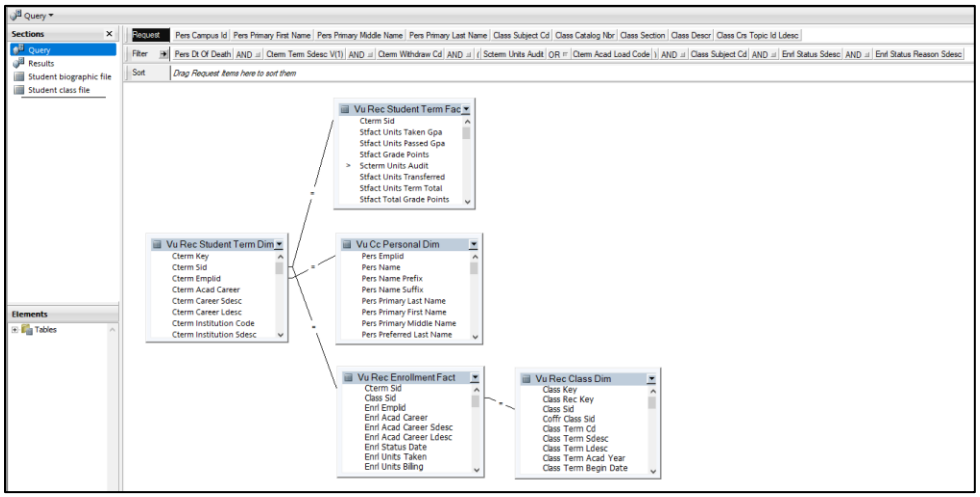
Converting Hyperion Queries to Power BI

This guide will cover the process of obtaining SQL text from Hyperion BQY files, editing and updating that query text to recreate calculated columns and other items which are not exported, and then using that modified SQL query to create a new Power BI data set. This document will **not** cover the process of joining data sets in a Power BI data model or show you how to create visualizations to present your data.

The goals of this document are to 1) provide instructions for individual units to preserve critical UWM queries; 2) introduce the basic functionality and core concepts of Power BI; 3) share additional training and educational resources which will allow campus authors to explore more advanced Power BI concepts.

This document assumes that you will be connecting to the UWM production data warehouse (dwprod) and that you have already installed the 64-bit Power BI Desktop tool through the **Software Center** client on your campus desktop. A couple of important notes before beginning the process:

- You will likely **not** be able to connect to the UWM data warehouse if you use the Power BI Desktop install file from the Microsoft website. The Software Center package includes additional Windows components which are required for Power BI operation. If you do not see this software available in the Software Center, please submit a support ticket requesting the Power BI Desktop tool.
- The procedure shown here will only replicate the tables, fields, join relationships, filters, and sorts from *a single **Query** section of your Hyperion BQY file*:



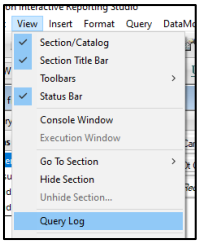
The process below works by exporting a single Hyperion query and importing it to Power BI, where you execute that query to recreate the same data set. **It is most appropriate for preserving and migrating simpler queries with only a few Query sections.** Additionally, any business logic *after* this initial query (i.e., filters, sort orders, or calculated columns added in your **Results** section or subsequent tables) will need to be recreated manually in your SQL statement or (ideally) in Power BI.

If you have a file with multiple Query sections, you will need to repeat the process in this guide for each Query section in your BQY file to generate separate data sets which can be joined in the Power BI data model. This data modeling is beyond the scope of this guide, but some web-based resources and training materials are provided at the end of this document.

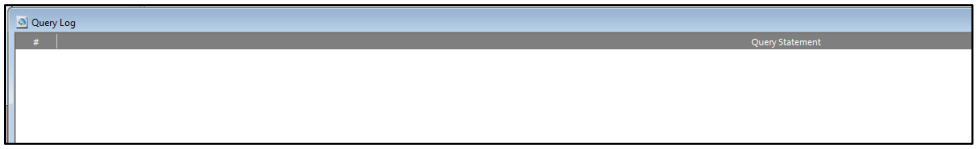
For more complicated multi-section Hyperion queries, I would strongly recommend taking some time to analyze and understand your query needs before simply using this process to recreate each Query section in Power BI (we'll cover some examples of this type of analysis later in this guide).

Generating Your SQL

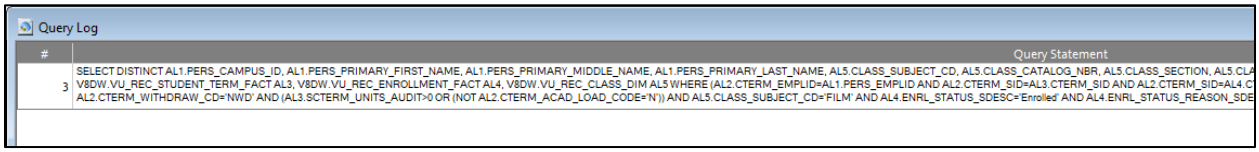
Open your BQY file in Hyperion, and then go to the **View** menu and select **Query Log**.



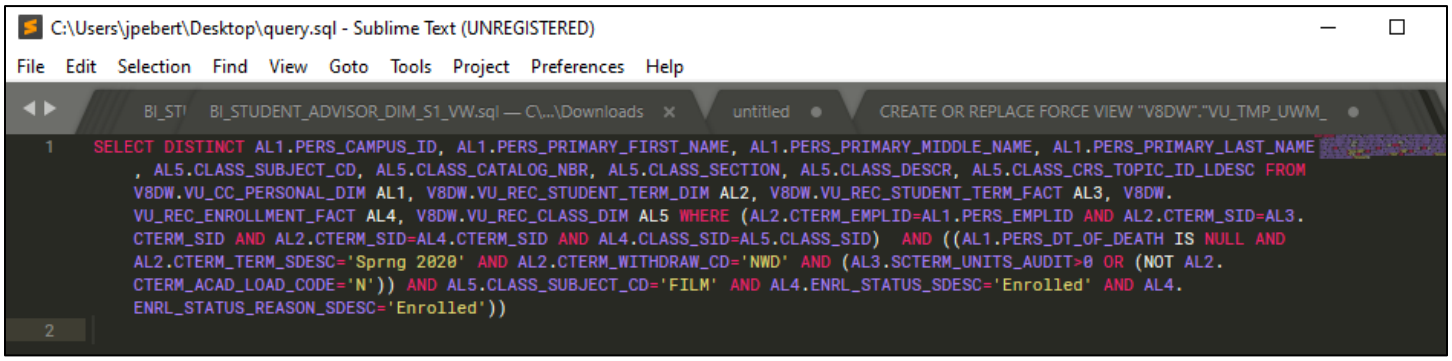
You should see an empty window appear at the bottom of your Hyperion application:



