Auto date/time* option is enabled, it applies to all date columns (except calculated columns) in Import tables that aren't the "many" side of a relationship. It can't be selectively enabled or disabled on a column-by-column basis.
- **Calendar periods only:** The year and quarter columns relate to calendar periods. It means that the year begins on January 1 and finishes on December 31. There's no ability to customize the year commencement (or completion) date.
- **Customization:** It's not possible to customize the values used to describe time periods. Further, it's not possible to add additional columns to describe other time periods, for example, weeks.
- **Year filtering:** The **Quarter**, **Month**, and **Day** column values don't include the year value. For example, the **Month** column contains the month names only (that is, January, February, etc.). The values are not fully self-describing, and in some report designs may not communicate the year filter context.

  That's why it's important that filters or grouping must take place on the **Year** column. When drilling down by using the hierarchy year will be filtered, unless the **Year** level is intentionally removed. If there's no filter or group by year, a grouping by month, for example, would summarize values across all years for that month.
- **Single table date filtering:** Because each date column produces its own (hidden) auto date/time table, it's not possible to apply a time filter to one table and have it propagate to multiple model tables. Filtering in this way is a common modeling requirement when reporting on multiple subjects (fact-type tables) like sales and sales budget. When using auto date/time, the report author will need to apply filters to each different date column.
- **Model size:** For each date column that generates a hidden auto date/time table, it will result in an increased model size and also extend the data refresh time.
- **Other reporting tools:** It isn't possible to work with auto date/time tables when:
  - Using Analyze in Excel.
  - Using Power BI paginated report Analysis Services query designers.
  - Connecting to the model using non-Power BI report designers.

Recommendations

We recommended that you keep the *Auto date/time* option enabled only when you work with calendar time periods, and when you have simplistic model requirements in relation to time. Using this option can also be

convenient when creating ad hoc models or performing data exploration or profiling.

When your data source already defines a date dimension table, this table should be used to consistently define time within your organization. It will certainly be the case if your data source is a data warehouse. Otherwise, you can generate date tables in your model by using the DAX CALENDAR or CALENDARAUTO functions. You can then add calculated columns to support the known time filtering and grouping requirements. This design approach may allow you to create a single date table that propagates to all fact-type tables, possibly resulting a single table to apply time filters. For further information on creating date tables, read the Set and use date tables in Power BI Desktop article.

If the *Auto date/time* option isn't relevant to your projects, we recommend that you disable the global *Auto date/time* option. It will ensure that all new Power BI Desktop files you create won't enable the *Auto date/time* option.

Next steps

For more information related to this article, check out the following resources:

- Auto date/time in Power BI Desktop
- Set and use date tables in Power BI Desktop
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

One-to-one relationship guidance

5/13/2020 • 7 minutes to read • Edit Online

This article targets you as a data modeler working with Power BI Desktop. It provides you with guidance on working with one-to-one model relationships. A one-to-one relationship can be created when both tables each contain a column of common and unique values.

> **NOTE**
>
> An introduction to model relationships is not covered in this article. If you're not completely familiar with relationships, their properties or how to configure them, we recommend that you first read the Model relationships in Power BI Desktop article.
>
> It's also important that you have an understanding of star schema design. For more information, see Understand star schema and the importance for Power BI.

There are two scenarios that involve one-to-one relationships:

- Degenerate dimensions: You can derive a degenerate dimension from a fact-type table.
- Row data spans across tables: A single business entity or subject is loaded as two (or more) model tables, possibly because their data is sourced from different data stores. This scenario can be common for dimension-type tables. For example, master product details are stored in an operational sales system, and supplementary product details are stored in a different source.

  It's unusual, however, that you'd relate two fact-type tables with a one-to-one relationship. It's because both fact-type tables would need to have the same dimensionality and granularity. Also, each fact-type table would need unique columns to allow the model relationship to be created.

Degenerate dimensions

When columns from a fact-type table are used for filtering or grouping, you can consider making them available in a separate table. This way, you separate columns used for filter or grouping, from those columns used to summarize fact rows. This separation can:

- Reduce storage space
- Simplify model calculations
- Contribute to improved query performance
- Deliver a more intuitive **Fields** pane experience to your report authors

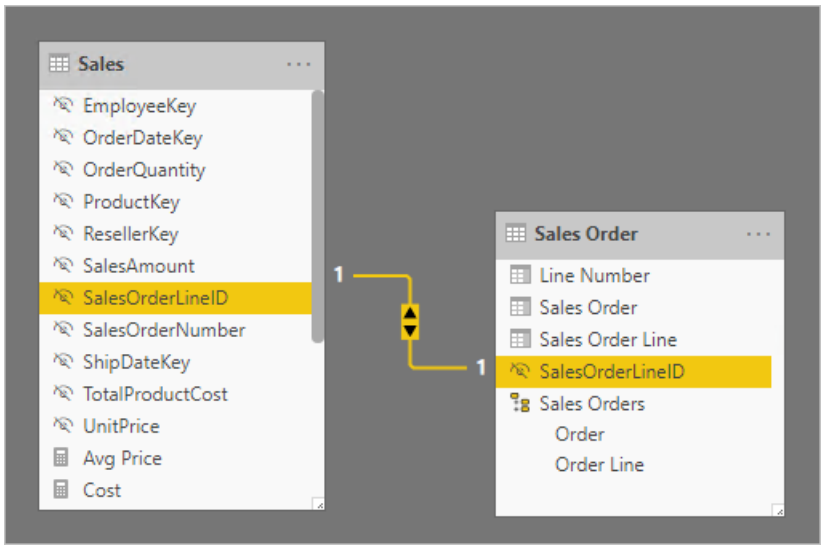
Consider a source sales table that stores sales order details in two columns.

	OrderNumber	OrderLineNumber	OrderDate	DueDate	ShipDate	ProductID	CustomerID
1	43659	1	2018-12-29	2019-01-10	2019-01-05	349	676
2	43659	2	2018-12-29	2019-01-10	2019-01-05	350	676
3	43659	3	2018-12-29	2019-01-10	2019-01-05	351	676
4	43660	1	2018-12-29	2019-01-10	2019-01-05	326	117
5	43660	2	2018-12-29	2019-01-10	2019-01-05	319	117

The **OrderNumber** column stores the order number, and the **OrderLineNumber** column stores a sequence of lines within the order.

In the following model diagram, notice that the order number and order line number columns haven't been loaded to the **Sales** table. Instead, their values were used to create a surrogate key column named **SalesOrderLineID**.

(The key value is calculated by multiplying the order number by 1000, and then adding the order line number.)



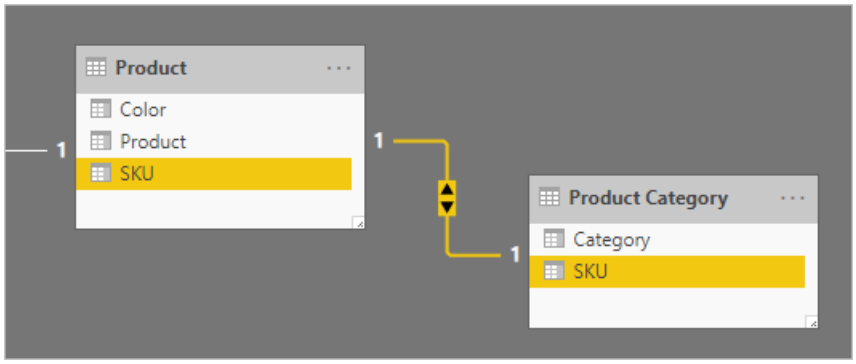
The **Sales Order** table provides a rich experience for report authors with three columns: **Sales Order**, **Sales Order Line**, and **Line Number**. It also includes a hierarchy. These table resources support report designs that need to filter, group by, or drill down through orders and order lines.

As the **Sales Order** table is derived from the sales data, there should be exactly the same number of rows in each table. Further, there should be matching values between each **SalesOrderLineID** column.

Row data spans across tables

Consider an example involving two one-to-one related dimension-type tables: **Product**, and **Product Category**. Each table represents imported data and has a **SKU** (Stock-Keeping Unit) column containing unique values.

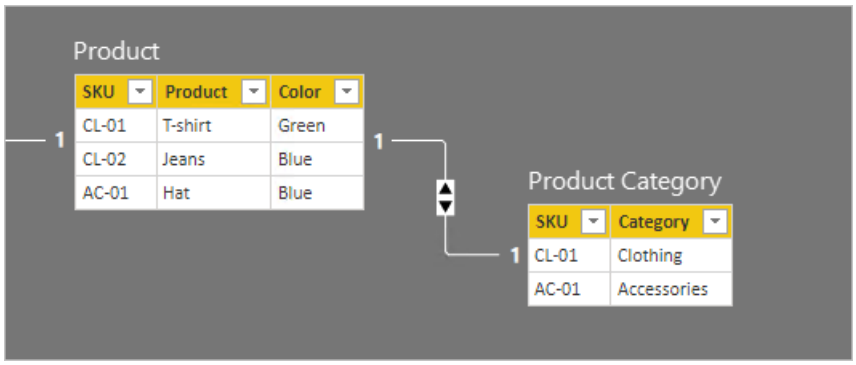
Here's a partial model diagram of the two tables.



The first table is named **Product**, and it contains three columns: **Color**, **Product**, and **SKU**. The second table is named **Product Category**, and it contains two columns: **Category**, and **SKU**. A one-to-one relationship relates the two **SKU** columns. The relationship filters in both directions, which is always the case for one-to-one relationships.

To help describe how the relationship filter propagation works, the model diagram has been modified to reveal the table rows. All examples in this article are based on this data.

> **NOTE**
>
> It's not possible to display table rows in the Power BI Desktop model diagram. It's done in this article to support the discussion with clear examples.

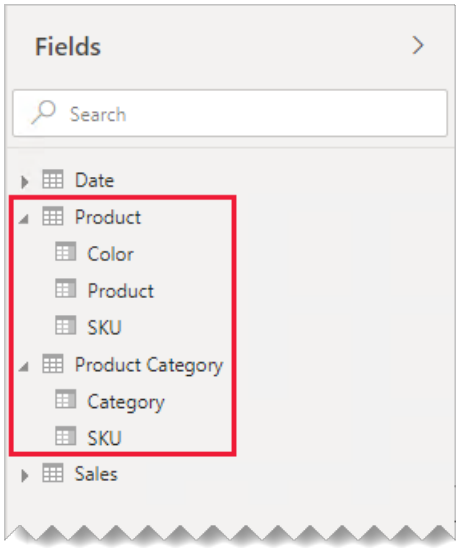


The row details for the two tables are described in the following bulleted list:

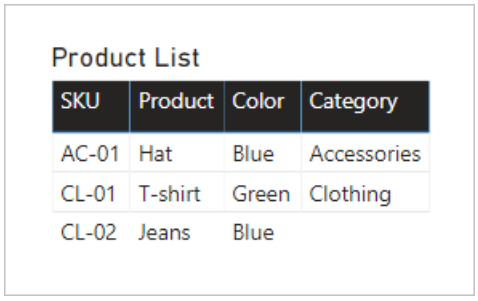
- The **Product** table has three rows:
  - **SKU** CL-01, **Product** T-shirt, **Color** Green
  - **SKU** CL-02, **Product** Jeans, **Color** Blue
  - **SKU** AC-01, **Product** Hat, **Color** Blue
- The **Product Category** table has two rows:
  - **SKU** CL-01, **Category** Clothing
  - **SKU** AC-01, **Category** Accessories

Notice that the **Product Category** table doesn't include a row for the product SKU CL-02. We'll discuss the consequences of this missing row later in this article.

In the **Fields** pane, report authors will find product-related fields in two tables: **Product** and **Product Category**.



Let's see what happens when fields from both tables are added to a table visual. In this example, the **SKU** column is sourced from the **Product** table.



Notice that the **Category** value for product SKU CL-02 is BLANK. It's because there's no row in the **Product Category** table for this product.

Recommendations

When possible, we recommend you avoid creating one-to-one model relationships when row data spans across model tables. It's because this design can:

- Contribute to **Fields** pane clutter, listing more tables than necessary
- Make it difficult for report authors to find related fields, because they're distributed across multiple tables
- Limit the ability to create hierarchies, as their levels must be based on columns from the *same table*
- Produce unexpected results when there isn't a complete match of rows between the tables

Specific recommendations differ depending on whether the one-to-one relationship is *intra-island* or *inter-island*. For more information about relationship evaluation, see Model relationships in Power BI Desktop (Relationship evaluation).

Intra-island one-to-one relationship

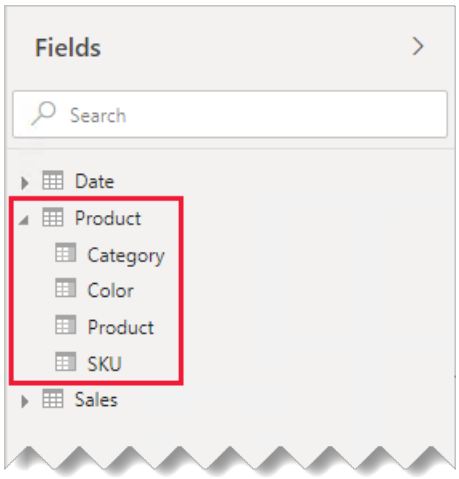
When a one-to-one *intra-island* relationship exists between tables, we recommend consolidating the data into a single model table. It's done by merging the Power Query queries.

The following steps present a methodology to consolidate and model the one-to-one related data:

1. **Merge queries**: When combining the two queries, give consideration to the completeness of data in each query. If one query contains a complete set of rows (like a master list), merge the other query with it. Configure the merge transformation to use a *left outer join*, which is the default join type. This join type ensures you'll keep all rows of the first query, and supplement them with any matching rows of the second query. Expand all required columns of the second query into the first query.

2. **Disable query load**: Be sure to disable the load of the second query. This way, it won't load its result as a model table. This configuration reduces the data model storage size, and helps to unclutter the **Fields** pane.

   In our example, report authors now find a single table named **Product** in the **Fields** pane. It contains all product-related fields.

3. **Replace missing values**: If the second query has unmatched rows, NULLs will appear in the columns introduced from it. When appropriate, consider replacing NULLs with a token value. Replacing missing values is especially important when report authors filter or group by the column values, as BLANKs could appear in report visuals.

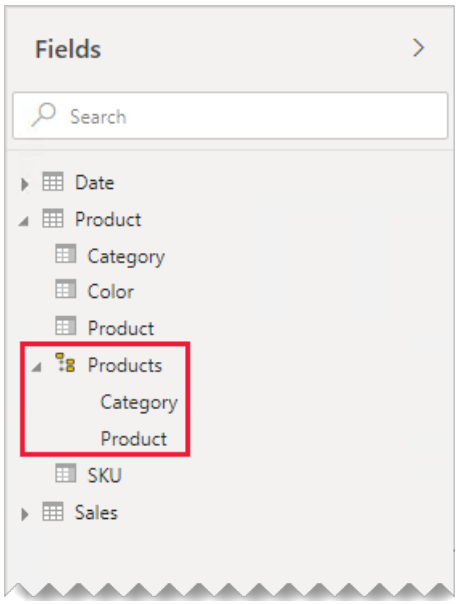
   In the following table visual, notice that the category for product SKU CL-02 now reads *[Undefined]*. In the query, null categories were replaced with this token text value.

Product List

SKU	Product	Color	Category
AC-01	Hat	Blue	Accessories
CL-01	T-shirt	Green	Clothing
CL-02	Jeans	Blue	[Undefined]

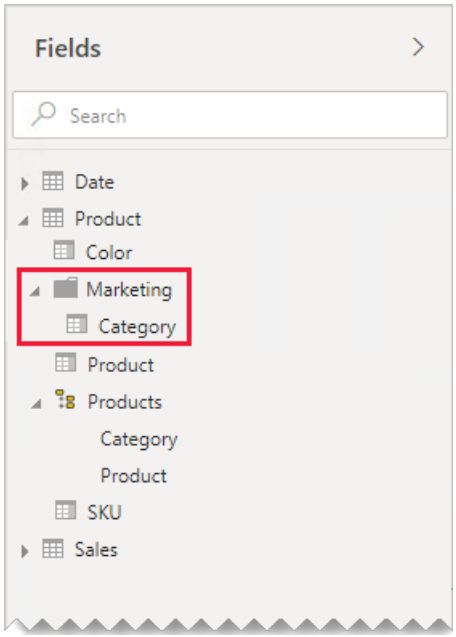
4. **Create hierarchies**: If relationships exist *between the columns* of the now-consolidated table, consider creating hierarchies. This way, report authors will quickly identify opportunities for report visual drilling.

   In our example, report authors now can use a hierarchy that has two levels: **Category** and **Product**.



If you like how separate tables help organize your fields, we still recommend consolidating into a single table. You can still organize your fields, but by using *display folders* instead.

In our example, report authors can find the **Category** field within the **Marketing** display folder.



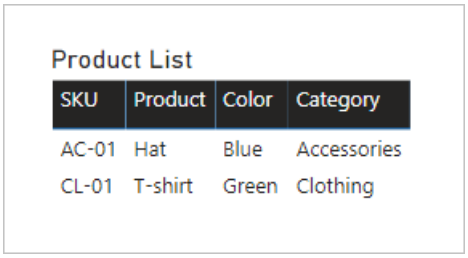
Should you still decide to define one-to-one intra-island relationships in your model, when possible, ensure there are matching rows in the related tables. As a one-to-one intra-island relationship is evaluated as a strong relationship, data integrity issues could surface in your report visuals as BLANKs. (You can see an example of a

BLANK grouping in the first table visual presented in this article.)

Inter-island one-to-one relationship

When a one-to-one *inter-island* relationship exists between tables, there's no alternative model design—unless you pre-consolidate the data in your data sources. Power BI will evaluate the one-to-one model relationship as a weak relationship. Therefore, take care to ensure there are matching rows in the related tables, as unmatched rows will be eliminated from query results.

Let's see what happens when fields from both tables are added to a table visual, and a weak relationship exists between the tables.



The table displays two rows only. Product SKU CL-02 is missing because there's no matching row in the **Product Category** table.

Next steps

For more information related to this article, check out the following resources:

- Model relationships in Power BI Desktop
- Understand star schema and the importance for Power BI
- Relationship troubleshooting guidance
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Many-to-many relationship guidance

5/13/2020 • 14 minutes to read • Edit Online

This article targets you as a data modeler working with Power BI Desktop. It describes three different many-to-many modeling scenarios. It also provides you with guidance on how to successfully design for them in your models.

> **NOTE**
>
> An introduction to model relationships is not covered in this article. If you're not completely familiar with relationships, their properties or how to configure them, we recommend that you first read the Model relationships in Power BI Desktop article.
>
> It's also important that you have an understanding of star schema design. For more information, see Understand star schema and the importance for Power BI.

There are, in fact, three many-to-many scenarios. They can occur when you're required to:

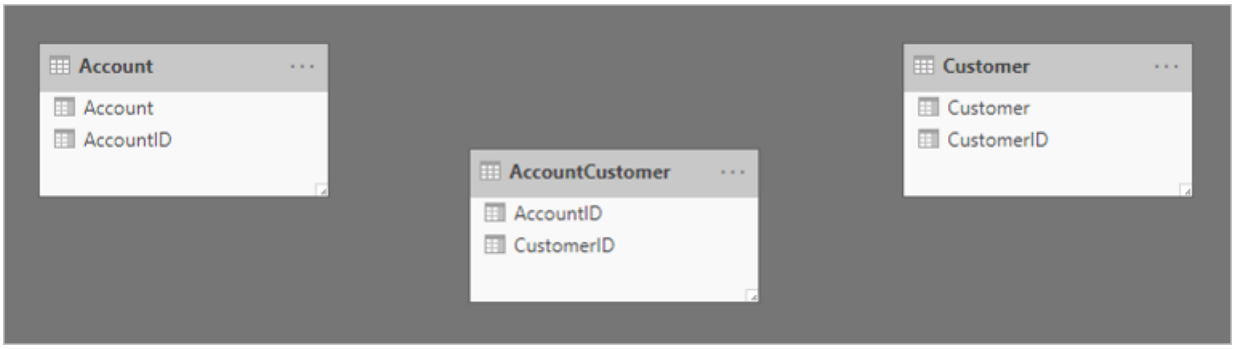
- Relate two dimension-type tables
- Relate two fact-type tables
- Relate higher grain fact-type tables, when the fact-type table stores rows at a higher grain than the dimension-type table rows

Relate many-to-many dimensions

Let's consider the first many-to-many scenario type with an example. The classic scenario relates two entities: bank customers and bank accounts. Consider that customers can have multiple accounts, and accounts can have multiple customers. When an account has multiple customers, they're commonly called *joint account holders*.

Modeling these entities is straight forward. One dimension-type table stores accounts, and another dimension-type table stores customers. As is characteristic of dimension-type tables, there's an ID column in each table. To model the relationship between the two tables, a third table is required. This table is commonly referred to as a *bridging table*. In this example, it's purpose is to store one row for each customer-account association. Interestingly, when this table only contains ID columns, it's called a *factless fact table*.

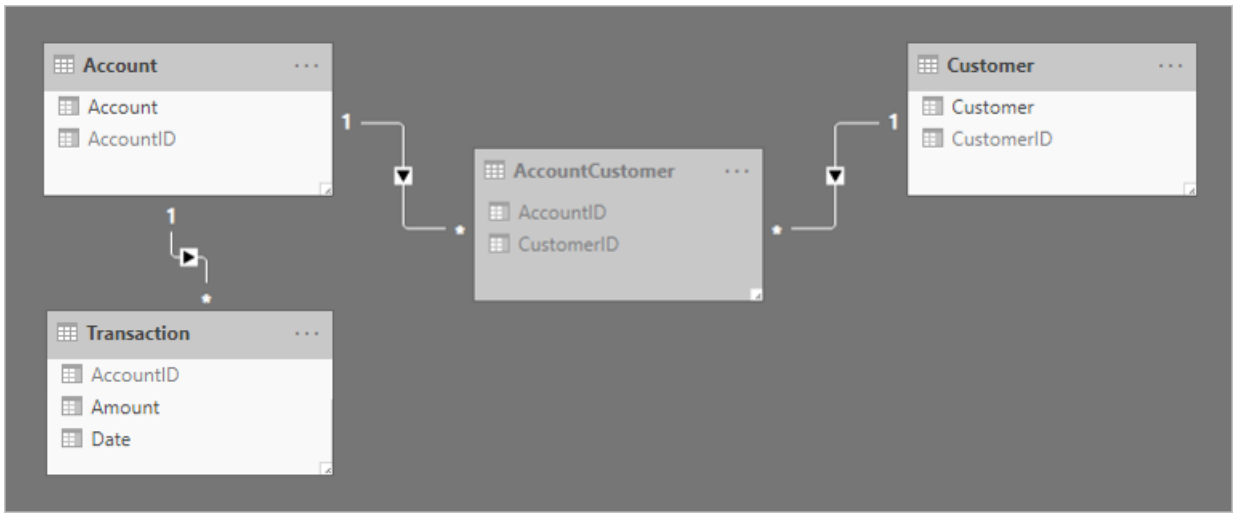
Here's a simplistic model diagram of the three tables.



The first table is named **Account**, and it contains two columns: **AccountID** and **Account**. The second table is named **AccountCustomer**, and it contains two columns: **AccountID** and **CustomerID**. The third table is named **Customer**, and it contains two columns: **CustomerID** and **Customer**. Relationships don't exist between any of the tables.

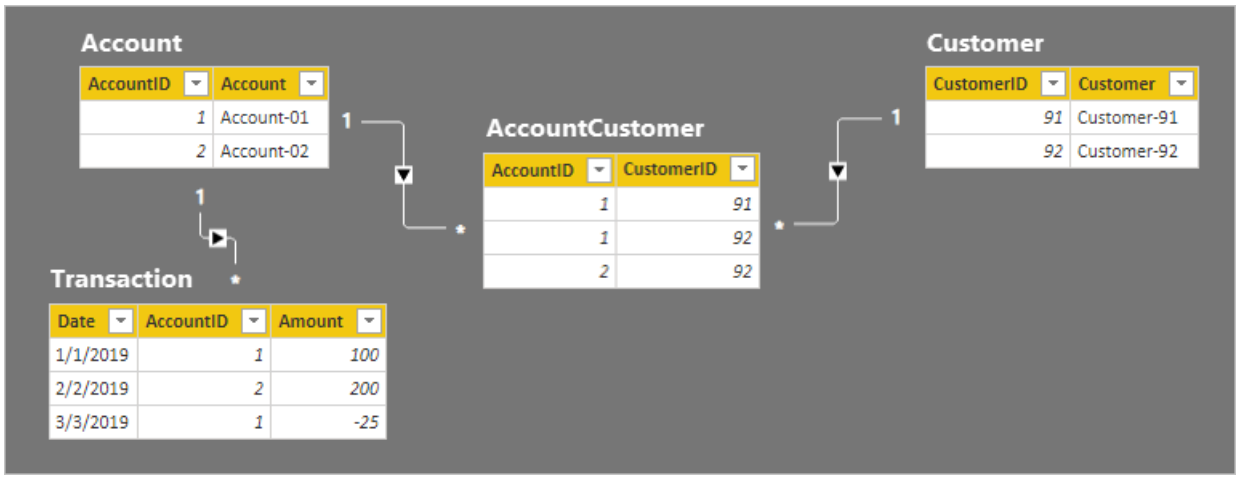
Two one-to-many relationships are added to relate the tables. Here's an updated model diagram of the related

tables. A fact-type table named **Transaction** has been added. It records account transactions. The bridging table and all ID columns have been hidden.



To help describe how the relationship filter propagation works, the model diagram has been modified to reveal the table rows.

> **NOTE**
> It's not possible to display table rows in the Power BI Desktop model diagram. It's done in this article to support the discussion with clear examples.



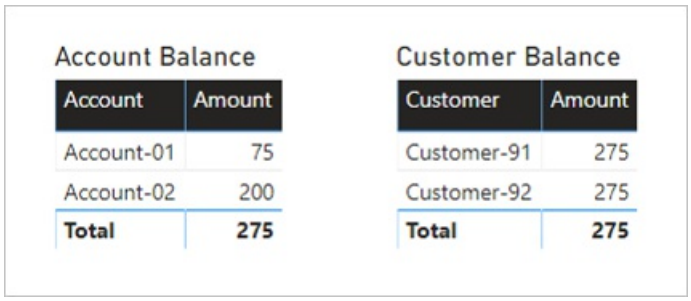
The row details for the four tables are described in the following bulleted list:

- The **Account** table has two rows:
  - **AccountID** 1 is for Account-01
  - **AccountID** 2 is for Account-02
- The **Customer** table has two rows:
  - **CustomerID** 91 is for Customer-91
  - **CustomerID** 92 is for Customer-92
- The **AccountCustomer** table has three rows:
  - **AccountID** 1 is associated with **CustomerID** 91
  - **AccountID** 1 is associated with **CustomerID** 92
  - **AccountID** 3 is associated with **CustomerID** 92
- The **Transaction** table has three rows:
  - **Date** January 1 2019, **AccountID** 1, **Amount** 100

- **Date** February 2 2019, **AccountID** 2, **Amount** 200
- **Date** March 3 2019, **AccountID** 1, **Amount** -25

Let's see what happens when the model is queried.

Below are two visuals that summarize the **Amount** column from the **Transaction** table. The first visual groups by account, and so the sum of the **Amount** columns represents the *account balance*. The second visual groups by customer, and so the sum of the **Amount** columns represents the *customer balance*.



Account Balance

Account	Amount
Account-01	75
Account-02	200
Total	**275**

Customer Balance

Customer	Amount
Customer-91	275
Customer-92	275
Total	**275**

The first visual is titled **Account Balance**, and it has two columns: **Account** and **Amount**. It displays the following result:

- Account-01 balance amount is 75
- Account-02 balance amount is 200
- The total is 275

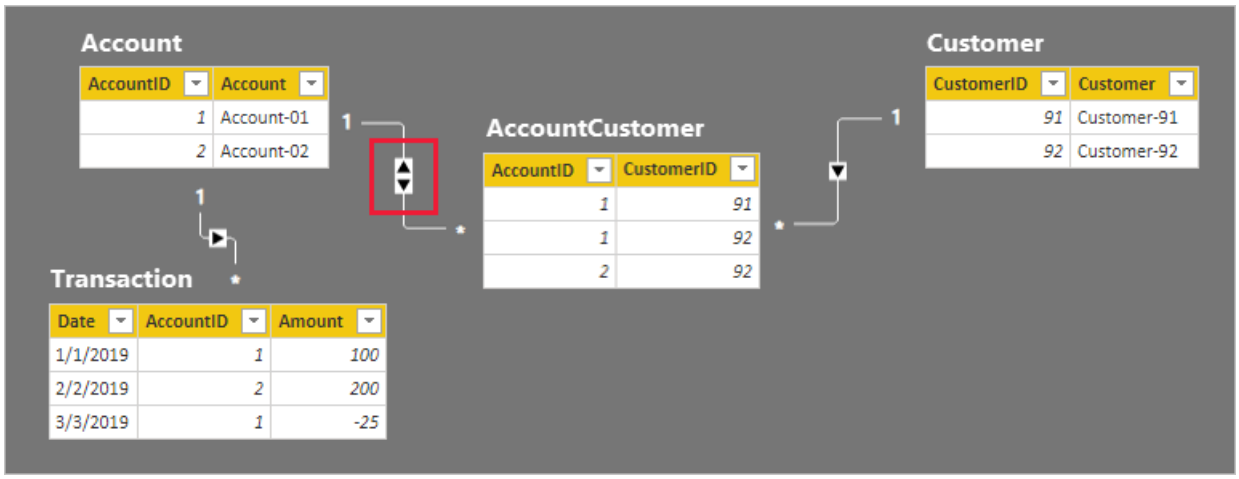
The second visual is titled **Customer Balance**, and it has two columns: **Customer** and **Amount**. It displays the following result:

- Customer-91 balance amount is 275
- Customer-92 balance amount is 275
- The total is 275

A quick glance at the table rows and the **Account Balance** visual reveals that the result is correct, for each account and the total amount. It's because each account grouping results in a filter propagation to the **Transaction** table for that account.

However, something doesn't appear correct with the **Customer Balance** visual. Each customer in the **Customer Balance** visual has the same balance as the total balance. This result could only be correct if every customer was a joint account holder of every account. That's not the case in this example. The issue is related to filter propagation. It's not flowing all the way to the **Transaction** table.

Follow the relationship filter directions from the **Customer** table to the **Transaction** table. It should be apparent that the relationship between the **Account** and **AccountCustomer** table is propagating in the wrong direction. The filter direction for this relationship must be set to **Both**.



Account Balance

Account	Amount
Account-01	75
Account-02	200
Total	**275**

Customer Balance

Customer	Amount
Customer-91	75
Customer-92	275
Total	**275**

As expected, there has been no change to the **Account Balance** visual.

The **Customer Balance** visuals, however, now displays the following result:

- Customer-91 balance amount is 75
- Customer-92 balance amount is 275
- The total is 275

The **Customer Balance** visual now displays a correct result. Follow the filter directions for yourself, and see how the customer balances were calculated. Also, understand that the visual total means *all customers*.

Someone unfamiliar with the model relationships could conclude that the result is incorrect. They might ask: *Why isn't the total balance for **Customer-91** and **Customer-92** equal to 350 (75 + 275)?*

The answer to their question lies in understanding the many-to-many relationship. Each customer balance can represent the addition of multiple account balances, and so the customer balances are *non-additive*.

Relate many-to-many dimensions guidance

When you have a many-to-many relationship between dimension-type tables, we provide the following guidance:

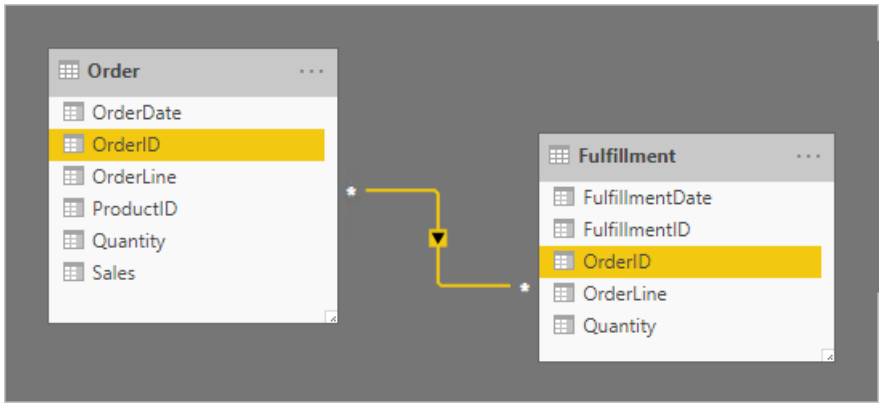
- Add each many-to-many related entity as a model table, ensuring it has a unique identifier (ID) column
- Add a bridging table to store associated entities
- Create one-to-many relationships between the three tables
- Configure **one** bi-directional relationship to allow filter propagation to continue to the fact-type tables
- When it isn't appropriate to have missing ID values, set the **Is Nullable** property of ID columns to FALSE—data refresh will then fail if missing values are sourced
- Hide the bridging table (unless it contains additional columns or measures required for reporting)
- Hide any ID columns that aren't suitable for reporting (for example, when IDs are surrogate keys)
- If it makes sense to leave an ID column visible, ensure that it's on the "one" slide of the relationship—always hide the "many" side column. It results in the best filter performance.
- To avoid confusion or misinterpretation, communicate explanations to your report users—you can add descriptions with text boxes or visual header tooltips

We don't recommend you relate many-to-many dimension-type tables directly. This design approach requires configuring a relationship with a many-to-many cardinality. Conceptually it can be achieved, yet it implies that the related columns will contain duplicate values. It's a well-accepted design practice, however, that dimension-type tables have an ID column. Dimension-type tables should always use the ID column as the "one" side of a relationship.

Relate many-to-many facts

The second many-to-many scenario type involves relating two fact-type tables. Two fact-type tables can be related directly. This design technique can be useful for quick and simple data exploration. However, and to be clear, we generally don't recommend this design approach. We'll explain why later in this section.

Let's consider an example that involves two fact-type tables: **Order** and **Fulfillment**. The **Order** table contains one row per order line, and the **Fulfillment** table can contains zero or more rows per order line. Rows in the **Order** table represent sales orders. Rows in the **Fulfillment** table represent order items that have been shipped. A many-to-many relationship relates the two **OrderID** columns, with filter propagation only from the **Order** table (**Order** filters **Fulfillment**).



The relationship cardinality is set to many-to-many to support storing duplicate **OrderID** values in both tables. In the **Order** table, duplicate **OrderID** values can exist because an order can have multiple lines. In the **Fulfillment** table, duplicate **OrderID** values can exist because orders may have multiple lines, and order lines can be fulfilled by many shipments.

Let's now take a look at the table rows. In the **Fulfillment** table, notice that order lines can be fulfilled by multiple shipments. (The absence of an order line means the order is yet to be fulfilled.)

Order

OrderDate	OrderID	OrderLine	ProductID	OrderQuantity	Sales
01/01/2019	1	1	Prod-A	5	50
01/01/2019	1	2	Prod-B	10	80
02/02/2019	2	1	Prod-B	5	40
02/02/2019	2	2	Prod-C	1	20
03/03/2019	3	1	Prod-C	5	100

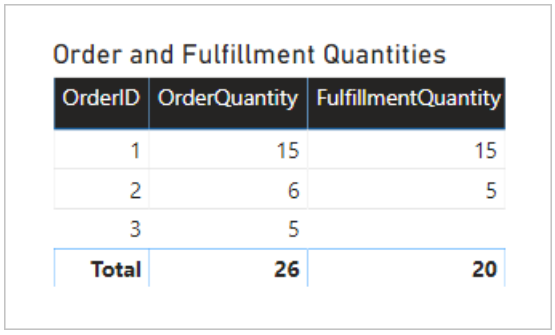
Fulfillment

FulfillmentDate	FulfillmentID	OrderID	OrderLine	FulfillmentQuantity
01/01/2019	50	1	1	2
02/02/2019	51	2	1	5
02/02/2019	52	1	1	3
02/02/2019	53	1	2	10

The row details for the two tables are described in the following bulleted list:

- The **Order** table has five rows:
  - **OrderDate** January 1 2019, **OrderID** 1, **OrderLine** 1, **ProductID** Prod-A, **OrderQuantity** 5, **Sales** 50
  - **OrderDate** January 1 2019, **OrderID** 1, **OrderLine** 2, **ProductID** Prod-B, **OrderQuantity** 10, **Sales** 80
  - **OrderDate** February 2 2019, **OrderID** 2, **OrderLine** 1, **ProductID** Prod-B, **OrderQuantity** 5, **Sales** 40
  - **OrderDate** February 2 2019, **OrderID** 2, **OrderLine** 2, **ProductID** Prod-C, **OrderQuantity** 1, **Sales** 20
  - **OrderDate** March 3 2019, **OrderID** 3, **OrderLine** 1, **ProductID** Prod-C, **OrderQuantity** 5, **Sales** 100
- The **Fulfillment** table has four rows:
  - **FulfillmentDate** January 1 2019, **FulfillmentID** 50, **OrderID** 1, **OrderLine** 1, **FulfillmentQuantity** 2
  - **FulfillmentDate** February 2 2019, **FulfillmentID** 51, **OrderID** 2, **OrderLine** 1, **FulfillmentQuantity** 5
  - **FulfillmentDate** February 2 2019, **FulfillmentID** 52, **OrderID** 1, **OrderLine** 1, **FulfillmentQuantity** 3
  - **FulfillmentDate** January 1 2019, **FulfillmentID** 53, **OrderID** 1, **OrderLine** 2, **FulfillmentQuantity** 10

Let's see what happens when the model is queried. Here's a table visual comparing order and fulfillment quantities by the **Order** table **OrderID** column.



Order and Fulfillment Quantities

OrderID	OrderQuantity	FulfillmentQuantity
1	15	15
2	6	5
3	5	
Total	**26**	**20**

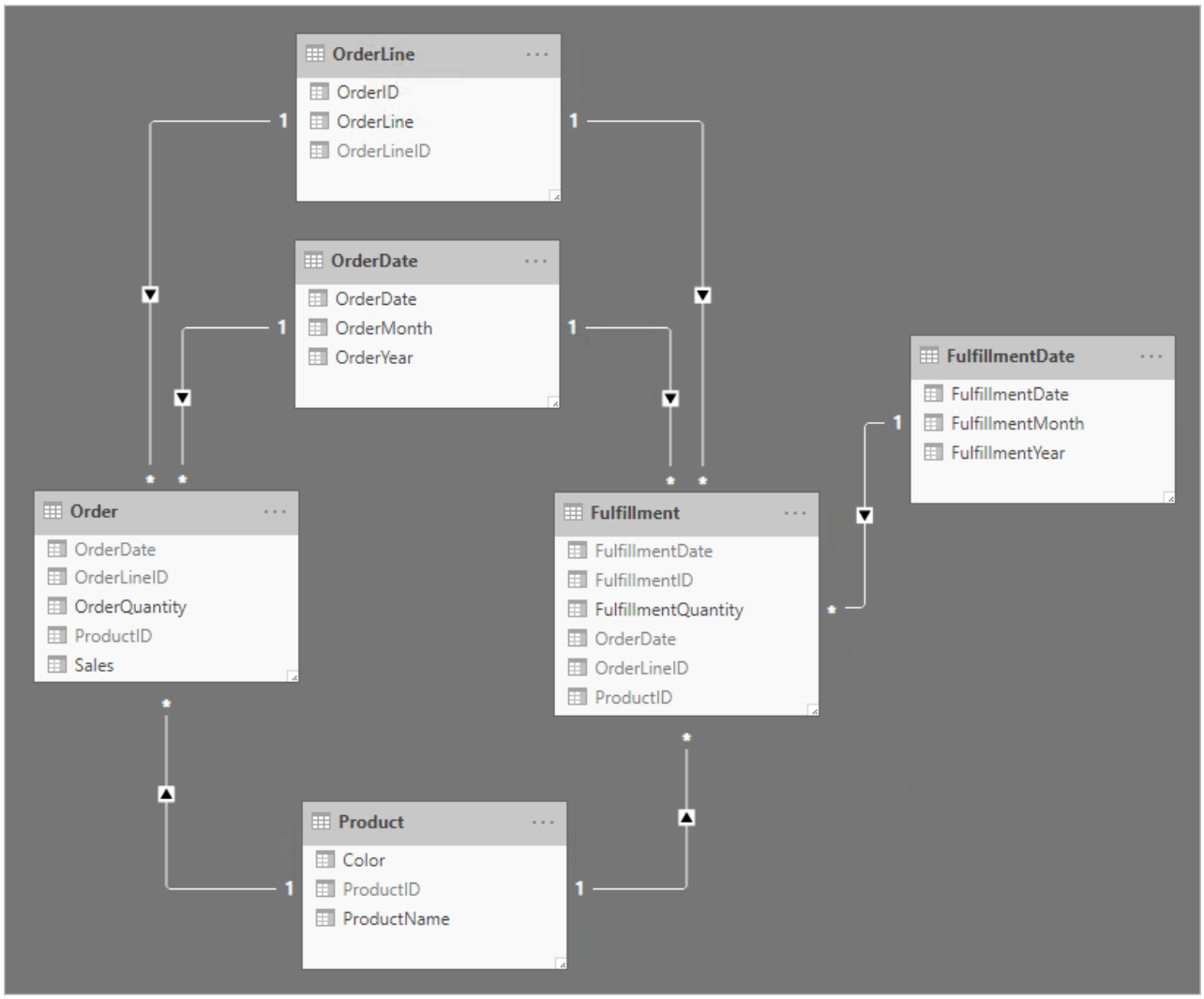
The visual presents an accurate result. However, the usefulness of the model is limited—you can only filter or group by the **Order** table **OrderID** column.

Relate many-to-many facts guidance

Generally, we don't recommend relating two fact-type tables directly using many-to-many cardinality. The main reason is because the model won't provide flexibility in the ways you report visuals filter or group. In the example, it's only possible for visuals to filter or group by the **Order** table **OrderID** column. An additional reason relates to the quality of your data. If your data has integrity issues, it's possible some rows may be omitted during querying due to the nature of the *weak relationship*. For more information, see Model relationships in Power BI Desktop (Relationship evaluation).

Instead of relating fact-type tables directly, we recommend you adopt Star Schema design principles. You do it by adding dimension-type tables. The dimension-type tables then relate to the fact-type tables by using one-to-many relationships. This design approach is robust as it delivers flexible reporting options. It lets you filter or group using any of the dimension-type columns, and summarize any related fact-type table.

Let's consider a better solution.



Notice the following design changes:

- The model now has four additional tables: **OrderLine**, **OrderDate**, **Product**, and **FulfillmentDate**
- The four additional tables are all dimension-type tables, and one-to-many relationships relate these tables to the fact-type tables
- The **OrderLine** table contains an **OrderLineID** column, which represents the **OrderID** value multiplied by 100, plus the **OrderLine** value—a unique identifier for each order line
- The **Order** and **Fulfillment** tables now contain an **OrderLineID** column, and they no longer contain the **OrderID** and **OrderLine** columns
- The **Fulfillment** table now contains **OrderDate** and **ProductID** columns
- The **FulfillmentDate** table relates only to the **Fulfillment** table
- All unique identifier columns are hidden

Taking the time to apply star schema design principles delivers the following benefits:

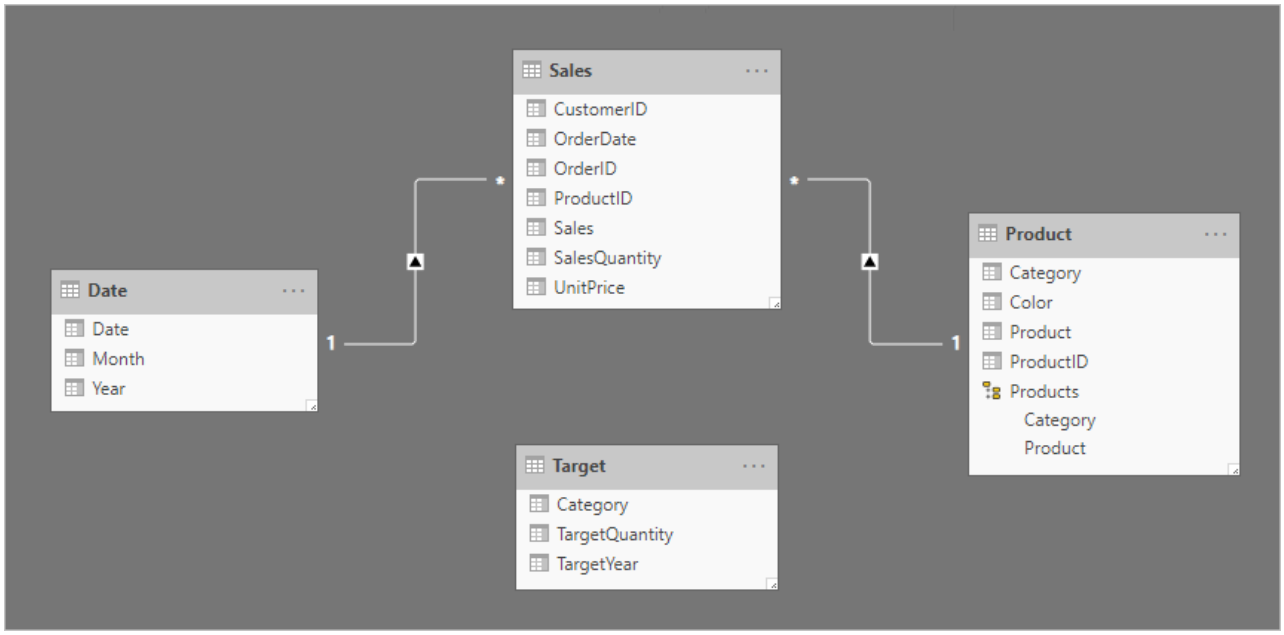
- Your report visuals can *filter or group* by any visible column from the dimension-type tables
- Your report visuals can *summarize* any visible column from the fact-type tables
- Filters applied to the **OrderLine**, **OrderDate**, or **Product** tables will propagate to both fact-type tables
- All relationships are one-to-many, and each relationship is a *strong relationship*. Data integrity issues won't be masked. For more information, see Model relationships in Power BI Desktop (Relationship evaluation).

Relate higher grain facts

This many-to-many scenario is very different from the other two already described in this article.

Let's consider an example involving four tables: **Date**, **Sales**, **Product**, and **Target**. The **Date** and **Product** are

dimension-type tables, and one-to-many relationships relate each to the **Sales** fact-type table. So far, it represents a good star schema design. The **Target** table, however, is yet to be related to the other tables.



The **Target** table contains three columns: **Category**, **TargetQuantity**, and **TargetYear**. The table rows reveal a granularity of year and product category. In other words, targets—used to measure sales performance—are set each year for each product category.

TargetYear	Category	TargetQuantity
01/01/2019	Clothing	*10*
01/01/2019	Accessories	*20*
01/01/2019	Clothing	*30*
01/01/2020	Accessories	*40*
01/01/2020	Food	*100*
01/01/2020	Food	*200*

Because the **Target** table stores data at a higher level than the dimension-type tables, a one-to-many relationship cannot be created. Well, it's true for just one of the relationships. Let's explore how the **Target** table can be related to the dimension-type tables.

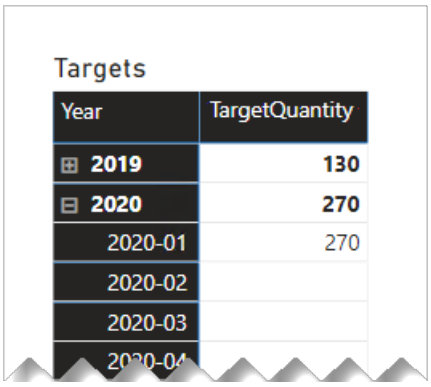
Relate higher grain time periods

A relationship between the **Date** and **Target** tables should be a one-to-many relationship. It's because the **TargetYear** column values are dates. In this example, each **TargetYear** column value is the first date of the target year.

> **TIP**
>
> When storing facts at a higher time granularity than day, set the column data type to **Date** (or **Whole number** if you're using date keys). In the column, store a value representing the first day of the time period. For example, a year period is recorded as January 1 of the year, and a month period is recorded as the first day of that month.

Care must be taken, however, to ensure that month or date level filters produce a meaningful result. Without any special calculation logic, report visuals may report that target dates are literally the first day of each year. All other days—and all months except January—will summarize the target quantity as BLANK.

The following matrix visual shows what happens when the report user drills from a year into its months. The visual is summarizing the **TargetQuantity** column. (The Show items with no data option has been enabled for the matrix rows.)

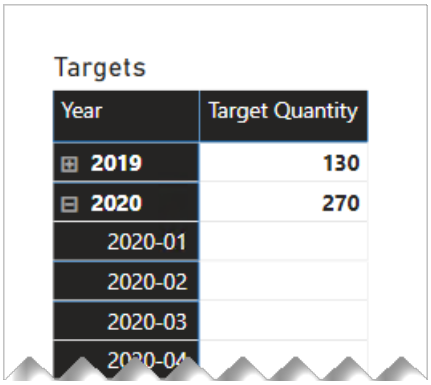


To avoid this behavior, we recommend you control the summarization of your fact data by using measures. One way to control the summarization is to return BLANK when lower-level time periods are queried. Another way—defined with some sophisticated DAX—is to apportion values across lower-level time periods.

Consider the following measure definition that uses the ISFILTERED DAX function. It only returns a value when the **Date** or **Month** columns aren't filtered.

```
Target Quantity =
IF(
    NOT ISFILTERED('Date'[Date])
        && NOT ISFILTERED('Date'[Month]),
    SUM(Target[TargetQuantity])
)
```

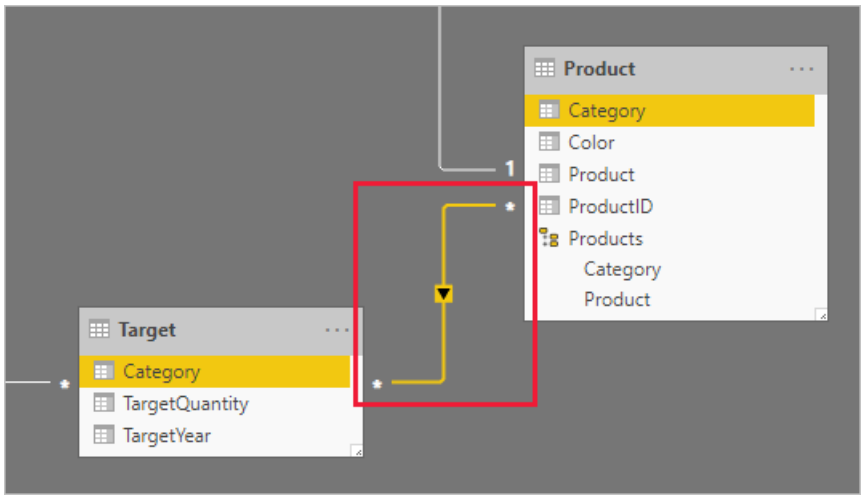
The following matrix visual now uses the **Target Quantity** measure. It shows that all monthly target quantities are BLANK.



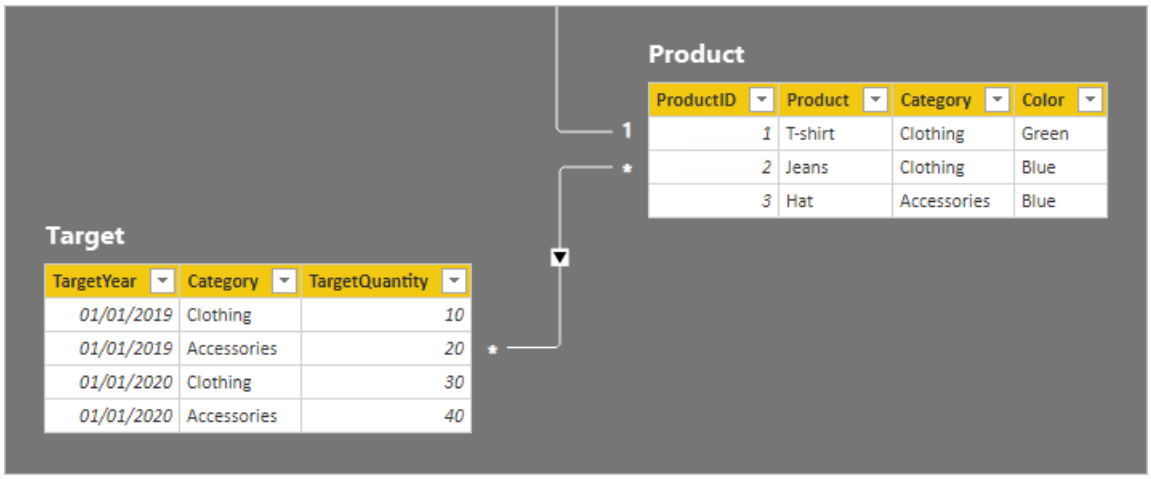
Relate higher grain (non-date)

A different design approach is required when relating a non-date column from a dimension-type table to a fact-type table (and it's at a higher grain than the dimension-type table).

The **Category** columns (from both the **Product** and **Target** tables) contains duplicate values. So, there's no "one" for a one-to-many relationship. In this case, you'll need to create a many-to-many relationship. The relationship should propagate filters in a single direction, from the dimension-type table to the fact-type table.



Let's now take a look at the table rows.



In the **Target** table, there are four rows: two rows for each target year (2019 and 2020), and two categories (Clothing and Accessories). In the **Product** table, there are three products. Two belong to the clothing category, and one belongs to the accessories category. One of the clothing colors is green, and the remaining two are blue.

A table visual grouping by the **Category** column from the **Product** table produces the following result.

Category Targets

Category	TargetQuantity
Accessories	60
Clothing	40
Total	**100**

This visual produces the correct result. Let's now consider what happens when the **Color** column from the **Product** table is used to group target quantity.

Color Targets?

Color	TargetQuantity
Blue	100
Green	40
Total	**100**

The visual produces a misrepresentation of the data. What is happening here?

A filter on the **Color** column from the **Product** table results in two rows. One of the rows is for the Clothing category, and the other is for the Accessories category. These two category values are propagated as filters to the **Target** table. In other words, because the color blue is used by products from two categories, *those categories* are used to filter the targets.

To avoid this behavior, as described earlier, we recommend you control the summarization of your fact data by using measures.

Consider the following measure definition. Notice that all **Product** table columns that are beneath the category level are tested for filters.

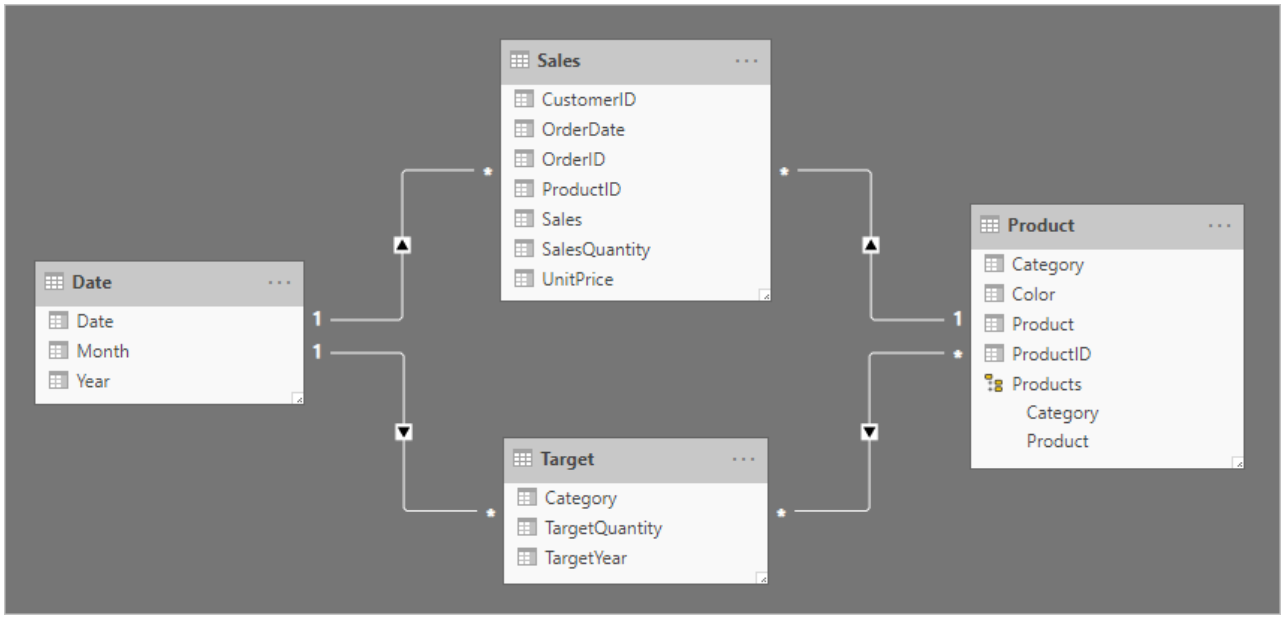
```
Target Quantity =
IF(
    NOT ISFILTERED('Product'[ProductID])
        && NOT ISFILTERED('Product'[Product])
        && NOT ISFILTERED('Product'[Color]),
    SUM(Target[TargetQuantity])
)
```

The following table visual now uses the **Target Quantity** measure. It shows that all color target quantities are BLANK.

Color Targets?

Color	Target Quantity
Blue	
Green	
Total	**100**

The final model design looks like the following.



