ASPE-4 Standardization and Querying of Data Quality Metrics and Characteristics for Electronic Health Data Project

Technical Documentation

Prepared by: Sentinel Coordinating Center

**December 31, 2019**

ASPE-4 is funded by the US Food and Drug Administration through the Department of Health and Human Services (HHS) Contract number HHSF223201400030I / HHSF22301006T.

ASPE-4 DATA QUALITY METRICS

Technical Documentation

**Table of Contents**

[I. background 1](#_Toc27738350)

[A. system overview 1](#_Toc27738351)

[B. Key Functional Components 2](#_Toc27738352)

[1. Metrics 2](#_Toc27738353)

[2. Measures 2](#_Toc27738354)

[3. Explore DQM 2](#_Toc27738355)

[II. data model and entities 2](#_Toc27738356)

[C. Data Quality Data Model 3](#_Toc27738357)

[D. Entity details 5](#_Toc27738358)

[III. Website configuration settings 7](#_Toc27738359)

[E. Configuration Settings 8](#_Toc27738360)

[IV. developer set up 10](#_Toc27738361)

[A. DQM Application Requirements 10](#_Toc27738362)

[B. DQM application instructions 11](#_Toc27738363)

[C. PopMedNet Application Requirements 11](#_Toc27738364)

[D. popmednet application instructions 12](#_Toc27738365)

[V. Appendix 13](#_Toc27738366)

[A. DQ Metrics & DQ Measures Load Script Details 13](#_Toc27738367)

[B. Registering a sheet in the dqm site 24](#_Toc27738368)

# background

The goal of Data Quality Metrics project and system was to provide a harmonized approach to data characterization across multiple data sources to enable researchers to better understand candidate data sources before querying and analyzing them. This work included the creation of a system that operationalizes existing data quality (DQ) parameters and methodologies in a way that is compatible across multiple Common Data Models (CDMs) to increase research planning efficiency and improve the interpretability of analytic results.

We created and implemented a data quality data model to contain a set of metadata standards and metrics describing: 1) Data quality and characteristics; 2) Data sources and institutional characteristics; and 3) Fitness-for-use. These standards were the basis for a flexible data quality collation system that is able to incorporate data metrics from any data source. The system was designed to enable flexible exploration of DQ characteristics for multiple data sources at the same time.

Together, the information contained in the data model provides a standardized data source “fingerprint” that can be expanded to provide additional granularity. Additionally, the DQM system was enabled to maintain and query the data model and is available as open source web-based technology such that the system provides approaches to access the data model and can use any business intelligence tool of choice to interact with the data and explore and describe the quality, completeness, and stability of data sources. This Technical Documentation report is intended for technical stakeholders who have expertise in electronic health data resources and/or software development processes.

## system overview

We proposed a pragmatic approach to developing consistent data quality metrics through development of an extensible data model based on a collection of data quality standards and metrics included in the Harmonized Data Quality framework put forth by Kahn et al1. An extensible data quality data model must be flexible and independent of the source data model. The Kahn framework describes and defines data quality standards and metrics in a general and harmonized fashion and this system applies it to a variety of data sources and research needs. Operationalizing that framework and developing a tool for analyses allows researchers to evaluate data quality at any life stage of a data source in a consistent manner, and to effectively compare data sources based on the same metrics. A standard data quality metric data model will assist researchers in determining fitness-for-use of various data sources and research purposes.

We have demonstrated our “data fingerprinting” system using synthetic data sets that reflect those used by existing networks, such as PCORnet and Sentinel, with consideration as to how our system can be used by an open network where anyone can review, contribute to, and utilize the DQ data model and explore database fingerprints approved for public consumption— a priority interest for the NIH community and others2-6{, #94;Curtis, 2014 #52;Fleurence, 2014 #51;Vogel, 2014 #76;, 2018 #93}.

Although several groups and researchers have done thorough evaluations of DQ metrics for specific data sources (e.g., birth defect surveillance systems, primary care data, medical registries), to our knowledge there is not currently a data model in place for generic quality measures that can be tailored to specific data sources 7-12. While study-specific data characterization work provides a framework to evaluate data, it lacks a focus on extensibility and generalizability. Our model will enable users to add any data quality metric of value from their work, thus expanding the initial DQ metrics included in this reference implementation.

We have articulated 78 **use cases**, and the implemented version of the data model captures 25 items of interest (metadata) describing the source system and its measures, as well as 15 items of metadata describing each metric. This information informed the development of the data quality data model and design of the DQM system. Based on the use cases and review of current data quality standards, we identified the following structures to contextualize the quality of data:

* Time component (e.g., number of encounters by clinical setting per year)
* Person-based construct (e.g., number of prescriptions ordered per person per year)
* External context (e.g., rates of asthma by age compared to expected population rates)

## Key Functional Components

### Metrics

Metrics are the descriptions of quantitative measurements that can be executed on data sources to characterize a specific aspect of the source data in a data model agnostic way. The DQM tool captures metadata about each Metric in a standardized way, regardless of the context or use cases. Metric authors describe the metric in enough detail for a data holder to interpret and generate the results of the Metric from their source data. These results, or measures, enable apples-to-apples comparisons across data sources irrespective of the CDM or data structure.

### Measures

A Measure is the numeric representation of a metric that has been executed against a data source. Measures include the data characteristics defined in the metric, as well as metadata about the data source, metric details, and information regarding when the measurement was calculated. The Measures can be explored in the visualization tools found in Explore DQM.

### Explore DQM

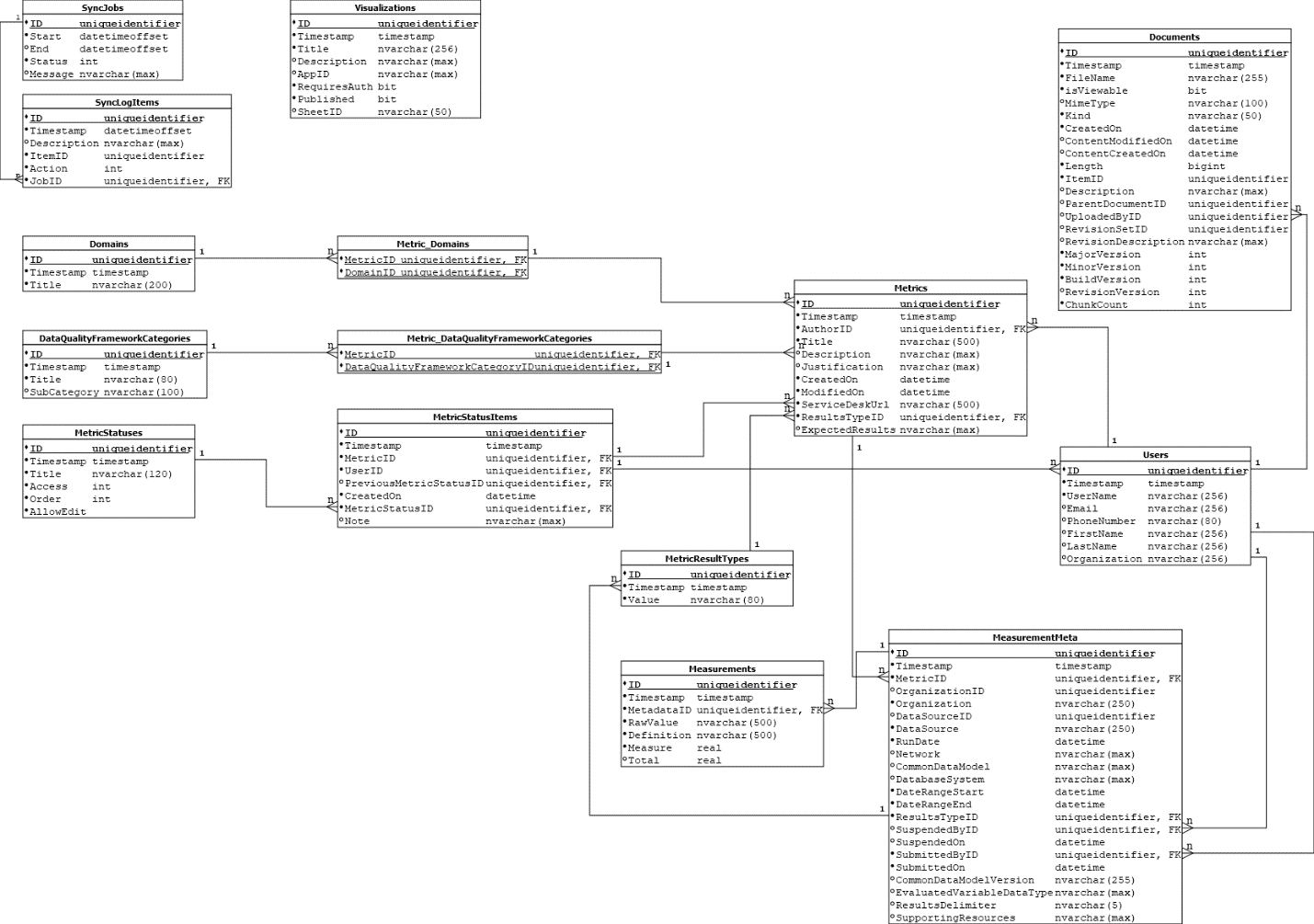
The DQM visualization tools overlay the metadata, metrics, and measures. Users can explore and evaluate data sources for specific characteristics, trends, and quality. DQM does not determine whether a data source passes or fails the executing of a metric, but rather provides a view of data characteristics that enable a user to determine if the data are fit for their purpose.

