Developing Power BI for Dataverse + PowerApps, and Dynamics 365

Author: Scott Sewell  
Revision Date: 5/19/2022 11:41:00 AM

Contents

[Overview 2](#_Toc104178257)

[Value Proposition: Use Power BI to Expand the Visibility of your Biz Apps 2](#_Toc104178258)

[Decision Factors for Getting Dataverse Data into Power BI 3](#_Toc104178259)

[Performance Tolerance 3](#_Toc104178260)

[Data Volume 4](#_Toc104178261)

[Security Complexity 4](#_Toc104178262)

[Configuration Complexity 5](#_Toc104178263)

[Best Practices for Developing Power BI Reports for Dataverse 5](#_Toc104178264)

[Focus on the specific use case at hand - rather than trying to solve ‘everything.’ 6](#_Toc104178265)

[Embrace a Star Schema rather than replicating the Dataverse Schema in Power BI 6](#_Toc104178266)

[Minimize the number of columns in the query rather than starting with ‘all.’ 6](#_Toc104178267)

[Optimize Power Query to improve the ETL from Dataverse into Power BI 7](#_Toc104178268)

[Complex Query / Filtering can be accomplished using SQL queries 7](#_Toc104178269)

[When using SQL Queries against Dataverse, use the ‘[EnableFolding=true]’ option 7](#_Toc104178270)

[Speed up the ‘Evaluating’ stage during a dataset refresh by ignoring relationships 8](#_Toc104178271)

[Helpful Hint – If labels are ‘blank’, Dataverse-customizations may need to be Published. 9](#_Toc104178272)

[Replicate the “My […]” filter from Dataverse Views in Power Query 10](#_Toc104178273)

[Extra Considerations for DirectQuery 11](#_Toc104178274)

[Consider Dual-Mode Dimension tables 12](#_Toc104178275)

[Don’t forget the “Use Report Viewer Identity” checkmark when publishing DirectQuery 12](#_Toc104178276)

[Enterprise Scale using Azure Synapse Link 14](#_Toc104178277)

[Creating the environment. 14](#_Toc104178278)

[Initial Configuration 14](#_Toc104178279)

[Initial Challenges 15](#_Toc104178280)

[Use a Second Serverless SQL instance for improved Ease of Use. 15](#_Toc104178281)

[Creating the Serverless SQL Instance to hold custom views 16](#_Toc104178282)

[Join existing Choice values with their Labels for a friendlier view. 16](#_Toc104178283)

[Using the Metadata views in a query 17](#_Toc104178284)

[Use custom views for data modeling ease. 18](#_Toc104178285)

[Creating the views for your organization 18](#_Toc104178286)

[Connecting Power BI to the Synapse Serverless SQL Views 19](#_Toc104178287)

[Conclusion 19](#_Toc104178288)

# Overview

Dataverse is the standard data platform of the Microsoft Business Application such as Dynamics 365 CE and Power Apps Canvas Apps – along with Forms Pro, Power Automate Approvals, Power Portals, and others.

This document will look at some critical learnings about connecting Power BI to Dataverse data, understand the difference between a Dataverse schema and an optimized Power BI schema, and provide best practices for lighting up Dataverse/PowerApps/Dynamics 365 CE with Power BI.

This document is intended to serve as a high-level guide to the process, a prompt for the decision process you’ll work through as you evaluate your options, and a pointer to relevant and useful documentation found online at <https://docs.microsoft.com>.

# Value Proposition: Use Power BI to Expand the Visibility of your Biz Apps

Because of its ease of configuration, rapid deployment, and widespread adoption, there’s now an increasing volume of data being managed and stored in Dataverse environments across an organization – and an even greater need and opportunity to bring the value of Analytics into those processes.

Power BI is a powerful accelerator for users in these applications – and their leadership. From the perspective of the organization using these apps, there are several key benefits:

1. Add a 360º overview for users/managers by including data from both inside and outside of Dataverse and filtering to the context of a specific record.
2. Report on all the data in Dataverse without the constraints of the built-in charts.
3. Take advantage of Power BI’s built-in AI over your data without code or complexity.
4. Provide easy access to relevant, contextually filtered reports within a specific record.
5. Increase speed and completeness of adoption of Power Platform applications by increasing their usability and value to the end-users.
6. Deliver the incredible value of the data in your app to Business Decision Makers.

# Decision Factors for Getting Dataverse Data into Power BI

There are three primary ways to connect Power BI to Dataverse data. (There are, of course, many more variations, but we’ll cover these as the most common examples.)

1. DirectQuery from Power BI to Dataverse. This method queries the data on-demand / real-time using the credentials of the report viewer.
2. Import into Power BI from Dataverse. This method pre-caches data from Dataverse either directly or through incremental refresh to provide blazing fast and responsive reports.
3. Import into Power BI from Dataverse via the Azure Synapse Link. This is another import method, primarily designed for enterprise-scale data and with the ability to integrate into the organization’s larger data strategy easily.

*Before* choosing a connection method, it’s critical to evaluate the report’s use across several decision factors. This will help you choose the right approach for the current reports.

Organizations will need to rank or choose the factors according to the report’s needs – and different reports within the same organization may take different approaches.

The most common decision factors are

1. Performance tolerance
2. Data volume
3. Security complexity
4. Configuration Complexity

## Performance Tolerance

Users will have different experiences interacting with a report based on imported data versus one built on DirectQuery. – The Imported data approach will result in a report that is dramatically faster to load and begin interacting with since the data is pre-cached and calculated. In contrast, the DirectQuery report is re-querying the data each time the report is opened, or new filters are applied.