Relate higher grain facts guidance

When you need to relate a dimension-type table to a fact-type table, and the fact-type table stores rows at a higher grain than the dimension-type table rows, we provide the following guidance:

- For higher grain fact dates:

    - In the fact-type table, store the first date of the time period
    - Create a one-to-many relationship between the date table and the fact-type table
- For other higher grain facts:
    - Create a many-to-many relationship between the dimension-type table and the fact-type table
- For both types:
    - Control summarization with measure logic—return BLANK when lower-level dimension-type columns are used to filter or group
    - Hide summarizable fact-type table columns—this way, only measures can be used to summarize the fact-type table

Next steps

For more information related to this article, check out the following resources:

- Model relationships in Power BI Desktop
- Understand star schema and the importance for Power BI
- Relationship troubleshooting guidance
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Active vs inactive relationship guidance

5/13/2020 • 6 minutes to read • Edit Online

This article targets you as a data modeler working with Power BI Desktop. It provides you with guidance on when to create active or inactive model relationships. By default, active relationships propagate filters to other tables. Inactive relationship, however, only propagate filters when a DAX expression activates (uses) the relationship.

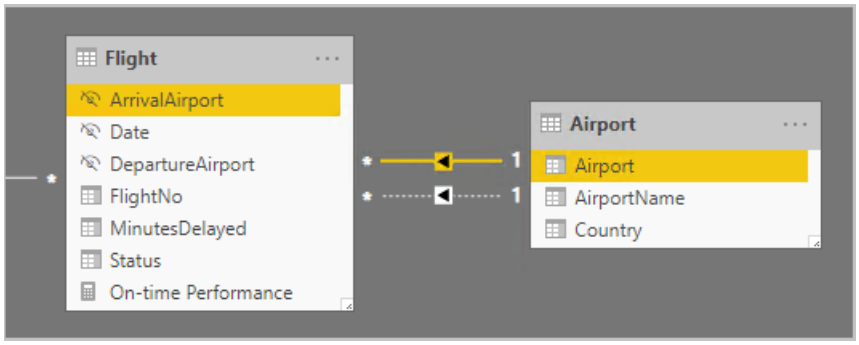
> **NOTE**
>
> An introduction to model relationships is not covered in this article. If you're not completely familiar with relationships, their properties or how to configure them, we recommend that you first read the Model relationships in Power BI Desktop article.
>
> It's also important that you have an understanding of star schema design. For more information, see Understand star schema and the importance for Power BI.

Active relationships

Generally, we recommend defining active relationships whenever possible. They widen the scope and potential of how your model can be used by report authors, and users working with Q&A.

Consider an example of an Import model designed to analyze airline flight on-time performance (OTP). The model has a **Flight** table, which is a fact-type table storing one row per flight. Each row records the flight date, flight number, departure and arrival airports, and any delay time (in minutes). There's also an **Airport** table, which is a dimension-type table storing one row per airport. Each row describes the airport code, airport name, and the country.

Here's a partial model diagram of the two tables.



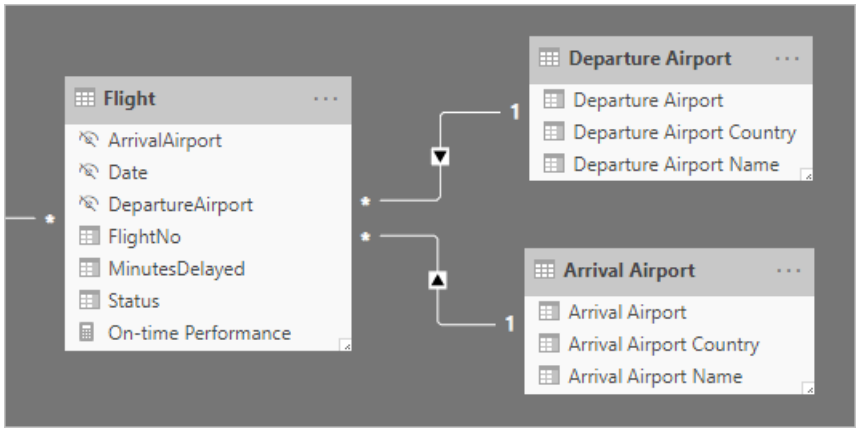
There are two model relationships between the **Flight** and **Airport** tables. In the **Flight** table, the **DepartureAirport** and **ArrivalAirport** columns relate to the **Airport** column of the **Airport** table. In star schema design, the **Airport** table is described as a role-playing dimension. In this model, the two roles are *departure airport* and *arrival airport*.

While this design works well for relational star schema designs, it doesn't for Power BI models. It's because model relationships are paths for filter propagation, and these paths must be deterministic. For this reason, a model cannot have multiple active relationships between two tables. Therefore—as described in this example—one relationship is active while the other is inactive (represented by the dashed line). Specifically, it's the relationship to the **ArrivalAirport** column that's active. This means filters applied to the **Airport** table automatically propagate to the **ArrivalAirport** column of the **Flight** table.

This model design imposes severe limitations on how the data can be reported. Specifically, it's not possible to filter the **Airport** table to automatically isolate flight details for a departure airport. As reporting requirements involve

filtering (or grouping) by departure and arrival airports *at the same time*, two active relationships are needed. Translating this requirement into a Power BI model design means the model must have two airport tables.

Here's the improved model design.



The model now has two airport tables: **Departure Airport** and **Arrival Airport**. The model relationships between these tables and the **Flight** table are active. Notice also that the column names in the **Departure Airport** and **Arrival Airport** tables are prefixed with the word *Departure* or *Arrival*.

The improved model design supports producing the following report design.

Month

2019 December

Departure Airport

Melbourne

On-time Departures

Arrival Airport	Flights	On-time Performance
Perth	61	99.12%
Adelaide	30	98.21%
Brisbane	62	95.48%
Canberra	75	95.05%
Hobart	29	94.14%
Sydney	119	92.31%
Darwin	28	91.75%
Total	**404**	**94.87%**

The report page filters by Melbourne as the departure airport, and the table visual groups by arrival airports.

> **NOTE**
>
> For Import models, the additional table has resulted in an increased model size, and longer refresh times. As such, it contradicts the recommendations described in the Data reduction techniques for Import modeling article. However, in the example, the requirement to have only active relationships overrides these recommendations.
>
> Further, it's common that dimension-type tables contain low row counts relative to fact-type table row counts. So, the increased model size and refresh times aren't likely to be excessively large.

Refactoring methodology

Here's a methodology to refactor a model from a single role-playing dimension-type table, to a design with *one table per role*.

1. Remove any inactive relationships.
2. Consider renaming the role-playing dimension-type table to better describe its role. In the example, the **Airport** table is related to the **ArrivalAirport** column of the **Flight** table, so it's renamed as **Arrival Airport**.

3. Create a copy of the role-playing table, providing it with a name that reflects its role. If it's an Import table, we recommend defining a calculated table. If it's a DirectQuery table, you can duplicate the Power Query query.

   In the example, the **Departure Airport** table was created by using the following calculated table definition.

   ```
   Departure Airport = 'Arrival Airport'
   ```

4. Create an active relationship to relate the new table.

5. Consider renaming the columns in the tables so they accurately reflect their role. In the example, all columns are prefixed with the word *Departure* or *Arrival*. These names ensure report visuals, by default, will have self-describing and non-ambiguous labels. It also improves the Q&A experience, allowing users to easily write their questions.

6. Consider adding descriptions to role-playing tables. (In the **Fields** pane, a description appears in a tooltip when a report author hovers their cursor over the table.) This way, you can communicate any additional filter propagation details to your report authors.

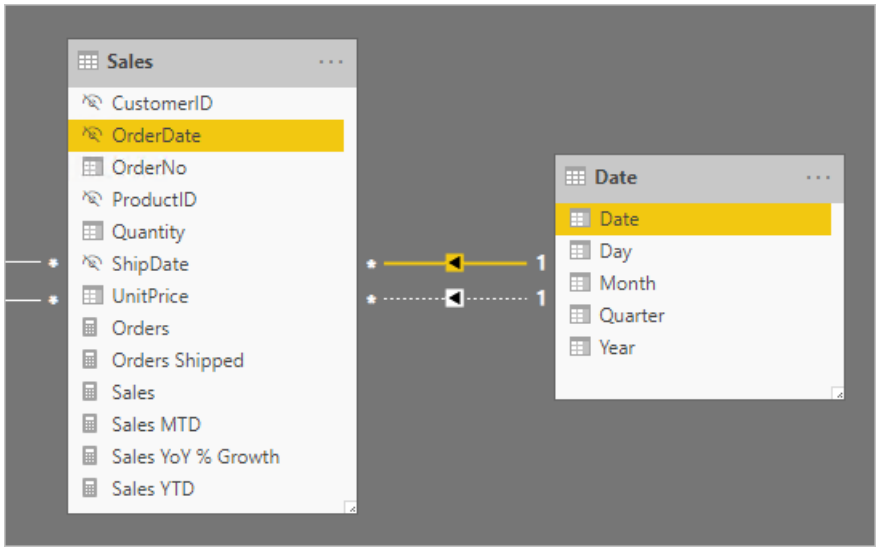
Inactive relationships

In specific circumstances, inactive relationships can address special reporting needs.

Let's now consider different model and reporting requirements:

- A sales model contains a **Sales** table that has two date columns: **OrderDate** and **ShipDate**
- Each row in the **Sales** table records a single order
- Date filters are almost always applied to the **OrderDate** column, which always stores a valid date
- Only one measure requires date filter propagation to the **ShipDate** column, which can contain BLANKs (until the order is shipped)
- There's no requirement to simultaneously filter (or group by) order *and* ship date periods

Here's a partial model diagram of the two tables.



There are two model relationships between the **Sales** and **Date** tables. In the **Sales** table, the **OrderDate** and **ShipDate** columns relate to the **Date** column of the **Date** table. In this model, the two roles for the **Date** table are *order date* and *ship date*. It's the relationship to the **OrderDate** column that's active.

All of the six measures—except one—must filter by the **OrderDate** column. The **Orders Shipped** measure, however, must filter by the **ShipDate** column.

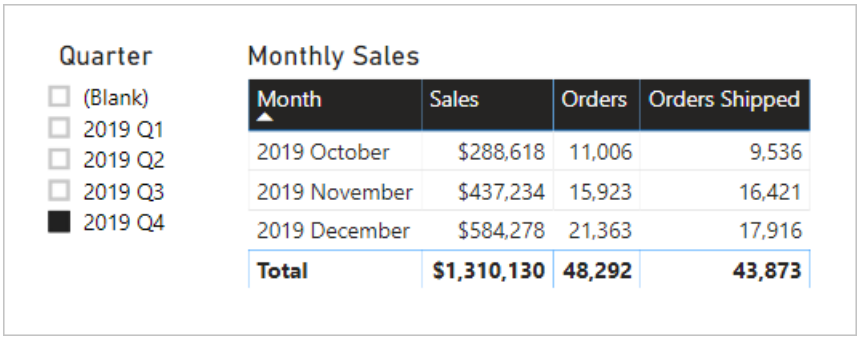
Here's the **Orders** measure definition. It simply counts the rows of the **Sales** table within the filter context. Any filters applied to the **Date** table will propagate to the **OrderDate** column.

```
Orders = COUNTROWS(Sales)
```

Here's the **Orders Shipped** measure definition. It uses the USERELATIONSHIP DAX function, which activates filter propagation for a specific relationship only during the evaluation of the expression. In this example, the relationship to the **ShipDate** column is used.

```
Orders Shipped =
CALCULATE(
    COUNTROWS(Sales)
    ,USERELATIONSHIP('Date'[Date], Sales[ShipDate])
)
```

This model design supports producing the following report design.



The report page filters by quarter 2019 Q4. The table visual groups by month and displays various sales statistics. The **Orders** and **Orders Shipped** measures produce different results. They each use the same summarization logic (count rows of the **Sales** table), but different **Date** table filter propagation.

Notice that the quarter slicer includes a BLANK item. This slicer item appears as a result of table expansion. While each **Sales** table row has an order date, some rows have a BLANK ship date—these orders are yet to be shipped. Table expansion considers inactive relationships too, and so BLANKs can appear due to BLANKs on the many-side of the relationship, or due to data integrity issues.

Recommendations

In summary, we recommend defining active relationships whenever possible. They widen the scope and potential of how your model can be used by report authors, and users working with Q&A. It means that role-playing dimension-type tables should be duplicated in your model.

In specific circumstances, however, you can define one or more inactive relationships for a role-playing dimension-type table. You can consider this design when:

- There's no requirement for report visuals to simultaneously filter by different roles
- You use the USERELATIONSHIP DAX function to activate a specific relationship for relevant model calculations

Next steps

For more information related to this article, check out the following resources:

- Model relationships in Power BI Desktop
- Understand star schema and the importance for Power BI
- Relationship troubleshooting guidance

- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Bi-directional relationship guidance

5/13/2020 • 5 minutes to read • Edit Online

This article targets you as a data modeler working with Power BI Desktop. It provides you with guidance on when to create bi-directional model relationships. A bi-directional relationship is one that filters in *both directions*.

> **NOTE**
>
> An introduction to model relationships is not covered in this article. If you're not completely familiar with relationships, their properties or how to configure them, we recommend that you first read the Model relationships in Power BI Desktop article.
>
> It's also important that you have an understanding of star schema design. For more information, see Understand star schema and the importance for Power BI.

Generally, we recommend minimizing the use of bi-directional relationships. They can negatively impact on model query performance, and possibly deliver confusing experiences for your report users.

There are three scenarios when bi-directional filtering can solve specific requirements:

- Special model relationships
- Slicer items "with data"
- Dimension-to-dimension analysis

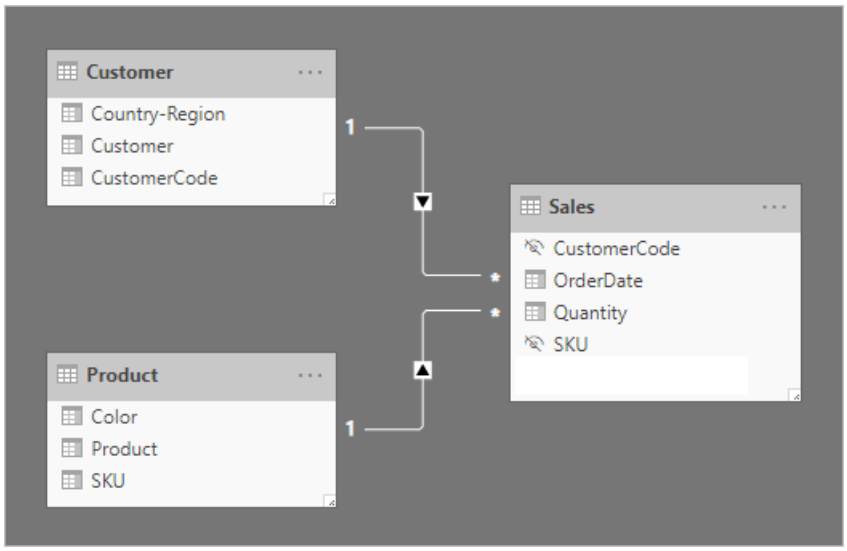
Special model relationships

Bi-directional relationships play an important role when creating the following two special model relationship types:

- **One-to-one**: All one-to-one relationships must be bi-directional—it isn't possible to configure otherwise. Generally, we don't recommend creating these types of relationships. For a complete discussion and alternative designs, see One-to-one relationship guidance.
- **Many-to-many**: When relating two dimension-type tables, a bridging table is required. A bi-directional filter is required to ensure filters propagate across the bridging table. For more information, see Many-to-many relationship guidance (Relate many-to-many dimensions).

Slicer items "with data"

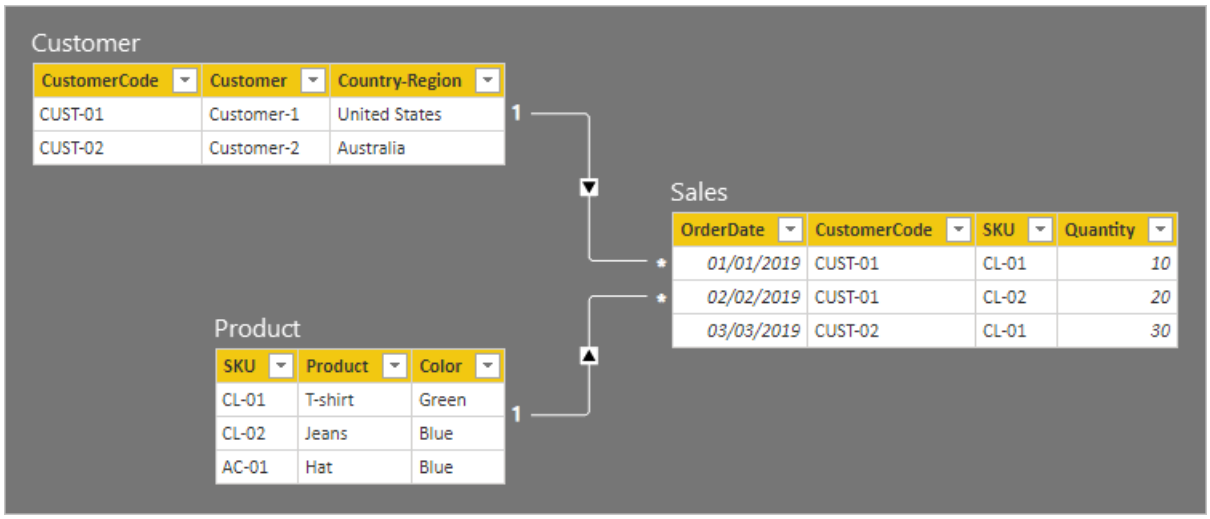
Bi-directional relationships can deliver slicers that limit items to where data exists. (If you're familiar with Excel PivotTables and slicers, it's the default behavior when sourcing data from a Power BI dataset, or an Analysis Services model.) To help explain what it means, first consider the following model diagram.



The first table is named **Customer**, and it contains three columns: **Country-Region**, **Customer**, and **CustomerCode**. The second table is named **Product**, and it contains three columns: **Color**, **Product**, and **SKU**. The third table is named **Sales**, and it contains four columns: **CustomerCode**, **OrderDate**, **Quantity**, and **SKU**. The **Customer** and **Product** tables are dimension-type tables, and each has a one-to-many relationship to the **Sales** table. Each relationship filters in a single direction.

To help describe how bi-directional filtering works, the model diagram has been modified to reveal the table rows. All examples in this article are based on this data.

> **NOTE**
> It's not possible to display table rows in the Power BI Desktop model diagram. It's done in this article to support the discussion with clear examples.



The row details for the three tables are described in the following bulleted list:

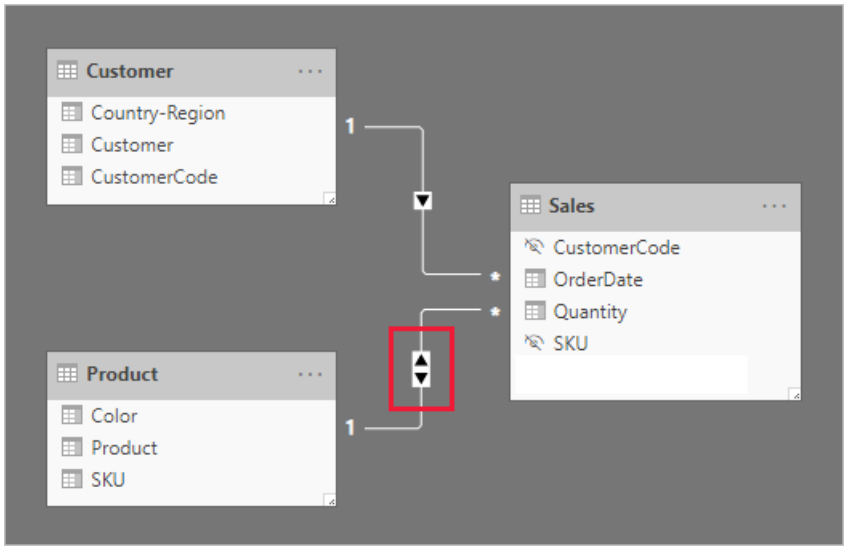
- The **Customer** table has two rows:
  - **CustomerCode** CUST-01, **Customer** Customer-1, **Country-Region** United States
  - **CustomerCode** CUST-02, **Customer** Customer-2, **Country-Region** Australia
- The **Product** table has three rows:
  - **SKU** CL-01, **Product** T-shirt, **Color** Green
  - **SKU** CL-02, **Product** Jeans, **Color** Blue
  - **SKU** AC-01, **Product** Hat, **Color** Blue
- The **Sales** table has three rows:

- **OrderDate** January 1 2019, **CustomerCode** CUST-01, **SKU** CL-01, **Quantity** 10
- **OrderDate** February 2 2019, **CustomerCode** CUST-01, **SKU** CL-02, **Quantity** 20
- **OrderDate** March 3 2019, **CustomerCode** CUST-02, **SKU** CL-01, **Quantity** 30

Now consider the following report page.

The page consists of two slicers and a card visual. The first slicer is for **Country-Region** and it has two items: Australia and United States. It currently slices by Australia. The second slicer is for **Product**, and it has three items: Hat, Jeans, and T-shirt. No items are selected (meaning *no products* are filtered). The card visual displays a quantity of 30.

When report users slice by Australia, you might want to limit the **Product** slicer to display items where data *relates* to Australian sales. It's what's meant by showing slicer items "with data". You can achieve this behavior by configuring the relationship between the **Product** and **Sales** table to filter in both directions.



The **Product** slicer now lists a single item: T-shirt. This item represents the only product sold to Australian customers.

We first suggest you consider carefully whether this design works for your report users. Some report users find the experience confusing. They don't understand why slicer items dynamically appear or disappear when they interact with other slicers.

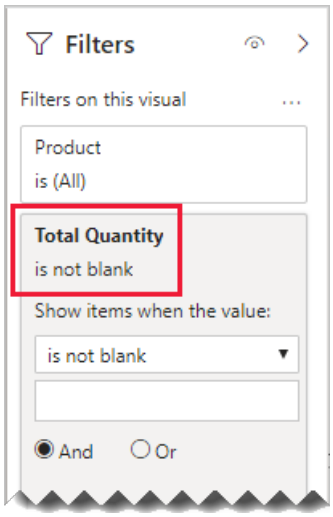
If you do decide to show slicer items "with data", we don't recommend you configure bi-directional relationships. Bi-directional relationships require more processing and so they can negatively impact on query performance—especially as the number of bi-directional relationships in your model increases.

There's a better way to achieve the same result: Instead of using bi-directional filters, you can apply a visual-level filter to the **Product** slicer itself.

Let's now consider that the relationship between the **Product** and **Sales** table no longer filters in both directions. And, the following measure definition has been added to the **Sales** table.

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

To show the **Product** slicer items "with data", it simply needs to be filtered by the **Total Quantity** measure using the "is not blank" condition.



Dimension-to-dimension analysis

A different scenario involving bi-directional relationships treats a fact-type table like a bridging table. This way, it supports analyzing dimension-type table data within the filter context of a different dimension-type table.

Using the example model in this article, consider how the following questions can be answered:

- How many colors were sold to Australian customers?
- How many countries purchased jeans?

Both questions can be answered *without* summarizing data in the bridging fact-type table. They do, however, require that filters propagate from one dimension-type table to the other. Once filters propagate via the fact-type table, summarization of dimension-type table columns can be achieved using the DISTINCTCOUNT DAX function—and possibly the MIN and MAX DAX functions.

As the fact-type table behaves like a bridging table, you can follow the many-to-many relationship guidance to relate two dimension-type tables. It will require configuring at least one relationship to filter in both directions. For more information, see Many-to-many relationship guidance (Relate many-to-many dimensions).

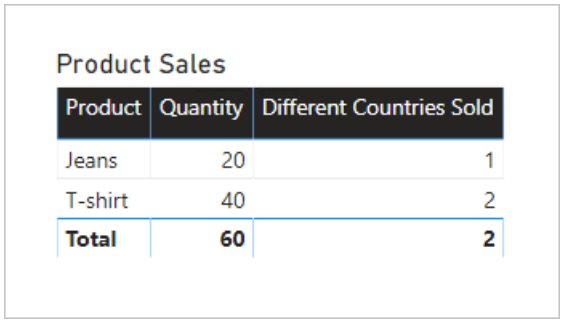
However, as already described in this article, this design will likely result in a negative impact on performance, and the user experience consequences related to slicer items "with data". So, we recommend that you activate bi-directional filtering *in a measure definition* by using the CROSSFILTER DAX function instead. The CROSSFILTER function can be used to modify filter directions—or even disable the relationship—during the evaluation of an expression.

Consider the following measure definition added to the **Sales** table. In this example, the model relationship between the **Customer** and **Sales** tables has been configured to filter in a *single direction*.

```
Different Countries Sold =
CALCULATE(
    DISTINCTCOUNT(Customer[Country-Region]),
    CROSSFILTER(
        Customer[CustomerCode],
        Sales[CustomerCode],
        BOTH
    )
)
```

During the evaluation of the **Different Countries Sold** measure expression, the relationship between the **Customer** and **Sales** tables filters in both directions.

The following table visual present statistics for each product sold. The **Quantity** column is simply the sum of quantity values. The **Different Countries Sold** column represents the distinct count of country-region values of all customers who have purchased the product.



Product Sales

Product	Quantity	Different Countries Sold
Jeans	20	1
T-shirt	40	2
Total	**60**	**2**

Next steps

For more information related to this article, check out the following resources:

- Model relationships in Power BI Desktop
- Understand star schema and the importance for Power BI
- One-to-one relationship guidance
- Many-to-many relationship guidance
- Relationship troubleshooting guidance
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Relationship troubleshooting guidance

5/13/2020 • 3 minutes to read • Edit Online

This article targets you as a data modeler working with Power BI Desktop. It provides you with guidance on how to troubleshoot specific issues you may encounter when developing models and reports.

> **NOTE**
>
> An introduction to model relationships is not covered in this article. If you're not completely familiar with relationships, their properties or how to configure them, we recommend that you first read the Model relationships in Power BI Desktop article.
>
> It's also important that you have an understanding of star schema design. For more information, see Understand star schema and the importance for Power BI.

Troubleshooting checklist

When a report visual is configured to use fields from two (or more) tables, and it doesn't present the correct result (or any result), it's possible that the issue is related to model relationships.

In this case, here's a general troubleshooting checklist to follow. You can progressively work through the checklist until you identify the issue(s).

1. Switch the visual to a table or matrix, or open the "See Data" pane—it's easier to troubleshoot issues when you can see the query result
2. If there's an empty query result, switch to Data view—verify that tables have been loaded with rows of data
3. Switch to Model view—it's easy to see the relationships and quickly determine their properties
4. Verify that relationships exist between the tables
5. Verify that cardinality properties are correctly configured—they could be incorrect if a "many"-side column presently contains unique values, and has been incorrectly configured as a "one"-side
6. Verify that the relationships are active (solid line)
7. Verify that the filter directions support propagation (interpret arrow heads)
8. Verify that the correct columns are related—either select the relationship, or hover the cursor over it, to reveal the related columns
9. Verify that the related column data types are the same, or at least compatible—it's possible to relate a text column to a whole number column, but filters won't find any matches to propagate
10. Switch to Data view, and verify that matching values can be found in related columns

Troubleshooting guide

Here's a list of issues together with possible solutions.

ISSUE	POSSIBLE REASON(S)

ISSUE	POSSIBLE REASON(S)
The visual displays no result	- The model is yet to be loaded with data - No data exists within the filter context - Row-level security is enforced - Relationships aren't propagating between tables—*follow checklist above* - Row-level security is enforced, but a bi-directional relationship isn't enabled to propagate—see Row-level security (RLS) with Power BI Desktop
The visual displays the same value for each grouping	- Relationships don't exist - Relationships aren't propagating between tables—*follow checklist above*
The visual displays results, but they aren't correct	- Visual is incorrectly configured - Measure logic is incorrect - Model data needs to be refreshed - Source data is incorrect - Relationship columns are incorrectly related (for example, **ProductID** column maps to **CustomerID**) - It's a relationship between two DirectQuery tables, and the "one"-side column of a relationship contains duplicate values
BLANK groupings or slicer/filter items appear, and the source columns don't contain BLANKs	- It's a strong relationship, and "many"-side column contain values not stored in the "one"-side column—see Model relationships in Power BI Desktop (Strong relationships) - It's a strong one-to-one relationship, and related columns contain BLANKs—see Model relationships in Power BI Desktop (Strong relationships) - An inactivate relationship "many"-side column stores BLANKs, or has values not stored on the "one"-side
The visual is missing data	- Incorrect/unexpected filters are applied - Row-level security is enforced - It's a weak relationship, and there are BLANKs in related columns, or data integrity issues—see Model relationships in Power BI Desktop (Weak relationships) - It's a relationship between two DirectQuery tables, the relationship is configured to assume referential integrity, but there are data integrity issues (mismatched values in related columns)
Row-level security is not correctly enforced	- Relationships aren't propagating between tables—*follow checklist above* - Row-level security is enforced, but a bi-directional relationship isn't enabled to propagate—see Row-level security (RLS) with Power BI Desktop

Next steps

For more information related to this article, check out the following resources:

- Model relationships in Power BI Desktop
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

DirectQuery model guidance in Power BI Desktop

5/13/2020 • 16 minutes to read • Edit Online

This article targets data modelers developing Power BI DirectQuery models, developed by using either Power BI Desktop or the Power BI service. It describes DirectQuery use cases, limitations, and guidance. Specifically, the guidance is designed to help you determine whether DirectQuery is the appropriate mode for your model, and to improve the performance of your reports based on DirectQuery models. This article applies to DirectQuery models hosted in the Power BI service or Power BI Report Server.

This article is not intended to provide a complete discussion on DirectQuery model design. For an introduction, refer to the DirectQuery models in Power BI Desktop article. For a deeper discussion, refer directly to the DirectQuery in SQL Server 2016 Analysis Services whitepaper. Bear in mind that the whitepaper describes using DirectQuery in SQL Server Analysis Services. Much of the content, however, is still applicable to Power BI DirectQuery models.

This article does not directly cover Composite models. A Composite model will consist of at least one DirectQuery source, and possibly more. The guidance described in this article is still relevant—at least in part—to Composite model design. However, the implications of combining Import tables with DirectQuery tables are not in scope for this article. For more information, see Use composite models in Power BI Desktop.

It is important to understand that DirectQuery models impose a different workload on the Power BI environment (Power BI service or Power BI Report Server) and also on the underlying data sources. If you determine that DirectQuery is the appropriate design approach, we recommend that you engage the right people on the project. We often see that a successful DirectQuery model deployment is the result of a team of IT professionals working closely together. The team usually consists of model developers and the source database administrators. It can also involve data architects, and data warehouse and ETL developers. Often, optimizations need to be applied directly to the data source to achieve good performance results.

Design in Power BI Desktop

Both Azure SQL Data Warehouse and Azure HDInsight Spark data sources can be connected to directly, without the need to use Power BI Desktop. It is achieved in the Power BI service by "Getting Data" and choosing the Databases tile. For more information, see Azure SQL Data Warehouse with DirectQuery.

While direct connect is convenient, we don't recommend that you use this approach. The main reason is that it is not possible to refresh the model structure should the underlying data source schema change.

We recommend that you use Power BI Desktop to create and manage all of your DirectQuery models. This approach provides you with complete control to define the model that you need, including the use of supported features like hierarchies, calculated columns, measures, and more. It will also allow you to revise the model design should the underlying data source schema change.

Optimize data source performance

The relational database source can be optimized in several ways, as described in the following bulleted list.

> **NOTE**
>
> We understand that not all modelers have the permissions or skills to optimize a relational database. While it is the preferred layer to prepare the data for a DirectQuery model, some optimizations can also be achieved in the model design, without modifying the source database. However, best optimization results are often achieved by applying optimizations to the source database.

- **Ensure data integrity is complete:** It is especially important that dimension-type tables contain a column of unique values (dimension key) that maps to the fact-type table(s). It's also important that fact-type dimension columns contain valid dimension key values. They will allow configuring more efficient model relationships that expect matched values on both sides of relationships. When the source data lacks integrity, it's recommended that an "unknown" dimension record is added to effectively repair the data. For example, you can add a row to the **Product** table to represent an unknown product, and then assign it an out-of-range key, like -1. If rows in the **Sales** table contain a missing product key value, substitute them with -1. It will ensure every **Sales** product key value has a corresponding row in the **Product** table.

- **Add indexes:** Define appropriate indexes—on tables or views—to support the efficient retrieval of data for the expected report visual filtering and grouping. For SQL Server, Azure SQL Database or Azure SQL Data Warehouse sources, see SQL Server Index Architecture and Design Guide for helpful information on index design guidance. For SQL Server or Azure SQL Database volatile sources, see Get started with Columnstore for real-time operational analytics.

- **Design distributed tables:** For Azure SQL Data Warehouse sources, which leverage Massively Parallel Processing (MPP) architecture, consider configuring large fact-type tables as hash distributed, and dimension-type tables to replicate across all the compute nodes. For more information, see Guidance for designing distributed tables in Azure SQL Data Warehouse.

- **Ensure required data transformations are materialized:** For SQL Server relational database sources (and other relational database sources), computed columns can be added to tables. These columns are based on an expression, like **Quantity** multiplied by **UnitPrice**. Computed columns can be persisted (materialized) and, like regular columns, sometimes they can be indexed. For more information, see Indexes on Computed Columns.

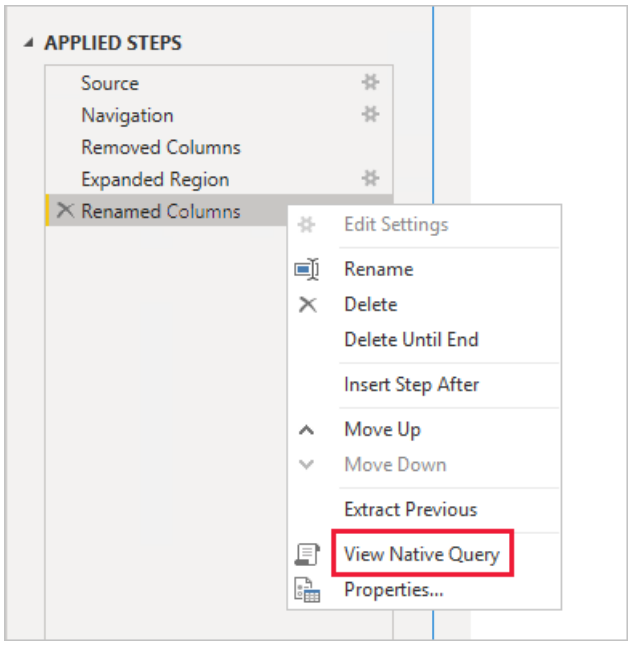
  Consider also indexed views that can pre-aggregate fact table data at a higher grain. For example, if the **Sales** table stores data at order line level, you could create a view to summarize this data. The view could be based on a SELECT statement that groups the **Sales** table data by date (at month level), customer, product, and summarizes measure values like sales, quantity, etc. The view can then be indexed. For SQL Server or Azure SQL Database sources, see Create Indexed Views.

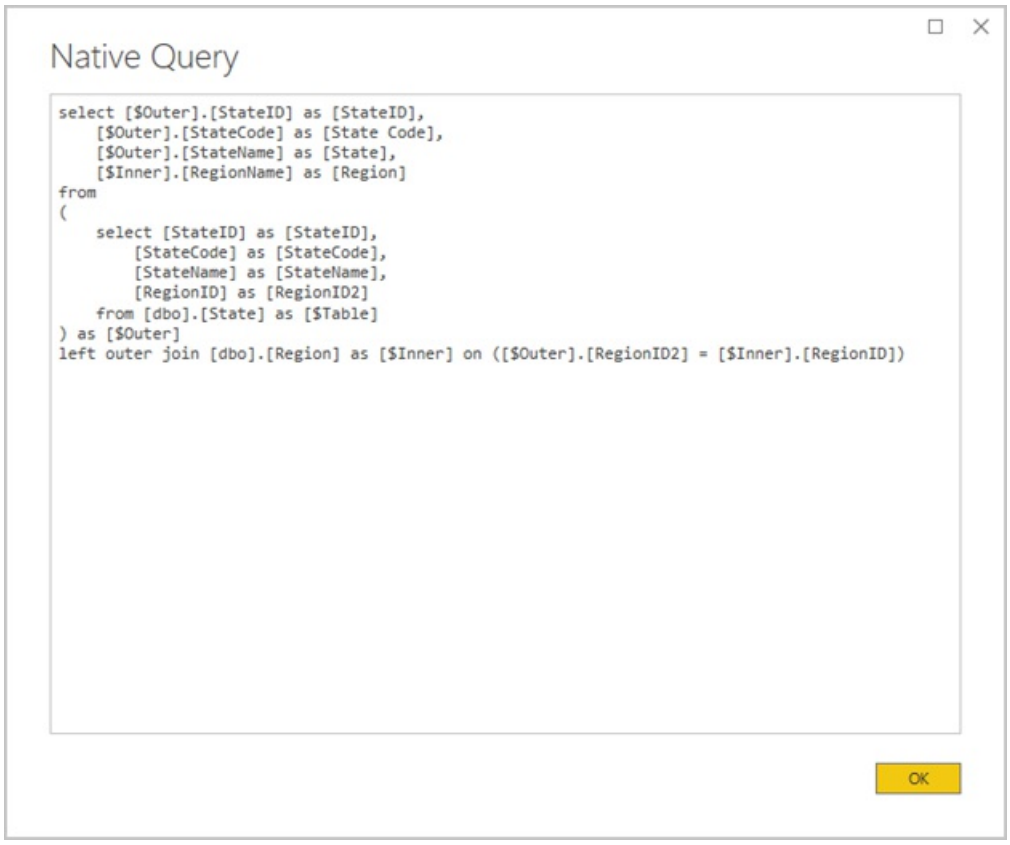
- **Materialize a date table:** A common modeling requirement involves adding a date table to support time-based filtering. To support the known time-based filters in your organization, create a table in the source database, and ensure it is loaded with a range of dates encompassing the fact table dates. Also ensure that it includes columns for useful time periods, like year, quarter, month, week, etc.

Optimize model design

A DirectQuery model can be optimized in many ways, as described in the following bulleted list.

- **Avoid complex Power Query queries:** An efficient model design can be achieved by removing the need for the Power Query queries to apply any transformations. It means that each query maps to a single relational database source table or view. You can preview a representation of the actual SQL query statement for a Power Query applied step, by selecting the **View Native Query** option.





- **Examine the use of calculated columns and data type changes:** DirectQuery models support adding calculations and Power Query steps to convert data types. However, better performance is often achieved by materializing transformation results in the relational database source, when possible.

- **Do not use Power Query relative date filtering:** It's possible to define relative date filtering in a Power Query query. For example, to retrieve to the sales orders that were created in the last year (relative to today's date). This type of filter translates to an inefficient native query, as follows:

```
…
from [dbo].[Sales] as [_]
where [_].[OrderDate] >= convert(datetime2, '2018-01-01 00:00:00') and [_].[OrderDate] <
convert(datetime2, '2019-01-01 00:00:00'))
```

A better design approach is to include relative time columns in the date table. These columns store offset

values relative to the current date. For example, in a **RelativeYear** column, the value zero represents current year, -1 represents previous year, etc. Preferably, the **RelativeYear** column is materialized in the date table. While less efficient, it could also be added as a model calculated column, based on the expression using the TODAY and DATE DAX functions.

- **Keep measures simple:** At least initially, it's recommended to limit measures to simple aggregates. The aggregate functions include SUM, COUNT, MIN, MAX, and AVERAGE. Then, if the measures are sufficiently responsive, you can experiment with more complex measures, but paying attention to the performance for each. While the CALCULATE DAX function can be used to produce sophisticated measure expressions that manipulate filter context, they can generate expensive native queries that do not perform well.

- **Avoid relationships on calculated columns:** Model relationships can only relate a single column in one table to a single column in a different table. Sometimes, however, it is necessary to relate tables by using multiple columns. For example, the **Sales** and **Geography** tables are related by two columns: **Country** and **City**. To create a relationship between the tables, a single column is required, and in the **Geography** table, the column must contain unique values. Concatenating the country and city with a hyphen separator could achieve this result.

  The combined column can be created with either a Power Query custom column, or in the model as a calculated column. However, it should be avoided as the calculation expression will be embedded into the source queries. Not only is it inefficient, it commonly prevents the use of indexes. Instead, add materialized columns in the relational database source, and consider indexing them. You can also consider adding surrogate key columns to dimension-type tables, which is a common practice in relational data warehouse designs.

  There is one exception to this guidance, and it concerns the use of the COMBINEVALUES DAX function. The purpose of this function is to support multi-column model relationships. Rather than generate an expression that the relationship uses, it generates a multi-column SQL join predicate.

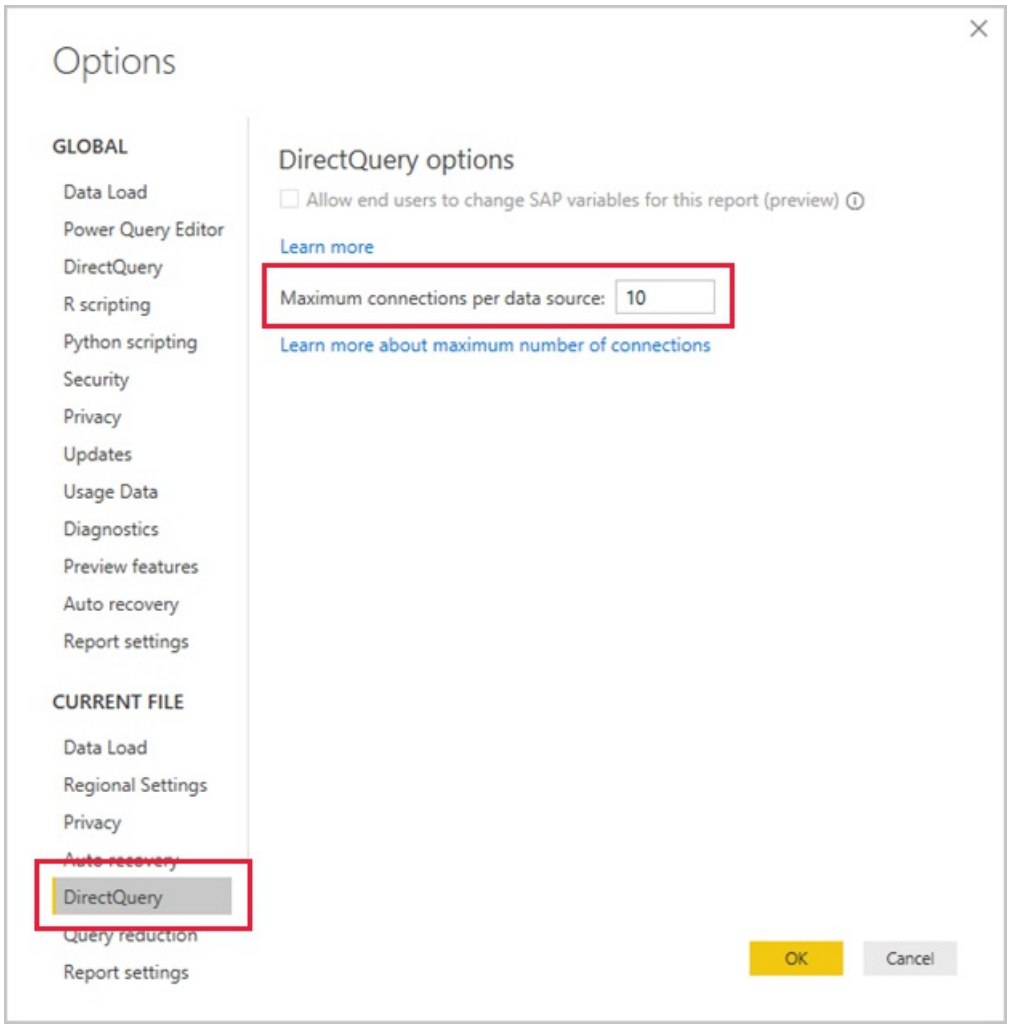
- **Avoid relationships on "Unique Identifier" columns:** Power BI does not natively support the unique identifier (GUID) data type. When defining a relationship between columns of this type, Power BI will generate a source query with a join involving a cast. This query-time data conversion commonly results in poor performance. Until this case is optimized, the only workaround is to materialize columns of an alternative data type in the underlying database.

- **Hide the one-side column of relationships:** The one-side column of a relationship should be hidden. (It is usually the primary key column of dimension-type tables.) When hidden, it is not available in the **Fields** pane and so cannot be used to configure a visual. The many-side column can remain visible if it is useful to group or filter reports by the column values. For example, consider a model where a relationship exists between **Sales** and **Product** tables. The relationship columns contain product SKU (Stock-Keeping Unit) values. If product SKU must be added to visuals, it should be visible only in the **Sales** table. When this column is used to filter or group in a visual, Power BI will generate a query that does not need to join the **Sales** and **Product** tables.

- **Set relationships to enforce integrity:** The **Assume Referential Integrity** property of DirectQuery relationships determines whether Power BI will generate source queries using an inner join rather than an outer join. It generally improves query performance, though it does depend on the specifics of the relational database source. For more information, see Assume referential integrity settings in Power BI Desktop.

- **Avoid use of bi-directional relationship filtering:** Use of bi-directional relationship filtering can lead to query statements that don't perform well. Only use this relationship feature when necessary, and it's usually the case when implementing a many-to-many relationship across a bridging table. For more information, see Relationships with a many-many cardinality in Power BI Desktop.

- **Limit parallel queries:** You can set the maximum number of connections DirectQuery opens for each underlying data source. It controls the number of queries concurrently sent to the data source.



The setting is only enabled when there's at least one DirectQuery source in the model. The value applies to all DirectQuery sources, and to any new DirectQuery sources added to the model.

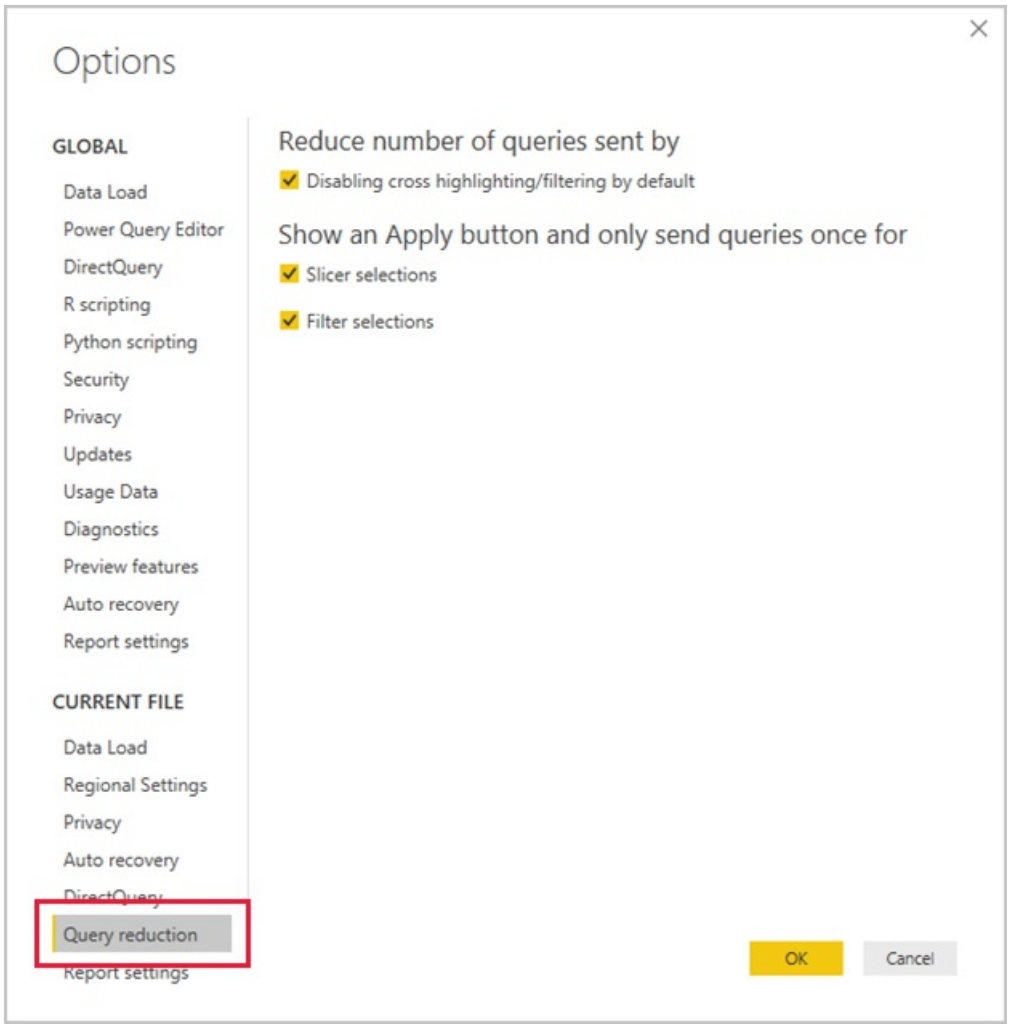
Increasing the **Maximum Connections per Data Source** value ensures more queries (up to the maximum number specified) can be sent to the underlying data source, which is useful when numerous visuals are on a single page, or many users access a report at the same time. Once the maximum number of connections is reached, further queries are queued until a connection becomes available. Increasing this limit does result in more load on the underlying data source, so the setting isn't guaranteed to improve overall performance.

When the model is published to Power BI, the maximum number of concurrent queries sent to the underlying data source also depends on the environment. Different environments (such as Power BI, Power BI Premium, or Power BI Report Server) each can impose different throughput constraints. For more information about Power BI Premium capacity resource limitations, see Deploying and Managing Power BI Premium Capacities.

Optimize report designs

Reports based on a DirectQuery dataset can be optimized in many ways, as described in the following bulleted list.

- **Enable query reduction techniques:** Power BI Desktop *Options and Settings* includes a Query Reduction page. This page has three helpful options. It's possible to disable cross-highlighting and cross-filtering by default, though it can be overridden by editing interactions. It is also possible to show an Apply button on slicers and filters. The slicer or filter options will not be applied until the report user clicks the button. If you enable these options, we recommend that you do so when first creating the report.



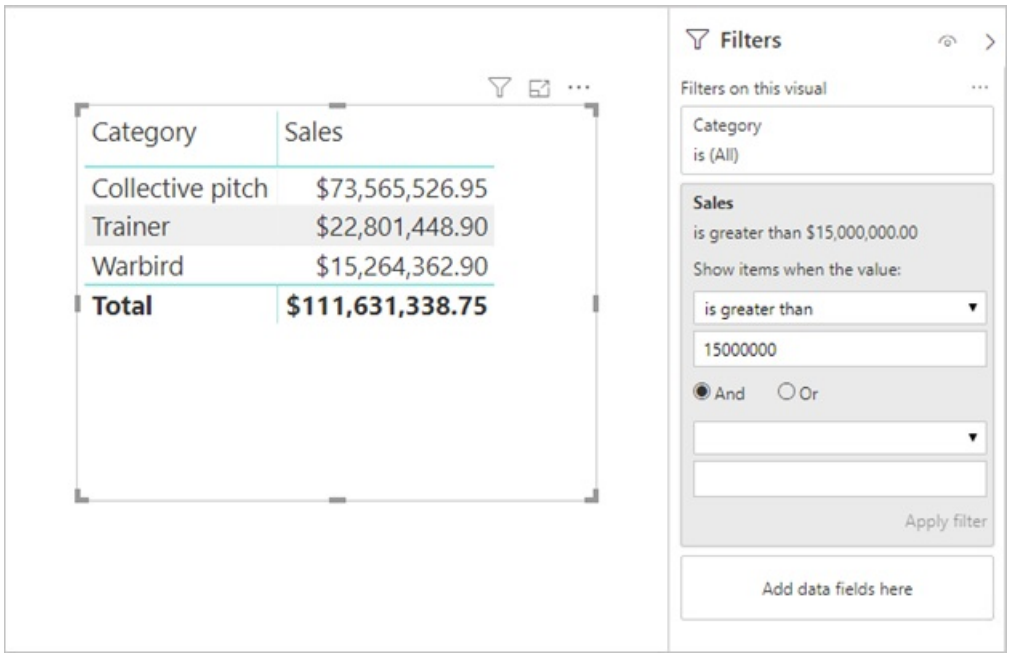
- **Apply filters first:** When first designing reports, we recommend that you apply any applicable filters—at report, page, or visual level—before mapping fields to the visual fields. For example, rather than dragging in the **Country** and **Sales** measures, and then filtering by a particular year, apply the filter on the **Year** field first. It's because each step of building a visual will send a query, and whilst it's possible to then make another change before the first query has completed, it still places unnecessary load on the underlying data source. By applying filters early, it generally makes those intermediate queries less costly and faster. Also, failing to apply filters early can result in exceeding the 1 million-row limit, as described above.

- **Limit the number of visuals on a page:** When a report page is opened (and when page filters are applied) all of the visuals on a page are refreshed. However, there is a limit on the number of queries that can be sent in parallel, imposed by the Power BI environment and the **Maximum Connections per Data Source** model setting, as described above. So, as the number of page visuals increases, there is higher chance that they will be refreshed in a serial manner. It increases the time taken to refresh the entire page, and it also increases the chance that visuals may display inconsistent results (for volatile data sources). For these reasons, it's recommended to limit the number of visuals on any page, and instead have more simpler pages. Replacing multiple card visuals with a single multi-row card visual can achieve a similar page layout.

- **Switch off interaction between visuals:** Cross-highlighting and cross-filtering interactions require queries be submitted to the underlying source. Unless these interactions are necessary, it's recommended they be switched off if the time taken to respond to users' selections would be unreasonably long. These interactions can be switched off, either for the entire report (as described above for Query Reduction options), or on a case-by-case basis. For more information, see How visuals cross-filter each other in a Power BI report.

In addition to the above list of optimization techniques, each of the following reporting capabilities can contribute to performance issues:

- **Measure filters:** Visuals containing measures (or aggregates of columns) can have filters applied to those measures. For example, the visual below shows **Sales** by **Category**, but only for categories with more than $15 million of sales.



  It may result in two queries being sent to the underlying source:

  - The first query will retrieve the categories meeting the condition (Sales > $15 million)
  - The second query will then retrieve the necessary data for the visual, adding the categories that met the condition to the WHERE clause

  It generally performs fine if there are hundreds or thousands of categories, as in this example. Performance can degrade, however, if the number of categories is much larger (and indeed, the query will fail if there are more than 1 million categories meeting the condition, due to the 1 million-row limit discussed above).

- **TopN filters:** Advanced filters can be defined to filter on only the top (or bottom) N values ranked by a measure. For example, to display only the top five categories in the above visual. Like the measure filters, it will also result in two queries being sent to the underlying data source. However, the first query will return all categories from the underlying source, and then the top N are determined based on the returned results. Depending on the cardinality of the column involved, it can lead to performance issues (or query failures due to the 1 million-row limit).

- **Median:** Generally, any aggregation (Sum, Count Distinct, etc.) is pushed to the underlying source. However, it's not true for Median, as this aggregate is not supported by the underlying source. In such cases, detail data is retrieved from the underlying source, and Power BI evaluates the median from the returned results. It's fine when the median is to be calculated over a relatively small number of results, but performance issues (or query failures due to the 1 million-row limit) will occur if the cardinality is large. For example, median country population might be reasonable, but median sales price might not be.

- **Multi-select slicers:** Allowing multi-selection in slicers and filters can cause performance issues. It's because as the user selects additional slicer items (for example, building up to the 10 products they are interested in), each new selection results in a new query being sent to the underlying source. Whilst the user can select the next item prior to the query completing, it results in extra load on the underlying source. This situation can be avoided by showing the Apply button, as described above in the query reduction techniques.

- **Visual totals:** By default, tables and matrices display totals and subtotals. In many cases, additional queries must be sent to the underlying source to obtain the values for the totals. It applies whenever using Count Distinct or Median aggregates, and in all cases when using DirectQuery over SAP HANA or SAP Business

Warehouse. Such totals should be switched off (by using the Format pane) if not necessary.

Convert to a Composite Model

The benefits of Import and DirectQuery models can be combined into a single model by configuring the storage mode of the model tables. The table storage mode can be Import or DirectQuery, or both, known as Dual. When a model contains tables with different storage modes, it is known as a Composite model. For more information, see Use composite models in Power BI Desktop.

There are many functional and performance enhancements that can be achieved by converting a DirectQuery model to a Composite model. A Composite model can integrate more than one DirectQuery source, and it can also include aggregations. Aggregation tables can be added to DirectQuery tables to import a summarized representation of the table. They can achieve dramatic performance enhancements when visuals query higher-level aggregates. For more information, see Aggregations in Power BI Desktop.

Educate users

It is important to educate your users on how to efficiently work with reports based on DirectQuery datasets. Your report authors should be educated on the content described in the [Optimize report designs](#optimize-report-designs section).

We recommend that you educate your report consumers about your reports that are based on DirectQuery datasets. It can be helpful for them to understand the general data architecture, including any relevant limitations described in this article. Let them know to expect that refresh responses and interactive filtering may at times be slow. When report users understand why performance degradation happens, they are less likely to lose trust in the reports and data.

When delivering reports on volatile data sources, be sure to educate report users on the use of the Refresh button. Let them know also that it may be possible to see inconsistent results, and that a refresh of the report can resolve any inconsistencies on the report page.