Additional details on implementation of the visualization tools can be found in documentation developed by Analytics8 – a data and analytics consulting firm that engaged in the work – in the appendix.

# data model and entities

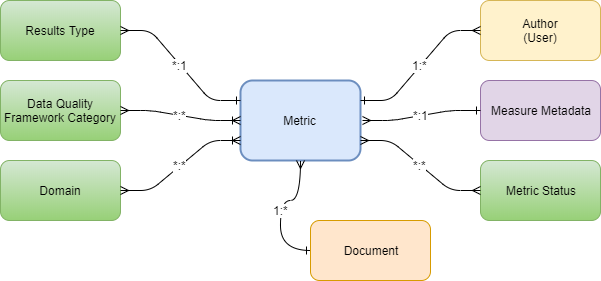
Following iterative design discussions, a final data quality data model was implemented as the underlying structure of the system.

## Data Quality Data Model



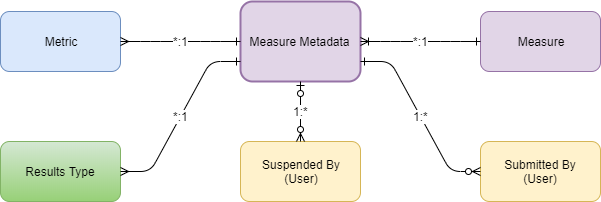
* Solid dots indicate non-nullable fields.
* Underlined fields indicate Primary Keys.
* Relations are indicated by the connecting lines and their connectors.
* All non-collection tables have a primary key that is named ID.
* A non-nullable timestamp field is included on all tables that require optimistic concurrency for Entity Framework.

The root entities are Metric and Measure Metadata; all other entities support defining attribute of those entities. Entity relationships are depicted below in figures 1 and 2 and further detailed in [Section D](#_Entity_details).



**Figure 1. Metric Entity**

* A User can author zero or more Metrics. A metric must have an author.
* A Metric has a collection of statuses, each status item is immutable.
  + A new status item is created for each status change, and the most current item is the current status of the Metric.
  + A metric status item contains the date the status changed, the status, the User that changed the status, a reference to the previous status item, and an optional note regarding the status change.
* A Metric has a single Results Type association. A Results Type can be associated to more than one Metric.
* A Metric has one or more Data Quality Framework Category associations. A Data Quality Framework Category can be associated to more than one Metric.
* A Metric has one or more Domain associations. A domain can be associated to more than one Metric.
* A Metric has zero or more Measure Metadata associations. Measure Metadata must be associated to a Metric.
* A Metric has zero or more Document associations. A document must be associated to an entity.



**Figure 2. Measure Metadata Entity**

* Measure metadata represents the metadata about a collection of measures.
* Measure metadata must be associated to a single Metric. A Metric can be associated to more than one Measure metadata.
* Measure metadata must be associated to a single Results Type. A Results Type can be associated to more than one Measure metadata.
* Measure metadata is associated to more than one Measure. A Measure must be associated to a single Measure metadata.
* Measure metadata must be associated to a single User representing who submitted the measure data. A User can be associated to more than one Measure metadata as the submitter.
* Measure metadata may have an association to a single User representing who suspended the measure data. A User can be associated to more than one Measure metadata as the suspender.

## Entity details

|  |  |
| --- | --- |
| Entity | Details |
| User | * Represents a "person" * Requires a UserName, optionally: a first and last name, email address, phone number, and associated organization name |
| Results type | * Indicates the Results Type of a Metric, and/or Measure * Comprised of a display title * Can be associated with many Metrics |
| Data Quality Framework Category | * Indicates the category a Metric could be classified as   + The category classifications are based on definitions defined by the Khan framework. * Comprised of a Title and optional Sub-category * Can be associated with many Metrics |
| Domain | * Indicates the domain a Metric belongs to   + A domain is comprised of a title. * Can be associated with many Metrics |
| Metric status | * The definition of a status a Metric can be assigned   + Comprised of a title, an access level, a logical order value, and if editing of the Metric is allowed while in the status * The access levels define which users have access to a Metric, and are comprised of the following values:   + None = no access level specified   + Author = only the author of the Metric has access   + System Administrator = only Users with the System Administrator claim can access the Metric   + Authenticated Users = only Users who have been authenticated can access the Metric   + Public = any User can access the Metric |
| Metric status item | * The instance of a status for a Metric   + Comprised of the Metric, User, Metric Status, the previous Metric Status, Creation date, and a note * A Metric will have one or more status items; the one with the most recent creation date is the current status. |
| Metric | * The definition of a Metric is comprised of:   + Title, Description, Justification, Expected Results, Created On and Modified On dates, Service Desk URL   + An Author - the User creating the Metric   + ResultsType   + One or more Data Quality Framework Categories   + One or more Domains   + One or more Metric Status Items   + Zero or more Measures (Measure Metadata) |
| Measure Metadata | * Represents the metadata about a collection of Measures * The definition of a Measure Metadata is comprised of:   + A Metric; Measures are the quantitative result of a query based on a Metric definition   + Organization name, and optionally it's ID   + DataSource name, and optionally it's ID   + A run date for when the data was collected   + The network the Data Source belongs to   + The Common Data Model the data may belong to   + The Database System the data was stored in   + Date Range Start is the earliest date of the data set   + Date Range End is the latest date of the data set   + Results Type ID, the ID of the Results Type associated to the Measures. Must match the Results Type defined on the associated Metric.   + Suspended By, the User who suspended the Measures excluding it from available queries   + Submitted By, the User who uploaded the Measures to DQM   + Common Data Model Version, the version number of the CDM the Measure data may belong to   + Evaluated Variable Data Type   + Results Delimiter, the delimiter used if the values of the Measures are compounded and the result of more than one value.   + Supporting Resources, a URL to a location providing resources (application, scripts, documentation, etc.) used to obtain the measures.   + A collection of one or more Measures |
| Measure | * Represents the instance of a Measure * Comprised of:   + Raw Value represents the unmodified value of the stratifier the measure is for   + Definition represents a display value for the Raw Value: i.e. Raw Value = 'M' and the Definition = 'Male'   + Measure is the numerical quantity of the result. Depending on the Results Type defined by the Metric it could be a count, percentage, range, or vector.   + Total is the optional value representing the total of all the Measure values, it could be greater than the sum of the Measure values included in the |

# Website configuration settings

The website application and web jobs use the standard ASP.Net Core configuration framework to manage and access application configuration settings. The default base configuration file (appsettings.json) contains the default configuration values; the local developer base configuration settings are located in appsettings. Development.json with local configuration values overridden via the Debug environment variables found in the project properties. The local settings are stored in the launchSettings.json file for the specific launch profile, each developer should create their own launch profile.

