Research Data Management Platform

User Manual

# Background

## What is RDMP?

RDMP is a tool for the curation of research datasets. This includes many typical ETL tasks but also tools for management of the research lifecycle. The software focuses on ensuring thorough documentation of datasets, reliable loading of often poorly structured/volatile data, cohort linkage and reproducibility of project extracts.

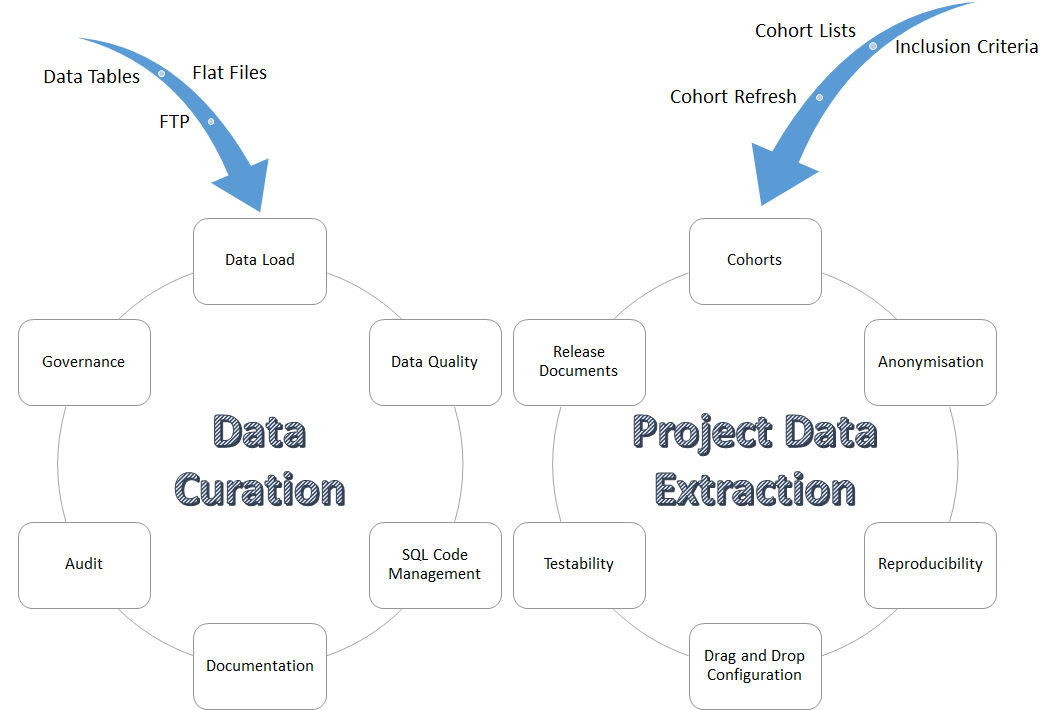


Figure 1 - RDMP Features Diagram

## Why is research data curation important?

Data management and data curation of long-term study and research databases are time consuming and complex activities that demand the attention of experts with very specific skills. Some of the most costly and complex data management activities emerge from consideration of two common scenarios. The first considers a single cohort used in a longitudinal study accruing data in distinct phases where the new data must be reconciled and merged with the existing data sets. The second scenario occurs when distinct cohorts from different studies of the same disease are merged to create greater scale in the research data. Again the data must be merged and reconciled in order to create an aggregate data set that is valid in its totality.

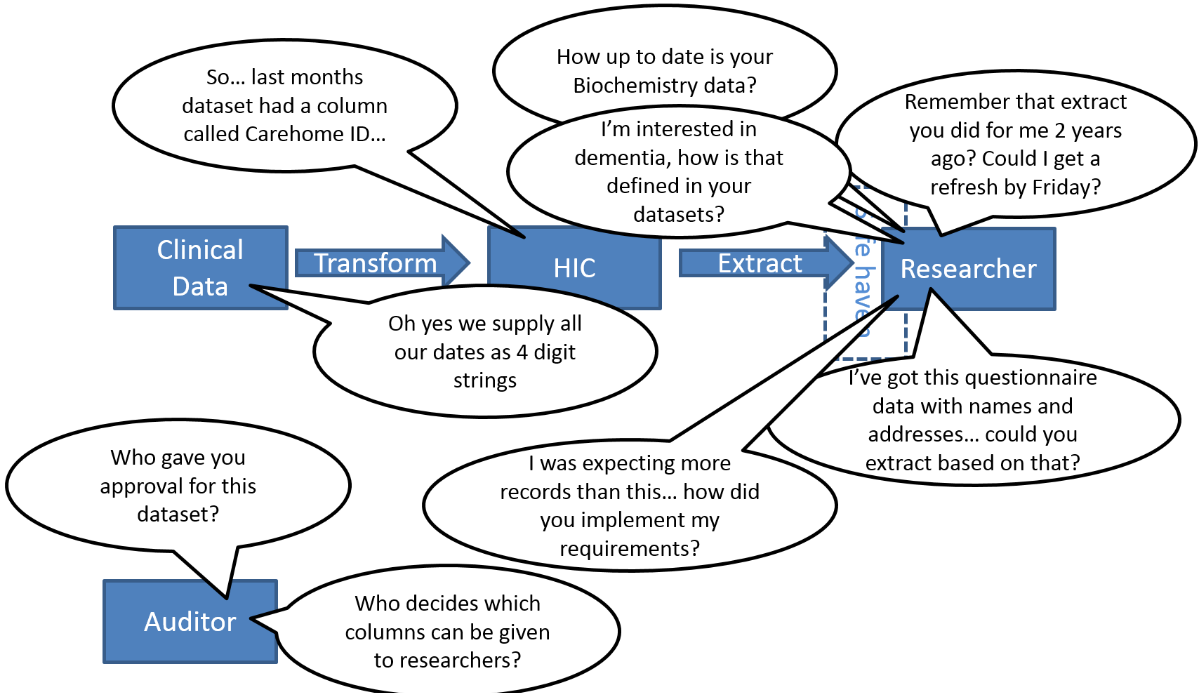


Figure 2-Common questions asked of a research data host

Existing data management approaches are focused on the initial generation and preparation of project research data and on preservation techniques that promote reuse of the data at the end of individual research projects. These approaches do not consider longer term studies and research programmes and fail to account for the key data merge, transformation and enrichment processes that are applied over life-time study lengths and that shape the data to support analysis and results. Failure to capture the project level transformation processes represents a major loss for long lived research data sets, as data improvements identified by individual studies and cohorts are not fed back into larger aggregated data sets to extend the data and improve the data quality.

Continuing dissatisfaction within the academic community with the lack of transparency in research data management and the inability to reproduce study results and understand the provenance of study data calls for further revision and extension of the research data management techniques. This software aims to resolve the major data management issues associated with long term study data management through a distinct life cycle for research data merge management. It focuses on transformation processes used within research projects brings transparency and reproducibility benefits through process mining. It also accommodates variation in the data and allowing multiple simultaneous versions and potentially conflicting views to exist through the application of competing transformation processes.

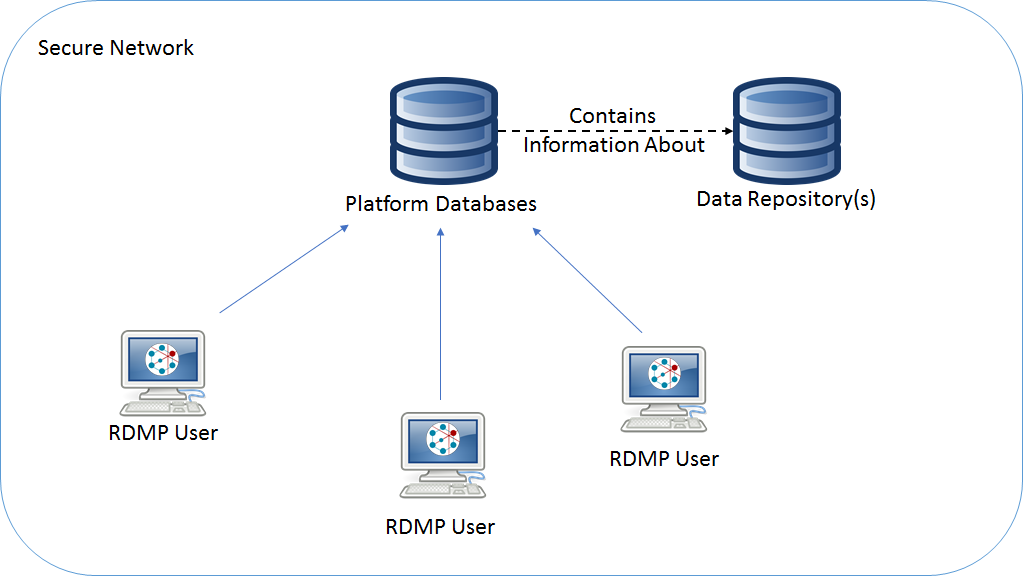
# Getting Started

## Installing RDMP Client Software

RDMP users access a central shared Metadata database (Platform Databases). You can install the client software via the installer link on the Github site

<https://github.com/HicServices/RDMP#research-data-management-platform>

Once installed the software will guide you through the process of setting up platform databases / connecting to existing platform databases.



## Prerequisites

You will need an Sql Server instance. If you do not already have one, you can use the Express edition for free which is available from microsoft.com (<https://www.microsoft.com/en-us/sql-server/sql-server-editions-express> ).

RDMP supports accessing research data repositories stored in MySql, Sql Server and Oracle databases but stores it’s own metadata (platform databases) in Sql Server only.

## Platform Database Setup

The RDMP uses SQL Server databases to store metadata (dataset/column descriptions, validation rules, validation results, data load configurations etc) as well as to store logging data, caching repository data etc. These ‘platform databases’ are separate from your ‘data repository’ which is the location that you store your live research data.

The first time you start RDMP you will be prompted to create the metadata databases that allow RDMP to function. The simplest approach is to create them all on a single server, to do this enter your Sql Server name and a prefix for the databases.



Figure 3 - Create Metadata Databases

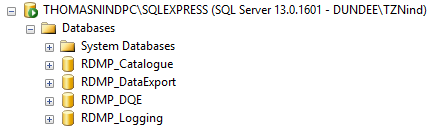


Figure 4 - Example Metadata Databases

This creates 4 databases which are the minimum set of platform databases required for RDMP to function. The databses fulfil the following roles:

Catalogue: Stores all descriptive, technical, validation, extraction information, attachments etc about your datasets (both externally accessible and internal). Also stores cohort identification criteria and data load configurations.

Data Export: Stores all project extraction configurations allowing for long term versioning and reproducibility into a variety destination formats (CSV, extract to database etc). Also stores which final cohort lists were used with which extraction projects (The actual lists are not stored in this database, see ­Functionality - Cohorts).

DQE: Stores a longitudinal record of the results of Data Quality Engine runs on your datasets. This includes how many rows passed validation rules, how many rows there are per month etc.

Logging: Stores in a relational format an audit of all flows of data from one place to another. This includes data being loaded, data being extracted and even internal processes like DQE executions.

There are a couple of other database types which can be created as and when you need them. These include

Query Caching: which helps improve the performance of complex cohort identification configurations and anonymisation

Cohort Databases: which store identifiable patient identifier mappings to release identifiers (on a per project basis, see ­Functionality - Cohorts).

Anonymisation Databases: which provide a way of performing identifier dropping / substitution on data load for when you want an entirely anonymous data repository.

Finally the RDMP also has support for plugin modules which can have their own platform databases.

## Setting up Datasets

### Generating Test Data

Once you have created the platform databases you will want to start telling RDMP about your datasets. A good place to start is with the RDMP example data. You can generate this by selecting Diagnostics=>Generate Test Data.

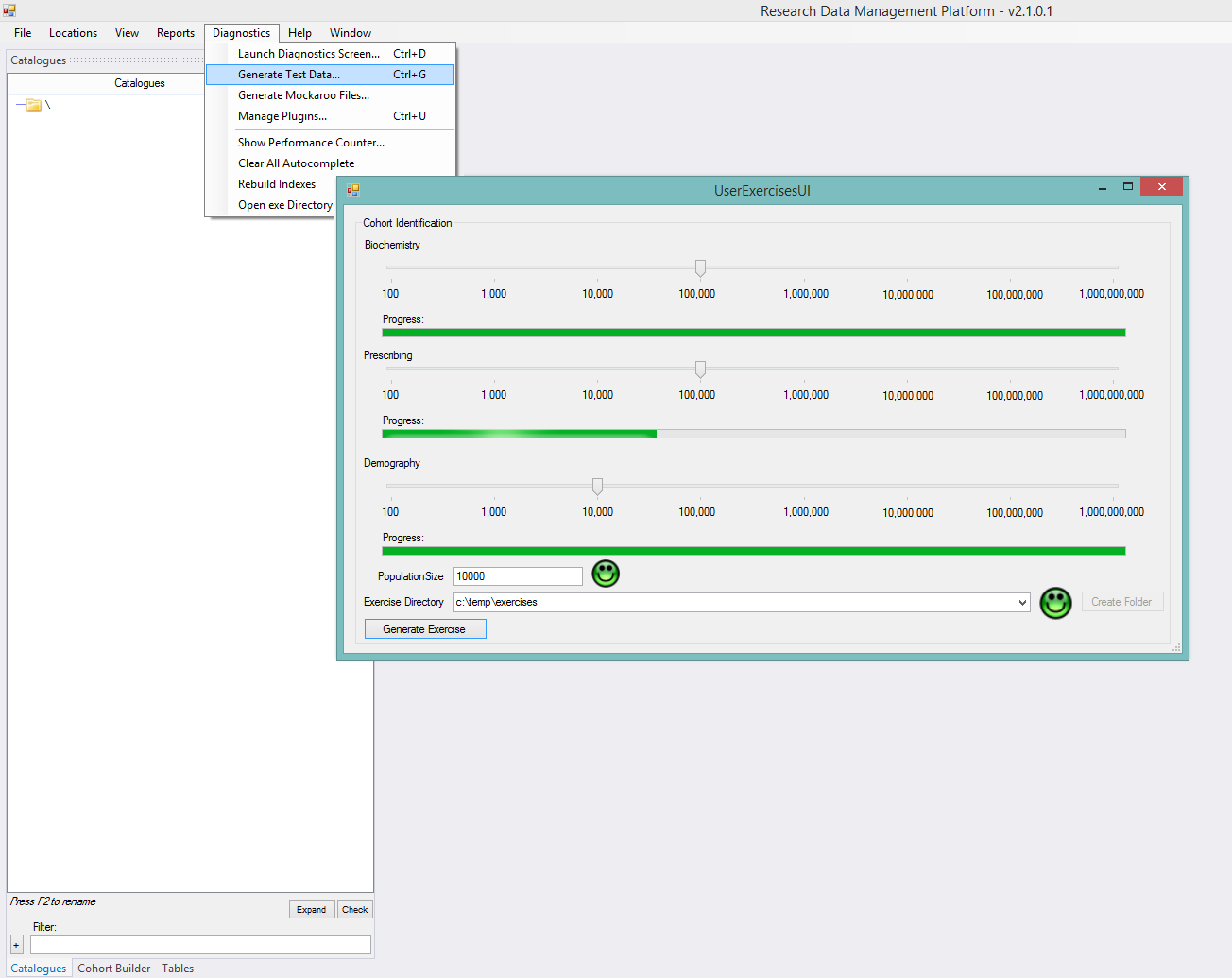


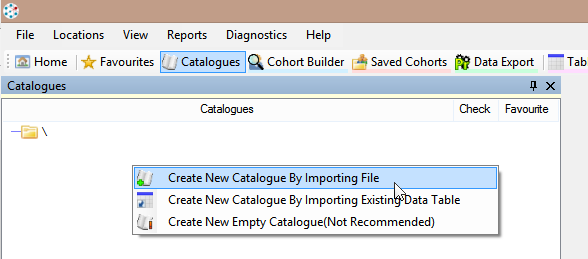
Figure 5 - Generating Test Data

Generating the test data will give you 3 CSV flat files (RDMP supports a wide range of data sources but CSV is nice and simple for now). Alternatively if you have some CSV files of your own you can skip this step and start using those. If you already have your data in a database then you can instead tell RDMP where to find this table (See Referencing an existing database table) and skip this stage entirely.

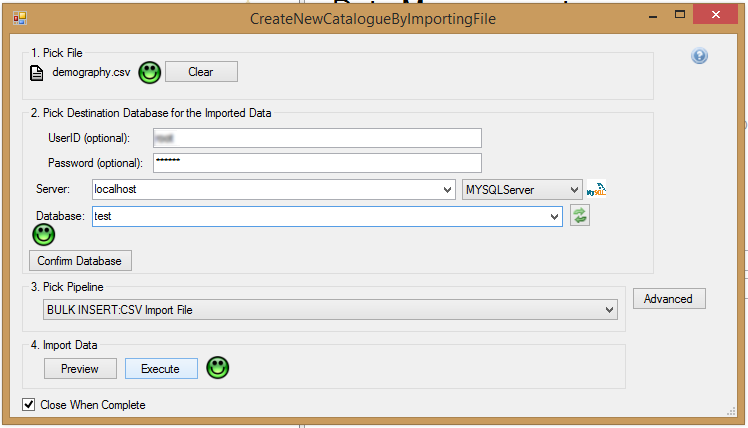
There are many problems that can occur in the daily handling of research data by data analysts. It can be helpful to discover how the RDMP handles various problems and what problems it cannot handle (and how it communicates this to you as a user). Problems with files you receive can consist of anything from primary key duplication to missing columns to extra commas in flat files. It can be helpful to learn how to use the software first with this test data created by the system before you unleash it on your own problem files.

### Importing a Flat File as new dataset

We want to import these files and turn them into datasets. You can do this by right clicking in the ‘Catalogues Toolbox’ and selecting ‘Create new Catalogue by importing flat files’.



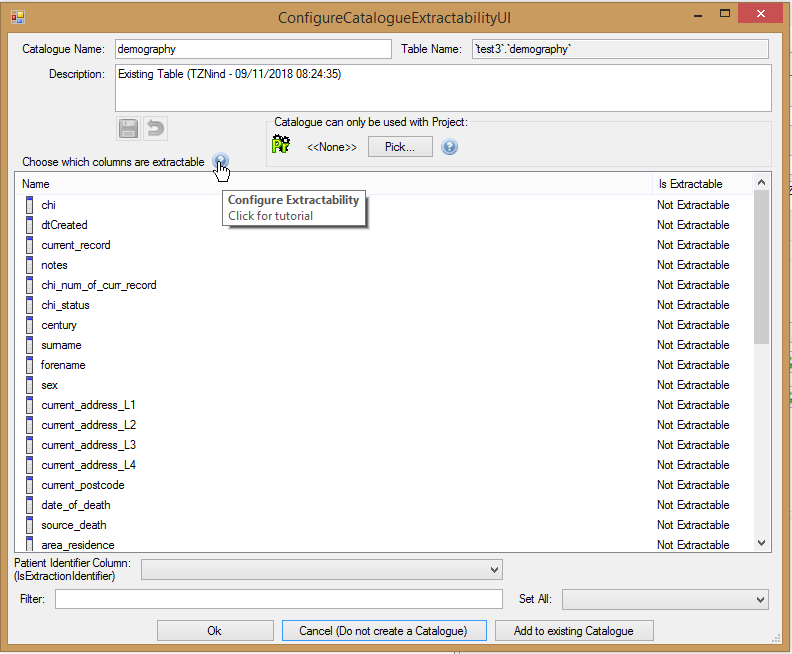
Choose a target server and database (do not use any of the platform databases!). When prompted to select a load pipeline use the default selection ‘BULK INSERT:CSV Import File’. You can click ‘Preview’ to see what the data looks like. Then hit Execute.



If the import fails you can click the Smiley face to see results of the file. If you have a particularly large file or are having problems with the import you can use the Advanced user interface to adjust the import pipeline and view import progress.

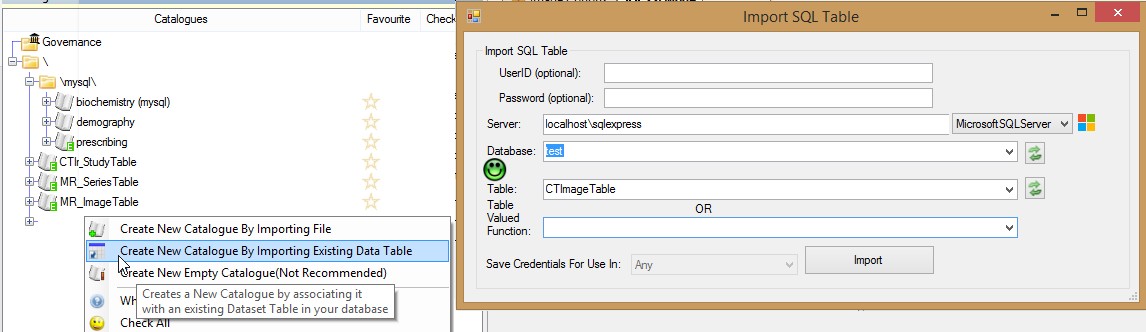
Once the import has been completed you will be asked to make a decision about which columns should be available to researchers in project extracts (extractable) and which should be kept private. You will also be asked to select which column is the patient identifier that can be used to link between datasets (e.g. chi).

For testing RDMP with the test data you should ‘Set All’ to Core extractability and set the patient identifier to ‘chi’ (Community Health Index).



### Referencing an existing database table

An alternative to importing data is to reference an existing table. This is for when you already have research data in a database and you want RDMP to know about it. This can be done by right clicking in the ‘Catalogues Toolbox’ and selecting ‘Create new Catalogue by referencing an existing database table’

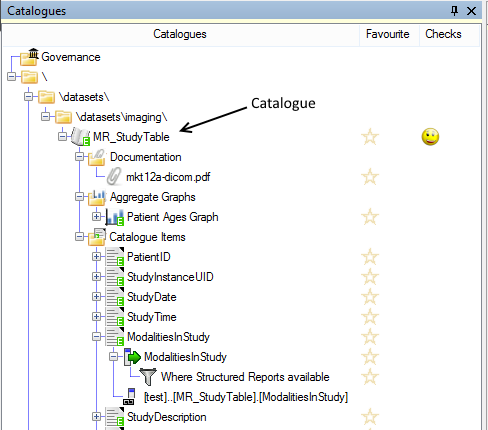


Choose the server, database and table name (also supports views and table valued functions. When prompted to pick extractable columns/patient identifier follow the same steps described above.