Next steps

For more information about DirectQuery, check out the following resources:

- DirectQuery models in Power BI Desktop
- Use DirectQuery in Power BI Desktop
- DirectQuery model troubleshooting in Power BI Desktop
- Questions? Try asking the Power BI Community

Composite model guidance in Power BI Desktop

5/13/2020 • 5 minutes to read • Edit Online

This article targets data modelers developing Power BI Composite models. It describes Composite model use cases, and provides you with design guidance. Specifically, the guidance is to help you determine whether a Composite model is appropriate for your solution. If it is, then this article will also help you design an optimal model.

> **NOTE**
>
> An introduction to Composite models is not covered in this article. If you're not completely familiar with Composite models, we recommend you first read the Use composite models in Power BI Desktop article.
>
> Because Composite models consist of at least one DirectQuery source, it's also important that you have a thorough understanding of model relationships, DirectQuery models, and DirectQuery model design guidance.

Composite model use cases

Whenever possible, it's best to develop a model in Import mode. This mode provides the greatest design flexibility, and best performance.

However, challenges related to large data volumes, or reporting on near real-time data, cannot be solved by Import models. In either of these cases, you can consider a DirectQuery model, providing your data is stored in a single data source that's supported by DirectQuery mode.

Further, you can consider developing a Composite model in the following situations.

- Your model could be a DirectQuery model, but you want to boost performance. In a Composite model, performance can be improved by configuring appropriate storage for each table. You can also add aggregations. Both of these optimizations are discussed later in this article.
- You want to combine a DirectQuery model with additional data, which must be imported into the model. Imported data can be loaded from a different data source, or from calculated tables.
- You want to combine two or more DirectQuery data sources into a single model.

> **NOTE**
>
> Composite models cannot combine Live Connection sources or DirectQuery analytic database sources. Live Connection sources include external-hosted models, and Power BI datasets. DirectQuery analytic database sources include SAP Business Warehouse, and SAP HANA.

Optimize model design

A Composite model can be optimized by configuring table storage modes, and by adding aggregations.

Table storage mode

In a Composite model, you can configure the storage mode for each table (except calculated tables):

- **DirectQuery**: We recommend you set this mode for tables that represent large data volumes, or which need to deliver near real-time results. Data will never be imported into these tables. Usually, these tables will be fact-type tables—tables used for summarization.
- **Import**: We recommend you set this mode for dimension-type tables—tables used for filtering and grouping. In

fact, it's the only option for tables based on sources not supported by DirectQuery mode. Calculated tables are always Import tables.

- **Dual**: We recommend you set this mode for dimension-type tables, when there's a possibility they'll be queried together with DirectQuery fact-type tables from the same source.

There are several possible scenarios when Power BI queries a Composite model:

- **Queries only Import or Dual table(s)**: All data is retrieved from the model cache. It will deliver the fastest possible performance. This scenario is common for dimension-type tables queried by filters or slicer visuals.
- **Queries Dual table(s) or DirectQuery table(s) from the same source**: All data is retrieved by sending one or more native queries to the DirectQuery source. It will deliver the fastest possible performance, especially when appropriate indexes exist on the source tables. This scenario is common for queries that relate Dual dimension-type tables and DirectQuery fact-type tables. These queries are *intra-island*, and so all one-to-one or one-to-many relationships are evaluated as strong relationships.
- **All other queries**: These queries involve cross-island relationships. It's either because an Import table relates to a DirectQuery table, or a Dual table relates to a DirectQuery table from a different source—in which case it behaves as an Import table. All relationships are evaluated as weak relationships. It also means that groupings applied to non-DirectQuery tables must be sent to the DirectQuery source as a virtual table. In this case, the native query can be inefficient, especially for large grouping sets. And, it has the potential to expose sensitive data in the native query.

In summary, we recommend that you:

- Consider carefully that a Composite model is the right solution—while it allows model-level integration of different data sources, it also introduces design complexities with possible consequences
- Set the storage mode to **DirectQuery** when a table is a fact-type table storing large data volumes, or it needs to deliver near real-time results
- Set the storage mode to **Dual** when a table is a dimension-type table, and it will be queried together with **DirectQuery** fact-type tables based on the same source
- Configure appropriate refresh frequencies to keep the model cache for Dual tables (and any dependent calculated tables) in sync with the source database(s)
- Strive to ensure data integrity across data sources (including the model cache)—weak relationships will eliminate rows when related column values don't match
- Optimize DirectQuery data sources with appropriate indexes for efficient joins, filtering and grouping
- Don't load sensitive data into Import or Dual tables if there's risk of a native query being intercepted—for more information, see Use composite models in Power BI Desktop (Security implications)

Aggregations

You can add aggregations to DirectQuery tables in your Composite model. Aggregations are cached in the model, and so they behave as Import tables (although they can't be used like a model table). Their purpose is to improve performance for "higher grain" queries. For more information, see Aggregations in Power BI Desktop.

We recommend that an aggregation table follows a basic rule: Its row count should be at least a factor of 10 smaller than the underlying table. For example, if the underlying table stores 1 billion rows, then the aggregation table shouldn't exceed 100 million rows. This rule ensures there's an adequate performance gain relative to the cost of creating and maintaining the aggregation table.

Next steps

For more information related to this article, check out the following resources:

- Use composite models in Power BI Desktop
- Model relationships in Power BI Desktop
- DirectQuery models in Power BI Desktop

- Use DirectQuery in Power BI Desktop
- DirectQuery model troubleshooting in Power BI Desktop
- Power BI data sources
- Storage mode in Power BI Desktop
- Aggregations in Power BI Desktop
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

DAX: DIVIDE function vs divide operator (/)

5/13/2020 • 2 minutes to read • Edit Online

As a data modeler, when you write a DAX expression to divide a numerator by a denominator, you can choose to use the DIVIDE function or the divide operator (/ - forward slash).

When using the DIVIDE function, you must pass in numerator and denominator expressions. Optionally, you can pass in a value that represents an *alternate result*.

```
DIVIDE(<numerator>, <denominator> [,<alternateresult>])
```

The DIVIDE function was designed to automatically handle division by zero cases. If an alternate result is not passed in, and the denominator is zero or BLANK, the function returns BLANK. When an alternate result is passed in, it's returned instead of BLANK.

The DIVIDE function is convenient because it saves your expression from having to first test the denominator value. The function is also better optimized for testing the denominator value than the IF function. The performance gain is significant since checking for division by zero is expensive. Further using DIVIDE results in a more concise and elegant expression.

Example

The following measure expression produces a safe division, but it involves using four DAX functions.

```
Profit Margin =
IF(
    OR(
        ISBLANK([Sales]),
        [Sales] == 0
    ),
    BLANK(),
    [Profit] / [Sales]
)
```

This measure expression achieves the same outcome, yet more efficiently and elegantly.

```
Profit Margin =
DIVIDE([Profit], [Sales])
```

Recommendations

We recommend that you use the DIVIDE function whenever the denominator is an expression that *could* return zero or BLANK.

In the case that the denominator is a constant value, we recommend that you use the divide operator. In this case, the division is guaranteed to succeed, and your expression will perform better because it will avoid unnecessary testing.

Carefully consider whether the DIVIDE function should return an alternate value. For measures, it's usually a better design that they return BLANK. Returning BLANK is better because report visuals—by default—eliminate groupings when summarizations are BLANK. It allows the visual to focus attention on groups where data exists.

When necessary, you can configure the visual to display all groups (that return values or BLANK) within the filter context by enabling the Show items with no data option.

Next steps

For more information about this article, check out the following resources:

- Data Analysis Expressions (DAX) Reference
- Questions? Try asking the Power BI Community

DAX: Appropriate use of error functions

11/22/2019 • 2 minutes to read • Edit Online

As a data modeler, when you write a DAX expression that might raise an evaluation-time error, you can consider using two helpful DAX functions.

- The ISERROR function, which takes a single expression and returns TRUE if that expression results in error.
- The IFERROR function, which takes two expressions. Should the first expression result in error, the value for the second expression is returned. It is in fact a more optimized implementation of nesting the ISERROR function inside an IF function.

However, while these functions can be helpful and can contribute to writing easy-to-understand expressions, they can also significantly degrade the performance of calculations. It can happen because these functions increase the number of storage engine scans required.

Most evaluation-time errors are due to unexpected BLANKs or zero values, or invalid data type conversion.

Recommendations

It's better to avoid using the ISERROR and IFERROR functions. Instead, apply defensive strategies when developing the model and writing expressions. Strategies can include:

- **Ensuring quality data is loaded into the model:** Use Power Query transformations to remove or substitute invalid or missing values, and to set correct data types. A Power Query transformation can also be used to filter rows when errors, like invalid data conversion, occur.

  Data quality can also be controlled by setting the model column **Is Nullable** property to Off, which will fail the data refresh should BLANKs be encountered. If this failure occurs, data loaded as a result of a successful refresh will remain in the tables.

- **Using the IF function:** The IF function logical test expression can determine whether an error result would occur. Note, like the ISERROR and IFERROR functions, this function can result in additional storage engine scans, but will likely perform better than them as no error needs to be raised.

- **Using error-tolerant functions:** Some DAX functions will test and compensate for error conditions. These functions allow you to enter an alternate result that would be returned instead. The DIVIDE function is one such example. For additional guidance about this function, read the DAX: DIVIDE function vs divide operator (/) article.

Example

The following measure expression tests whether an error would be raised. It returns BLANK in this instance (which is the case when you do not provide the IF function with a value-if-false expression).

```
Profit Margin
= IF(ISERROR([Profit] / [Sales]))
```

This next version of the measure expression has been improved by using the IFERROR function in place of the IF and ISERROR functions.

```
Profit Margin
= IFERROR([Profit] / [Sales], BLANK())
```

However, this final version of the measure expression achieves the same outcome, yet more efficiently and elegantly.

```
Profit Margin
= DIVIDE([Profit], [Sales])
```

Next steps

For more information about this article, check out the following resources:

- Data Analysis Expressions (DAX) Reference
- Questions? Try asking the Power BI Community

DAX: Use SELECTEDVALUE instead of VALUES

12/2/2019 • 2 minutes to read • Edit Online

As a data modeler, sometimes you might need to write a DAX expression that tests whether a column is filtered by a specific value.

In earlier versions of DAX, this requirement was safely achieved by using a pattern involving three DAX functions. The functions are IF, HASONEVALUE and VALUES. The following measure definition presents an example. It calculates the sales tax amount, but only for sales made to Australian customers.

```
Australian Sales Tax =
IF(
    HASONEVALUE(Customer[Country-Region]),
    IF(
        VALUES(Customer[Country-Region]) = "Australia",
        [Sales] * 0.10
    )
)
```

In the example, the HASONEVALUE function returns TRUE only when a single value filters the **Country-Region** column. When it's TRUE, the VALUES function is compared to the literal text "Australia". When the VALUES function returns TRUE, the **Sales** measure is multiplied by 0.10 (representing 10%). If the HASONEVALUE function returns FALSE—because more than one value filters the column—the first IF function returns BLANK.

The use of the HASONEVALUE is a defensive technique. It's required because it's possible that multiple values filter the **Country-Region** column. In this case, the VALUES function returns a table of multiple rows. Comparing a table of multiple rows to a scalar value results in an error.

Recommendation

We recommend that you use the SELECTEDVALUE function. It achieves the same outcome as the pattern described in this article, yet more efficiently and elegantly.

Using the SELECTEDVALUE function, the example measure definition is now rewritten.

```
Australian Sales Tax =
IF(
    SELECTEDVALUE(Customer[Country-Region]) = "Australia",
    [Sales] * 0.10
)
```

TIP

It's possible to pass an *alternate result* value into the SELECTEDVALUE function. The alternate result value is returned when either no filters—or multiple filters—are applied to the column.

Next steps

For more information about this article, check out the following resources:

- Data Analysis Expressions (DAX) Reference

- Questions? Try asking the Power BI Community

DAX: Use COUNTROWS instead of COUNT

12/2/2019 • 2 minutes to read • Edit Online

As a data modeler, sometimes you might need to write a DAX expression that counts table rows. The table could be a model table or an expression that returns a table.

Your requirement can be achieved in two ways. You can use the COUNT function to count column values, or you can use the COUNTROWS function to count table rows. Both functions will achieve the same result, providing that the counted column contains no BLANKs.

The following measure definition presents an example. It calculates the number of **OrderDate** column values.

```
Sales Orders =
COUNT(Sales[OrderDate])
```

Providing that the granularity of the **Sales** table is one row per sales order, and the **OrderDate** column does not contain BLANKs, then the measure will return a correct result.

However, the following measure definition is a better solution.

```
Sales Orders =
COUNTROWS(Sales)
```

There are three reasons why the second measure definition is better:

- It's more efficient, and so it will perform better.
- It doesn't consider BLANKs contained in any column of the table.
- The intention of formula is clearer, to the point of being self-describing.

Recommendation

When it's your intention to count table rows, we recommend that you always use the COUNTROWS function.

Next steps

For more information about this article, check out the following resources:

- Data Analysis Expressions (DAX) Reference
- Questions? Try asking the Power BI Community

DAX: Use variables to improve your formulas

12/2/2019 • 3 minutes to read • Edit Online

As a data modeler, writing and debugging some DAX calculations can be challenging. It's common that complex calculation requirements often involve writing compound or complex expressions. Compound expressions can involve the use of many nested functions, and possibly the reuse of expression logic.

Using variables in your DAX formulas helps you write complex and efficient calculations. Variables can:

- Improve performance
- Improve readability
- Simplify debugging
- Reduce complexity

In this article, we'll demonstrate the first three benefits by using an example measure for year-over-year (YoY) sales growth. (The formula for YoY sales growth is: period sales _fewer sales for the same period last year, *divided by* sales for the same period last year.)

Let's start with the following measure definition.

```
Sales YoY Growth % =
DIVIDE(
    ([Sales] - CALCULATE([Sales], PARALLELPERIOD('Date'[Date], -12, MONTH))),
    CALCULATE([Sales], PARALLELPERIOD('Date'[Date], -12, MONTH))
)
```

The measure produces the correct result, yet let's now see how it can be improved.

Improve performance

Notice that the formula repeats the expression that calculates "same period last year". This formula is inefficient, as it requires Power BI to evaluate the same expression twice. The measure definition can be made more efficient by using a variable.

The following measure definition represents an improvement. It uses an expression to assign the "same period last year" result to a variable named **SalesPriorYear**. The variable is then used twice in the RETURN expression.

```
Sales YoY Growth % =
VAR SalesPriorYear =
    CALCULATE([Sales], PARALLELPERIOD('Date'[Date], -12, MONTH))
RETURN
    DIVIDE(([Sales] - SalesPriorYear), SalesPriorYear)
```

The measure continues to produce the correct result, and does so in about half the query time.

Improve readability

In the previous measure definition, notice how the choice of variable name makes the RETURN expression simpler to understand. The expression is short and self-describing.

Simplify debugging

Variables can also help you debug a formula. To test an expression assigned to a variable, you temporarily rewrite the RETURN expression to output the variable.

The following measure definition returns only the **SalesPriorYear** variable. Notice how it comments-out the intended RETURN expression. This technique allows you to easily revert it back once your debugging is complete.

```
Sales YoY Growth % =
VAR SalesPriorYear =
    CALCULATE([Sales], PARALLELPERIOD('Date'[Date], -12, MONTH))
RETURN
    --DIVIDE(([Sales] - SalesPriorYear), SalesPriorYear)
    SalesPriorYear
```

Reduce complexity

In earlier versions of DAX, variables were not yet supported. Complex expressions that introduced new filter contexts were required to use the EARLIER or EARLIEST DAX functions to reference outer filter contexts. Unfortunately, data modelers found these functions difficult to understand and use.

Variables are always evaluated outside the filters your RETURN expression applies. For this reason, when you use a variable within a modified filter context, it achieves the same result as the EARLIEST function. The use of the EARLIER or EARLIEST functions can therefore be avoided. It means you can now write formulas that are less complex, and that are easier to understand.

Consider the following calculated column definition added to the **Subcategory** table. It evaluates a rank for each product subcategory based on the **Subcategory Sales** column values.

```
Subcategory Sales Rank =
COUNTROWS(
    FILTER(
        Subcategory,
        EARLIER(Subcategory[Subcategory Sales]) < Subcategory[Subcategory Sales]
    )
) + 1
```

The EARLIER function is used to refer to the **Subcategory Sales** column value *in the current row context*.

The calculated column definition can be improved by using a variable instead of the EARLIER function. The **CurrentSubcategorySales** variable stores the **Subcategory Sales** column value *in the current row context*, and the RETURN expression uses it within a modified filter context.

```
Subcategory Sales Rank =
VAR CurrentSubcategorySales = Subcategory[Subcategory Sales]
RETURN
    COUNTROWS(
        FILTER(
            Subcategory,
            CurrentSubcategorySales < Subcategory[Subcategory Sales]
        )
    ) + 1
```

Next steps

For more information about this article, check out the following resources:

- VAR DAX article
- Questions? Try asking the Power BI Community

DAX: Avoid converting BLANKs to values

5/13/2020 • 2 minutes to read • Edit Online

As a data modeler, when writing measure expressions you might come across cases where a meaningful value can't be returned. In these instances, you may be tempted to return a value—like zero—instead. We suggest you carefully determine whether this design is efficient and practical.

Consider the following measure definition that explicitly converts BLANK results to zero.

```
Sales (No Blank) =
IF(
    ISBLANK([Sales]),
    0,
    [Sales]
)
```

Consider another measure definition that also converts BLANK results to zero.

```
Profit Margin =
DIVIDE([Profit], [Sales], 0)
```

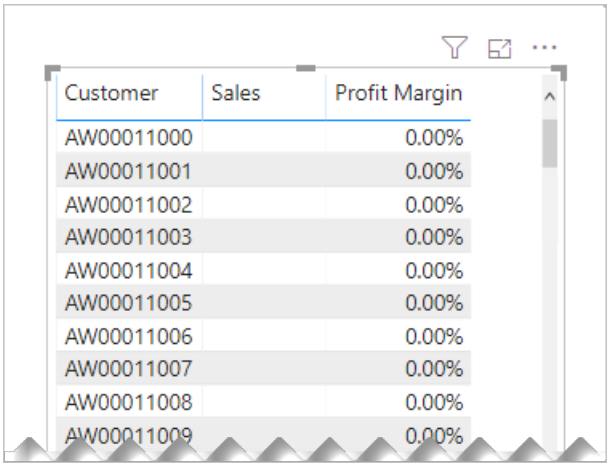
The DIVIDE function divides the **Profit** measure by the **Sales** measure. Should the result be zero or BLANK, the third argument—the alternate result (which is optional)—is returned. In this example, because zero is passed as the alternate result, the measure is guaranteed to always return a value.

These measure designs are inefficient and lead to poor report designs.

When they're added to a report visual, Power BI attempts to retrieve all groupings within the filter context. The evaluation and retrieval of large query results often leads to slow report rendering. Each example measure effectively turns a sparse calculation into a dense one, forcing Power BI to use more memory than necessary.

Also, too many groupings often overwhelm your report users.

Let's see what happens when the **Profit Margin** measure is added to a table visual, grouping by customer.



Customer	Sales	Profit Margin
AW00011000		0.00%
AW00011001		0.00%
AW00011002		0.00%
AW00011003		0.00%
AW00011004		0.00%
AW00011005		0.00%
AW00011006		0.00%
AW00011007		0.00%
AW00011008		0.00%
AW00011009		0.00%

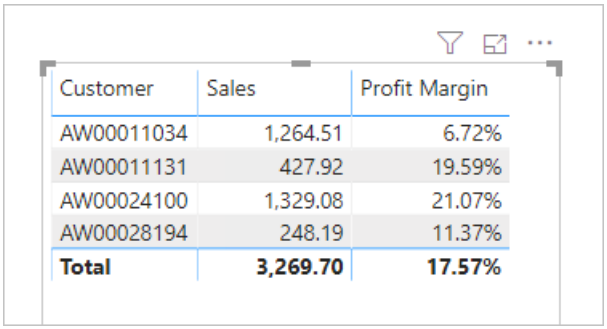
The table visual displays an overwhelming number of rows. (There are in fact 18,484 customers in the model, and so the table attempts to display all of them.) Notice that the customers in view haven't achieved any sales. Yet, because the **Profit Margin** measure always returns a value, they are displayed.

> **NOTE**
>
> When there are too many data points to display in a visual, Power BI may use data reduction strategies to remove or summarize large query results. For more information, see Data point limits and strategies by visual type.

Let's see what happens when the **Profit Margin** measure definition is improved. It now returns a value only when the **Sales** measure isn't BLANK (or zero).

```
Profit Margin =
DIVIDE([Profit], [Sales])
```

The table visual now displays only customers who have made sales within the current filter context. The improved measure results in a more efficient and practical experience for your report users.



Customer	Sales	Profit Margin
AW00011034	1,264.51	6.72%
AW00011131	427.92	19.59%
AW00024100	1,329.08	21.07%
AW00028194	248.19	11.37%
Total	**3,269.70**	**17.57%**

> **TIP**
>
> When necessary, you can configure a visual to display all groupings (that return values or BLANK) within the filter context by enabling the Show Items With No Data option.

Recommendation

We recommend that your measures return BLANK when a meaningful value cannot be returned.

This design approach is efficient, allowing Power BI to render reports faster. Also, returning BLANK is better because report visuals—by default—eliminate groupings when summarizations are BLANK.

Next steps

For more information about this article, check out the following resources:

- Data Analysis Expressions (DAX) Reference
- Questions? Try asking the Power BI Community

DAX: Column and measure references

5/13/2020 • 2 minutes to read • Edit Online

As a data modeler, your DAX expressions will refer to model columns and measures. Columns and measures are always associated with model tables, but these associations are different. So, we have different recommendations on how you'll reference them in your expressions.

Columns

A column is a table-level object, and column names must be unique within a table. So it's possible that the same column name is used multiple times in your model—providing they belong to different tables. There's one more rule: a column name cannot have the same name as a measure name or hierarchy name that exists in the same table.

In general, DAX will not force using a *fully qualified* reference to a column. A fully qualified reference means that the table name precedes the column name.

Here's an example of a calculated column definition using only column name references. The **Sales** and **Cost** columns both belong to a table named **Orders**.

```
Profit = [Sales] - [Cost]
```

The same definition can be rewritten with fully qualified column references.

```
Profit = Orders[Sales] - Orders[Cost]
```

Sometimes, however, you'll be required to use fully qualified column references when Power BI detects ambiguity. When entering a formula, a red squiggly and error message will alert you. Also, some DAX functions like the LOOKUPVALUE DAX function, require the use of fully qualified columns.

We recommend you always fully qualify your column references. The reasons are provided in the Recommendations section.

Measures

A measure is a model-level object. For this reason, measure names must be unique within the model. However, in the **Fields** pane, report authors will see each measure associated with a single model table. This association is set for cosmetic reasons, and you can configure it by setting the **Home Table** property for the measure. For more information, see Measures in Power BI Desktop (Organizing your measures).

It's possible to use a fully qualified measure in your expressions. DAX intellisense will even offer the suggestion. However, it isn't necessary, and it's not a recommended practice. If you change the home table for a measure, any expression that uses a fully qualified measure reference to it will break. You'll then need to edit each broken formula to remove (or update) the measure reference.

We recommend you never qualify your measure references. The reasons are provided in the Recommendations section.

Recommendations

Our recommendations are simple and easy to remember:

- Always use fully qualified column references
- Never use fully qualified measure references

Here's why:

- **Formula entry**: Expressions will be accepted, as there won't be any ambiguous references to resolve. Also, you'll meet the requirement for those DAX functions that require fully qualified column references.
- **Robustness**: Expressions will continue to work, even when you change a measure home table property.
- **Readability**: Expressions will be quick and easy to understand—you'll quickly determine that it's a column or measure, based on whether it's fully qualified or not.

Next steps

For more information about this article, check out the following resources:

- Data Analysis Expressions (DAX) Reference
- Questions? Try asking the Power BI Community

DAX: Avoid using FILTER as a filter argument

1/16/2020 • 2 minutes to read • Edit Online

As a data modeler, it's common you'll write DAX expressions that need to be evaluated in a modified filter context. For example, you can write a measure definition to calculate sales for "high margin products". We'll describe this calculation later in this article.

> **NOTE**
> This article is especially relevant for model calculations that apply filters to Import tables.

The CALCULATE and CALCULATETABLE DAX functions are important and useful functions. They let you write calculations that remove or add filters, or modify relationship paths. It's done by passing in filter arguments, which are either Boolean expressions, table expressions, or special filter functions. We'll only discuss Boolean and table expressions in this article.

Consider the following measure definition, which calculates red product sales by using a table expression. It will replace any filters that might be applied to the **Product** table.

```
Red Sales =
CALCULATE(
    [Sales],
    FILTER('Product', 'Product'[Color] = "Red")
)
```

The CALCULATE function accepts a table expression returned by the FILTER DAX function, which evaluates its filter expression for each row of the **Product** table. It achieves the correct result—the sales result for red products. However, it could be achieved much more efficiently by using a Boolean expression.

Here's an improved measure definition, which uses a Boolean expression instead of the table expression. The KEEPFILTERS DAX function ensures any existing filters applied to the **Color** column are preserved, and not overwritten.

```
Red Sales =
CALCULATE(
    [Sales],
    KEEPFILTERS('Product'[Color] = "Red")
)
```

We recommend you pass filter arguments as Boolean expressions, whenever possible. It's because Import model tables are in-memory column stores. They are explicitly optimized to efficiently filter columns in this way.

There are, however, restrictions that apply to Boolean expressions when they're used as filter arguments. They:

- Cannot compare columns to other columns
- Cannot reference a measure
- Cannot use nested CALCULATE functions
- Cannot use functions that scan or return a table

It means that you'll need to use table expressions for more complex filter requirements.

Consider now a different measure definition.

```
High Margin Product Sales =
CALCULATE(
    [Sales],
    FILTER(
        'Product',
        'Product'[ListPrice] > 'Product'[StandardCost] * 2
    )
)
```

The definition of a *high margin product* is one that has a list price exceeding double its standard cost. In this example, the FILTER function must be used. It's because the filter expression is too complex for a Boolean expression.

Here's one more example. The requirement this time is to calculate sales, but only for months that have achieved a profit.

```
Sales for Profitable Months =
CALCULATE(
    [Sales],
    FILTER(
        VALUES('Date'[Month]),
        [Profit] > 0)
    )
)
```

In this example, the FILTER function must also be used. It's because it requires evaluating the **Profit** measure to eliminate those months that didn't achieve a profit. It's not possible to use a measure in a Boolean expression when it's used as a filter argument.

Recommendations

For best performance, we recommend you use Boolean expressions as filter arguments, whenever possible.

Therefore, the FILTER function should only be used when necessary. You can use it to perform filter complex column comparisons. These column comparisons can involve:

- Measures
- Other columns
- Using the OR DAX function, or the OR logical operator (||)

Next steps

For more information about this article, check out the following resources:

- Data Analysis Expressions (DAX) Reference
- Filter functions (DAX)
- Questions? Try asking the Power BI Community

Separate reports from models in Power BI Desktop

5/13/2020 • 3 minutes to read • Edit Online

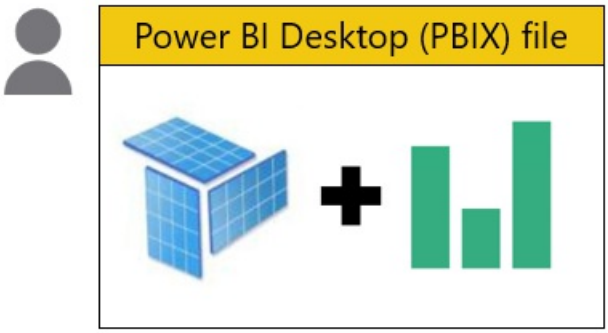
When creating a new Power BI Desktop solution, one of the first tasks you need to do is "get data". Getting data can result in two distinctly different outcomes. It could:

- Create a Live Connection to an already-published model, which could be a Power BI dataset or a remote-hosted Analysis Services model.
- Commence the development of a new model, which could be either an Import, DirectQuery, or Composite model.

This article is concerned with the second scenario. It provides guidance on whether a report and model should be combined into a single Power BI Desktop file.

Single file solution

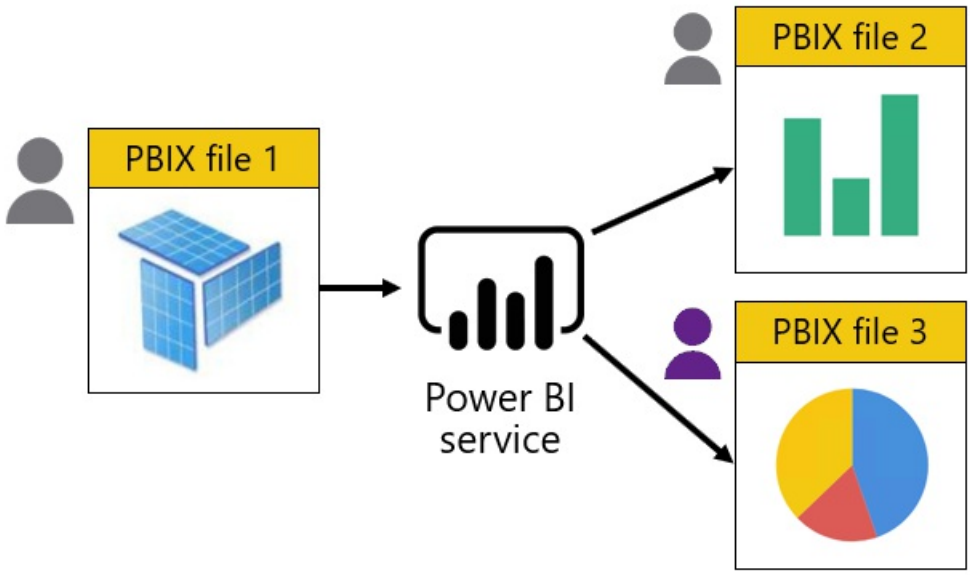
A *single file solution* works well when there's only ever a single report based on the model. In this case, it's likely that both the model and report are the efforts of the same person. We define it as a *Personal BI* solution, though the report could be shared with others. Such solutions can represent role-scoped reports or one-time assessments of a business challenge—often described as *ad hoc* reports.



Separate report files

It makes sense to separate model and report development into separate Power BI Desktop files when:

- Data modelers and report authors are different people.
- It's understood that a model will be the source for multiple reports, now or in the future.



Data modelers can still use the Power BI Desktop report authoring experience to test and validate their model designs. However, just after publishing their file to the Power BI service they should remove the report from the workspace. And, they must remember to remove the report each time they republish and overwrite the dataset.

Preserve the model interface

Sometimes, model changes are inevitable. Data modelers must take care, then, not the break the model interface. If they do, it's possible that related report visuals or dashboard tiles will break. Broken visuals appear as errors, and they can result in frustration for report authors and consumers. And worse—they can reduce trust in the data.

So, manage model changes carefully. If possible, avoid the following changes:

- Renaming tables, columns, hierarchies, hierarchy levels, or measures.
- Modifying column data types.
- Modifying measure expressions so they return a different data type.
- Moving measures to a different home table. It's because moving a measure could break report-scoped measures that fully qualify measures with their home table name. We don't recommend you write DAX expressions using fully qualified measures names. For more information, see DAX: Column and measure references.

Adding new tables, columns, hierarchies, hierarchy levels, or measures is safe, with one exception: It's possible that a new measure name could collide with a report-scoped measure name. To avoid collision, we recommend report authors adopt a naming convention when defining measures in their reports. They can prefix report-scoped measure names with an underscore or some other character(s).

If you must make breaking changes to your models, we recommend you either:

- View related content for the dataset in the Power BI service.
- Explore Data lineage view in the Power BI service.

Both options allow you to quickly identify any related reports and dashboards. Data lineage view is probably the better choice because it's easy to see the contact person for each related artifact. In fact, it's a hyperlink that opens an email message addressed to the contact.

We recommend you contact the owner of each related artifact to let them know of any planned breaking changes. This way, they can be prepared and ready to fix and republish their reports, helping to minimize downtime and frustration.

Next steps

For more information related to this article, check out the following resources:

- Connect to datasets in the Power BI service from Power BI Desktop
- View related content in the Power BI service
- Data lineage
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Extend visuals with report page tooltips

5/13/2020 • 3 minutes to read • Edit Online

This article targets you as a report author designing Power BI reports. It provides suggestions and recommendations when creating report page tooltips.

Suggestions

Report page tooltips can enhance the experience for your report users. Page tooltips allow your report users to quickly and efficiently gain deeper insights from a visual. They can be associated with different report objects:

- **Visuals:** On a visual-by-visual basis, you can configure which visuals will reveal your page tooltip. Per visual, it's possible to have the visual reveal no tooltip, default to the visual tooltips (configured in the visual fields pane), or use a specific page tooltip.
- **Visual headers:** You can configure specific visuals to display a page tooltip. Your report users can reveal the page tooltip when they hover their cursor over the visual header icon—be sure to educate your users about this icon.

> **NOTE**
>
> A report visual can only reveal a page tooltip when tooltip page filters are compatible with the visual's design. For example, a visual that groups by *product* is compatible with a tooltip page that filters by *product*.
>
> Page tooltips don't support interactivity. If you want your report users to interact, create a drillthrough page instead.
>
> Power BI visuals do not support page tooltips.

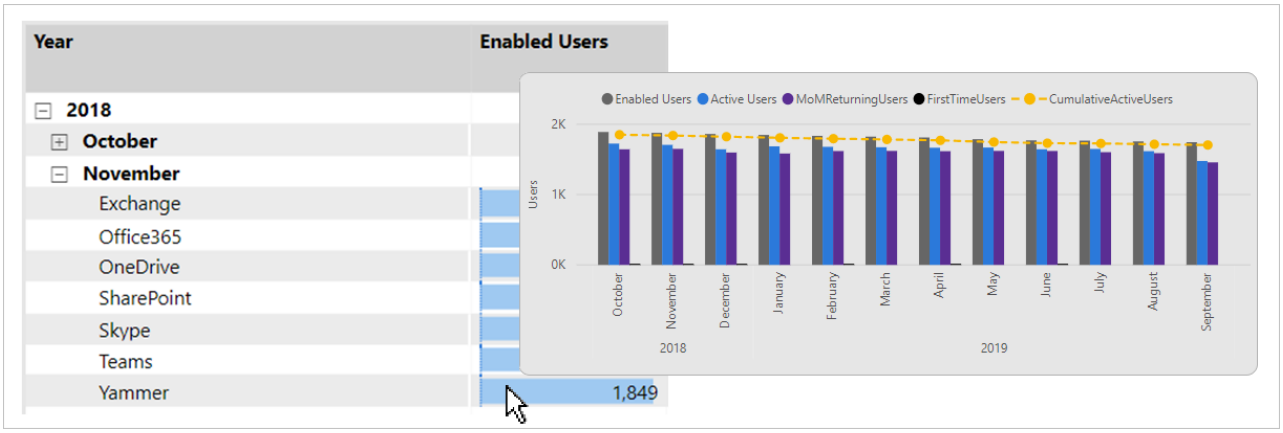
Here are some suggested design scenarios:

- Different perspective
- Add detail
- Add help

Different perspective

A page tooltip can visualize the same data as the source visual. It's done by using the same visual and pivoting groups, or by using different visual types. Page tooltips can also apply different filters than those filters applied to the source visual.

The following example shows what happens when the report user hovers their cursor over the **EnabledUsers** value. The filter context for the value is Yammer in November 2018.

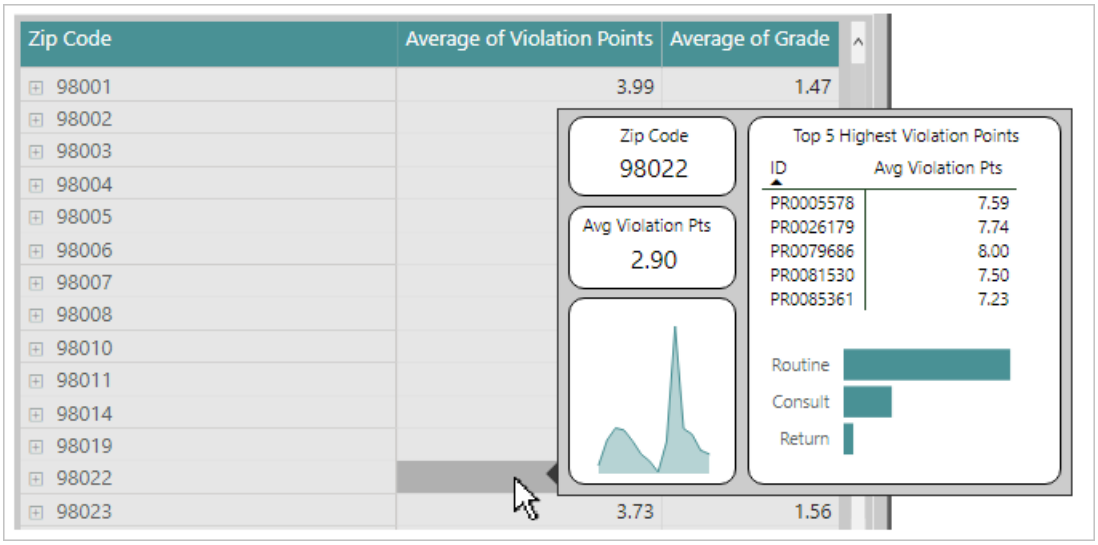


A page tooltip is revealed. It presents a different data visual (line and clustered column chart) and applies a contrasting time filter. Notice that the filter context for the data point is November 2018. Yet the page tooltip displays trend over *a full year of months*.

Add detail

A page tooltip can display additional details and add context.

The following example shows what happens when the report user hovers their cursor over the **Average of Violation Points** value, for zip code 98022.



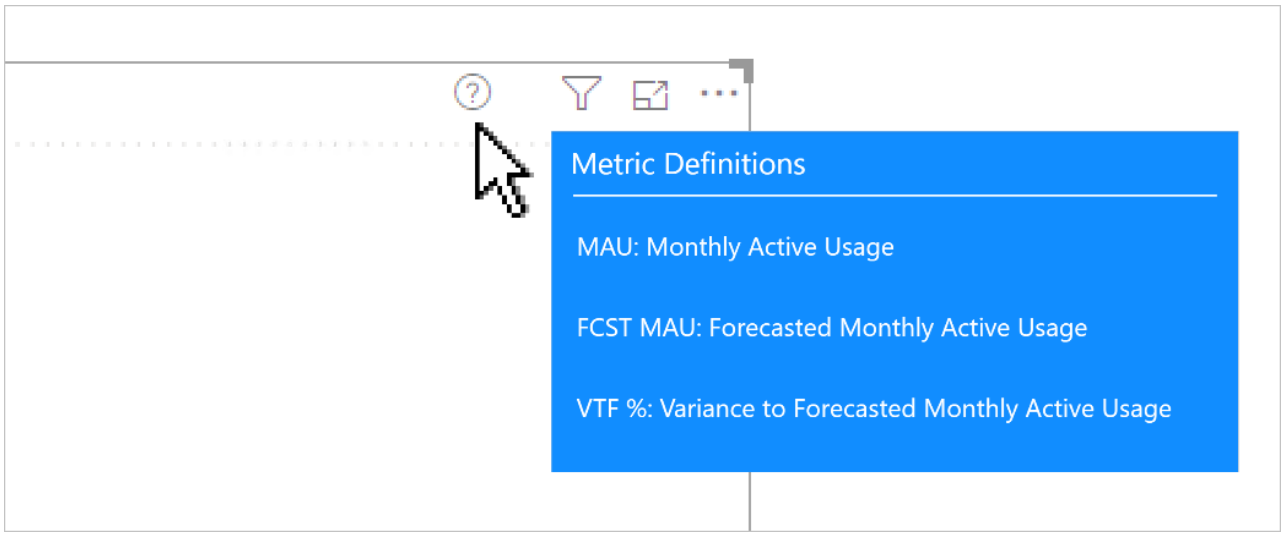
A page tooltip is revealed. It presents specific attributes and statistics for zip code 98022.

Add help

Visual headers can be configured to reveal page tooltips to visual headers. You can add help documentation to a page tooltip by using richly formatted text boxes. It's also possible to add images and shapes.

Interestingly, buttons, images, text boxes, and shapes can also reveal a visual header page tooltip.

The following example shows what happens when the report user hovers their cursor over the visual header icon.



A page tooltip is revealed. It presents rich formatted text in four text boxes, and a shape (line). The page tooltip conveys help by describing each acronym displayed in the visual.

Recommendations

At report design time, we recommend the following practices:

- **Page size:** Configure your page tooltip to be small. You can use the built-in **Tooltip** option (320 pixels wide, 240 pixels high). Or, you can set custom dimensions. Take care not to use a page size that's too large—it can obscure the visuals on the source page.
- **Page view:** In report designer, set the page view to **Actual Size** (page view defaults to **Fit to Page**). This way, you can see the true size of the page tooltip as you design it.
- **Style:** Consider designing your page tooltip to use the same theme and style as the report. This way, users feel like they are in the same report. Or, design a complimentary style for your tooltips, and be sure to apply this style to all page tooltips.
- **Tooltip filters:** Assign filters to the page tooltip so that you can preview a realistic result as you design it. Be sure to remove these filters before you publish your report.
- **Page visibility:** Always hide tooltip pages—users shouldn't navigate directly to them.

Next steps

For more information related to this article, check out the following resources:

- Create tooltips based on report pages in Power BI Desktop
- Customizing tooltips in Power BI Desktop
- Use visual elements to enhance Power BI reports
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Use report page drillthrough

5/13/2020 • 2 minutes to read • Edit Online

This article targets you as a report author who designs Power BI reports. It provides suggestions and recommendations when creating report page drillthrough.

It's recommended that you design your report to allow report users to achieve the following flow:

1. View a report page.
2. Identify a visual element to analyze more deeply.
3. Right-click the visual element to drillthrough.
4. Perform complimentary analysis.
5. Return to the source report page.

Suggestions

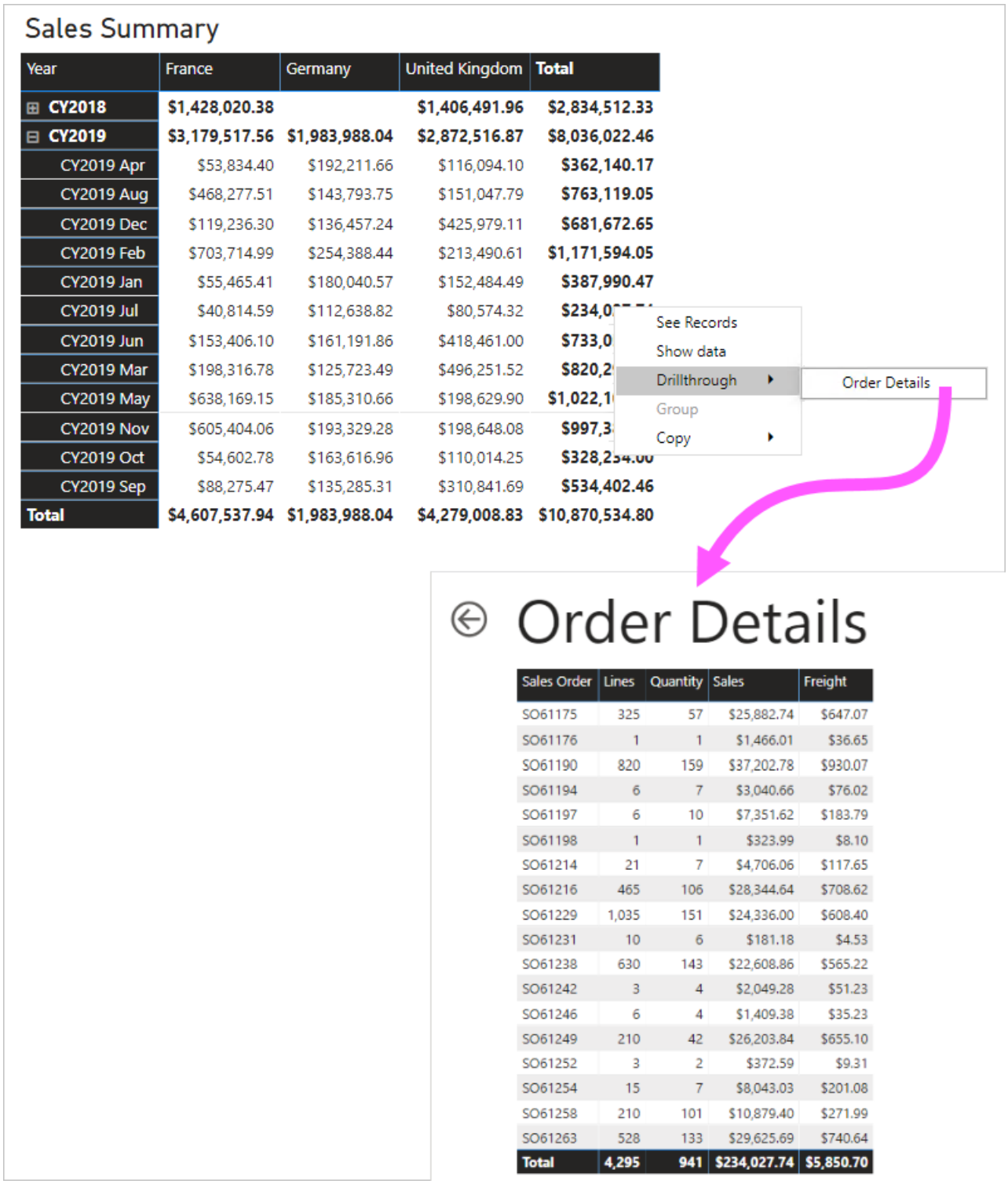
We suggest that you consider two types of drillthrough scenarios:

- Additional depth
- Broader perspective

Additional depth

When your report page displays summarized results, a drillthrough page can lead report users to transaction-level details. This design approach allows them to view supporting transactions, and only when needed.

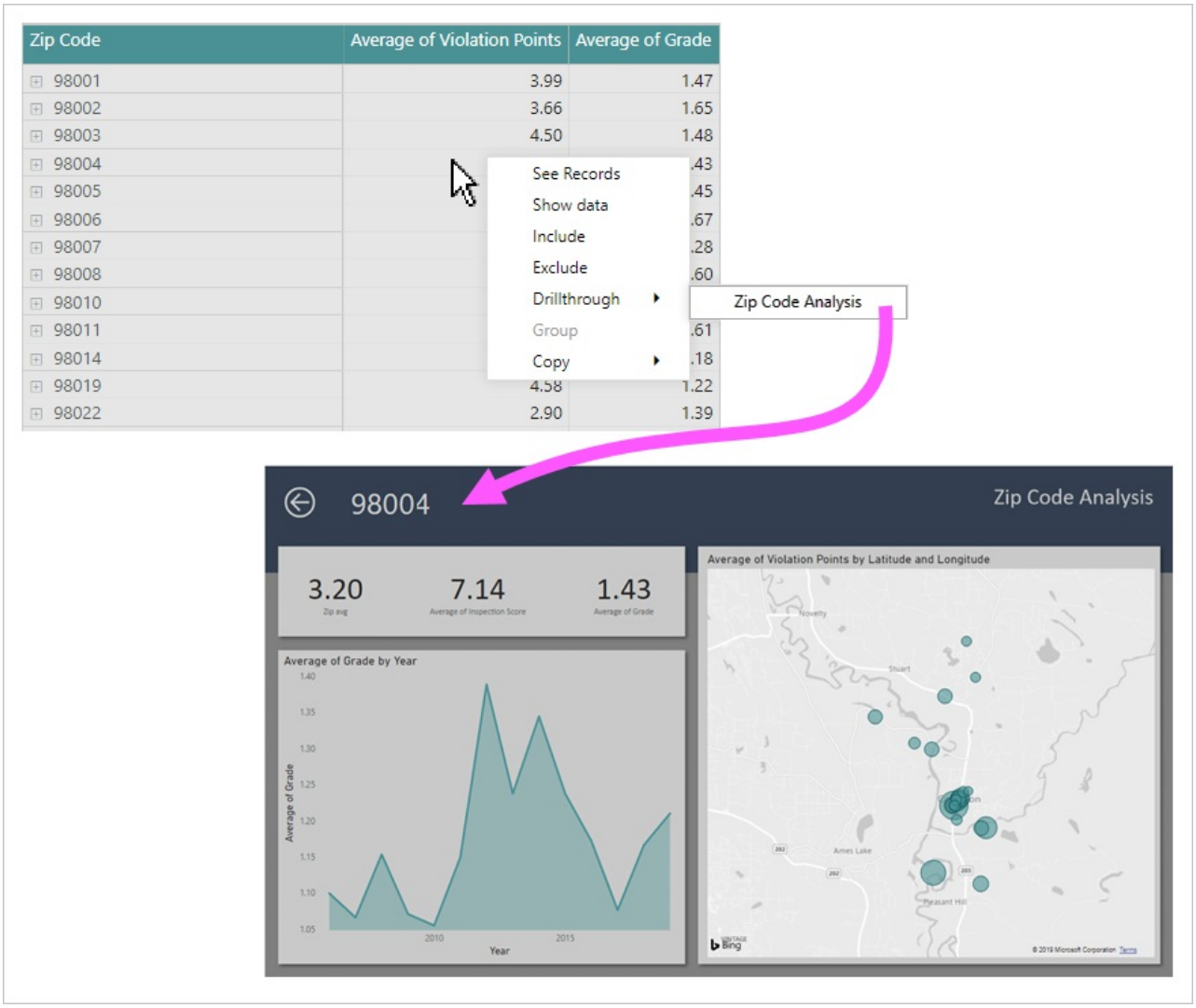
The following example shows what happens when a report user drills through from a monthly sales summary. The drillthrough page contains a detailed list of orders for a specific month.



Broader perspective

A drillthrough page can achieve the opposite of additional depth. This scenario is great for drilling through to a holistic view.

The following example shows what happens when a report user drills through from a zip code. The drillthrough page displays general information about that zip code.



Recommendations

At report design time, we recommend the following practices:

- **Style:** Consider designing your drillthrough page to use the same theme and style as the report. This way, users feel like they are in the same report.
- **Drillthrough filters:** Set drillthrough filters so you can preview a realistic result as you design the drillthrough page. Be sure to remove these filters before you publish the report.
- **Additional capabilities:** A drillthrough page is like any report page. You can even enhance it with additional interactive capabilities, including slicers or filters.
- **Blanks:** Avoid adding visuals that could display BLANK, or produce errors when drillthrough filters are applied.
- **Page visibility:** Consider hiding drillthrough pages. If you decide to keep a drillthrough page visible, be sure to add a button that allows users to clear any previously-set drillthrough filters. Assign a bookmark to the button. The bookmark should be configured to remove all filters.
- **Back button:** A back button is added automatically when you assign a drillthrough filter. It's a good idea to keep it. This way, your report users can easily return to the source page.
- **Discovery:** Help promote awareness of a drillthrough page by setting visual header icon text, or adding instructions to a text box. You can also design an overlay, as described in this blog post.

> **TIP**
>
> It's also possible to configure drillthrough to your Power BI paginated reports. You can do this be adding links to Power BI reports. Links can define URL parameters.

Next steps

For more information related to this article, check out the following resources:

- Use drillthrough in Power BI Desktop
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Tips to manage axes in Power BI reports

5/13/2020 • 2 minutes to read • Edit Online

This article targets you as a report author designing Power BI reports, working with visuals that have axes. These visuals include bar charts, columns charts, line charts, and many others.

Watch the video demonstrating eight tips to effectively manage axes in your Power BI reports.

Tips

In summary, the top eight tips to effectively manage axes in Power BI reports include:

1. Visualize nominal categories
2. Visualize interval categories
3. Adjust X-axis labels
4. Adjust Y-axis labels
5. Manage X-axis hierarchies
6. Manage Y-axis hierarchies
7. Avoid the X-axis scrollbar
8. Remove axes to create sparklines

Next steps

For more information related to this article, check out the following resources:

- Tips for creating stunning reports
- biDezine video: Top 8 Tips To Effectively Manage Axes in Power BI
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Tips to control chart gridlines in Power BI reports

5/13/2020 • 2 minutes to read • Edit Online

This article targets you as a report author designing Power BI reports, working with chart visuals that have gridlines.

Watch the video demonstrating the top three tips to control gridlines in your Power BI reports.

Tips

In summary, the top three tips to control chart gridlines in Power BI reports include:

1. Sort by value
2. Sort by time/numerical sequences
3. Sort by categorical sequences

Next steps

For more information related to this article, check out the following resources:

- Tips for creating stunning reports
- biDezine video: Top 3 Tips to Control Chart Gridlines in Power BI
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Tips to optimize the use of labels in Power BI reports

5/13/2020 • 2 minutes to read • Edit Online

This article targets you as a report author designing Power BI reports, working with visuals that have labels.

Watch the video demonstrating the top four tips to optimize the use of labels in your Power BI reports.

Tips

In summary, the top four tips to optimize the use of labels in Power BI reports include:

1. Adjust label position
2. Adjust label color for contrast
3. Format labels for easy tracking
4. Avoid overwhelming labels

Next steps

For more information related to this article, check out the following resources:

- Tips for creating stunning reports
- biDezine video: Top 4 Tips to Optimize the Use of Labels in Power BI
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Tips to format and implement legends in Power BI reports

5/13/2020 • 2 minutes to read • Edit Online

This article targets you as a report author designing Power BI reports, when formatting and implementing legends.

Watch the video demonstrating the top six tips to format and implement legends in your Power BI reports.

Tips

In summary, the top six tips to format and implement legends in Power BI reports include:

1. Sort legend variables
2. Emulate a legend
3. Size and positioning legends
4. Solve truncated legends
5. Match style format options
6. Craft comprehensive visuals with context

Next steps

For more information related to this article, check out the following resources:

- Tips for creating stunning reports
- biDezine video: Top 6 Tips for Legend Formatting & Implementation in Power BI
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Tips to optimize visual colors in Power BI reports

5/13/2020 • 2 minutes to read • Edit Online

This article targets you as a report author designing Power BI reports, configuring visual colors.

Watch the video demonstrating the top eight tips to optimize visual colors in your Power BI reports.

Tips

In summary, the top eight tips to optimize visual colors in Power BI reports include:

1. Apply contrast within visuals
2. Implement optimal color palettes
3. Highlight important data with color
4. Consider color vision deficiencies
5. Use color with purpose
6. Choose appropriate color scales
7. Use color to differentiate key elements
8. Distinguish color and data

Next steps

For more information related to this article, check out the following resources:

- Tips for creating stunning reports
- biDezine video: Top 8 Tips To Optimize Visual Colors in Power BI
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Tips to optimize visual colors in Power BI reports

5/13/2020 • 2 minutes to read • Edit Online

This article targets you as a report author designing Power BI reports, configuring visual colors.

Watch the video demonstrating the top eight tips to optimize visual colors in your Power BI reports.

Tips

In summary, the top eight tips to optimize visual colors in Power BI reports include:

1. Apply contrast within visuals
2. Implement optimal color palettes
3. Highlight important data with color
4. Consider color vision deficiencies
5. Use color with purpose
6. Choose appropriate color scales
7. Use color to differentiate key elements
8. Distinguish color and data

Next steps

For more information related to this article, check out the following resources:

- Tips for creating stunning reports
- biDezine video: Top 8 Tips To Optimize Visual Colors in Power BI
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Tips to sort and distribute data plots in Power BI reports

5/13/2020 • 2 minutes to read • Edit Online

This article targets you as a report author designing Power BI reports, when using data plot visuals.

Watch the video demonstrating the top nine tips to sort and distribute data plots in Power BI reports.

Tips

In summary, the top nine tips to sort and distribute data plots in Power BI reports include:

1. Sort nominal categories
2. Sort interval categories
3. Sort ordinal categories
4. Visualize distant plots
5. Visualize crowded categorical plots
6. Visualize crowded time plots
7. Distribute multiple dimensions
8. Categorize data plots
9. Differentiate value plots

Next steps

For more information related to this article, check out the following resources:

- Tips for creating stunning reports
- biDezine video: Top 9 Tips to Sort and Distribute data plots in Power BI
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Tips to improve analysis with shapes, images, and icons in Power BI reports

5/13/2020 • 2 minutes to read • Edit Online

This article targets you as a report author designing Power BI reports, enhancing visuals with shapes, images, or icons.

Watch the video demonstrating the top four tips to improve analysis with shapes, images, and icons in Power BI reports.

Tips

In summary, the top four tips to improve analysis with shapes, images, and icons in Power BI reports include:

1. Design reports for accessibility
2. Use plot area images for context
3. Use images to convey information
4. Add icons to enrich report designs

Next steps

For more information related to this article, check out the following resources:

- Tips for creating stunning reports
- biDezine video: Top 4 Tips to Improve Analysis with Pictograms in Power BI
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

When to use paginated reports in Power BI

5/11/2020 • 5 minutes to read • Edit Online

This article targets you as a report author who designs reports for Power BI. It provides suggestions to help you choose when to develop Power BI paginated reports.

> **NOTE**
> Publishing Power BI paginated reports requires a Power BI Premium subscription. Reports will render only when they're in a workspace on a dedicated capacity that has the Paginated Reports workload enabled.

Power BI paginated reports are optimized for **printing**, or **PDF generation**. They also provide you with the ability to produce highly formatted, pixel-perfect layouts. So, paginated reports are ideal for operational reports, like sales invoices.

In contrast, Power BI reports are optimized for **exploration and interactivity**. Also, they can present your data using a comprehensive range of ultra-modern visuals. Power BI reports, therefore, are ideal for analytic reports, enabling your report users to explore data, and to discover relationships and patterns.