The front-end report experience for a user viewing a report built on imported data is typically a much faster, more responsive report versus one built over DirectQuery. Since the data is already loaded into the data model, any interaction such as a filter/cross-filter or drill-through doesn’t require a re-query of the database. This results in a faster, more responsive report interaction experience for the user.

DirectQuery should be used only if other decision points, like the freshness of data, or enforcement of the Dataverse security model, require it. In cases where DirectQuery *is* needed, the report should be simplified. The amount of data being analyzed in the report are minimized to reduce DirectQuery performance bottlenecks as much as possible. We’ll cover specific DirectQuery performance strategies later in this document.

A good resource on design implications of DirectQuery is [Using DirectQuery in Power BI - Power BI](https://docs.microsoft.com/en-us/power-bi/connect-data/desktop-directquery-about#implications-of-using-directquery).

## Data Volume

The amount of data being transferred from the source into the report has a strong, direct impact on which strategy is used. With any approach other than DirectQuery, the impact on the user is minimized since the data is already in Power BI. With DirectQuery, as more data is requested from Dataverse, the time it takes to query and transfer that data also increases, and consequently, the user will wait longer for the report to refresh.

A report designed for front-line workers might be tightly constrained by security or other filters to provide the real-time operational report they need. On the other hand, a report for a group manager or planning team that for example might need to look at many users or a much wider timeframe would result in needing a much larger dataset in their report. The difference between the data volume in these two use cases has a strong influence on the data model design.

DirectQuery is considered most viable when the report’s result-set is not significantly large (meaning that there are fewer than 100K records in the report’s source tables, or the results returned to the report after filters are applied is less than 100K records.) [Create a Power BI report using the Dataverse connector - Power Apps](https://docs.microsoft.com/en-us/powerapps/maker/data-platform/data-platform-powerbi-connector) (Note that the 100K records size a hard limit but general guidance. Note that each query must return less than 80MB in under 2 minutes. Later in this document, we’ll talk about adapting to those limits and other essential strategies in Dataverse DirectQuery implementations.)

When the query result size is in the range of a few hundred thousand records, Power BI Premium Dataflows or Datasets with Incremental Refresh might be more appropriate. [Incremental refresh for datasets and real-time data in Power BI - Power BI | Microsoft Docs](https://docs.microsoft.com/en-us/power-bi/connect-data/incremental-refresh-overview) or [Using incremental refresh with dataflows - Power Query](https://docs.microsoft.com/en-us/power-query/dataflows/incremental-refresh)  Either of these would be an option for an organization that has somewhat more data needed in the report and can limit the queries to only recently modified records for the incremental query.

Larger datasets of a few hundred records upwards to multi-million record datasets can utilize the enterprise-scale supported by Dataverse Azure Synapse Link. This approach sets up an ongoing managed pipeline to copy Dataverse data into Azure Data Lake as CSV or Parquet files. The data can then be ingested from the lake via Azure Synapse and imported into Power BI. [Create an Azure Synapse Link for Dataverse with your Azure Synapse Workspace - Power Apps](https://docs.microsoft.com/en-us/powerapps/maker/data-platform/azure-synapse-link-synapse).

## Security Complexity

Dataverse supports incredibly complex security models to control the visibility of records to different users. (For example, one sales user might be able only to see their opportunities, but other opportunities are entirely hidden from them, yet a team manager would see all their team’s opportunities but not opportunities from another team. – The level of complexity implemented from one organization to another will vary with each organization’s needs.) Not every organization implements that level of complexity – and not every report needs to partition the data to accommodate that individualized partitioning of the data.

Suppose the report will be shown to users who each have different permissions controlling which records they are allowed to see. In that case, DirectQuery over Dataverse ensures that Power BI can display data to the user based on that user’s permissions from Dataverse. Two users with different permissions in Dataverse will see only the data they are allowed to see reflected in Power BI when viewing the same report.

Other organizations can manage security by simply controlling which users are allowed to see the report. Any user with permission to the report can see all data in that report. This could be where there are no rules in dynamics partitioning the data – or where, even if there are rules; the users are managers who have permission to see all data anyway. The more flexible and performant approach would be a direct import of the data, whether from Dataverse or via Synapse.

An advantage of imported data may be for organizations that wish to present summaries of a team’s key performance indicators such as “Total $ in Pipeline,” or “New Lead count” - even when the specifics of those records are not shared with the individual team members. The data can be extracted and summarized before being presented to the report viewer as a KPI without sharing details of the individual records.

Another scenario would be where relatively straightforward data access rules could be implemented using Power BI’s Row-Level security. [Row-level security (RLS) in Power BI Report Server - Power BI](https://docs.microsoft.com/en-us/power-bi/report-server/row-level-security-report-server#:~:text=%20Row-level%20security%20%28RLS%29%20in%20Power%20BI%20Report,Server%2C%20you%20manage%20security%20and%20add...%20See%20More.) This approach would support the data partitioning with improved performance over the DirectQuery method.

We caution that an RLS implementation should not attempt to mimic the Dataverse security model if the model implemented in Dataverse has any level of complexity implemented through rules around Ownership, Teams, Sharing, Inheritance, etc.