### What is a Catalogue?

A Catalogue is RDMP's representation of one of your datasets e.g. 'Hospital Admissions'. A Catalogue consists of:

* Human readable names/descriptions of what is in the dataset it is
* A collection of items mapped to underlying columns in your database. Each of these:
* an be extractable or not, or extractable only with SpecialApproval
* Can involve a transform on the underlying column (E.g. hash on extraction, UPPER etc)
* Have a human readable name/description of the column/transform
* Can have curated WHERE filters defined on them which can be reused for project extraction/cohort generation etc
* Validation rules for each of the extractable items in the dataset
* Graph definitions for viewing the contents of the dataset (and testing filters / cohorts built)
* Attachments which help understand the dataset (e.g. a pdf file)



A Catalogue can be a part of project extraction configurations, used in cohort identification configurations. They can be marked as Deprecated, Internal etc.

The separation of dataset and underlying table allows you to have multiple datasets both of which draw data from the same table. It also makes it easier to handle moving a table/database (e.g. to a new server or database) / renaming etc.

If you expand a Catalogue (e.g. biochemistry) you can see the ‘Catalogue Items’ node. These are the extractable columns in the dataset ‘biochemistry’. If you expand hb\_extract you can see two nodes, the first is the Extraction Information logic for the column and the second is the underlying database column reference (ColumnInfo). Right click the column and select ‘View Aggregate’.

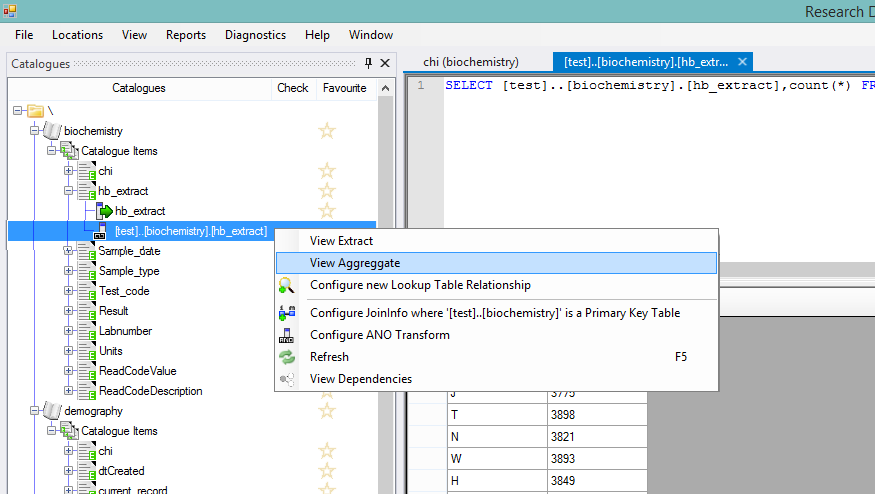


Figure 6 - Basic Catalogue objects for biochemistry

Assuming you have imported the test datasets succesfully you should now have 3 datasets with shared patient identifiers that you can use to explore the functionality of RDMP.

# Functionality – Data Quality

## Introduction

There are two summarisation components to the RDMP. The first is the DQE (Data Quality Engine), this allows you to create row level validation rules for the columns in your datasets (If column A is populated then column B should also have a value in it, column C must match Regex Z etc). The results of DQE executions are stored longitudinaly in the DQE database, this allows you to pipoint when your data became corrupt or inspect the differences in quality before and after a data load at any time.

The second summarisation component is ‘Aggregate Graphs’, these are real time charts which provide a view of the data in your data repository. These graphs can be reused during cohort identification and data extract building for testing filter configurations. For example you could build a graph showing ‘All drugs prescribed over time’ and reuse it in a cohort identification set ‘People prescribed painkillers’ to confirm that you have configured your query filters correctly.

These graphs can be marked Extractable which will to generate graphs of the data in your datasets (Catalogues).

## Data Quality Engine

### Background

Validation and data quality monitoring are vital processes when operating a healthy data repository. There are many existing tools which will analyse your data tables and report on core metrics such as referential integrity, column ranges, null/missingness rates etc. The RDMP includes a DQE that differentiates itself from these tools in a number of ways (See Figure 15 - Data Quality Engine flow of records and results):

1. Operates only on extractable datasets’ columns (which can include transforms) rather than on raw tables
2. Allows for greater specificity in validation rules (e.g. Regex, boundary, bounded by another column etc). These run in C# and not SQL so can be more advanced.
3. Allows for consequence assignment by rule (e.g. if the date of admission is after the date of discharge then record contains ‘Wrong’ data but if there is no patient identifier then the entire record is unuseable - ‘Invalid’)
4. Validation results are computed along multiple useful dimensions:
   1. Dataset time (e.g. when hospital admission was)
   2. Real time (i.e. when the data quality engine was run)
   3. Data load batch number (i.e. the unique number of the DLE run that contributed that specific record)
   4. One user chosen dataset specific pivot dimension (e.g. healthboard)
5. Support for writing domain specific plugins for unique validation. This could be as simple as a unique checksum calculation or as complex as contacting a web service and downloading a valid values whitelist.

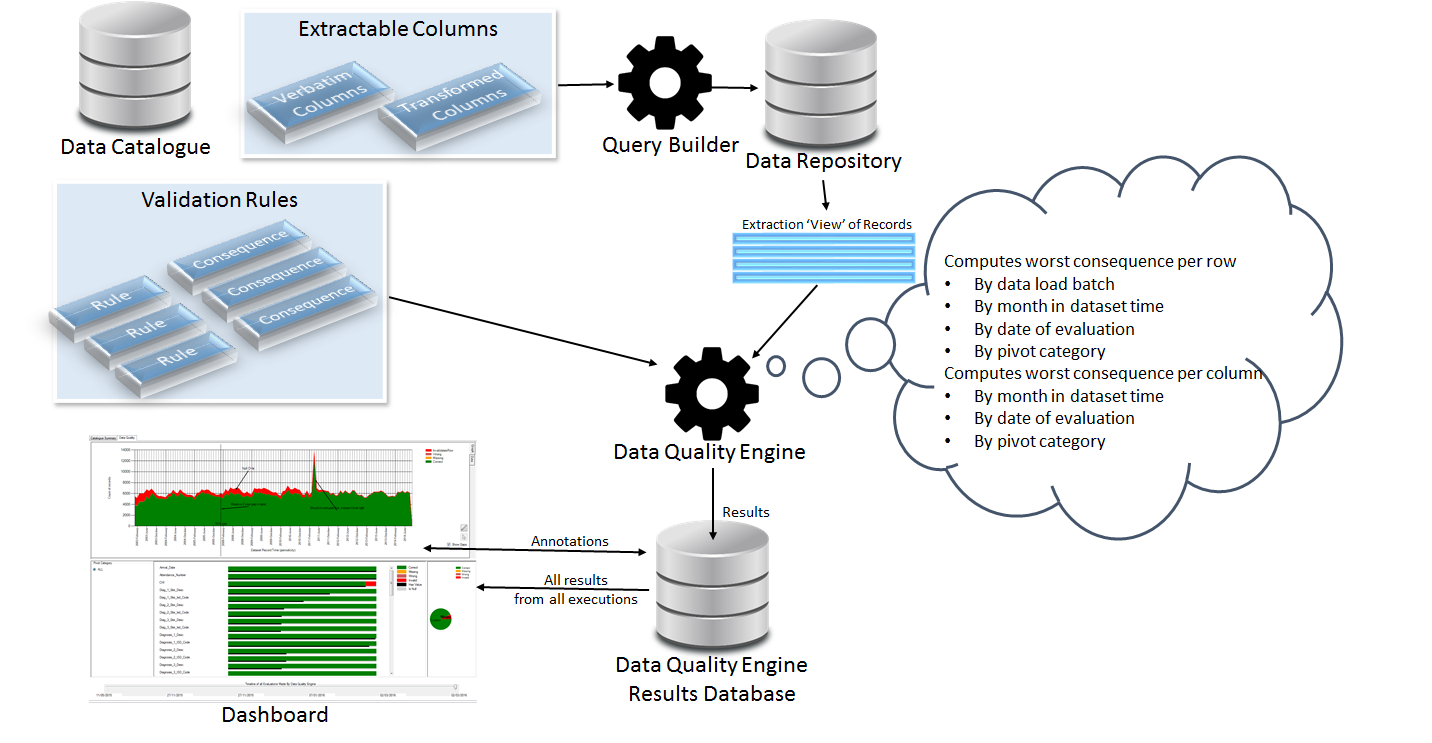


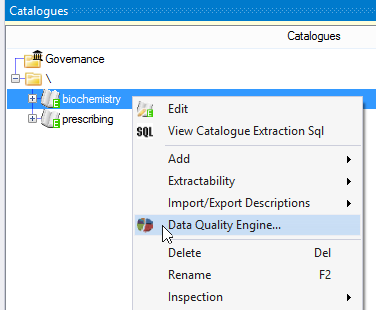
Figure 7 - Data Quality Engine flow of records and results

### Prerequisites

Make sure you have imported the 3 test datasets (See Setting up Datasets) or have a dataset of your own already imported that you want to setup rules on.

### Running the DQE

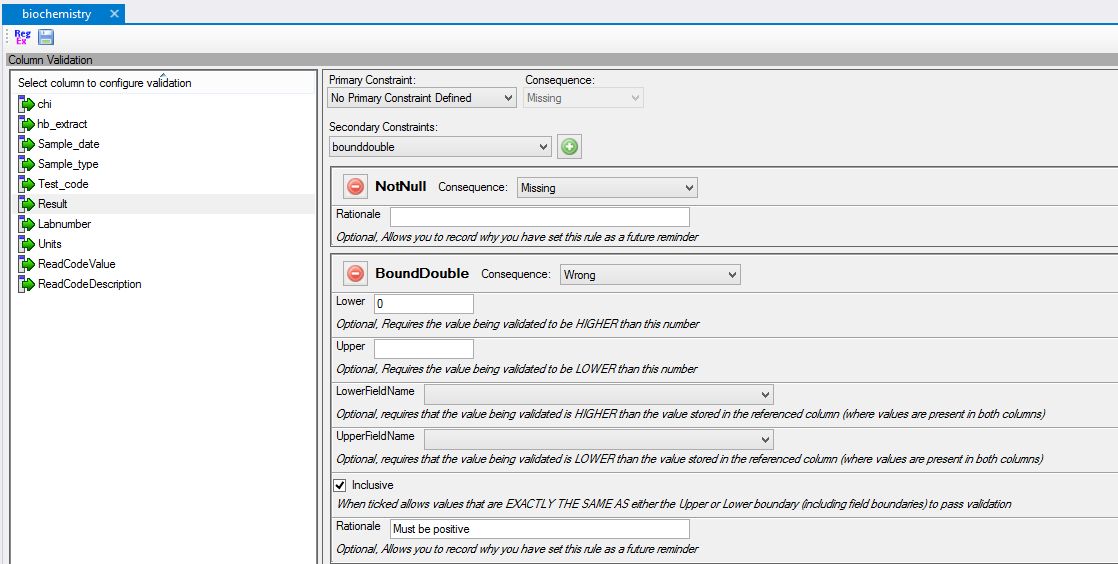
In order to run the DQE needs a validation rules. Right click the ‘biochemistry’ Catalogue and select Data Quality Engine=>Pick Time Coverage Column and select the ‘Sample\_date’ column.



Select ‘Validation Rules…’ and pick a Time Period Column. This is a date column which records in a meaningful way the origin date of a given record in a dataset. For example ‘date of birth’ in a birth certificates dataset or ‘date of address change’ in demography. If your dataset doesn’t have any date columns that are at all meaningful you can use the load date of the record.

Next set the ‘Pivot Column to ‘hb\_extract’ this will stratify the results of the DQE across the various healthboards contained in the dataset.

Select the ‘Result’ column and add a BoundDouble constraint and a NotNull constraint. Set the Lower boundary to 0 and ensure Inclusive is ticked. Also set the Consequence of the BoundDouble constraint to Wrong.



igure 8 - Configuring validation on the biochemistry test dataset (must have a value and the value must be 0 or higher)

Once you have made the changes Save the configuration by clicking on the disk (or closing the tab) and execute the DQE. Once the DQE executes succesfully you can view the results.

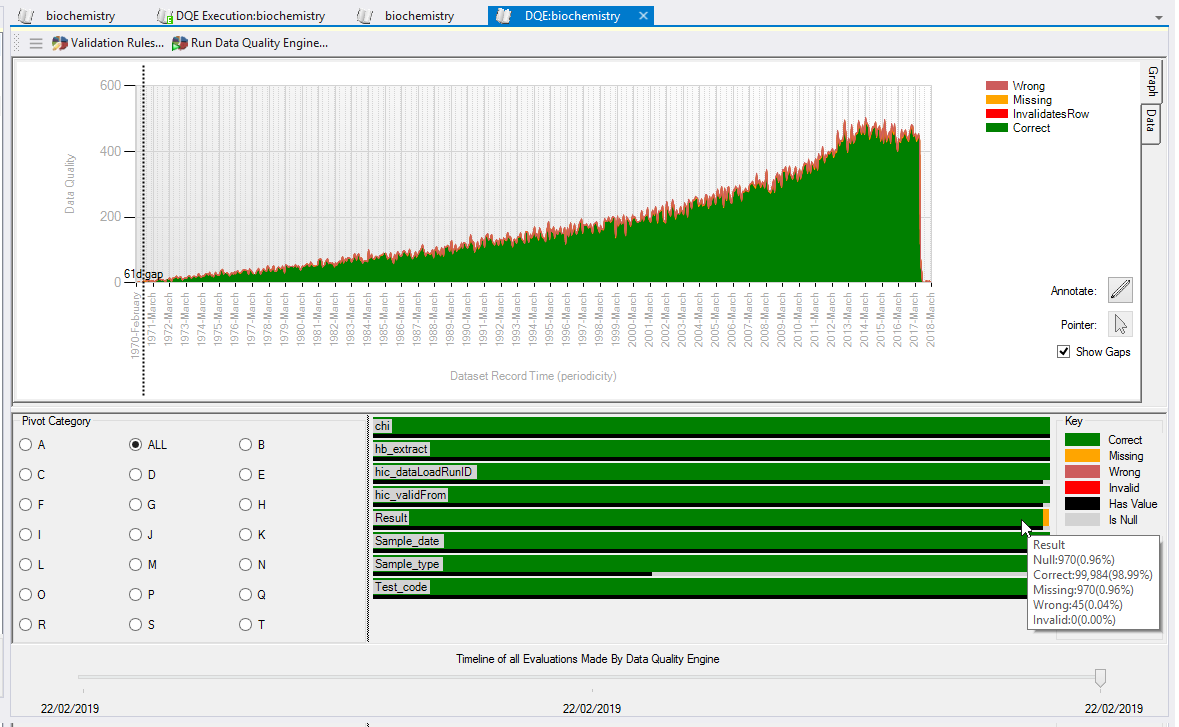


Figure 9 - DQE results for biochemistry

### Understanding DQE results

The DQE results graph shows a breakdown over time of the number of records failing validation (on any column) as well as a break down by column overall. The Pivot Category (if specified) is a further subdivision, selecting a Pivot value will show you a graph only for records with that pivot value.

Once you have run the DQE again the results tab will show a longitudinal history of all executions, by adjusting the ‘Timeline of Evaluations’ slider you can see what the quality of your data was like in the past (or before/after modifying the validation rules), this can be useful in identifying when corruption occurred and tracking down the causes of data loss etc.

### Validation Rules and Consequences

The only way rows can fail validation is if there is a rule on one of the columns that is broken. You only need to put rules on columns that you are worried about. For example with the biochemistry test dataset we can see that only ~40% of records have the Units field populated. If this is to be expected then putting a NotNull constraint would be a bad idea as it would mask genuine data problems.

While the validation rules support most common use cases through regular expressions, boundaries and referential integrity rules you may want to create your own unique rule. This can be done by writing a C# plugin and inheritting from PrimaryConstraint or SecondaryConstraint and implementing the validation logic (for example if you have a unique checksum calculation you want to perform on rows).

### Why not build rules directly into the SQL table schema?

Sometimes this is the right descision, for example in the biochemistry test dataset you might want to make it impossible to have records without a Result (in which case you could set the Sql Server Table schema to not null). This is particularly important if a field will be part of the table primary key. However sometimes you have to handle dirty research data with missing / wrong values (often spotty errors in an otherwise useful record). Rather than creating a schema that makes it impossible to store this data the DQE gives you a way of allowing low quality data into your repository but monitoring it.

## Aggregate Graphs

### Background

In addition to the DQE, the RDMP supports a more direct form of data summarisation through the Aggregate Graphs functionality. These are configurations that result in an SQL query that is executed in real time on your dataset to produce a graph. These configurations can be reused during cohort identification and project extraction as means of validating extraction activities.

### Prerequisites

Make sure you have imported the 3 test datasets (See Setting up Datasets) or have a dataset of your own already imported that you want to generate graphs for.

### Generating a simple graph

Right click the ‘Prescribing dataset’ and choose ‘Add New Aggregate Graph’. This will create an empty graph with no selected columns and no filters.

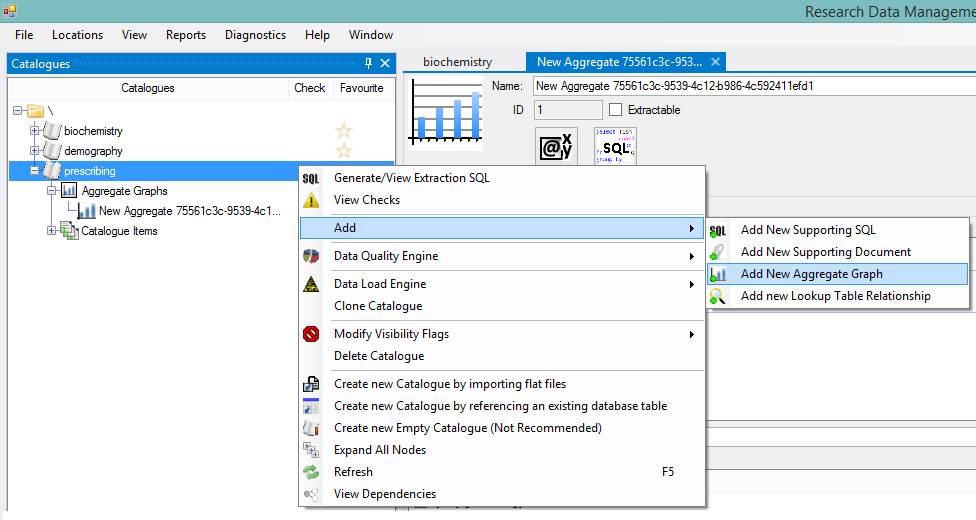


Figure 10 - Adding a new Aggregate Graph

Next tick the ‘approved\_name’ column and enter 10 in the TOP section of the query (order by count(\*) descending. This will give us a graph of the top 10 prescribed drugs in the dataset (See Figure 12 and Figure 13.