We recommend you consider using a Power BI paginated report when:

- You know the report must be printed, or output as a PDF document.
- Data grid layouts could expand and overflow. Consider that a table, or matrix, in a Power BI report can't dynamically resize to display all data—it will provide scroll bars instead. But, if printed, it won't be possible to scroll to reveal any out-of-view data.
- Power BI paginated features and capabilities work in your favor. Many such report scenarios are described later in this article.

Legacy reports

When you already have SQL Server Reporting Services (SSRS) Report Definition Language (RDL) reports, you can choose to redevelop them as Power BI reports, or migrate them as paginated reports to Power BI. For more information, see Migrate SQL Server Reporting Services reports to Power BI.

Once published to a Power BI workspace, paginated reports are available side by side with Power BI reports. They can then be easily distributed using Power BI apps.

You might consider redeveloping SSRS reports, rather than migrating them. It's especially true for those reports that are intended to deliver analytic experiences. In these cases, Power BI reports will likely deliver better report user experiences.

Paginated report scenarios

There are many compelling scenarios when you might favor developing a Power BI paginated report. Many are features or capabilities not supported by Power BI reports.

- **Print-ready**: Paginated reports are optimized for printing, or PDF generation. When necessary, data regions can expand and overflow to multiple pages in a controlled way. Your report layouts can define margins, and page headers and footers.
- **Render formats**: Power BI can render paginated reports in different formats. Formats include Microsoft Excel, Microsoft Word, Microsoft PowerPoint, PDF, CSV, XML, and MHTML. (The MHTML format is used by the Power BI

service to render reports.) Your report users can decide to export in the format that suits them.

- **Precision layout**: You can design highly formatted, pixel-perfect layouts—to the exact size and location configured in fractions of inches, or centimeters.
- **Dynamic layout**: You can produce highly responsive layouts by setting many report properties to use VB.NET expressions. Expressions have access to many core .NET Framework libraries.
- **Render-specific layout**: You can use expressions to modify the report layout based on the rendering format applied. For example, you can design the report to disable toggling visibility (to achieve drill down and drill up) when it's rendered using a non-interactive format, like PDF.
- **Native queries**: You don't need to first develop a Power BI dataset. It's possible to author native queries (or use stored procedures) for any supported data source. Queries can include parameterization.
- **Graphic query designers**: Power BI Report Builder includes graphic query designers to help you write, and test, your dataset queries.
- **Static datasets**: You can define a dataset, and enter data directly into your report definition. This capability is especially useful to support a demo, or for delivering a proof of concept (POC).
- **Data integration**: You can combine data from different data sources, or with static datasets. It's done by creating custom fields using VB.NET expressions.
- **Parameterization**: You can design highly customized parameterization experiences, including data-driven, and cascading parameters. It's also possible to define parameter defaults. These experiences can be designed to help your report users quickly set appropriate filters. Also, parameters don't need to filter report data; they can be used to support "what-if" scenarios, or dynamic filtering or styling.
- **Image data**: Your report can render images when they're stored in binary format in a data source.
- **Custom code**: You can develop code blocks of VB.NET functions in your report, and use them in any report expression.
- **Subreports**: You can embed other Power BI paginated reports (from the same workspace) into your report.
- **Flexible data grids**: You have fine-grained control of grid layouts by using the tablix data region. It supports complex layouts, too, including nested and adjacent groups. And, it can be configured to repeat headings when printed over multiple pages. Also, it can embed a subreport or other visualizations, including data bars, sparklines, and indicators.
- **Spatial data types**: The map data region can visualize SQL Server spatial data types. So, the GEOGRAPHY and GEOMETRY data types can be used to visualize points, lines, or polygons. It's also possible to visualize polygons defined in ESRI shape files.
- **Modern gauges**: Radial and linear gauges can be used to display KPI values and status. They can even be embedded into grid data regions, repeating within groups.
- **HTML rendering**: You can display richly formatted text when it's stored as HTML.
- **Mail merge**: You can use text box placeholders to inject data values into text. This way, you can produce a mail merge report.
- **Interactivity features**: Interactive features include toggling visibility (to achieve drill down and drill up), links, interactive sorting, and tooltips. You can also add links that drillthrough to Power BI reports, or other Power BI paginated reports. Links can even jump to another location within the same report.
- **Subscriptions**: Power BI can deliver paginated reports on a schedule as emails, with report attachments in any supported format.
- **Per-user layouts**: You can create responsive report layouts based on the authenticated user who opens the report. You can design the report to filter data differently, hide data regions or visualizations, apply different formats, or set user-specific parameter defaults.

Next steps

For more information related to this article, check out the following resources:

- What are paginated reports in Power BI Premium?

- Migrate SQL Server Reporting Services reports to Power BI
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Data retrieval guidance for paginated reports

5/13/2020 • 10 minutes to read • Edit Online

This article targets you as a report author designing Power BI paginated reports. It provides recommendations to help you design effective and efficient data retrieval.

Data source types

Paginated reports natively support both relational and analytic data sources. These sources are further categorized, as either cloud-based or on-premises. On-premises data sources—whether hosted on-premises, or in a virtual machine—require a data gateway so Power BI can connect. Cloud-based means that Power BI can connect directly using an Internet connection.

If you can choose the data source type (possibly the case in a new project), we recommend that you use cloud-based data sources. Paginated reports can connect with lower network latency, especially when the data sources reside in the same region as your Power BI tenant. Also, it's possible to connect to these sources by using Single Sign-On (SSO). It means the report user's identity can flow to the data source, allowing per-user row-level permissions to be enforced. Currently, SSO isn't supported for on-premises data sources (meaning SQL Server Analysis Services cannot enforce per-user row-level permissions).

> **NOTE**
>
> While it's currently not possible to connect to on-premises databases using SSO, you can still enforce row-level permissions. It's done by passing the **UserID** built-in field to a dataset query parameter. The data source will need to store User Principal Name (UPN) values in a way that it can correctly filter query results.
>
> For example, consider that each salesperson is stored as a row in the **Salesperson** a table. The table has columns for UPN, and also the salesperson's sales region. At query time, the table is filtered by the UPN of the report user, and it's also related to sales facts using an inner join. This way, the query effectively filters sales fact rows to those of the report user's sales region.

Relational data sources

Generally, relational data sources are well suited to operational style reports, like sales invoices. They're also suited for reports that need to retrieve very large datasets (in excess of 10,000 rows). Relational data sources can also define stored procedures, which can be executed by report datasets. Stored procedures deliver several benefits:

- Parameterization
- Encapsulation of programming logic, allowing for more complex data preparation (for example, temporary tables, cursors, or scalar user-defined functions)
- Improved maintainability, allowing stored procedure logic to be easily updated. In some cases, it can be done without the need to modify and republish paginated reports (providing column names and data types remain unchanged).
- Better performance, as their execution plans are cached for reuse
- Reuse of stored procedures across multiple reports

In Power BI Report Builder, you can use the relational query designer to graphically construct a query statement—but only for Microsoft data sources.

Analytic data sources

Analytic data sources are well suited to both operational and analytic reports, and can deliver fast summarized

query results even over very large data volumes. Model measures and KPIs can encapsulate complex business rules to achieve summarization of data. These data sources, however, are not suited to reports that need to retrieve very large datasets (in excess of 10,000 rows).

In Power BI Report Builder, you have a choice of two query designers: The Analysis Services DAX query designer, and the Analysis Services MDX query designer. These designers can be used for Power BI dataset data sources, or any SQL Server Analysis Services or Azure Analysis Services model—tabular or multidimensional.

We suggest you use the DAX query designer—providing it entirely meets your query needs. If the model doesn't define the measures you need, you'll need to switch to query mode. In this mode, you can customize the query statement by adding expressions (to achieve summarization).

The MDX query designer requires your model to include measures. The designer has two capabilities not supported by the DAX query designer. Specifically, it allows you to:

- Define query-level calculated members (in MDX).
- Configure data regions to request server aggregates in non-detail groups. If your report needs to present summaries of semi- or non-additive measures (like time intelligence calculations, or distinct counts), it will likely be more efficient to use server aggregates than to retrieve low-level detail rows and have the report compute summarizations.

Query result size

In general, it's best practice to retrieve only the data required by your report. So, don't retrieve columns or rows that aren't required by the report.

To limit rows, you should always apply the most restrictive filters, and define aggregate queries. Aggregate queries group and summarize source data to retrieve higher-grain results. For example, consider that your report needs to present a summary of salesperson sales. Instead of retrieving all sales order rows, create a dataset query that groups by salesperson, and summarizes sales for each group.

Expression-based fields

It's possible to extend a report dataset with fields based on expressions. For example, if your dataset sources customer first name and last name, you might want a field that concatenates the two fields to produce the customer full name. To achieve this calculation, you have two options. You can:

- Create a *calculated field*, which is a dataset field based on an expression.
- Inject an expression directly into the dataset query (using the native language of your data source), which results in a regular dataset field.

We recommend the latter option, whenever possible. There are two good reasons why injecting expressions directly into your dataset query is better:

- It's possible your data source is optimized to evaluate the expression more efficiently than Power BI (it's especially the case for relational databases).
- Report performance is improved because there's no need for Power BI to materialize calculated fields prior to report rendering. Calculated fields can noticeably extend report render time when datasets retrieve a large number of rows.

Field names

When you create a dataset, its fields are automatically named after the query columns. It's possible these names aren't friendly or intuitive. It's also possible that source query column names contain characters prohibited in Report Definition Language (RDL) object identifiers (like spaces and symbols). In this case, the prohibited characters are replaced with an underscore character (_).

We recommend that you first verify that all field names are friendly, concise, yet still meaningful. If not, we suggest you rename them *before* you commence the report layout. It's because renamed fields don't ripple changes through to the expressions used in your report layout. If you do decide to rename fields after you've commenced the report layout, you'll need to find and update all broken expressions.

Filter vs parameter

It's likely that your paginated report designs will have report parameters. Report parameters are commonly used to prompt your report user to filter the report. As a paginated report author, you have two ways to achieve report filtering. You can map a report parameter to:

- A dataset *filter*, in which case the report parameter value(s) are used to filter the data already retrieved by the dataset.
- A dataset *parameter*, in which case the report parameter value(s) are injected into the native query sent to the data source.

> **NOTE**
> All report datasets are cached on a *per-session basis* for up to 10 minutes *beyond their last use*. A dataset can be re-used when submitting new parameter values (filtering), rendering the report in a different format, or interacting with the report design in some way, like toggling visibility, or sorting.

Consider, then, an example of a sales report that has a single report parameter to filter the report by a single year. The dataset retrieves sales for *all years*. It does so because the report parameter maps to the dataset filters. The report displays data for the requested year, which is a subset of the dataset data. When the report user changes the report parameter to a different year—and then views the report—Power BI doesn't need to retrieve any source data. Instead, it applies a different filter to the already-cached dataset. Once the dataset is cached, filtering can be very fast.

Now, consider a different report design. This time the report design maps the sales year report parameter to a dataset parameter. This way, Power BI injects the year value into the native query, and the dataset retrieves data only for that year. Each time the report user changes the year report parameter value—and then views the report—the dataset retrieves a new query result for just that year.

Both design approaches can filter report data, and both designs can work well for your report designs. An optimized design, however, will depend on the anticipated volumes of data, data volatility, and the anticipated behaviors of your report users.

We recommend *dataset filtering* when you anticipate a different subset of the dataset rows will be reused many times (thereby saving rendering time because new data doesn't need to be retrieved). In this scenario, you recognize that the cost of retrieving a larger dataset can be traded off against the number of times it will be reused. So, it's helpful for queries that are time consuming to generate. But take care—caching large datasets on a per-user basis may negatively impact on performance, and capacity throughput.

We recommend *dataset parameterization* when you anticipate it's unlikely that a different subset of dataset rows will be requested—or, when the number of the dataset rows to be filtered is likely to be very large (and inefficient to cache). This design approach work well, too, when your data store is volatile. In this case, each report parameter value change will result in the retrieval of up-to-date data.

Non-native data sources

If you need to develop paginated reports based on data sources that aren't natively supported by paginated reports, you can first develop a Power BI Desktop data model. This way, you can connect to over 100 Power BI data sources. Once published to the Power BI service, you can then develop a paginated report that connects to the

Power BI dataset.

Data integration

If you need to combine data from multiple data sources, you have two options:

- **Combine report datasets**: If the data sources are natively supported by paginated reports, you can consider creating calculated fields that use the Lookup or LookupSet Report Builder functions.
- **Develop a Power BI Desktop model**: It's likely more efficient, however, that you develop a data model in Power BI Desktop. You can use Power Query to combine queries based on any supported data source. Once published to the Power BI service, you can then develop a paginated report that connects to the Power BI dataset.

SQL Server complex data types

Because SQL Server is an on-premises data source, Power BI must connect via a gateway. The gateway, however, doesn't support retrieving data for complex data types. Complex data types include built-in types like the GEOMETRY and GEOGRAPHY spatial data types, and hierarchyid. They can also include user-defined types implemented through a class of an assembly in the Microsoft.NET Framework common language runtime (CLR).

Plotting spatial data and analytics in the map visualization requires SQL Server spatial data. Therefore, it's not possible to work with the map visualization when SQL Server is your data source. To be clear, it will work if your data source is Azure SQL Database because Power BI doesn't connect via a gateway.

Data-related images

Images can be used to add logos or pictures to your report layout. When images relate to the rows retrieved by a report dataset, you have two options:

- It's possible that image data can also be retrieved from your data source (if already stored in a table).
- When the images are stored on a web server, you can use a dynamic expression to create the image URL path.

For more information and suggestions, see Image guidance for paginated reports.

Redundant data retrieval

It's possible your report retrieves redundant data. This can happen when you delete dataset query fields, or the report has unused datasets. Avoid these situations, as they result in an unnecessary burden on your data sources, the network, and Power BI capacity resources.

Deleted query fields

On the **Fields** page of the **Dataset Properties** window, it's possible to delete dataset *query fields* (query fields map to columns retrieved by the dataset query). However, Report Builder doesn't remove corresponding columns from the dataset query.

If you need to delete query fields from your dataset, we recommend you remove the corresponding columns from the dataset query. Report Builder will automatically remove any redundant query fields. If you do happen to delete query fields, be sure to also modify the dataset query statement to remove the columns.

Unused datasets

When a report is run, all datasets are evaluated—even if they're not bound to report objects. For this reason, be sure to remove any test or development datasets before you publish a report.

Next steps

For more information related to this article, check out the following resources:

- Supported data sources for Power BI paginated reports
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Image use guidance for paginated reports

3/7/2020 • 2 minutes to read • Edit Online

This article targets you as a report author designing Power BI paginated reports. It provides suggestions when working with images. Commonly, images in report layouts can display a graphic like a company logo, or pictures.

Images can be stored in three different locations:

- Within the report (embedded)
- On a web server
- In a database, which can be retrieved by a dataset

They can then be used in a variety of scenarios in your report layouts:

- Free-standing logo, or picture
- Pictures associated with rows of data
- Background for certain report items:
  - Report body
  - Textbox
  - Rectangle
  - Tablix data region (table, matrix, or list)

Suggestions

Consider the following suggestions to deliver professional report layouts, ease of maintenance, and optimized report performance:

- **Use smallest possible size**: We recommend you prepare images that are small in size, yet still look sharp, and crisp. It's all about a balance between quality and size. Consider using a graphics editor (like MS Paint) to reduce the image file size.

- **Avoid embedded images**: First, embedded images can bloat the report file size, which can contribute to slower report rendering. Second, embedded images can quickly become a maintenance nightmare if you need to update many report images (as might be the case should your company logo change).

- **Use web server storage**: Storing images on a web server is a good option, especially for the company logo, which may be sourced from the company website. However, take care if your report users will access reports outside your network. In this case, be sure that the images are available over the Internet.

  When images relate to your data (like pictures of your salespeople), name image files so a report expression can dynamically produce the image URL path. For example, you could name the salespeople pictures using each salesperson's employee number. Providing the report dataset retrieves the employee number, you can write an expression to produce the full image URL path.

- **Use database storage**: When a relational database stores image data, it makes sense to source the image data directly from the database tables—especially when the images are not too large.

- **Appropriate background images**: If you decide to use background images, take care not to distract the report user from your report data.

  Also, be sure to use *watermark styled images*. Generally, watermark styled images have a transparent background (or have the same background color used by the report). They also use faint colors. Common

examples of watermark styled images include the company logo, or sensitivity labels like "Draft" or "Confidential".

Next steps

For more information related to this article, check out the following resources:

- What are paginated reports in Power BI Premium?
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Use cascading parameters in paginated reports

3/7/2020 • 6 minutes to read • Edit Online

This article targets you as a report author designing Power BI paginated reports. It provides scenarios for designing cascading parameters. Cascading parameters are report parameters with dependencies. When a report user selects a parameter value (or values), it's used to set available values for another parameter.

> **NOTE**
> An introduction to cascading parameters, and how to configure them, isn't covered in this article. If you're not completely familiar with cascading parameters, we recommend you first read Add Cascading Parameters to a Report (Report Builder and SSRS).

Design scenarios

There are two design scenarios for using cascading parameters. They can be effectively used to:

- Filter *large sets* of items
- Present *relevant* items

Example database

The examples presented in this article are based on an Azure SQL Database. The database records sales operations, and contains various tables storing resellers, products, and sales orders.

A table named **Reseller** stores one record for each reseller, and it contains many thousands of records. The **Reseller** table has these columns:

- ResellerCode (integer)
- ResellerName
- Country-Region
- State-Province
- City
- PostalCode

There's a table named **Sales**, too. It stores sales order records, and has a foreign key relationship to the **Reseller** table, on the **ResellerCode** column.

Example requirement

There's a requirement to develop a Reseller Profile report. The report must be designed to display information for a single reseller. It's not appropriate to have the report user enter a reseller code, as they rarely memorize them.

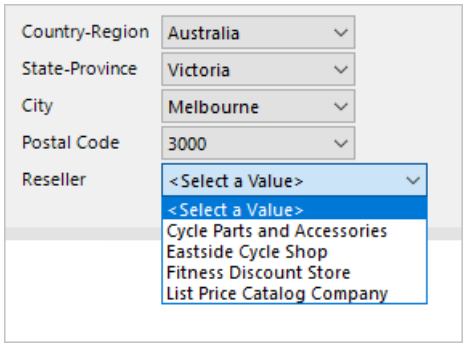
Filter large sets of items

Let's take a look at three examples to help you limit large sets of available items, like resellers. They are:

- Filter by related columns
- Filter by a grouping column
- Filter by search pattern

Filter by related columns

In this example, the report user interacts with five report parameters. They must select country-region, state-province, city, and then postal code. A final parameter then lists resellers that reside in that geographic location.



Here's how you can develop the cascading parameters:

1. Create the five report parameters, ordered in the correct sequence.
2. Create the **CountryRegion** dataset that retrieves distinct country-region values, using the following query statement:

   ```
   SELECT DISTINCT
     [Country-Region]
   FROM
     [Reseller]
   ORDER BY
     [Country-Region]
   ```

3. Create the **StateProvince** dataset that retrieves distinct state-province values for the selected country-region, using the following query statement:

   ```
   SELECT DISTINCT
     [State-Province]
   FROM
     [Reseller]
   WHERE
     [Country-Region] = @CountryRegion
   ORDER BY
     [State-Province]
   ```

4. Create the **City** dataset that retrieves distinct city values for the selected country-region and state-province, using the following query statement:

   ```
   SELECT DISTINCT
     [City]
   FROM
     [Reseller]
   WHERE
     [Country-Region] = @CountryRegion
     AND [State-Province] = @StateProvince
   ORDER BY
     [City]
   ```

5. Continue this pattern to create the **PostalCode** dataset.
6. Create the **Reseller** dataset to retrieve all resellers for the selected geographic values, using the following query statement:

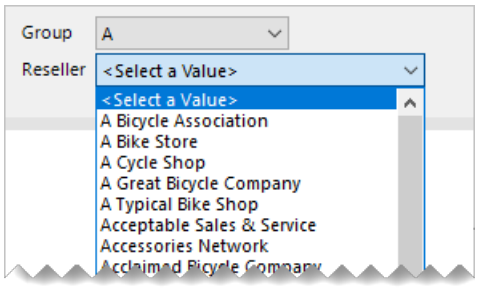
```
SELECT
  [ResellerCode],
  [ResellerName]
FROM
  [Reseller]
WHERE
  [Country-Region] = @CountryRegion
  AND [State-Province] = @StateProvince
  AND [City] = @City
  AND [PostalCode] = @PostalCode
ORDER BY
  [ResellerName]
```

7. For each dataset except the first, map the query parameters to the corresponding report parameters.

> **NOTE**
>
> All query parameters (prefixed with the @ symbol) shown in these examples could be embedded within SELECT statements, or passed to stored procedures.
>
> Generally, stored procedures are a better design approach. It's because their query plans are cached for quicker execution, and they allow you develop more sophisticated logic, when needed. However, they aren't currently supported for gateway relational data sources, which means SQL Server, Oracle, and Teradata.
>
> Lastly, you should always ensure suitable indexes exist to support efficient data retrieval. Otherwise, your report parameters could be slow to populate, and the database could become overburdened. For more information about SQL Server indexing, see SQL Server Index Architecture and Design Guide.

Filter by a grouping column

In this example, the report user interacts with a report parameter to select the first letter of the reseller. A second parameter then lists resellers when the name commences with the selected letter.



Here's how you can develop the cascading parameters:

1. Create the **ReportGroup** and **Reseller** report parameters, ordered in the correct sequence.
2. Create the **ReportGroup** dataset to retrieve the first letters used by all resellers, using the following query statement:

```
SELECT DISTINCT
  LEFT([ResellerName], 1) AS [ReportGroup]
FROM
  [Reseller]
ORDER BY
  [ReportGroup]
```

3. Create the **Reseller** dataset to retrieve all resellers that commence with the selected letter, using the following query statement:

```
SELECT
  [ResellerCode],
  [ResellerName]
FROM
  [Reseller]
WHERE
  LEFT([ResellerName], 1) = @ReportGroup
ORDER BY
  [ResellerName]
```

4. Map the query parameter of the **Reseller** dataset to the corresponding report parameter.

It's more efficient to add the grouping column to the **Reseller** table. When persisted and indexed, it delivers the best result. For more information, see Specify Computed Columns in a Table.

```
ALTER TABLE [Reseller]
ADD [ReportGroup] AS LEFT([ResellerName], 1) PERSISTED
```

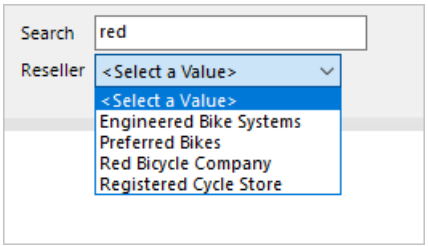
This technique can deliver even greater potential. Consider the following script that adds a new grouping column to filter resellers by *pre-defined bands of letters*. It also creates an index to efficiently retrieve the data required by the report parameters.

```
ALTER TABLE [Reseller]
ADD [ReportGroup2] AS CASE
  WHEN [ResellerName] LIKE '[A-C]%' THEN 'A-C'
  WHEN [ResellerName] LIKE '[D-H]%' THEN 'D-H'
  WHEN [ResellerName] LIKE '[I-M]%' THEN 'I-M'
  WHEN [ResellerName] LIKE '[N-S]%' THEN 'N-S'
  WHEN [ResellerName] LIKE '[T-Z]%' THEN 'T-Z'
  ELSE '[Other]'
END PERSISTED
GO

CREATE NONCLUSTERED INDEX [Reseller_ReportGroup2]
ON [Reseller] ([ReportGroup2]) INCLUDE ([ResellerCode], [ResellerName])
GO
```

Filter by search pattern

In this example, the report user interacts with a report parameter to enter a search pattern. A second parameter then lists resellers when the name contains the pattern.



Here's how you can develop the cascading parameters:

1. Create the **Search** and **Reseller** report parameters, ordered in the correct sequence.
2. Create the **Reseller** dataset to retrieve all resellers that contain the search text, using the following query statement:

```
SELECT
  [ResellerCode],
  [ResellerName]
FROM
  [Reseller]
WHERE
  [ResellerName] LIKE '%' + @Search + '%'
ORDER BY
  [ResellerName]
```

3. Map the query parameter of the **Reseller** dataset to the corresponding report parameter.

> **TIP**
> You can improve upon this design to provide more control for your report users. It lets them define their own pattern matching value. For example, the search value "red%" will filter to resellers with names that *commence* with the characters "red".
>
> For more information, see LIKE (Transact-SQL).

Here's how you can let the report users define their own pattern.

```
WHERE
  [ResellerName] LIKE @Search
```

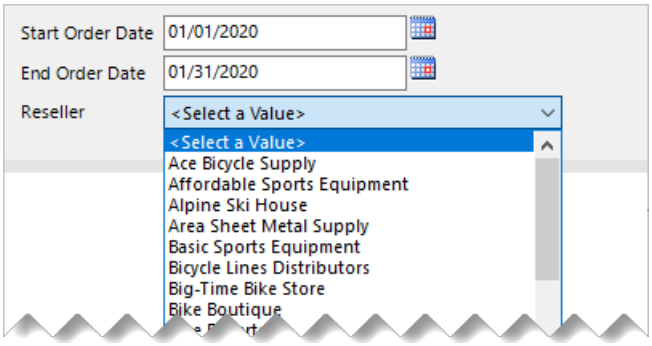
Many non-database professionals, however, don't know about the percentage (%) wildcard character. Instead, they're familiar with the asterisk (*) character. By modifying the WHERE clause, you can let them use this character.

```
WHERE
  [ResellerName] LIKE SUBSTITUTE(@Search, '%', '*')
```

Present relevant items

In this scenario, you can use fact data to limit available values. Report users will be presented with items where activity has been recorded.

In this example, the report user interacts with three report parameter. The first two set a date range of sales order dates. The third parameter then lists resellers where orders have been created during that time period.



Here's how you can develop the cascading parameters:

1. Create the **OrderDateStart**, **OrderDateEnd**, and **Reseller** report parameters, ordered in the correct sequence.

2. Create the **Reseller** dataset to retrieve all resellers that created orders in the date period, using the following

query statement:

```
SELECT DISTINCT
  [r].[ResellerCode],
  [r].[ResellerName]
FROM
  [Reseller] AS [r]
INNER JOIN [Sales] AS [s]
  ON [s].[ResellerCode] = [r].[ResellerCode]
WHERE
  [s].[OrderDate] >= @OrderDateStart
  AND [s].[OrderDate] < DATEADD(DAY, 1, @OrderDateEnd)
ORDER BY
  [r].[ResellerName]
```

Recommendations

We recommend you design your reports with cascading parameters, whenever possible. It's because they:

- Provide intuitive and helpful experiences for your report users
- Are efficient, because they retrieve smaller sets of available values

Be sure to optimize your data sources by:

- Using stored procedures, whenever possible
- Adding appropriate indexes for efficient data retrieval
- Materializing column values—and even rows—to avoid expensive query-time evaluations

Next steps

For more information related to this article, check out the following resources:

- Report parameters in Power BI Report Builder
- Add Cascading Parameters to a Report (Report Builder and SSRS)
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Avoid blank pages when printing paginated reports

3/7/2020 • 2 minutes to read • Edit Online

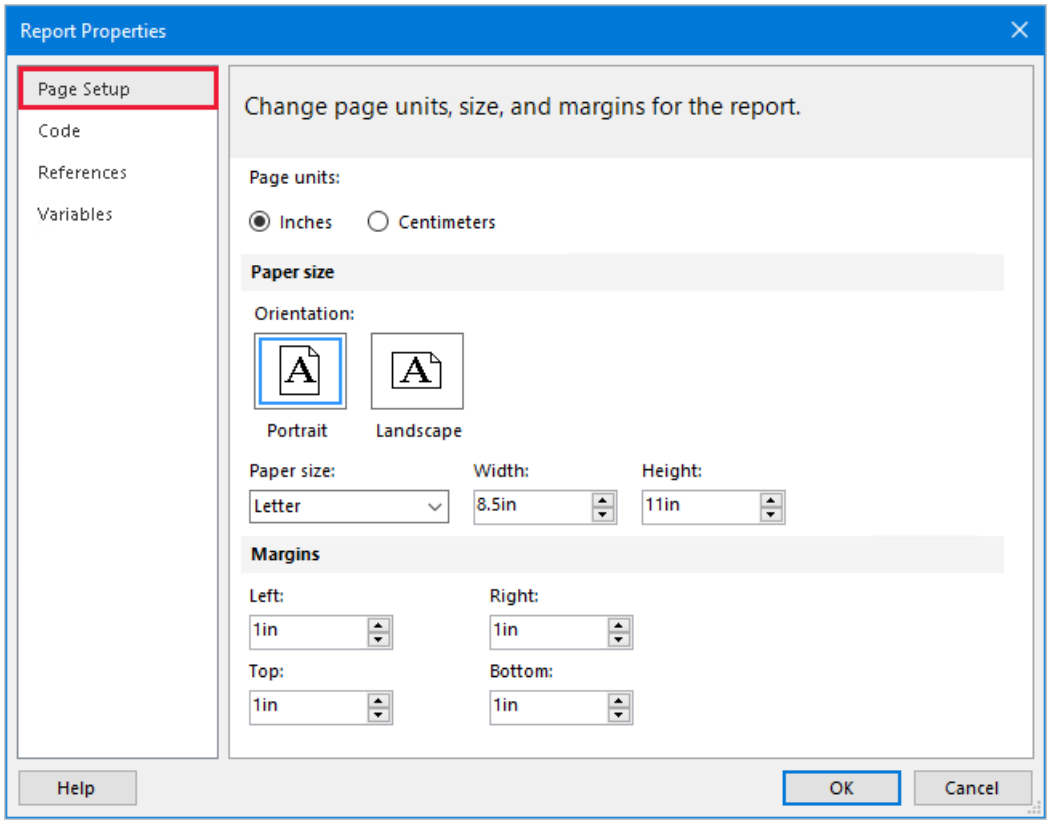
This article targets you as a report author designing Power BI paginated reports. It provides recommendations to help you avoid blank pages when your report is exported to a hard-page format—like PDF or Microsoft Word—or, is printed.

Page setup

Report page size properties determine the page orientation, dimensions, and margins. Access these report properties by:

- Using the report **Property Page**: Right-click the dark gray area outside the report canvas, and then select *Report Properties*.
- Using the **Properties** pane: Click the dark gray area outside the report canvas to select the report object. Ensure the **Properties** pane is open.

The **Page Setup** page of the report **Property Page** provides a friendly interface to view and update the page setup properties.



Ensure all page size properties are correctly configured:

PROPERTY	RECOMMENDATION
Page units	Select the relevant units—inches or centimeters.
Orientation	Select the correct option—portrait or landscape.

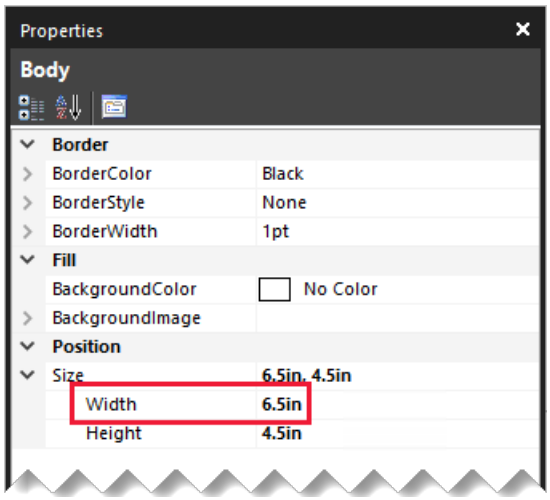
PROPERTY	RECOMMENDATION
Paper size	Select a paper size, or assign custom width and height values.
Margins	Set appropriate values for the left, right, top, and bottom margins.

Report body width

The page size properties determine the available space available for report objects. Report objects can be data regions, data visualizations, or other report items.

A common reason why blank pages are output, is the report body width *exceeds the available page space*.

You can only view and set the report body width using the **Properties** pane. First, click anywhere in an empty area of the report body.



Ensure the width value doesn't exceed available page width. Be guided by the following formula:

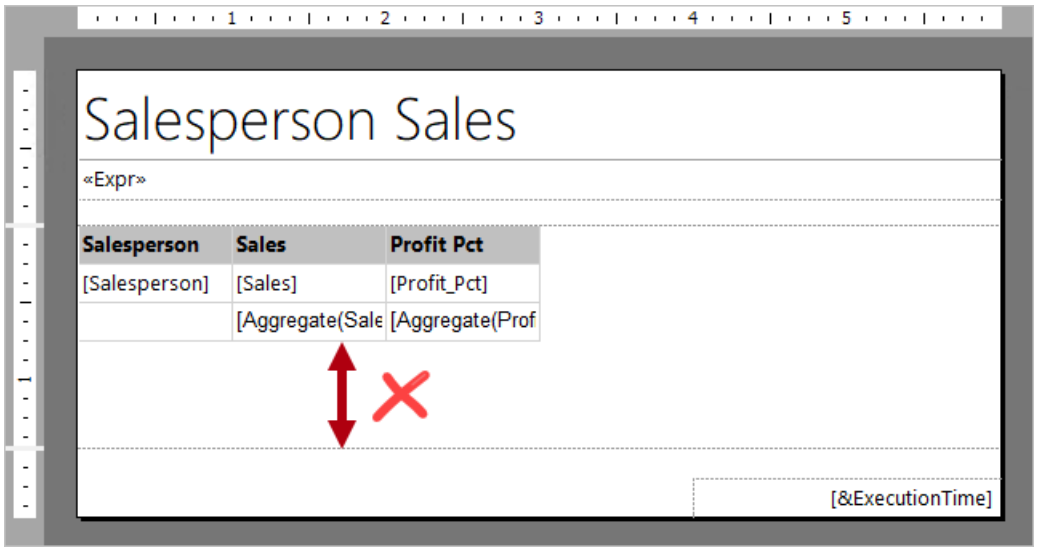
```
Report body width <= Report page width - (Left margin + Right margin)
```

> **NOTE**
>
> It's not possible to reduce the report body width when there are report objects already in the space you want to remove. You must first reposition or resize them before reducing the width.
>
> Also, the report body width can increase automatically when you add new objects, or resize or reposition existing objects. The report designer always widens the body to accommodate the position and size of its contained objects.

Report body height

Another reason why a blank page is output, is there's excess space in the report body, after the last object.

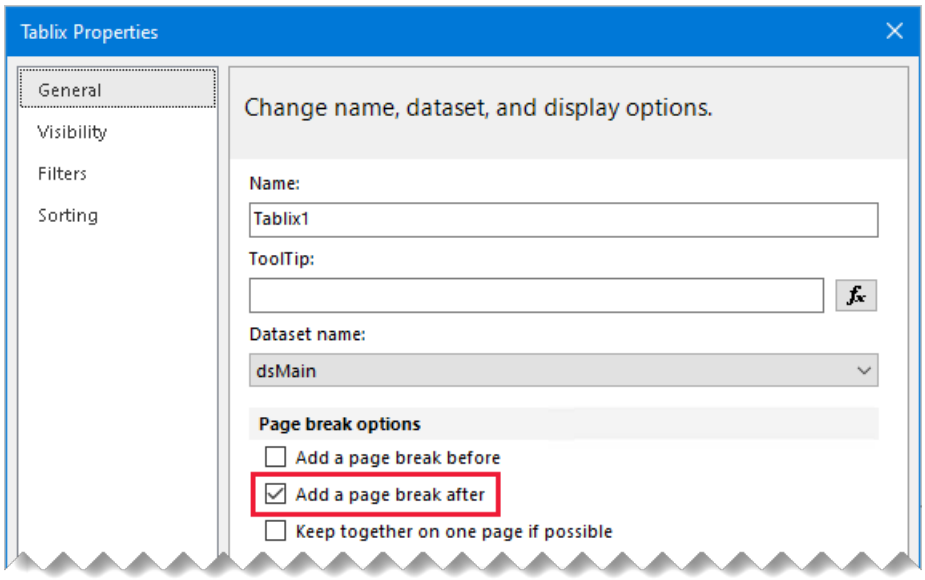
We recommend you always reduce the height of the body to remove any trailing space.



Page break options

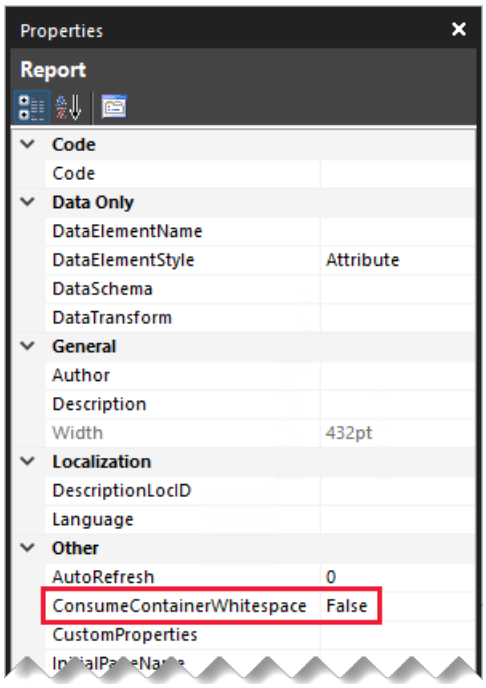
Each data region and data visualization has page break options. You can access these options in its property page, or in the **Properties** pane.

Ensure the **Add a page break after** property isn't unnecessarily enabled.



Consume Container Whitespace

If the blank page issue persists, you can also try disabling the report **ConsumeContainerWhitespace** property. It can only be set in the **Properties** pane.



By default, it's enabled. It directs whether minimum whitespace in containers, such as the report body or a rectangle, should be consumed. Only whitespace to the right of, and below, the contents is affected.

Printer paper size

Lastly, if you're printing the report to paper, ensure the printer has the correct paper loaded. The physical paper size should correspond to the report paper size.

Next steps

For more information related to this article, check out the following resources:

- What are paginated reports in Power BI Premium?
- Pagination in Power BI paginated reports
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Migrate SQL Server Reporting Services reports to Power BI

5/18/2020 • 8 minutes to read • Edit Online

This article targets SQL Server Reporting Services (SSRS) report authors and Power BI administrators. It provides you with guidance to help you migrate your Report Definition Language (RDL) reports to Power BI.

> **NOTE**
> It's only possible to migrate RDL reports. In Power BI, RDL reports are called *paginated reports*.

Guidance is divided into four stages. We recommend that you first read the entire article prior to migrating your reports.

1. Before you start
2. Pre-migration stage
3. Migration stage
4. Post-migration stage

You can achieve migration without downtime to your SSRS servers, or disruption to your report users. It's important to understand that no data or reports need to be removed. So, it means you can keep your current environment in place until you're ready for it to be retired.

Before you start

Before you start the migration, you should verify that your environment meets certain prerequisites. We'll describe these prerequisites, and also introduce you to a helpful migration tool.

Preparing for migration

As you prepare to migrate your reports to Power BI, first verify that your organization has a Power BI Premium subscription. This subscription is required to host and run your Power BI paginated reports.

Supported versions

You can migrate SSRS instances running on-premises, or on Virtual Machines hosted by cloud providers, like Azure.

The following list describes the SQL Server versions supported for migration to Power BI:

- SQL Server 2012
- SQL Server 2014
- SQL Server 2016
- SQL Server 2017
- SQL Server 2019

Migration from Power BI Report Server is possible, too.

Migration tool

We recommend you use the RDL Migration Tool to help prepare, and migrate your reports. This tool was developed by Microsoft to help customers migrate RDL reports from their SSRS servers to Power BI. It's available on GitHub, and it documents an end-to-end walkthrough of the migration scenario.

The tool automates the following tasks:

- Checks for unsupported data sources and unsupported report features
- Converts any *shared* resources to *embedded* resources:
  - Shared **data sources** become embedded data sources
  - Shared **datasets** become embedded datasets
- Publishes reports (that pass checks) as paginated reports, to a specified Power BI workspace (on a Premium capacity)

It doesn't modify or remove your existing reports. On completion, the tool outputs a summary of all actions completed—successful or unsuccessful.

Over time, the tool may be improved by Microsoft. The community is encouraged to contribute and help enhance it, too.

Pre-migration stage

After verifying that your organization meets the pre-requisites, you're ready to start the *Pre-migration* stage. This stage has three phases:

1. Discover
2. Assess
3. Prepare

Discover

The goal of the *Discover* phase is to identify your existing SSRS instances. This process involves scanning the network to identify all SQL Server instances in your organization.

You can use the Microsoft Assessment and Planning Toolkit. Also known as the "MAP Toolkit", it discovers and reports on your SQL Server instances, versions, and installed features. It's a powerful inventory, assessment, and reporting tool that can simplify your migration planning process.

Assess

Having discovered your SSRS instances, the goal of the *Assess* phase is to understand any SSRS reports—or server items—that can't be migrated.

Only RDL reports can be migrated from your SSRS servers to Power BI. Each migrated RDL report will become a Power BI paginated report.

The following SSRS item types, however, can't be migrated to Power BI:

- Shared data sources [1]
- Shared datasets [1]
- Resources, like image files
- KPIs (SSRS 2016, or later—Enterprise Edition only)
- Mobile reports (SSRS 2016, or later—Enterprise Edition only)
- Report models (deprecated)
- Report parts (deprecated)

[1] The RDL Migration Tool automatically converts shared data sources and shared datasets—providing they're using supported data sources.

If your RDL reports rely on features not yet supported by Power BI paginated reports, you can plan to redevelop them as Power BI reports. Even if your RDL reports can migrate, we recommend you consider modernizing them as Power BI reports, when it makes sense.

If your RDL reports need to retrieve data from *on-premises data sources*, they cannot use single sign-on (SSO). Currently, all data retrieval from these sources will be done by using the security context of the *gateway data source user account*. It's not possible for SQL Server Analysis Services (SSAS) to enforce row-level security (RLS) on a per-user basis.

Generally, Power BI paginated reports are optimized for **printing**, or **PDF generation**. Power BI reports are optimized for **exploration and interactivity**. For more information, see When to use paginated reports in Power BI.

Prepare

The goal of the *Prepare* phase involves getting everything ready. It covers setting up the Power BI environment, planning how you'll secure and publish your reports, and ideas for redeveloping SSRS items that won't migrate.

1. Ensure the Paginated Reports workload is enabled for your Power BI Premium capacity, and that it has sufficient memory.
2. Verify support for your report data sources, and set up a Power BI Gateway to allow connectivity with any on-premises data sources.
3. Become familiar with Power BI security, and plan how you'll reproduce your SSRS folders and permissions with Power BI workspaces and workspace roles.
4. Become familiar with Power BI sharing, and plan how you'll distribute content by publishing Power BI apps.
5. Consider using shared Power BI datasets in place of your SSRS shared data sources.
6. Use Power BI Desktop to develop mobile-optimized reports, possibly using the Power KPI custom visual in place of your SSRS mobile reports and KPIs.
7. Reevaluate the use of the **UserID** built-in field in your reports. If you rely on the **UserID** to secure report data, then understand that for paginated reports (when hosted in the Power BI service) it returns the User Principal Name (UPN). So, instead of returning the NT account name, for example *AW\mblythe*, the built-in field will return something like *m.blythe@adventureworks.com*. You will need to revise your dataset definitions, and possibly the source data. Once revised and published, we recommend you thoroughly test your reports to ensure data permissions work as expected.
8. Reevaluate the use of the **ExecutionTime** built-in field in your reports. For paginated reports (when hosted in the Power BI service), the built-in field returns the date/time *in Coordinated Universal Time (or UTC)*. It could impact on report parameter default values, and report execution time labels (typically added to report footers).
9. If your data source is SQL Server (on-premises), verify that reports aren't using map visualizations. The map visualization depends on SQL Server spatial data types, and these aren't supported by the gateway. For more information, see Data retrieval guidance for paginated reports (SQL Server complex data types).
10. Ensure your report authors have Power BI Report Builder installed, and that later releases can be easily distributed throughout your organization.

Migration stage

After preparing your Power BI environment and reports, you're ready for the *Migration* stage.

There are two migration options: *manual* and *automated*. Manual migration is suited to a small number of reports, or reports requiring modification before migration. Automated migration is suited to the migration of a large number of reports.

Manual migration

Anyone with permission to access to the SSRS instance and the Power BI workspace can manually migrate reports to Power BI. Here are the steps to follow:

1. Open the SSRS portal that contains the reports you want to migrate.
2. Download each report definition, saving the .rdl files locally.
3. Open *the latest version* of Power BI Report Builder, and connect to the Power BI service using your Azure AD

credentials.

4. Open each report in Power BI Report Builder, and then:
   a. Verify all data sources and datasets are embedded in the report definition, and that they're supported data sources.
   b. Preview the report to ensure it renders correctly.
   c. Choose the *Save As* option, and then select *Power BI service*.
   d. Select the workspace that will contain the report.
   e. Verify that the report saves. If certain features in your report design aren't yet supported, the save action will fail. You'll be notified of the reasons. You'll then need to revise your report design, and try saving again.

Automated migration

There are two options for automated migration. You can use:

- The RDL Migration Tool
- The publicly available APIs for SSRS and Power BI

The RDL Migration Tool has already been described in this article.

You can also use the publicly available SSRS and Power BI APIs to automate the migration of your content. While the RDL Migration Tool already uses these APIs, you can develop a custom tool suited to your exact requirements.

For more information about the APIs, see:

- Power BI REST API Reference
- SQL Server Reporting Services REST APIs

Post-migration stage

After you've successfully completed the migration, you're ready for the *Post-migration* stage. This stage involves working through a series of post-migration tasks to ensure everything is functioning correctly and efficiently.

Configure data sources

Once reports have been migrated to Power BI, you'll need to ensure their data sources are correctly set up. It can involve assigning to gateway data sources, and securely storing data source credentials. These actions aren't done by the RDL Migration Tool.

Review report performance

We highly recommended you complete the following actions to ensure the best possible report user experience:

1. Test the reports in each browser supported by Power BI to confirm the report renders correctly.
2. Run tests to compare report rending times in SSRS and Power BI. Check that Power BI reports render in an acceptable time.
3. If Power BI reports fail to render because of insufficient memory, allocate additional resources to the Power BI Premium capacity.
4. For long-rendering reports, consider having Power BI deliver them to your report users as email subscriptions with report attachments.
5. For Power BI reports based on Power BI datasets, review model designs to ensure they're fully optimized.

Reconcile issues

The Post-migration phase is crucial for reconciling any issues, and that you address any performance concerns. Adding the paginated reports workload to a capacity can contribute to slow performance—for paginated reports *and other content* stored in the capacity.

For more information about these issues, including specific steps to understand and mitigate them, see the

following articles:

- Optimizing Premium capacities
- Monitor Premium capacities within the app

Next steps

For more information about this article, check out the following resources:

- What are paginated reports in Power BI Premium?
- Data retrieval guidance for paginated reports
- When to use paginated reports in Power BI
- Paginated reports in Power BI: FAQ
- Online course: Paginated Reports in a Day
- Power BI Premium FAQ
- RDL Migration Tool
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Power BI partners are available to help your organization succeed with the migration process. To engage a Power BI partner, visit the Power BI partner portal.

Tenant admin settings guidance

5/13/2020 • 5 minutes to read • Edit Online

This article targets Power BI administrators who are responsible for setting up and configuring the Power BI environment in their organization.

We provide guidance for specific tenant settings that help improve the Power BI experience, or could expose your organization to risk. We recommend you always configure your tenant to align with your organization's policies and processes.

Tenant settings are managed in the Admin portal, and can be configured by a Power BI service administrator. Many tenant settings can restrict capabilities and features to a limited set of users. So, we recommend you first become familiar with the settings to plan the security groups you'll need. You might find that you can apply the same security group to multiple settings.

Improve Power BI experience

Publish "Get Help" information

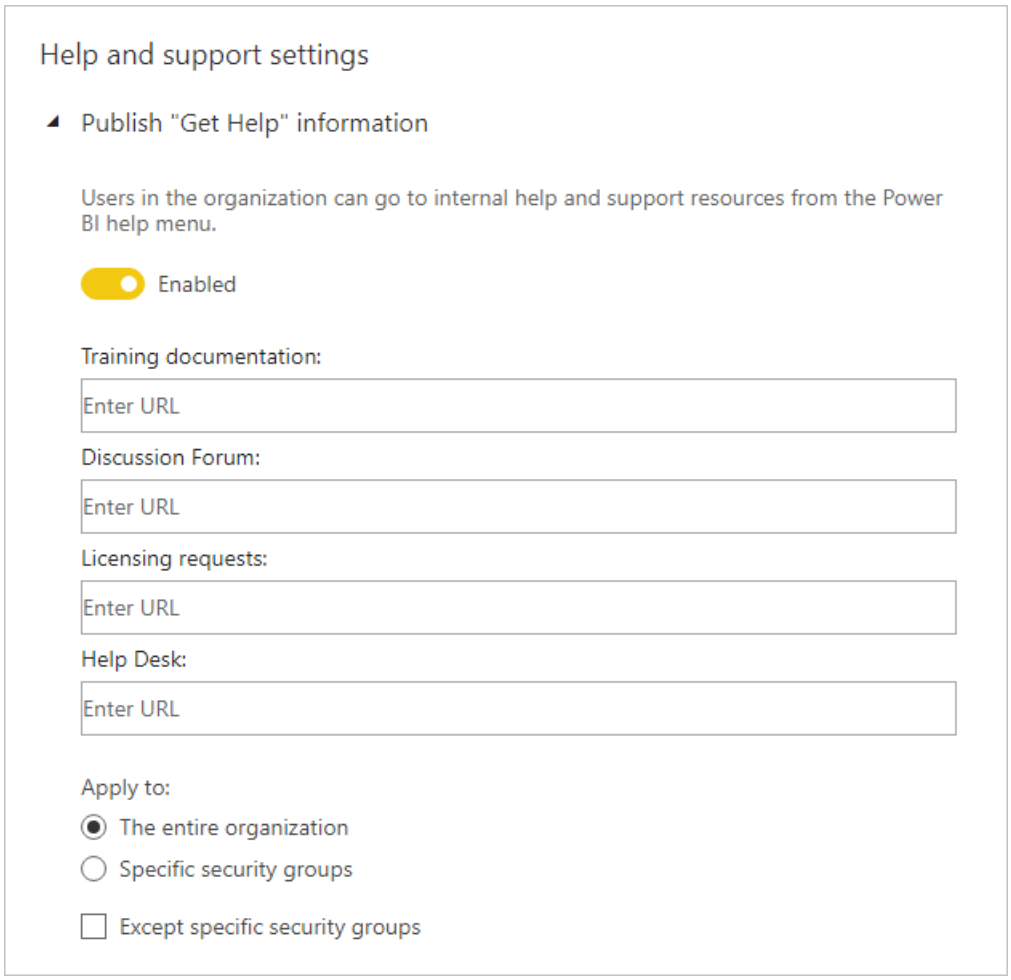
We encourage you to set up internal Power BI-related sites using Microsoft Teams, or other collaboration platform. These sites can be used to store training documentation, host discussions, make requests for licenses, or respond to help.

If you do so, we recommend you then enable the **Publish "Get Help" information** setting *for the entire organization*. It's found in the **Help and support settings** group. You can set URLs for your:

- Training documentation
- Discussion forum
- Licensing requests
- Help desk

These URLs will become available as links in the Power BI help menu.

> **NOTE**
>
> Supplying the **Licensing requests** URL will prevent individual users from signing up for the free 60-day Power BI Pro trial. Instead, they'll be directed to your internal site with information on how to acquire a license—Free or Pro.

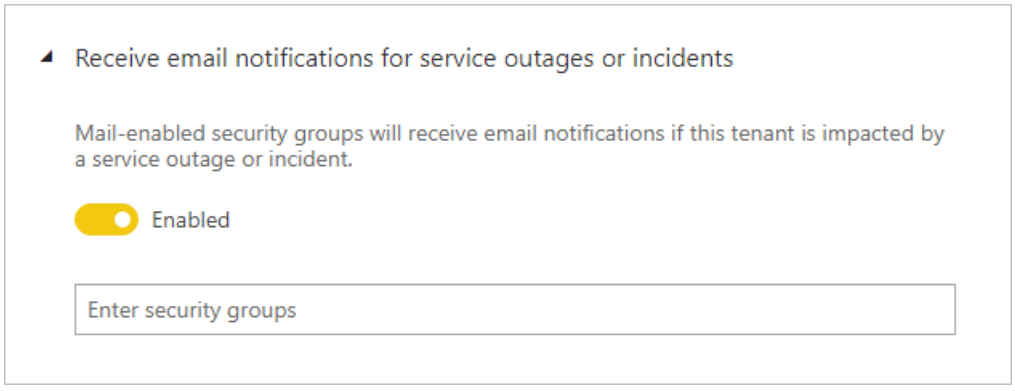


Manage risk

Receive email notification service outages or incidents

You can be notified by email if your tenant is impacted by a service outage or incident. This way, you can proactively respond to incidents.

We recommend you enable the **Receive email notification service outages or incidents** setting. It's found in the **Help and support settings** group. Assign one or more *mail-enabled* security groups.



Information protection

Information protection allows enforcing protection settings—such as encryption or watermarks—when exporting data from the Power BI service.

There are two tenant settings related to information protection. By default, both settings are disabled for the entire organization.

We recommend you enable these settings when you need to handle and protect sensitive data. For more information, see Data protection in Power BI.

Create workspaces

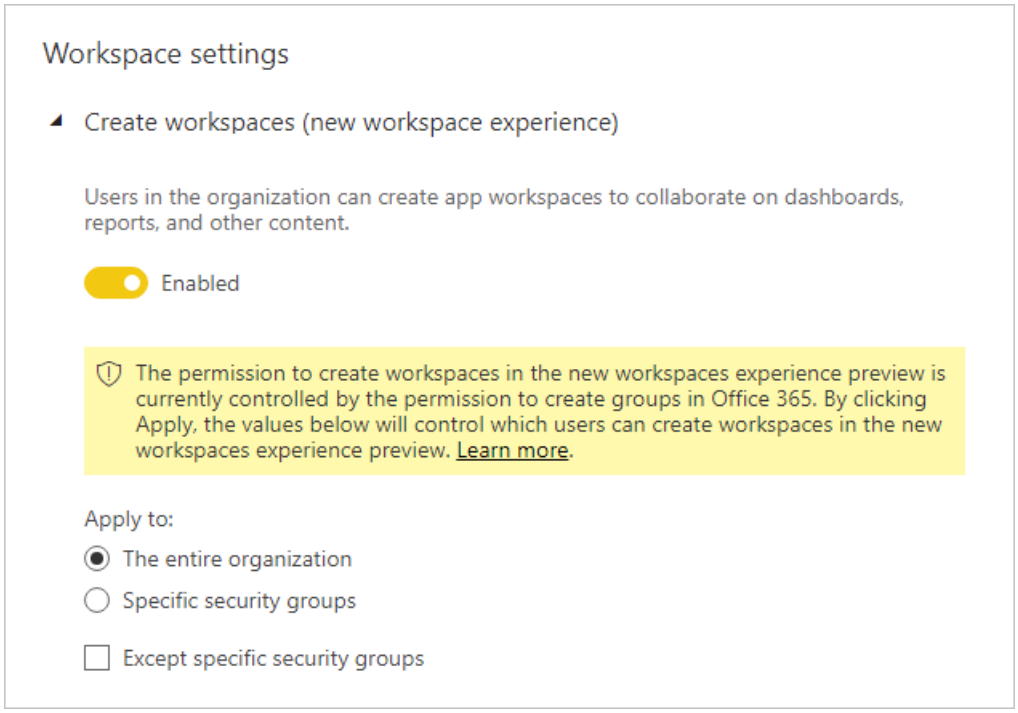
You can restrict users from creating workspaces. This way, you can govern what is created within your organization.

> **NOTE**
>
> Currently there's a transition period between the old workspace experience and the new. This tenant setting applies only to the new experience.

The **Create workspaces** setting is enabled by default for the entire organization. It's found in the **Workspace settings** group.

We recommend you assign one or more security groups. These groups can be granted *or denied* permission to create workspaces.

Be sure to include instructions in your documentation letting users (who don't have workspace creation rights) know how they can request a new workspace.

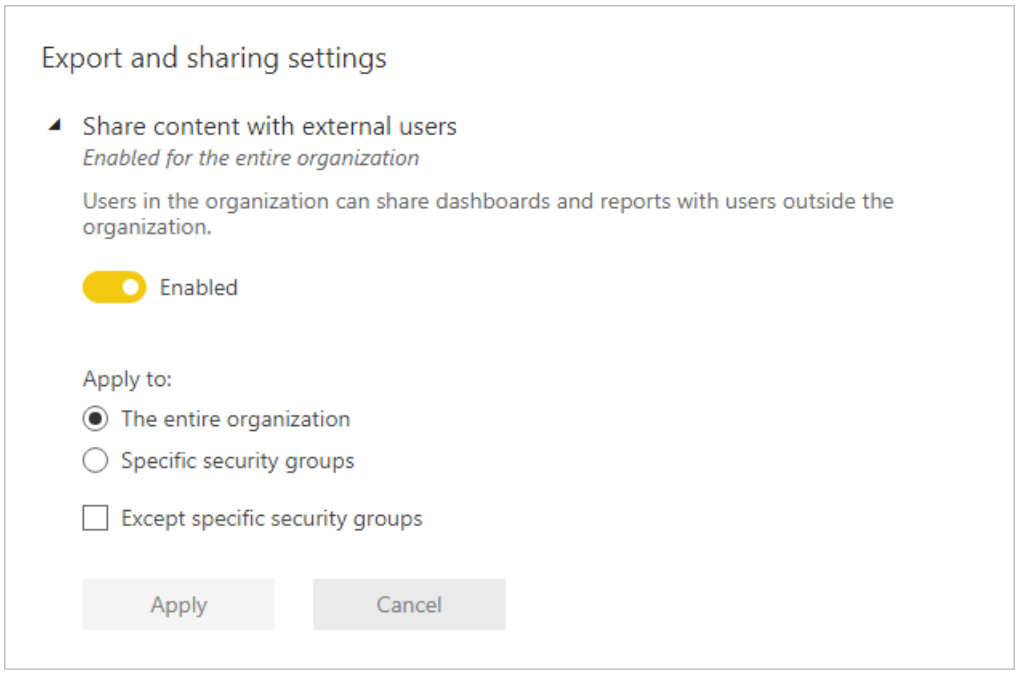


Share content with external users

Users can share reports and dashboards with people outside your organization.