Settings for the Azure deployed application are specified as environment variables within the Azure App Service configuration.

The configuration files are specified using json in a hierarchical structure. The hierarchical path of a specific setting can be stated by delimiting the path using a colon.

Example default configuration found in appsettings.json.

{

  "Logging": {

    "LogLevel": {

      "Default": "Warning"

    }

  },

  "AllowedHosts": "\*",

  "PMNApiUrl": "",

  "PMNPortal": "",

  "PMNoAuthKey": "",

  "PMNoAuthHash": "",

  "QlikServer": "",

  "QlikServerQPSPort": "4243",

  "QlikQPSPrefix": "",

  "QlikUserDirectory": "",

  "QlikUserID": "",

  "QlikQPSCertThumbprint": "",

  "QlikCertLocation": "",

  "Files": {

    "Type": "ASPE.DQM.Files.LocalStorageFileService, ASPE.DQM.Files",

    "UploadDirectory": "",

    "StorageConnectionString": "",

    "FileStorageShare": "",

    "DataLakeStorageAccountName": "",

    "DataLakeStorageClientID": "",

    "DataLakeStorageClientSecret": "",

    "DataLakeStorageTenantID": "",

    "DataLakeStorageDirectory": ""

  },

  "ConnectionStrings": {

    "IdentityContextConnection": ""

  },

  "Sync": {

    "ServiceKey": ""

  }

}

## Configuration Settings

| Setting Key | Description |
| --- | --- |
| **Logging:LogLevel:Default** | Specifies the logging level by default for system logging. |
| Logging:LogLevel:{namespace[classname]} | Specifies the logging level for a specific namespace within the source. Examples include: "System", and "Microsoft" |
| Serilog:\* | The configuration settings for Serilog. Refer to <https://github.com/serilog/serilog-settings-configuration> for documentation. |
| **AllowedHosts** | See: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-2.2#host-filtering-1> |
| PMNApiUrl | The url to the API for the CNDS PMN instance. |
| PMNPortal | The url to the SSO login endpoint of the CNDS PMN portal instance. |
| PMNoAuthKey | The oauth authentication key for interacting with the PMN single sign-on. |
| PMNoAuthHash | The security hash for interacting with the PMN single sign-on. |
| QlikServer | The root domain of the Qlik server. Does not include the http scheme or trailing slash. |
| QlikServerQPSPort | The port of the QPS for the Qlik installation. |
| QlikQPSPrefix | The url prefix of the Qlik proxy |
| QlikUserDirectory | The user directory for Qlik authentication. |
| QlikUserID | The ID of the Qlik user DQM will use for impersonation. |
| QlikQPSCertThumbprint | The thumbprint of the certificate used to validate the connection to the Qlik server |
| QlikCertLocation | The certificate installation location, default is "LocalMachine" |
| **ConnectionStrings:IdentityContextConnection** | The SQL Server connection string to the DQM database. |
| Sync:ServiceKey | The authentication key used for the CNDS/DQM user synchronization service. |
| **Files** | Configuration settings for file storage. Required settings depend upon the type of file storage. |
| **Files:type** | The type of file storage provider to use. Default is local file storage. The provider type is specified as "class name, assembly name". |
| Files:UploadDirectory | The path to the folder files should be saved. Required for LocalStorageFileService. |
| Files:StorageConnectionString | The connection string to the Azure storage account. Required for AzureBlobStorageFileService, and AzureFileStorageFileService. |
| Files:FileStorageShare | The Azure storage share key. Required for AzureBlobStorageFilesService, and AzureFileStorageFileService. |
| Files:DataLakeStorageAccountName | The Azure Data Lake storage account name. Required for AzureDataLakeFileService. |
| Files:DataLakeStorageClientID | The Azure Data Lake storage account client ID. Required for AzureDataLakeFileService. |
| Files:DataLakeStorageClientSecret | The Azure Data Lake storage account client secret. Required for AzureDataLakeFileService. |
| Files:DataLakeStorageTenantID | The Azure Data Lake storage account tenant ID. Required for AzureDataLakeFileService. |
| Files:DataLakeStorageDirectory | The Azure Data Lake storage account directory name. Required for AzureDataLakeFileService. |

\* Settings that have their key in **bold** are required.

# developer set up

## DQM Application Requirements

* Windows 10
* Microsoft Visual Studio 2017+, all editions supported
* Microsoft SQL Server 2014 or greater
* WebPack Test Runner for Visual Studio by Mads Kristensen, not required but makes running WebPack builds much easier.
* NodeJS
* Typescript SDK
* .NET Core 2.2 SDK

## DQM application instructions

1. Install Visual Studio, and apply any updates.
   1. Confirm the ASP.NET and web development option has been selected
   2. Confirm that .NET Core 2.2 is selected if available.
2. Install .NET Core SDK if not installed via Visual Studio.
3. Install SQL Server, and apply any updates. Make sure the current Windows User is authorized for the database, and Integrated Security is enabled.
4. Install Typescript SDK found at <https://www.typescriptlang.org/#download-links>
5. Install NodeJS found at <https://nodejs.org/en/download/>
6. Install the WebPack Test Runner from the Visual Studio Extensions gallery.
7. If support for Qlik applications is required, install the Qlik certificate into the Local Computer store
   1. Certificate and instructions are found in ~/QlikCert folder of the the source
8. After installing the software dependencies and obtaining the source code for the application, the ASPE.DQM.sln can be opened using Visual Studio. Perform a build only of the solution and confirm all projects compiled successfully. Open the Task Runner Explorer panel from the "View => Other Windows" menu, under the webpack.config.js item expand "Run" and double click the "Run-Development" option. This will initiate the WebPack build which will compile the typescript, placing the output into the wwwroot/scripts folder of the web application.
9. If an existing copy of the DQM database is available, restore the database to SQL Server with the name "ASPE\_DQM".
10. If starting without a copy of the DQM database, it can be created by running the migrations via the Package Manager Console in Visual Studio.
11. At this point the DQM web application can be launched using IIS Express via Visual Studio.

## PopMedNet Application Requirements

For the DQM project, the final CNDS version was used. Any version greater than 6.2 of PopMedNet is compatible.