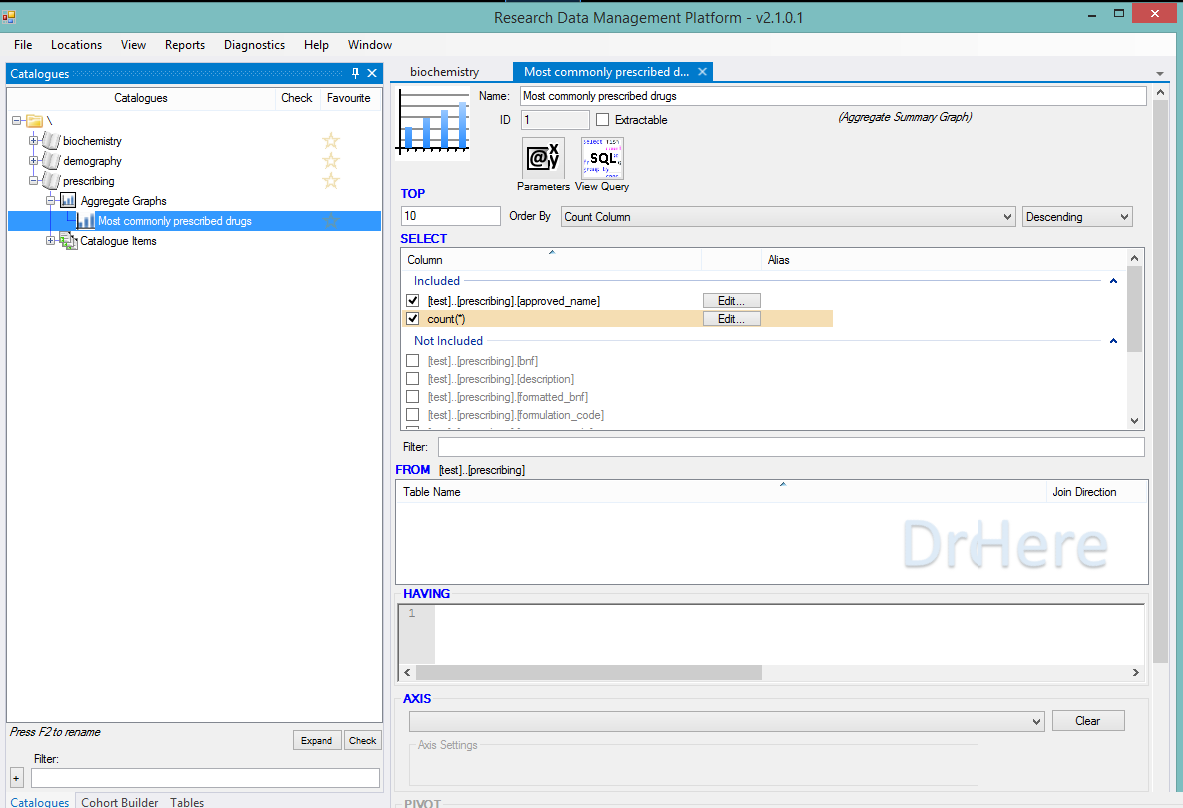


Figure 11 - Configuring a simple Aggregate Graph of Top 10 drug names

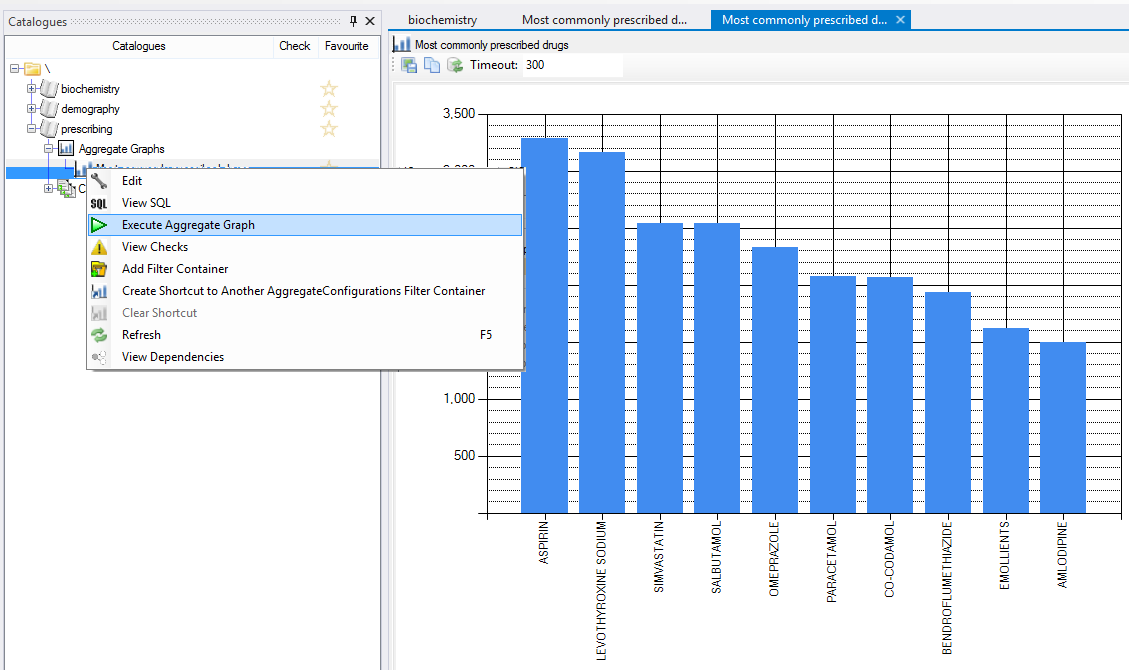
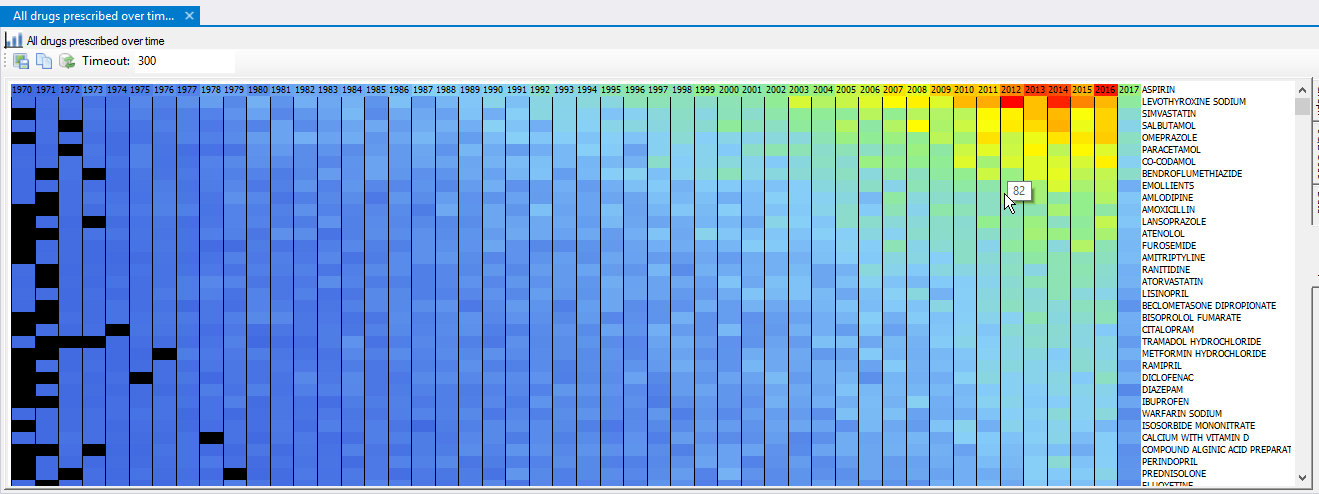


Figure 12 - Resultant graph of top 10 prescribed drugs

In isolation this graph is pretty useless but when combined with a cohort in ‘Cohort Identification’ it can be used to answer questions such as ‘Given my filter of BNF code 2.0.\*, what are the top 10 approved drug names’ or ‘Given my filter of Drugs prescribed in the last 6 months of a patients life, what are the top 10 approved drug names’.

### Generating a more complex graph

Create another graph for Prescribing called ‘All drugs prescribed over time’. Tick the approved\_name and prescribed\_date columns. Set the Axis to ‘prescribed\_date’ with a StartDate of ‘1970-01-01’ and an Increment of ‘Year’. Finally set the Pivot column to approved\_name and execute the graph.

The result is pretty unreadable but clearly shows the most common drugs. Switching to the Heatmap tab will show a more useful view.

This graph again isn’t super complex but is useful for identifying when prescriptions for a given drug start/end and gives a good idea of the sheer number of drugs in the dataset. Remember this is a test dataset in which all data is generated randomly so many of the interesting patterns you would see in a real dataset (drugs going in and out of favour or replacing one another etc) are not visible.

Since this graph is more useful you should tick the Extractable checkbox. This will allow you to execute the graph on project extracts in order to supply a baseline for what records the researcher will receive. This is useful not only because the cohort of patients the researcher is getting is only a subset of all patients (their drug prescriptions will be related to the study conditions) but there might also be extraction filters (for example the researcher only has ethics approval to receive prescription records for diabetic drugs).

## Dashboards

### Background

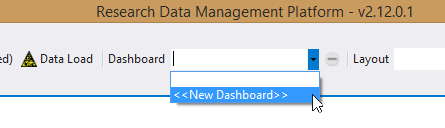
The RDMP Allows you to build modular dashboards from a range of available components that let you monitor the healthiness of your datasets, the extent of your metadata cover etc. This API is also tied into plugins allowing you to build custom controls that interact with RDMP and are presented on dashboards should you need to.

### Prerequisites

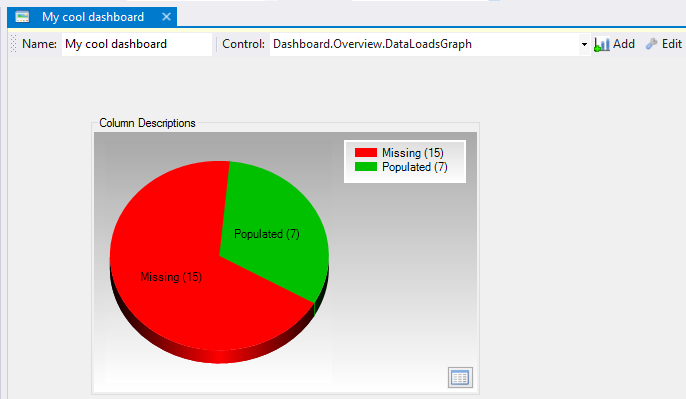
Make sure you have imported some datasets and run the DQE engine on them (See Running the DQE).

### Creating a simple Dashboard

You can create a new dashboard by selecting <<New Dashboard>> from the RDMP main toolbar.



Select a ‘GoodBadCataloguePieChart’ and click Add to add it to the new Dashboard.



This graph shows you the number of Extractable Columns amongst your datasets (Catalogues) which are missing column descriptions. Double Click a CatalogueItem that doesn’t have a Description and type in a description then Save the result. Now refresh the Dashboard, notice that the count has been recalculated and the graph has a little more green.

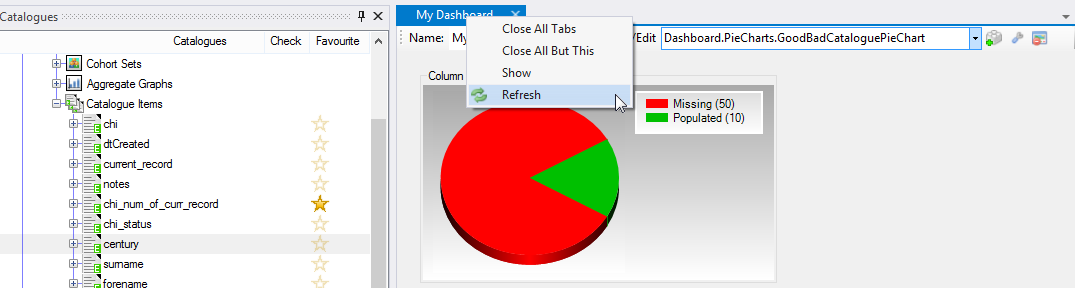
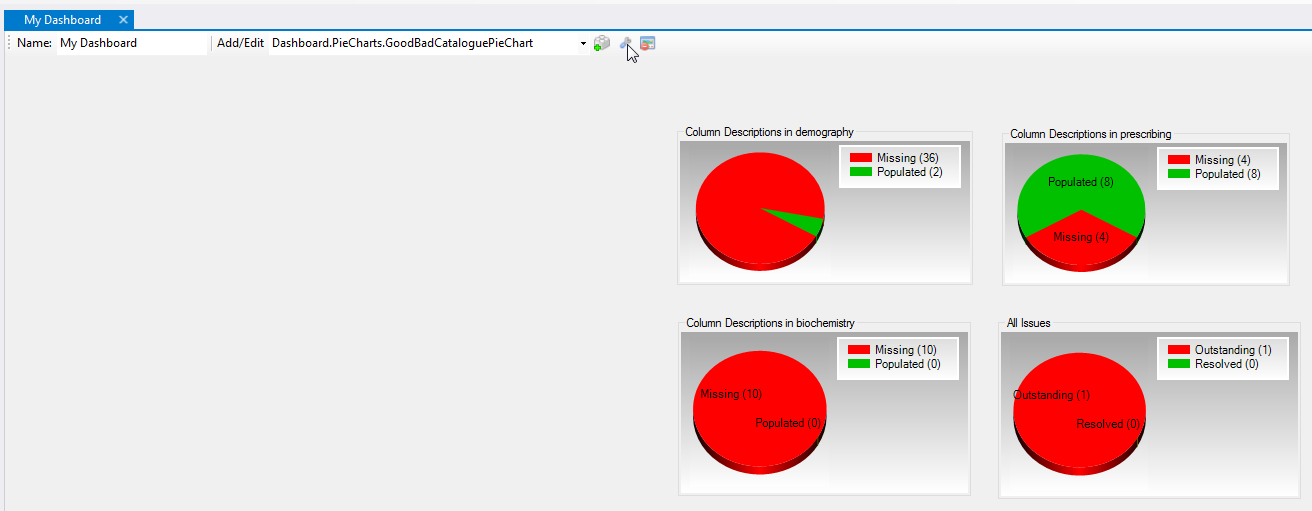


Figure 13- Populate a CatalogueItem description and then refresh the chart

Click the spanner to enter edit mode. This will let you move the pie chart and change what it shows (e.g. only show column descriptions for a single Catalogue, show Issues count.



Finally we can add a Dataset Raceway. This control provides an overview of the DQE graphs of all the datasets that it is configured to show. Add the Raceway and click edit mode and choose all the Catalogues.

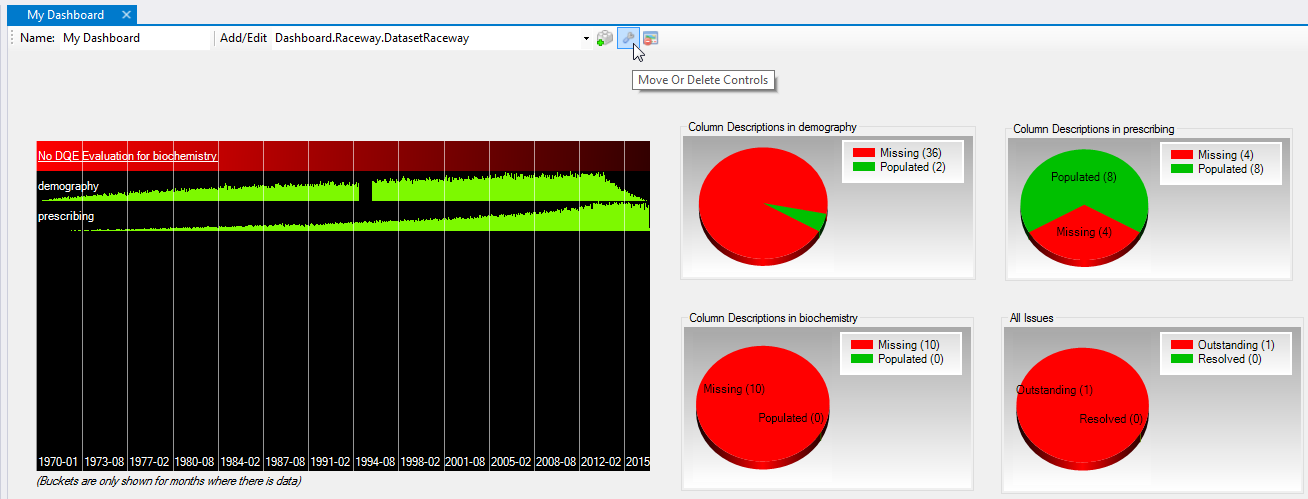
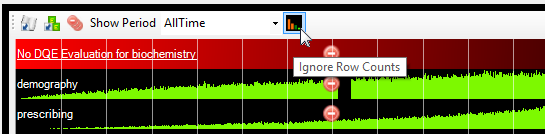
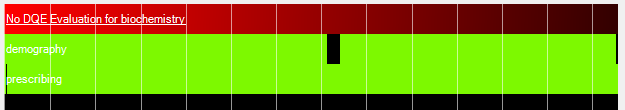


Figure 14 - Dashboard showing a dataset raceway in which there is missing data in demography and no DQE has been run on biochemistry yet.

The raceway graph should show that the data starts in 1970 and runs to the current date with increasing record counts each month. The height of the bar is the number of records in that month relative to the average number of records per month. Dates are not continous if your dataset is sparse e.g. if you have datasets that span 1950 – 2010 then you have lots of records in datasets for 1900-01-01 (dirty data) then that you will have a single bar for 1900-01-01 then the next bar will be 1950-01-01. If you want to identify which datasets are responsible for corruption and outliers then it can be useful to select ‘Ignore Row Counts’.



Enabling ‘Ignore Row Counts’ will render every bar at full height allowing you to see problems in very sparsely populated months. It also lets you tell the difference between ‘very few records’ and ‘no records at all’ for a given month/period.



If you want to try this out you can delete some data from the demography dataset (like I did) and rerunning the DQE. While the DQE results view shows all Evaluations over time the DatasetRaceway will only ever show the most recent.

Finally hovering the mouse over a bar will display a tool tip showing the date/time and absolute row counts passing/failing validation.

# Functionality – SQL Code Management

## Introduction

One difficulty facing any long running data management agency is how to preserve and document the code files/tools used for data loading, linkage and extraction. RDMP avoids the need for lengthy script files by dividing code into conceptual blocks. Once created, these blocks are stored as reusable documented components in the Catalogue database. When an analyst needs to use a given set of concepts he can assemble a query using drag and drop without having to worry about the underlying code implementation.

## Extraction Filters

### Background

Extraction Filters are lines of WHERE SQL which can be used as part of cohort creation, data summarisation and project extraction. Once created and documented a filter can be reused in any context simply by adding a reference to it. In this example we will create a ‘Diabetic drugs’ filter on the prescribing dataset.

### Prerequisites

Make sure you have imported the 3 test datasets (See Setting up Datasets) or have a dataset of your own already imported that you want to generate graphs for.

### Creating a filter

The National Institute for Health and Care Excellence has an online dictionary for BNF codes. It lists 6.1 as ‘Drugs used in diabetes’. In order to build a filter that identifies prescriptions for diabetic drugs we will need to first identify the BNF column in the prescribing dataset. Expand the CatalogueItems node of the prescribing dataset. Notice how there are two columns refering to bnf, right click the linked column and view an Extract of the column.

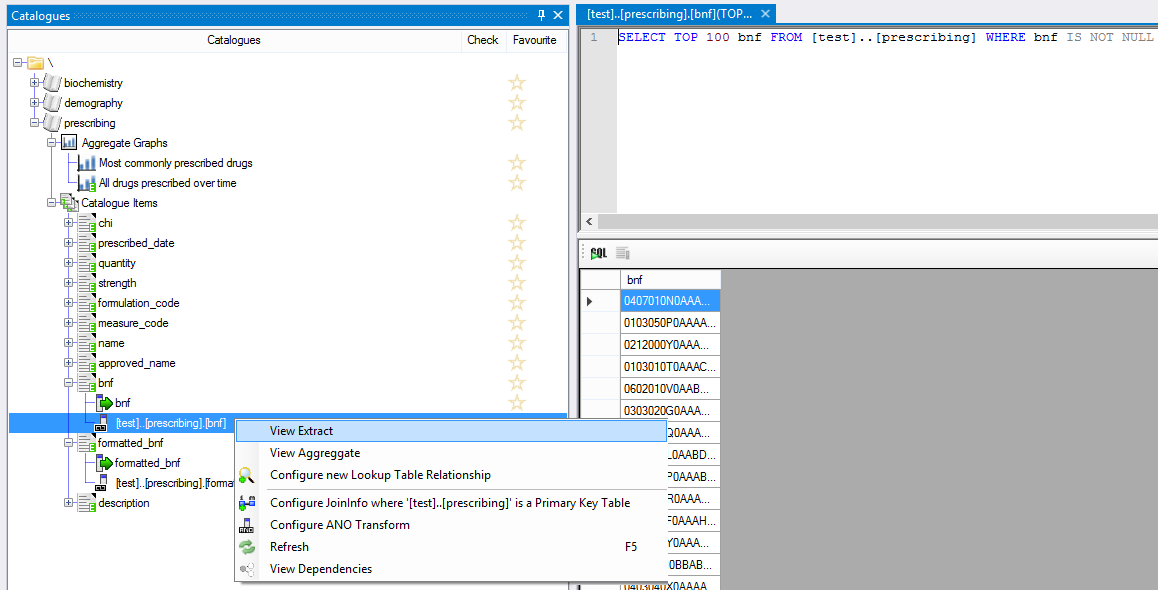


Figure 15 - View Extract of bnf column

Based on the records returned we can clearly see that formatted\_bnf is going to be easier to apply a filter to than the bnf column. Right click bnf and add a new Filter. Set the SQL to LEFT 3 = 6.1 (See Figure 14 - Adding a new filter)

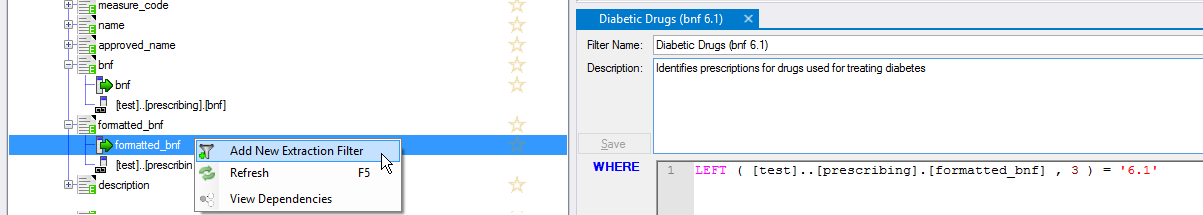


Figure 16 - Adding a new filter

### Checking that it works

We can check that the filter is working by right clicking it and either viewing an extract or combining it with the graph we created earlier (See Generating a more complex graph).

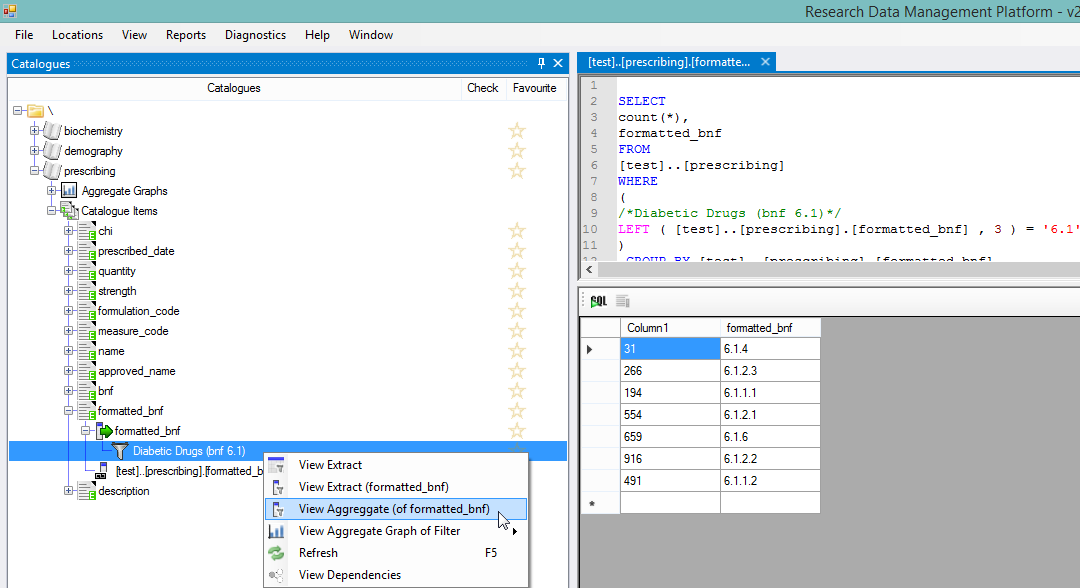


Figure 17 - Checking a filter works by viewing an extract of the column

When building a complex query with lots of inclusion / exclusion criteria, it is important to be confident that each section is working correctly. The ability to create and test small self contained filters for each concept goes a long way to improving the consistency and reliability of query creation. Another way of checking that a filter is implemented correctly is to combine it with an Aggregate Graph.