Process your query like you normally would, and when it's done, notice that the query log window is no longer empty:



Click anywhere on the new text to select it, and then open a text editing program like Notepad and paste in your SQL query. (You can use Word, but it may helpfully include "smart quotes" which can introduce errors into your queries):



The application above is called Sublime Text (it's free!) – one of my favorite features of this program is the color-coding for the different elements of SQL text (provided the file is saved with a .sql file extension). It also allows searching and string replacement with regular expressions which can simplify query creation and make things easier to read.

I would also recommend pasting your query text into a tool like SQL Format (https://sqlformat.org/) to add line breaks and indentations which make it more readable (especially if you're new to SQL!) – this is a good way to break down the language into recognizable parts.

Editing Your SQL

This step is where you could take your SQL query and add missing calculated columns, filters, or sorts. If you're generally familiar with SQL or don't have any computed columns and just want the instructions for integrating your query into a new tool, you can skip to the **Core Concepts of Power BI** section.

This section is intended to explain the basics of SQL, to help authors understand what their queries are doing and how they can be modified using SQL, and finally to provide some resources for curious authors to learn more about the SQL language.

If you are unfamiliar with both SQL and Power BI, I would **highly recommend** taking this opportunity to perform your query modifications in Power BI instead, as it provides a more forgiving and user-friendly development environment, and minimizes the total number of new concepts you will have to learn.

A Brief Introduction to SQL

Here's our query, after some help from SQL Format and color-coding to help explain the various sections:

```
SELECT DISTINCT AL1.PERS_CAMPUS_ID,
                AL1.PERS_PRIMARY_FIRST_NAME,
                AL1.PERS_PRIMARY_MIDDLE_NAME,
                AL1.PERS_PRIMARY_LAST_NAME,
                AL5.CLASS_SUBJECT_CD,
                AL5.CLASS_CATALOG_NBR,
                AL5.CLASS_SECTION,
                AL5.CLASS_DESCR,
                AL5.CLASS_CRS_TOPIC_ID_LDESC
FROM V8DW.VU_CC_PERSONAL_DIM AL1,
     V8DW.VU_REC_STUDENT_TERM_DIM AL2,
     V8DW.VU_REC_STUDENT_TERM_FACT AL3,
     V8DW.VU_REC_ENROLLMENT_FACT AL4,
     V8DW.VU_REC_CLASS_DIM AL5
WHERE (AL2.CTERM_EMPLID=AL1.PERS_EMPLID
       AND AL2.CTERM_SID=AL3.CTERM_SID
       AND AL2.CTERM_SID=AL4.CTERM_SID
       AND AL4.CLASS_SID=AL5.CLASS_SID)
  AND ((AL1.PERS_DT_OF_DEATH IS NULL
        AND AL2.CTERM_TERM_SDESC='Sprng 2020'
        AND AL2.CTERM_WITHDRAW_CD='NWD'
        AND (AL3.SCTERM_UNITS_AUDIT>0
             OR (NOT AL2.CTERM_ACAD_LOAD_CODE='N'))
        AND AL5.CLASS_SUBJECT_CD='FILM'
        AND AL4.ENRL_STATUS_SDESC='Enrolled'
        AND AL4.ENRL_STATUS_REASON_SDESC='Enrolled'))
```

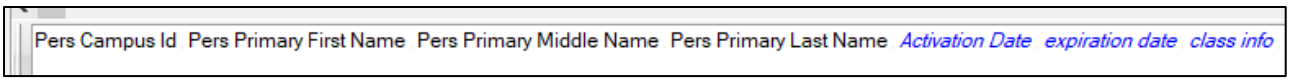
- All your SQL queries will start with a **SELECT** clause – if you've chosen to "return unique rows" in your query, you'll also have the word **DISTINCT**, as above.
- The orange section indicates the fields which are "selected" in your query, in the format of **TABLE ALIAS.FIELD** (you'll see the **ALx** "aliases" defined in the green section)
- The green section has the **FROM** clause, specifying the tables included in your query and their aliases, in the format of **SCHEMA.TABLE ALIAS**
- The yellow section has the **WHERE** clause, with the table join conditions (also using the table aliases)
- The blue section of the **WHERE** clause has the filters from your query, including any nested logic (as you can see in the load code and audit condition) and again, uses the table aliases.

There are different SQL variants used by different database vendors (e.g., Microsoft uses SQL Server, MySQL and MariaDB are open-source alternatives). While the basic syntax is similar, the functions and other advanced features can vary (think British vs. American English). Our data warehouse databases and the functions below use **PLSQL**, the Oracle standard SQL language – when Googling or otherwise learning about SQL techniques, make sure you specify the PLSQL variety.