There’s no fixed guidance on precisely when the rules become too complex to implement in Power BI’s Row Level Security, but keep in mind that a Power BI RLS-based security model is entirely separate and would likely need ongoing management to ensure changes in Dynamics are in synch with the security rules in Power BI.

## Configuration Complexity

Connections to Dataverse – whether they be direct or import mode – are relatively straightforward and do not require any additional software or elevated permissions in Dataverse. This is an advantage for organizations or departments that are just getting started. It’s also an advantage for organizations that do not need the enterprise features available in the Synapse solution.

The Synapse option requires system administrator access to Dataverse and an Azure subscription. Configuration is not complex, but experience with Azure Storage and SQL is beneficial. This may limit which organizations can take advantage of this enterprise-oriented feature.

# Best Practices for Developing Power BI Reports for Dataverse

A few design patterns and anti-patterns are considered when developing a Power BI report over Dataverse data. – Only a few of these are unique to Dataverse, but they tend to be common challenges for Dataverse makers when they tackle building Power BI reports

## Focus on the specific use case at hand - rather than trying to solve ‘everything.’

It’s probably the most common and easily the most challenging anti-pattern to avoid - attempting to build a single data model to enable all self-service reporting needs with the hope that this master model will allow anyone to find any answers they need everyone will live happily ever after.

The reality is that successful data models are built to answer questions around a central set of facts over a single core topic. While that might initially seem to limit the model, it’s pretty empowering as the model can be tuned and optimized for answering questions within that topic.

Before developing a report model, consider questions such as these to help ensure you and your team have a crisp understanding of the objective.

* What topic area is this report supporting?
* Who is the audience of this report?
* What question is the report trying to answer?
* What’s the MVD - Minimum Viable Dataset?

Be hesitant to combine topic areas into a single data model “only” because the end customer asks for questions from multiple topic areas to be answered on a single dashboard. – Separating the portions of the reports derived from different core sources (“fact tables”) into their models will ultimately result in much more efficient and scalable (and manageable) models.

## Embrace a Star Schema rather than replicating the Dataverse Schema in Power BI

For Dataverse developers and administrators comfortable with the Dataverse schema, there is an immediate assumption that the model from Dataverse should be just copied or rebuilt within Power BI. This anti-pattern is probably the toughest to overcome since it just ‘feels right’ to maintain consistency between the two schemas.

Dataverse, as a relational model, is well suited for its purpose. It is not designed as an analytics model. The most prevalent pattern for modeling analytics data is a ‘Star’ Schema. At the center of the star is your ‘fact table’, representing what you want to count, sum, measure, etc. Surrounding the fact table are the ‘dimensions’ (the points on the star.) The dimension tables hold the descriptive elements that the report will use to slice, filter, and categorize the facts.



The topic surrounding how and why of the ‘Star’ schema approach is beyond this document, but to find more, please review the guidance here: [<https://docs.microsoft.com/en-us/power-bi/guidance/star-schema>](https://docs.microsoft.com/en-us/power-bi/guidance/star-schema).

## Minimize the number of columns in the query rather than starting with ‘all.’

By default, when you select a Dataverse table in Power BI, it retrieves all rows and all columns. In the case of the system user table, this could be more than 1,000 columns. (The columns in the metadata include relationships to other entities and lookups to option labels, so the total number of columns grows with the complexity of the Dataverse table.)

Attempting to extract all columns is an anti-pattern that creates performance bottlenecks in refreshes and can cause the Dataverse connector to fail when the size of the data being retrieved exceeds 80MB.

The recommended pattern is to only retrieve the columns from the source that are immediately needed for the report – and to be careful to re-evaluate and refactor the source query later as the report development progresses, often rendering previously required fields no longer necessary.

Additionally, ensuring that the ‘remove columns’ step is early enough in the query to ensure that it folds back to the source will avoid taking the time to extract data from the source into Power BI - only to later discard it in a later (unfolded) step.

An approach that has worked in cases where the table is too large to start with the interactive query builder step is to start with a Blank query, and in the ‘Advanced Editor’ window, paste in a minimal query as your starting point.

Graphical user interface, text, application

Description automatically generated

A simple query like this example will get you started and allow you to build up the query quickly:

Let

Source = CommonDataService.Database("demo.crm.dynamics.com"),

dataverse\_entity = Source{[Schema="dbo",Item="account"]}[Data],

#"Removed Other Columns" = Table.SelectColumns(dataverse\_entity,{"accountid", "name"})

in

#"Removed Other Columns"

## Optimize Power Query to improve the ETL from Dataverse into Power BI

A Power Query script is a step-by-step set of instructions for getting data out of the source and into the destination – each step builds on the result of the previous step. As a result, the types of transformations and even the sequence of steps can have a surprisingly high impact on the performance of the query.

“Query Folding” describes the process of Power Query sending as much of the transformation back to the source for it to be completed before any results are brought over into Power BI. A query that is properly ‘folding’ takes advantage of the power of the source server to handle as much of the effort as possible. This is optimum, but not all queries fold by default. Often some editing of the query can unblock the folding and allow this optimization to occur, greatly improving the performance. More details about determining whether a query is folding or not – and possible steps to unblock folding can be found in the document: [Query folding - Power Query | Microsoft Docs](https://docs.microsoft.com/en-us/power-query/power-query-folding)

Optimizing Power Query is a broad topic with many opportunities to ensure you’re getting the best performance. [Query Diagnostics - Power Query | Microsoft Docs](https://docs.microsoft.com/en-us/power-query/querydiagnostics) in the Microsoft Docs can shed light on performance and help you fine-tune the query or diagnose when a query isn’t functioning as efficiently as possible.