* Windows 10
* Microsoft Visual Studio 2017+, all editions supported
* Microsoft SQL Server 2014 or greater
* Typescript SDK version 3.2
* ASP.Net MVC 4 if the PopMedNet version is less than 6.12.0.0
* RazorGenerator extension for Visual Studio (<https://github.com/RazorGenerator/RazorGenerator>), only required if making changes to .cshtml files
* Less compiler, only required if making changes to .less files
* .NET SDK 4.7.2

## popmednet application instructions

1. PopMedNet is used by DQM to manage User registration, and user permissions. No development is required for the usage and integration of PMN with DQM. The PMN instance can either be run via IIS Express using Visual Studio, or it can be compiled and deployed to an IIS instance.
2. After installing the software dependencies, and obtaining the source for the application, the PMN websites are ready to be built and optionally deployed.
3. Restore a compatible version of the PMN and CNDS databases to SQL Server, update the connection strings in the ConnectionStrings.config files found in the Lpp.Dns.Api, Lpp.Dns.Portal, and Lpp.CNDS.Api project folders. The ConnectionStrings.config can be created by making a copy of the ConnectionStrings-template.config file, and should not be added to source control.
4. Open the Lpp.Dns.Api solution with Visual Studio and build the entire solution. Using the Package Manager Console in Visual Studio confirm the PMN database is up to date by executing any pending migrations.
5. Open the DistributedNetworkSolution solution with Visual Studio and build the entire solution.
6. Open the Lpp.CNDS solution with Visual Studio and build the entire solution. Using the Package Manager Console in Visual Studio confirm the CNDS database is up to date by executing any pending migrations.
7. The CNDS website is only required if CNDS integration is part of the PMN instance being used. DQM does not have a dependency on CNDS, only PMN.
8. After confirming the solutions compile without errors, the websites can be run using IIS via Visual Studio or by publishing to a local folder and configuring websites in an IIS instance.
9. Depending upon how it is desired to run PMN; confirm that the correct url's are configured in the DQM appsettings.Development.json file. The PMNApiUrl value should be the root url of the PMN API website (i.e. http://localhost:24592), and the PMNPortal value should be the url to the ssologin action for the PMN Portal website (i.e. <http://localhost:60344/ssologin>).
10. Confirm that the PMNoAuthKey and PMNoAuthHash values in the DQM configuration settings match the values specified in the Lpp.Dns.Portal/web.config for the settings SsoKey and SsoHash. The PMN SSO site does not need to be used, however DQM uses the SSO infrastructure in the PMN Portal site to enable cross-application authentication.

# Appendix

## DQ Metrics & DQ Measures Load Script Details

Harvard Pilgrim Health Care

DQMetrics & DQMeasures Load Script Details



October 16, 2019

written by: Chris domain

cdomain@analytics8.com

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 10/16/2019 | 1.0 | Initial Document Creation | Chris Domain |

**DOCUMENT OVERVIEW**

This document provides details on all the load scripts used in the DQMetrics Final and DQMeasures final applications. For each app I will be describing what each script is being used for and how it affects the final application.

**DQ METRICS APPLICATION**

**API/REST Connections:**

The DQMetrics Application pulls data from five separate API’s using five rest connectors. Below I’ve listed the names of the rest connectors as well as the API URL’s that they are connected to:

REST\_METRICS:<https://dataquality.healthdatacollaboration.net/api/qlik-export/metrics>

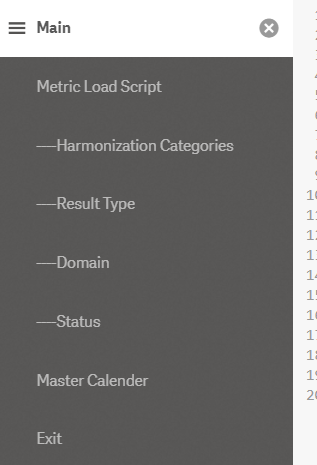
REST\_HARMONIZATION\_CATEGORIES:<https://dataquality.healthdatacollaboration.net/api/qlik-export/data-quality-harmonization-categories>

REST\_RESULTS\_TYPES:<https://dataquality.healthdatacollaboration.net/api/qlik-export/results-types>

REST\_DOMAINS:<https://dataquality.healthdatacollaboration.net/api/qlik-export/domains>  
REST\_STATUSES: <https://dataquality.healthdatacollaboration.net/api/qlik-export/metric-statuses>

From here on out I will be referring to the connections by their rest connector names.

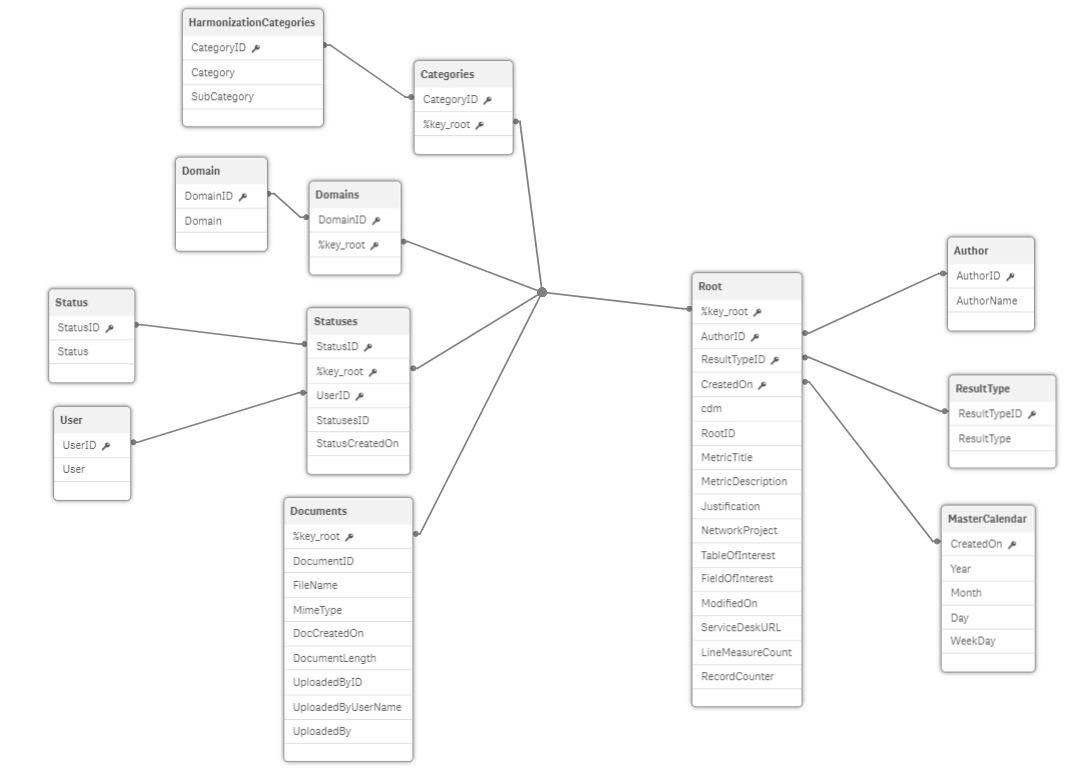
**Load Script Overview:**



The DQMetrics script is broken up into eight sections. **Main** initializes the qlik sense settings. **Metric Load Script** connects using the REST\_METRIC connector, it pulls in seven tables: Root, Author, User, Statuses, Categories, Domains, and Documents. **----Harmonization Categories** connects using the

REST\_HARMONIZATION\_CATEGORIES connector, it pulls in one table: HarmonizationCategories. **----Result Type** uses the REST\_RESULT\_TYPES connector, it pulls in one table: ResultType. **----Domain** uses the REST\_DOMAINS connector, it pulls in one table: Domain. **----Status** uses the REST\_STATUSES connector, it pulls in one table: Status. **Master Calendar** creates and additional table MasterCalender used for date visualizations. Finally **Exit** just contains the Exit Script to stop the script.