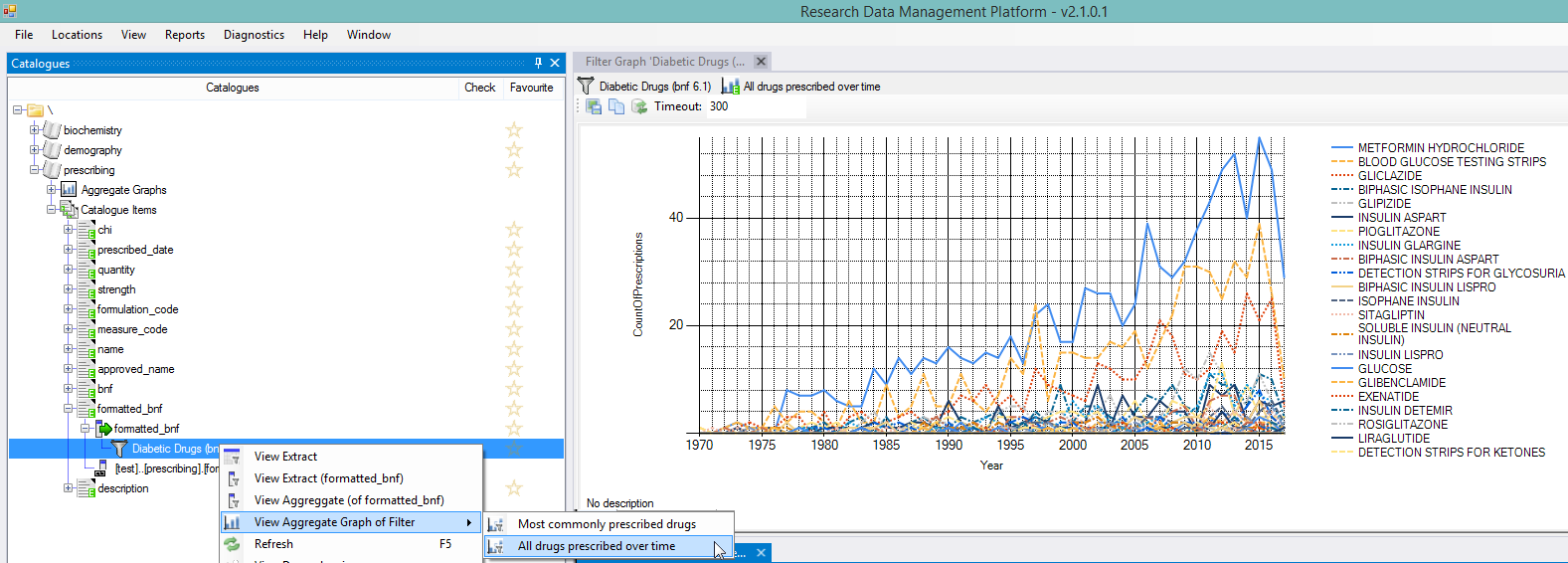


Figure 18 - Combining a Filter with an existing Aggregate Graph

### Creating some more useful filters

Filters created under a Catalogue are reusable throughout RDMP therefore it is a good idea to create standardised filters for important concepts at Catalogue level so they are available for everyone to use without having to nessesarily know the exact implementation.

Create a new filter under date\_of\_death in demograph called ‘Patient is dead’ with the SQL date\_of\_death is not null.

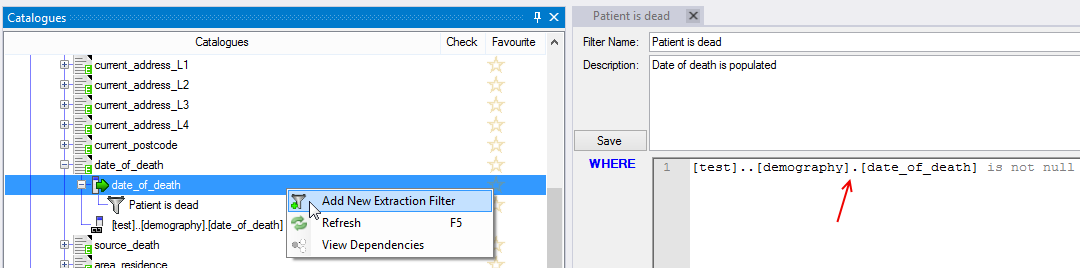


Figure 19 - Creating a master filter for identifying dead patients in demography

Check the filter works by right clicking it and select ‘View Extract’ the query should execute and only return records where the date\_of\_death is populated.

Next we will create a parameter driven filter. Parameters are how RDMP handles adaptive filters that can be adjusted easily by other data analysts without compromising the master and without having to know the exact implementation of the SQL code powering the filter. Parameters are already an inbuilt part of SQL so if you know SQL this should be very familiar.

Create a Filter on column prescribed\_date called ‘Prescription is BEFORE Date X’ set the SQL to prescribed\_date < @upperBoundaryDate. Save the filter, notice that a new parameter appears in the Params section of the filter.

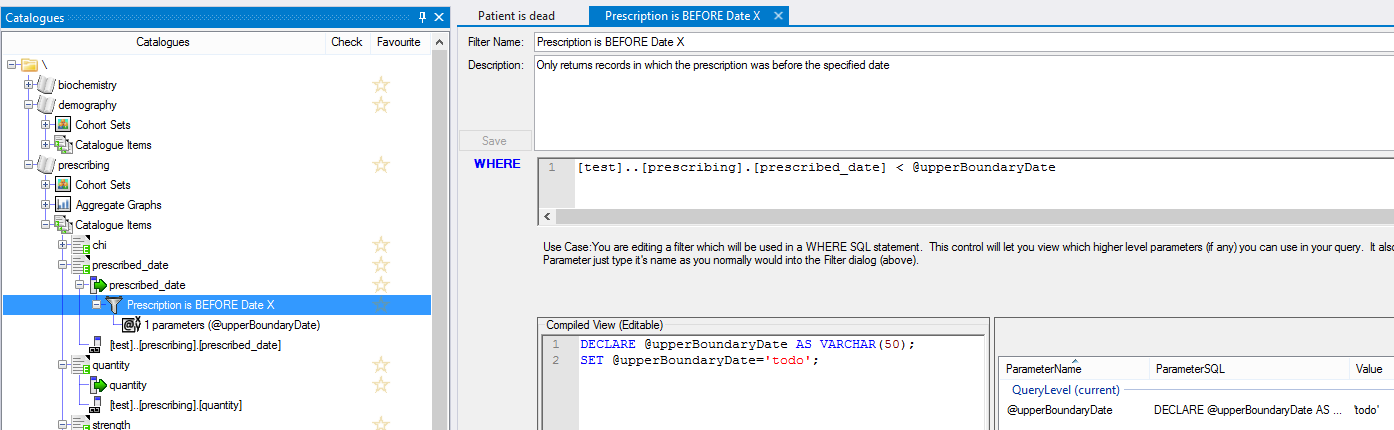


Figure 20 - Creating a filter with a parameter

Edit the parameter declaration to make it a date and give it a default value.

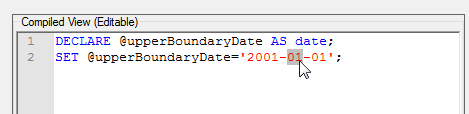


Figure 21 - setting the parameter to the correct data type and giving it a default value

Check the filter works by executing the ‘All drugs prescribed over time’ Aggregate Graph with the filter. Notice how the graph stops at 2000 indicating that the filter is working correctly.



Figure 22 - Filter 'Prescriptions before date X' combined with Aggregate Graph 'All drugs prescribed over time'

## Extraction Transforms

### Background

Sometimes you need to be able to provide a given column in more than one format or you may want to hold it in one format and release it in another. The most common use case is sensitive or identifiable data. The RDMP allows you to write SQL code to change how a given column is extracted (or create multiple versions) and assign it an extraction category (Core, Supplemental, Special Approval etc).

Data analysts often achieve the functionality described in this chapter through views and/or table valued functions. RDMP supports this approach too, if you decide that is a better solution for your agency (for example if you already have a lot of legacy extraction views).

### Prerequisites

Make sure you have imported the 3 test datasets (See Setting up Datasets) or have a dataset of your own already imported that you want to configure column extractability on.

### RDMP Extraction Data Model

In the previous chapter (See Extraction Filters) we saw how RDMP stores WHERE logic as Filters, next we will see how SELECT logic is curated and how you configure which columns are extractable and under what circumstances. The following objects take part in this process.

|  |  |
| --- | --- |
| Catalogue | A dataset in which some or all columns are extractable. Also contains the description of the dataset, supporting documents etc. |
| CatalogueItem | A description of a column or an extraction transform which may or may not be extractable (the  indicates extractable) |
| ExtractionInformation | SELECT SQL associated with a CatalogueItem/Column. Having one of these makes a column extractable. |
| ColumnInfo | A record of a physical column that exists in your data repository database which might or might not be extractable. |

### Removing a columns extractability

By default when importing/creating a Catalogue, all columns in the table it is based on will have an ExtractionInformation created with the SQL being simply the column name. This has the effect of making every column extractable. The first thing you should do when adding a new Catalogue is to disable extraction on any columns you don’t want to ever give out to researchers in a data extract (There is the option not to create any ExtractionInformations when creating a Catalogue if you prefer).

The ‘demography’ test dataset is very sensitive, containing lots of personal patient details (these are randomly invented of course). The first thing we should do is make the name columns unextractable. Open the ‘demography’ dataset in Catalogue toolbox and expand the CatalogueItems node and expand and delete the ExtractionInformation node from surname and forename.

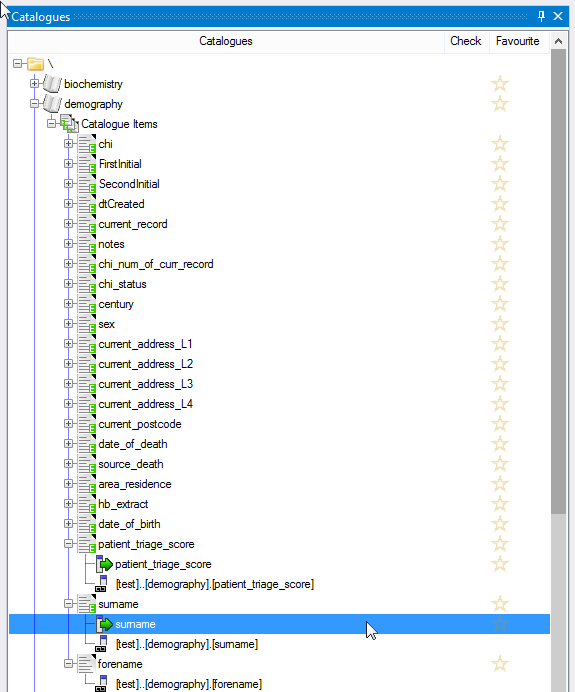


Figure 23 - Select the ExtractionInformation and press Delete to make the column surname nonextractable

This will leave the surname CatalogueItem and it’s relationship to the underlying column [test]..[demography].[surname] intact. This is done deliberately becuase there could be a lot of useful descriptive data / validation rules etc stored in the CatalogueItem which would otherwise be lost. This way if you decide later on that the column should be extractable again that information is not lost.

### Adding an extraction transform

Now that we have removed extractability of forename/surname we might decide that releasing the patients initials would be ok. Right click the ‘Catalogue Items’ node under the demography Catalogue. Select ‘New CatalogueItem’ and select the forename column. When prompted for a name give the name ‘FirstInitial’.

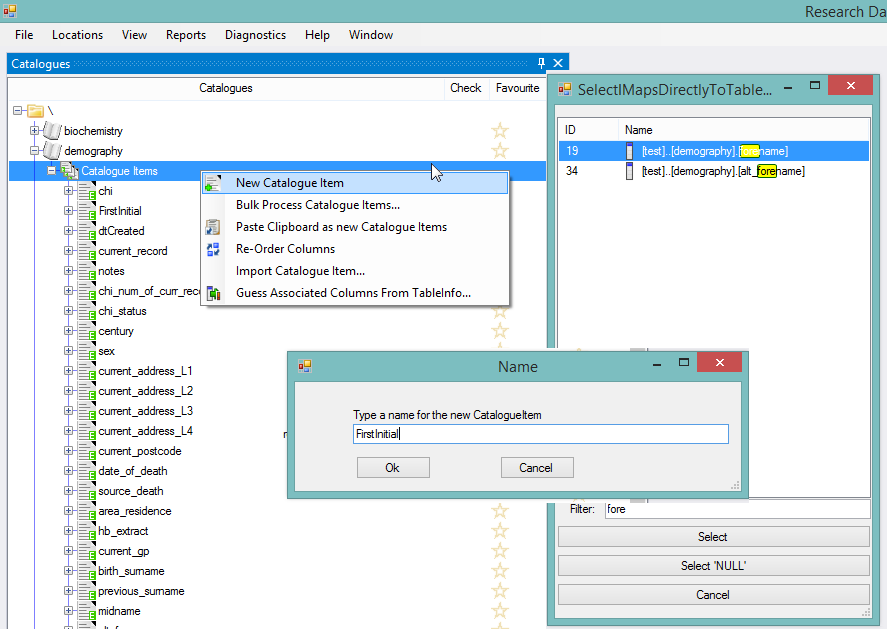


Figure 24 - Adding a new CatalogueItem

This has given us a new CatalogueItem called FirstInitial linked to the underlying column ‘forename’. Now we need to make it extractable and configure it to only supply the first letter. Right click the CatalogueItem ‘FirstInitial’ and select ‘Add Extraction Logic’. Replace the SELECT sql shown with a LEFT 1 (See Figure 19 - Transform to supply only the first initial of patient's forename). Now do the same for surname (SecondInitial).

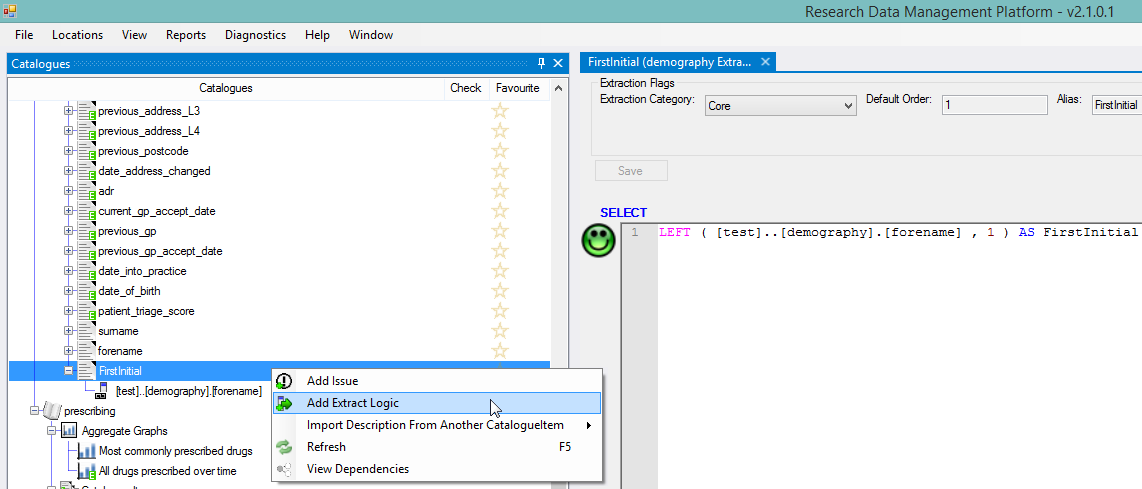


Figure 25 - Transform to supply only the first initial of patient's forename

### View Dependencies

To see how RDMP relates each object in the demograph catalogue you can right click it and select ‘View Dependencies’. With a little dragging objects around you should see the following (Figure 20). We can see that the column forename is used by both ‘forename’ CatalogueItem which is not extractable and ‘FirstInitial’ CatalogueItem which is extractable with a transform. We can also see that the forename column does not feed any other Catalogues (sometimes a single database column will be part of multiple extractable datasets).

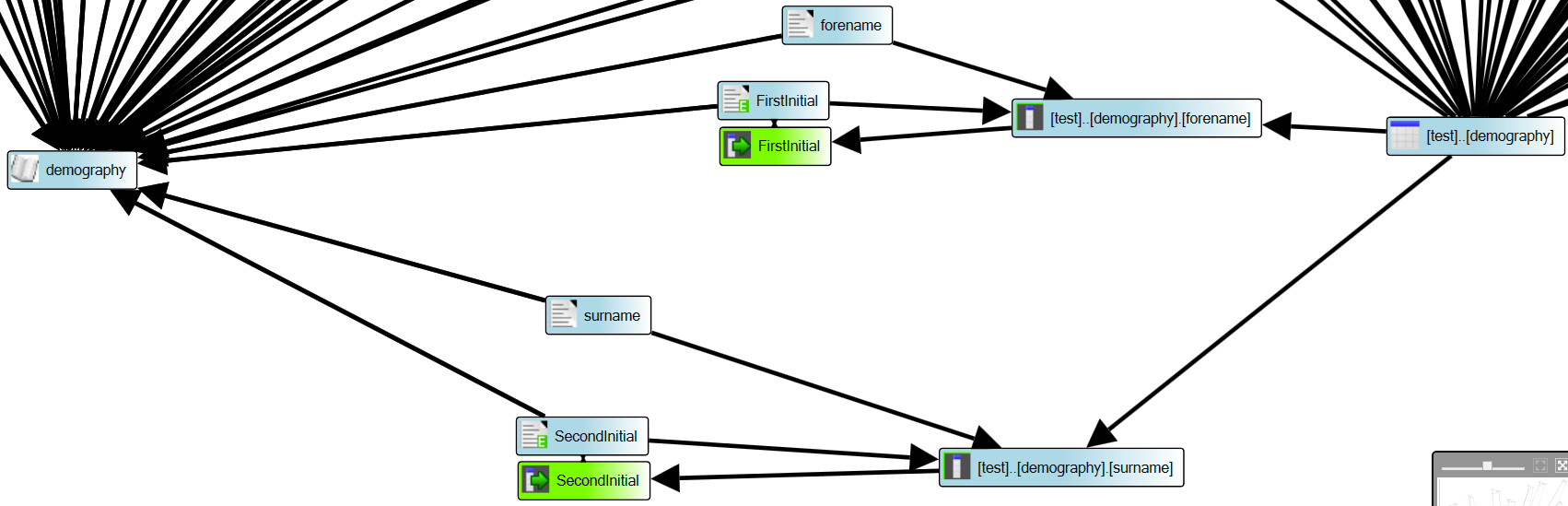
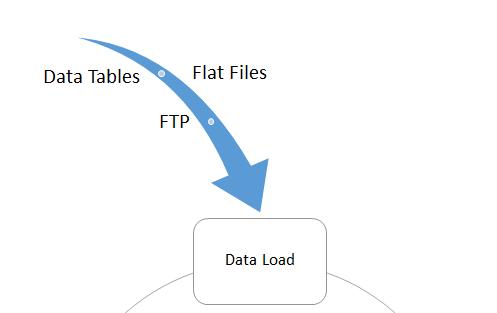


Figure 26 RDMP Data Model, Demography dataset with forename/surname extractable only as the first initial letter

# Functionality - Data Load



## Introduction

Research/Clinical datasets are often poorly formed, change structure over time and can require significant transform/cleaning before they are useful for statistical analysis by end users. Often the technical implementation of each datasets ETL is known only to the data analysts who built the original data load (and who might not even work at the agency any more) or the ETL may be built into bespoke legacy applications. The RDMP data load engine was developed by the Health Informatics Centre at a time when it was struggling to cope with reliable data loading of a high volume of clinical datasets using a mixture of commercial packages such as SSIS, bespoke dataset specific applications and undocumented SQL stored procedures. The design core requirements for the Data Load Engine were:

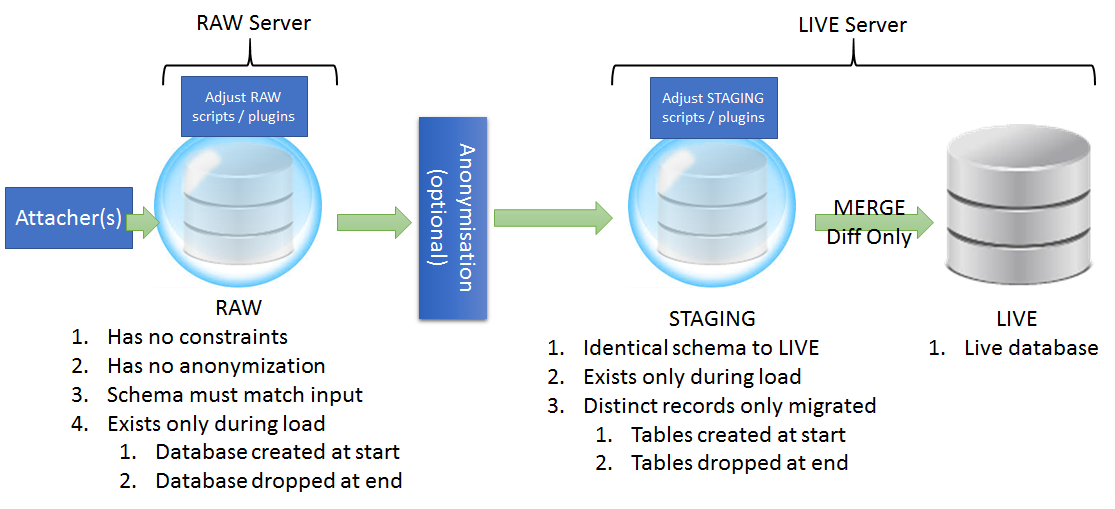
* Eliminate the possibility of duplication by loading the same source rows multiple times
  + DLE requires primary keys that come from the source data provider
* Prevent failed loads from impacting the live dataset (e.g. crashing half way through a load)
  + All data is loaded first into an unstructured unconstrained RAW environment
  + Then loaded into mirror area called STAGING which matches the LIVE environment
  + Finally, the load is committed as a MERGE from STAGING to LIVE
  + Any failure at any step results in zero affected changes to the LIVE dataset
* Centralise load logic into one place that is accessible to every data analyst
* Handle strange file types and bespoke/proprietary file layouts
  + Plugin architecture allows agency specific requirements
* Useable by data analysts who familiar only with SQL
* Ensure reusability of components where possible (e.g. where previously each load application would use a different FTP library and store credentials in different places or be hard coded or be a manual task!)
* Be automatable through a single point of automation

It is important to note that these features are designed for the robust loading of routine data (supplied on a regular basis). The RDMP has more lightweight use cases for one off loads (e.g. when you have a lookup table you just want to get into your database once and don’t plan to update later).

RDMP is primarily a data management tool and therefore does not seek to replace existing ETL tools that you might use for complex data transforms. It’s data load component is optimised for rapidly and safely loading and versioning data tables in as close a schema to the original source as possible (e.g. if a dataset is produced by a clinical scanner and comes out in with 30 headers then the data should ideally be held in your repository in the same format). This not only simplifies data load but means there is no bias / interpretation introduced at the data management layer, researchers receive the data exactly the same way you receive it. The only time extensive transformation is required during ETL is when the data includes duplication / errors that make the data otherwise unloadable. If you plan to build a star schema or fact table representation of data then there are probably better ETL tools for the job.

### RAW Bubble, STAGING Bubble, LIVE Model

Core to the data load process implemented in RMDP is the migration of the data you are trying to load through increasingly structured states (RAW=>STAGING=>LIVE).