## Complex Query / Filtering can be accomplished using SQL queries

If a table requires additional joins or ‘interesting’ relationships, it may be beneficial to write the query in SQL. The Dataverse TDS endpoint can accept a SQL statement to request a specific portion of the data. In my experience, I’ve found it useful for reducing the size of a dimension table’s size by filtering to only records that have a corresponding record in the fact table. (e.g. select only the accounts/contacts with an opportunity or support case.) Additional details can be found here: [Use SQL to query data (Microsoft Dataverse) - Power Apps | Microsoft Docs](https://docs.microsoft.com/en-us/power-apps/developer/data-platform/dataverse-sql-query) There’s documentation on which SQL features are available the TDS endpoint: [How Dataverse SQL Differs from Transact-SQL - Power Apps | Microsoft Docs](https://docs.microsoft.com/en-us/power-apps/developer/data-platform/how-dataverse-sql-differs-from-transact-sql?tabs=supported)

## When using SQL Queries against Dataverse, use the ‘[EnableFolding=true]’ option

When using SQL Queries against Dataverse in a Value.NativeQuery statement, add the ‘[EnableFolding=true]’ option in Power Query to ensure queries are 'folding' back to the Dataverse service to gain dramatically improved performance in many Power BI queries. (Up to 97% faster in some cases!)

let

    Source = CommonDataService.Database("dataverseadl.crm.dynamics.com"),

    dbo\_account = Value.NativeQuery(  
Source,  
"SELECT A.accountid, A.name FROM Account A"  
,null  
,[EnableFolding=true]  
)

in

     dbo\_account

A PowerQuery is said to be "Folding" when subsequent filters are merged into the original query and sent to the source for processing. Folding is an optimization that can in many cases dramatically reduce the time it takes to load the data into the dataset/report.

By default, SQL queries against Dataverse's TDS endpoint do not fold, unless the 'EndableFolding' option is added.

The ‘Value.NativeQuery’ Power Query M function allows you to pass an option to the source. For the Dataverse (née “CommonDataService”) source, enabling ‘Folding’ allows in many cases additional parameters from the power query and from the report to be passed back up to the Dataverse server and evaluated there. – This can result in a measurable performance improvement when a large dataset can be filtered to a small result in the Dataverse service (as opposed to downloading the entire dataset first, only to filter it after it has been loaded into Power BI.)

In practice, the greatest performance impact will be seen on queries against large datasets where only a limited subset of the data is needed for the result.

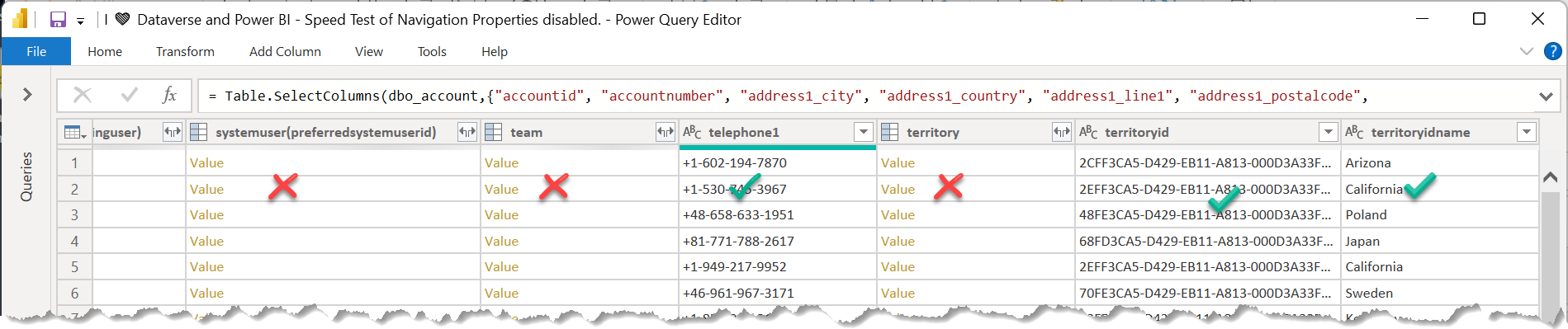
The performance improvement will also vary based on the way Power BI executes some measures or queries. - I found that one of my measures ("COUNTDISTINCT") showed almost no improvement with or without the folding hint. I changed the measure to count the distinct rows with an iterator (using SUMX.) That change allowed the measure to begin folding and resulted in a 97% improvement over the same query without the hint.

The syntax is documented here [Value.NativeQuery - PowerQuery M | Microsoft Docs](https://docs.microsoft.com/en-us/powerquery-m/value-nativequery#syntax) – (the [EnableFolding] option is not listed since it is native to the source.)

## No alt text provided for this imageSpeed up the ‘Evaluating’ stage during a dataset refresh by ignoring relationships

If you're using the Dataverse connector (*née* "CommonDataService") in PowerQuery/Power BI, try adding the [CreateNavigationProperties=false] as a hint to dramatically speed up the "Evaluation" stage of a data import / refresh. (Note: This only is appropriate as long as you're not "expanding" any of the relationship columns - which, by the way, is a terrible hit on performance and should be avoided.)