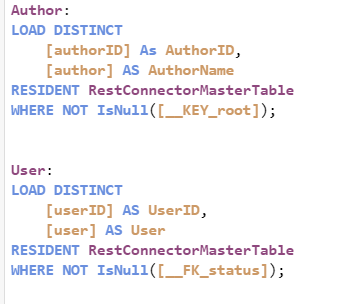
The final data model looks like this:



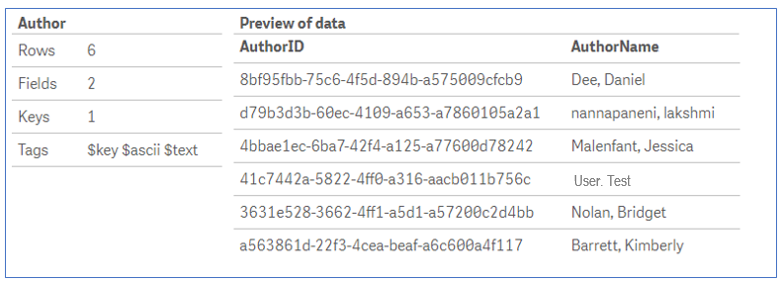
**Metric Load Script:**

In this section all the data from REST\_METRICS is pulled into a temporary table named

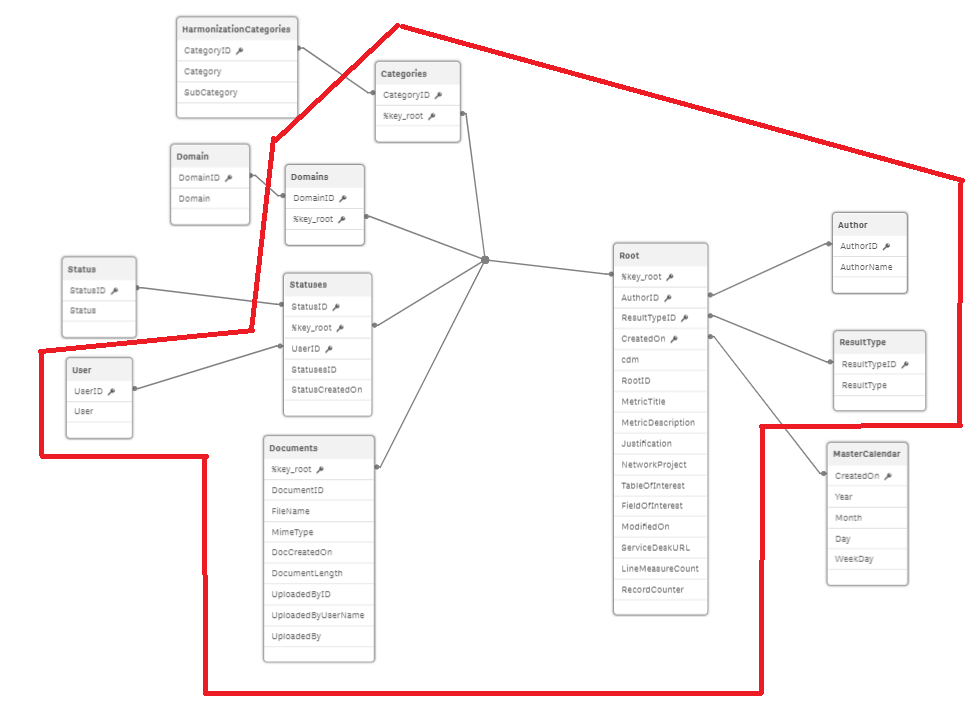
RestConnectorMasterTable, the seven final tables are created using resident loads from the master table. Once the seven final tables are created the RestConnectorMasterTable is dropped. In this section the only editing done mostly is by renaming fields. In the Root, Categories, Domains, Statuses, and Documents tables I have renamed their key values to %key\_values, this is how the supporting tables are linked to the Root table. At the bottom of the Root table you will see “1 AS RecordCounter”, the one measure in this application (# Metrics) sums this field to get the count of Metrics. Summing is more efficient than counting in qlik.



The only other additions to this section is the creation of the Author and User tables. I created these tables as a distinct load so all the Authors/Users can be visualized in one place. Knowing this information helps with visualization creation as well as for filtering, the tables look like this:

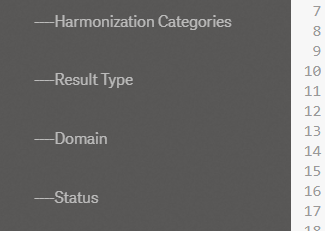


This section alone is responsible for this portion of the data model:

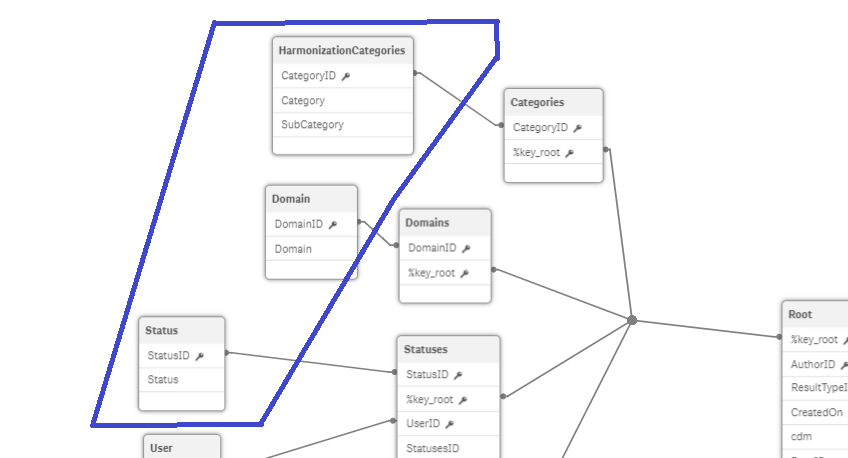


The rest of the sections are supporting tables used for filtering and visualization enhancement.

**The ‘----’ Indented Sections:**



The four sections above pull data from the other four rest connectors: REST\_HARMONIZATION\_CATEGORIES, REST\_RESULTS\_TYPES, REST\_DOMAINS, REST\_STATUSES. Each one comes with two fields, [id] and [title] (harmonization categories has an extra for subcategory). The [title]s are renamed to what their value represents and the [id]s are renamed to match the ID in the metric tables: DomainID, StatusID, etc.



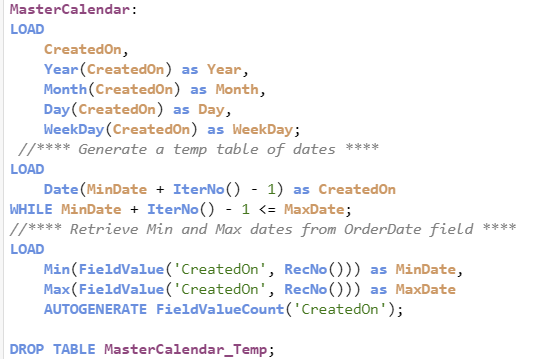
As seen above in the data model, these tables sit on the peripheral of the schema connected by their renamed ID’s. They exist to show all the possible values a given category, domain, or status can be regardless if those values exist in the metrics data. These tables allow us to have a complete view of possible values and is important when it comes to filtering and creating visuals later on.

**Master Calendar:**

The master calendar is the last section in the metric script. It’s connected to the Root table by the CreatedOn date field. The way it works is by finding the minimum and maximum date in the CreatedOn dataset. It then fills in a table with every single day between the min and max date to create a full date dataset.



The rest of the script is just for formatting. The reason we use a master calendar is so we have all the date values in a given time regardless of whether or not data was gathered on that day. In the application we use the master calendar CreatedOn value in our visualizations instead of the one from the root table. It allows line charts or any other chart of date vs value to be distributed properly across a time span instead of clumping the dates together.