The **Share content with external users** setting is enabled by default for the entire organization. It's found in the **Export and sharing settings** group.

We recommend you assign one or more security groups. These groups can be granted *or denied* permission to share content with external users.



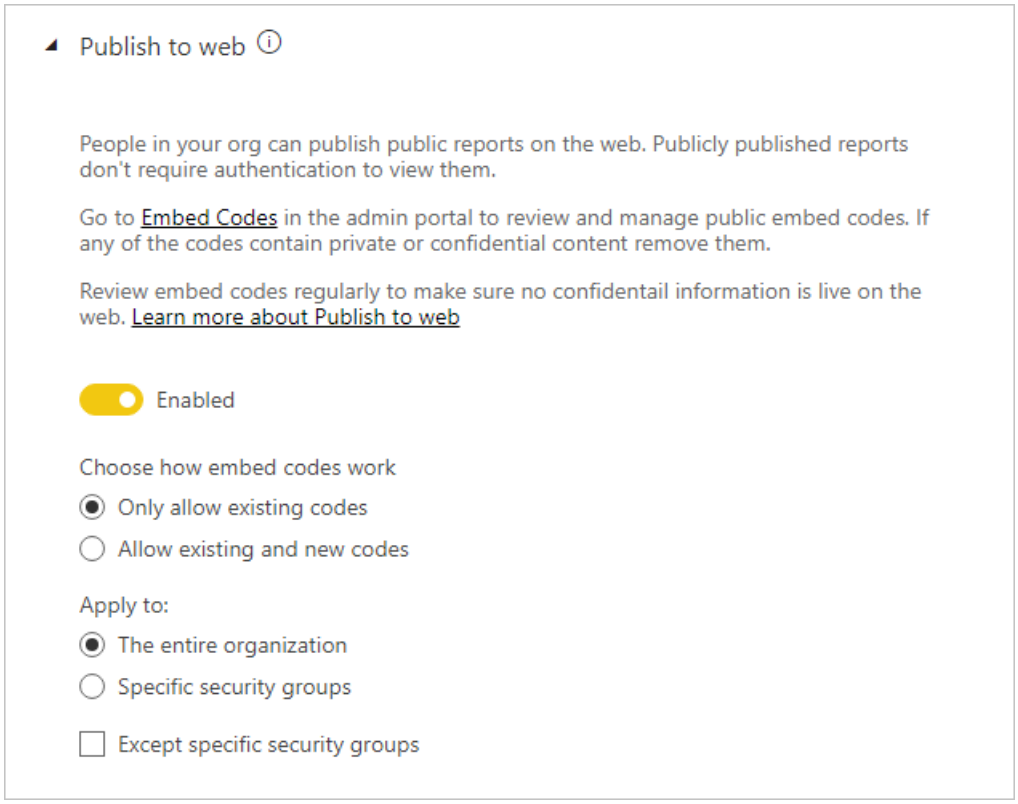
Publish to web

The publish to web feature allows publishing public reports on the web. If used inappropriately, there's risk that confidential information could be made available live on the web.

The **Publish to web** setting is enabled by default for the entire organization, but restricting the ability for non-admin users to create embed codes. It's found in the **Export and sharing settings** group.

If enabled, we recommend you assign one or more security groups. These groups can be granted *or denied* permission to publish reports.

Further, there's an option to choose how your embed codes work. By default, it's set to **Only allow existing codes**. It means users will be asked to contact a Power BI admin to create an embed code.



We also recommend you review publish to web embed codes regularly. Remove codes if they result in the publication of private or confidential information.

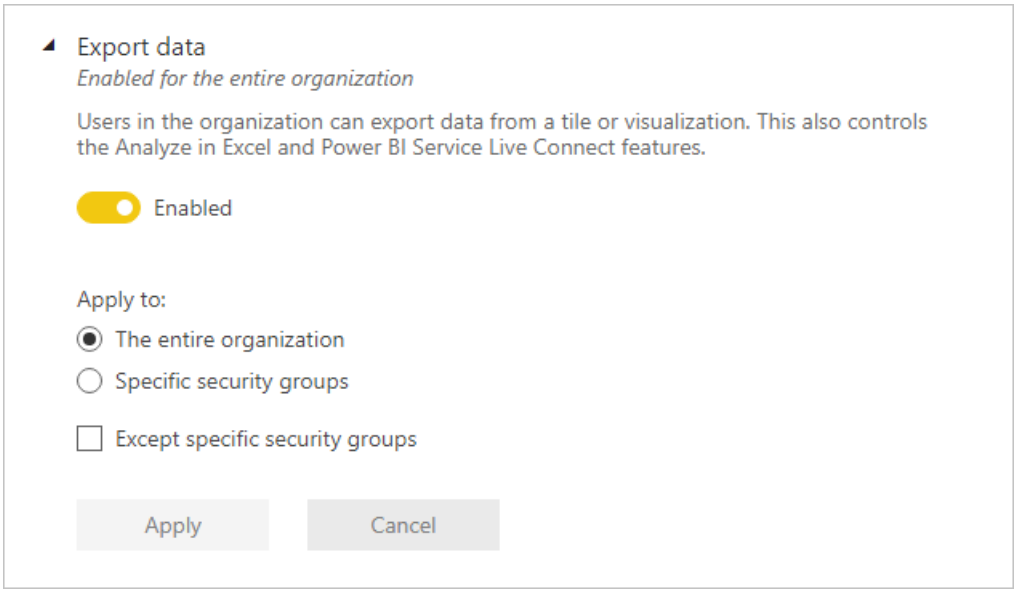
Export data

You can restrict users from exporting data from dashboard tiles or report visuals.

The **Export data** setting is enabled by default for the entire organization. It's found in the **Export and sharing settings** group.

We recommend you assign one or more security groups. These groups can be granted *or denied* permission to publish reports.

> **IMPORTANT**
> Disabling this setting also restricts the use of the Analyze in Excel and Power BI service live connection features.



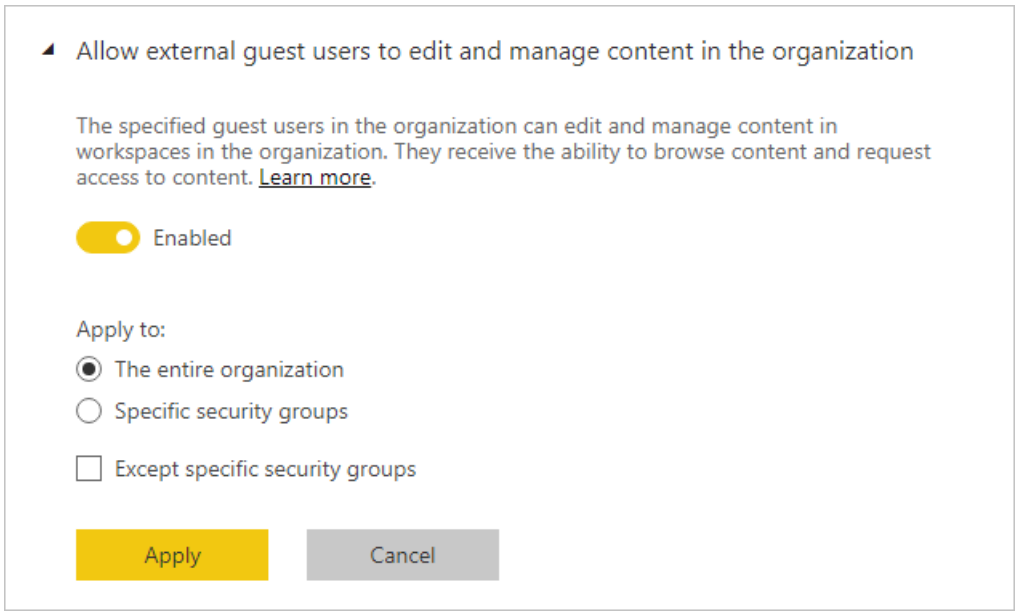
> **NOTE**
> If users allow users to export data, you can add a layer of protection by enforcing data protection. When configured, unauthorized users will be blocked from exporting content with sensitivity labels.

Allow external guest users to edit and manage content in the organization

It's possible that external guest users can edit and manage Power BI content. For more information, see Distribute Power BI content to external guest users with Azure AD B2B.

The **Allow external guest users to edit and manage content in the organization** setting is disabled by default for the entire organization. It's found in the **Export and sharing settings** group.

If you need to authorize external users to edit and manage content, we recommend you assign one or more security groups. These groups can be granted *or denied* permission to publish reports.

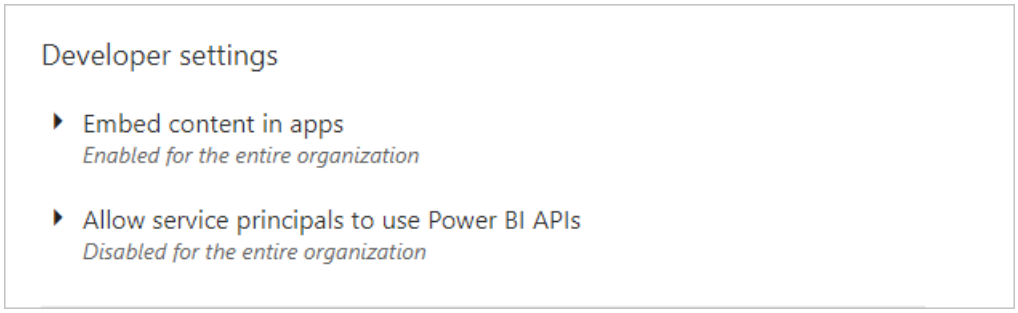


Developer settings

There are two tenant settings related to embedding Power BI content. They are:

- Embed content in apps (enabled by default)
- Allow service principals to user Power BI APIs (disabled by default)

If you have no intention of using the developer APIs to embed content, we recommend you disable them. Or, at least configure specific security groups that would be doing this work.



Next steps

For more information related to this article, check out the following resources:

- What is Power BI administration?
- Administering Power BI in the admin portal
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

On-premises data gateway sizing

5/13/2020 • 4 minutes to read • Edit Online

This article targets Power BI administrators who need to install and manage the on-premises data gateway.

The gateway is required whenever Power BI must access data that isn't accessible directly over the Internet. It can be installed on a server on-premises, or VM-hosted Infrastructure-as-a-Service (IaaS).

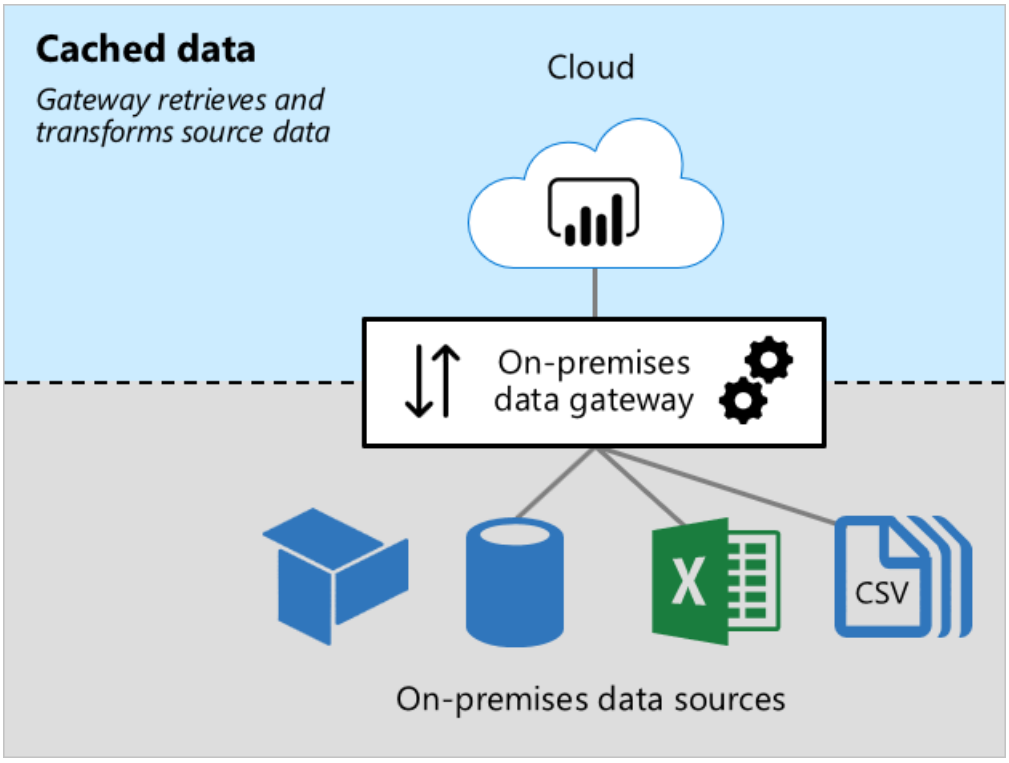
Gateway workloads

The on-premises data gateway supports two workloads. It's important you first understand these workloads before we discuss gateway sizing and recommendations.

Cached data workload

The *Cached data* workload retrieves and transforms source data for loading into Power BI datasets. It does so in three steps:

1. **Connection**: The gateway connects to source data
2. **Data retrieval and transformation**: Data is retrieved, and when necessary, transformed. Whenever possible, the Power Query mashup engine pushes transformation steps to the data source—it's known as *query folding*. When it's not possible, transformations must be done by the gateway. In this case, the gateway will consume more CPU and memory resources.
3. **Transfer**: Data is transferred to the Power BI service—a reliable and fast Internet connection is important, especially for large data volumes



Live Connection and DirectQuery workloads

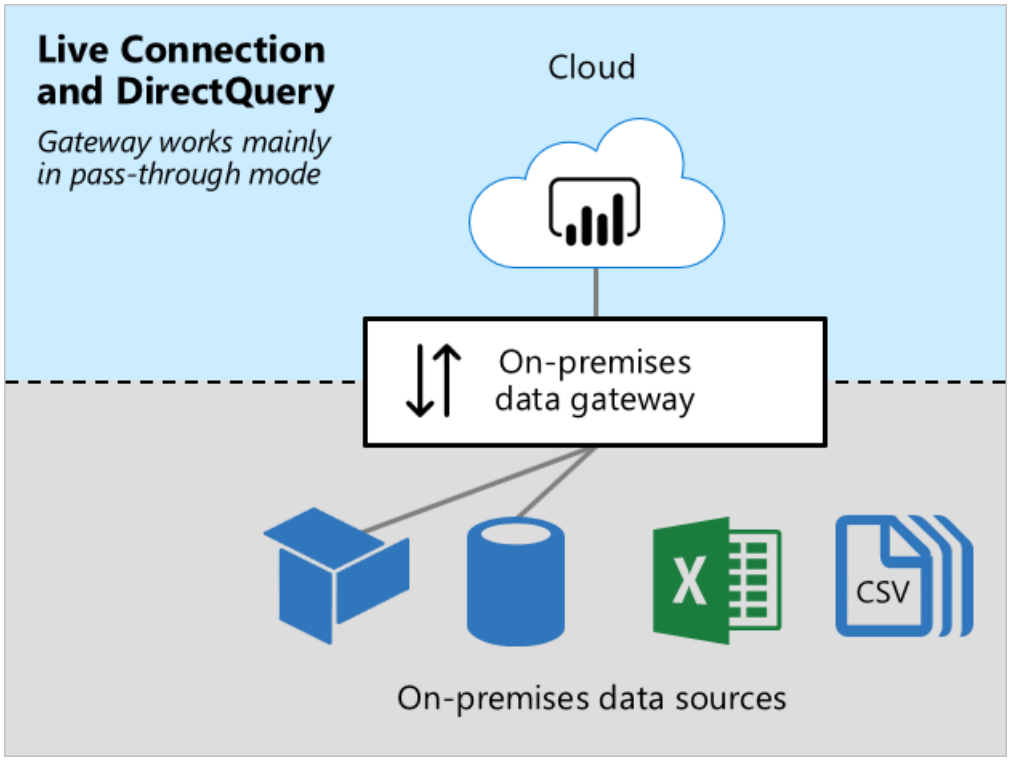
The *Live Connection and DirectQuery* workload works mostly in pass-through mode. The Power BI service sends queries, and the gateway responds with query results. Generally, query results are small in size.

- For more information about Live Connection, see Datasets in the Power BI service (Externally-hosted models).

- For more information about DirectQuery, see Dataset modes in the Power BI service (DirectQuery mode).

This workload requires CPU resources for routing queries and query results. Usually there's much less demand for CPU than is required by the Cache data workload—especially when it's required to transform data for caching.

Reliable, fast, and consistent connectivity is important to ensure report users have responsive experiences.



Sizing considerations

Determining the correct sizing for your gateway machine can depend on the following variables:

- For Cache data workloads:
  - The number of concurrent dataset refreshes
  - The types of data sources (relational database, analytic database, data feeds, or files)
  - The volume of data to be retrieved from data sources
  - Any transformations required to be done by the Power Query mashup engine
  - The volume of data to be transferred to the Power BI service
- For Live Connection and DirectQuery workloads:
  - The number of concurrent report users
  - The number of visuals on report pages (each visual sends at least one query)
  - The frequency of Power BI dashboard query cache updates
  - The number of real-time reports using the Automatic page refresh feature
  - Whether datasets enforce Row-level Security (RLS)

Generally, Live Connection and DirectQuery workloads require sufficient CPU, while Cache data workloads require more CPU and memory. Both workloads depend on good connectivity with the Power BI service, and the data sources.

NOTE

Power BI capacities impose limits on model refresh parallelism, and Live Connection and DirectQuery throughput. There's no point sizing your gateways to deliver more than what the Power BI service supports. Limits differ by Premium SKU (and equivalently sized A SKU). For more information, see What is Power BI Premium? (Capacity nodes).

Recommendations

Gateway sizing recommendations depend on many variables. In this section, we provide you with general recommendations that you can take into consideration.

Initial sizing

It can be difficult to accurately estimate the right size. We recommend that you start with a machine with at least 8 CPU cores, 8 GB of RAM, and multiple Gigabit network adapters. You can then measure a typical gateway workload by logging CPU and memory system counters. For more information, see Monitor and optimize on-premises data gateway performance.

Connectivity

Plan for the best possible connectivity between the Power BI service and your gateway, and your gateway and the data sources.

- Strive for reliability, fast speeds, and low, consistent latencies
- Eliminate—or reduce—machine hops between the gateway and your data sources
- Remove any network throttling imposed by your firewall proxy layer. For more information about Power BI endpoints, see Power BI URLs for whitelisting.
- Configure Azure ExpressRoute to establish private, managed connections to Power BI
- For data sources in Azure VMs, ensure the VMs are colocated with the Power BI service
- For Live Connection workloads to SQL Server Analysis Services (SSAS) involving dynamic RLS, ensure good connectivity between the gateway machine and the on-premises Active Directory

Clustering

For large-scale deployments, you can create a gateway of cluster installations. Clusters avoid single points of failure, and can load balance traffic across gateways. You can:

- Install one or more gateways in a cluster
- Isolate workloads to standalone gateways, or clusters of gateway servers

For more information, see Manage on-premises data gateway high-availability clusters and load balancing.

Dataset design and settings

Dataset design, and their settings, can impact on gateway workloads. To reduce gateway workload, you can consider the following actions.

For Import datasets:

- Configure less frequent data refresh
- Configure incremental refresh to minimize the amount of data to transfer
- Whenever possible, ensure query folding takes place
- Especially for large data volumes or a need for low-latency results, convert the design to a DirectQuery or Composite model

For DirectQuery datasets:

- Optimize data sources, model, and report designs—for more information, see DirectQuery model guidance in

Power BI Desktop

- Create aggregations to cache higher-level results to reduce the number of DirectQuery requests
- Restrict Automatic page refresh intervals, in report designs and capacity settings
- Especially when dynamic RLS is enforced, restrict dashboard cache update frequency
- Especially for smaller data volumes or for non-volatile data, convert the design to an Import or Composite model

For Live Connection datasets:

- Especially when dynamic RLS is enforced, restrict dashboard cache update frequency

Next steps

For more information related to this article, check out the following resources:

- Guidance for deploying a data gateway for Power BI
- Configure proxy settings for the on-premises data gateway
- Monitor and optimize on-premises data gateway performance
- Troubleshoot gateways - Power BI
- Troubleshoot the on-premises data gateway
- The importance of query folding
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Monitor report performance in Power BI

5/13/2020 • 3 minutes to read • Edit Online

Monitor report performance in Power BI Desktop by using the Power BI Premium Metrics app, learn where the bottlenecks are, and learn how you can improve report performance.

Monitoring performance is relevant in the following situations:

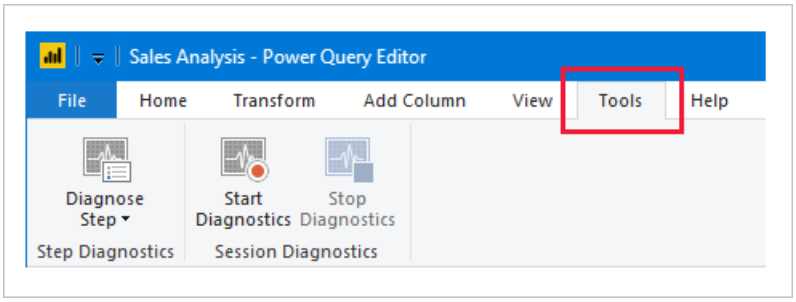
- Your Import data model refresh is slow.
- Your DirectQuery or Live Connection reports are slow.
- Your model calculations are slow.

Slow queries or report visuals should be a focal point of continued optimization.

Use Query Diagnostics

Use Query Diagnostics in Power BI Desktop to determine what Power Query is doing when previewing or applying queries. Further, use the *Diagnose Step* function to record detailed evaluation information for each query step. The results are made available in a Power Query, and you can apply transformations to better understand query execution.

> **NOTE**
> Query Diagnostics is currently a preview feature, and so you must enable it in *Options and Settings*. Once enabled, its commands are available in the Power Query Editor window, on the **Tools** ribbon tab.



Use Performance Analyzer

Use Performance Analyzer in Power BI Desktop to find out how each of your report elements—such as visuals and DAX formulas—are doing. It's especially useful to determine whether it's the query or visual rendering that's contributing to performance issues.

Use SQL Server Profiler

You can also use SQL Server Profiler to identify queries that are slow.

> **NOTE**
> SQL Server Profiler is available as part of SQL Server Management Studio.

Use SQL Server Profiler when your data source is either:

- SQL Server
- SQL Server Analysis Services
- Azure Analysis Services

Caution

Power BI Desktop supports connecting to a diagnostics port. The diagnostic port allows for other tools to make connections to perform traces for diagnostic purposes. Making any changes to the Power Desktop data model is not supported. Changes to the data model may lead to corruption and data loss.

To create a SQL Server Profiler trace, follow these instructions:

1. Open your Power BI Desktop report (so it will be easy to locate the port in the next step, close any other open reports).
2. To determine the port being used by Power BI Desktop, in PowerShell (with administrator privileges), or at the Command Prompt, enter the following command:

   ```
   netstat -b -n
   ```

   The output will be a list of applications and their open ports. Look for the port used by **msmdsrv.exe**, and record it for later use. It's your instance of Power BI Desktop.
3. To connect SQL Server Profiler to your Power BI Desktop report:
   a. Open SQL Server Profiler.
   b. In SQL Server Profiler, on the *File* menu, select *New Trace*.
   c. For **Server Type**, select *Analysis Services*.
   d. For **Server Name**, enter *localhost:[port recorded earlier]*.
   e. Click *Run*—now the SQL Server Profiler trace is live, and is actively profiling Power BI Desktop queries.
4. As Power BI Desktop queries are executed, you'll see their respective durations and CPU times. Depending on the data source type, you may see other events indicating how the query was executed. Using this information, you can determine which queries are the bottlenecks.

A benefit of using SQL Server Profiler is that it's possible to save a SQL Server (relational) database trace. The trace can become an input to the Database Engine Tuning Advisor. This way, you can receive recommendations on how to tune your data source.

Monitor Premium metrics

For Power BI Premium capacities, use the **Power BI Premium Metrics app** to monitor the health and capacity of your Power BI Premium subscription. For more information, see Power BI Premium Metrics app.

Next steps

For more information about this article, check out the following resources:

- Query Diagnostics
- Performance Analyzer
- Troubleshoot report performance in Power BI
- Power BI Premium Metrics app
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Troubleshoot report performance in Power BI

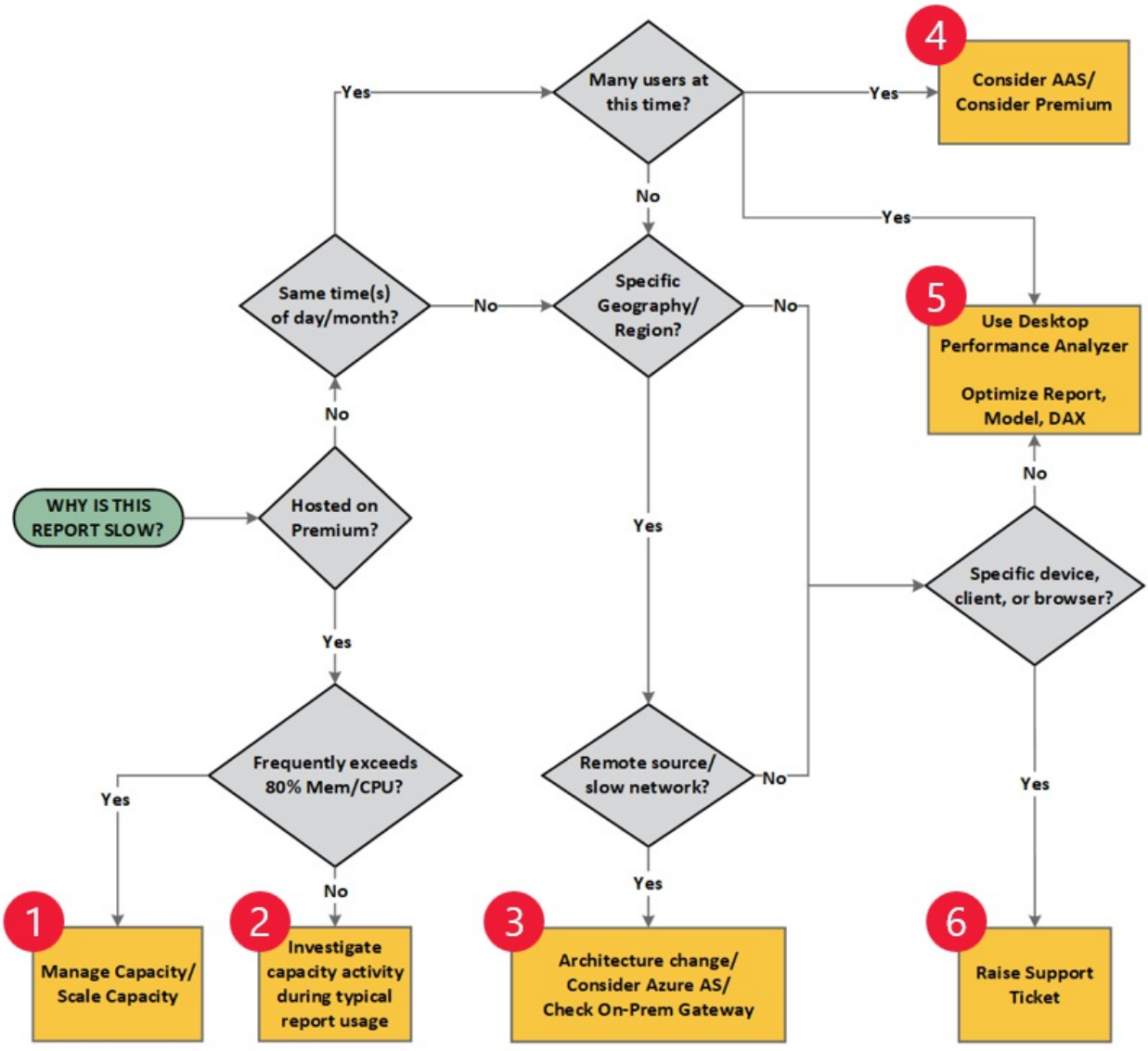
5/13/2020 • 2 minutes to read • Edit Online

This article provides guidance that enables developers and administrators to troubleshoot slow report performance. It applies to Power BI reports, and also Power BI paginated reports.

Slow reports can be identified by report users who experience reports that are slow to load, or slow to update when interacting with slicers or other features. When reports are hosted on a Premium capacity, slow reports can also be identified by monitoring the Power BI Premium Metrics app. This app helps you to monitor the health and capacity of your Power BI Premium subscription.

Follow flowchart steps

Use the following flowchart to help understand the cause of slow performance, and to determine what action to take.



There are six flowchart terminators, each describing action to take:

TERMINATOR	ACTION(S)
1	Manage capacity Scale capacity
2	Investigate capacity activity during typical report usage
3	Architecture change Consider Azure Analysis Services Check on-premises gateway
4	Consider Azure Analysis Services Consider Power BI Premium
5	Use Power BI Desktop Performance Analyzer Optimize report, model, or DAX
6	Raise support ticket

Take action

The first consideration is to understand if the slow report is hosted on a Premium capacity.

Premium capacity

When the report is hosted on a Premium capacity, use the **Power BI Premium Metrics app** to determine if the report-hosting capacity frequently exceeds capacity resources. It's the case for CPU when it frequently exceeds 80%. For memory, it's when the active memory metric exceeds 50. When there's pressure on resources, it may be time to manage or scale the capacity (flowchart terminator 1). When there are adequate resources, investigate capacity activity during typical report usage (flowchart terminator 2).

Shared capacity

When the report is hosted on shared capacity, it's not possible to monitor capacity health. You'll need to take a different investigative approach.

First, determine if slow performance occurs at specific times of the day or month. If it does—and many users are opening the report at these times—consider two options:

- Increase query throughput by migrating the dataset to Azure Analysis Services, or a Premium capacity (flowchart terminator 4).
- Use Power BI Desktop Performance Analyzer to find out how each of your report elements—such as visuals and DAX formulas—are doing. It's especially useful to determine whether it's the query or visual rendering that's contributing to performance issues (flowchart terminator 5).

If you determine there's no time pattern, next consider if slow performance is isolated to a specific geography or region. If it is, it's likely that the data source is remote and there's slow network communication. In this case, consider:

- Changing architecture by using Azure Analysis Services (flowchart terminator 3).
- Optimizing on-premises data gateway performance (flowchart terminator 3).

Finally, if you determine there's no time pattern *and* slow performance occurs in all regions, investigate whether slow performance occurs on specific devices, clients, or web browsers. If it doesn't, use Power BI Desktop Performance Analyzer, as described earlier, to optimize the report or model (flowchart terminator 5).

When you determine specific devices, clients, or web browsers contribute to slow performance, we recommend creating a support ticket through the Power BI support page (flowchart terminator 6).

Next steps

For more information about this article, check out the following resources:

- Power BI guidance
- Monitoring report performance
- Performance Analyzer
- Whitepaper: Planning a Power BI Enterprise Deployment
- Questions? Try asking the Power BI Community
- Suggestions? Contribute ideas to improve Power BI

Deployment pipelines best practices (preview)

5/13/2020 • 9 minutes to read • Edit Online

This article provides guidance for BI creators who are managing their content throughout its lifecycle. It focuses on leveraging deployment pipelines as a BI content lifecycle management tool.

The article is divided into four sections:

- **Content preparation** - Prepare your content for lifecycle management.
- **Development** - Learn about the best ways of creating content in the deployment pipelines development stage.
- **Test** - Understand how to use the deployment pipelines test stage, to test your environment.
- **Production** - Utilize the deployment pipelines production stage when making your content available for consumption.

Content preparation

Prepare your content for on-going management throughout its lifecycle. Make sure you review the information in this section, before you do any of the following:

- Release your content to production
- Start using a deployment pipeline for a specific workspace
- Publish your work

Treat each workspace as a complete package of analytics

Ideally, a workspace should contain a complete view of one aspect (such as department, business unit, project, or vertical) in your organization. This makes it easier to manage permissions for different users, and allows content releases for the entire workspace to be controlled according to a planned schedule.

If you're using centralized datasets that are used across the organization, we recommend that you create two types of workspaces:

- **Modeling and data workspaces** - These workspaces will contain all the centralized datasets
- **Reporting workspaces** - These workspaces will contain all dependent reports and dashboards

Plan your permission model

A deployment pipeline is a Power BI object, with its own permissions. In addition, the pipeline contains workspaces, that have their own permissions.

To implement a secure and easy workflow, plan who gets access to each part of the pipeline. Some of the considerations to take into account are:

- Who should have access to the pipeline?
- Which operations should users with pipeline access be able to perform in each stage?
- Who's reviewing content in the test stage?
- Should the test stage reviewers have access to the pipeline?
- Who will oversee deployment to the production stage?

- Which workspace are you assigning?
- Which stage are you assigning your workspace to?
- Do you need to make changes to the permissions of the workspace you're assigning?

Connect different stages to different databases

A production database should always be stable and available. It's better not to overload it with queries generated by BI creators for their development or test datasets. Build separate databases for development and testing. This helps protect production data, and doesn't overload the development database with the entire volume of production data, which can slow down things.

> **NOTE**
> If your organization is using shared centralized datasets, you can skip this recommendation.

Use parameters in your model

As you can't edit datasets data sources in Power BI service, we recommend using parameters to store connection details such as instance names and database names, instead of using a static connection string. This allows you to manage the connections through the Power BI service web portal, or using APIs, at a later stage.

In deployment pipelines, you can configure parameter rules to set specific values for the development, test, and production stages.

If you don't use parameters for your connection string, you can define data source rules to specify a connection string for a given dataset. However, in deployment pipelines, this isn't supported for all data sources. To verify that you can configure rules for your data source, see dataset rule limitations.

Parameters have additional uses, such as making changes to queries, filters, and the text displayed in the report.

Development

This section provides guidance for working with the deployment pipelines development stage.

Use Power BI Desktop to edit your reports and datasets

Consider Power BI Desktop as your local development environment. Power BI Desktop allows you to try, explore, and review updates to your reports and datasets. Once the work is done, you can upload your new version to the development stage. Due to the following reasons, it's recommended to edit .pbix files in the Desktop (and not in Power BI service):

- It is easier to collaborate with fellow creators on the same .pbix file, if all changes are being done on the same tool.
- Making online changes, downloading the .pbix file, and then uploading it again, creates reports and datasets duplication.
- You can use version control to keep your .pbix files up to date.

Version control for .pbix files

If you want to manage the version history of your reports and datasets, use Power BI's autosync with OneDrive. This will keep your files updated with the latest version. It will also enable you to retrieve older versions if needed.

> **NOTE**
>
> Use auto-sync with OneDrive (or any other repository) only with the .pbix files in the deployment pipelines development stage. Do not sync .pbix files into the deployment pipelines test and production stages. This will cause problems with deploying content across the pipeline.

Separate modeling development from report and dashboard development

For enterprise scale deployments, it's recommended to separate dataset development, and the development of reports and dashboards. To promote changes to only a report or a dataset, use the deployment pipelines selective deploy option.

This approach should start from Power BI Desktop, by creating a separate .pbix file for datasets and reports. For example, you can create a dataset .pbix file and uploaded it to the development stage. Later, your report authors can create a new .pbix only for the report, and connect it to the published dataset using a live connection. This technic allows different creators to separately work on modeling and visualizations, and deploy them to production independently.

With shared datasets, you can also use this method across workspaces.

Manage your models using XMLA read/write capabilities

Separating modeling development from report and dashboard development, allows you to use advanced capabilities such as source control, merging diff changes, and automated processes. These changes should be done in the development stage, so that finalized content can be deployed to the test and production stages. This allows changes to go through a unified process with other dependent items, before they're deployed to the production stage.

You can separate modeling development from visualizations, by managing a shared dataset in an external workspace, using XMLA r/w capabilities. The shared dataset can connect to multiple reports in various workspaces that are managed in multiple pipelines.

Test

This section provides guidance for working with the deployment pipelines test stage.

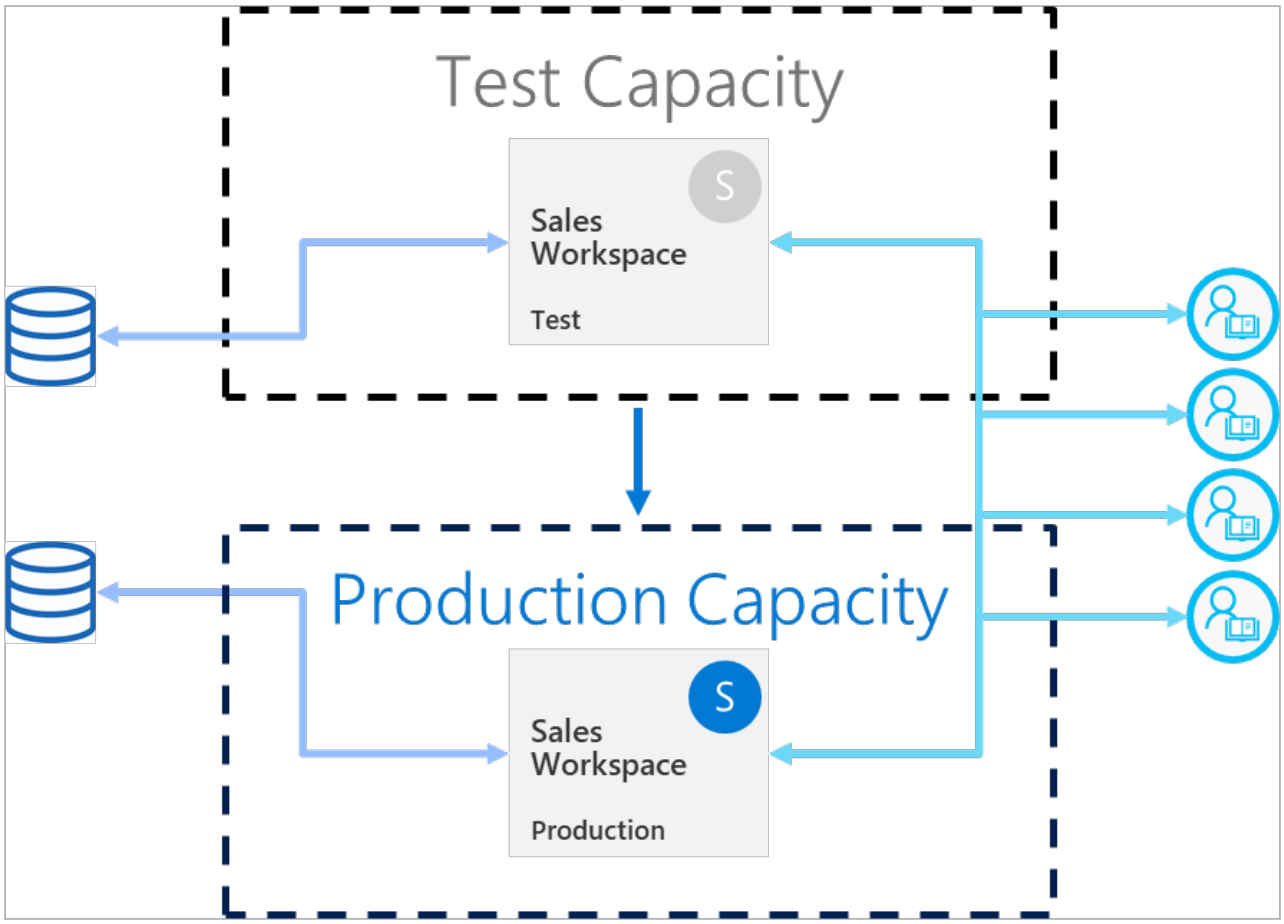
Simulate your production environment

Other than verifying that new reports or dashboards look alright, it's also important to see how they perform from an end user's perspective. The deployment pipelines test stage, allows you to simulate a real production environment for testing purposes.

Make sure that these three factors are addressed in your test environment:

- Data volume
- Usage volume
- A similar capacity as in production

When testing, you can use the same capacity as the production stage. However, this can make production unstable during load testing. To avoid unstable production, use another capacity similar in resources to the production capacity, for testing. To avoid extra costs, you can use Azure A capacities to pay only for the testing time.



Use dataset rules with a real-life data source

If you're using the test stage to simulate real life data usage, it's recommended to separate the development and test data sources. The development database should be relatively small, and the test database should be as similar as possible to the production database. Use data source rules to switch data sources in the test stage.

Controlling the amount of data you import from your data source, is useful if you're using a production data source in the test stage. To do this, add a parameter to your data source query in Power BI Desktop. Use parameter rules to control the amount of imported data, or edit the parameter's value. You can also use this approach if you don't want to overload your capacity.

Measure performance

When you simulate a production stage, check the report load and the interactions, and find out if the changes you made impact them.

You also need to monitor the load on the capacity, so that you can catch extreme loads before they reach production.

> **NOTE**
> It's recommended to monitor capacity loads again, after deploying updates to the production stage.

Check related items

Related times can be affected by changes to datasets or reports. During testing, verify that your changes don't impact or break the performance of existing items, which can be dependent on the updated ones.

You can easily find the related items using the workspace lineage view.

Test your app

If you are distributing content to your end users through an app, review the app's new version, before it's in production. As each deployment pipeline stage has its own workspace, you can easily publish and update apps for

development and test stages. This will allow you to test the app from an end user's point of view.

> **IMPORTANT**
> The deployment process does not include updating the app content or settings. To apply changes to content or settings, you need to manually update the app in the required pipeline stage.

Production

This section provides guidance to the deployment pipelines production stage.

Manage who can deploy to production

As deploying to production should be handled carefully, it's good practice to let only specific people manage this sensitive operation. However, you probably want all BI creators for a specific workspace to have access to the pipeline. This can be managed using production workspace permissions.

To deploy content between stages, users need to have either member or admin permissions for both stages. Make sure that only the people you want deploying to production, will have production workspace permissions. Other users can have production workspace contributor or viewer roles. They will be able to see content from within the pipeline but won't be able to deploy.

In addition, you should limit access to the pipeline by only enabling pipeline permissions to users that are part of the content creation process.

Set rules to ensure production stage availability

Dataset rules are a powerful way to ensure the data in production is always connected and available to users. Once dataset rules are applied, deployments can run while you have the assurance that end users will see the relevant info without disturbance.

Make sure that you set production dataset rules for data sources and parameters defined in the dataset.

Update the production app

Deployment in a pipeline updates the workspace content, but it doesn't update the associated app automatically. If you're using an app for content distribution, don't forget to update the App after deploying to production, so that end users will immediately be able to use the latest version.

Quick fixes to content

In case there are bugs in production that require a quick fix, don't be tempted to either upload a new .pbix version directly to the production stage, or make an online change in Power BI service. Deploying backwards to test and development stages isn't possible when there's already content in those stages. Furthermore, deploying a fix without testing it first is bad practice. Therefore, the correct way to treat this problem, is to implement the fix in the development stage, and push it to the rest of the deployment pipeline stages. This allows checking that the fix works, before deploying it to production. Deploying across the pipeline and takes only a few minutes.

Next steps

Introduction to deployment pipelines

Get started with deployment pipelines

Understand the deployment pipelines process

Deployment pipelines troubleshooting

Whitepapers for Power BI

5/28/2020 • 2 minutes to read • Edit Online

Whitepapers allow you to explore Power BI topics at a deeper level. Here you can find a list of available whitepapers for Power BI.

WHITEPAPER	DESCRIPTION	DATE
Planning a Power BI Enterprise Deployment	This updated technical whitepaper outlines considerations and best practices for a well-performing and secure organizational Power BI deployment.	May 2020
Power BI and Dataflows	This whitepaper describes dataflows in technical detail, and describes the capabilities and initiatives behind dataflow features and functionality.	November 2018
Microsoft Power BI Premium	Describes Power BI Premium, both as it exists when launched and also how it will evolve.	October 2017
Power BI Premium Planning and Deployment	The content of this whitepaper has been incorporated into general guidance. See the link for guidance and best practices for planning and deploying Premium capacity for well-defined workloads.	March 2019
Capacity planning guidance for Power BI Report Server	This paper aims to offer guidance on capacity planning for Power BI Report Server by sharing results of numerous load test executions of various workloads against a report server.	March 2018
Security	Provides a detailed explanation of security within Power BI.	March 2019
Distribute Power BI content to external guest users using Azure Active Directory B2B	This paper outlines how to distribute content to users outside the organization using the integration of Azure Active Directory Business-to-business (AAD B2B).	March 2019
Advanced Analytics with Power BI	Describes the advanced analytics capabilities of Power BI, including predictive analytics, custom visualizations, R integration, and data analysis expressions.	February 2017
Bidirectional filtering	Explains bidirectional cross-filtering in Power BI Desktop (the whitepaper also covers SQL Server Analysis Services 2016, both have the same behavior).	July 2018

WHITEPAPER	DESCRIPTION	DATE
DirectQuery in SQL Server 2016 Analysis Services	For SQL Server 2016, DirectQuery was redesigned for dramatically improved speed and performance, however, it is also now more complex to understand and implement.	January 2017
Power BI and SAP BW	This document describes how SAP customers can benefit from connecting Power BI to their existing SAP Business Warehouse (BW) systems. Updated in November 2019.	November 2019
Securing the Tabular BI Semantic Model	This paper introduces the security model for tabular BI semantic and Power BI. You will learn how to create roles, implement dynamic security, configure impersonation settings, manage roles, and choose a method for connecting to models that works in your network security context.	April 2016
Power BI and GDPR	This link takes you to the list of whitepapers on the Service Trust Portal, including the Microsoft Power BI GDPR whitepaper.	April 2018

NOTE

If you're interested in viewing or deleting personal data, please review Microsoft's guidance in the Windows Data Subject Requests for the GDPR site. If you're looking for general information about GDPR, see the GDPR section of the Service Trust portal.

More questions? Try asking the Power BI Community

Power BI security whitepaper

5/21/2020 • 37 minutes to read • Edit Online

Summary: Power BI is an online software service (*SaaS*, or Software as a Service) offering from Microsoft that lets you easily and quickly create self-service Business Intelligence dashboards, reports, datasets, and visualizations. With Power BI, you can connect to many different data sources, combine and shape data from those connections, then create reports and dashboards that can be shared with others.

Writer: David Iseminger

Technical Reviewers: Pedram Rezaei, Cristian Petculescu, Siva Harinath, Tod Manning, Haydn Richardson, Adam Wilson, Ben Childs, Robert Bruckner, Sergei Gundorov, Kasper de Jonge

Applies to: Power BI SaaS, Power BI Desktop, Power BI Embedded, Power BI Premium

> **NOTE**
> You can save or print this whitepaper by selecting **Print** from your browser, then selecting **Save as PDF**.

Introduction

Power BI is an online software service (*SaaS*, or Software as a Service) offering from Microsoft that lets you easily and quickly create self-service Business Intelligence dashboards, reports, datasets, and visualizations. With Power BI, you can connect to many different data sources, combine and shape data from those connections, then create reports and dashboards that can be shared with others.

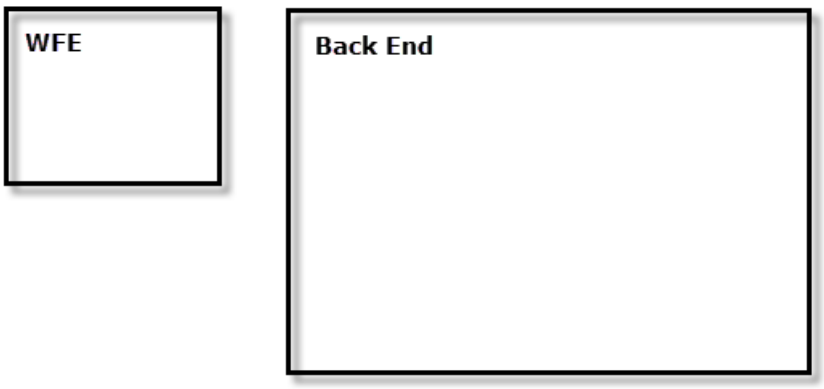
The Power BI service is governed by the Microsoft Online Services Terms, and the Microsoft Enterprise Privacy Statement. For the location of data processing, refer to the Location of Data Processing terms in the Microsoft Online Services Terms. For compliance information, the Microsoft Trust Center is the primary resource for Power BI. The Power BI team is working hard to bring its customers the latest innovations and productivity. Power BI is currently in Tier D of the Microsoft 365 Compliance Framework. Learn more about compliance in the Microsoft Trust Center.

This article describes Power BI security by providing an explanation of the Power BI architecture, then explaining how users authenticate to Power BI and data connections are established, and then describing how Power BI stores and moves data through the service. The last section is dedicated to security-related questions, with answers provided for each.

Power BI Architecture

The **Power BI** service is built on **Azure**, which is Microsoft's cloud computing platform. Power BI is currently deployed in many datacenters around the world – there are many active deployments made available to customers in the regions served by those datacenters, and an equal number of passive deployments that serve as backups for each active deployment.

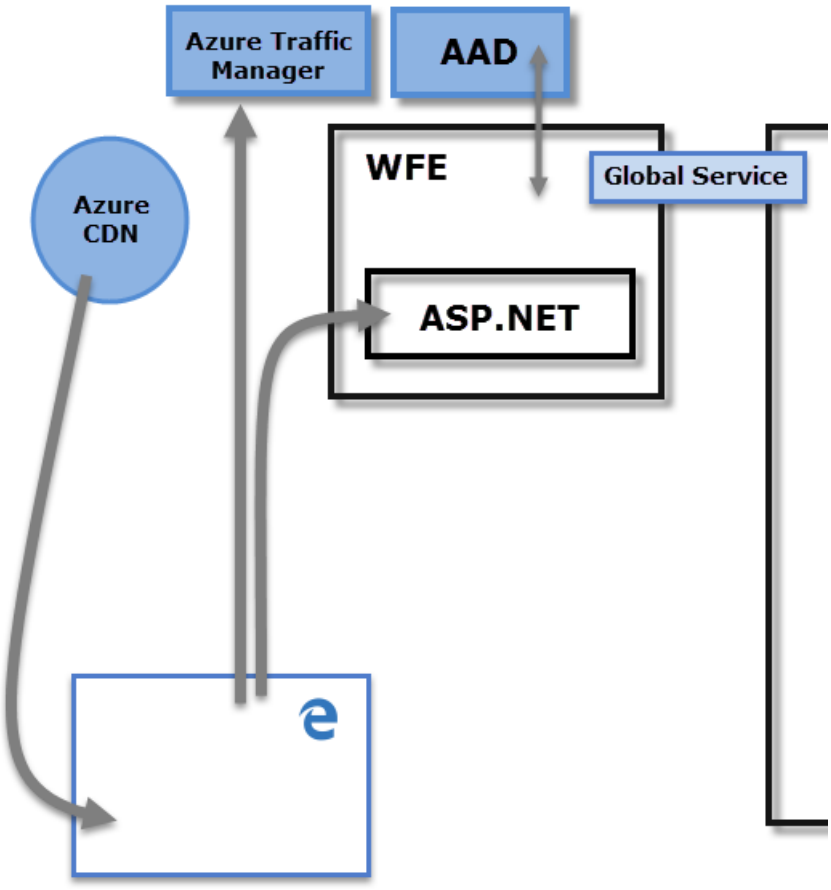
Each Power BI deployment consists of two clusters – a Web Front End (**WFE**) cluster, and a **Back-End** cluster. These two clusters are shown in the following image, and provide the backdrop for the rest of this article.



Power BI uses Azure Active Directory (**AAD**) for account authentication and management. Power BI also uses the **Azure Traffic Manager (ATM)** to direct user traffic to the nearest datacenter, determined by the DNS record of the client attempting to connect, for the authentication process and to download static content and files. Power BI uses the geographically closest WFE to efficiently distribute the necessary static content and files to users, with the exception of Power BI visuals which are delivered using the **Azure Content Delivery Network (CDN)**.

The WFE Cluster

The **WFE** cluster manages the initial connection and authentication process for Power BI, using AAD to authenticate clients and provide tokens for subsequent client connections to the Power BI service.



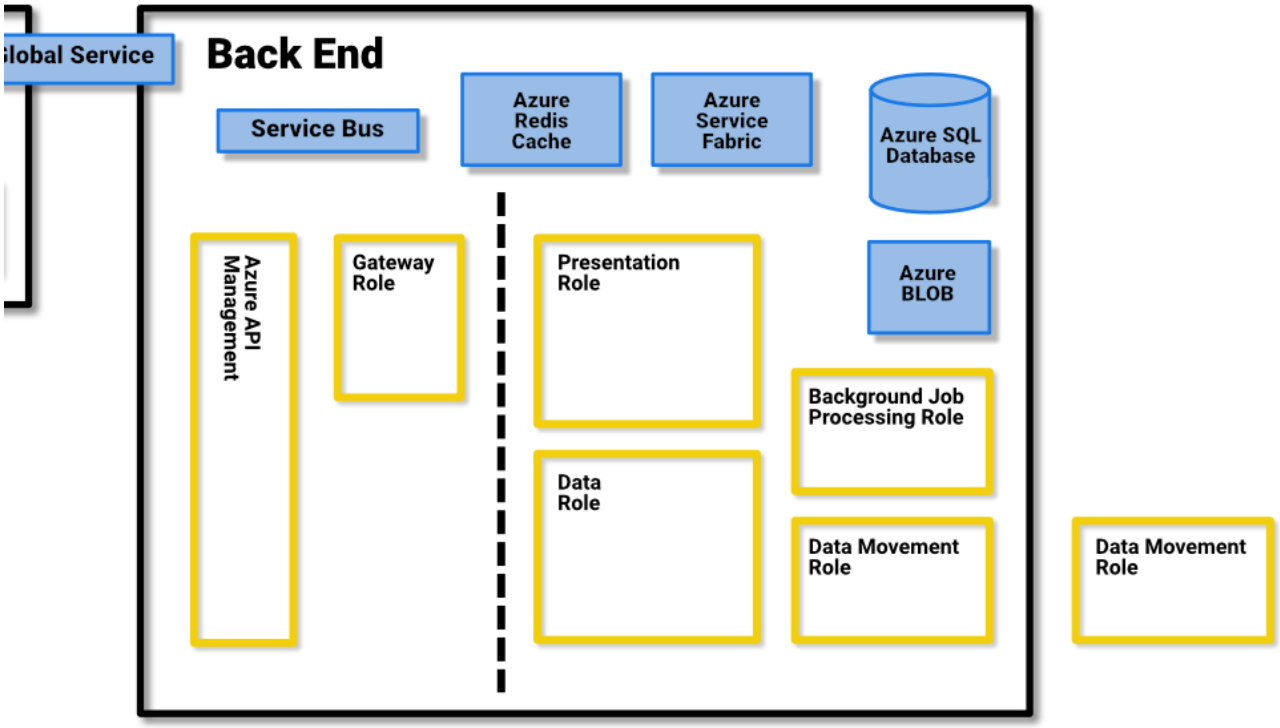
When users attempt to connect to the Power BI service, the client's DNS service may communicate with the **Azure Traffic Manager** to find the nearest datacenter with a Power BI deployment. For more information about this process, see Performance traffic routing method for Azure Traffic Manager.

The WFE cluster nearest to the user manages the login and authentication sequence (described later in this article),

and provides an AAD token to the user once authentication is successful. The ASP.NET component within the WFE cluster parses the request to determine which organization the user belongs to, and then consults the Power BI **Global Service**. The Global Service is a single Azure Table shared among all worldwide WFE and Back-End clusters that maps users and customer organizations to the datacenter that houses their Power BI tenant. The WFE specifies to the browser which Back-End cluster houses the organization's tenant. Once a user is authenticated, subsequent client interactions occur with the Back-End cluster directly, without the WFE being an intermediator for those requests.