The "Evaluating" step of a Power BI "Refresh" iterates through the metadata of its source to evaluate all the relationships to the table being queried. - That metadata can be, especially in Dataverse, extensive. - Adding a hint to the query to let Power Query know that you're not using those relationships will allow Power BI to skip that part of the refresh and get on with the business of retrieving your data. –



In this example from the account table, once the hint is applied, the TerritoryID and TerritoryID name will remain - but the 'relationship columns' (The ones with "Value" in the field) will be excluded. (BTW - this has nothing to do with the relationships you create inside Power BI - those are entirely different.)

How To: When using the current Dataverse connector, (still named 'CommonDataService' in PowerQuery), add the hint "[CreateNavigationProperties=false]" after the Dataverse service name. This tells Power Query to ignore the table's relationship metadata from Dataverse and just pull fields from the entity itself.  


The impact will be felt more when you're interacting with tables that naturally have lots of relationships with other tables in Dataverse. For example, because it's connected to every other table in the system in multiple ways the SystemUser table is a particularly ‘heavy’ table to evaluate.

*Note: this doesn't speed up the interactive cross-filtering in Direct Query, as those interactions do not require refreshed metadata. - It does speed up any import or refresh of Dual Mode datasets - and speeds up the process of 'applying changes' from Power Query Editor over to Power BI.*

## Helpful Hint – If labels are ‘blank’, Dataverse-customizations may need to be Published.

I recently worked with two talented report builders wrestling with a challenge using Dataverse with Power BI. They had developed and tested their reports in DEV – and everything looked fine, however after the report was migrated to PROD, all the choice labels were BLANK! 😲

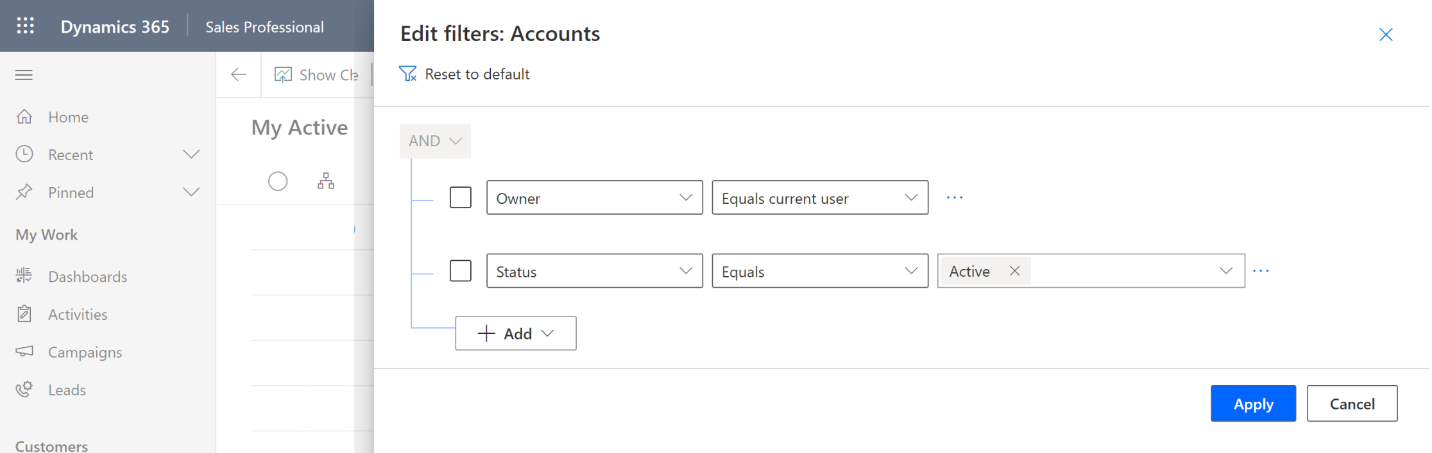
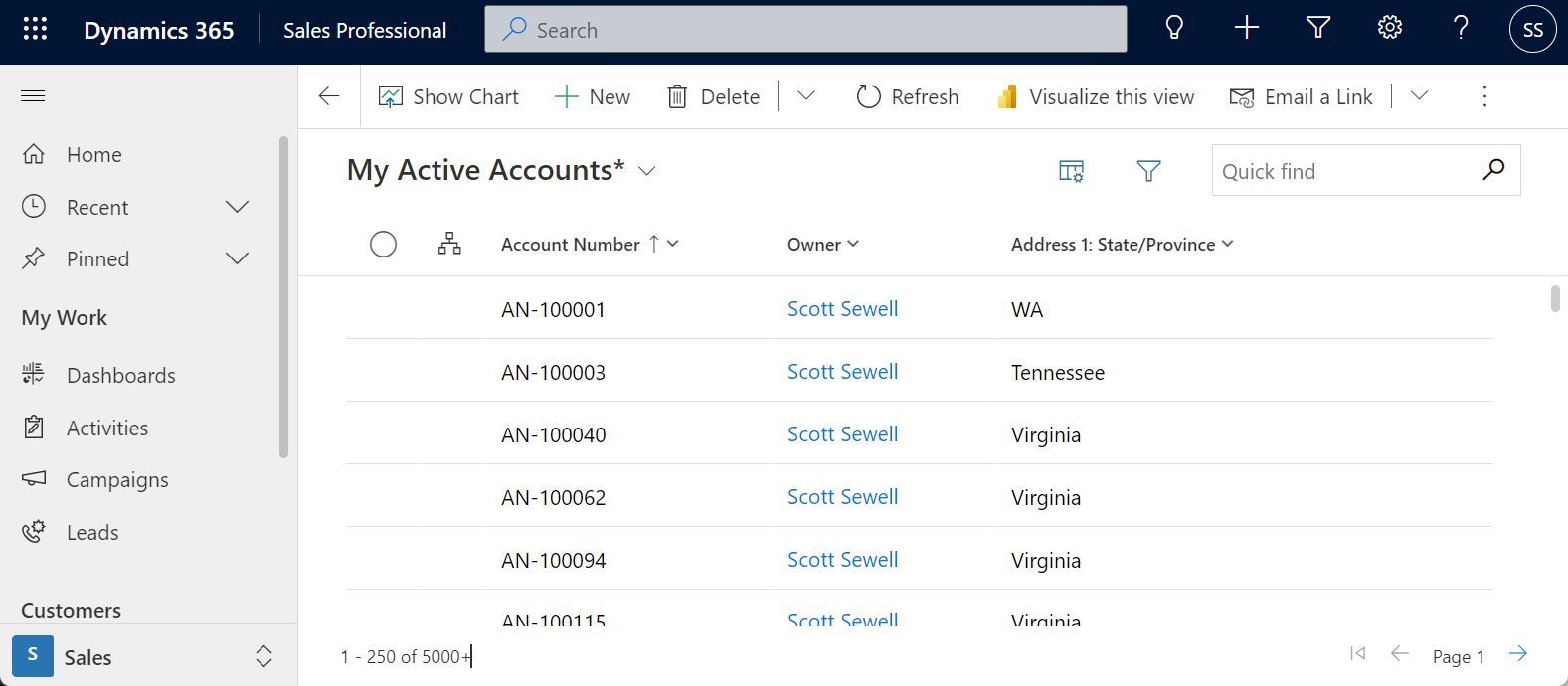
The fix is to open the Dataverse Maker Portal ([make.powerapps.com](http://make.powerapps.com/)) Go to the "Solutions" area, and then click the 'Publish all customizations' button.