**In Analysis:**

All the fields used to create visualizations in this application I have made as master dimensions and measures. When editing a sheet in qlik sense you can go to the left side of the screen and click on master items below the fields tab. There are seven master dimensions and one master measure. They are the only fields I used to create every visual in this app.

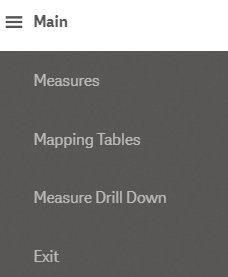
**DQ MEASURES APPLICATION**

**API/REST Connection:**

The DQMeasures Application pulls data from one API. The rest connector name and API URL are listed below:

REST\_MEASURES\_BY\_METRICS:<https://dataquality.healthdatacollaboration.net/api/qlik-export/measures-by-metric>

**Load Script Overview:**

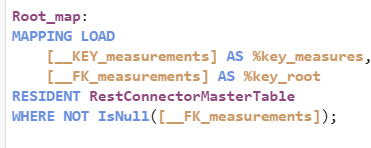


The DQMeasures script is broken up into five sections. **Main** initializes the qlik sense settings. **Measures** connects using the REST\_MEASURES\_BY\_METRIC connector, it pulls in three tables: Root, Metadata, and MeasuresMaster. **Mapping Tables** contains all the mapping tables used in Measure Drill Down. **Measure Drill Down** are resident tables based off the MeasuresMaster table, and are used to create visualizations focused on a particular type of measure. Finally **Exit** just contains the Exit Script to stop the script.

**Measures Section:**

In this section the three main tables are pulled into qlik, most of the editing here is just renaming fields but there are a few important things to note.

1. The Root table which contains all the measure types is connected to the Metadata table by a field I have named %key\_root, and the Metadata table is connected to all the rest of the tables including the MeasuresMaster table by a field I have named %key\_measures.
2. To be able to divide up the MeasuresMaster table I needed to map the %key\_root value to the MeasuresMaster table, and I have renamed that field to RootValue. This is what the Root\_map table it for, it is not seen in the final data model.



1. Likewise there is a Suspended\_Map table which maps the [suspendedOn] date value from the Metadata table to the MeasuresMaster table. This allowed me to write a condition at the bottom of the Metadata and MeasuresMaster tables that states only records which have not been suspend are pulled into qlik. If someone suspends a record in the website then when the app is refreshed that record will no longer appear in the app. Allows junk data to be cleaned by the end user.



1. The final notable thing in this section is that I added counter values in the Metadata table and MeasureMaster table. In the analysis these values are summed to create the # Rows and # Submissions master measures.

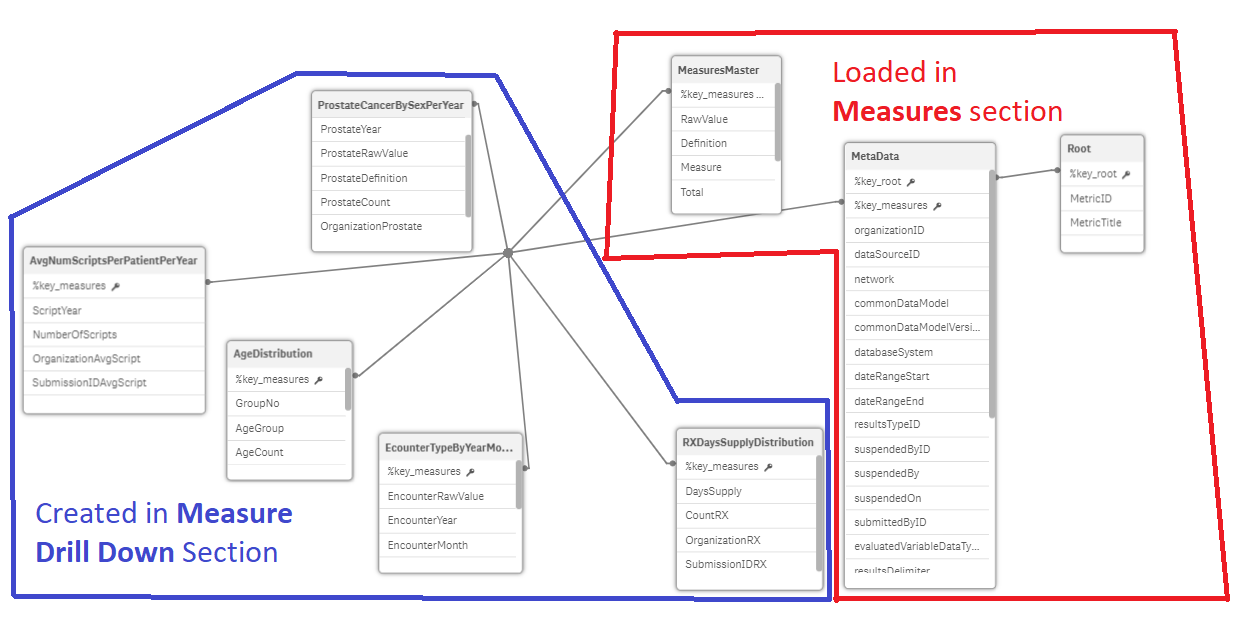
**Mapping Tables:**

This section contains five additional mapping tables that are used by the tables in the Measure Drill Down section. This application has a lot of filters that are based in the Metadata table. When creating visualizations we can easily pull the fields directly from the Metadata table to filter our data but it presents a slight problem. For example if we wanted to filter by Organization for RX Counts Per Year data and we pulled the Organization field for the filter directly from the Metadata table it will show every single Organization for any type of submissions as options. Even if those Organizations have no submissions for RX Counts Per Year. When the organization is mapped to RX Counts Per year then used as a filter then only Organizations that have submissions for RX Count will appear in the filter.

**Measure Drill Down:**

This section pulls data from the MeasuresMasters by filtering on the RootValue mentioned early. Five tables are created here for five focus areas: Age Distribution, Average Number of Scripts Per Patient Per Year, Prostate Cancer By Sex Per Year, Encounter Type By Year Month, and RX Days Supply Distribution. These tables utilize the maps from the previous sections for filter values. Whenever there was multiple data in a single column I split it using the subfield() function.

The resulting data model looks like this:

****

## Registering a sheet in the dqm site



Harvard Pilgrim Health Care

Registering a sheet in the DQM site

October 16, 2019

written by: Chris domain

cdomain@analytics8.com

**REVISION HISTORY**

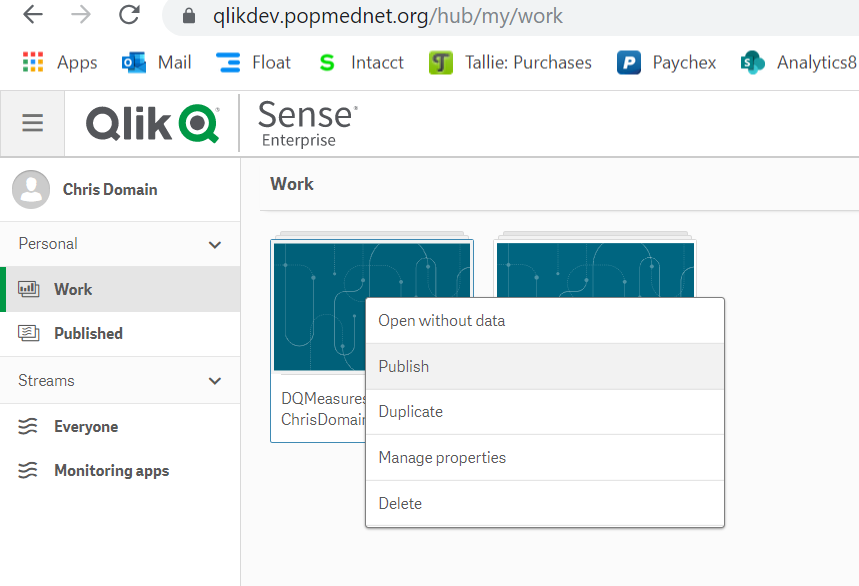
|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 10/16/2019 | 1.0 | Initial Document Creation | Chris Domain |