You may see PLSQL examples with a different join syntax than the one shown above with the conditions in the WHERE clause. Both methods perform identically. For a comparison of both methods with side-by-side examples, check out this page: ANSI ISO SQL Support In Oracle 9i

Modifying Your SQL Query

For the purpose of introducing SQL a bit more, I'll show you how you would update your SQL statement to create three *very* basic types of computed columns from my query (again, with the caveat that I would **strongly recommend** performing these operations in Power BI instead!):



- **Activation Date** is a simple date column with the current date
- **Expiration Date** is a constant string with the end date of the term
- **Class Info** is a field which conditionally combines different class elements into one field

Field Name	Hyperion Formula	SQL
ACTIVATION DATE	Sysdate()	SYSDATE as TODAY
EXPIRATION DATE	'05/31/20'	'05/31/20' as EXPIRATION_DATE
CLASS INFO	if (Class_Crs_Topic_Id_Ldesc != null) {Class_Subject_Cd + Class_Catalog_Nbr + '-' + Class_Section + ' '+ Class_Descr + ' (' + Class_Crs_Topic_Id_Ldesc + ')' } else {Class_Subject_Cd + Class_Catalog_Nbr + '-' + Class_Section + ' '+ Class_Descr }	CASE WHEN Class_Crs_Topic_Id_Ldesc is not null THEN Class_Subject_Cd || Class_Catalog_Nbr || '-' || Class_Section || ' ' || Class_Descr || ' (' || Class_Crs_Topic_Id_Ldesc || ')' ELSE Class_Subject_Cd || Class_Catalog_Nbr || '-' || Class_Section || ' ' || Class_Descr END as CLASS_INFO

I want to draw your attention to two things – first, the syntax for the first couple of columns is nearly identical. Even in the third, while it uses a CASE statement for testing a condition and a pipe operator for combining strings, is also fairly straightforward. Second, I've removed all spaces from column names – this isn't necessary, as you can wrap column names in double-quotes to preserve spaces, but it's easier to maintain consistency between warehouse columns and other computed items.

So, with this new SQL, we can add our Hyperion computed columns back to the query:

TODAY	EXPIRATION_DATE	CLASS_INFO
09-MAR-20	05/31/20	FILM 107-802 Dig Filmmaking for Non-Majors
09-MAR-20	05/31/20	FILM 150-003 Multicultural America (D)
09-MAR-20	05/31/20	FILM 231-804 Concpt Devel for Film Practice
09-MAR-20	05/31/20	FILM 118-806 Sound and Image
09-MAR-20	05/31/20	FILM 231-804 Concpt Devel for Film Practice
09-MAR-20	05/31/20	FILM 201-001 Intro: Experimental Media Arts
09-MAR-20	05/31/20	FILM 118-803 Sound and Image
09-MAR-20	05/31/20	FILM 231-804 Concpt Devel for Film Practice

You'll notice that the basic **SYSDATE** formula above brings our date back in a different format than in our string date column. To fix this, we could use the TO_CHAR() function to change the formula slightly and convert our system date to a character column to match. Changing our column to:

```
TO_CHAR(SYSDATE, 'MM/DD/YY') as TODAY
```

Results in the same date format in both columns:

TODAY	EXPIRATION_DATE	CLASS_INFO
03/09/20	05/31/20	FILM 116-401 Listening and Recording
03/09/20	05/31/20	FILM 150-003 Multicultural America (D)
03/09/20	05/31/20	FILM 118-803 Sound and Image
03/09/20	05/31/20	FILM 117-801 Filmmaking Technlgy & Tchnques
03/09/20	05/31/20	FILM 116-802 Listening and Recording
03/09/20	05/31/20	FILM 231-805 Concpt Devel for Film Practice
03/09/20	05/31/20	FILM 118-806 Sound and Image

Note that I am just using the example above to illustrate how you can easily modify SQL statements to meet your needs – **I do not suggest converting dates to strings to make the formatting look nicer**. If anything, you should be converting the many string date fields in our warehouse **back** to actual date fields to take advantage of the powerful time reporting capabilities in Power BI.

Common SQL Functions

Here's a list of some common PLSQL functions:

SQL Function	Description
CASE	Replaces the if/else conditional logic from Hyperion
LISTAGG	Combines text string values from multiple rows into one row
DENSE_RANK	Creates a running rank/order between columns over a specific group
DECODE	Identical to the DECODE syntax used in Hyperion
MAX	Used to return the max value of a column over a certain group
SUM	Used to return the sum of two (or more) fields or the sum of multiple rows over a given group
NVL	Identical to the NVL syntax used in Hyperion

And a list of other helpful links, from basic to more complex:

- Google – honestly, the best resource for figuring out a) the proper terminology for what you're trying to do; b) whether or not that thing is possible; and c) if it's possible, how to do it (and again, remember to use PLSQL as a keyword to confine results to only Oracle SQL solutions)
- A basic introduction to SQL syntax and concepts: https://www.w3schools.com/sql/
- A list of common Oracle SQL functions: https://www.techonthenet.com/oracle/functions/
- The Oracle DevGym for basic classes and "workouts" on specific topics and Oracle Live SQL sites with examples of advanced functionality (these two sites require a free Oracle.com account)
- A Gentle Introduction to Common SQL Window Functions
- Oracle's SQL for Analysis and Reporting documentation

Core Power BI Concepts

Before we delve into the import process, I want to introduce the two Power BI query languages – M and DAX – and some core concepts to help you understand the full potential of the tool and add business value to your data sets. Before that, there are three important points that I want to mention.

First, there is a ton of overlap in the capabilities of SQL, DAX, and M, and most of the operations shown here can be done in any one of them. The goal of this document is to provide a foundation and resources for learning each language, present relative strengths and weaknesses, and give some recommendations for when to use each tool in the toolbox.

Second, there is an enormous amount of depth to Power BI and this document only scratches the surface, largely by simplifying concepts for ease of introduction. Wherever possible, I've included links to more in-depth articles on important concepts to offer opportunities for you to learn more about the complexities of Power BI – please use them!

Third, I want to emphasize that this guide is intended to show the process of converting a single Query from a simple Hyperion BQY file. If you have a more complicated BQY file with things like multiple Query sections, results from other data sources (i.e., Access, CSVs), or many calculated items or tables that further refine results, you may not benefit from the process shown here. Instead, you may want to spend time analyzing your query and rebuilding it in Power BI using a more traditional query development process. Some scenarios for analysis and benefits to developing directly in Power BI are provided in the **Four Steps to Analyzing and Learning (Power BI)** section near the end of this document.

Power BI Languages: M and DAX

Power Query (also commonly called **M**) is a language that allows you to transform and modify elements of an individual data set (like your Hyperion SQL) before you import it into the Power BI data model. **DAX** allows you to build calculated columns and measures from multiple data sets after they've been imported into the data model and integrate user selections into *context-aware* data elements.