Publishing will update the TDS endpoint with the latest metadata and make those option labels available in the report, like magic!

# Replicate the “My […]” filter from Dataverse Views in Power Query

Dynamics 365 CE, and Model Driven Power Apps built on Dataverse can create views to be filtered to only show records where a username field on that record, such as 'owner' equals the current user. (e.g. “My Open Opportunities”, “My Active Cases”, “My Sharona” etc.)

Here's an example of how a Dynamics 365 "My Active Accounts" view has been filtered based on "Owner Equals current user"

You can replicate the same functionality for Power BI / Power Query against the Dataverse TDS End Point by using a NativeQuery with a “CURRENT\_USER” keyword.

Here's an example query where the result delivered back into Power BI will be filtered to only the records where the accounts’ ownerid field is the same as the currently authenticated user. Line 15 is where the magic happens.

Graphical user interface, text

Description automatically generated

Note this pulls the data in using this filter for the current user, so it’s a natural for Direct Query to limit the scope of the data to a manageable size and scope. - When published to the service, be sure to 'check' the “Use Report Viewer Identity” checkmark when setting up the authentication.

# Extra Considerations for DirectQuery

DirectQuery is a feature that allows the report to send a query back to the source data at the moment the report is opened. The query can pull effectively real-time results using the current report user's permissions. There are lots of use cases where DirectQuery solves important requirements. However, those features come at a performance cost.

A report built using DirectQuery will never be as fast a report as one built over an 'imported' data model - and data volume needs to be carefully managed to avoid unnecessary slowdowns. Managing both scope and expectations is important.

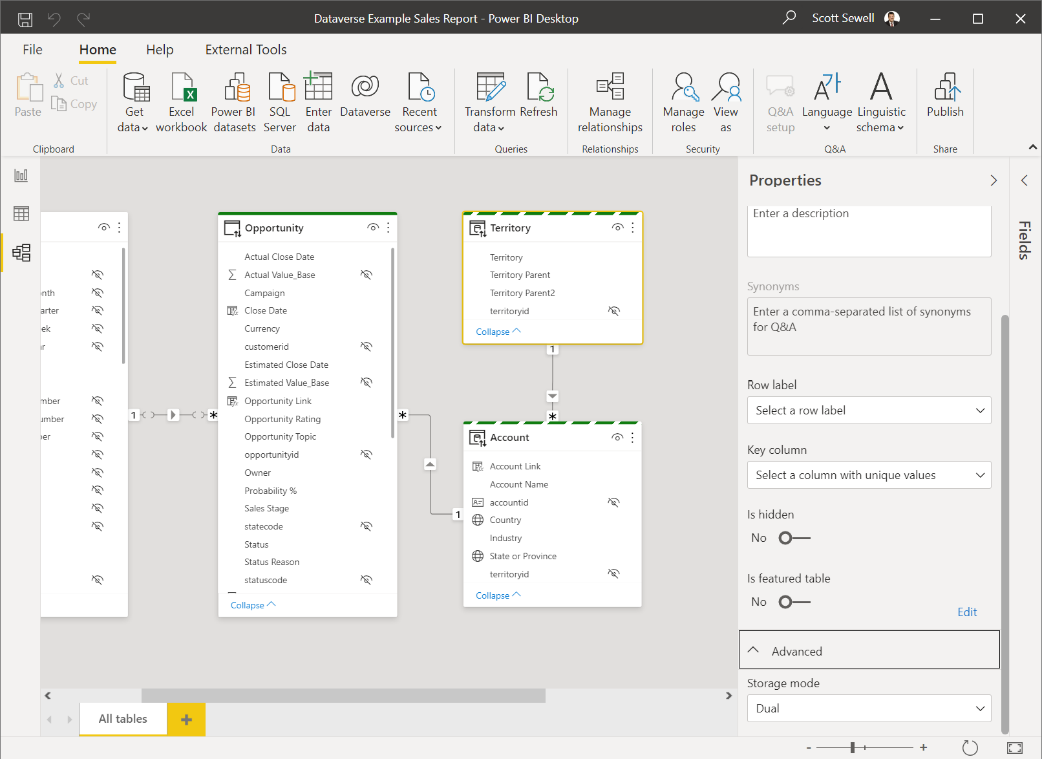
That said, the above best practices are doubly important to ensure they are being followed with DirectQuery reports - along with these reminders.