The purpose of the RAW/STAGING/LIVE Model is:

* Isolating failed loads from affecting live data
* Divide data load problems into either loading (RAW) or database constraint /anonymization issues (STAGING)
* Allow ALTER / UPDATE logic e.g. to merge columns etc to be done in a safe environment (RAW) with a language the user is familiar with (SQL).
* Freezing failed data in the state in which it failed so anlysts can evaluate it

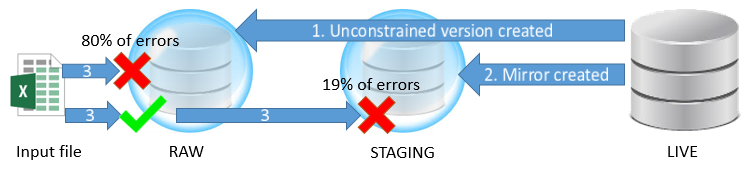


Figure 27 - Data load process

There are many reasons why a dataset load might fail. Some of the most common scenarios encountered by HIC include:

1. Data supplier changes the name of a column(s)
   1. Is there a semantic change to the data that should be documented?
   2. Is the contents of the column still rational with previously loaded data?
2. A column starts containing null values where previously it was fully populated
   1. Is the column still conceptually the same as before?
   2. Is it that the provider has started coding 0 values as null?
   3. Is it valid that the column contains nulls and the LIVE schema should be updated?
3. Data supplier has primary key duplication within a load file
   1. Is it the analyst/dataset which has the wrong primary keys?
   2. Is it possible to resolve by merging missing data into a composite record?
   3. Is it possible to identify one record which is clearly wrong / out of date?

Of the above cases, the first would break on loading to RAW and require adjustments to the load configuration (which would then work on all subsequent loads). This is because even though RAW does not have constraints (primary key etc) it follows the same column naming and datatypes as LIVE. Cases 2 and 3 would fail when trying to move from the unconstrained RAW to the constrained STAGING bubbles. In both cases after the load crashes the RAW database would be left accessible so the analyst could run SQL queries against it to explore the problem data directly (without having to resort to peering at the source file).

### Data Load Engine Error Messages

The DLE tries to give as much information about problems as possible. Every message generated by the system (either during pre-load checking or loading data) is logged to a logging server as well as displayed in an interactive data load user interface. It is intended that the data loader user interface be used to debug failing loads and build new loads but that once a load is stable it be setup as an automated run in the Data Load Engine Service.

The DLE gives you messages at every level and aimed at users of varying skill level. For example, a data load failure on column renaming would result in the following messages being generated:

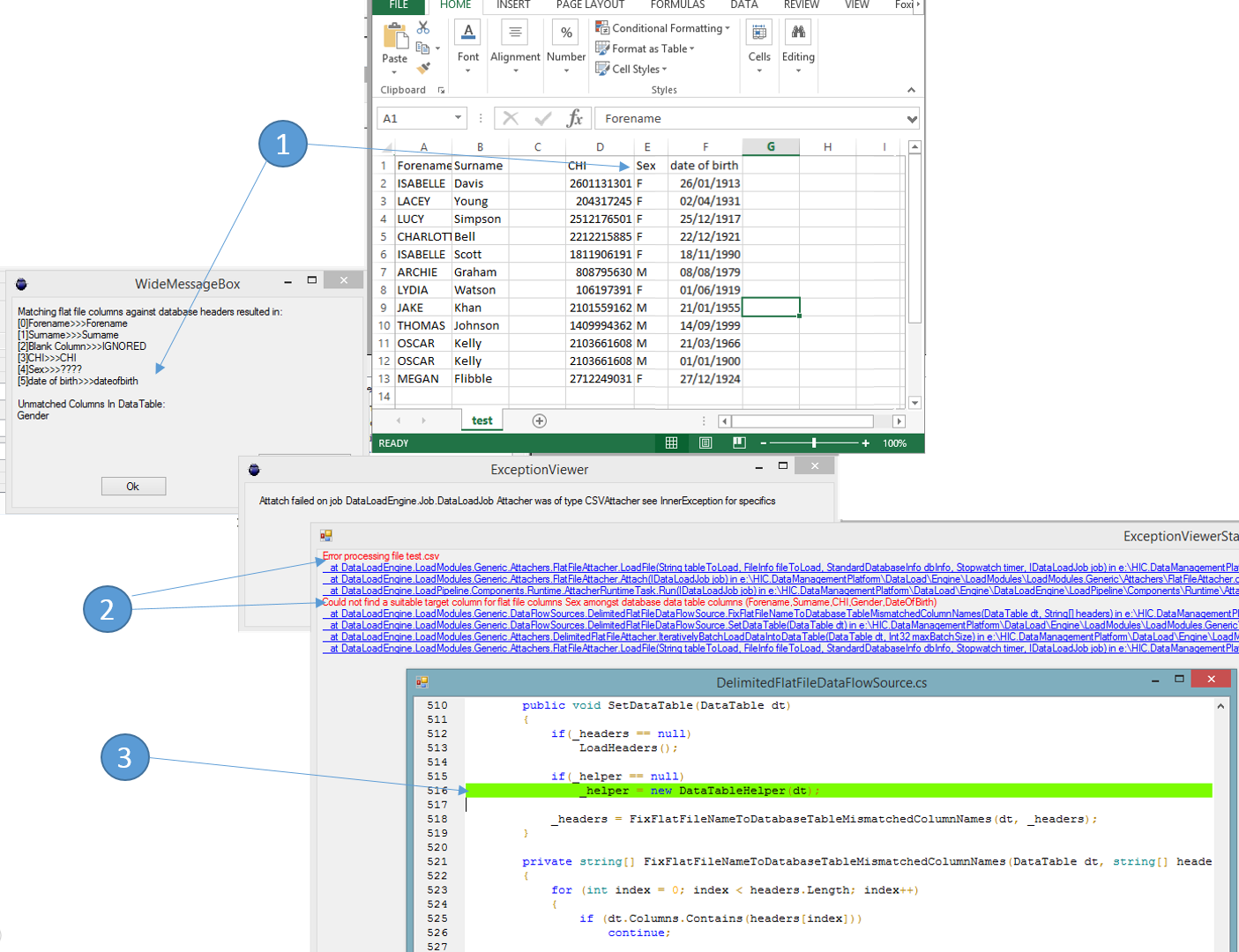


Figure 28 - Messages generated by the data load engine on failing to load a flat file with a renamed column (data is fictional)

1. The first message generated in the log is double clicked to popup and informs the user of how the system was interpreting the columns as it attempted to match the RAW schema. It resolved the blank column (it ignored it), and it resolved the date of birth field having spaces in it but was unable to match the field called Sex against the field called Gender in the database.
2. The second error message is the failure that resulted from not matching the raw file columns to the raw database and is intended for a more experienced user. It shows the name of the file it was trying to load “test.csv” and describes the exact problem (column ‘Could not find a suitable target colum for flat file column Sex’).
3. For the C# developer the RDMP system actually embeds the source code that matches the build of the RDMP program and you can click a hyperlink in any error or information message to view the actual program source code that generated the error in an internal source code browser!

### Shadow Archive Tables

As part of the data load engines implementation it will create a longitudinal history version (Shadow Archive Table) of each data table being loaded. This is done to provide traceability and reproducability of data extracts. This process is transparent to the data analyst building the load and requires only that appropriate primary keys exist in the source data and are enforced in the repository data table (See Figure 5 - Merging of old and new records during the final stages of data loading).

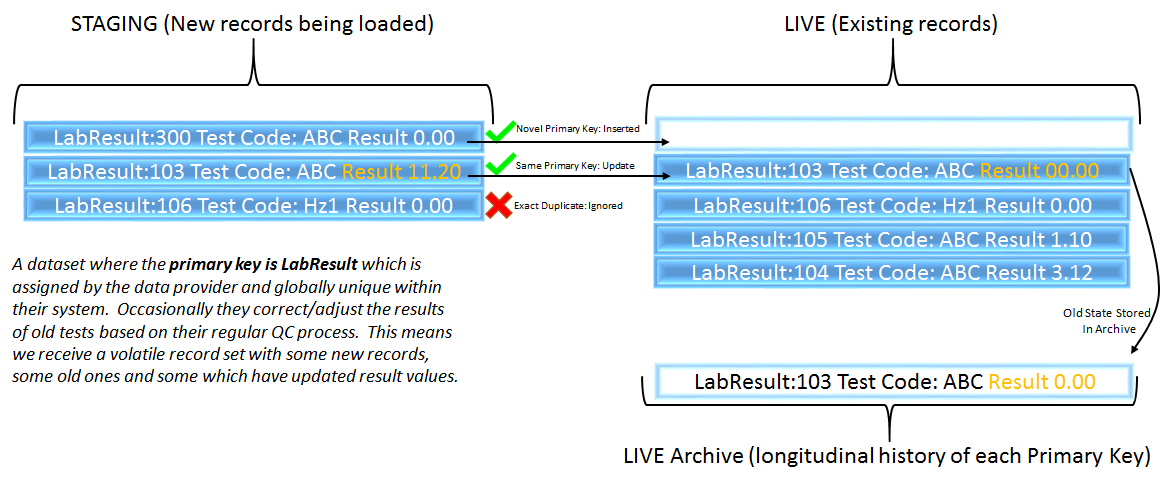


Figure 29 - Merging of old and new records during the final stages of data loading

The technical implementation for this procedure is to (within a database transaction) issue an INSERT INTO sql command for all unique primary keys (very fast) and a MERGE sql command on the primary key (also very fast). The versioning of old records is achieved using an SQL Trigger which is triggered on UPDATE and inserts the record into the archive table. Of these processes the archiving trigger is the only one which does not scale linearly with dataset size/available compute. The trigger scales by dataset load variance e.g. if you load the entire dataset each time and the dataset is 3 million records but each time a single field changes you will not only have a very slow load but also completely replicate the entire dataset each time into the archive! Therefore, it is important to understand the data pattern and treat it accordingly (for example if you have a record time stamp created by the data provider you can either drop the field or delete exact duplicates from STAGING in the adjust staging section of your load). As a last resort (though not advised) you can issue a command during the load to disable the archive trigger.

Along with the archive table the data load engine will create a ‘way back machine’ table valued function which will merge records from the archive and the live table to produce a view of what the table looked like at the supplied time:

|  |
| --- |
| select \* from DMP\_Test.[dbo].[TestTableForDMP\_Legacy]('2016-01-01') |

## Creating a Data Load Configuration

### Background

Data is loaded through a data load configuration (Called a LoadMetadata). A given configuration will load one or more tables which must have primary keys. The tables loaded includes all the tables that underly the Catalogues ­associated with the LoadMetadata. For example if a Catalogue ‘Biochemistry’ includes doing a join on two tables ‘Header’ and ‘Results’ then a LoadMetadata which includes ‘Biochemistry’ would include both the ‘Header’ and ‘Results’ tables.

### Prerequisites

Make sure you have imported the 3 test datasets (See Setting up Datasets) or have a dataset of your own already imported that you want to configure column extractability on.

Next generate another set of test data files which we can load into the datasets.

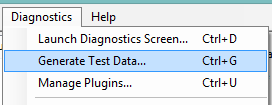


Figure 30 - How to generate random test data for use practicing with the RDMP software

Finally since we require primary keys on the datasets you will need to add a primary key through your SQL editor (e.g. Sql Management Studio). The Biochemistry test dataset can have primary key of chi (patient identifier), Sample\_date and Test\_code. Prescribing should be given the primary key chi, prescribed\_date and name. Finally demography should have the primary key chi, dtCreated (See Figure 32).

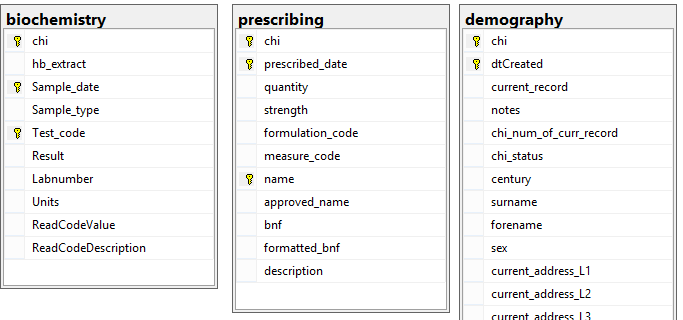


Figure 31 - Primary keys for test dataset

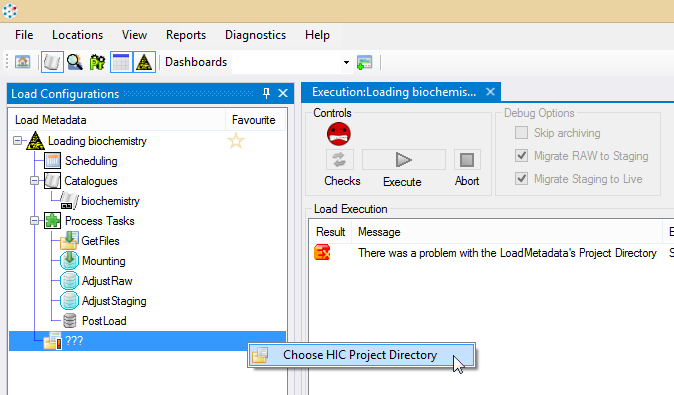
### Creating an Empty Load

Click ‘Create New Data Load Configuration’ on the Home screen and select biochemistry. This will create a new load configuration (LoadMetadata) called ‘Loading biochemistry’ and a logging task ‘Loading biochemistry’ in the logging database. RDMP will then open up the editing screen configuration.

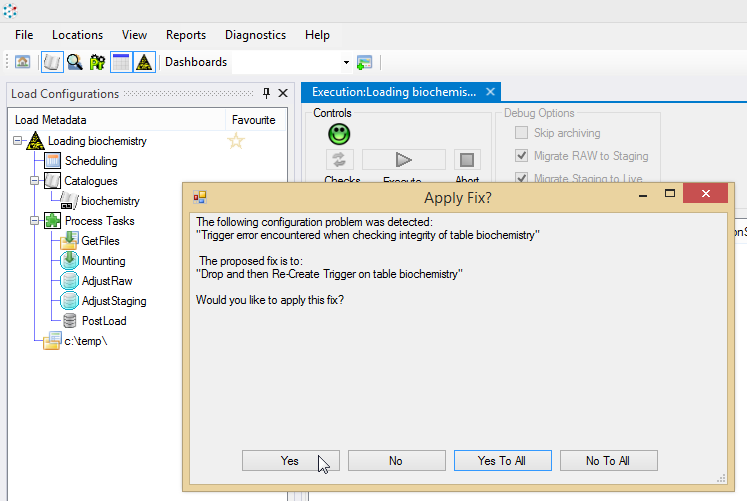


Figure 32 - Load metadata editing screen configuration

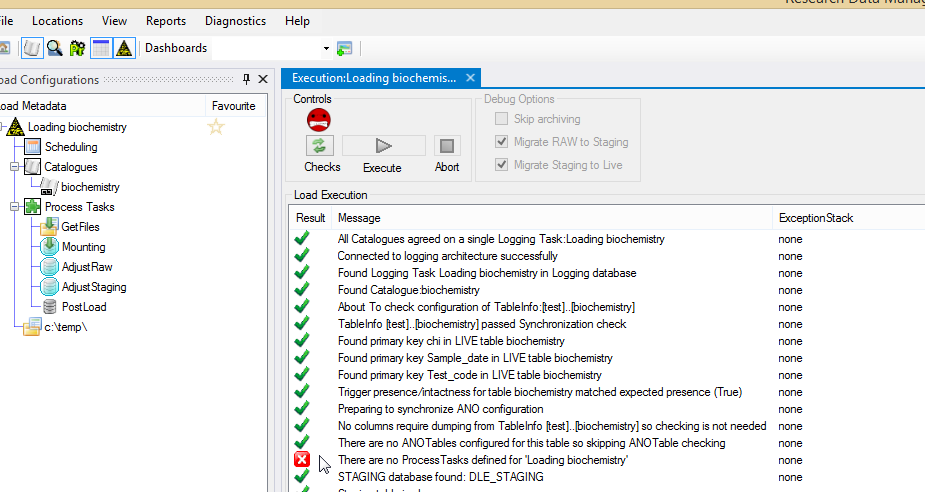
Start by clicking ‘Checks’. The first problem detected by the RDMP is that there is no ‘Project Directory’. Right click the directory icon under the load and create a new project directory (e.g. in c:\temp\myload).



Now when we click ‘Checks’ the RDMP will detect a problem that it can solve. Namely that there is no shadow archive table configured for your dataset (or if you do not have any primary keys configured it might complain about that). If a problem can be solved by the RDMP then a ‘Apply Fix’ dialog will appear. ALWAYS READ THE FIX before applying it. In this case the solution is to put a trigger on the table and create an \_Archive table. If this was on a highly volatile production table then you really don’t want to do it and might want to rethink your load strategy e.g. create a load that instead targets a static ‘research ready’ extract table.



If any fixes were applied you will still need to re-run checks until they pass. After this you should only be getting 1 error ‘There are no ProcessTasks defined for ‘Loading biochemistry’. This is because we haven’t defined what data to load, where from or what file types etc it comes in.



### Adding Load Modules

RDMP supports doing pretty much anything you like during a load. You could could have plugin module which contacts a webservice and downloads new records direct from a network socket and populates the RAW tables directly. Or you could have a module which fetches files from an FTP server and another which reads them from forLoading. Fortunately most commonly required operations are provided by RDMP as ‘out of the box’ modules (e.g. FTP download, reading excel files and csv/tsv etc files). These modules are optimised for dealing with the kinds of problematic files typical of a research environment (columns being renamed, aribitrary errors in file layouts etc, unescaped fields etc).

In the case of our test example we will use an AnySeparatorFileAttacher to read a comma (,) seperated file - namely the biochemistry.csv file you created above (See 5.2.2). Assuming this is not identical to the file you created when you first added biochemistry then the widths of columns may vary vs the live dataset (since the data in the files is randomly generated). This gives us a good opportunity to explore how RDMP reports problems. If you are loading biochemistry with a live data table varchar(10) and you get a string field of width 11 in a file you are loading what should you do? If it is a patient identifier then you might want to bin the record since all chis are 10 in width or invalid. If however it’s a ReadCodeDescription then you might just want to edit your live table to tolerate wider values (e.g. varchar(max)).

Copy the ‘biochemistry.csv’ file into the ForLoading directory of your loads working directory (you can open this by double clicking it). Make sure not to open/save the CSV in Microsoft Excel since it does horrible things with csv files like rounding values and trimming leading 0s (also depending on the size of the csv it might not open at all).

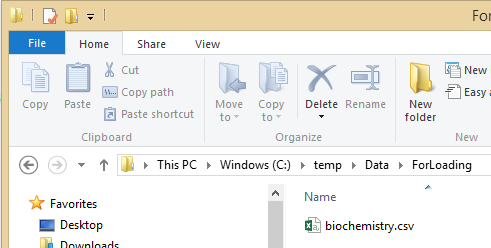


Figure 33 - Copy the csv file to ForLoading

To read the csv file we will use an ‘AnySeparatorFileAttacher’. This module will load any files in forLoading which matches a given file pattern (e.g. \*.csv) and read it as a file deliminated by the supplied Separator e.g. comma.

The module is an Attacher which means that it is designed to populate RAW with data. We will add it to the ‘Mounting’ stage which is the load stage concerned with getting data into RAW.

Set the Separator to , (literally type a single , and nothing else). Set the File Pattern to \*.csv. Set the Table Name to biochemistry. Then set the Name to ‘loading biochemistry.csv to RAW’ and save the changes. This should result in green smiles for the checks.

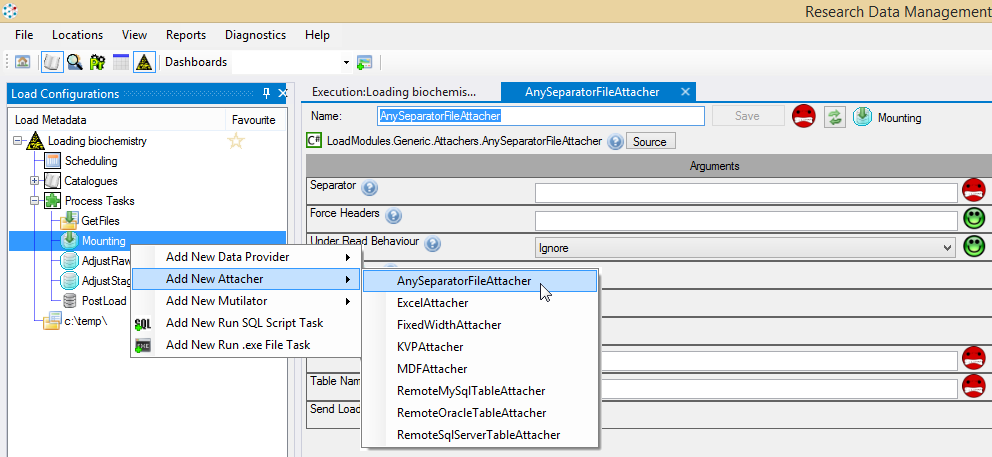
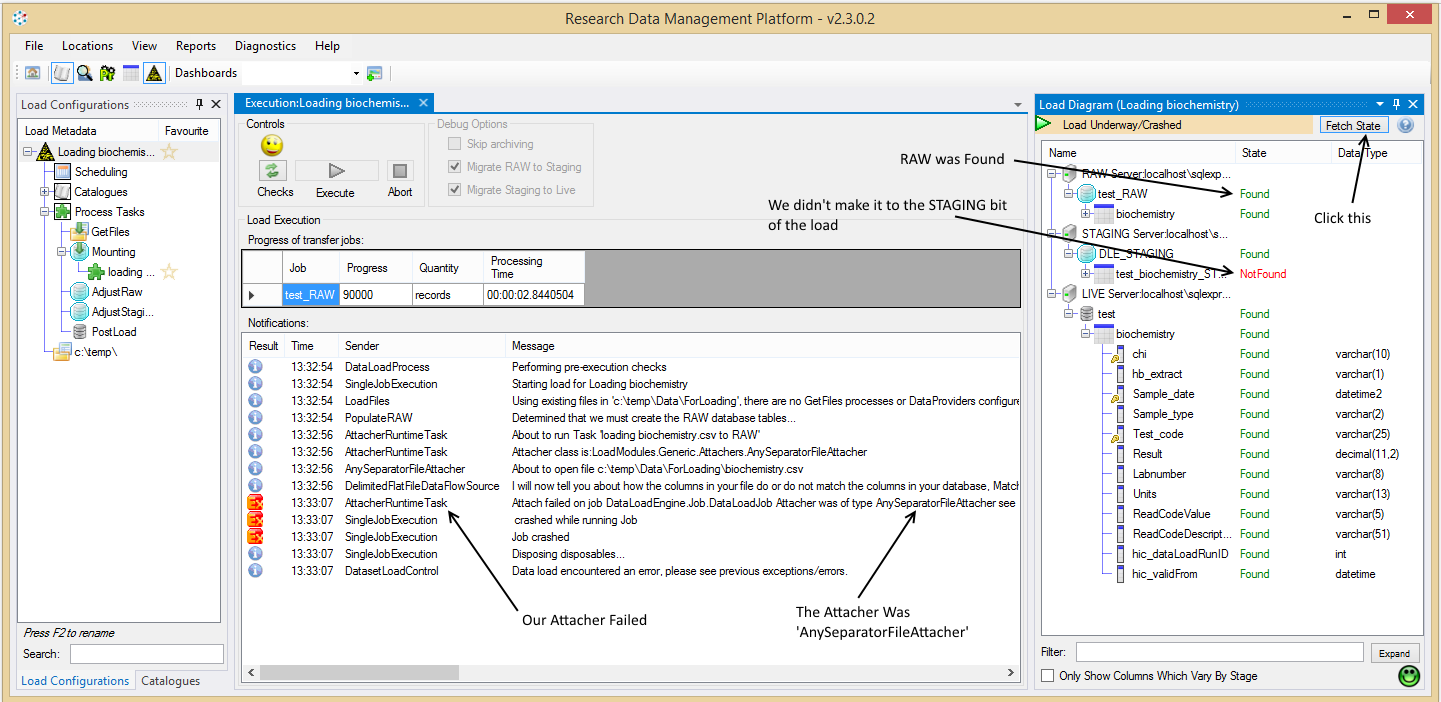


Figure 34 - Adding an AnyseparatorFileAttacher

Close the current tab so that you can see the Execution window again (or right click the LoadMetadata and select execute, or go to home and select ‘Edit Existing Load Configuration’).