**DOCUMENT OVERVIEW**

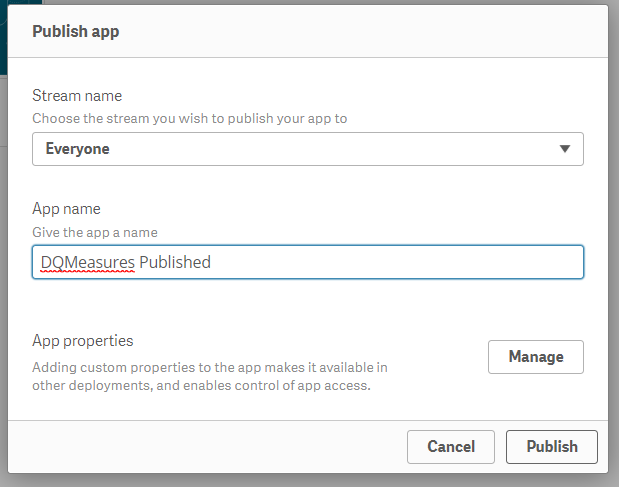
This document provides details on how to register a qlik sheet into the DQM measures website for viewing.

**STEPS**

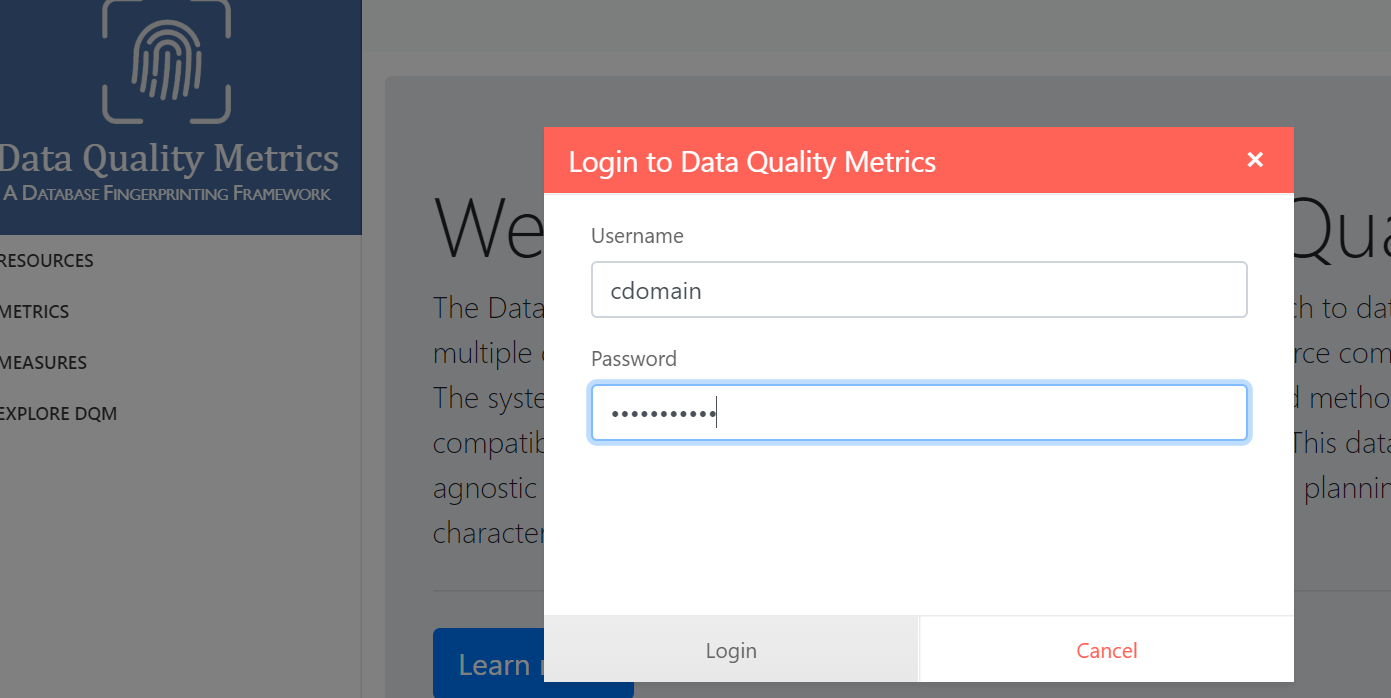
1. Open up the qlikdev hub and right click on the application you want to register, select publish.



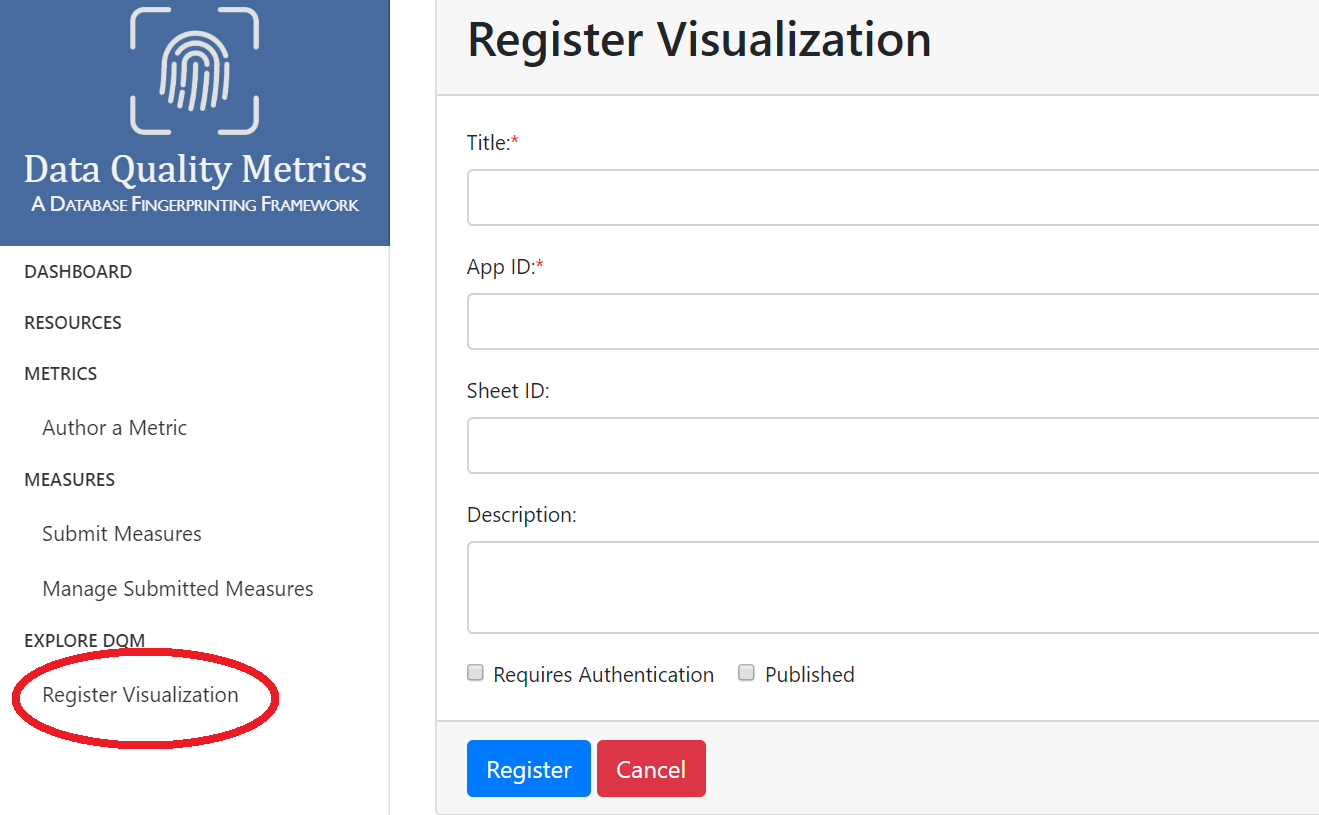
1. Select the ‘Everyone’ stream and give the application a name, hit Publish.