## Consider Dual-Mode Dimension tables

When working with DirectQuery, some tables serve as dimension (i.e. ‘lookup’) tables for labels or slicers. – By setting the storage mode to “Dual,” these tables can act as either cached or not cached, depending on the context of the query submitted to the Power BI dataset. In some cases, you fulfill queries from cached data. In other instances, you fulfill queries by executing an on-demand query to the data source.

This ability to act as a ‘cached’ table dramatically improves the speed and responsiveness of the data model.

Note that if the dimension table needs to inherit the security model of Dataverse, Dual Mode is not appropriate. Setting the dimension tables to ‘Dual’ mode is reversible to DirectQuery mode if needed. (A change to ‘Import’ mode is not reversible.)



For a deeper explanation of the advantages and implications of ‘Dual’ mode, see [Use storage mode in Power BI Desktop - Power BI | Microsoft Docs](https://docs.microsoft.com/en-us/power-bi/transform-model/desktop-storage-mode).

## Don’t forget the “Use Report Viewer Identity” checkmark when publishing DirectQuery

Finally, it’s not so much a performance issue, but an easily overlooked setting when a new DirectQuery report is published. If the report needs to inherit the security from the source, the connection setting “Report Viewers can only access this data source with their own Power BI identities using DirectQuery” needs to be checked to ensure that the connection is set to render the report with the security settings of the viewer.

Graphical user interface, text, website

Description automatically generated

# Enterprise Scale using Azure Synapse Link

Built into Dataverse is the ability to synchronize tables to an Azure Data Lake and then connect to that data through an Azure Synapse Workspace. With only a few clicks an organization can populate Dataverse data into Azure Synapse and enable data teams to discover deeper insights.

Azure Synapse Link enables a continuous replication of the data and metadata from Dataverse into the data lake and provides a Serverless SQL as a convenient data source for the Power BI queries.

The strengths of this approach are significant. Customers gain the ability to run analytics, business intelligence, and machine learning across Dataverse data using a variety of advanced tooling available such as Apache Spark, Power BI, Azure Data Factory, Azure Databricks, and Azure Machine Learning.

## Creating the environment.

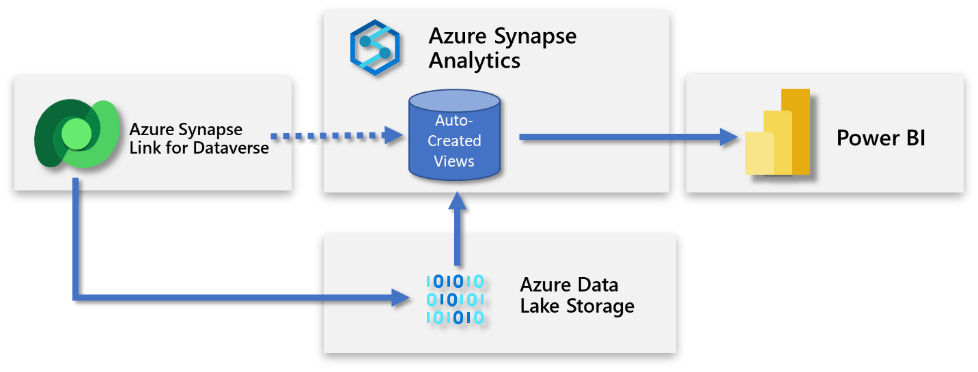
You will need:

1. System Administrator access to the Dataverse environment.
2. Azure Data Lake Storage Gen2: You must have an Azure Data Lake Storage Gen2 account and Owner and Storage Blob Data Contributor role access. Your storage account must enable Hierarchical namespace, and it is recommended that replication is set to read-access geo-redundant storage (RA-GRS).
3. Synapse workspace: You must have a Synapse workspace and the Synapse Administrator role access within the Synapse Studio. The Synapse workspace must be in the same region as your Azure Data Lake Storage Gen2 account. The storage account must be added as a linked service within the Synapse Studio.

For additional information and an expansion of the requirements for the environment, see the complete documentation here: [Create an Azure Synapse Link for Dataverse with your Azure Synapse Workspace - Power Apps | Microsoft Docs](https://docs.microsoft.com/en-us/powerapps/maker/data-platform/azure-synapse-link-synapse#prerequisites)

## Initial Configuration

Completing the setup wizard will provide an environment that can serve as a source for Power BI. - The Synapse link syncs data to the Azure Data Lake storage and configures views in a Serverless SQL in Synapse. Power BI can connect to those views, and the data model can be built for reporting.