What do I mean by context-aware? Let's say I created a Hyperion data set showing the total Fall 2019 credits for all subject areas and showed credit totals by college – by creating a column with a sum of credits with a 'break' column of Academic Group. Pretty easy, but what if the user wanted to filter that table to just credits in MATH courses? In Hyperion, I would have needed to predict that need and define a second column which summed credits with a break column of the Subject. And what if the user wanted to filter further to just MATH *lecture* courses? I would have had to create a third column with a break column of Subject AND Course Component, and so on.

As the complexity of the user's questions and filter selections increases, our ability to predict their query needs and answer their business questions is greatly reduced with Hyperion (and even to some extent SQL – apart from "window functions" which allow you to aggregate/reference data from other rows in the same data set). Over time, this reduction just results in more query requests as users consume data, and inevitably return with more informed questions which require more custom calculations. In Power BI, a single "total credits" measure written in DAX would return accurate credits totals on the fly for all three of the requests above using the *context* of the user's current filter selections to perform the appropriate calculations.

One helpful blog post uses the analogy that M is the "sous chef" of Power BI – chopping up vegetables and making sauces to improve the flavor of the final product. DAX is the head chef who knows what's appetizing, plans out the meal, and ultimately combines those ingredients into tasty meals for your customers. To extend this metaphor a bit, SQL might be the food delivery service – choosing and delivering the raw ingredients for DAX and M to use.

Data Sets, Data Models, and DAX Measures

As I've explained earlier, this guide covers the process of creating a single Power BI *data set* from a single Hyperion Query section. If you need to join the results from multiple Hyperion Query sections, you'll need to repeat this process and then create a Power BI *data model* to create relationships (joins) between these different data sets. Keep in mind that your data sets can also be from non-BQY query data sources (such as Excel/CSV files or SharePoint lists) which Power BI can import.

The Power BI data model is a complicated topic that is beyond the scope of this guide (see the **Power BI Resources** section at the end of this document for some related training videos and documentation) but I want to take a moment to explain these core concepts in Hyperion terms. Think of each Power BI data set as the final "local results" table you build from each Query section, with SQL and M being the languages that you would use to select your tables/fields and joins and transform the results of your query by doing things like filtering out unnecessary records and creating calculated columns. The Power BI data model is like your final Hyperion Query section, where you would join all your local results (data sets) into one collection of data to answer business questions.

The major difference here is that the Power BI data model is not a query, but more like an OBIEE subject area where you define relationships to *facilitate* future reports and dashboards. Along with these relationships, the data model can also contain *measures* (written in DAX) which are based on values from multiple rows from one or more data sets (like our total credits example earlier). Measures are different from *calculated columns* (which can also be written in DAX), which are simple calculations performed on a single row (think of something like "% of Capacity" for a course, which divides enrollment by capacity for each course).

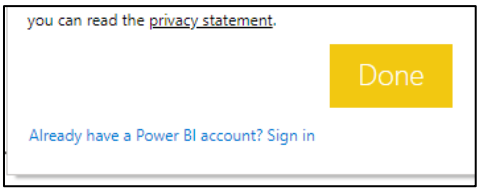
For more information about the differences between Power BI columns and measures along with specific examples, check out SQLBI's Calculated Columns and Measures in DAX article.

Creating a Power BI Data Set from SQL

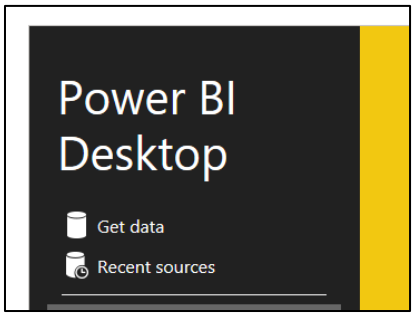
In this section, we'll cover the process of importing your SQL to Power BI Desktop and creating a data set, how you can use DAX and the Power BI GUI to create the same missing columns from before.

Setup and Importing Hyperion SQL

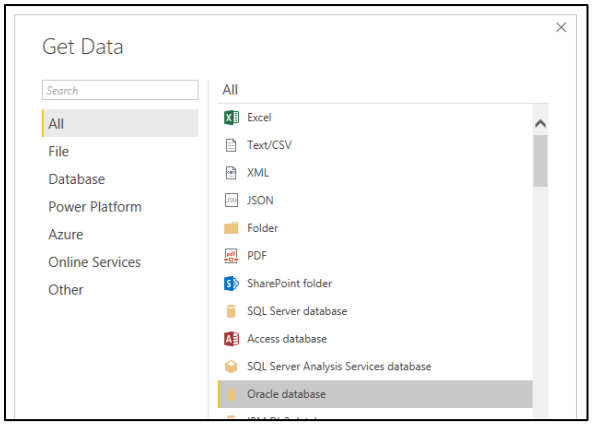
To begin the process, open Power BI Desktop. If this is your first time, you may need to login with your Office 365 credentials by clicking the **Sign in** text at the bottom of the popup window:



Then, enter your UWM email address in the window that appears. After doing so, you may need to enter your ePanther password to open the application. Then, click the **Get Data** link in the upper left of the Power BI Desktop splash screen:

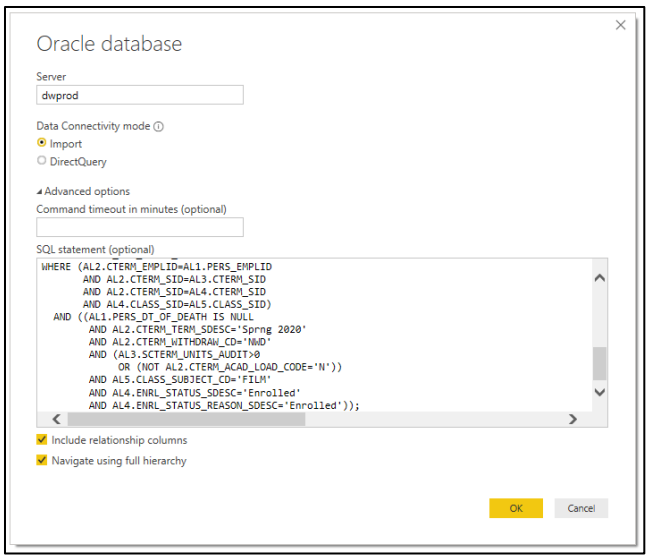


In the **Get Data** popup window, click to select the **Oracle database** option, and then click the **Connect** button.