Re run the load checks and click Execute.



Click FetchState in the LoadDiagram to see what tables exist and what data they have in them. Then double click the crashing message. The error should include a section something like (your exact message will depend on what random data was generated).

Second Pass Exception:BulkInsert complained on data row 8873 the complaint was about column number 4: <<Test\_code>> which had value <<THYROID RECEPTOR ANTIBODIES>> destination data type was <<varchar(25)>>

Fix this by resizing your live database table ‘biochemistry’ to have varchar(50) for Test\_code and varchar(50) for Units and varchar(100) for ReadCodeDescription. You will also need to resize the columns in \_Archive shadow table too (If you don’t do this the checks will identify this as a problem but won’t automatically fix it).

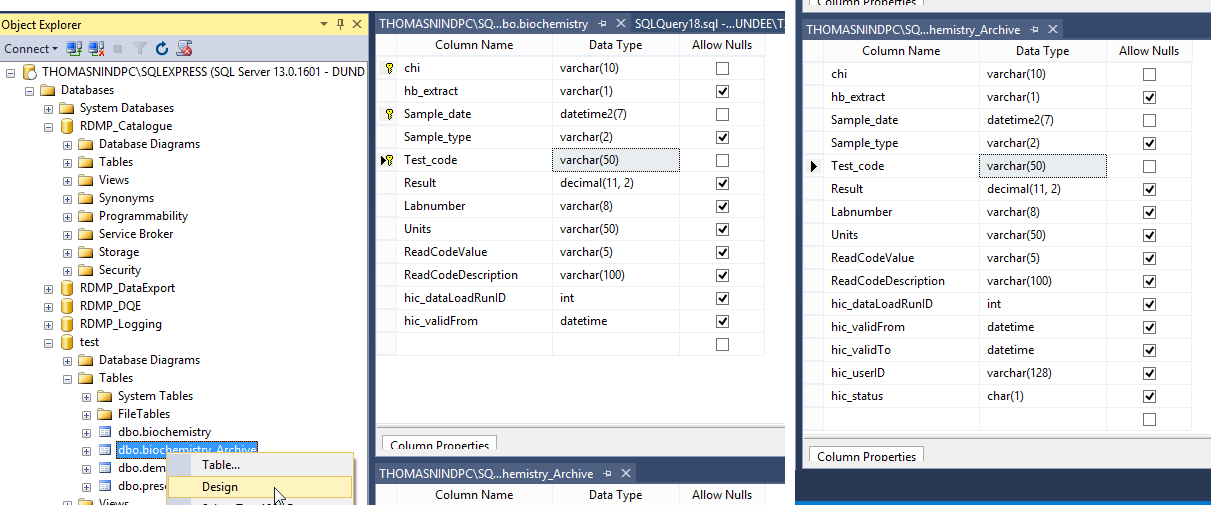


Figure 35 - Resize columns to allow the wider data in

Go back to RDMP and Re run the load checks. This will want to sync the ColumnInfos that you changed to the new datatypes it discovers. Then it will want to drop RAW. Cleaning up RAW/STAGING is an important part of preparing to re-run the data load after a failed data load (when a load is succesful this happens automatically). Since the load failed in RAW population we don’t have to worry about any data having hit our live databases as a result of the failed load.

Assuming there are no further errors you should find the data load completes succesfully and the LIVE data table biochemistry has the new records. Also note that the ForLoading directory is now empty and the ForArchiving folder contains a zip file (named after the dataLoadRunID) containing the biochemistry.csv file.

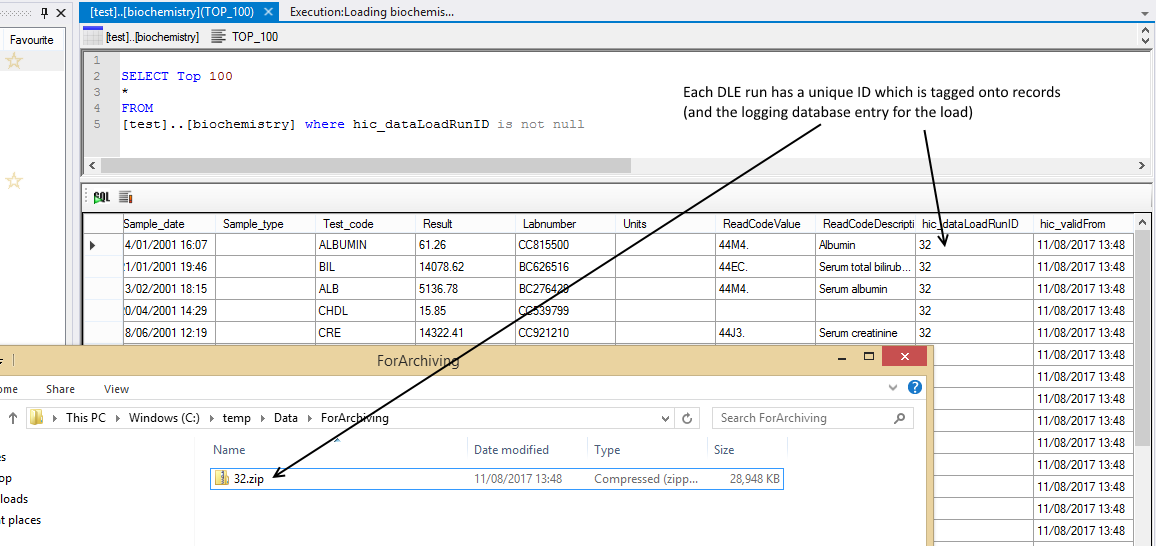


Figure 36 - Archived loaded file

### Record Versioning

In order to see how RDMP handles duplication / updates we can load the same file again with a script that introduces some artificial differences. Copy the biochemistry.csv file out of the archive into ForLoading. Add an SQL Script in AdjustRaw called ‘introduce differences.csv’ and update the [Sample\_type] column to null. Since this runs in AdjustRaw it will only affect the data in the batch being loaded.

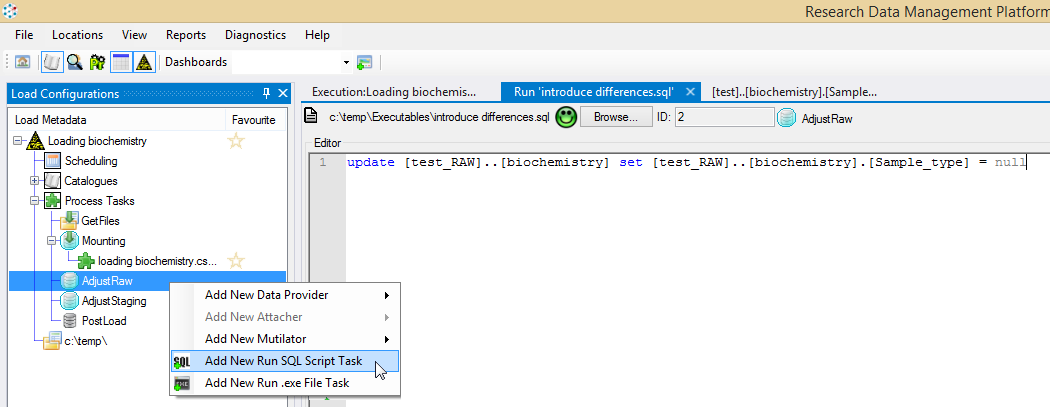
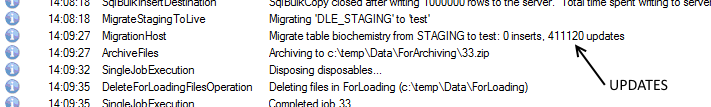
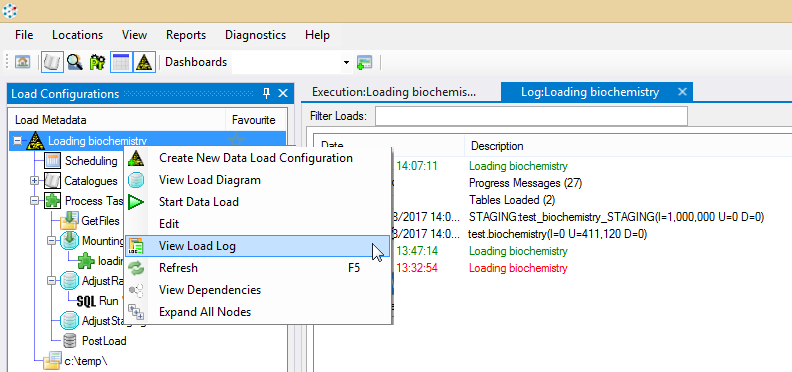


Figure 37 - Adding an SQL task

Execute the data load (re run the checks if you need to) and notice the results. While the file contained 1 million records (in this eample case), there were only about 40% as many updates to the dataset.



This is because Sample\_type is only populated in about 60% of records anyway which means that even with the adjustment script nulling every record the actual delta vs live records is only 40%. We can see which records were affected by right clicking the LoadMetadata and selecting View Log.



Right click the table test..biochemistry and select ‘View Updates/INSERTS’

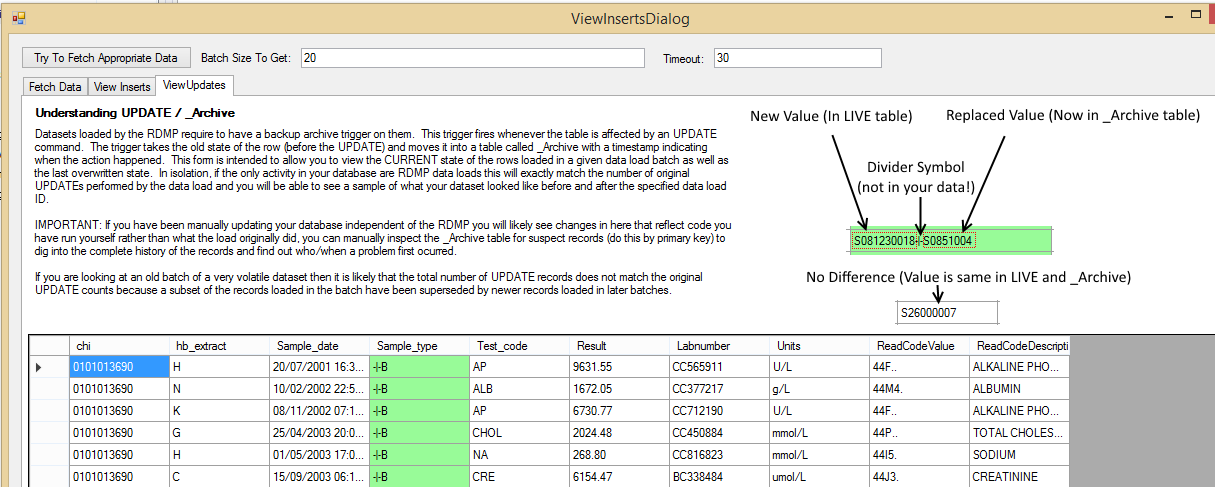


Figure 38 - Differences introduced in the load

You can get a view of the dataset at any time by running the table valued function:

--what it looked like before we did any loads

select count(\*) FROM [test].[dbo].[biochemistry\_Legacy]('2001-01-01')

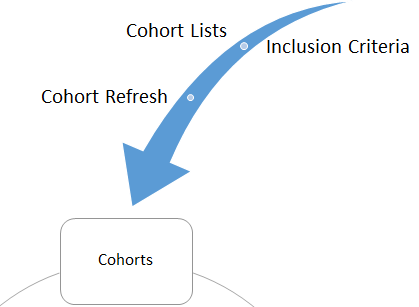
--what it looks like now

select count(\*) FROM [test].[dbo].[biochemistry\_Legacy](GETDATE())

Note that because the first record batch was loaded by bulk insert there is no hic\_validFrom or hic\_dataLoadRunID (in fact these columns were only added during the data load creation steps) the initial records never ‘came online’ and will always appear regardless of how early a date you give (unless they have been overwritten by updates).

If you want to try and unwind a data load that has been succesfully commited you might try something like deleting the offending records (hic\_dataLoadRunID = x) from the live table and then inserting the \_Archive records back into the LIVE table). Attempting to ‘unwind’ a data load in this manner is NOT recommended. You should have a backup/recovery method for your databases and something like this should be treated as an absolute last resort. If you are set on attempting this, make sure you disable the Archive trigger (e.g. DISABLE TRIGGER [biochemistry\_OnUpdate] ON biochemistry) or will get in a circle where your DELETE commands just further populate the \_Archive.

# ­Functionality - Cohorts



## Introduction

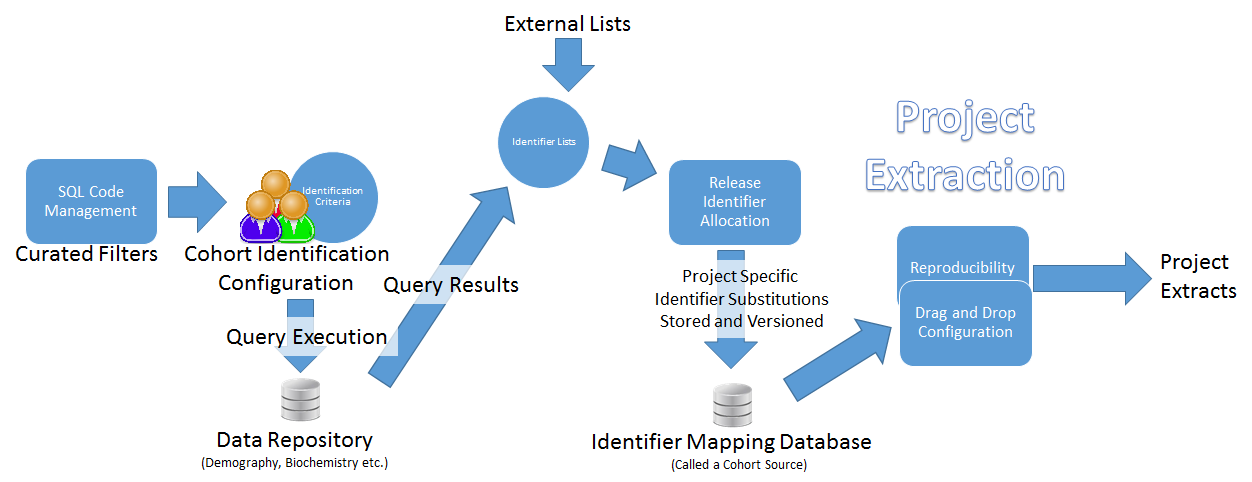
The RDMP defines a cohort as a list of identifiers which uniquely identify subjects and can be used to do dataset linkage, anonymisation and extraction. This chapter relates to how you arrive at your final project cohort identifier list and how RDMP supports this activity.

Cohort identification is one of the most complicated parts of meeting projects’ extraction needs. It is also very sensitive and has considerable risk potential (for example incorrectly contacting patients about conditions they don’t actually have).

## Cohort Lifecycle

Cohorts are central to the extraction functionality provided by RDMP. This means that while Cohorts can be considered as a piece of functionality in it’s own right it is integrated tightly with the following satellite functionality.

Each Cohort starts life either as an ‘External Identifier List’ or a ‘Cohort Identification Configuration’. An ‘External Identifier List’ is the easy use case when your researcher knows exactly what private identifiers they want to extract data for (e.g. from doing a patient case note review). More often however you have a set of ‘Identification Criteria’ (e.g. ‘all patients with Type 2 diabetes currently living in Tayside on drug X). Identification Criteria are built in RDMP in a ‘Cohort Identification Configuration’ (See Cohort Identification Criteria), this is done by dragging and dropping curated Filters and Datasets and arranging them in SET operation containers (e.g. inclusion / exclusion criteria). Each part of this configuration can be tested individually (See Figure 41) to ensure the configuration accurately reflects the researchers requirements. Once you have finalised your ‘Cohort Identification Configuration’ it is executed on the live data repository and the resulting identifiers are treated exactly like an ‘External Identifier List’. At this point the Cohort Identification Configuration can be optionally frozen so that it cannot be edited/used again (although it can be cloned). The primary reason you might want to freeze the configuration is to preserve reproducibility because although the final identifier list is snapshotted and versioned you might need to go back and debug your configuration for errors/refinements and you don’t want people to have modified it since the project extract was done.



Once you have an ‘Identifier List’ (either via a Cohort Identification Configuration or directly from an External source) you can import the private identifiers into the Cohort Database, this contains all the private and project specific release identifiers for the project patients that you will extract data for in your Project Extraction. The reason we store the identifiers as a static immutable list is so that you can always reproduce a Project Extract exactly even years later. If you identify a problem with a cohort list, the prefered method for refinement post data extraction is commit a new version of the cohort (V2, V3 etc).

The RDMP is designed to allow maximum flexibility on how you allocate your release identifiers and how you treat your cohorts (See Functionality – Release Identifier Allocation) . The Cohort Sources RDMP offers out of the box generate GUIDs but you can modify them or point RMDP directly at your own custom database. This means that you can manually delete identifier lists / assign extraction identifiers yourself if you choose (for example if there is a governance problem around holding an incorrect identifier list).

## Cohort Identification Criteria

### Background

RDMP supports cohort identification by simplifying dividing complex identification requirements into small self contained testable sets. This helps reduce the ambiguity and supports transparency. Consider a simple cohort query as might be requested by a researcher:

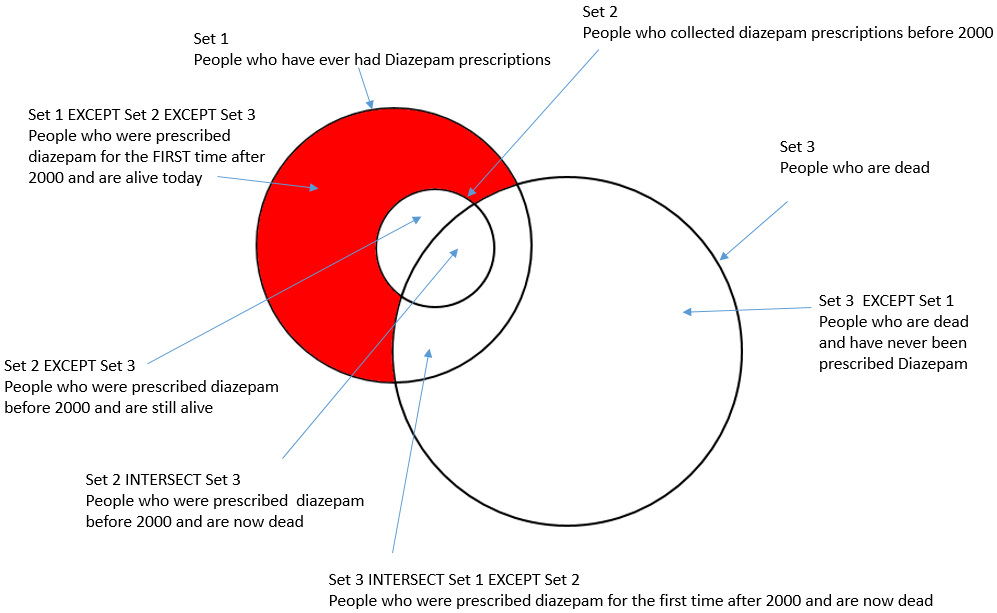
“I want all **patients who have been prescribed Diazepam1** for the **first time after 20002** and who are **still alive today3**”

We begin by identifying each set (see in red). These sets are combined to produce the final distinct patient list. Criteria 3 (‘still alive today’) will be based on the demography dataset while the other 2 will be based on the prescribing dataset.

Each set is built by selecting filters from the Data Catalogue. Let us assume there are some useful filters already set up in the Catalogue:

|  |  |  |
| --- | --- | --- |
| Dataset | Available Filter | Implementation |
| Prescribing | Drug name = @X | name like '%DIAZEPAM%' |
| Prescribing | Prescribed before @date | prescribed\_date < '2000-01-01' |
| Demography | Patient is Dead | date\_of\_death is not null |

In this simple case we can use a single filter per set but in more complicated cases you might need to combine filters (e.g. people who died in Tayside).



The use of set theory gives us many advantages.

* Testability. It is easier to test that your SET of dead patient identifiers is correct than to try and test the entire configuration as a whole.
* Reusability. A well tested SET can be reused later on in different cohort generation tasks
* Performance. Relational database engines are built on set theory so are super fast

### Prerequisites

Make sure you have imported the 3 test datasets (See Setting up Datasets). In addition make sure you have created the suggested filters in ‘Creating some more useful filters’.

### Generating a simple cohort

Right click in Cohort Builder and add a new ‘Cohort Identification Configuration’. Add a name and description and save these (See Figure 29)

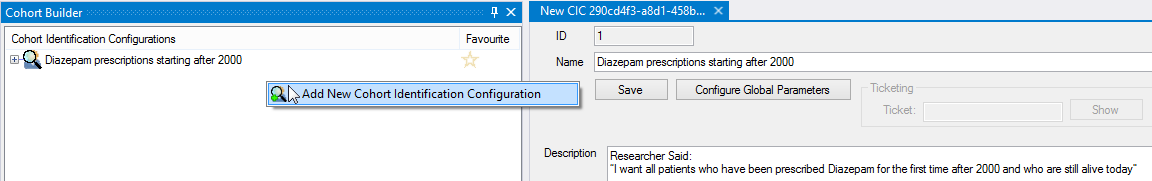


Figure 39 – Adding a new Cohort Identification Configuration

Based on the breakdown of requirements (See 6.2.1 Background above) we know we have to reference prescribing and demograph. Expand the configuration and add the prescribing and demograph catalogues.