## Initial Challenges

The above model is straightforward but puts an additional burden on the user building the model in Power BI to sort through the source schema and filter out inactive records, remove irrelevant fields, and add in the text values for ‘choice’ values.

For example, the un-customized Account table view contains more than 200 columns. The view also shows both active and inactive records. Depending on the customer use case, much of this is irrelevant and would need to be excluded in the query in Power BI.

Additionally, fields of type Status, Status Reason, and Choice are represented in the default views with only their numeric values. The corresponding text label is available via the metadata delivered in a JSON file. This works in principle but is unwieldy in practice when writing a query from Power BI to these views.

## Use a Second Serverless SQL instance for improved Ease of Use.

An alternative approach is to add customized views tailored to the business. This allows you to present a simplified set of data to the end-user, thus allowing them to begin building the data model based on generally relevant data.

This new serverless SQL database now becomes the report builder’s primary source connection and a friendly front-end to the data coming from the Data Lake.

Diagram

Description automatically generated with medium confidence

With this approach, the data exported from Dataverse can be focused, enriched, and filtered before presenting to the Power BI Data Model developer.

### Creating the Serverless SQL Instance to hold custom views

Begin by creating a Serverless SQL database in the same Synapse Workspace. ([Serverless SQL pool - Azure Synapse Analytics | Microsoft Docs](https://docs.microsoft.com/en-us/azure/synapse-analytics/sql/on-demand-workspace-overview)) – Select “Serverless” as the Pool Type and give it a name. (Your users will use this database name in their connections, so making it ‘friendly’ has benefits.)

Graphical user interface, text, application

Description automatically generated

### Join existing Choice values with their Labels for a friendlier view.

[Access Dataverse choice labels directly from Azure Synapse Link for Dataverse - Power Apps | Microsoft Docs](https://docs.microsoft.com/en-us/power-apps/maker/data-platform/azure-synapse-link-choice-labels)

[Expand with Examples]

After creating an Azure Synapse Link, the following five tables are created in a folder named **OptionsetMetadata** in Azure Data Lake Storage Gen2:

* OptionsetMetadata
* GlobalOptionsetMetadata
* StateMetadata
* StatusMetadata
* TargetMetadata

### Using the Metadata views in a query

In any query, you can now join the Metadata views to the table views in a custom Serverless SQL database

Here’s an example query that pulls the base campaign table together with the optionset and the statuscode metadata in order to return the campaign typecode and statuscode labels.

Graphical user interface, application

Description automatically generated

## Use custom views for data modeling ease.

In the architecture model, we’re adding views to the database I’ve labeled ‘custom report views.’ – The views will serve as a more straightforward, cleaner source for Power BI report models to consume.

Diagram

Description automatically generated with medium confidence

The objectives for the views are:

1. Provide the needed labels associated with Choice fields to make it much easier to add to queries in Power BI without additional complexity.
2. Make it simple – present (only) the needed tables and columns to the report modeler. Dataverse tables often contain a few dozen to several hundred columns that are created and added automatically by the system. Many of these columns are empty or would generally be unneeded in analytics reporting.
3. Remove unnecessary rows such as inactive records – or filter out unnecessary rows such as leads or accounts without cases or opportunities. – The specifics will be dependent on the customer’s use case.

### Creating the views for your organization

The specifics of your views are, of course, unique to your organization. Here’s an example of a product view. Notice how only a few columns from the product table are included, the stringmap view is referenced to retrieve the productstructure label value, and the view is filtered to include only the rows where statecode = 0 (where Status is “Active”)

CREATE VIEW <Custom\_SQL\_Database>.[dbo].[Product]

AS

SELECT P.[productid],

P.[productnumber],

P.[name],

productstructure.LocalizedLabel productstructurename,

P.[parentproductid],

P.[parentproductidname],

P.[subjectidname]

FROM <Synapse\_SQL\_LakeDatabase>.[dbo].[product\_partitioned] P

JOIN <Custom\_SQL\_Database>.dbo.stringmap productstructure

ON P.productstructure = productstructure.[option]

AND productstructure.LocalizedLabelLanguageCode = 1033

AND productstructure.EntityName = 'product'

AND productstructure.OptionSetName = 'productstructure'

where P.statecode = 0;

Continue creating the views needed for your reports. There’s a balance between creating these views with only a specific use case in mind – versus creating a sprawling unmanaged view that misses the value of the focused and tailored approach. You’re encouraged to implement a documentation and governance process to ensure that new changes won’t break existing reports.

### Connecting Power BI to the Synapse Serverless SQL Views

The query to connect to the Serverless SQL pool is the same as other SQL connections.

To find the address of your SQL server in Synapse Studio,

1. Open the ‘Manage’ blade
2. Click on the Built-in SQL Pool
3. Copy the Workspace SQL endpoint

Graphical user interface, application

Description automatically generated

Choose “Get Data” in Power BI and select the “Azure Synapse Analytics SQL” connector.

Graphical user interface, application

Description automatically generated

Then paste the “Workspace SQL endpoint” from the synapse studio into the ‘Server’ field.  
Graphical user interface, text, application

Description automatically generated

From this point, you can choose the database, tables, and specific fields you want to include in the data model.

# Conclusion

Power BI is amazing as a tool for exposing the value of data stored in Dataverse to Business Decision Makers. Getting the data out of Dataverse and into Power BI is the first step on a rewarding journey of discovery. Have fun and I hope this document has been helpful in getting you started.