On the next page, enter in **dwprod** as your Server – the server name on this screen will use the server name and other connection information from the production data warehouse entry in your **tnsnames.ora** file.

Next, click the triangle to the left of **Advanced options** to expand that section. In the SQL statement box, paste in the SQL statement from your Hyperion query:

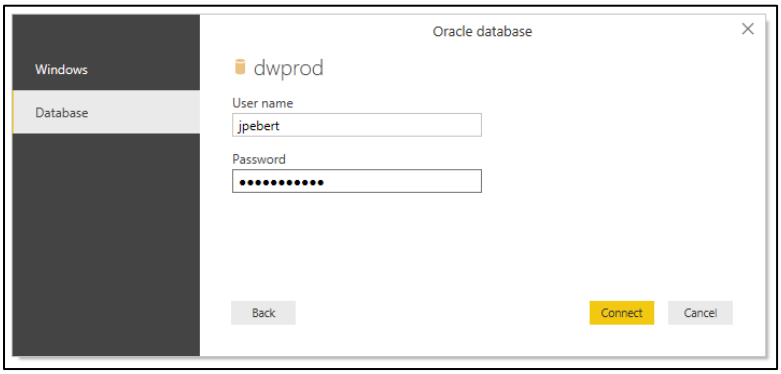


The **Data Connectivity mode** option in this window allows you to import a copy of all data from the table right away, or to directly query the table, showing a subset of live data from the table. Both methods have strengths and weaknesses depending on the size of your data set and the calculations you'll be performing – Microsoft has a great write-up about Using Direct Query in Power BI Desktop that may prove helpful in your selection.

For queries using our data warehouse, the **Import** option will likely be your best bet for a few reasons: 1) warehouse data will not change throughout the day; 2) relatively speaking for Power BI queries, the amount of data is likely to be small; 3) importing data allows Power BI to use its "high performance query engine" (rather than querying the database every time you add a computed item or change a filter) so you can develop your data sets faster. The import process might take more time, but you will save time as you transform your data, and you can always refresh your data later to capture changes.

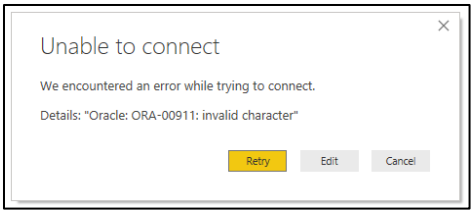
When you've selected the option for you, click the **OK** button to enter your credentials for the **dwprod** data source.

In the credentials window that appears, click the **Database** tab on the left side, and then enter your Oracle username and password for the data warehouse, and then click the **Connect** button.

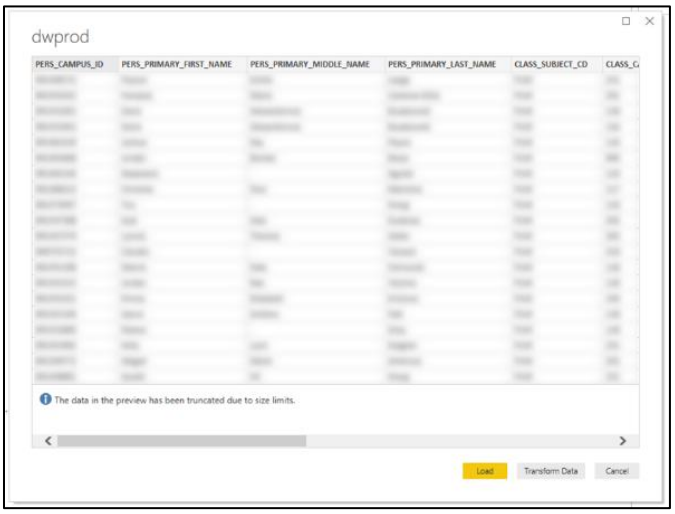


Note that you'll only need to provide the credentials for each data source once; after that, they are securely stored and will not need to be entered for subsequent connections to each connection (to view and modify your saved connection information, go to the **File** menu, select **Options and settings**, and click **Data source settings**).

As an aside, after pasting in my exact SQL statement from above, I received the following error:



This is caused by the semicolon at the end of the SQL statement copied from Hyperion. If we click the **Edit** button here and remove the semicolon, our SQL statement executes and a preview of our data is shown, along with the options to **Load** or **Transform Data**:



If you click the **Load** button, the data from your query will be added to the data model as a completed data set and you can use DAX expressions to add new columns and transform your data. If you click the **Transform Data** button, you can use M expressions to filter and transform your data set before it's added to the data model.

It's important to understand that the process described in this guide is not representative of typical Power BI query development. Because we are using a pre-defined query to recreate an existing data set, *less data transformation is necessary at this stage.* For example, we might want to apply filters from the Results section of our Hyperion query or create new calculated columns, but most of the transformation work (selecting fields and applying filters) has already been done in Hyperion.

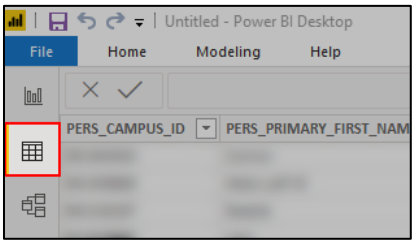
If you were creating a new Power BI query from scratch, you would likely begin by selecting one or more warehouse tables and then use the **Transform Data** option here. Using M (Power Query), you would eliminate unnecessary columns and add filters before adding those tables as data sets in your data model. After adding multiple data sets, you would define join relationships between them and use DAX to create additional calculated columns and measures using both data sets and/or user filters.