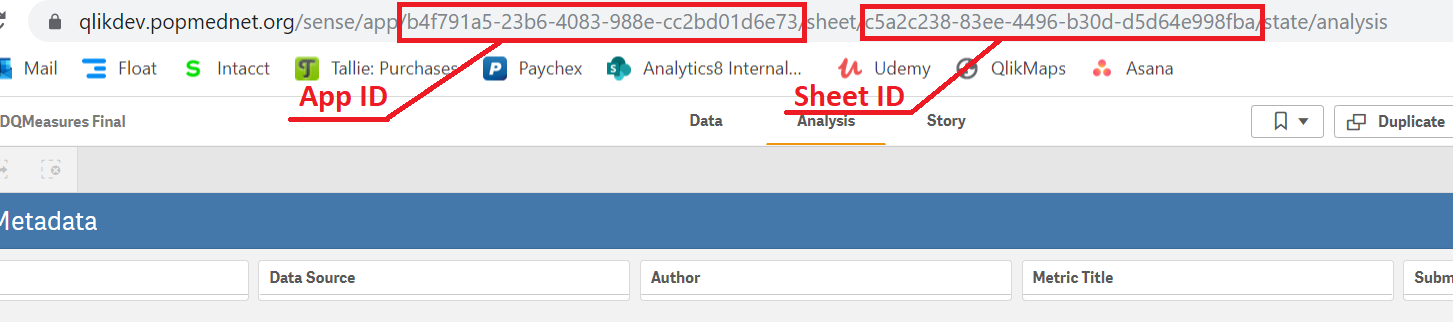
1. Open the application you just published in the Everyone stream, keep this page open then open a new tab.
2. Go to the DQM site: <https://dataquality.healthdatacollaboration.net/> , click Login and enter your credentials, click Login.



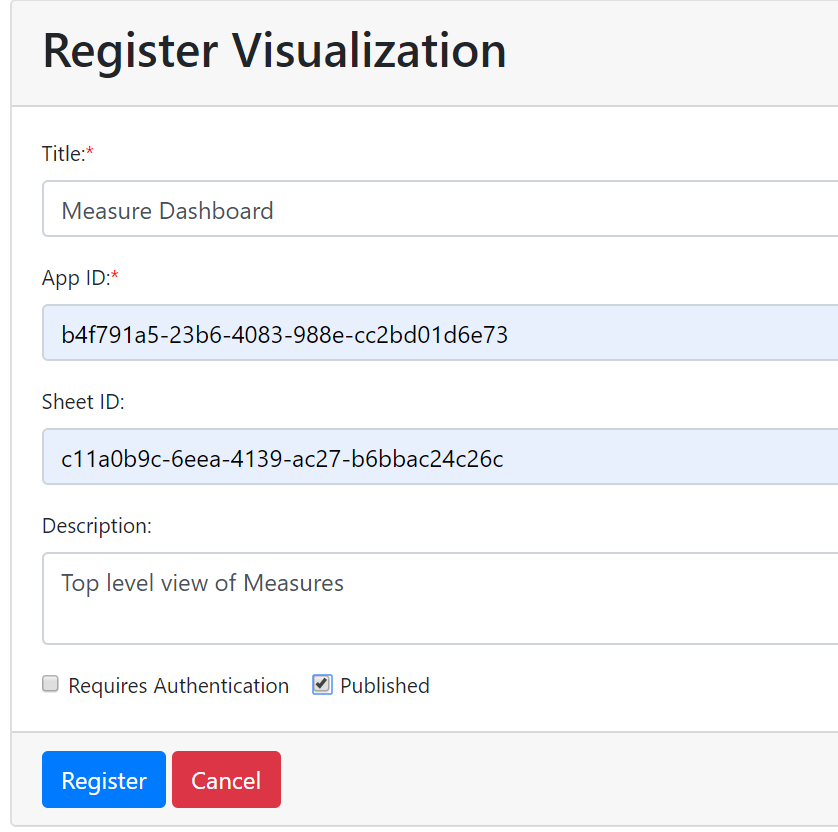
1. On the bottom of the left menu select ‘Register Visualization’, you will be brought to the screen below.



1. The title and description can be anything you want. To get App ID and Sheet ID navigate to the sheet you want to register, the ID’s will be located in the url.



1. Once you enter all the information, check the ‘Published’ box and click register.



1. To see the report simply select ‘Explore DQM’ from the menu and select the sheet you just registered!

References

1. Kahn MG, Callahan TJ, Barnard J, et al. A Harmonized Data Quality Assessment Terminology and Framework for the Secondary Use of Electronic Health Record Data. *EGEMS (Wash DC).* 2016;4(1):1244.

2. Curtis LH, Brown J, Platt R. Four Health Data Networks Illustrate The Potential For A Shared National Multipurpose Big-Data Network. *Health affairs (Project Hope).* 2014;33(7):1178-1186.

3. Fleurence RL, Curtis LH, Califf RM, Platt R, Selby JV, Brown JS. Launching PCORnet, a national patient-centered clinical research network. *Journal of the American Medical Informatics Association.* 2014;21(4):578-582.

4. Vogel J, Brown JS, Land T, Platt R, Klompas M. MDPHnet: Secure, Distributed Sharing of Electronic Health Record Data for Public Health Surveillance, Evaluation, and Planning. 2014;104(12):2265-2270.

5. Electronic medical record Support for Public Health 2018. 2018.

6. Natinal Intitutes of Health (NIH) Strategic Plan for Data Science. 2016-2020.

7. Callahan TJ, Bauck AE, Bertoch D, et al. A Comparison of Data Quality Assessment Checks in Six Data Sharing Networks. *EGEMS (Washington, DC).* 2017;5(1):8.

8. Huser V, Kahn MG, Brown JS, Gouripeddi R. Methods for examining data quality in healthcare integrated data repositories. *Pacific Symposium on Biocomputing Pacific Symposium on Biocomputing.* 2018;23:628-633.

9. Huser V, DeFalco FJ, Schuemie M, et al. Multisite Evaluation of a Data Quality Tool for Patient-Level Clinical Data Sets. *EGEMS (Washington, DC).* 2016;4(1):1239.

10. Liaw ST, Rahimi A, Ray P, et al. Towards an ontology for data quality in integrated chronic disease management: a realist review of the literature. *International journal of medical informatics.* 2013;82(1):10-24.

11. Arts DG, De Keizer NF, Scheffer GJ. Defining and improving data quality in medical registries: a literature review, case study, and generic framework. *Journal of the American Medical Informatics Association : JAMIA.* 2002;9(6):600-611.

12. Anderka M, Mai CT, Romitti PA, et al. Development and implementation of the first national data quality standards for population-based birth defects surveillance programs in the United States. *BMC public health.* 2015;15:925-925.