The Power BI Back-End Cluster

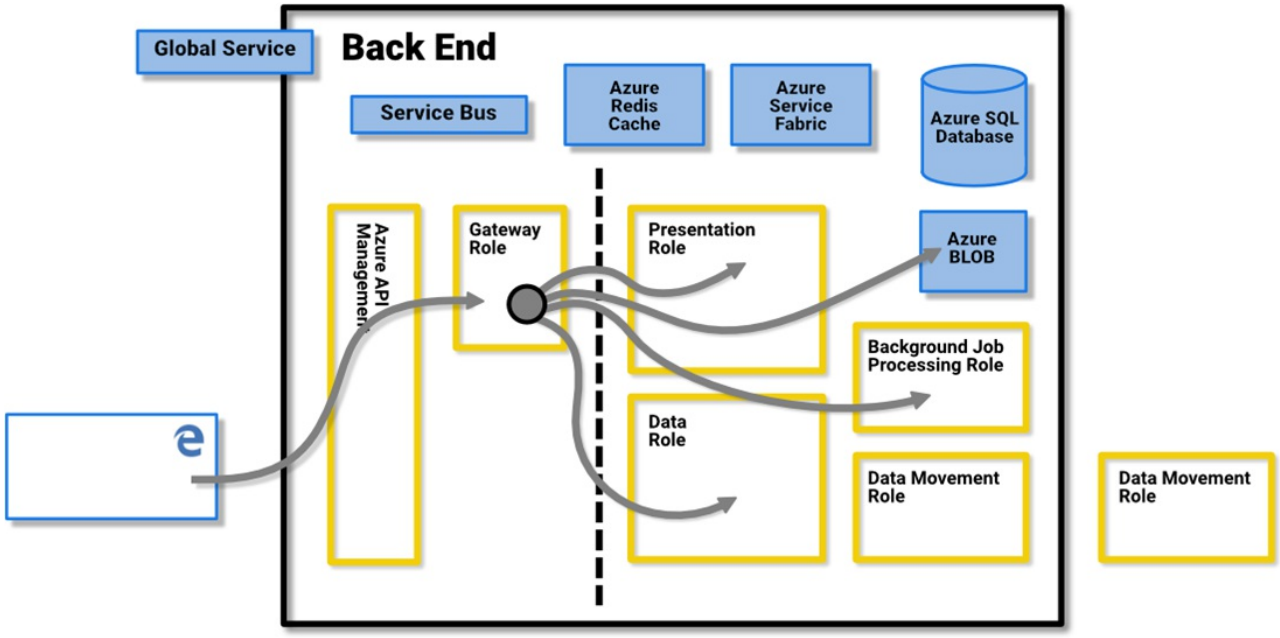
The **Back-End** cluster is how authenticated clients interact with the Power BI service. The **Back-End** cluster manages visualizations, user dashboards, datasets, reports, data storage, data connections, data refresh, and other aspects of interacting with the Power BI service.



The **Gateway Role** acts as a gateway between user requests and the Power BI service. Users do not interact directly with any roles other than the Gateway Role.

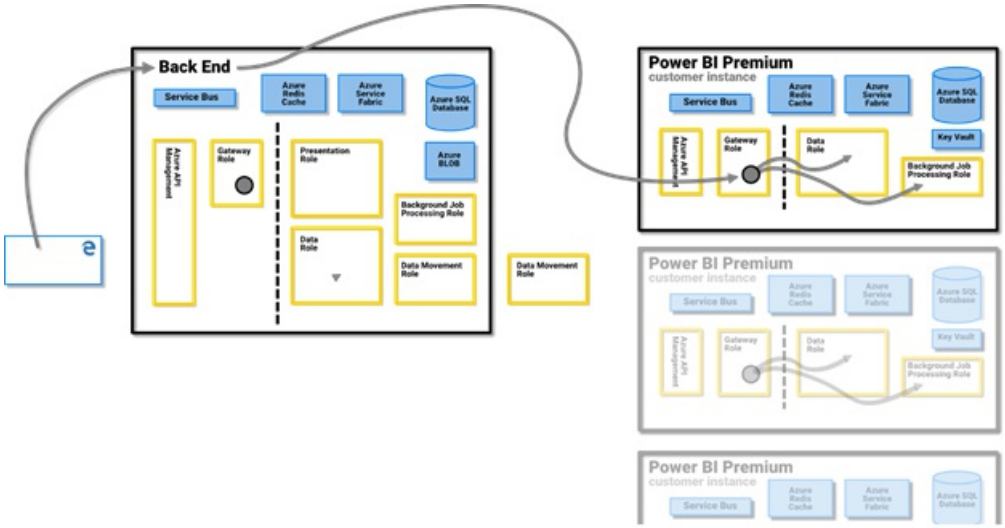
Important: It is imperative to note that *only* Azure API Management (**APIM**) and Gateway (**GW**) roles are accessible through the public Internet. They provide authentication, authorization, DDoS protection, Throttling, Load Balancing, Routing, and other capabilities.

The dotted line in the **Back-End** cluster image, above, clarifies the boundary between the only two roles that are accessible by users (left of the dotted line), and roles that are only accessible by the system. When an authenticated user connects to the Power BI Service, the connection and any request by the client is accepted and managed by the **Gateway Role** and **Azure API Management**, which then interacts on the user's behalf with the rest of the Power BI Service. For example, when a client attempts to view a dashboard, the **Gateway Role** accepts that request then separately sends a request to the **Presentation Role** to retrieve the data needed by the browser to render the dashboard.



Power BI Premium

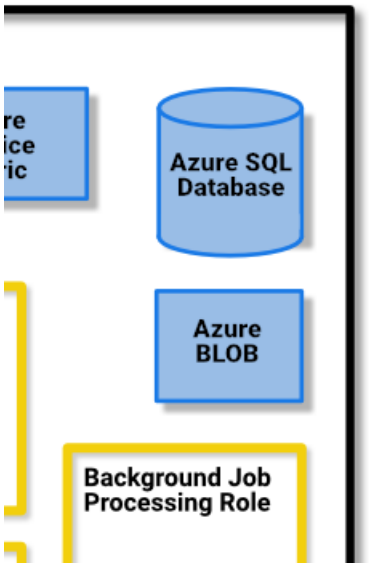
Power BI Premium offers a dedicated, provisioned, and partitioned service workspace for subscribers that need dedicated resources for their Power BI activities. When a customer signs up for a Power BI Premium subscription, the Premium capacity is created through the **Azure Resource Manager**. The rollout of that subscription assigns a set of virtual machines commensurate with the subscription level, in the datacenter where their Power BI tenant is hosted (with the exception of multi-geo environments, as described later in this document), initiated as an **Azure Service Fabric** deployment.



Once created, all communication with the Premium cluster is routed through the Power BI Back-End cluster, where a connection to the client's dedicated **Power BI Premium** subscription virtual machines is established.

Data Storage Architecture

Power BI uses two primary repositories for storing and managing data: data that is uploaded from users is typically sent to **Azure Blob** storage, and all metadata as well as artifacts for the system itself are stored behind a firewall in **Azure SQL Database**.



For example, when a user imports an Excel workbook into the Power BI service, an in-memory Analysis Services tabular database is created, and the data is stored in-memory for up to one hour (or until memory pressure occurs on the system). The data is also sent to **Azure Blob** storage.

Metadata about a user's Power BI subscription, such as dashboards, reports, recent data sources, workspaces, organizational information, tenant information, and other metadata about the system is stored and updated in **Azure SQL Database**. All information stored in Azure SQL Database is fully encrypted using Azure SQL's Transparent Data Encryption (TDE) technology. All data that is stored in Azure Blob storage is also encrypted. More information about the process of loading, storing, and moving data is described in the **Data Storage and Movement** section.

Tenant Creation

A tenant is a dedicated instance of the Azure AD service that an organization receives and owns when it signs up for a Microsoft cloud service such as Azure, Microsoft Intune, Power BI, or Microsoft 365. Each Azure AD tenant is distinct and separate from other Azure AD tenants.

A tenant houses the users in a company and the information about them - their passwords, user profile data, permissions, and so on. It also contains groups, applications, and other information pertaining to an organization and its security. For more information, see What is an Azure AD tenant.

A Power BI tenant is created in the datacenter deemed closest to the country (or region) and state information provided for the tenant in Azure Active Directory, which was provided when the Microsoft 365 or Power BI service was initially provisioned. The Power BI tenant does not move from that datacenter location today.

Multiple Geographies (Multi-geo)

Some organizations require a Power BI presence in multiple geographies, or regions, based on business needs. For example, a business may have its Power BI tenant in the United States but may also do business in other geographical areas, such as Australia, and need certain Power BI data to remain at rest in that remote region to comply with local regulations. Beginning in the second half of 2018, organizations with their home tenant in one geography can also provision and access Power BI resources located in another geography. This feature is referred to as **multi-geo** for convenience and reference throughout this document.

The most current and primary article for multi-geo information is the configure Multi-Geo support for Power BI Premium article.

There are multiple technical details that should be evaluated in the context of local laws and regulations when operating in different geographies. These details include the following:

- A remote query execution layer is hosted in the remote capacity region, to ensure that the data model, caches,

and most data processing remain in the remote capacity region. There are some exceptions, as detailed on the multi-geo for Power BI Premium article.

- A cached query text and corresponding result stored in a remote region will stay in that region at rest, however other data in transit may go back and forth between multiple geographies.
- PBIX or XLSX files that are published (uploaded) to a multi-geo capacity of the Power BI service may result in a copy being temporarily stored in Azure Blob storage in Power BI's tenant region. In such circumstances, the data is encrypted using Azure Storage Service Encryption (SSE), and the copy is scheduled for garbage collection as soon as the file content processing and transfer to the remote region is completed.
- When moving data across regions in a multi-geo environment, the instance of the data in the source region will be deleted within 7-30 days.

Datacenters and Locales

Power BI is offered in certain regions, based on where Power BI clusters are deployed in regional datacenters. Microsoft plans to expand its Power BI infrastructure into additional datacenters.

The following links provide additional information about Azure datacenters.

- Azure Regions – information about Azure's global presence and locations
- Azure Services, by region – a complete listing of Azure services (both infrastructure services and platform services) available from Microsoft in each region.

Currently, the Power BI service is available in specific regions, serviced by datacenters as described in the Microsoft Trust Center. The following link shows a map of Power BI datacenters, you can hover over a region to see the datacenters located there:

- Power BI Datacenters

Microsoft also provides datacenters for sovereignties. For more information about Power BI service availability for national clouds, see Power BI national clouds.

For more information on where your data is stored and how it is used, refer to the Microsoft Trust Center. Commitments about the location of customer data at rest are specified in the **Data Processing Terms** of the Microsoft Online Services Terms.

User Authentication

User authentication to the Power BI service consists of a series of requests, responses, and redirects between the user's browser and the Power BI service or the Azure services used by Power BI. That sequence describes the process of user authentication in Power BI. For more information about options for an organization's user authentication models (sign-in models), see Choosing a sign-in model for Microsoft 365.

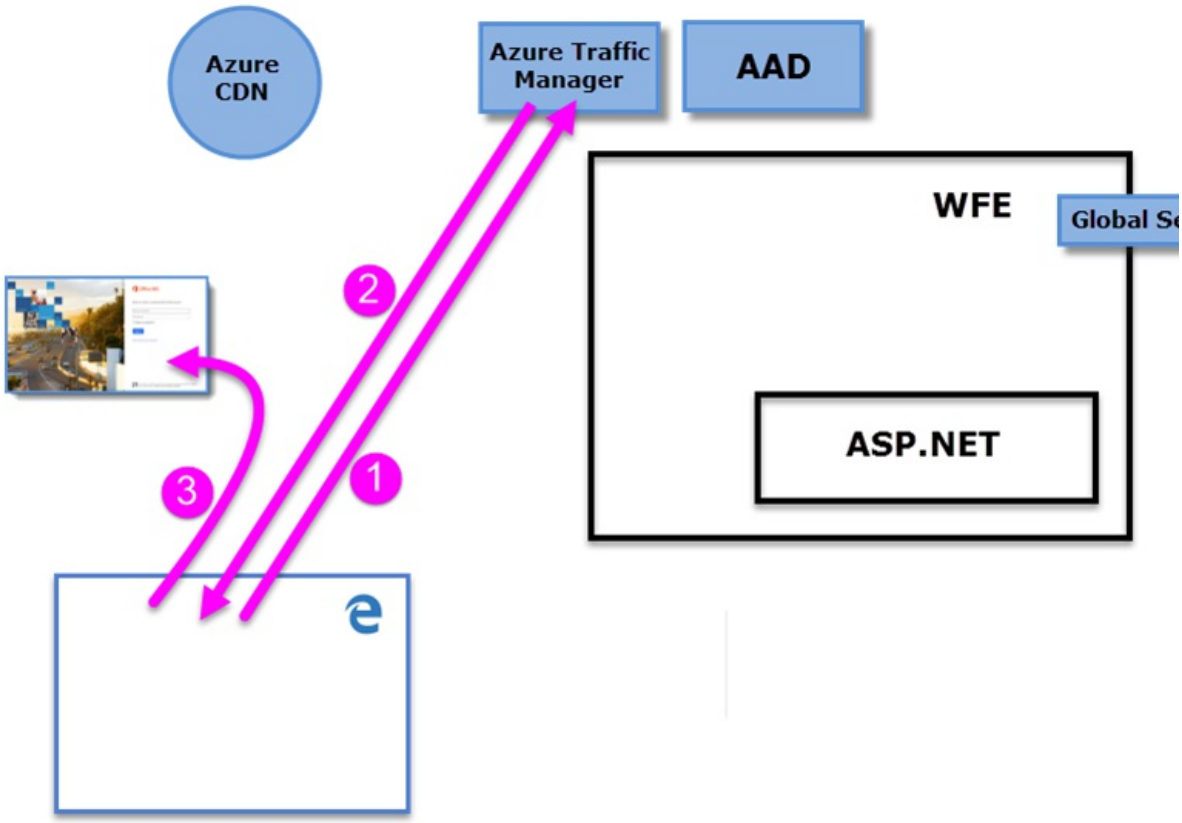
Authentication Sequence

The user authentication sequence for the Power BI service occurs as described in the following steps, which are illustrated in the following images.

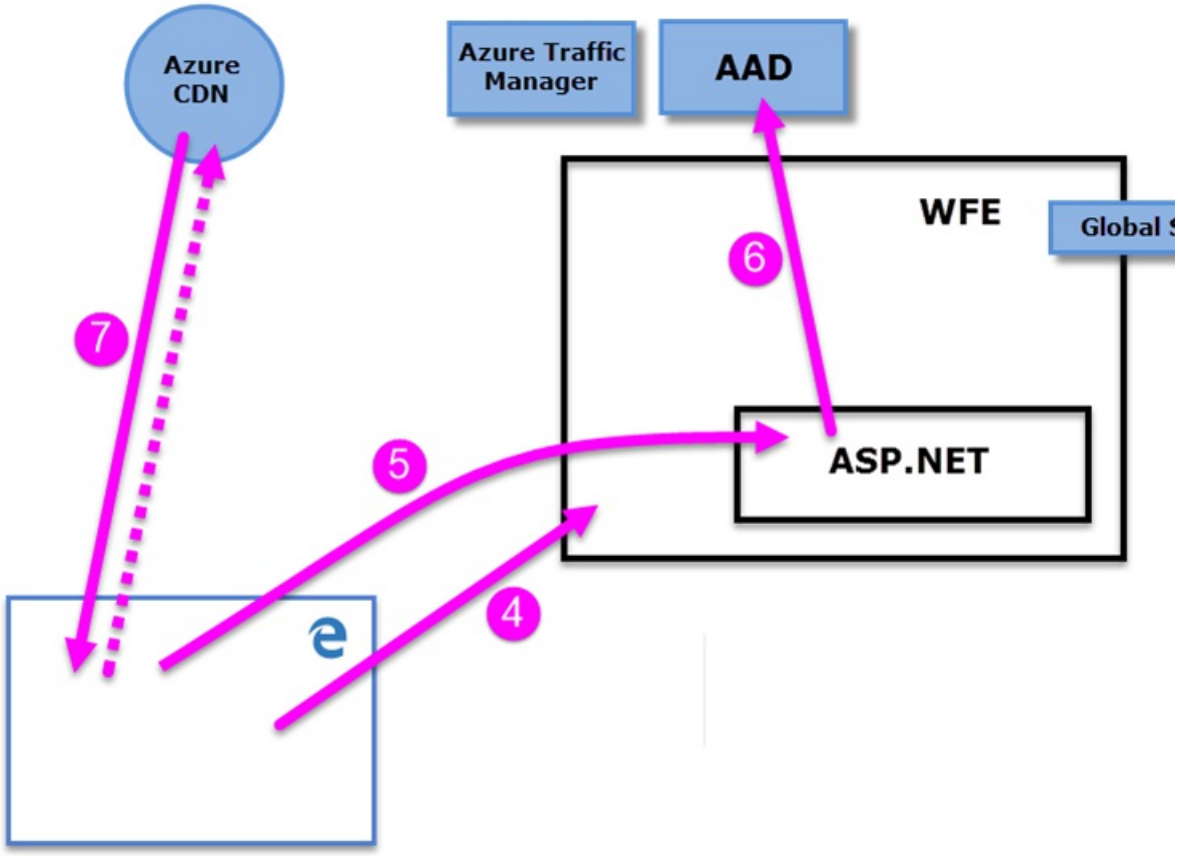
1. A user initiates a connection to the Power BI service from a browser, either by typing in the Power BI address in the address bar (such as `https://app.powerbi.com`) or by selecting *Sign In* from the Power BI landing page (https://powerbi.microsoft.com). The connection is established using TLS 1.2 and HTTPS, and all subsequent communication between the browser and the Power BI service uses HTTPS. The request is sent to the **Azure Traffic Manager**.

2. The **Azure Traffic Manager** checks the user's DNS record to determine the nearest datacenter where Power BI is deployed, and responds to the DNS with the IP address of the WFE cluster to which the user should be sent.

3. WFE then redirects the user to Microsoft Online Services login page.



4. Once the user is authenticated, the login page redirects the user to the previously determined nearest Power BI service **WFE cluster**.
5. The browser submits a cookie that was obtained from the successful login to Microsoft Online Services, which is inspected by the **ASP.NET service** inside the **WFE cluster**.
6. The WFE cluster checks with the **Azure Active Directory** (**AAD**) service to authenticate the user's Power BI service subscription, and to obtain an AAD security token. When AAD returns successful authentication of the user and returns an AAD security token, the WFE cluster consults the **Power BI**** Global Service,** which maintains a list of tenants and their Power BI Back-End cluster locations, and determines which Power BI service cluster contains the user's tenant. The WFE cluster then directs the user to the Power BI cluster where its tenant resides, and returns a collection of items to the user's browser:
    - The **AAD security token**
    - **Session information**
    - The web address of the **Back-End** cluster the user can communicate and interact with
7. The user's browser then contacts the specified Azure CDN, or for some of the files the WFE, to download the collection of specified common files necessary to enable the browser's interaction with the Power BI service. The browser page then includes the AAD token, session information, the location of the associated Back-End cluster, and the collection of files downloaded from the Azure CDN and WFE cluster, for the duration of the Power BI service browser session.



Once those items are complete, the browser initiates contact with the specified Back-End cluster and the user's interaction with the Power BI service commences. From that point forward, all calls to the Power BI service are with the specified Back-End cluster, and all calls include the user's AAD token. The AAD token has a timeout of one hour; the WFE refreshes the token periodically if a user's session remains open, in order to preserve access.

Data Storage and Movement

In the Power BI service, data is either *at rest* (data available to a Power BI user that is not currently being acted upon), or it is *in process* (for example: queries being run, data connections and models being acted upon, data and/or models being uploaded into the Power BI service, and other actions that users or the Power BI service may take on data that is actively being accessed or updated). Data that is in process is referred to as *data in process*. Data at rest in Power BI is encrypted. Data that is in transit, which means data being sent or received by the Power BI service, is also encrypted.

The Power BI service also manages data differently based on whether the data is accessed with a **DirectQuery**, or import. So there are two categories of user data for Power BI: data that is accessed by DirectQuery, and data which is not accessed by DirectQuery.

A **DirectQuery** is a query for which a Power BI user's query has been translated from Microsoft's Data Analysis Expressions (DAX) language – which is the language used by Power BI and other Microsoft products to create queries – in the data source's native data language (such as T-SQL, or other native database languages). The data associated with a DirectQuery is stored by reference only, which means source data is not stored in Power BI when the DirectQuery is not active (except for visualization data used to display dashboards and reports, as described in the *Data in process (data movement)* section, below). Rather, references to DirectQuery data are stored which allow access to that data when the DirectQuery is run. A DirectQuery contains all the necessary information to execute the query, including the connection string and the credentials used to access the data sources, which allow the DirectQuery to connect to the included data sources for automatic refresh. With a DirectQuery, underlying data model information is incorporated into the DirectQuery.

A query for an import dataset consist of a collection of DAX queries that are *not* directly translated to the native language of any underlying data source. Import queries do not include credentials for the underlying data, and the underlying data is loaded into the Power BI service unless it is on-premises data accessed through a Power BI Gateway, in which case the query only stores references to on-premises data.

The following table describes Power BI data based on the type of query being used. An **X** indicates the presence of Power BI data when using the associated query type.

	IMPORT	DIRECTQUERY	LIVE CONNECT
Schema	X	X	
Row data	X		
Visuals data caching	X	X	X

The distinction between a DirectQuery and other queries determines how the Power BI service handles the data at rest, and whether the query itself is encrypted. The following sections describe data at rest and in movement, and explain the encryption, location, and process for handling data.

Data at rest

When data is at rest, the Power BI service stores datasets, reports, and dashboard tiles in the manner described in the following subsections. As mentioned earlier, data at rest in Power BI is encrypted. ETL stands for Extract, Transform and Load in the following sections.

Encryption Keys

- The encryption keys to Azure Blob keys are stored, encrypted, in Azure Key Vault.
- The encryption keys for Azure SQL Database TDE technology is managed by Azure SQL itself.
- The encryption key for Data Movement service and on-premises data gateway are stored:
  - In the on-premises data gateway on customer's infrastructure – for on-premises data sources
  - In the Data Movement Role – for cloud-based data sources

The Content Encryption Key (CEK) used to encrypt the Microsoft Azure Blob Storage is a randomly generated 256-bit key. The algorithm that the CEK uses to encrypt the content is AES_CBC_256.

The Key Encryption Key (KEK) that is used to then encrypt the CEK is a pre-defined 256-bit key. The algorithm by KEK to encrypt the CEK is A256KW.

Gateway encryption keys based on the recovery key never leave an on-premises infrastructure. Power BI cannot access the encrypted on-premises credentials values, and cannot intercept those credentials; web clients encrypt the credential with a public key that's associated with the specific gateway with which it is communicating.

For cloud-based data sources, the Data Movement Role encrypts encryption keys using Always Encrypted methods. You can learn more about the Always Encrypted database feature.

Datasets

1. Metadata (tables, columns, measures, calculations, connection strings, etc.)

   a. For Analysis Services on-premises nothing is stored in the service except for a reference to that database stored encrypted in Azure SQL.

   b. All other metadata for ETL, DirectQuery, and Push Data is encrypted and stored in Azure Blob storage.

2. Credentials to the original data sources

   a. Analysis Services on-premises – No credentials are needed and, therefore, no credentials are stored.

b. DirectQuery – This depends whether the model is created in the service directly in which case it is stored in the connection string and encrypted in Azure Blob, or if the model is imported from Power BI Desktop in which case the credentials are stored encrypted in Data Movement's Azure SQL Database. The encryption key is stored on the machine running the Gateway on customer's infrastructure.

c. Pushed data – not applicable

d. ETL

- For **Salesforce** or **OneDrive** – the refresh tokens are stored encrypted in the Azure SQL Database of the Power BI service.
- Otherwise:
  - If the dataset is set for refresh, the credentials are stored encrypted in Data Movement's Azure SQL Database. The encryption key is stored on the machine running the Gateway on customer's infrastructure.
  - If the dataset is not set for refresh, there are no credentials stored for the data sources

3. Data

a. Analysis Services on-premises, and DirectQuery – nothing is stored in the Power BI Service.

b. ETL – encrypted in Azure Blob storage, but all data currently in Azure Blob storage of the Power BI service uses Azure Storage Service Encryption (SSE), also known as server-side encryption. Multi-geo uses SSE as well.

c. Push data v1 – stored encrypted in Azure Blob storage, but all data currently in Azure Blob storage in the Power BI service uses Azure Storage Service Encryption (SSE), also known as server-side encryption. Multi-geo uses SSE as well. Push data v1 were discontinued beginning 2016.

d. Push data v2 – stored encrypted in Azure SQL.

Power BI uses the client-side encryption approach, using cipher block chaining (CBC) mode with advanced encryption standard (AES), to encrypt its Azure Blob storage. You can learn more about client-side encryption.

Power BI provides data integrity monitoring in the following ways:

- For data at rest in Azure SQL, Power BI uses dbcc, TDE, and constant page checksum as part of the native offerings of SQL.
- For data at rest in Azure Blob storage, Power BI uses client-side encryption and HTTPS to transfer data into storage which includes integrity checks during the retrieval of the data. You can learn more about Azure Blob storage security.

Reports

1. Metadata (report definition)

a. Reports can either be Excel for Microsoft 365 reports, or Power BI reports. The following applies for metadata based on the type of report:

a. Excel Report metadata is stored encrypted in SQL Azure. Metadata is also stored in Microsoft 365.

b. Power BI reports are stored encrypted in Azure SQL database.

2. Static data

Static data includes artifacts such as background images and Power BI visuals.

a. For reports created with Excel for Microsoft 365, nothing is stored.

b. For Power BI reports, the static data is stored and is encrypted in Azure Blob storage.

3. Caches

    a. For reports created with Excel for Microsoft 365, nothing is cached.

    b. For Power BI reports, the data for the reports' visuals shown is cached and stored in the Visual Data Cache described in the following section.

4. Original Power BI Desktop (.pbix) or Excel (.xlsx) files published to Power BI

    Sometimes a copy or a shadow copy of the .xlsx or .pbix files are stored in Power BI's Azure Blob storage, and when that occurs, the data is encrypted. All such reports stored in the Power BI service, in Azure Blob storage, use Azure Storage Service Encryption (SSE), also known as server-side encryption. Multi-geo uses SSE as well.

Dashboards and Dashboard Tiles

1. Caches – The data needed by the visuals on the dashboard is usually cached and stored in the Visual Data Cache described in the following section. Other tiles such as pinned visuals from Excel or SQL Server Reporting Services (SSRS) are stored in Azure Blob as images, and are also encrypted.

2. Static data – that includes artifacts such as background images and Power BI visuals that are stored, encrypted, in Azure Blob storage.

Regardless of the encryption method used, Microsoft manages the key encryption on customers' behalf.

Visual Data Cache

Visual data is cached in different locations depending on whether the dataset is hosted on a Power BI Premium Capacity. For datasets that are not hosted on a Capacity, the visual data is cached and stored encrypted in an Azure SQL Database. For datasets that are hosted on a Capacity, the visual data can be cached in any of the following locations:

- Azure Blob Storage
- Azure Premium Files
- The Power BI Premium Capacity node

Data Transiently Stored on Non-Volatile Devices

Non-volatile devices are devices that have memory that persists without constant power. The following describes data that is transiently stored on non-volatile devices.

Datasets

1. Metadata (tables, columns, measures, calculations, connection strings, etc.)

2. Some schema-related artifacts can be stored on the disk of the compute nodes for a limited period of time. Some artifacts can also be stored in Azure REDIS Cache unencrypted for a limited period of time.

3. Credentials to the original data sources

    a. Analysis Services on-premises – nothing is stored

    b. DirectQuery – This depends whether the model is created in the service directly in which case it is stored in the connection string, in encrypted format with the encryption key stored in clear text in the same place (alongside the encrypted information); or if the model is imported from Power BI Desktop in which case the credentials are not stored on non-volatile devices.

    > **NOTE**
    > The service-side model creation feature were discontinued beginning in 2017.

    c. Pushed data – none (not applicable)

d. ETL – none (nothing stored on the compute node nor different than explained in the **Data at Rest** section, above)

4. Data

   Some data artifacts can be stored on the disk of the compute nodes for a limited period of time.

Data in process

Data is in process when it is actively being used or accessed by a user. For example, data is in process when a user accesses a dataset, revises or modifies a dashboard or report, when refresh occurs, or other data access activities that may occur. When any of those events occur and put data in process, the **Data Role** in the Power BI service creates an in-memory Analysis Services (AS) database and the dataset is loaded into that in-memory Analysis Services database. Whether the dataset is based on a DirectQuery or not, data loaded in the AS database is unencrypted to allow for access by the **Data Role**, and held in memory for further access until the Power BI service no longer needs the dataset. For customers with a Power BI Premium subscription, Power BI creates an in-memory Analysis Services (AS) database in the customer's separately provisioned collection of Power BI virtual machines.

Once data is acted upon, which includes initially loading data into Power BI, the Power BI service may cache the visualization data in an encrypted **Azure SQL Database**, regardless of whether the dataset is based on a DirectQuery.

To monitor data integrity for data in process, Power BI uses HTTPS, TCP/IP and TLS to ensure data is encrypted and maintains integrity during the transport.

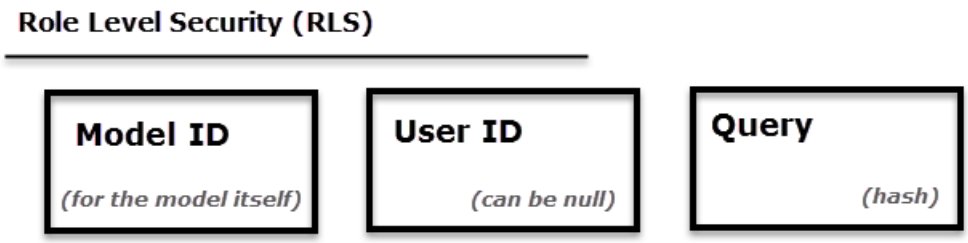
User Authentication to Data Sources

With each data source, a user establishes a connection based on their login, and accesses the data with those credentials. Users can then create queries, dashboards, and reports based on the underlying data.

When a user shares queries, dashboards, reports, or any visualization, access to that data and those visualizations is dependent on whether the underlying data sources support Role Level Security (RLS).

If an underlying data source is capable of **Power BI's**** Role Level Security (RLS)**, the Power BI service will apply that role level security, and users who do not have sufficient credentials to access the underlying data (which could be a query used in a dashboard, report, or other data artifact) will not see data for which the user does not have sufficient privileges. If a user's access to the underlying data is different from the user who created the dashboard or report, the visualizations and other artifacts will only show data based on the level of access that user has to the data.

If a data source does **not** apply RLS, then the Power BI login credentials are applied to the underlying data source, or if other credentials are supplied during the connection, those supplied credentials are applied. When a user loads data into the Power BI service from non-RLS data sources, the data is stored in Power BI as described in the **Data Storage and Movement** section found in this document. For non-RLS data sources, when data is shared with other users (such as through a dashboard or report) or a refresh of the data occurs, the original credentials are used to access or display the data.



For a quick example to contrast RLS and non-RLS data sources, imagine Sam creates a report and a dashboard, then shares them with Abby and Ralph. If the data sources used in the report and dashboard are from data sources

that **do not** support RLS, both Abby and Ralph will be able to see the data that Sam included in the dashboard (which was uploaded into the Power BI service) and both Abby and Ralph can then interact with the data. In contrast, if Sam creates a report and dashboard from data sources that do support RLS, then shares it with Abby and Ralph, when Abby attempts to view the dashboard the following occurs:

1. Since the dashboard is from an RLS data source, the dashboard visualizations will briefly show a "loading" message while the Power BI service queries the data source to retrieve the current dataset specified in the connection string associated with the dashboard's underlying query.
2. The data is accessed and retrieved based on Abby's credentials and role, and only data for which Abby has sufficient authorization is loaded into the dashboard and report.
3. The visualizations in the dashboard and report are displayed based on Abby's role level.

If Ralph were to access the shared dashboard or report, the same sequence occurs based on his role level.

Power BI Mobile

Power BI Mobile is a collection of apps designed for the three primary mobile platforms: Android, iOS, and Windows Mobile. Security considerations for Power BI Mobile apps falls into two categories:

- Device communication
- The application and data on the device

For **device communication**, all Power BI Mobile applications communicate with the Power BI service, and use the same connection and authentication sequences used by browsers, which are described in detail earlier in this whitepaper. The iOS and Android Power BI mobile applications bring up a browser session within the application itself, and the Windows mobile app brings up a broker to establish the communication channel with Power BI.

The following table lists support of certificate-based authentication (CBA) for Power BI Mobile based on mobile device platform:

CBA SUPPORT	IOS	ANDROID	WINDOWS
Power BI (sign in to service)	supported	supported	Not supported
SSRS ADFS (connect to SSRS server)	Not supported	Supported	Not supported

Power BI Mobile apps actively communicate with the Power BI service. Telemetry is used to gather mobile app usage statistics and similar data, which is transmitted to services that are used to monitor usage and activity; no personal data is sent with telemetry data.

The Power BI **application on the device** stores data on the device that facilitates use of the app:

- Azure Active Directory and refresh tokens are stored in a secure mechanism on the device, using industry-standard security measures.
- Data is cached in storage on the device, which is not directly encrypted by the application itself
- Settings are also stored on the device unencrypted, but no actual user data is stored.

The data cache from Power BI Mobile remains on the device for two weeks, or until: the app is removed; the user signs out of Power BI Mobile; or the user fails to sign in (such as a token expiration event, or password change). The data cache includes dashboards and reports previously accessed from the Power BI Mobile app.

Power BI Mobile applications do not look at folders on the device.

All three platforms for which Power BI Mobile is available support Microsoft Intune, a software service that provides mobile device and application management. With Intune enabled and configured, data on the mobile device is encrypted, and the Power BI application itself cannot be installed on an SD card. You can learn more about Microsoft Intune.

Power BI Security Questions and Answers

The following questions are common security questions and answers for Power BI. These are organized based on when they were added to this whitepaper, to facilitate your ability to quickly find new questions and answers when this paper is updated. The newest questions are added to the end of this list.

How do users connect to, and gain access to data sources while using Power BI?

- **Power BI credentials and domain credentials:** Users sign in to Power BI using an email address; when a user attempts to connect to a data resource, Power BI passes the Power BI login email address as credentials. For domain-connected resources (either on-premises or cloud-based), the login email is matched with a *User Principal Name* (UPN) by the directory service to determine whether sufficient credentials exist to allow access. For organizations that use work-based email addresses to sign in to Power BI (the same email they use to login to work resources, such as _david@contoso.com_), the mapping can occur seamlessly; for organizations that did not use work-based email addresses (such as _david@contoso.onmicrosoft.com_), directory mapping must be established in order to allow access to on-premises resources with Power BI login credentials.

- **SQL Server Analysis Services and Power BI:** For organizations that use on-premises SQL Server Analysis Services, Power BI offers the Power BI on-premises data gateway (which is a **Gateway**, as referenced in previous sections). The Power BI on-premises data gateway can enforce role-level security on data sources (RLS). For more information on RLS, see **User Authentication to Data Sources** earlier in this document. For more information about gateways, see on-premises data gateway.

  In addition, organizations can use Kerberos for **single sign-on** (SSO) and seamlessly connect from Power BI to on-premises data sources such as SQL Server, SAP HANA, and Teradata. For more information, and the specific configuration requirements, see **Use Kerberos for SSO from Power BI to on-premises data sources**.

- **Non-domain connections**: For data connections that are not domain-joined and not capable of Role Level Security (RLS), the user must provide credentials during the connection sequence, which Power BI then passes to the data source to establish the connection. If permissions are sufficient, data is loaded from the data source into the Power BI service.

How is data transferred to Power BI?

- All data requested and transmitted by Power BI is encrypted in transit using HTTPS to connect from the data source to the Power BI service. A secure connection is established with the data provider, and only once that connection is established will data traverse the network.

How does Power BI cache report, dashboard, or model data, and is it secure?

- When a data source is accessed, the Power BI service follows the process outlined in the **Data Storage and Movement** section earlier in this document.

Do clients cache web page data locally?

- When browser clients access Power BI, the Power BI web servers set the *Cache-Control* directive to *no-store*. The *no-store* directive instructs browsers not to cache the web page being viewed by the user, and not to store the web page in the client's cache folder.

What about role-based security, sharing reports or dashboards, and data connections? How does that

work in terms of data access, dashboard viewing, report access or refresh?

- For **non-Role Level Security (RLS)** enabled data sources, if a dashboard, report, or data model is shared with other users through Power BI, the data is then available for users with whom it is shared to view and interact with. Power BI *does not* re-authenticate users against the original source of the data; once data is uploaded into Power BI, the user who authenticated against the source data is responsible for managing which other users and groups can view the data.

  When data connections are made to an **RLS** -capable data source, such as an Analysis Services data source, only dashboard data is cached in Power BI. Each time a report or dataset is viewed or accessed in Power BI that uses data from the RLS-capable data source, the Power BI service accesses the data source to get data based on the user's credentials, and if sufficient permissions exist, the data is loaded into the report or data model for that user. If authentication fails, the user will see an error.

  For more information, see the **User Authentication to Data Sources** section earlier in this document.

Our users connect to the same data sources all the time, some of which require credentials that differ from their domain credentials. How can they avoid having to input these credentials each time they make a data connection?

- Power BI offers the Power BI Personal Gateway, which is a feature that lets users create credentials for multiple different data sources, then automatically use those credentials when subsequently accessing each of those data sources. For more information, see Power BI Personal Gateway.

How do Power BI Groups work?

- Power BI Groups allow users to quickly and easily collaborate on the creation of dashboards, reports, and data models within established teams. For example, if you have a Power BI Group that includes everyone in your immediate team, you can easily collaborate with everyone on your team by selecting the Group from within Power BI. Power BI Groups are equivalent to Office 365 Universal Groups (which you can learn about, create, and manage), and use the same authentication mechanisms used in Azure Active Directory to secure data. You can create groups in Power BI or create a Universal Group in Microsoft 365 admin center; either has the same result for group creation in Power BI.

  Note that data shared with Power BI Groups follows the same security consideration as any shared data in Power BI. For **non-RLS** data sources Power BI does **not** re-authenticate users against the original source of data, and once data is uploaded into Power BI, the user who authenticated against the source data is responsible for managing which other users and groups can view the data. For more information, see the **User Authentication to Data Sources** section earlier in this document.

  You can get more information about Groups in Power BI.

Which ports are used by on-premises data gateway and personal gateway? Are there any domain names that need to be allowed for connectivity purposes?

- The detailed answer to this question is available at the following link: Gateway ports

When working with the on-premises data gateway, how are recovery keys used and where are they stored? What about secure credential management?

- During gateway installation and configuration, the administrator types in a gateway **Recovery Key**. That **Recovery Key** is used to generate a strong **AES** symmetric key. An **RSA** asymmetric key is also created at the same time.

  Those generated keys (**RSA** and **AES**) are stored in a file located on the local machine. That file is also encrypted. The contents of the file can only be decrypted by that particular Windows machine, and only by that particular gateway service account.

When a user enters data source credentials in the Power BI service UI, the credentials are encrypted with the public key in the browser. The gateway decrypts the credentials using the RSA private key and re-encrypts them with an AES symmetric key before the data is stored in the Power BI service. With this process, the Power BI service never has access to the unencrypted data.

Which communication protocols are used by the on-premises data gateway, and how are they secured?

- The gateway supports the following two communications protocols:
  - **AMQP 1.0 – TCP + TLS**: This protocol requires ports 443, 5671-5672, and 9350-9354 to be open for outgoing communication. This protocol is preferred, since it has lower communication overhead.
  - **HTTPS – WebSockets over HTTPS + TLS**: This protocol uses port 443 only. The WebSocket is initiated by a single HTTP CONNECT message. Once the channel is established, the communication is essentially TCP+TLS. You can force the gateway to use this protocol by modifying a setting described in the on-premises gateway article.

What is the role of Azure CDN in Power BI?

- As mentioned previously, Power BI uses the **Azure Content Delivery Network (CDN)** to efficiently distribute the necessary static content and files to users based on geographical locale. To go into further detail, the Power BI service uses multiple **CDNs** to efficiently distribute necessary static content and files to users through the public Internet. These static files include product downloads (such as **Power BI Desktop**, the **on-premises data gateway**, or Power BI apps from various independent service providers), browser configuration files used to initiate and establish any subsequent connections with the Power BI service, as well as the initial secure Power BI login page.

  Based on information provided during an initial connection to the Power BI service, a user's browser contacts the specified Azure **CDN** (or for some files, the **WFE**) to download the collection of specified common files necessary to enable the browser's interaction with the Power BI service. The browser page then includes the AAD token, session information, the location of the associated **Back-End** cluster, and the collection of files downloaded from the Azure **CDN** and **WFE** cluster, for the duration of the Power BI service browser session.

For Power BI visuals, does Microsoft perform any security or privacy assessment of the custom visual code prior to publishing items to the Gallery?

- No. It is the customer's responsibility to review and determine whether custom visual code should be relied upon. All custom visual code is operated in a sandbox environment, so that any errant code in a custom visual does not adversely affect the rest of the Power BI service.

Are there other Power BI visuals that send information outside the customer network?

- Yes. Bing Maps and ESRI visuals transmit data out of the Power BI service for visuals that use those services.

For Template Apps, does Microsoft perform any security or privacy assessment of the Template app prior to publishing items to the Gallery?

- No. The app publisher is responsible for the content while the customer's responsibility to review and determine whether to trust the Template app publisher.

Are there Template apps that can send information outside the customer network?

- Yes. It is the customer's responsibility to review the publisher's privacy policy and determine whether to install the Template app on Tenant. Furthermore, the publisher is responsible to notify of the app's behavior and capabilities.

What about data sovereignty? Can we provision tenants in data centers located in specific

geographies, to ensure data doesn't leave the country borders?

- Some customers in certain geographies have an option to create a tenant in a national cloud, where data storage and processing is kept separate from all other datacenters. National clouds have a slightly different type of security, since a separate data trustee operates the national cloud Power BI service on behalf of Microsoft.

  Alternatively customers can also set up a tenant in a specific region, however, such tenants do not have a separate data trustee from Microsoft. Pricing for national clouds is different from the generally available commercial Power BI service. For more information about Power BI service availability for national clouds, see Power BI national clouds.

How does Microsoft treat connections for customers who have Power BI Premium subscriptions? Are those connections different than those established for the non-Premium Power BI service?

- The connections established for customers with Power BI Premium subscriptions implement an Azure Business-to-Business (B2B) authorization process, using Azure Active Directory (AD) to enable access control and authorization. Power BI handles connections from Power BI Premium subscribers to Power BI Premium resources just as it would any other Azure AD user.

Conclusion

The Power BI service architecture is based on two clusters – the Web Front End (WFE) cluster and the Back-End cluster. The WFE cluster is responsible for initial connection and authentication to the Power BI service, and once authenticated, the Back End handles all subsequent user interactions. Power BI uses Azure Active Directory (AAD) to store and manage user identities, and manages the storage of data and metadata using Azure Blob and Azure SQL Database, respectively.

Data storage and data processing in Power BI differs based on whether data is accessed using a DirectQuery, and is also dependent on whether data sources are in the cloud or on-premises. Power BI is also capable of enforcing Role Level Security (RLS) and interacts with Gateways that provide access to on-premises data.

Feedback and Suggestions

We appreciate your feedback. We're interested in hearing any suggestions you have for improvement, additions, or clarifications to this whitepaper, or other content related to Power BI. Send your suggestions to pbidocfeedback@microsoft.com.

Additional Resources

For more information on Power BI, see the following resources.

- Groups in Power BI
- Getting Started with Power BI Desktop
- Power BI REST API - Overview
- Power BI API reference
- On-premises data gateway
- Power BI National Clouds
- Power BI Premium
- Use Kerberos for SSO from Power BI to on-premises data sources

Power BI enterprise deployment whitepaper

5/20/2020 • 2 minutes to read • Edit Online

Deploying Power BI in a large enterprise is a complex task that requires a lot of thought and planning. Getting proper guidance and best practices can help you understand the choices you will make, gather requirements, and learn best practices that can make your Power BI enterprise deployment a success. To facilitate those steps, and more, Microsoft is providing the Power BI Enterprise Deployment whitepaper.

About the whitepaper

The purpose of the Enterprise Deployment whitepaper is to help make your Power BI deployment a success: it covers key considerations, the decisions which will be necessary throughout the process, and potential issues you may encounter. Best practices and suggestions are offered when possible.

The target audience for this whitepaper is technology professionals. Some knowledge of Power BI and general business intelligence concepts is assumed.

You can **download the complete enterprise deployment whitepaper** at the following link:

- **Planning a Power BI Enterprise Deployment whitepaper**

Next steps

For more information on Power BI, see the following resources:

- Whitepapers for Power BI
- Power BI security whitepaper
- Power BI Premium

Deploying and Managing Power BI Premium Capacities

5/13/2020 • 2 minutes to read • Edit Online

We have retired the Power BI Premium whitepaper in favor of providing up-to-date information in separate articles. Use the following table to find content from the whitepaper.

ARTICLES	DESCRIPTION
Basic concepts for designers in the Power BI service Datasets in the Power BI service Dataset modes in the Power BI service	Background information about Power BI service capacities, workspaces, dashboards, reports, workbooks, datasets, and dataflows.
What is Power BI Premium?	An overview of Power BI Premium, covering the basics of dedicated capacities, supported workloads, unlimited content sharing, and other features.
Managing Premium capacities Configure and manage capacities in Power BI Premium	
Configure workloads in a Premium capacity	Detailed information about configuring and managing dedicated capacities and workloads.
Optimizing Premium capacities	Best practices for performance optimization, optimizing models, capacity planning, and testing approaches.
Premium capacity scenarios	Common issues in real-world scenarios, with a focus on identifying and resolving those issues.
Monitor capacities in the Admin portal	Monitoring with Power BI Premium Capacity Metrics app, and interpreting the metrics you see in the app.
Power BI Premium FAQ	Answers to questions around purchase and licensing, features, and common scenarios.

Distribute Power BI content to external guest users using Azure Active Directory B2B

5/18/2020 • 42 minutes to read • Edit Online

Summary: This is a technical whitepaper outlining how to distribute content to users outside the organization using the integration of Azure Active Directory Business-to-business (Azure AD B2B).

Writers: Lukasz Pawlowski, Kasper de Jonge

Technical Reviewers: Adam Wilson, Sheng Liu, Qian Liang, Sergei Gundorov, Jacob Grimm, Adam Saxton, Maya Shenhav, Nimrod Shalit, Elisabeth Olson

> **NOTE**
> You can save or print this whitepaper by selecting **Print** from your browser, then selecting **Save as PDF**.

Introduction

Power BI gives organizations a 360-degree view of their business and empowers everyone in these organizations to make intelligent decisions using data. Many of these organizations have strong and trusted relationships with external partners, clients, and contractors. These organizations need to provide secure access to Power BI dashboards and reports to users in these external partners.

Power BI integrates with Azure Active Directory Business-to-business (Azure AD B2B) to allow secure distribution of Power BI content to guest users outside the organization – while still maintaining control and governing access to internal data.

This white paper covers the all the details you need to understand Power BI's integration with Azure Active Directory B2B. We cover its most common use case, setup, licensing, and row level security.

> **NOTE**
> Throughout this white paper, we refer to Azure Active Directory as Azure AD and Azure Active Directory Business to Business as Azure AD B2B.

Scenarios

Contoso is an automotive manufacturer and works with many diverse suppliers who provide it with all the components, materials, and services necessary to run its manufacturing operations. Contoso wants to streamline its supply chain logistics and plans to use Power BI to monitor key performance metrics of its supply chain. Contoso wants to share with external supply chain partners analytics in a secure and manageable way.

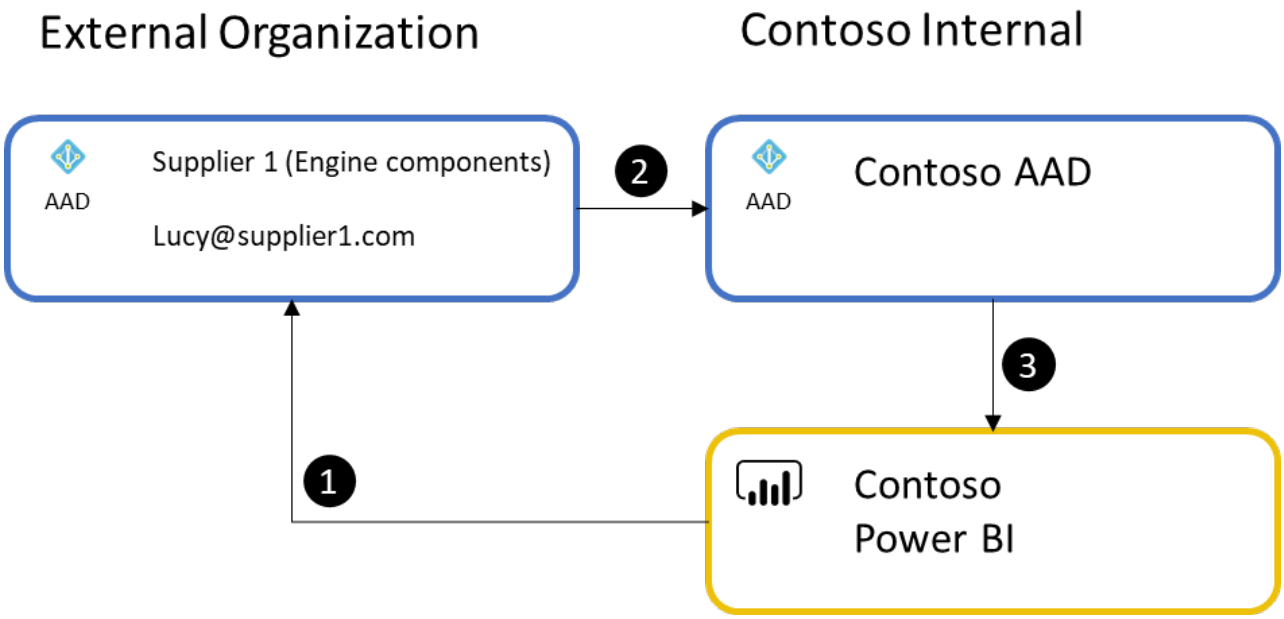
Contoso can enable the following experiences for external users using Power BI and Azure AD B2B.

Ad hoc per item sharing

Contoso works with a supplier who builds radiators for Contoso's cars. Often, they need to optimize the reliability of the radiators using data from all of Contoso's cars. An analyst at Contoso uses Power BI to share a radiator reliability report with an Engineer at the supplier. The Engineer receives an email with a link to view the report.

As described above, this ad-hoc sharing is performed by business users on an as needed basis. The link sent by

Power BI to the external user is an Azure AD B2B invite link. When the external user opens the link, they are asked to join Contoso's Azure AD organization as a Guest user. After the invite is accepted, the link opens the specific report or dashboard. The Azure Active Directory admin delegates permission to invite external users to the organization and chooses what those users can do once they accept the invite as described in the Governance section of this document. The Contoso analyst can invite the Guest user only because the Azure AD administrator allowed that action and the Power BI administrator allowed users to invite guests to view content in Power BI's tenant settings.



1. The process starts with a Contoso internal user sharing a dashboard or a report with an external user. If the external user is not already a guest in Contoso's Azure AD, they are invited. An email is sent to their email address that includes an invite to Contoso's Azure AD
2. The recipient accepts the invite to Contoso's Azure AD and is added as a Guest user in Contoso's Azure AD.
3. The recipient is then redirected to the Power BI dashboard, report, or app, which are read-only for the user.

The process is considered ad-hoc since business users in Contoso perform the invite action as needed for their business purposes. Each item shared is a single link the external user can access to view the content.

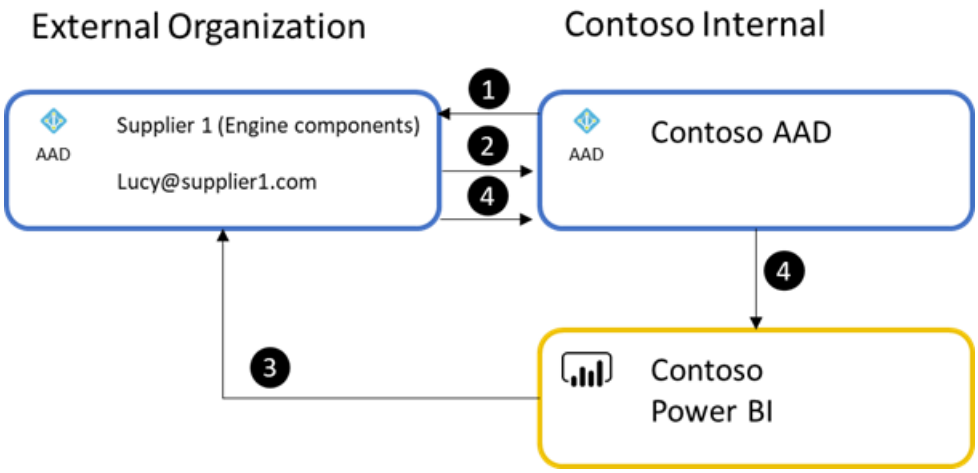
Once the external user has been invited to access Contoso resources, a shadow account may be created for them in Contoso Azure AD and they do not need to be invited again. The first time they try to access a Contoso resource like a Power BI dashboard, they go through a consent process, which redeems the invitation. If they do not complete the consent, they cannot access any of Contoso's content. If they have trouble redeeming their invitation via the original link provided, an Azure AD administrator can resent a specific invitation link for them to redeem.

Planned per item sharing

Contoso works with a subcontractor to perform reliability analysis of radiators. The subcontractor has a team of 10 people who need access to data in Contoso's Power BI environment. The Contoso Azure AD administrator is involved to invite all the users and to handle any additions/changes as personnel at the subcontractor change. The Azure AD administrator creates a security group for all the employees at the subcontractor. Using the security group, Contoso's employees can easily manage access to reports and ensure all required subcontractor personnel have access to all the required reports, dashboards, and Power BI apps. The Azure AD administrator can also avoid being involved in the invitation process altogether by choosing to delegate invitation rights to a trusted employee at Contoso or at the subcontractor to ensure timely personnel management.

Some organizations require more control over when external users are added, are inviting many users in an external organization, or many external organizations. In these cases, planned sharing can be used to manage the scale of sharing, to enforce organizational policies, and even to delegate rights to trusted individuals to invite and manage external users. Azure AD B2B supports planned invites to be sent directly from the Azure portal by an IT

administrator, or through PowerShell using the invitation manager API where a set of users can be invited in one action. Using the planned invites approach, the organization can control who can invite users and implement approval processes. Advanced Azure AD capabilities like dynamic groups can make it easy to maintain security group membership automatically.

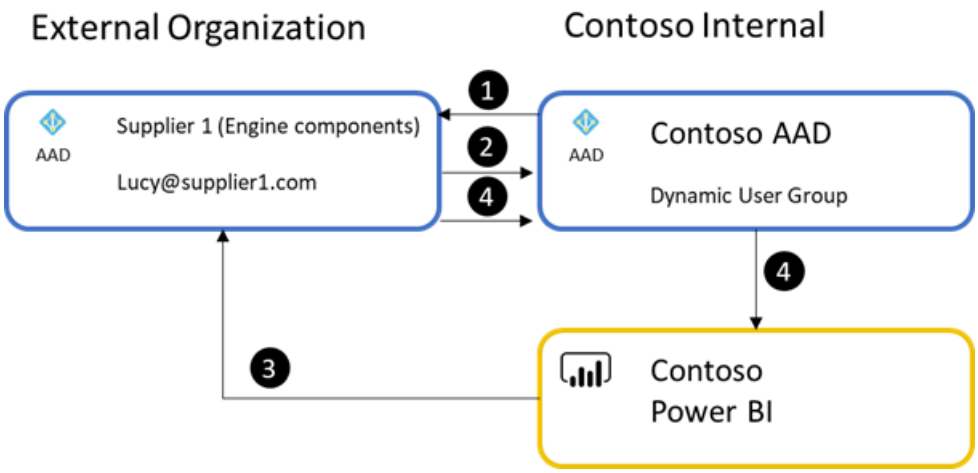


1. The process stars with an IT administrator inviting the guest user either manually or through the API provided by Azure Active Directory
2. The user accepts the invite to the organization.
3. Once the user has accepted the invitation, a user in Power BI can share a report or dashboard with the external user, or a security group they are in. Just like with regular sharing in Power BI the external user receives an email with the link to the item.
4. When the external user accesses the link, their authentication in their directory is passed to Contoso's Azure AD and used to gain access to the Power BI content.

Ad hoc or planned sharing of Power BI Apps

Contoso has a set of reports and dashboards they need to share with one or more Suppliers. To ensure all required external users have access to this content, it is packaged as a Power BI app. The external users are either added directly to the app access list or through security groups. Someone at Contoso then sends the app URL to all the external users, for example in an email. When the external users open the link, they see all the content in a single easy to navigate experience.

Using a Power BI app makes it easy for Contoso to build a BI Portal for its suppliers. A single access list controls access to all the required content reducing wasted time checking and setting item level permissions. Azure AD B2B maintains security access using the Supplier's native identity so users don't need additional login credentials. If using planned invites with security groups, access management to the app as personnel rotate into or out of the project is simplified. Membership in security groups manually or by using dynamic groups, so that all external users from a supplier are automatically added to the appropriate security group.



1. The process starts by the user being invited to Contoso's Azure AD organization through the Azure portal or PowerShell
2. The user can be added to a user group in Azure AD. A static or dynamic user group can be used, but dynamic groups help reduce manual work.
3. The external users are given access to the Power BI App through the user group. The app URL should be sent directly to the external user or placed on a site they have access to. Power BI makes a best effort to send an email with the app link to external users but when using user groups whose membership can change, Power BI is not able to send to all external users managed through user groups.
4. When the external user accesses the Power BI app URL, they are authenticated by Contoso's Azure AD, the app is installed for the user, and the user can see all the contained reports and dashboards within the app.

Apps also have a unique feature that allows app authors to install the application automatically for the user, so it is available when the user logs in. This feature only installs automatically for external users who are already part of Contoso's organization at the time the application is published or updated. Thus, it is best used with the planned invites approach, and depends on the app being published or updated after the users are added to Contoso's Azure AD. External users can always install the app using the app link.

Commenting and subscribing to content across organizations

As Contoso continues to work with its subcontractors or suppliers, the external Engineers need to work closely with Contoso's analysts. Power BI provides several collaboration features that help users communicate about content they can consume. Dashboard commenting (and soon Report commenting) allows users to discuss data points they see and communicate with report authors to ask questions.