Figure 40 - Adding two Catalogues to a cohort identification configuration

We will start by identifing patients who are currently dead. Right click demography and add a filter container then right click the filter container and select ‘Import Catalogue Filter’ and double click ‘Patient is dead’.

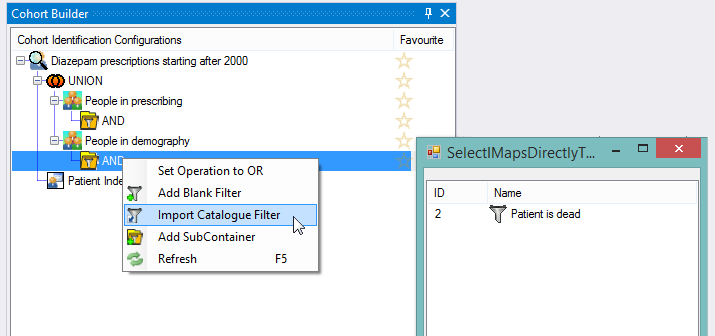


Figure 41 - Importing an existing Catalogue Filter into a cohort identification configuration

Rename ‘People in demography’ to ‘Dead patients’ by pressing F2. It is important to always update the name of a cohort set when you change the logic / configuration of filters. This way anyone checking the configuration can tell what the high level concept is (Dead patients) and then expand the container(s) to see what specific filters are used to achieve the concept.

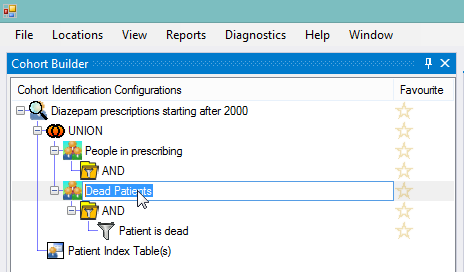


Figure 42 - Renaming a cohort set

For now we will leave prescribing and focus on the fact that ‘Dead patients’ is an exclusion criteria and hence the patients should be REMOVED from the dataset not included. Right click the UNION set operation and change it to an EXCEPT.

|  |  |
| --- | --- |
| UNION | Include ALL patients appearing in ANY of the cohort sets. |
| INTERSECT | Include ONLY patients who appear in EVERY cohort set. |
| EXCEPT | Include ALL patients in the FIRST set who do NOT APPEAR in any of the other sets |

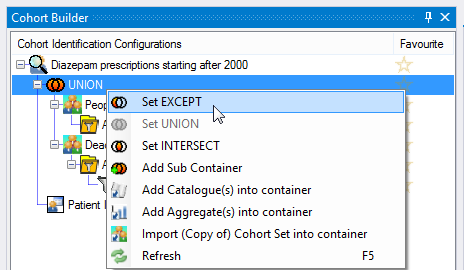


Figure 43 - Changing the SET operation to exclude dead patients

Now right click and execute the configuration. Since the test data is randomly generated the totals you have will vary from those shown below.

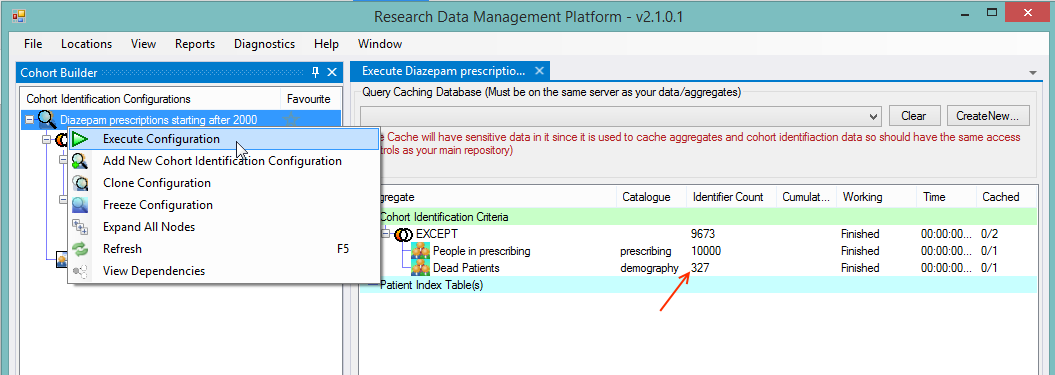


Figure 44 - Executing the Cohort Identification Configuration

In the example below we can see that:

1. There are 327 unique patients are dead.
2. There are 10,000 unique patients who have ever had a drug prescription
3. The final identifier list is the first set (10,000 patients) minus the 327 dead patients leaving 9,673. In the above example prescribing features every single patient making it a super set of ‘dead patients’. In a real world scenario the totals will be A-B because different patients will appear in each set.

Create a new blank filter in the ‘People in prescribing’ dataset with the SQL ‘name like '%DIAZEPAM%'’

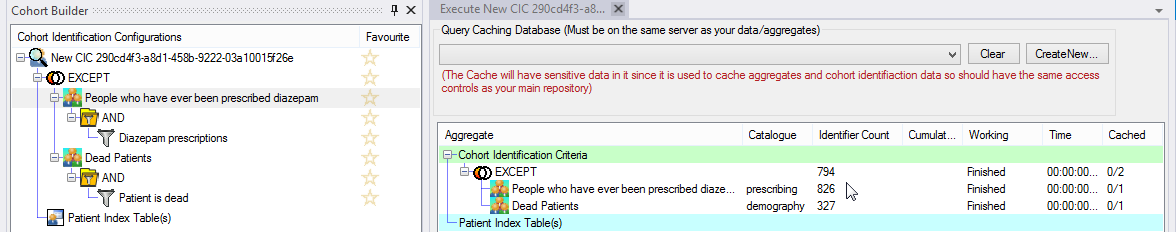


Figure 45 - With Diazepam restriction

Now to complete the requirements we need to exclude people who have diazepam prescriptions before 2000. To do this we will need to import another copy of Prescribing dataset. Right click the EXCEPT container and add Prescribing again. Drag the new Prescribing cohort set into the middle (between ‘People who have ever been prescribed diazepam’ and ‘Dead patients’. Rename it ‘Prescribed diazepam before 2000’.

Finally add two filters into the AND container of the new cohort set, one for Diazepam prescriptions (just copy and paste the SQL code from the first filter) and then import ‘Prescriptions BEFORE Date X’ and set the parameter to 2000 (creating this master filter is covered in Creating some more useful filters).

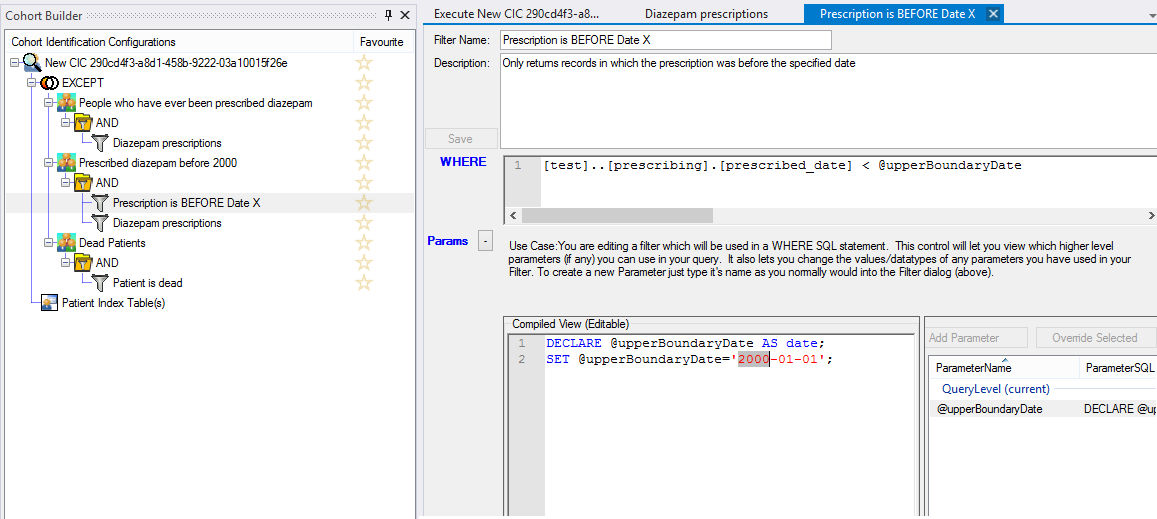


Figure 46 - Setting the parameter of the imported filter to 2000-01-01

Execute the entire configuration but this time tick ‘Include Cumulative Totals’. Once this completes right click the new set and view the Aggregate Graph ‘All drugs prescribed over time’. This will confirm that the cohort set you added is only returning diazepam and only for records before 2000.

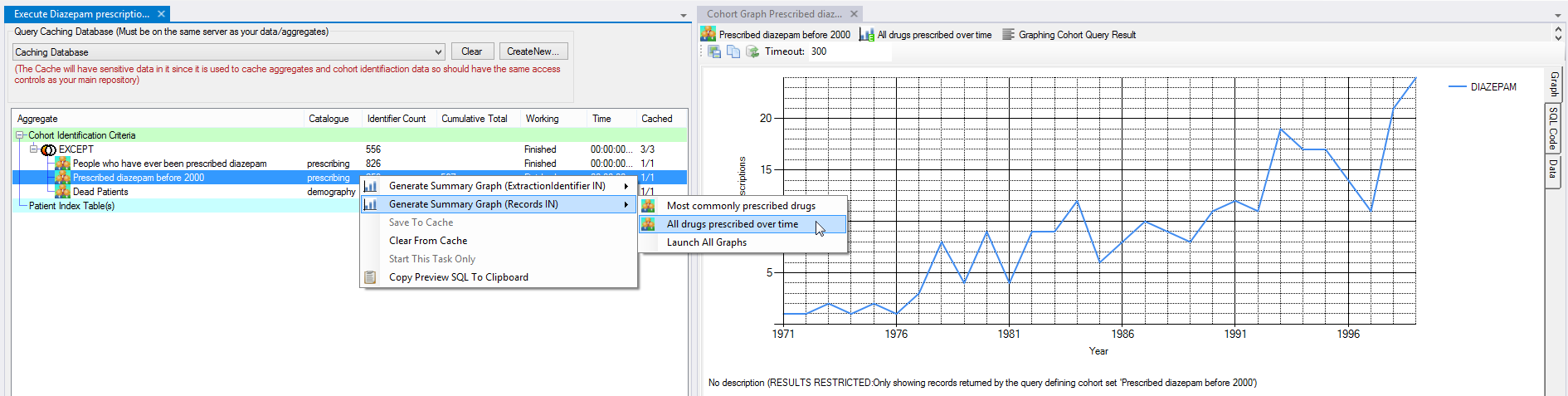


Figure 47 - Graphing the cohort set is one of the best ways of checking that it is returning what you think it is.

The execution view (Figure 42) shows the final answer including totals for each cohort set and the cumulative total. The final cohort contains 556 unique patients. This value is arrived by taking the first set ‘People who have ever been prescribed diazepam’ (826 unique patients) and removing those who were ever prescribed it before 2000 (259) to give 567 (again because the first set is a super set of the second the numbers match exactly). Finally we discard any patient who has died. The death set contains 327 unique patients but only reduces our total by 12 (See the cumulative total column) because ‘Dead Patients’ matches every patient who is dead, not just those who have diazepam prescriptions.

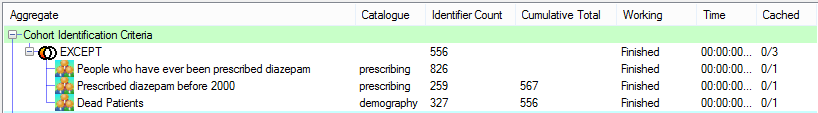


Figure 48 - Final answer to the Cohort Identification task

Notice that we have never actually seen any patient identifiers throughout this task as all the SQL queries have been assembled and executed automatically by the RDMP. The final identifier list can be imported as an Extractable cohort into the cohort database (See Functionality – Release Identifier Allocation).

### Scalability

Since each cohort set is built behind the scenes using SQL the entire process is very fast even for datasets with hundreds of millions of rows. This is however dependent on appropritate indexes etc existing on the Sql Server database/tables.

To further improve performance RDMP supports the creation of a caching database in which each cohort set can be executed and stored into before applying the SET operations (EXCEPT / UNION / INTERSECT). This allows truly huge configurations including dozens of cohort sets to execute quickly without the use of temp tables. The cache is transparent and automatically versions the query that produced the result set such that if a set is changed the cached result is automatically cleared.

## Identifier Lists

Sometimes a researcher will simply have a list of patient identifiers that they want extracted. These may be private identifiers collected via patient review or from a live clinical system or maybe release identifiers they were given previously as part of a data extraction.

RDMP allows you to import files directly to create cohorts. Simply right click the ‘Cohort Source’ and choose your file (which must have a column containing either the private or release identifiers). If you do not yet have a ‘Cohort Source’ read the section ‘Out of the box identifier allocation’.

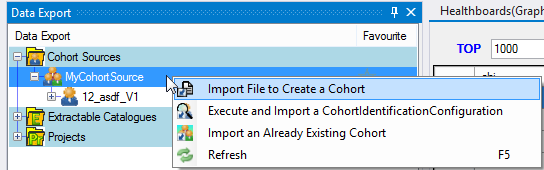


Figure 49 – Importing a Cohort Identifier List Directly

This will prompt you to enter details of the Project (or select an existing one) and describe the cohort. The actual importing of the identifier list takes place through a Pipeline (See Pipeline Components (Data Flow Component)) which allows you to model any agency specific requirements you may have (such as contacting identifier allocation services, bespoke auditting etc).

## Patient Index Tables And Custom Data

Often a researcher will have collected data which they wish to use as part of their cohort specification. For example, they may have collected questionnaire data and want to link it with your hosted research datasets.

This requires uploading the researcher’s questionnaire data into the Repository and using it as a source for cohort creation. By following a methodology in which data is stored in the format in which it is provided, researchers dataset can be treated exactly like any other data.

The DLE reduces ETL overhead from accepting this file whilst also allowing you to curate metadata (such as which researcher provided the data, when etc.) so that when a researcher has questions about a supplied cohort, you have a clear audit and a permanent record of what they originally supplied.

## Refinements of existing cohorts

Researchers are likely to question the content of their research data extracts and the implementation of their cohort generation requirements. Additionally, you may be asked to reproduce extracts far after the event (e.g. to support reproducing the findings of a paper). To support this, the RDMP ensures full traceability and reproducibility on all data management activities. If a researcher comes to you asking about a specific patient identifier in a record they received 5 years ago, you should be able to point to the exact cohort identification configuration and the implementation logic traced back to a record (or collection of records) in your repository (or archive shadow table if the data has since been deleted) and ultimately to trace that back to the original input file provided by a specific hospital/clinic/researcher.

# Functionality – Release Identifier Allocation

## Introduction

Data Export in the RDMP takes place through the linkage and substitution of a list of cohort identifiers for a (usually project specific) release identifier. A cohort is a list of is a collection of patient identifiers that you want to run an extraction for (See Cohort Lifecycle). These lists are stored in a ‘Cohort Source’ database. The RDMP provides a couple of ‘out of the box’ solutions for release identifier allocation (e.g. guid) but also allows you to code your own solution.

## Out of the box identifier allocation

### Background

The RDMP can evaluate your IsExtractionIdentifier columns and suggest an appropriate schema for storing your patient identifiers. This is important since different agencies can have different formatted identifiers for example a 10 digit chi which can have leading zeroes requires a varchar(10) column while a national insurance number would only need 9. Creating inflated (e.g. varchar(500)) columns is not recommended because it can lead to invalid identifiers being accepted and linked as well as causing potential performance problems.

Release Identifier allocation can either be done as part of the cohort committing pipeline or handled seperately in your database (either with a manual process or by creating a default on the column). This allows the maximum flexibility for existing identifier allocation practices you might have.

### Prerequisites

Make sure you have imported the 3 test datasets (See Setting up Datasets).

### Creating a Cohort Source

Select the File=>New=>Create New Cohort Database Using Wizard (See below).

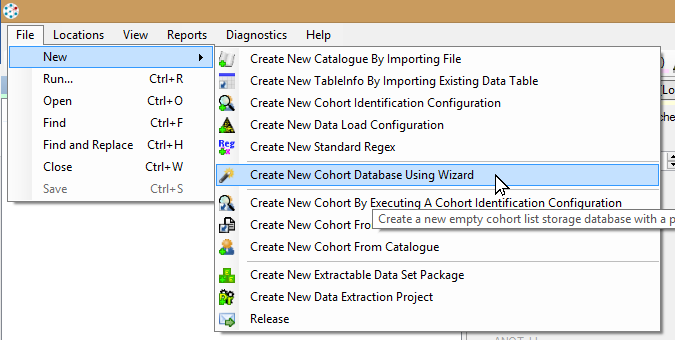


Figure 50 - Creating a Cohort Source Database

Click Next then ‘Attempt to figure out what name/datatype my patient identifiers are’ and pick the most comon / correct data type. If you have multiple unlinkable identifiers (e.g. NHS number / CHI) then you can create multiple cohort databases but you won’t be able to use cohort lists by NHS Number to extract CHI based datasets and vice versa.

You can now execute Cohort Identification Configurations and upload Cohort lists to create Extraction Projects (See Configuring an Extraction).

## Allocation Strategies

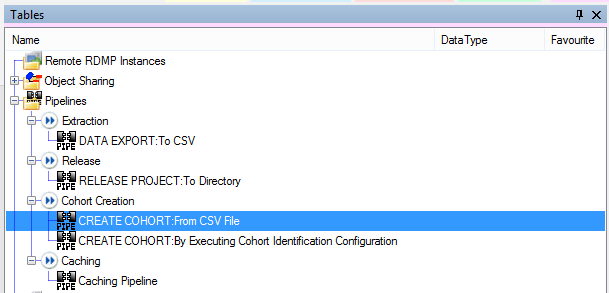
### Background

If you need to allocate identifiers yourself through an existing or proprietary identifier allocation method or need to send identifiers to a third party indexer, the RDMP supports this. Options you might consider include:

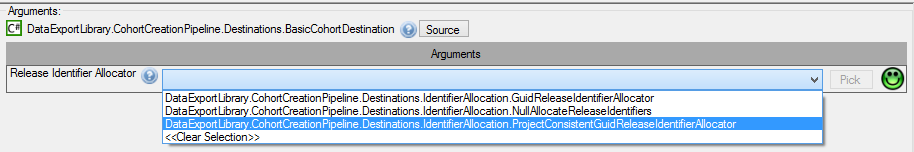
1. Putting a default constraint that calls a scalar function onto the ReleaseIdentifier column
2. Allowing nulls in the ReleaseIdentifiers and writing a bespoke application / stored proceedure for populating them asynchronously.
3. Creating an RDMP plugin which allocates the identifiers during cohort committing

### Changing allocation strategy

To change how release identifiers are allocated double click the Cohort Creation pipeline you want to change (you can have different allocation strategies for different pipelines if you really want).



Select the destination component and change the ‘Release Identifier Allocator’.



* NullAllocateReleaseIdentifiers will not assign anything (this will leave you with nulls in the ReleaseId column of your cohort). To use this you must have allow nulls enabled in the database column. It is assumed that you will run a stored proceedure later or otherwise fill in the ReleaseIds prior to using the cohort to extract data.
* GuidReleaseIdentifierAllocator will assign a new globally unique hex string for each patient e.g. d6689e15-a990-4809-b23a-b7ff2e34d339
* ProjectConsistentGuidReleaseIdentifierAllocator will also allocate a globally unique hex string for each patient but if the patient ID has been seen before in the Project then the previously assigned value will be used. This allows you to submit revised cohort lists into the same Project without the recipient getting all new release identifiers.

## Conclusions

There are as many algorithms for allocating Release Identifiers as there are data linkage teams. The RDMP supports linkage against a private/release column set with a few required columns (project number, cohort description etc) but how you implement this in your database is up to you.

# Functionality – Project Extraction

## Introduction

The data export section of RDMP allows you to link a cohort and a dataset(s) to extract only those records in a dataset that belong to patients in your cohort. This linkage is done through the user interface by selecting components from the data catalogue (which datasets you want, which columns, which filters etc). The configuration (which is permenantly stored in the Data Export database for reproducibility) results in an SQL query which will fetch an anonymise data table(s).

Additionally, the RDMP will make a ‘researcher copy’ of supporting documents and lookup tables as well as extracting the descriptive metadata that corresponds to the extracted columns.

Finally the Data Export Manager supports longitudial versioning of extracts to ensure that changes in column/dataset definitions are rationalised when performing an extract over an extended period of time.

## Extraction Configuration Model

Data Export Manager stores all it’s configuration data for projects in a relational Data Export database. This includes entities such as Projects, Extraction Configurations, Extraction/Release Logs etc. Performing a data extraction starts by identifying a cohort (See Cohort Manager) and importing it into a cohort database. From the perspective of the Data Export Manager tool, a cohort is fundamentally a list of private identifiers of patients which match those stored in your repository. In addition to a unique patient identifier, each patient should have a globally unique and project specific extraction identifier. This project identifier (aka release identifier) will be used as a substitute for the private identifier when the data is extracted and given to a researcher (See Figure 9).