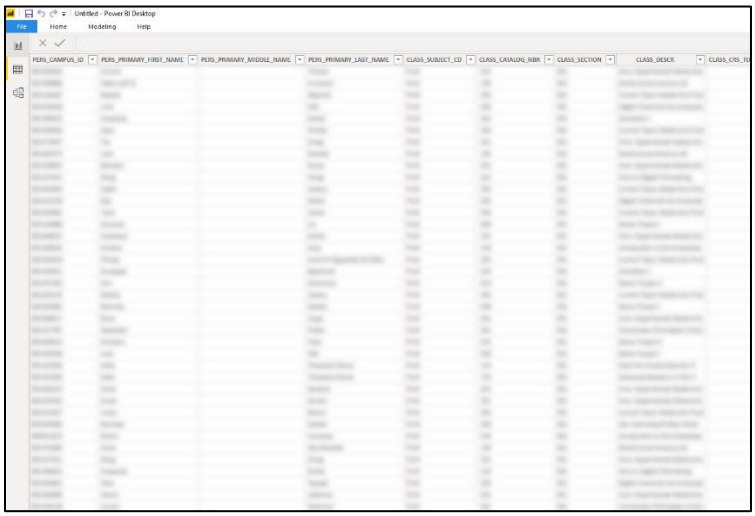
So, if you wanted to create a dashboard which would identify current students in a given academic plan, you would add VU_REC_ACAD_STRUCTURE_DIM and VU_UWM_ACAD_PLAN_TBL to your data model, transforming the ACAD_STRUCTURE data to only include "Active" students. In your data model, you would define that the tables should be joined on the value of ACAD_PLAN.

If you wanted to create a new Active Plan(s) column that would include all (active) plans for an Emplid, regardless of the user's filter selections, you would create that column in this **Transform Data** step. If, however, you wanted the Active Plan(s) column to respond to filter selections (e.g., if the user selected the School of Business, you would only return their active Business plans), then you would create that column in DAX after importing the data set to your data model.

For the purposes of this document, we will select the **Import** option here and move forward with demonstrating the process of creating columns in DAX, but please read this M or DAX: That is the Question article for more details about the differences between the languages and the best place to use each.

After a brief delay while your data is imported, you can view the query data in a "data sheet" format by clicking the **Data** icon in the upper left (shown to the right) to bring up all the rows from your query:





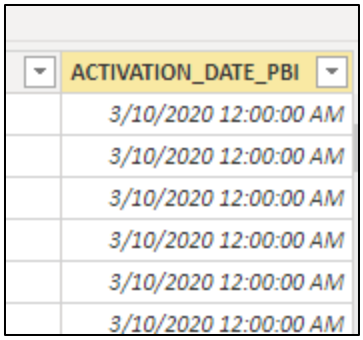
Creating Calculated Columns in DAX

As I mentioned earlier, you can also recreate any of your missing fields in Power BI after pasting in the original SQL statement directly from Hyperion. This section of the document shows how you can do that using Microsoft's DAX language (used in Power BI and Excel's Power Pivot). DAX is powerful, but also relatively easy to implement—Microsoft has some very good DAX documentation available and you can typically find Google search results to handle most common questions (not to mention the similarity to Excel formula syntax and helpful auto-complete functionality).

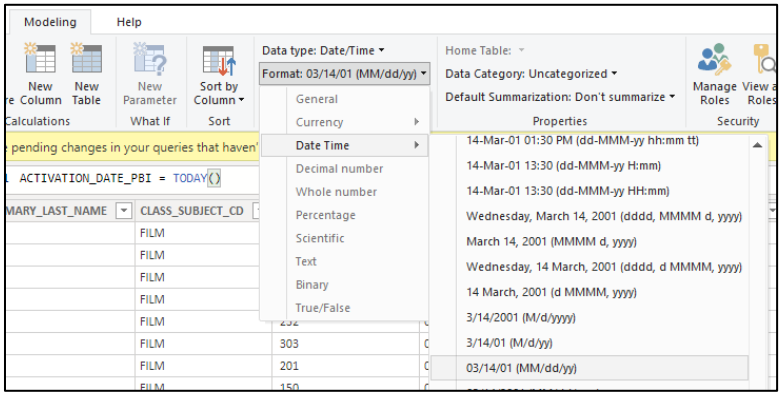
As we showed before with SQL, the syntax for these three columns in DAX is close to Hyperion:

Field Name	Hyperion Formula	DAX
ACTIVATION DATE	Sysdate()	TODAY()
EXPIRATION DATE	'05/31/20'	"05/31/20"
CLASS INFO	if (Class_Crs_Topic_Id_Ldesc != null) {Class_Subject_Cd + Class_Catalog_Nbr + '-' + Class_Section + ' '+ Class_Descr + ' (' + Class_Crs_Topic_Id_Ldesc + ')' } else {Class_Subject_Cd + Class_Catalog_Nbr + '-' + Class_Section + ' '+ Class_Descr }	IF(ISBLANK(Query1[CLASS_CRS_TOPIC_ID_LDESC]), Query1[CLASS_SUBJECT_CD] & Query1[CLASS_CATALOG_NBR] & "-" & Query1[CLASS_SECTION] & " " & Query1[CLASS_DESCR], Query1[CLASS_SUBJECT_CD] & Query1[CLASS_CATALOG_NBR] & "-" & Query1[CLASS_SECTION] & " " & Query1[CLASS_DESCR] & " (" & Query1[CLASS_CRS_TOPIC_ID_LDESC] & ")")

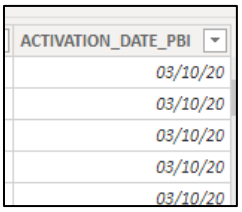
As before with the SQL current data column, our new column has more information than we want:



To change this, we can click the column heading to select the entire column, and then go to the **Modeling** tab on the top ribbon interface. Then, in the **Formatting** section, click on the Format dropdown and select your new date format:



And see that the format has updated:



This method is preferable to the SQL method of adding the date and changing formatting shown earlier, since the column type has not been changed from a date and this information can still be used for more advanced date-based reports and dashboards.