Currently, external guest users can participate in comments by leaving comments and reading the replies. However, unlike internal users, guest users cannot be @mentioned and do not receive notifications that they've received a comment. Guest users cannot use the subscriptions feature within Power BI at the time of writing. In an upcoming release, those restrictions will be lifted and the Guest user will receive an email when a comment @mentions them, or when a subscription is delivered to their email that contains a link to the content in Power BI.

Access content in the Power BI mobile apps

In an upcoming release, when Contoso's users share reports or dashboards with their external Guest counterparts, Power BI will send an email notifying the Guest. When the guest user opens the link to the report or dashboard on their mobile device, the content will open in the native Power BI mobile apps on their device, if they're installed. The guest user will then be able to navigate between content shared with them in the external tenant, and back to their own content from their home tenant.

> **NOTE**
>
> The guest user cannot open the Power BI mobile app and immediately navigate to the external tenant, they must start with a link to an item in the external tenant. Common workarounds are described in the Distributing links to content in the Parent organization's Power BI section later in this document.

Cross-organization editing and management of Power BI content

Contoso and its Suppliers and subcontractors work increasingly closely together. Often an analyst at the subcontractor needs additional metrics or data visualizations to be added to a report Contoso has shared with them. The data should reside in Contoso's Power BI tenant, but external users should be able to edit it, create new content, and even distribute it to appropriate individuals.

Power BI provides an option that enables **External guest users can edit and manage content** in the organization. By default, external users have a read-only consumption-oriented experience. However, this new setting allows the Power BI admin to choose which external users can edit and manage content within their own organization. Once allowed, the external user can edit reports, dashboards, publish or update apps, work in workspaces, and connect to data they have permission to use.

This scenario is described in detail in the section Enabling external users to edit and manage content within Power BI later in this document.

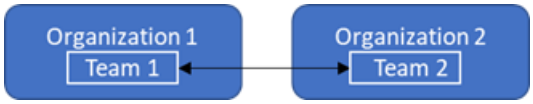
Organizational relationships using Power BI and Azure AD B2B

When all the users of Power BI are internal to the organization, there is no need to use Azure AD B2B. However, once two or more organizations want to collaborate on data and insights, Power BI's support for Azure AD B2B makes it easy and cost effective to do so.

Below are typically encountered organizational structures that are well suited for Azure AD B2B style cross-organization collaboration in Power BI. Azure AD B2B works well in most cases, but in some situations the Common alternative approaches covered at the end of this document are worth considering.

Case 1: Direct collaboration between organizations

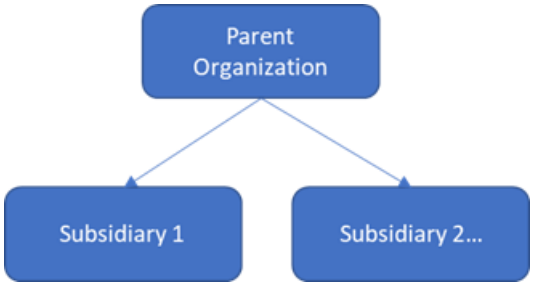
Contoso's relationship with its radiator supplier is an example of direct collaboration between organizations. Since there are relatively few users at Contoso and its supplier who need access to radiator reliability information, using Azure AD B2B based external sharing is ideal. It is easy to use and simple to administer. This is also a common pattern in consulting services where a consultant may need to build content for an organization.



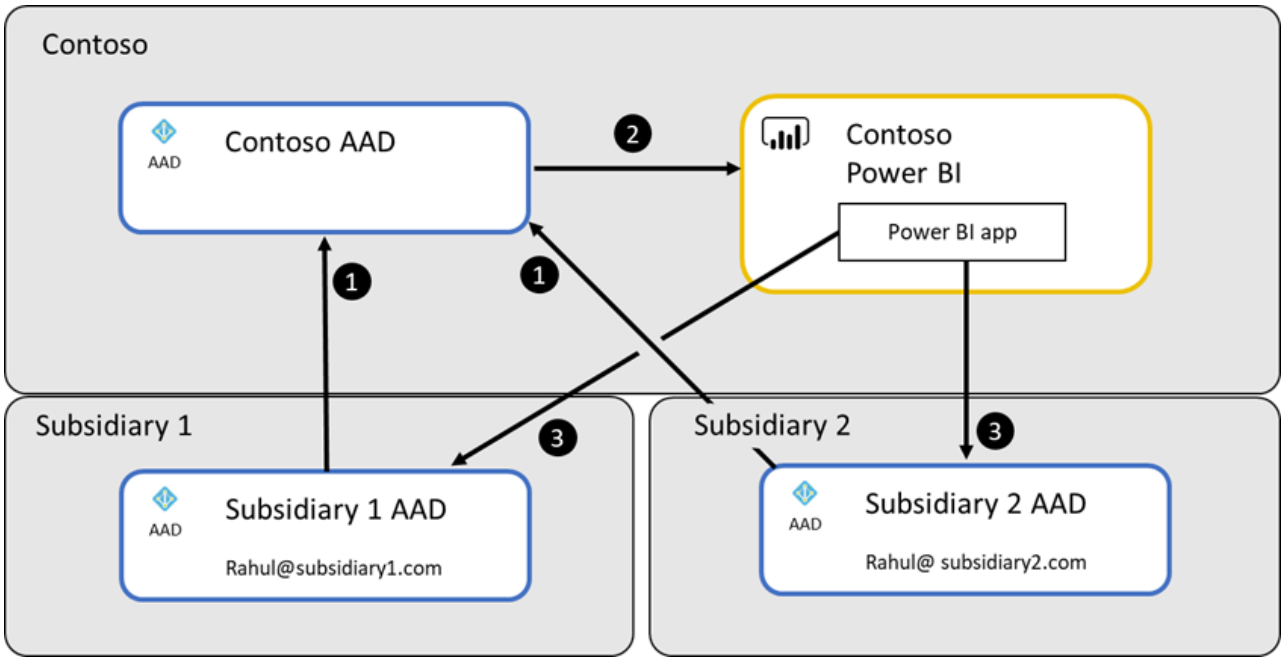
Typically, this sharing occurs initially using Ad hoc per item sharing. However, as teams grow or relationships deepen, the Planned per item sharing approach becomes the preferred method to reduce management overhead. Additionally, the Ad hoc or planned sharing of Power BI Apps, Commenting and subscribing to content across organizations, access to content in mobile apps can come into play as well, and cross-organization editing and management of Power BI content. Importantly, if both organizations' users have Power BI Pro licenses in their respective organizations, they can use those Pro licenses in each other's Power BI environments. This provides advantageous licensing since the inviting organization may not need to pay for a Power BI Pro license for the external users. This is discussed in more detail in the Licensing section later in this document.

Case 2: Parent and its subsidiaries or affiliates

Some organization structures are more complex, including partially or wholly owned subsidiaries, affiliated companies, or managed service provider relationships. These organizations have a parent organization such as a holding company, but the underlying organizations operate semi-autonomously, sometimes under different regional requirements. This leads to each organization having its own Azure AD environment and separate Power BI tenants.



In this structure, the parent organization typically needs to distribute standardized insights to its subsidiaries. Typically, this sharing occurs using the Ad hoc or planned sharing of Power BI Apps approach as illustrated in the following image, since it allows distribution of standardized authoritative content to broad audiences. In practice a combination of all the Scenarios mentioned earlier in this document is used.



This follows the following process:

1. Users from each Subsidiary are invited to Contoso's Azure AD
2. Then the Power BI app is published to give these users access to the required data
3. Finally, the users open the app through a link they've been given to see the reports

Several important challenges are faced by organizations in this structure:

- How to distribute links to content in the Parent organization's Power BI
- How to allow subsidiary users to access data source hosted by the parent organization

Distributing links to content in the Parent organization's Power BI

Three approaches are commonly used to distribute links to the content. The first and most basic is to send the link to the app to the required users or to place it in a SharePoint Online site from which it can be opened. Users can then bookmark the link in their browsers for faster access to the data they need.

The second approach relies on the cross-organization editing and management of Power BI content capability. The Parent organization allows users from the subsidiaries to access its Power BI and controls what they can access through permission. This gives access to Power BI Home where the user from the subsidiary sees a comprehensive list of content shared to them in the Parent organization's tenant. Then the URL to the Parent organizations' Power BI environment is given to the users at the subsidiaries.

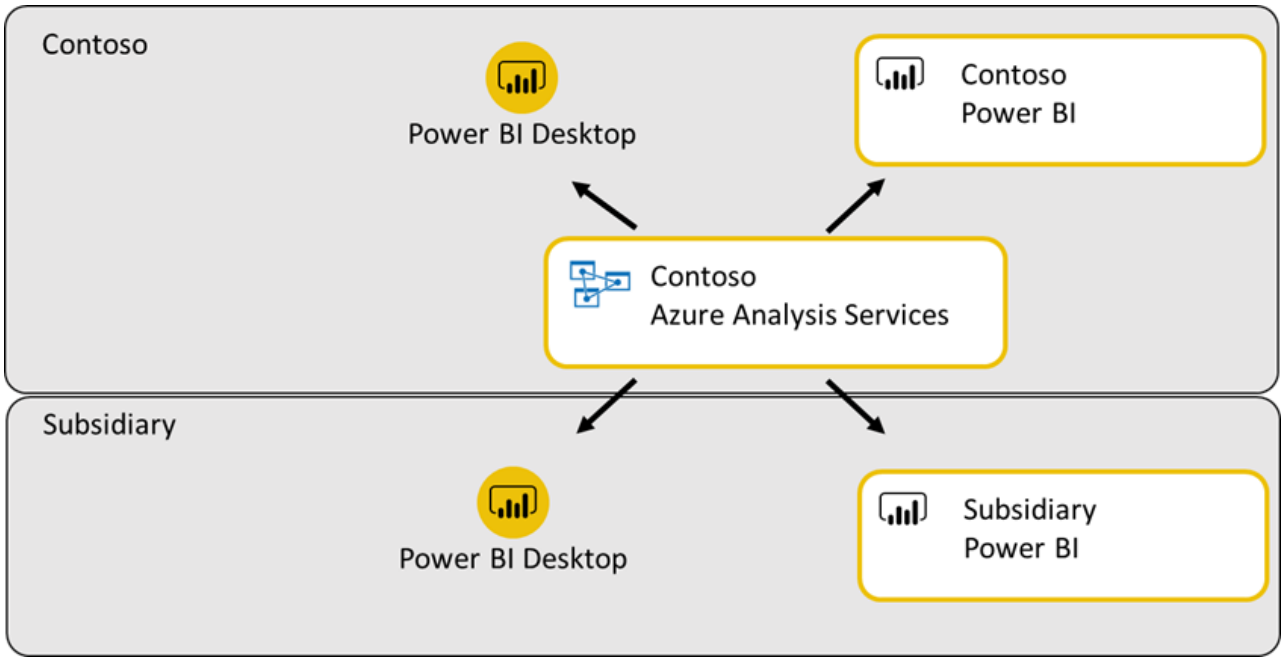
The final approach uses a Power BI app created within the Power BI tenant for each subsidiary. The Power BI app includes a dashboard with tiles configured with the external link option. When the user presses the tile, they are taken to the appropriate report, dashboard, or app in the parent organization's Power BI. This approach has the added advantage that the app can be installed automatically for all users in the subsidiary and is available to them whenever they sign in to their own Power BI environment. An added advantage of this approach is that it works well with the Power BI mobile apps that can open the link natively. You can also combine this with the second approach to enable easier switching between Power BI environments.

Allowing subsidiary users to access data sources hosted by the parent organization

Often analysts at a subsidiary need to create their own analytics using data supplied by the parent organization. In this case, commonly cloud data sources are used to address the challenge.

The first approach leverages Azure Analysis Services to build an enterprise grade data warehouse that serves the needs of Analysts across the parent and its subsidiaries as shown the following image. Contoso can host the data and use capabilities like row level security to ensure users in each subsidiary can access only their data. Analysts at each organization can access the data warehouse through Power BI Desktop and publish resulting analytics to their

respective Power BI tenants.

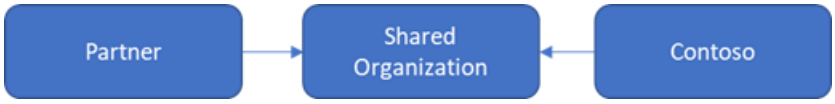


The second approach leverages Azure SQL Database to build a relational data warehouse to provide access to data. This works similarly to the Azure Analysis Services approach, though some capabilities like row level security may be harder to deploy and maintain across subsidiaries.

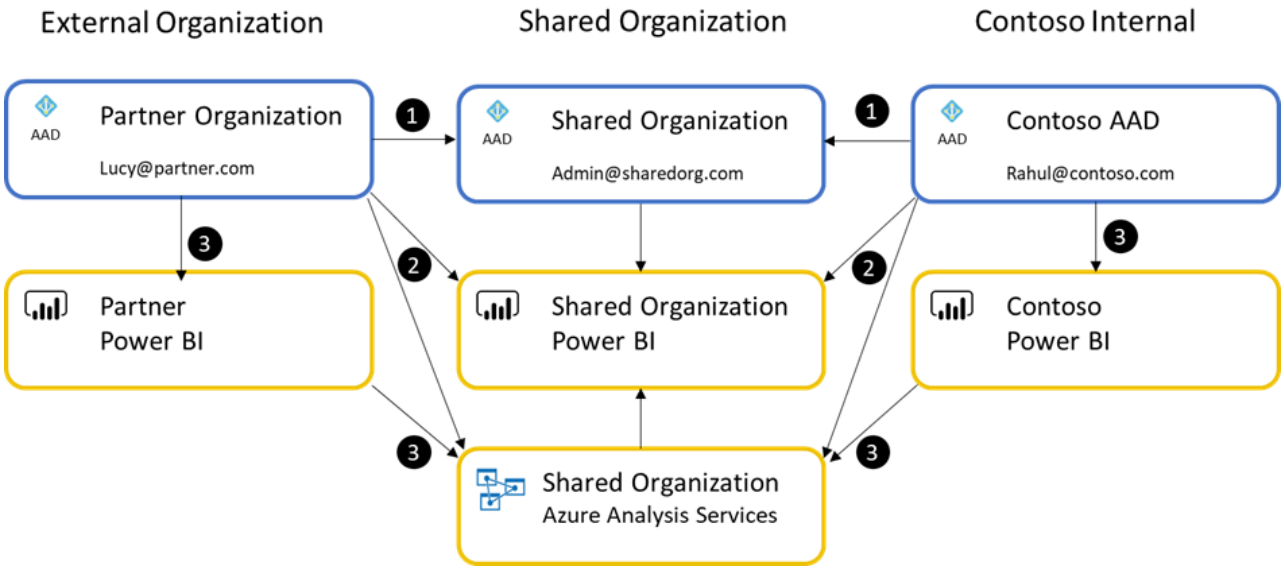
More sophisticated approaches are also possible, however the above are by far the most common.

Case 3: Shared environment across partners

Contoso may enter into a partnership with a competitor to jointly build a car on a shared assembly line, but to distribute the vehicle under different brands or in different regions. This requires extensive collaboration and co-ownership of data, intelligence, and analytics across organizations. This structure is also common in the consulting services industry where a team of consultants may do project-based analytics for a client.



In practice, these structures are complex as shown in the following image, and require staff to maintain. To be effective this structure relies on the cross-organization editing and management of Power BI content capability since it allows organizations to reuse Power BI Pro licenses purchased for their respective Power BI tenants.



To establish a shared Power BI tenant, an Azure Active Directory needs to be created and at least one Power BI Pro

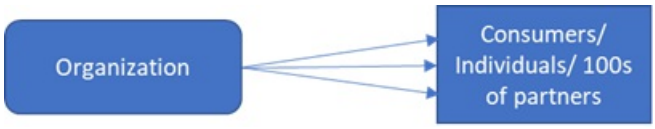
user account needs to be purchased for a user in that active directory. This user invites the required users to the shared organization. Importantly, in this scenario, Contoso's users are treated as external users when they operate within the Shared Organization's Power BI.

The process is as follows:

1. The Shared Organization is established as a new Azure Active Directory and at least one user account is created in the new organization. That user should have a Power BI Pro license assigned to them.
2. This user then establishes a Power BI tenant and invites the required users from Contoso and the Partner organization. The user also establishes any shared data assets like Azure Analysis Services. Contoso and the Partner's users can access the shared organization's Power BI as guest users. If allowed to edit and manage content in Power BI the external users can use Power BI home, use workspaces, upload, or edit content and share reports. Typically, all shared assets are stored and accessed from the shared organization.
3. Depending on how the parties agree to collaborate, it is possible for each organization to develop their own proprietary data and analytics using shared data warehouse assets. They can distribute those to their respective internal users using their internal Power BI tenants.

Case 4: Distribution to hundreds or thousands of external partners

While Contoso created a radiator reliability report for one Supplier, now Contoso desires to create a set of standardized reports for hundreds of Suppliers. This allows Contoso to ensure all suppliers have the analytics they need to make improvements or to fix manufacturing defects.



When an organization needs to distribute standardized data and insights to many external users/organizations, they can use the Ad hoc or planned sharing of Power BI Apps scenario to build a BI Portal quickly and without extensive development costs. The process to build such a portal using a Power BI app is covered in the Case Study: Building a BI Portal using Power BI + Azure AD B2B – Step-by-Step instructions later in this document.

A common variant of this case is when an organization is attempting to share insights with consumers, especially when looking to use Azure B2C with Power BI. Power BI does not natively support Azure B2C. If you're evaluating options for this case, consider using Alternative Option 2 in the Common alternative approaches the section later in this document.

Case Study: Building a BI Portal using Power BI + Azure AD B2B – Step-by-Step instructions

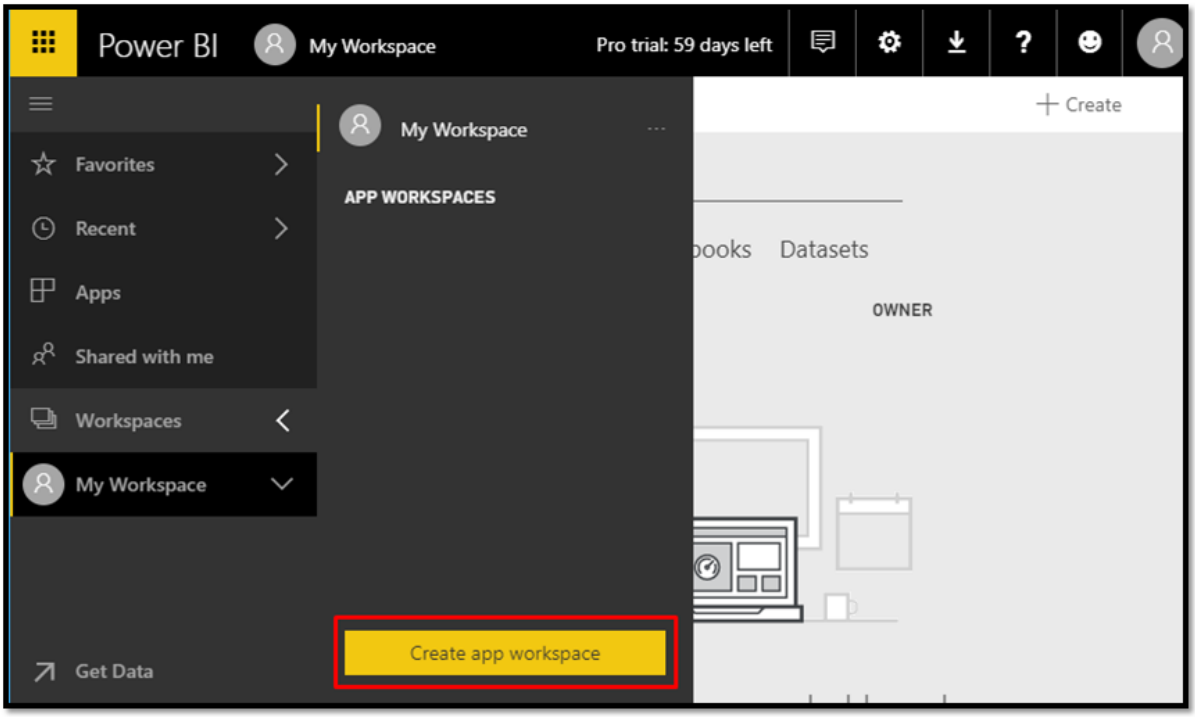
Power BI's integration with Azure AD B2B gives Contoso a seamless, hassle-free way to provide guest users with secure access to its BI portal. Contoso can set this up with three steps:

1. Create BI portal in Power BI

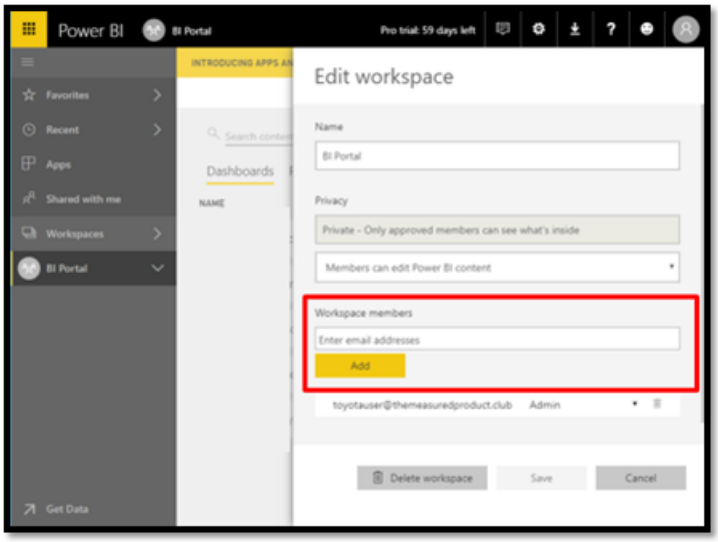
   The first task for Contoso is to create their BI portal in Power BI. Contoso's BI portal will consist of a collection of purpose-built dashboards and reports that will be made available to many internal and guest users. The

recommended way for doing this in Power BI is to build a Power BI app. Learn more about apps in Power BI.

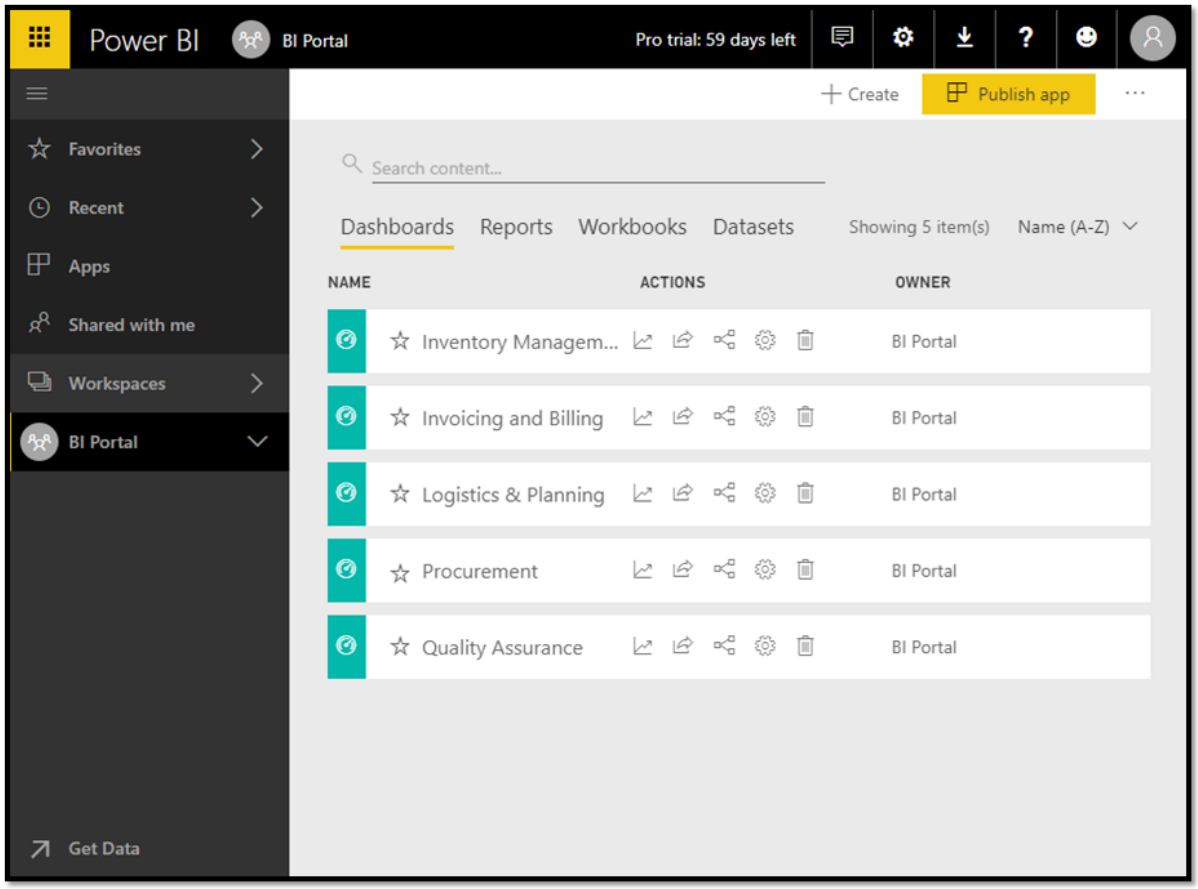
- Contoso's BI team creates a workspace in Power BI



- Other authors are added to the workspace



- Content is created inside the workspace



Now that the content is created in a workspace, Contoso is ready to invite guest users in partner organizations to consume this content.

2. Invite Guest Users

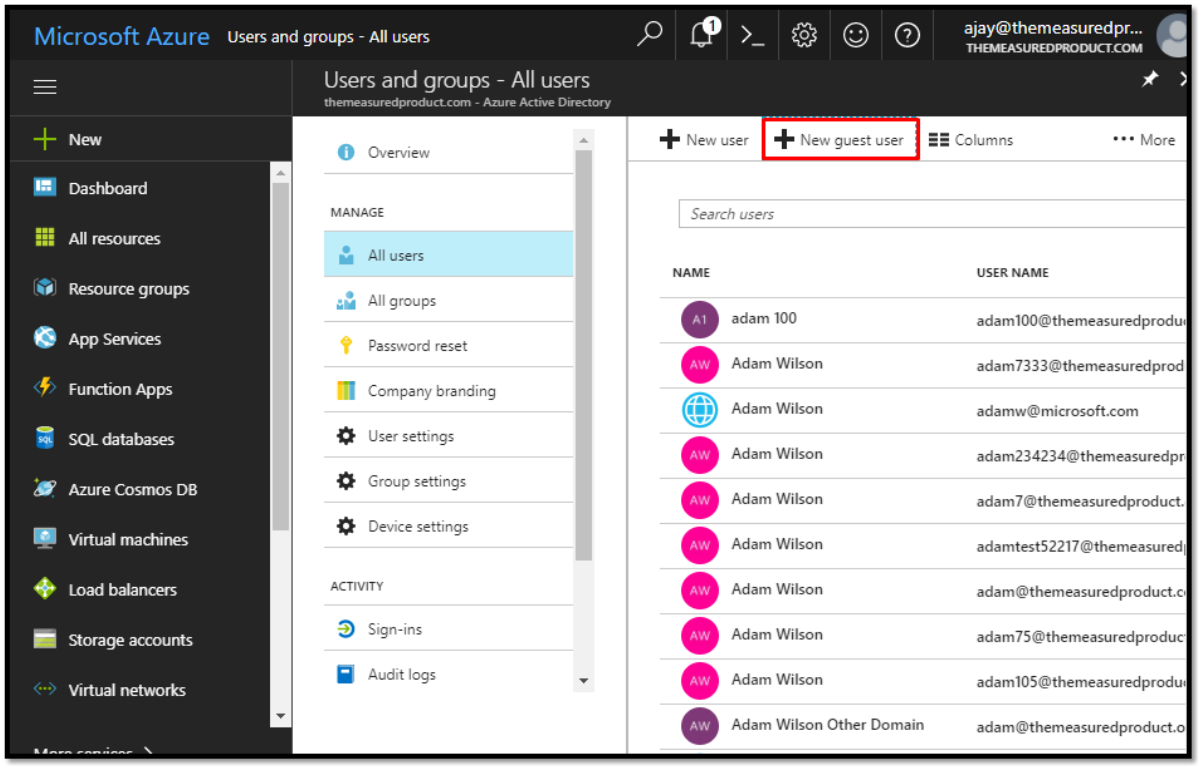
   There are two ways for Contoso to invite guest users to its BI portal in Power BI:

   - Planned Invites
   - Ad hoc Invites

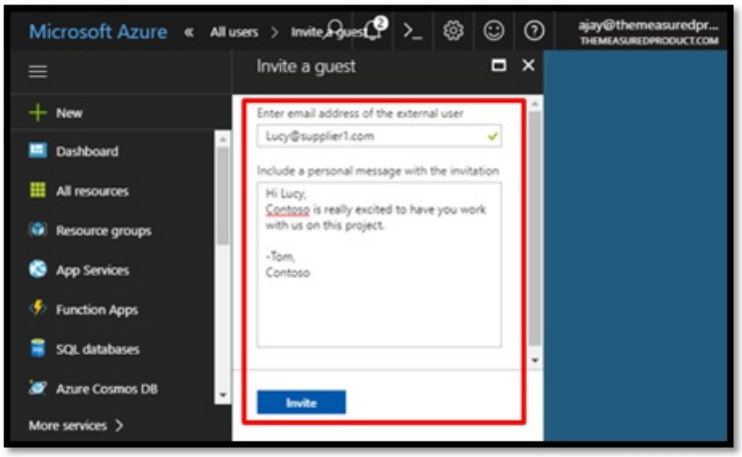
   Planned Invites

   In this approach, Contoso invites the guest users to its Azure AD ahead of time and then distributes Power BI content to them. Contoso can invite guest users from the Azure portal or using PowerShell. Here are the steps to invite guest users from the Azure portal:

   - Contoso's Azure AD administrator navigates to **Azure portal > Azure Active Directory > Users and groups > All users > New guest user**



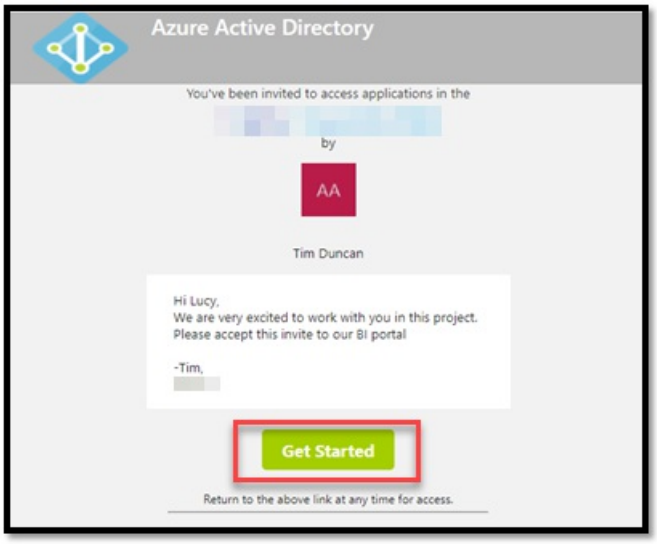
- Add an invitation message for the guest users and click Invite



> **NOTE**
>
> To invite guest users from the Azure portal, you need to an administrator for the Azure Active Directory of your tenant.

If Contoso wants to invite many guest users, they can do so using PowerShell. Contoso's Azure AD administrator stores the email addresses of all the guest users in a CSV file. Here are Azure Active Directory B2B collaboration code and PowerShell samples and instructions.

After the invitation, guest users receive an email with the invitation link.



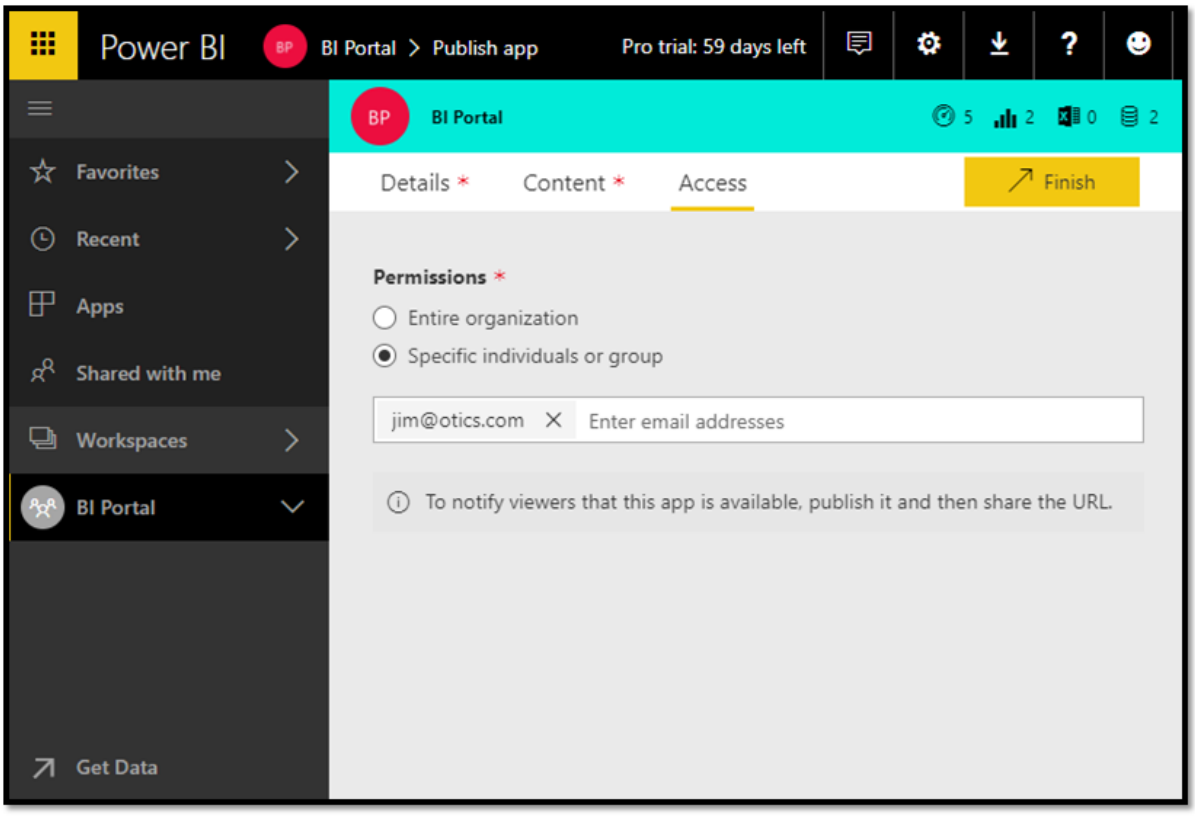
Once the guest users click the link, they can access content in the Contoso Azure AD tenant.

> **NOTE**
> It is possible to change the layout of the invitation email using the Azure AD branding feature as described here.

Ad hoc Invites

What if Contoso does not know all the guest users it wants to invite ahead of time? Or, what if the analyst in Contoso who created the BI portal wants to distribute content to guest users herself? We also support this scenario in Power BI with ad-hoc invites.

The analyst can just add the external users to the access list of the app when they are publishing it. The guest users gets an invite and once they accept it, they are automatically redirected to the Power BI content.



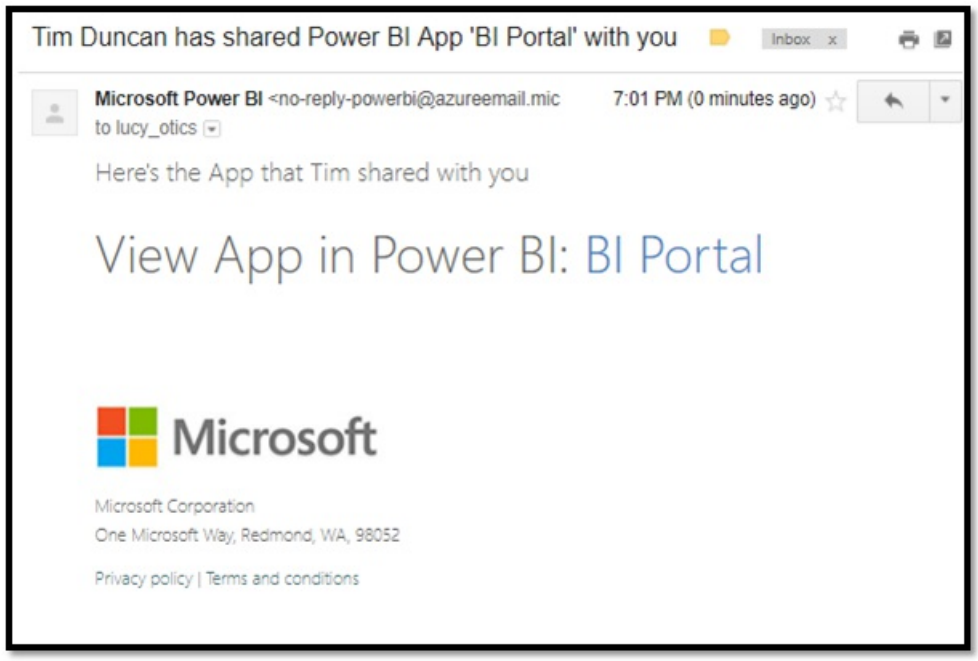
> **NOTE**
>
> Invites are needed only the first time an external user is invited to your organization.

3. Distribute Content

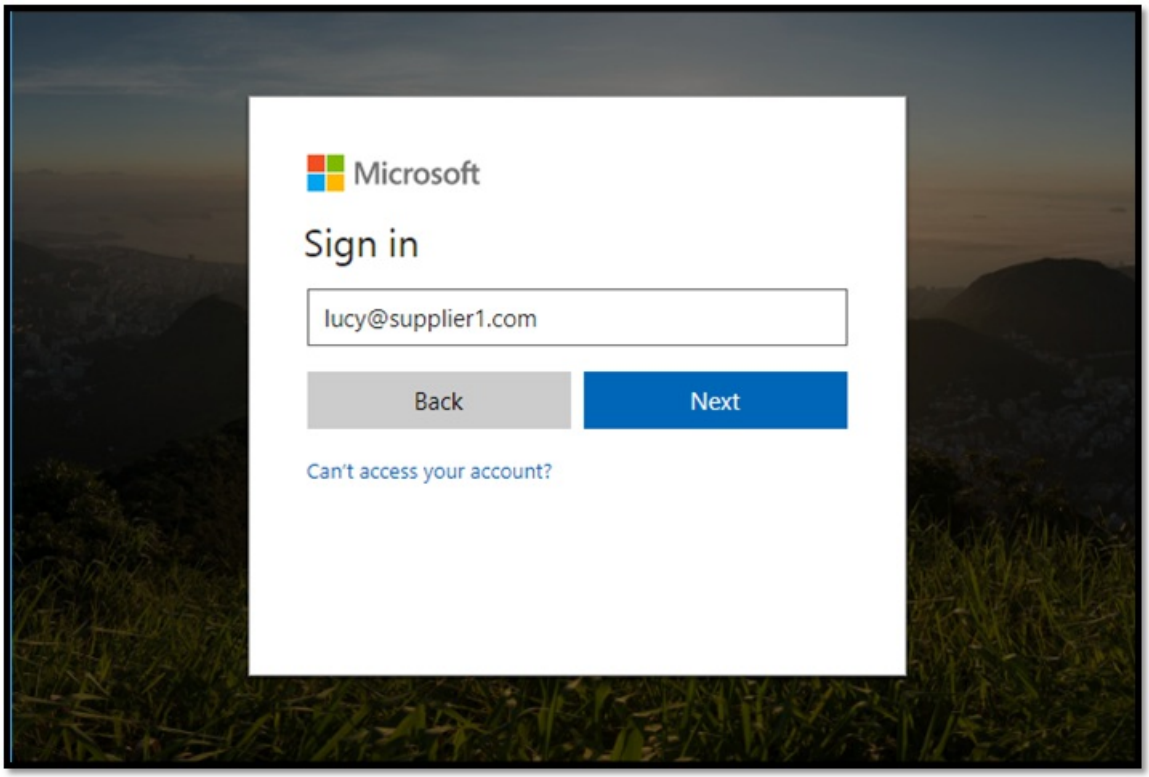
   Now that Contoso's BI team has created the BI portal and invited guest users, they can distribute their portal to their end users by giving guest users access to the app and publishing it. Power BI auto-completes names of guest users who have been previously added to the Contoso tenant. Adhoc invitations to other guest users can also be added at this point.

   > **NOTE**
   >
   > If using Security groups to manage access to the app for external users, use the Planned Invites approach and share the app link directly with each external user who must access it. Otherwise, the external user may not be able to install or view content from within the app._

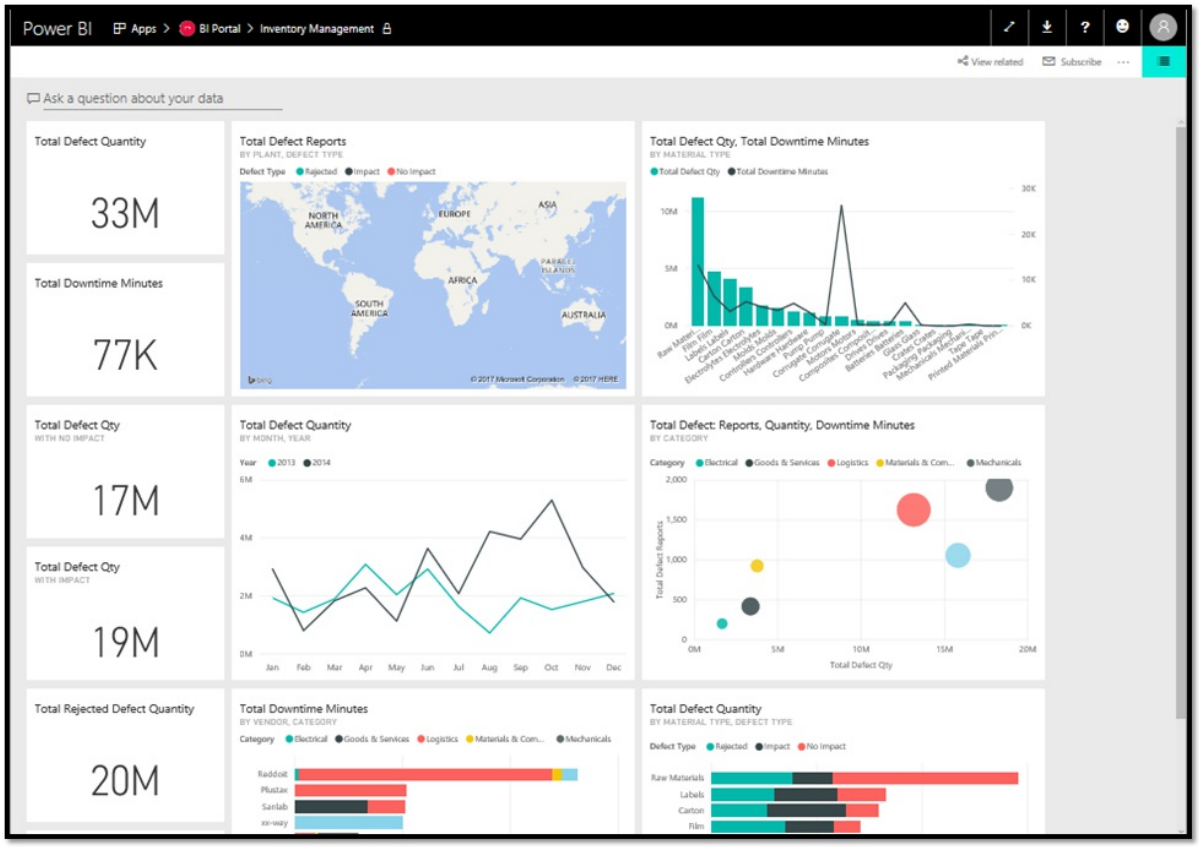
   Guest users get an email with a link to the app.



   On clicking this link, guest users are asked to authenticate with their own organization's identity.



Once they are successfully authenticated, they are redirected to Contoso's BI app.



Guest users can subsequently get to Contoso's app by clicking the link in the email or bookmarking the link. Contoso can also make it easier for guest users by adding this link to any existing extranet portal that the guest users already use.

4. Next steps

   Using a Power BI app and Azure AD B2B, Contoso was able to quickly create a BI Portal for its suppliers in a no-code way. This greatly simplified distributing standardized analytics to all the suppliers who needed it.

While the example showed how a single common report could be distributed among suppliers, Power BI can go much further. To ensure each partner sees only data relevant to themselves, Row Level Security can be added easily to the report and data model. The Data security for external partners section later in this document describes this process in details.

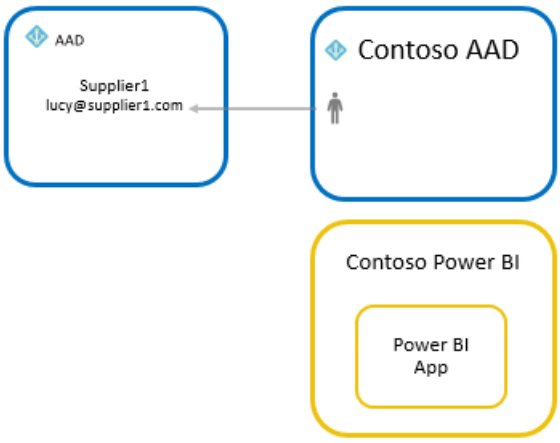
Often individual reports and dashboards need to be embedded into an existing portal. This can also be accomplished reusing many of the techniques shown in the example. However, in those situations it may be easier to embed reports or dashboards directly from a workspace. The process for inviting and assigning security permission to the require users remain the same.

Under the hood: How is Lucy from Supplier1 able to access Power BI content from Contoso's tenant?

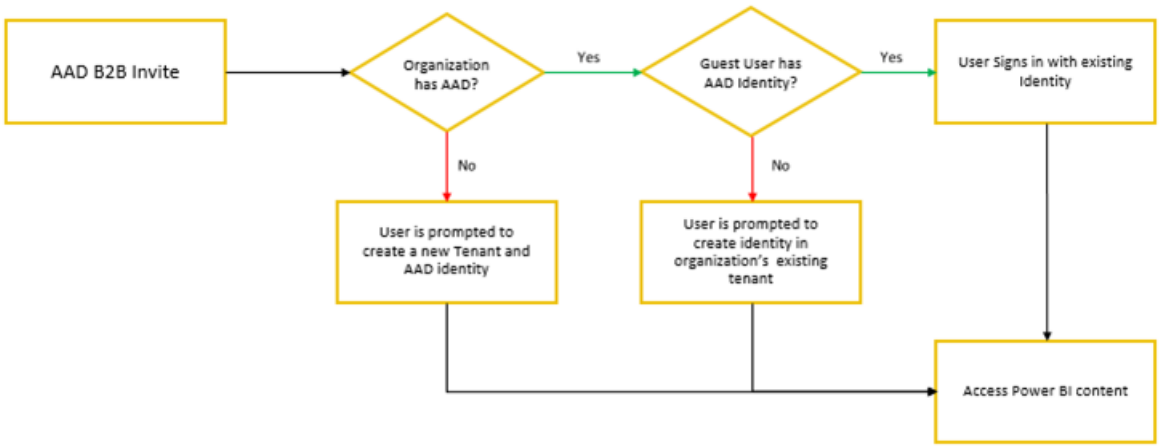
Now that we have seen how Contoso is able to seamlessly distribute Power BI content to guest users in partner organizations, let's look at how this works under the hood.

When Contoso invited lucy@supplier1.com to its directory, Azure AD creates a link between Lucy@supplier1.com and the Contoso Azure AD tenant. This link lets Azure AD know that Lucy@supplier1.com can access content in the Contoso tenant.

When Lucy tries to access Contoso's Power BI app, Azure AD verifies that Lucy can access the Contoso tenant and then provides Power BI a token that indicates that Lucy is authenticated to access content in the Contoso tenant. Power BI uses this token to authorize and ensure that Lucy has access to Contoso's Power BI app.



Power BI's integration with Azure AD B2B works with all business email addresses. If the user does not have an Azure AD identity, they may be prompted to create one. The following image shows the detailed flow:



It is important to recognize that the Azure AD account will be used or created in the external party's Azure AD, this will make it possible for Lucy to use their own username and password and their credentials will automatically stop working in other tenants whenever Lucy leaves the company when their organization also uses Azure AD.

Licensing

Contoso can choose one of three approaches to license guest users from its suppliers and partner organizations to have access to Power BI content.

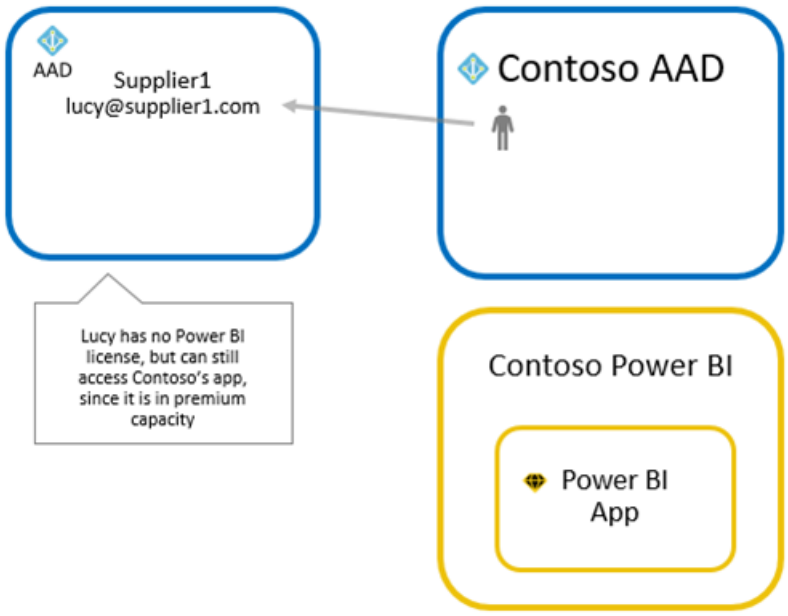
> **NOTE**
>
> *The Azure AD B2B's free tier is enough to use Power BI with Azure AD B2B. Some advanced Azure AD B2B features like dynamic groups require additional licensing. Please refer to the Azure AD B2B documentation for additional information: https://docs.microsoft.com/azure/active-directory/b2b/licensing-guidance*

Approach 1: Contoso uses Power BI Premium

With this approach, Contoso purchases Power BI Premium capacity and assigns its BI portal content to this capacity. This allows guest users from partner organizations to access Contoso's Power BI app without any Power BI license.

External users are also subject to the consumption only experiences offered to "Free" users in Power BI when consuming content within Power BI Premium.

Contoso can also take advantage of other Power BI premium capabilities for its apps like increased refresh rates, dedicated capacity, and large model sizes.

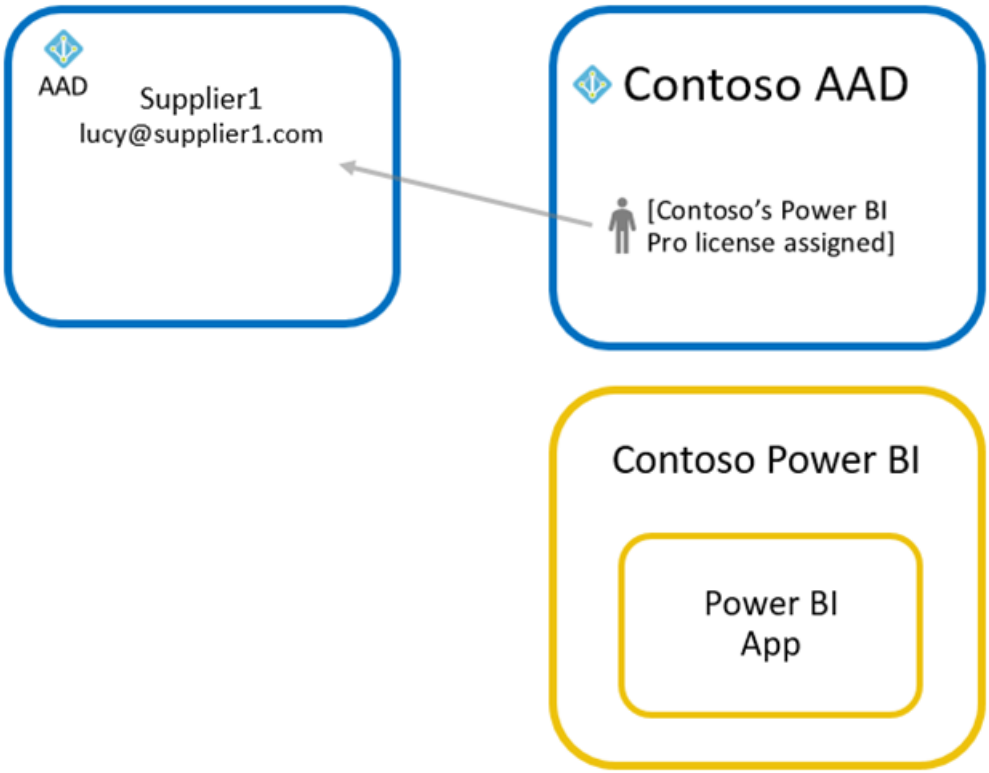


Approach 2: Contoso assigns Power BI Pro licenses to guest users

With this approach, Contoso assigns pro licenses to guest users from partner organizations – this can be done from Contoso's Microsoft 365 admin center. This allows guest users from partner organizations to access Contoso's Power BI app without purchasing a license themselves. This can be appropriate for sharing with external users whose organization has not adopted Power BI yet.

NOTE

Contoso's pro license applies to guest users only when they access content in the Contoso tenant. Pro licenses enable access to content that is not in a Power BI Premium capacity. However, external users with a Pro license are restricted by default to a consumption only experience. This can be changed by using the approach described in the *Enabling external users to edit and manage content within Power BI* section later in this document.

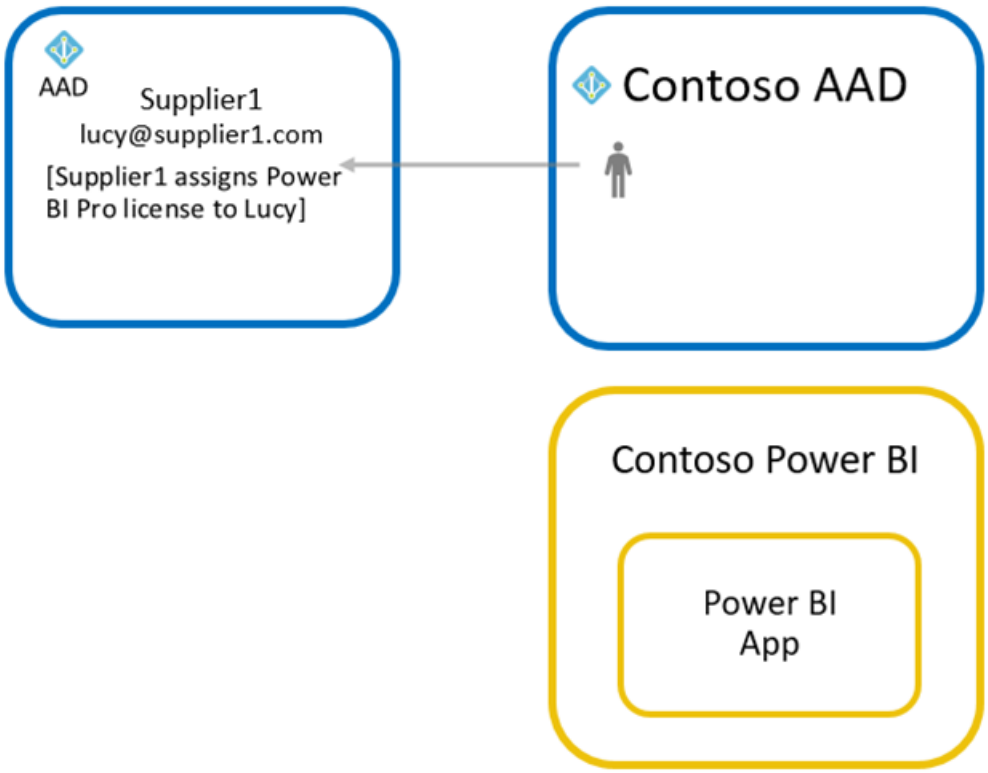


Approach 3: Guest users bring their own Power BI Pro license

With this approach, Supplier 1 assigns a Power BI Pro license to Lucy. They can then access Contoso's Power BI app with this license. Since Lucy can use their Pro license from their own organization when accessing an external Power BI environment, this approach is sometimes referred to as *bring your own license* (BYOL). If both organizations are using Power BI, this offers advantageous licensing for the overall analytics solution and minimizes overhead of assigning licenses to external users.

NOTE

The pro license given to Lucy by Supplier 1 applies to any Power BI tenant where Lucy is a guest user. Pro licenses enable access to content that is not in a Power BI Premium capacity. However, external users with a Pro license are restricted by default to a consumption only experience. This can be change by using the approach described in the *Enabling external users to edit and manage content within Power BI* section later in this document.



Data security for external partners

Commonly when working with multiple external suppliers, Contoso needs to ensure that each supplier sees data only about its own products. User-based security and dynamic row level security make this easy to accomplish with Power BI.

User-based security

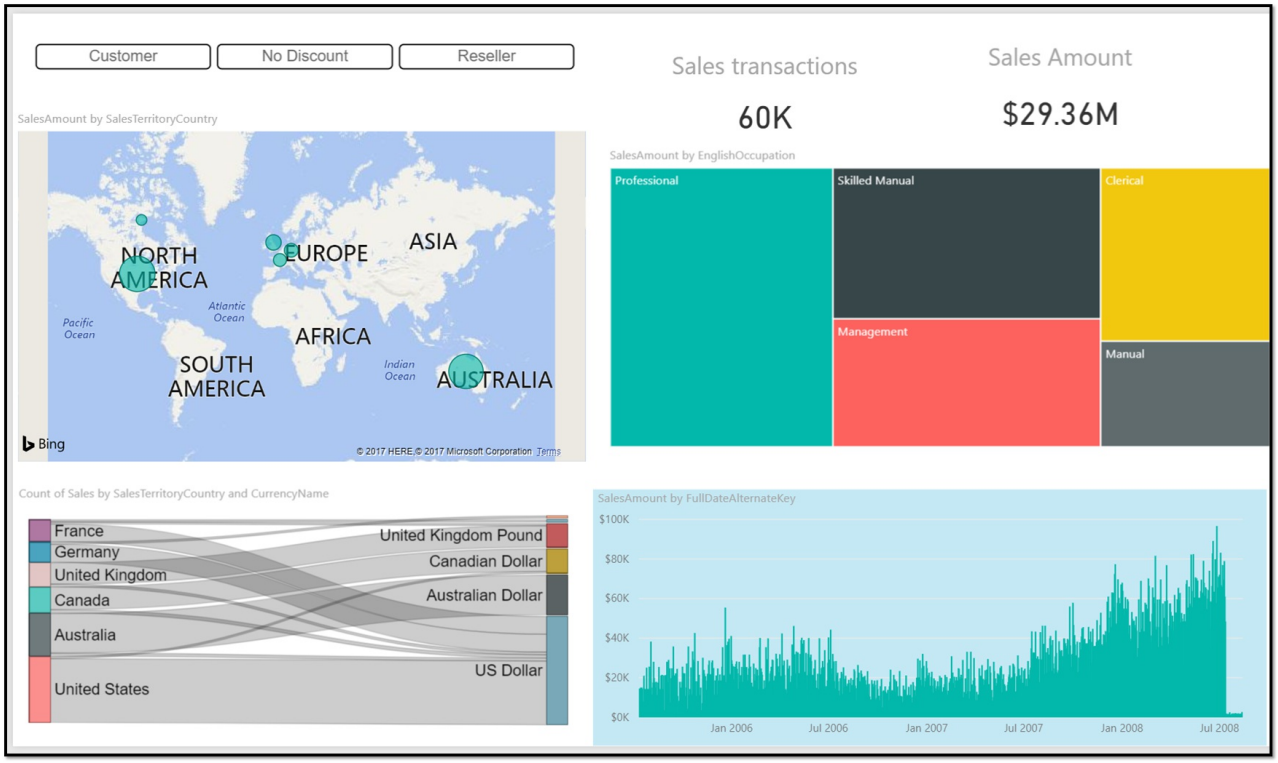
One of the most powerful features of Power BI is Row Level Security. This feature allows Contoso to create a single report and dataset but still apply different security rules for each user. For an in-depth explanation, see Row-level security (RLS).

Power BI's integration with Azure AD B2B allows Contoso to assign Row Level Security rules to guest users as soon as they are invited to the Contoso tenant. As we have seen before, Contoso can add guest users through either planned or ad-hoc invites. If Contoso wants to enforce row level security, it is strongly recommended to use planned invites to add the guest users ahead of time and assigning them to the security roles before sharing the content. If Contoso instead uses ad-hoc invites, there might be a short period of time where the guest users will not be able to see any data.

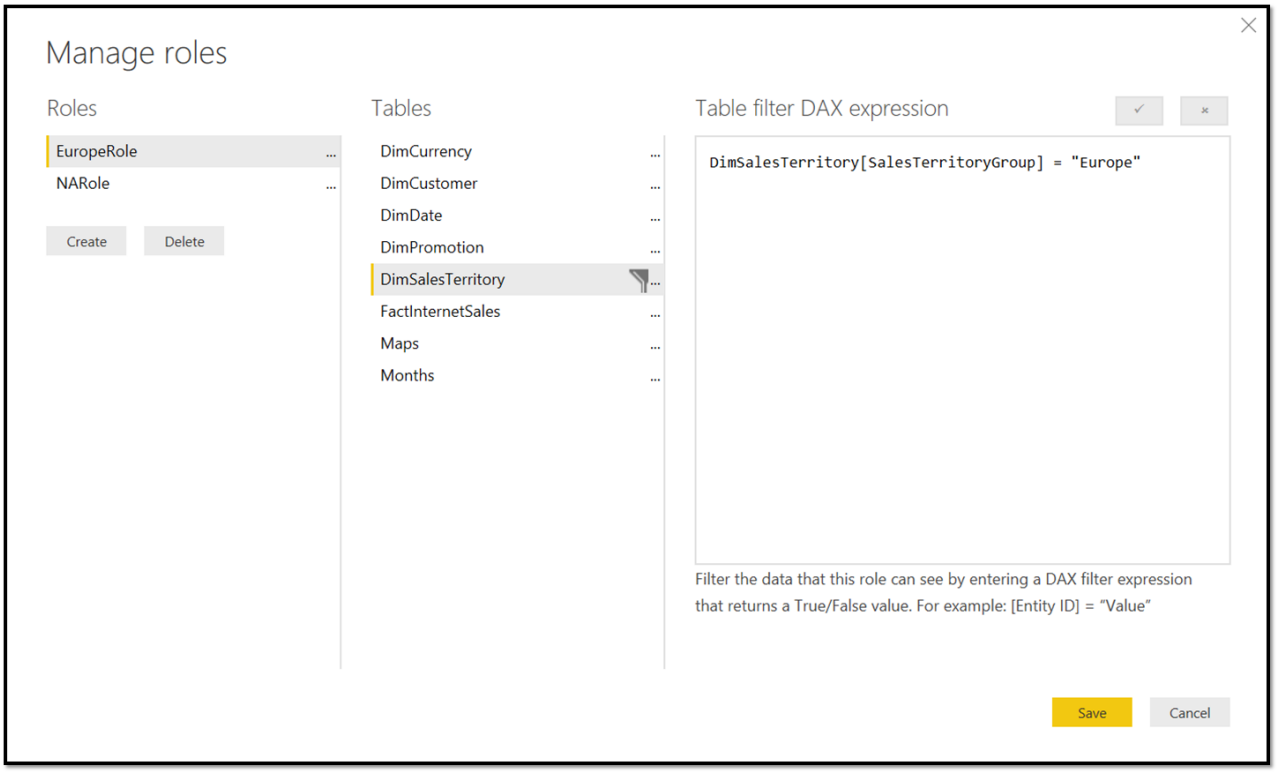
> **NOTE**
> This delay in accessing data protected by RLS when using ad-hoc invites can lead to support requests to your IT team because users will see either blank or broken looking reports/dashboards when opening a sharing link in the email they receive. Therefore, it is strongly recommended to use planned invites in this scenario.**

Let's walk through this with an example.

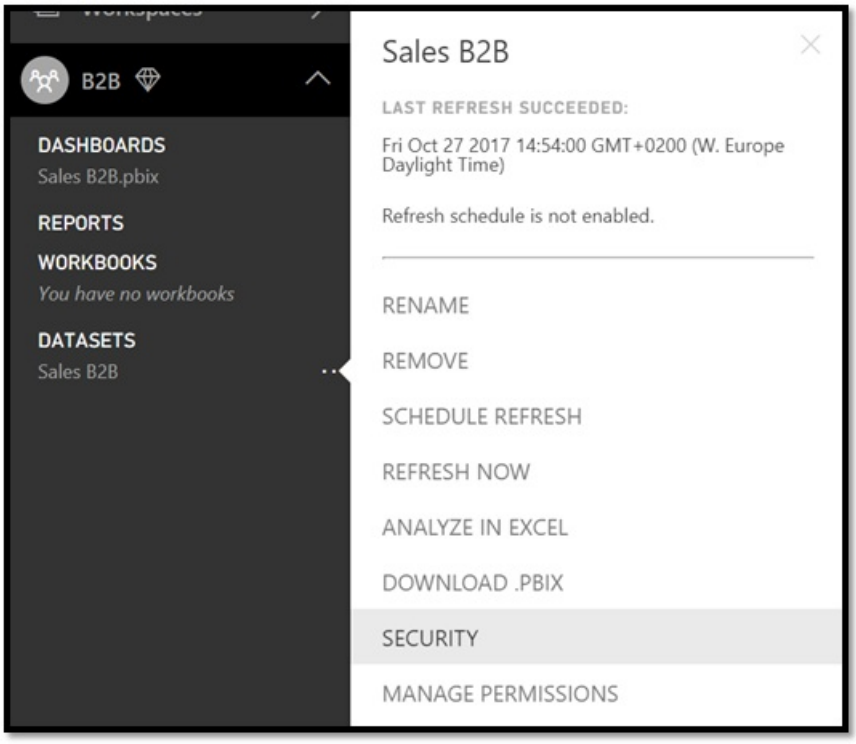
As mentioned before, Contoso has suppliers around the globe, and they want to make sure that the users from their supplier organizations get insights from data from just their territory. But users from Contoso can access all the data. Instead of creating several different reports, Contoso creates a single report and filters the data based the user viewing it.



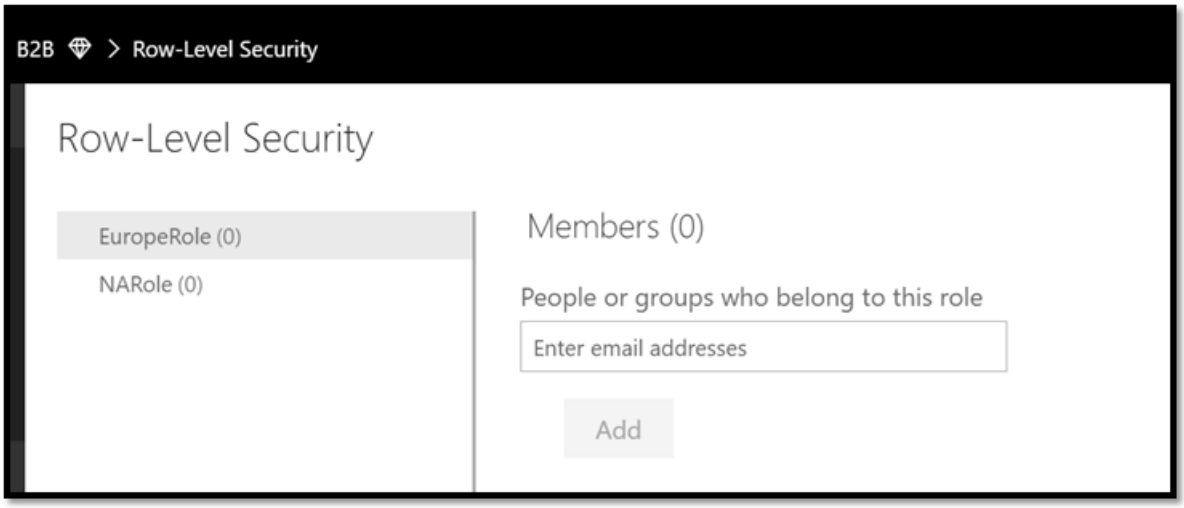
To make sure Contoso can filter data based on who is connecting, two roles are created in Power BI desktop. One to filter all the data from the SalesTerritory "Europe" and another for "North America".



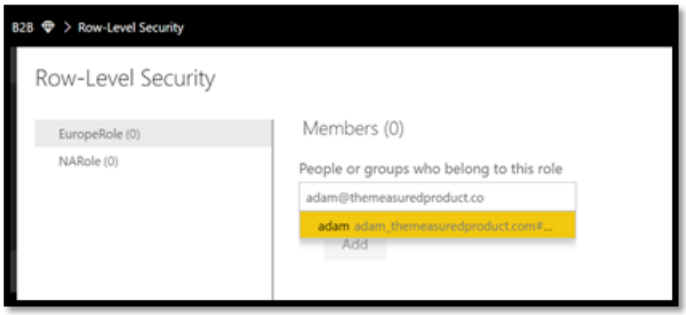
Whenever roles are defined in the report, a user must be assigned to a specific role for them to get access to any data. The assignment of roles happens inside the Power BI service (**Datasets > Security**)



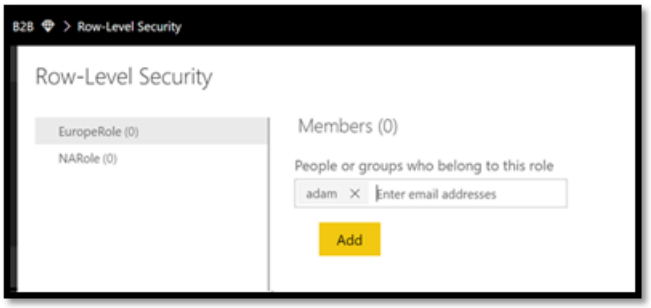
This opens a page where Contoso's BI team can see the two roles they created. Now Contoso's BI team can assign users to the roles.



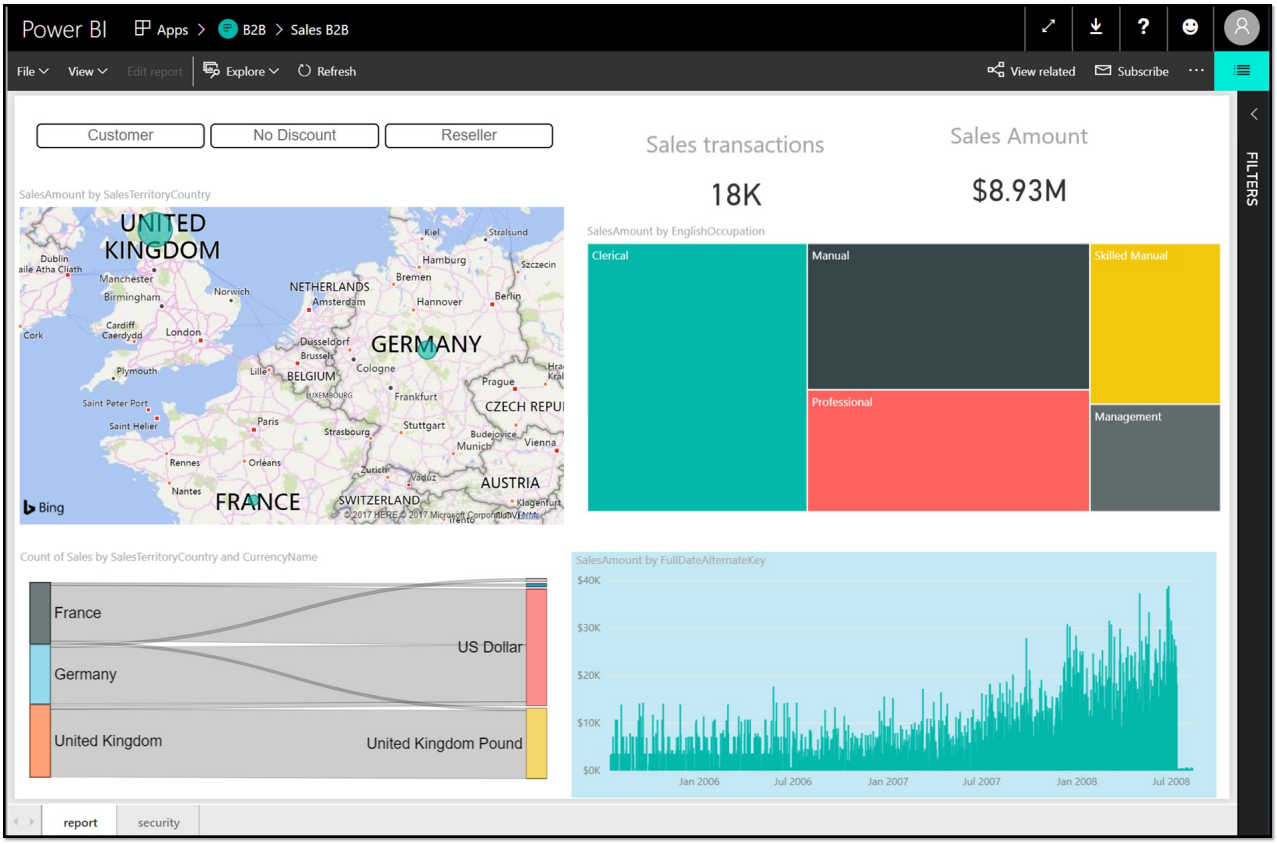
In the example Contoso is adding a user in a partner organization with email address "adam@themeasuredproduct.com" to the Europe role:



When this gets resolved by Azure AD, Contoso can see the name show up in the window ready to be added:



Now when this user opens the app that was shared with them, they only see a report with data from Europe:



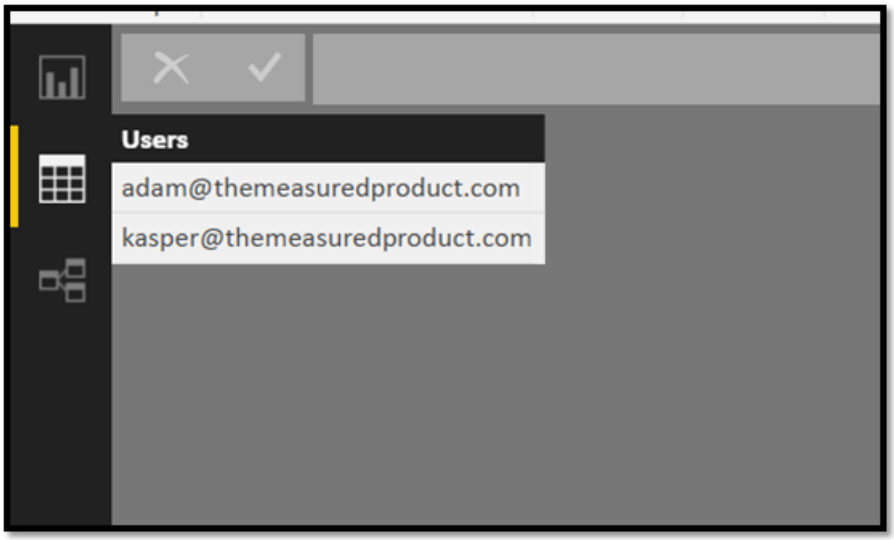
Dynamic row level security

Another interesting topic is to see how dynamic row level security (RLS) work with Azure AD B2B.

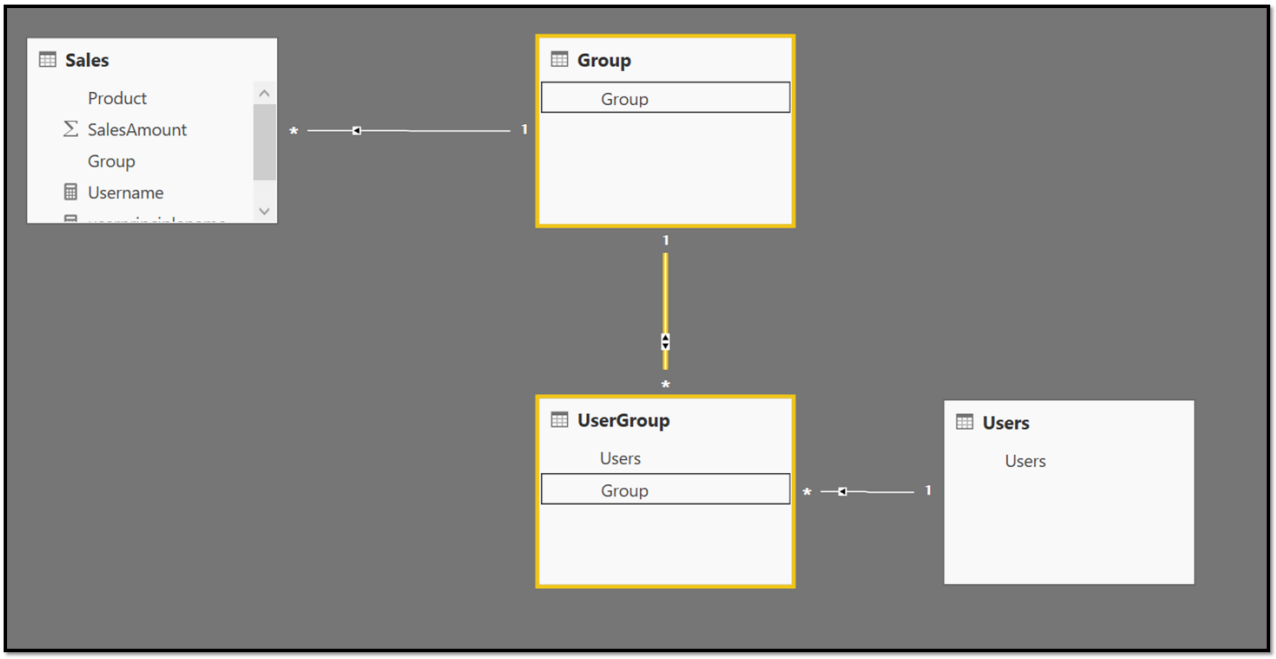
In short, Dynamic row level security works by filtering data in the model based on the username of the person connecting to Power BI. Instead of adding multiple roles for groups of users, you define the users in the model. We won't describe the pattern in detail here. Kasper de Jong offers a detailed write up on all the flavors of row level security in Power BI Desktop Dynamic security cheat sheet, and in this whitepaper .

Let's look at a small example - Contoso has a simple report on sales by groups:

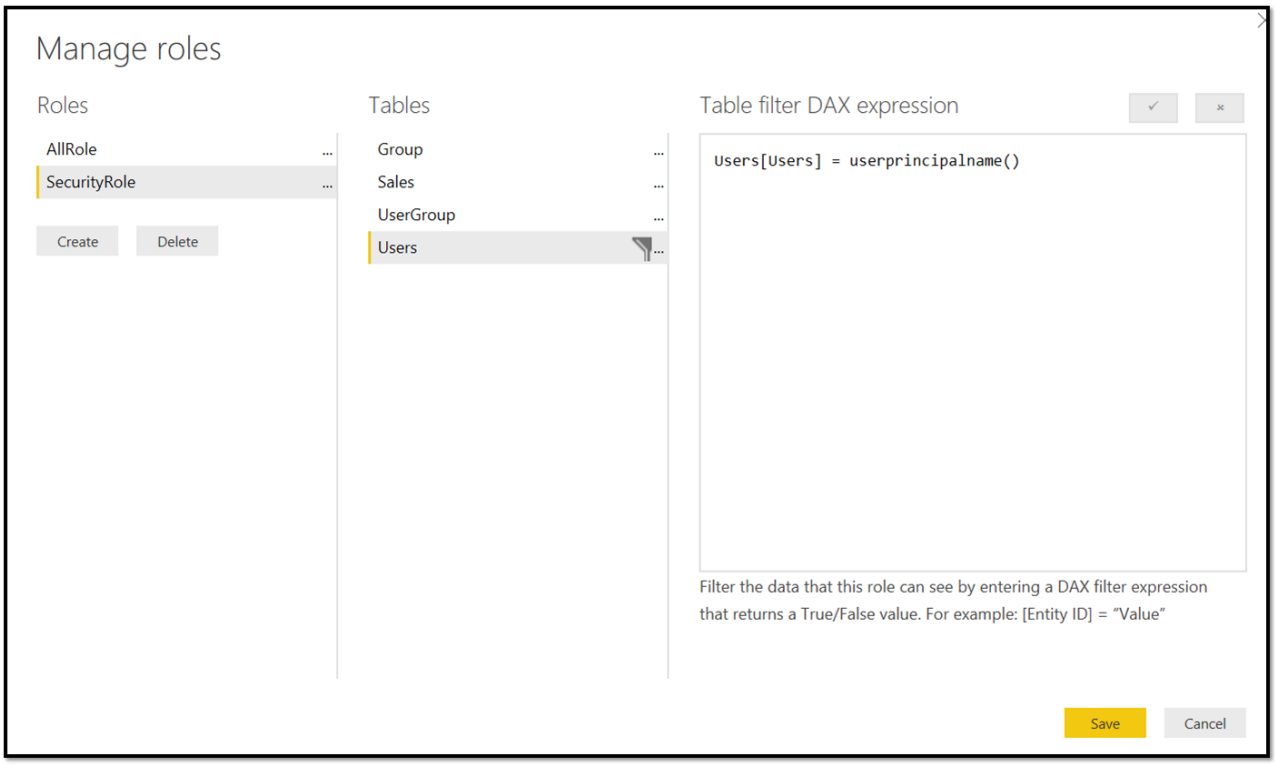
Now this report needs to be shared with two guest users and an internal user - the internal user can see everything, but the guest users can only see the groups they have access to. This means we must filter the data only for the guest users. To filter the data appropriately, Contoso uses the Dynamic RLS pattern as described in the whitepaper and blog post. This means, Contoso adds the usernames to the data itself:



Then, Contoso creates the right data model that filters the data appropriately with the right relationships:



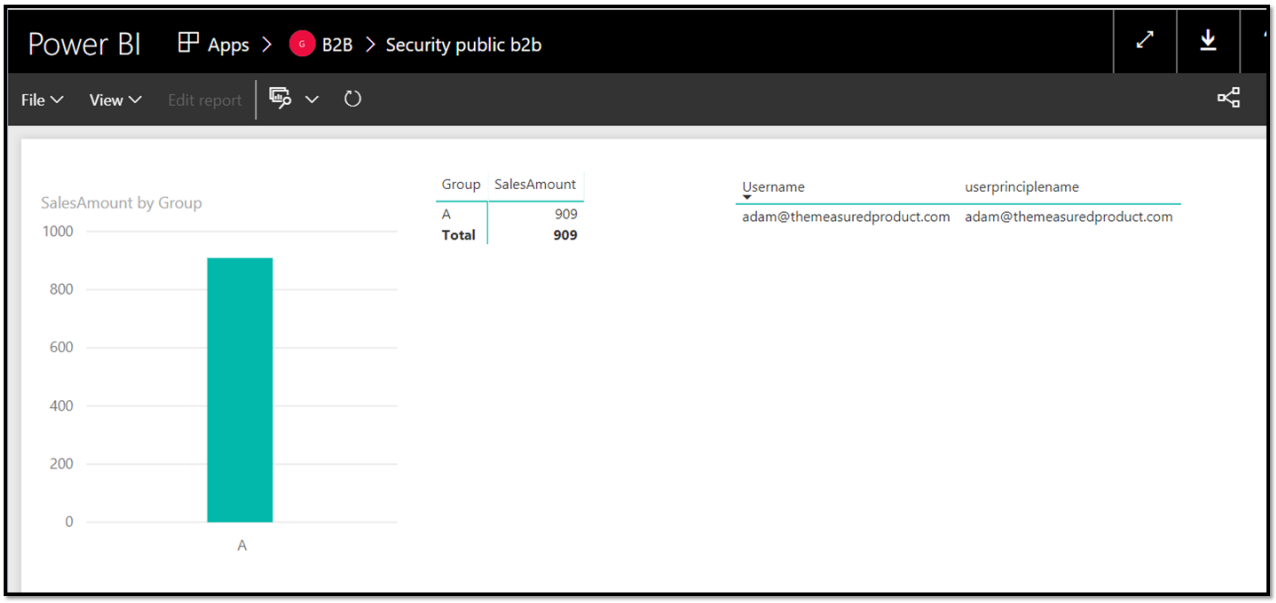
To filter the data automatically based on who is logged in, Contoso needs to create a role that passes in the user who is connecting. In this case, Contoso creates two roles – the first is the "securityrole" that filters the Users table with the current username of the user logged in to Power BI (this works even for Azure AD B2B guest users).



Contoso also creates another "AllRole" for its internal users who can see everything – this role does not have any security predicate.

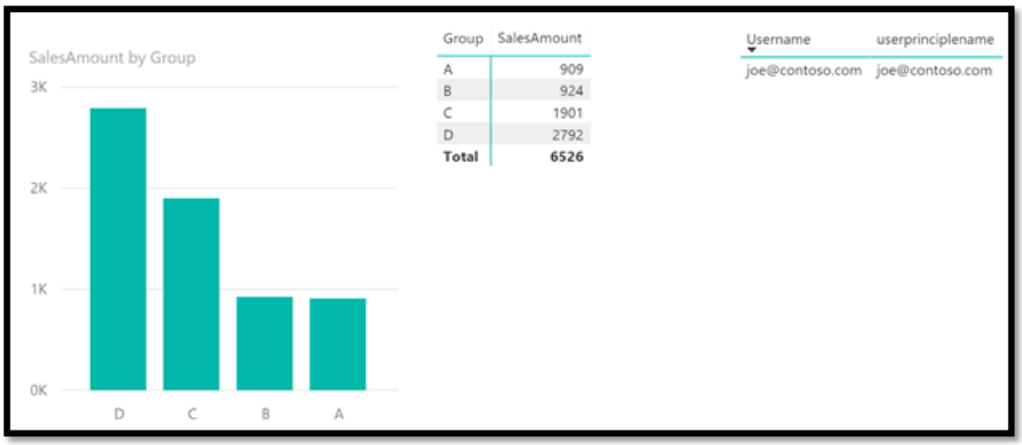
After uploading the Power BI desktop file to the service, Contoso can assign guest users to the "SecurityRole" and internal users to the "AllRole"

Now, when the guest users open the report, they only see sales from group A:



In the matrix to the right you can see the result of the USERNAME() and USERPRINCIPALNAME() function both return the guest users email address.

Now the internal user gets to see all the data:



As you can see, Dynamic RLS works with both internal or guest users.

> **NOTE**
>
> This scenario also works when using a model in Azure Analysis Services. Usually your Azure Analysis Service is connected to the same Azure AD as your Power BI - in that case, Azure Analysis Services also knows the guest users invited through Azure AD B2B.

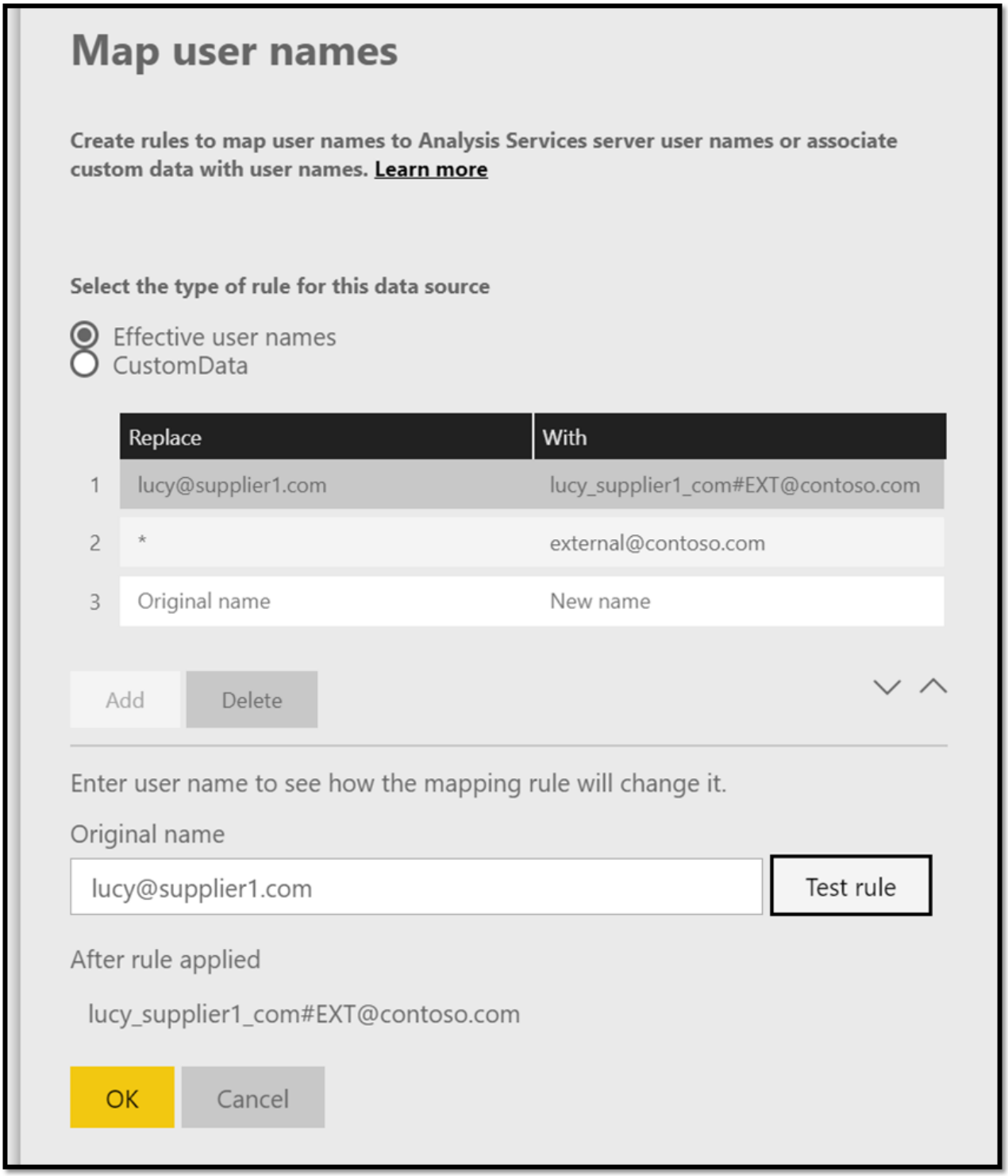
Connecting to on premises data sources

Power BI offers the capability for Contoso to leverage on premises data sources like SQL Server Analysis Services or SQL Server directly thanks to the On-Premises data gateway. It is even possible to sign on to those data sources with the same credentials as used with Power BI.

> **NOTE**
>
> When installing a gateway to connect to your Power BI tenant, you must use a user created within your tenant. External users cannot install a gateway and connect it to your tenant._

For external users, this might be more complicated as the external users are usually not known to the on-premises

AD. Power BI offers a workaround for this by allowing Contoso administrators to map the external usernames to internal usernames as described in Manage your data source - Analysis Services. For example, lucy@supplier1.com can be mapped to lucy_supplier1_com#EXT@contoso.com.



This method is fine if Contoso only has a handful of users or if Contoso can map all the external users to a single internal account. For more complex scenarios where each user needs their own credentials, there is a more advanced approach that uses custom AD attributes to do the mapping as described in Manage your data source - Analysis Services. This would allow the Contoso administrator to define a mapping for every user in your Azure AD (also external B2B users). These attributes can be set through the AD object model using scripts or code so Contoso can fully automate the mapping on invite or on a scheduled cadence.

Enabling external users to edit and manage content within Power BI

Contoso can allow external users to contribute content within the organization as described earlier in the cross-organization editing and management of Power BI content section.

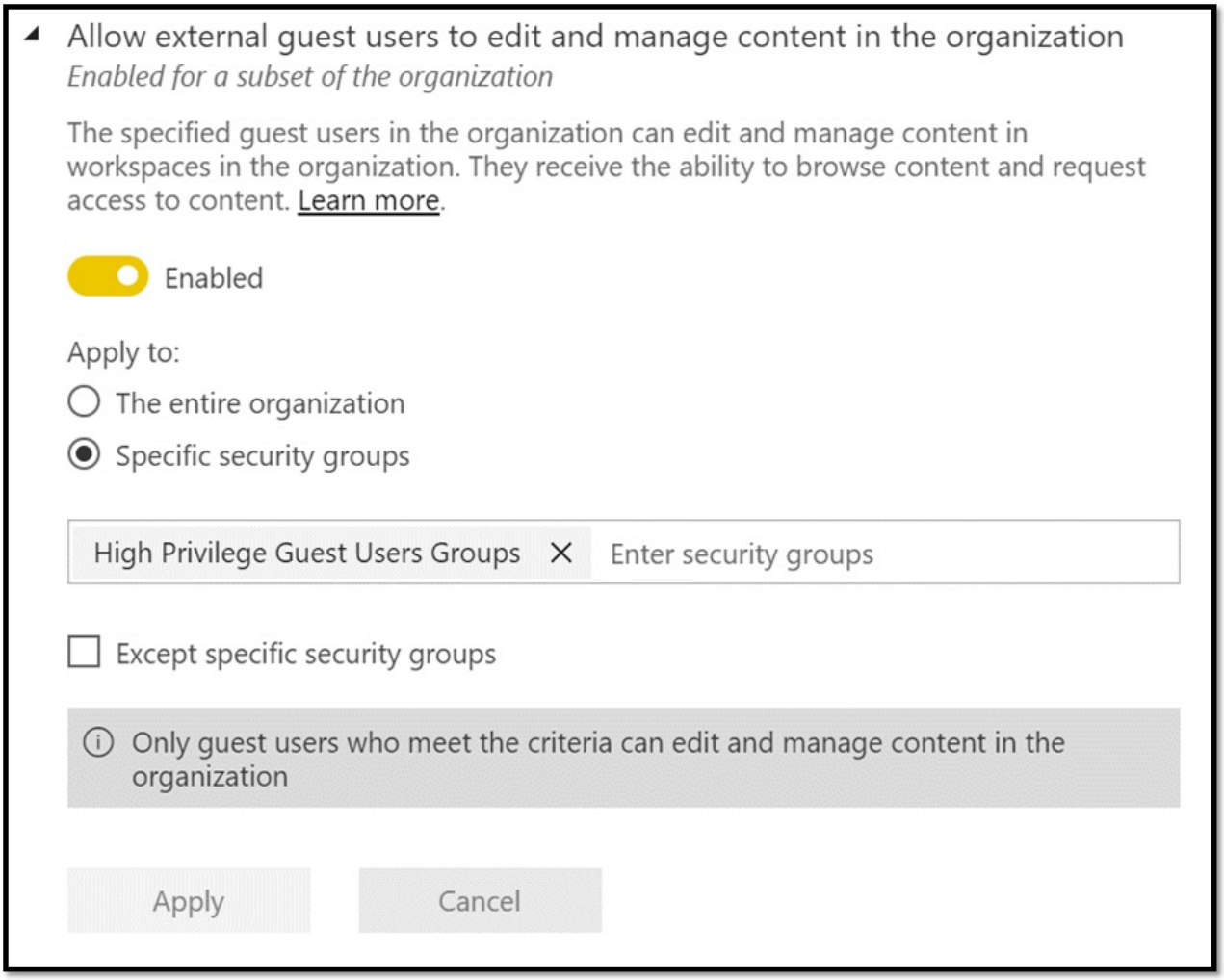
> **NOTE**
>
> To edit and manage content within your organization's Power BI, the user must have a Power BI Pro license in a workspace other than My workspace. Users can obtain Pro licenses as covered in the *Licensing* section of this document.

The Power BI Admin Portal provides the **allow external guest users to edit and manage content in the organization** setting in Tenant settings. By default, the setting is set to disabled, meaning external users get a constrained read-only experience by default. The setting applies to users with UserType set to Guest in Azure AD. The table below describes the behaviors users experience depending on their UserType and how the settings are configured.

USER TYPE IN AZURE AD	ALLOW EXTERNAL GUEST USERS TO EDIT AND MANAGE CONTENT SETTING	BEHAVIOR
Guest	Disabled for the user (Default)	Per item consumption only view. Allows read-only access to reports, dashboards, and apps when viewed through a URL sent to the Guest user. Power BI Mobile apps provide a read-only view to the guest user.
Guest	Enabled for the user	The external user gets access to the full Power BI experience, though some features are not available to them. The external user must log in to Power BI using the Power BI Service URL with the tenant information included. The user gets the Home experience, a My Workspace, and based on permissions can browse, view, and create content. Power BI Mobile apps provide a read-only view to the guest user.

> **NOTE**
>
> External users in Azure AD can also be set to UserType Member. This is not currently supported in Power BI.

In the Power BI Admin portal, the setting is shown in the following image.



Guest users get the read-only default experience and which can edit and manage content. The default is Disabled, meaning all Guest users have the read-only experience. The Power BI Admin can either enable the setting for all Guest users in the organization or for specific security groups defined in Azure AD. In the following image, the Contoso Power BI Admin created a security group in Azure AD to manage which external users can edit and manage content in the Contoso tenant.

To help these users to log in to Power BI, provide them with the Tenant URL. To find the tenant URL, follow these steps.

1. In the Power BI service, in the top menu, select help (**?**) then **About Power BI**.
2. Look for the value next to **Tenant URL**. This is the tenant URL you can share with your guest users.

When using the Allow external guest users to edit and manage content in the organization, the specified guest users get access to your organization's Power BI and see any content to which they have permission. They can

access Home, browse and contribute content to workspaces, install apps where they are on the access list, and have a My workspace. They can create or be an Admin of workspaces that use the new workspace experience.

> **NOTE**
> When using this option make sure to review the governance section of this document since default Azure AD settings prevent Guest users to use certain features like people pickers which can lead to a reduced experience.**

For guest users enabled through the Allow external guest users to edit and manage content in the organization tenant setting, some experiences are not available to them. To update or publish reports, guest users need to use the Power BI service web UI, including Get Data to upload Power BI Desktop files. The following experiences are not supported:

- Direct publishing from Power BI desktop to the Power BI service
- Guest users cannot use Power BI desktop to connect to service datasets in the Power BI service
- Classic workspaces tied to Microsoft 365 Groups: Guest user cannot create or be Admins of these workspaces. They can be members.
- Sending ad-hoc invites is not supported for workspace access lists
- Power BI Publisher for Excel is not supported for guest users
- Guest users cannot install a Power BI Gateway and connect it to your organization
- Guest users cannot install apps publish to the entire organization
- Guest users cannot use, create, update, or install organizational content packs
- Guest users cannot use Analyze in Excel
- Guest users cannot be @mentioned in commenting (this functionality will be added in an upcoming release)
- Guest users cannot use subscriptions (this functionality will be added in an upcoming release)
- Guest users who use this capability should have a work or school account. Guest users using Personal accounts experience more limitations due to sign-in restrictions.

Governance

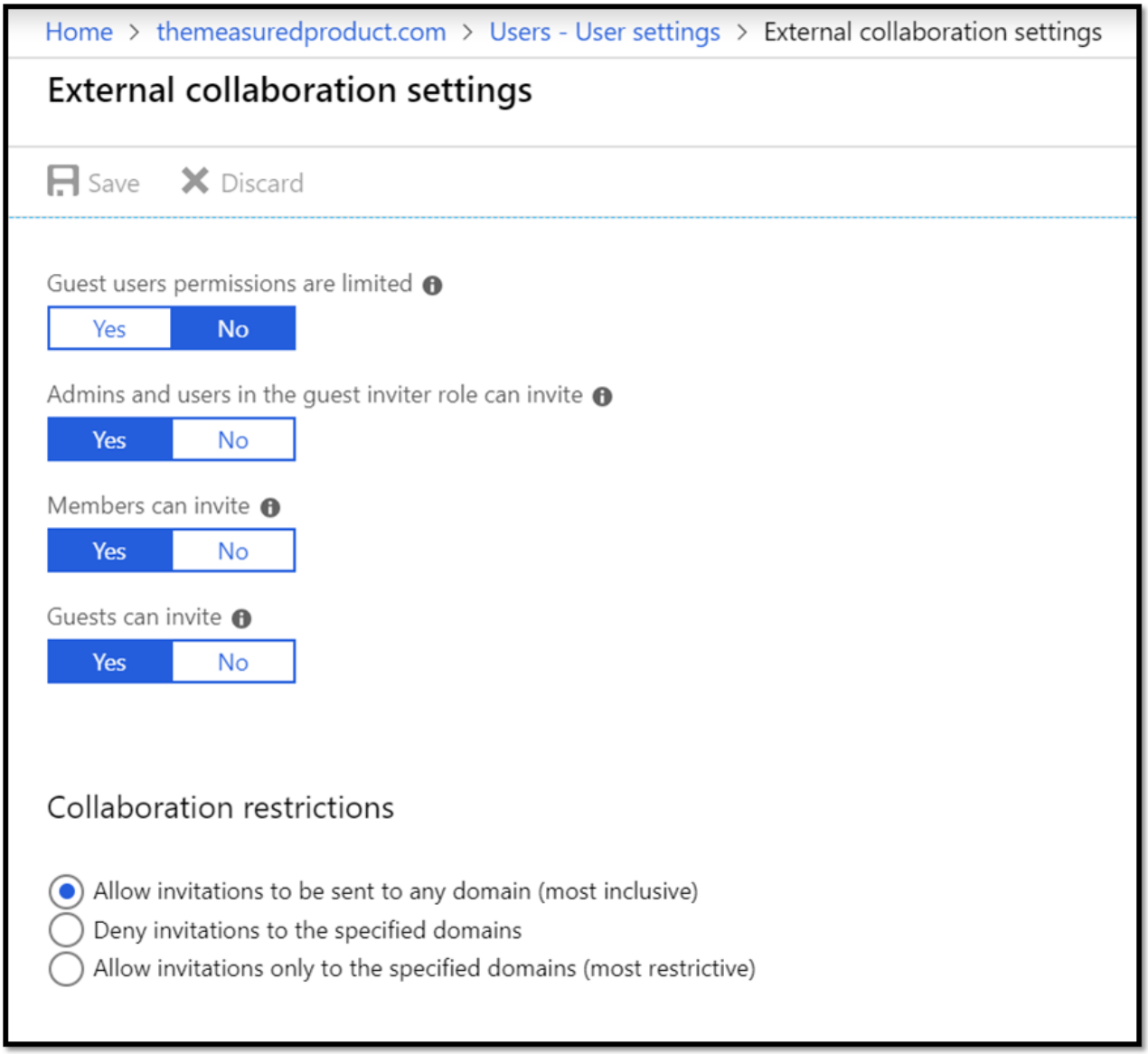
Additional Azure AD Settings that affect experiences in Power BI related to Azure AD B2B

When using Azure AD B2B sharing, the Azure Active Directory administrator controls aspects of the external user's experience. These are controlled on the External collaboration settings page within the Azure Active Directory settings for your Tenant.

Details on the settings are available here:

https://docs.microsoft.com/azure/active-directory/b2b/delegate-invitations

> **NOTE**
> By default, the Guest users permissions are limited option is set to Yes, so Guest users within Power BI have limited experiences especially surround sharing where people picker UIs do not work for those users. It is important to work with your Azure AD administrator to set it to No, as shown below to ensure a good experience.**



Control guest invites

Power BI administrators can control external sharing just for Power BI by visiting the Power BI admin portal. But tenant administrators can also control external sharing with various Azure AD policies. These policies allow tenant administrators to

- Turn off invitations by end users
- Only admins and users in the Guest Inviter role can invite
- Admins, the Guest Inviter role, and members can invite
- All users, including guests, can invite

You can read more about these policies in Delegate invitations for Azure Active Directory B2B collaboration.

All Power BI actions by external users are also audited in our auditing portal.

Conditional Access policies for guest users

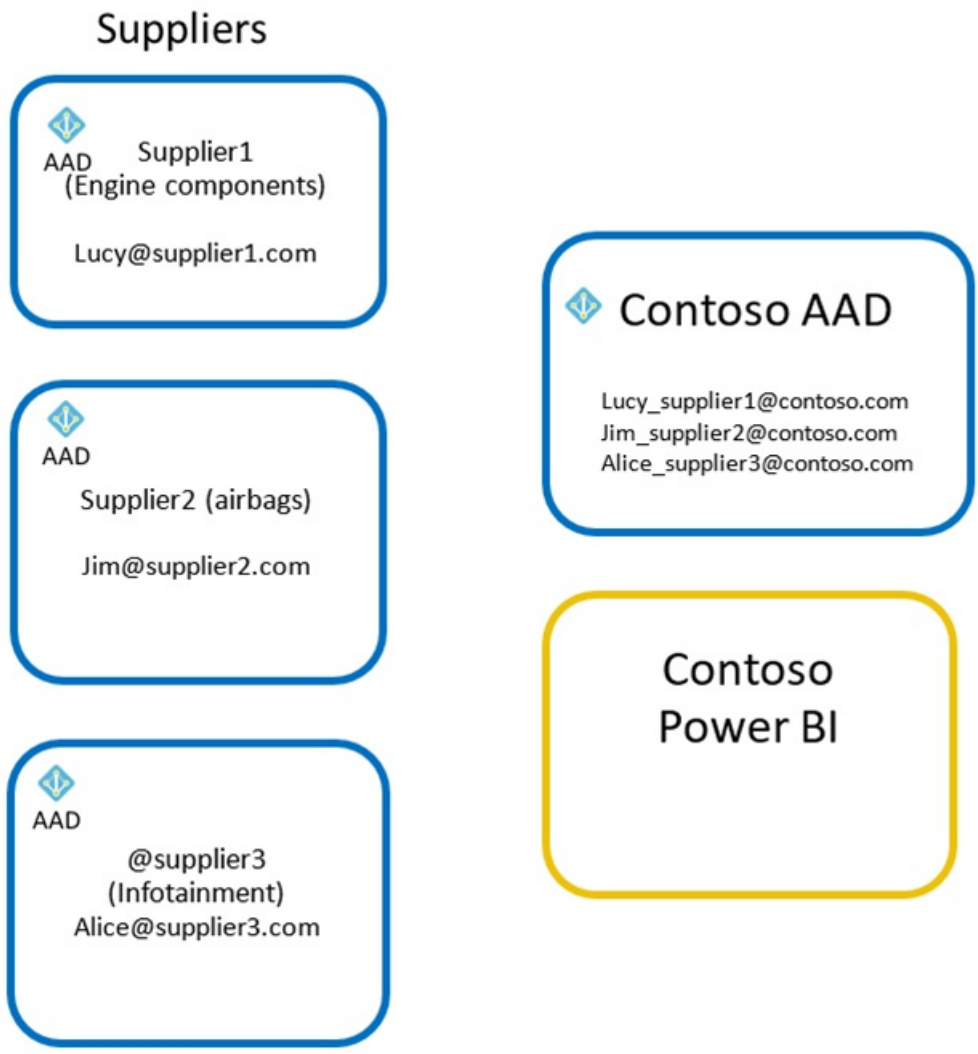
Contoso can enforce conditional access policies for guest users who access content from the Contoso tenant. You can find detailed instructions in Conditional access for B2B collaboration users.

Common alternative approaches

While Azure AD B2B makes it easy to share data and reports across organizations, there are several other approaches that are commonly used and may be superior in certain cases.

Alternative Option 1: Create duplicate identities for partner users

With this option, Contoso had to manually create duplicate identities for each partner user in the Contoso Tenant, as shown in the following image. Then within Power BI, Contoso can share to the assigned identities the appropriate reports, dashboards, or apps.



Reasons to choose this alternative:

- Since the user's identity is controlled by your organization, any related service such as email, SharePoint, etc. are also within the control of your organization. Your IT Administrators can reset passwords, disable access to accounts, or audit activities in these services.
- Users who use personal accounts for their business often are restricted from accessing certain services so may need an organizational account.
- Some services only work over your organization's users. For example, using Intune to manage content on the personal/mobile devices of external users using Azure B2B may not be possible.

Reasons not to choose this alternative:

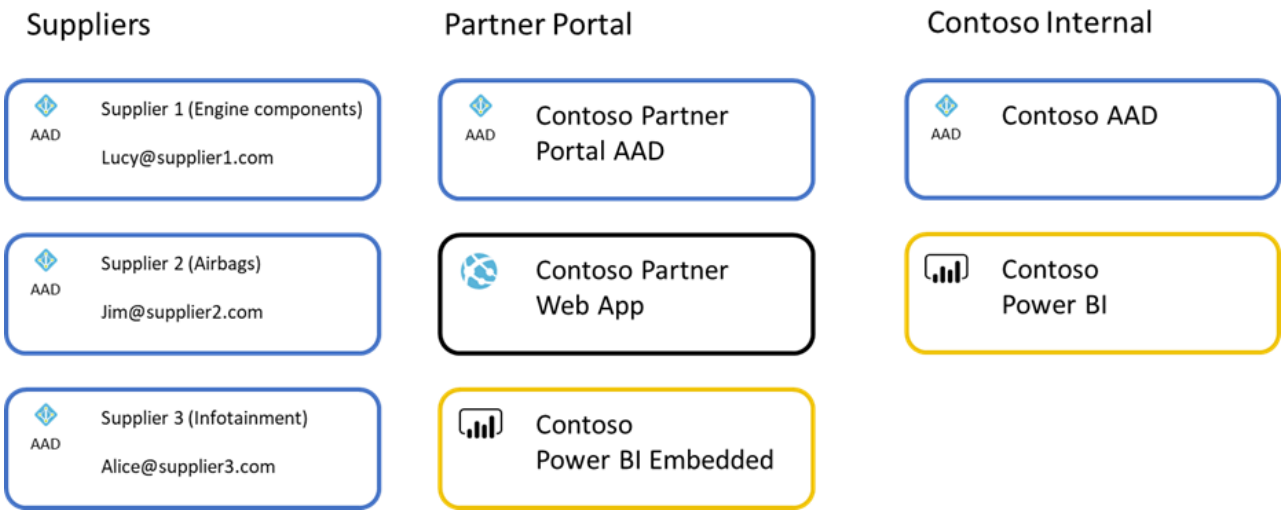
- Users from partner organizations must remember two sets of credentials– one to access content from their own organization and the other to access content from Contoso. This is a hassle for these guest users and many guest users are confused by this experience.
- Contoso must purchase and assign per-user licenses to these users. If a user needs to receive email or use office applications, they need the appropriate licenses, including Power BI Pro to edit and share content in Power BI.
- Contoso might want to enforce more stringent authorization and governance policies for external users compared to internal users. To achieve this, Contoso needs to create an in-house nomenclature for external users and all Contoso users need to be educated about this nomenclature.

- When the user leaves their organization, they continue to have access to Contoso's resources until the Contoso admin manually deletes their account
- Contoso admins have to manage the identity for the guest, including creation, password resets, etc.

Alternative Option 2: Create a custom Power BI Embedded application using custom authentication

Another option for Contoso is to build its own custom embedded Power BI application with custom authentication ('App owns data'). While many organizations do not have the time or resources to create a custom application to distribute Power BI content to their external partners, for some organizations this is the best approach and deserves serious consideration.

Often, organizations have existing partner portals that centralize access to all organizational resources for partners, provide isolation from internal organizational resources, and provide streamlined experiences for partners to support many partners and their individual users.



In the example above, users from each supplier login to Contoso's Partner Portal that uses AAD as an identity provider. It could use AAD B2B, Azure B2C, native identities, or federate with any number of other identity providers. The user would log in and access a partner portal build using Azure Web App or a similar infrastructure.

Within the web app, Power BI reports are embedded from a Power BI Embedded deployment. The web app would streamline access to the reports and any related services in a cohesive experience aimed to make it easy for suppliers to interact with Contoso. This portal environment would be isolated from the Contoso internal AAD and Contoso's internal Power BI environment to ensure suppliers could not access those resources. Typically, data would be stored in a separate Partner data warehouse to ensure isolation of data as well. This isolation has benefits since it limits the number of external users with direct access to your organization's data, limiting what data could potentially be available to the external user, and limiting accidental sharing with external users.

Using Power BI Embedded, the portal can leverage advantageous licensing, using app token or the master user plus premium capacity purchased in Azure model, which simplifies concerns about assigning licenses to end users, and can scale up/down based on expected usage. The portal can offer an overall higher quality and consistent experience since partners access a single portal designed with all of a Partner's needs in mind. Lastly, since Power BI Embedded based solutions are typically designed to be multi-tenant, it makes it easier to ensure isolation between partner organizations.

Reasons to choose this alternative:

- Easier to manage as the number of partner organizations grows. Since partners are added to a separate directory isolated from Contoso's internal AAD directory, it simplifies IT's governance duties and helps prevent accidental sharing of internal data to external users.
- Typical Partner Portals are highly branded experiences with consistent experiences across partners and streamlined to meet the needs of typical partners. Contoso can therefore offer a better overall experience to partners by integrating all required services into a single portal.

- Licensing costs for advanced scenarios like Editing content within the Power BI Embedded is covered by the Azure purchased Power BI Premium, and does not require assignment of Power BI Pro licenses to those users.
- Provides better isolation across partners if architected as a multi-tenant solution.
- The Partner Portal often includes other tools for partner beyond Power BI reports, dashboards, and apps.

Reasons not to choose this alternative:

- Significant effort is required to build, operate, and maintain such a portal making it a significant investment in resources and time.
- Time to solution is much longer than using B2B sharing since careful planning and execution across multiple workstreams is required.
- Where there are a smaller number of partners the effort required for this alternative is likely too high to justify.
- Collaboration with ad-hoc sharing is the primary scenario faced by your organization.
- The reports and dashboards are different for each partner. This alternative introduces management overhead beyond just sharing directly with Partners.

FAQ

Can Contoso send an invitation that is automatically redeemed, so that the user is just "ready to go"? Or does the user always have to click through to the redemption URL?

The end user must always click through the consent experience before they can access content.

If you will be inviting many guest users, we recommend that you delegate this from your core Azure AD admins by adding a user to the guest inviter role in the resource organization. This user can invite other users in the partner organization by using the sign-in UI, PowerShell scripts, or APIs. This reduces the administrative burden on your Azure AD admins to invite or resent invites to users at the partner organization.

Can Contoso force multi-factor authentication for guest users if its partners don't have multi-factor authentication?

Yes. For more information, see Conditional access for B2B collaboration users.

How does B2B collaboration work when the invited partner is using federation to add their own on-premises authentication?

If the partner has an Azure AD tenant that is federated to the on-premises authentication infrastructure, on-premises single sign-on (SSO) is automatically achieved. If the partner doesn't have an Azure AD tenant, an Azure AD account may be created for new users.

Can I invite guest users with consumer email accounts?

Inviting guest users with consumer email accounts is supported in Power BI. This includes domains such as hotmail.com, outlook.com and gmail.com. However, those users may experience limitations beyond what users with work or school accounts encounter.