Analysis Tips and Building Queries in Power BI

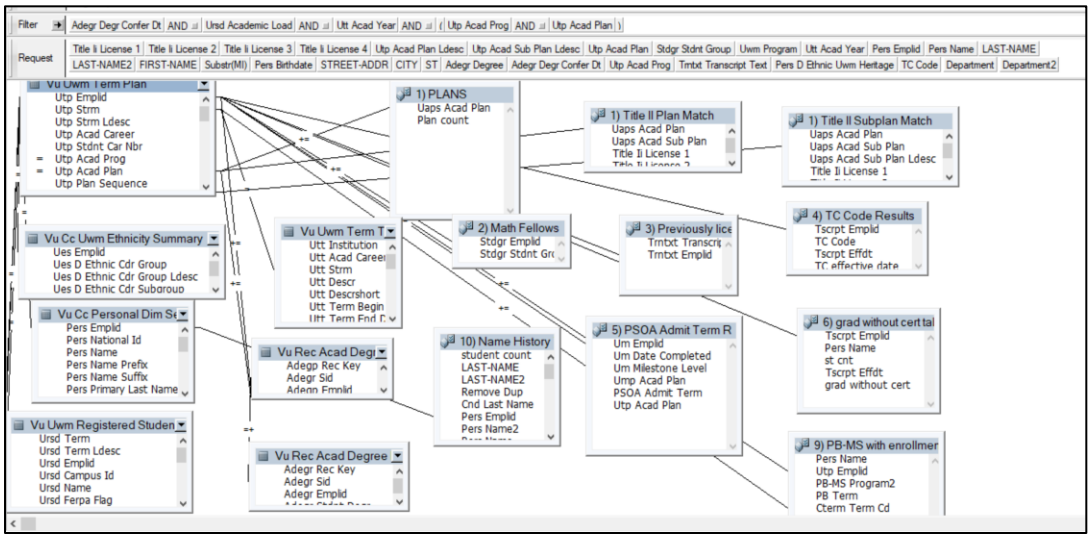
Now that we've spent 10 pages explaining how you *can* migrate your existing Hyperion queries into Power BI, let's look at some scenarios when you may want to take the longer route and rebuild, and some tips for analyzing your queries and learning how to rebuild them in Power BI.

Things to Consider

The method shown above is a great way to capture the logic and query parameters from simple Hyperion BQY files, but here are a few things you should consider before starting up Power BI and rolling up your sleeves:

- If you are unfamiliar with SQL **and** Power BI, you should probably rebuild in Power BI to minimize the number of new concepts and tools you will need to learn.
- If most of your logic and calculated columns are created in the Results section (or later!) in your BQY, you will be rebuilding most of your query in Power BI after importing the SQL anyway. Why not get practice with the Power BI interface while you still have a BQY file to test that you are getting the right results?

- If your query is something you are updating or running regularly with different filters, you should probably rebuild in Power BI. The Power BI interface is more forgiving than SQL, and the Power BI Query editor allows you to easily edit and "step through" your data changes by adding/removing and changing the order of the different transformations (column creations, sorts, filters, etc.) to manipulate your data.
- If you don't remember the purpose of half of the sections and/or your Hyperion query looks like this:



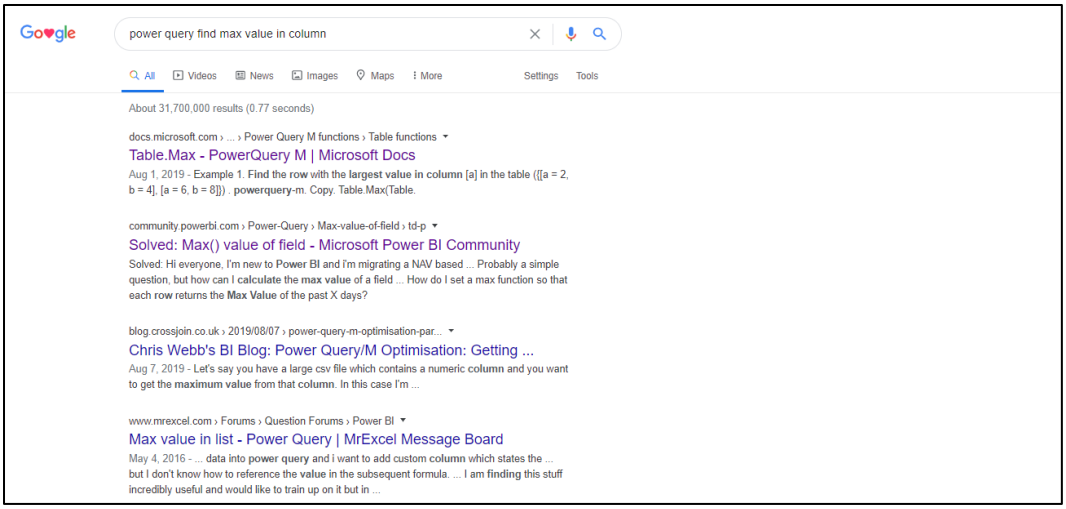
You should probably take this opportunity to rebuild your query and make sure it's still doing what you think it's doing—and start learning how to create queries from scratch using Power BI!

Four Steps to Analyzing and Learning (Power BI)

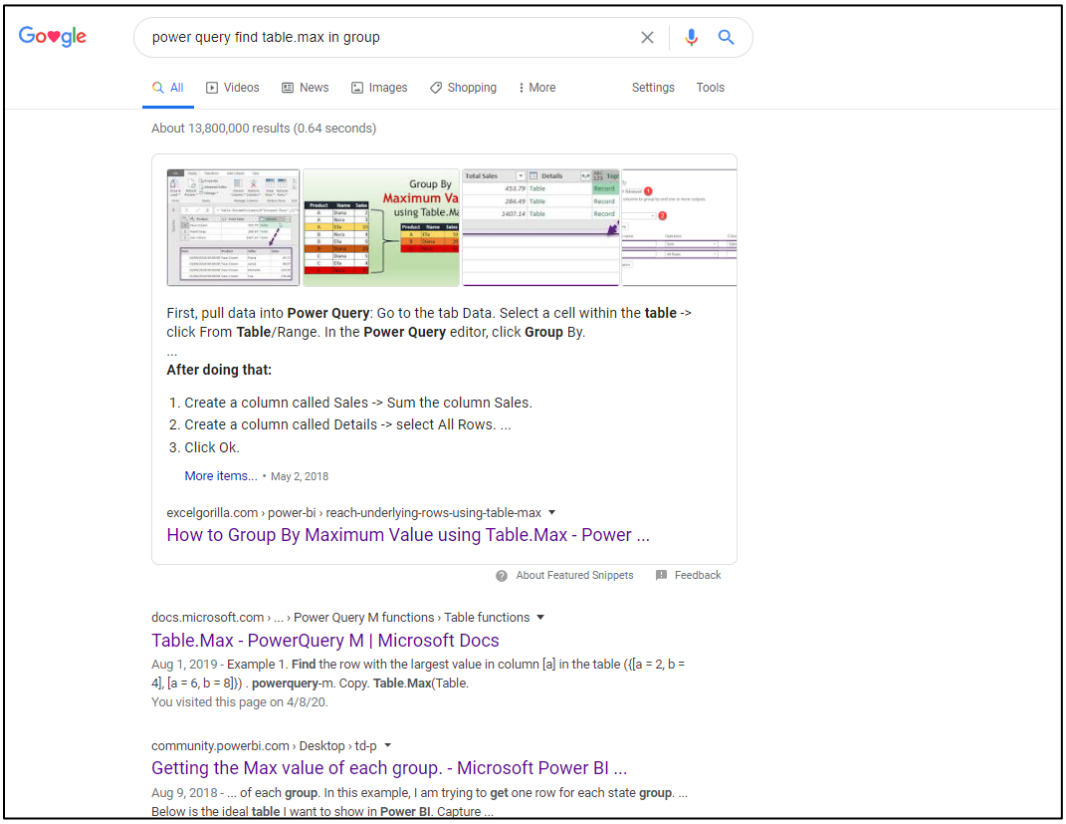
New tools will almost always provide more advanced functionality and techniques which enable different methods of problem-solving, and Power BI is no different. The best way to learn Power BI will be to figure out the core parts of your solution and the required steps, and then to use that understanding along with some internet sleuthing to figure out the best "Power BI way" to solve a problem, rather than simply applying the best "Hyperion translated to Power BI" method.

To help with that effort, I'm going to provide four steps I take for analyzing, understanding, and building technology solutions – they're applicable for Power BI queries, too!

1. **Gather the artifacts.** Bring in everything – BQY files, any external data sources (e.g., Excel files, Access databases), process documentation, and anything you know about the data request (past deliverables will also be helpful for testing the validity of your new queries). Familiarize yourself with the request from the starting point – at the least, this will be a good refresher and helps with the following step.
2. **Unpack the actions.** Figure out how to answer the business question that's being posed. If you have a BQY file, what are the steps you took to get the solution? Try to break these steps down into the basic actions (fetching fields, applying filters, sorting data, creating calculated items) and order. Then, go even deeper and try to unpack the logic behind each action, and the purpose it serves. For example, if your column returns the maximum term code for each student, why do you need this calculation? How is it helpful in getting the data you need? Keep in mind that the clearer you understand the steps in your process, the easier it will be to search for and identify new techniques and methods for completing those steps in your new query.
3. **Refine the terminology.** This is the most important, and most time-consuming, step of the process. With your newfound understanding of the core parts of your query, it's time for some internet sleuthing. So, if I wanted to find out how to recreate my max term per student query, I might start by Googling **power query find max value in column:**



Some basic reading of the results shows that we're on the right track, but that this method will find us the max value for the entire table, and not just the smaller group of rows for each Emplid. So, let's refine our query a bit to include the **Table.max** function and the **group** keyword. Let's try **power query find table.max in group**:



Which links us to the helpful [How to Group By Maximum Value using Table.Max] article (which references another helpful article about grouping in Power BI to create aggregated data sets).

This is a super simple example of the theme for this step – combine your new understanding of the current action with simple keywords to get *close* to the right answer. Then, read your results and see if you've found a solution – if not, refine your search to add any new terminology that seems to be closer to what you need.

As you perform this step more often, you will also come to recognize certain sites which come up more frequently and tend to have higher-quality results and explanations which work for you. Trust these sites and use them to identify other techniques which might be even better than the ones you're using now.

4. **Practice the techniques.** I'm going to share a secret with those of you who are still reading: you can't learn how to build Power BI queries by reading blog posts or watching videos. The only way to learn the Power BI query languages is to apply the techniques you find to actual business needs.

   Sometimes you might find a solution in the previous step that doesn't do what you expected or solve your current problem. However, you can probably learn something about the tool or the data from this misstep or find a way to share your experience and knowledge with the rest of the reporting community and help someone else who's struggling with the same situation.

   The bottom line is that your journey to learning Power BI is going to be an iterative process that requires a lot of experimentation, analysis, and community-building – in the same ways that the campus learned Hyperion!

Power BI Resources

There is a wealth of information about how to build on the basic SQL import process shown in this document with additional columns, joining between different sets of "local results," and building visualizations based on that data. Here's a list of some of these resources:

If you're a video learner…

- Microsoft Power BI Guided Learning (all of these are great, but pay special attention to Model data in Power BI, Use visuals in Power BI, and Introduction to DAX to build on the Hyperion conversion shown in this guide)
- SQLBI's free Introduction to Data Modeling for Power BI and Introducing DAX courses are another excellent way to build on the basic concepts in this guide.
- For a ground-up introduction to Power BI functionality and features, check out one of the many LinkedIn Power BI courses (formerly Lynda.com, UWM has a free subscription)
- If you'd prefer a project- and assignment-focused introduction to the tool, check out the Analyzing and Visualizing Data with Power BI course from EdX (no charge, but requires a free account)

If you'd prefer to read…

- Microsoft's Power BI documentation covers everything from the basics to data modeling and creating dashboards with your data set: https://docs.microsoft.com/en-us/power-bi/
- Microsoft's DAX Reference: https://docs.microsoft.com/en-us/dax/
- A comparison of SQL and DAX, and basic DAX querying: https://www.wiseowl.co.uk/blog/s2480/dax-query.htm
- DAX variables: https://docs.microsoft.com/en-us/power-bi/guidance/dax-variables
- Microsoft's guide to Transforming Data with M: https://docs.microsoft.com/en-us/powerquery-m/
- SQLBI (https://www.sqlbi.com/) is an excellent resource for beginner-friendly training videos, but they also have regular blog posts about new Power BI features and deep dives into the more powerful functionality of the tool.
- The DAX Guide: https://dax.guide/ -- the benefit of this resource over the Microsoft reference above is that it provides links to SQLBI posts and articles which cover the DAX function you're investigating.