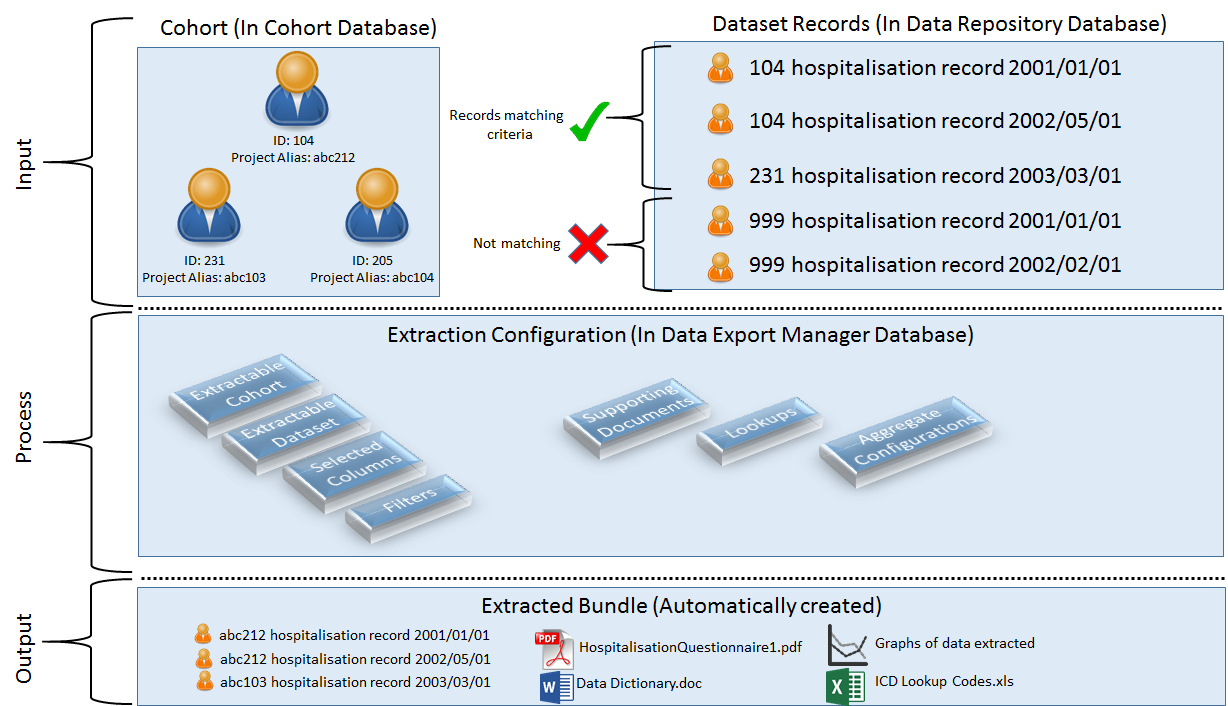


Figure 51 - Diagram showing data extraction process through Data Export Manager

As part of the extraction process, the Data Export Manager will collect artifacts for the dataset(s) being extracted such as supporting documents, lookup tables etc. It will also run each extractable aggregate graph against the cohort so the user can see how his dataset extraction compares to the main dataset. These live vs extract graphs rapidly allow the data analyst and researcher (who will receive the extract) identify any mistakes in the cohort or extraction filter configuration (e.g. study window is wrong, result set accidentally includes Tayside and Fife records when only Tayside records have governance for the project).

## Configuring an Extraction

Configuring a data extraction is designed to be a rapid process and not require in depth knowledge of the specifics of the dataset being extracted. This is achieved by relying on the Data Catalogue database to tell the application what is available for extraction, what the available filters are etc. This means that a senior data analyst with expert knowledge about a dataset can configure the complicated filters required to slice and dice the dataset once (including documenting how to use them e.g. what parameters if any to give them, any perculiarities about the filter). Then anyone performing a data extraction can mix and match these as required without having to understand the implementational logic underpinning them (See Figure 11).

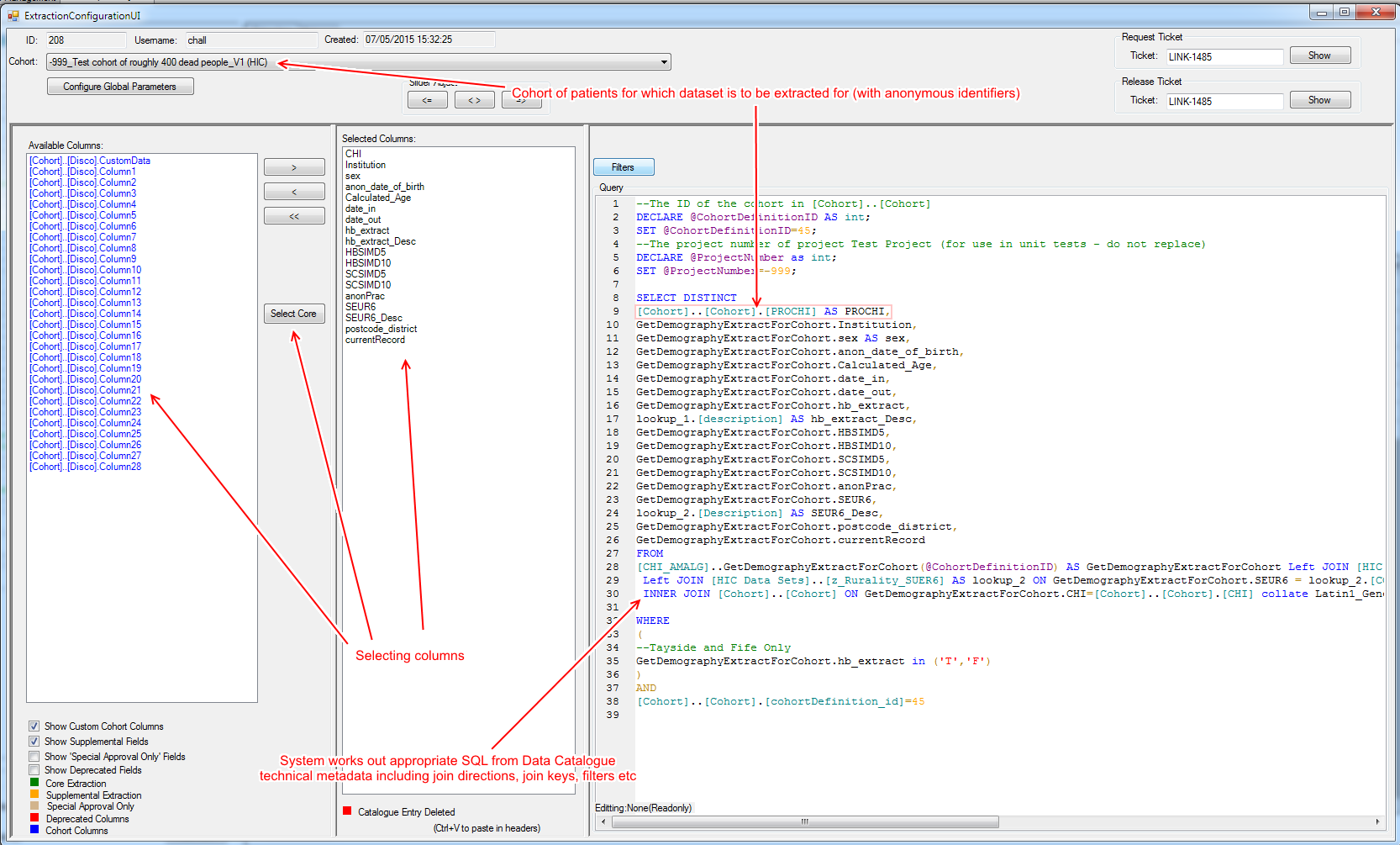


Figure 52 - The screen for adjusting an Data Export Manager configuration for a given dataset (demography)

While reliance on the Data Catalogue for filter/transform logic ensures that there is one central version of the truth, we also have to support reproducability, control for Filter definitions to be refined over time and extracts that require highly specific one off filters that would dillute Catalogue if stored centrally. For these reasons each element (ExtractionInformation, Filter etc) is copied locally out of the Catalogue (or created from scratch in the case of bespoke logic) by the program. The system will alert the user to differences between the Catalouge master and the local extract scoped copy. This allows you to open/clone an old extraction for a project from 5 years ago and provide a refresh of the data in a managed manner that will not surprise the researcher (Records they originally received should not have magically disapeared because of a change in a given fitlers implementation), See Figure 12. It is also possible to take a useful filter that you originally created as a bespoke extraction filter and publish it back to the Data Catalogue for use on other projects.

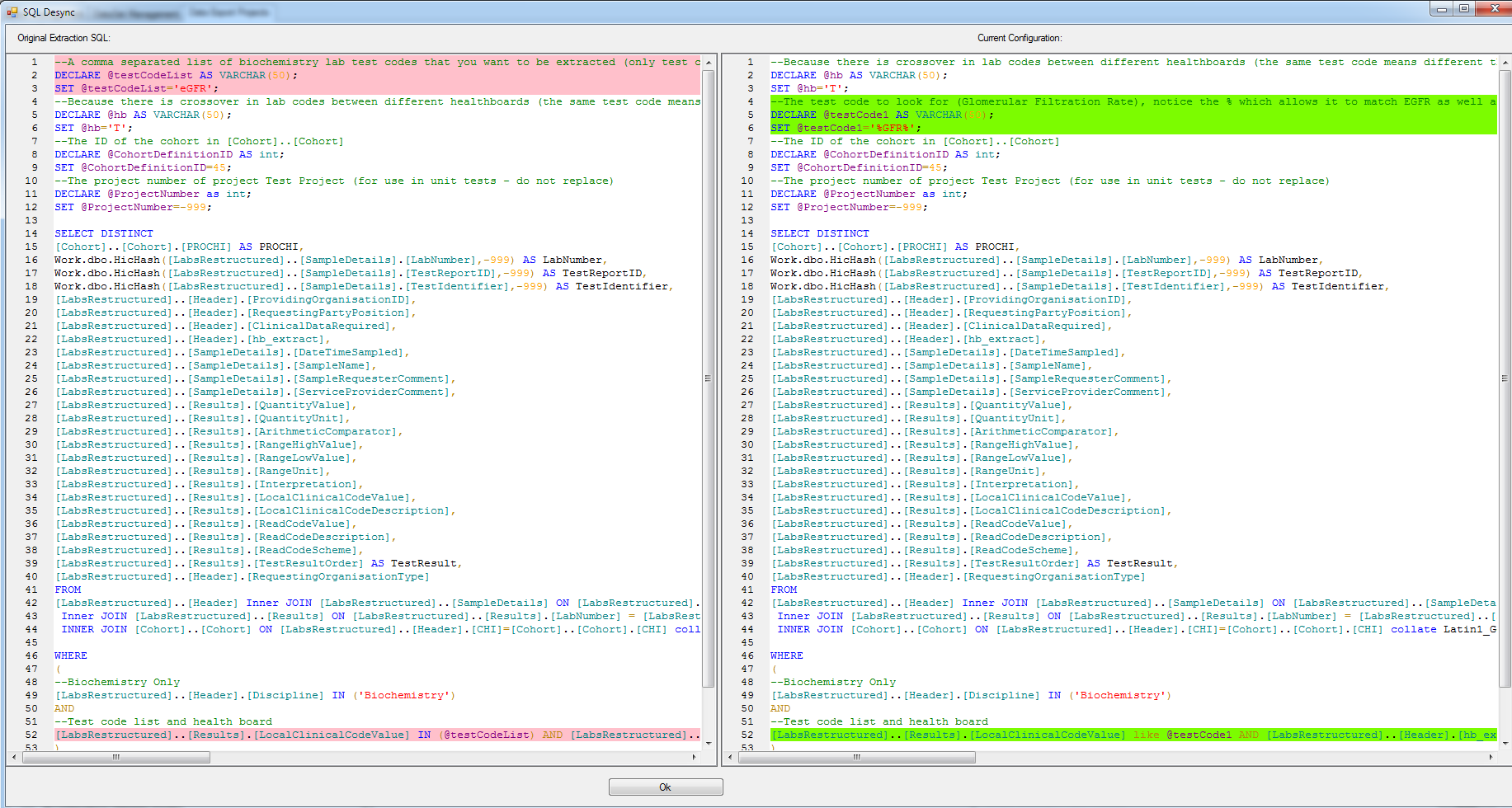


Figure 53 – Summary screen of ALL the Catalogue vs Extract definitions differences. Here we can see that the Catalogue definition of how to identify EGFR Biochemistry records has changed and we have the option to take the new definition or continue using our original one.

## Execution

Since data repositories can contain hundreds of millions of records and cohorts can be equally large, Data Export Manager is designed to be scalable. This includes executing dataset extraction in parallel and streaming results in batches (to prevent RAM problems). The entire process is automated and can be run by right clicking an Extraction Configuration and selecting ‘Execute Extraction’. Extractions are executed through a Pipeline (See A brief overview of what a pipeline is) which allows for agency specific functionality and bespoke buisness practices to be integrated into data extraction (See Plugins and Reusable Pipelines).

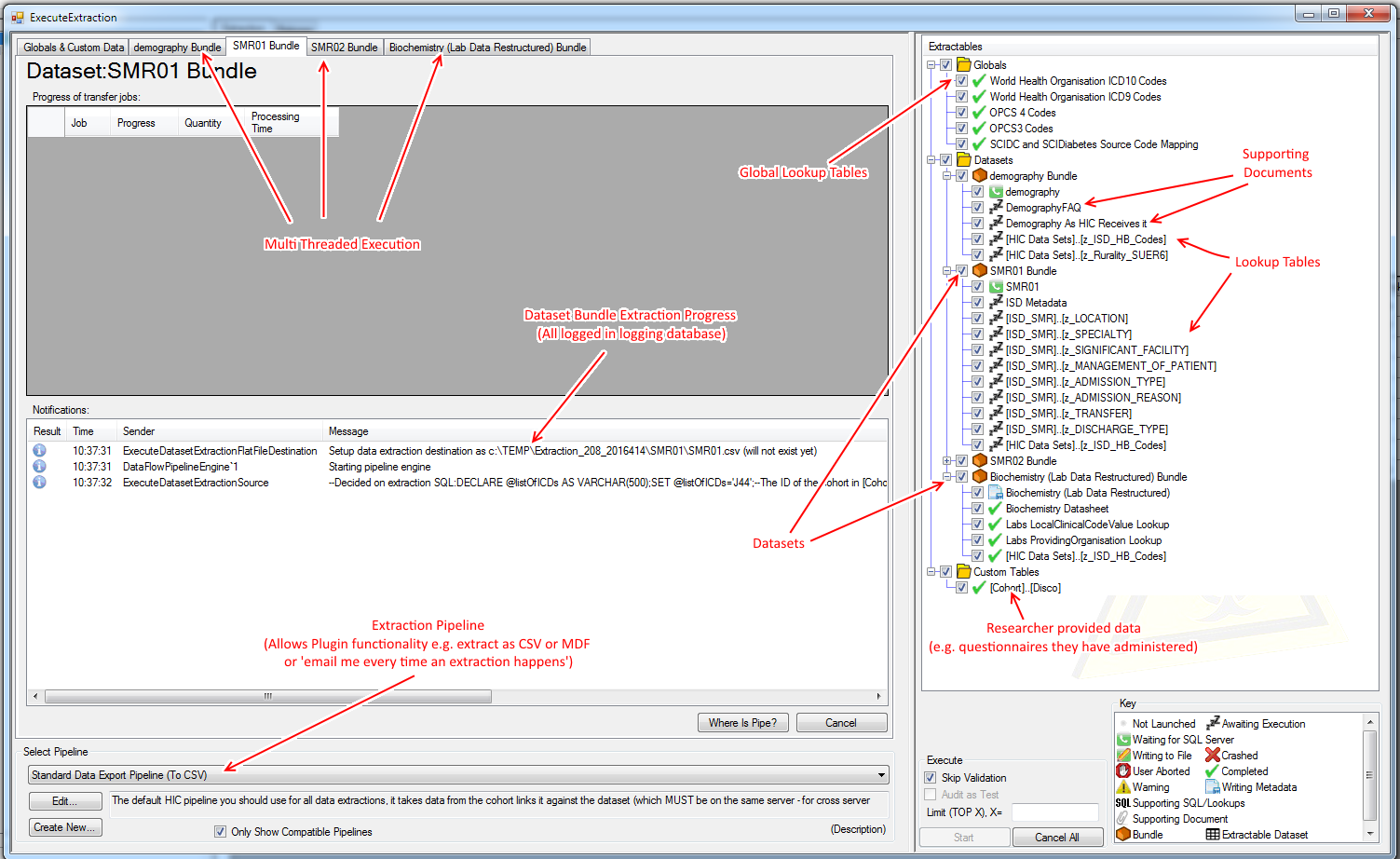


Figure 54 - Dataeset Extraction Execution user interface, we can see that there are 4 datasets in this configuration and that one (Biochemistry) has just completed (and is generating the data dictionary) while demography and SMR01 are still executing on the SQL Server.

## Project Specific Datasets

### Background

Sometimes you will have to host, link and extract datasets that are only needed/permitted for use with a single Project. For example you might have a case not review or questionnaire which was supplied by a researcher. Sometimes this file itself identifies the cohort of patients to extract.

To support such files you should import the Catalogue as normal, optionally you can create a Project/Cohort from the Catalogue then mark it as ProjectSpecific (See Figure 58).

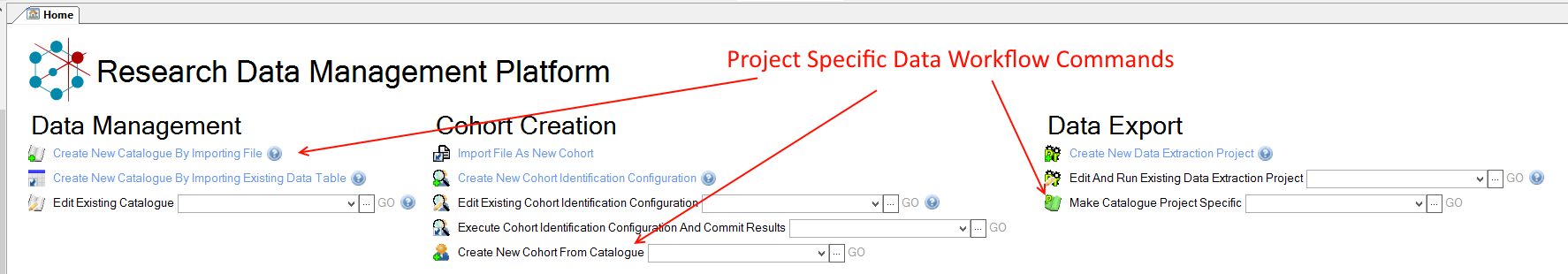


Figure 55 - Project Specific Data Management

### Differences to regular Catalogues

Project Specific Catalogues differ from regular Catalogues (datasets) in the following ways:

* Can only be extracted in ExtractionConfigurations that are part of the Project
* Can be joined at extraction time with regular datasets
  + To add additional columns (e.g. extract Prescribing but append my questionnaire answers to every record extracted – for the patient on that row)
  + To restrict the records extracted (e.g. extract only prescriptions for patients with a Q1 answer of > 5)

In order to make use of joins you will need to configure the join (left / right, which column(s) to join with etc). Usually this is just the patient identifier in the custom data and the patient identifier column in the dataset(s) you are extracting (See Figure 59).

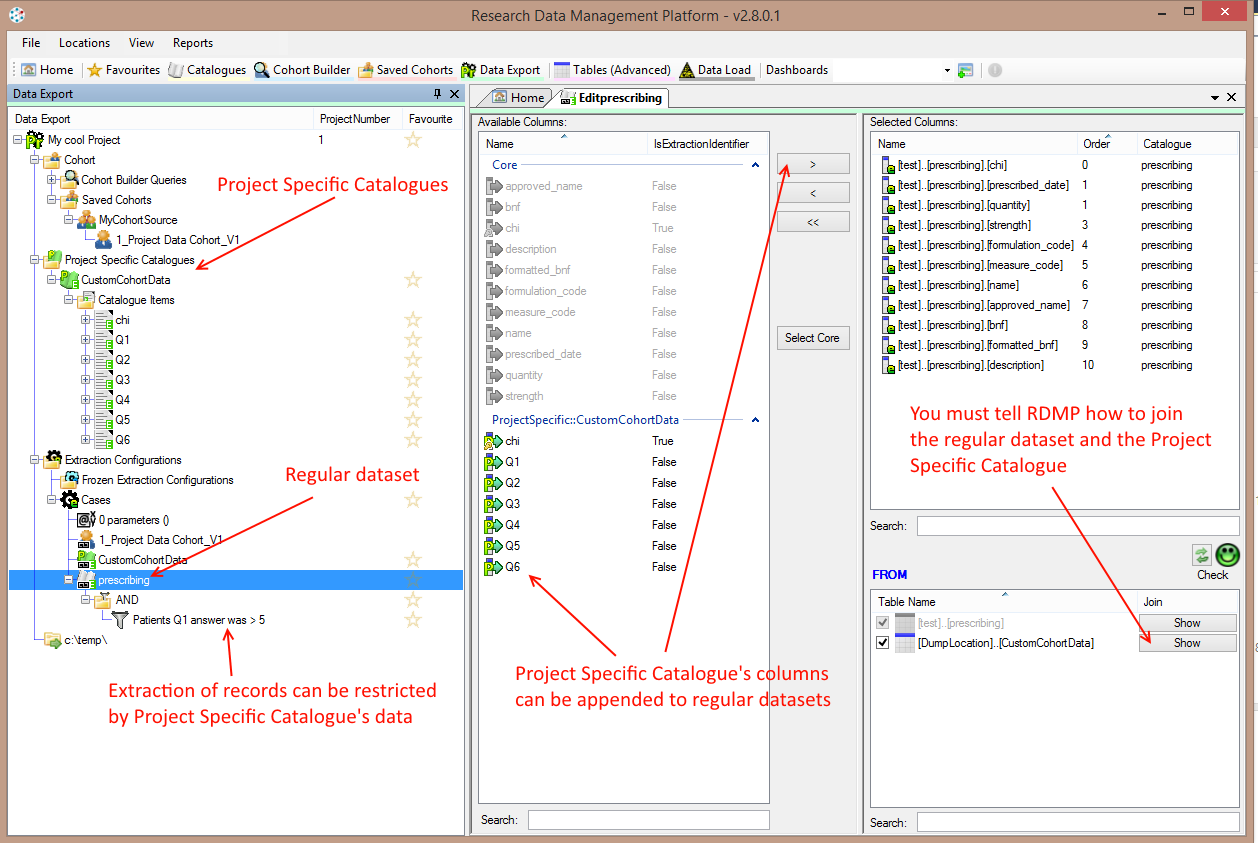


Figure 56 - Combining Project Specific and Regular Catalogues at extraction time

# Functionality - Documentation

## Introduction

At a very minimum you should aim to be able to provide an accurate description of each dataset and columns you make available to researchers. RDMP supports this by allowing descriptions to be stored directly with each metadata object (datasets, filters, extractable columns, cohort identification configurations etc). In addition you can add attachments and Lookup tables.

All this documentation is collected and automatically bundled with each project data extract. In addition, because the descriptions are stored on the same objects used for the other RDMP processes you do not have to worry about updating / synchronising the documentation with your data model.

## Catalogue Documentation

### Prerequisites

Make sure you have imported the 3 test datasets (See Setting up Datasets). In addition make sure you have created some Aggregate Graphs (See Generating a more complex graph).

### Setting a description

Double click the ‘prescribing’ dataset Type in and save a description. Now expand the CatalogueItems node and double click the ‘chi’ column. Enter and save a description. RDMP will detect that there are other columns called chi and allow you to automatically synchronise the description across all the columns. This is useful for maintaining a centralised description for common columns and to avoid having to update many copies.

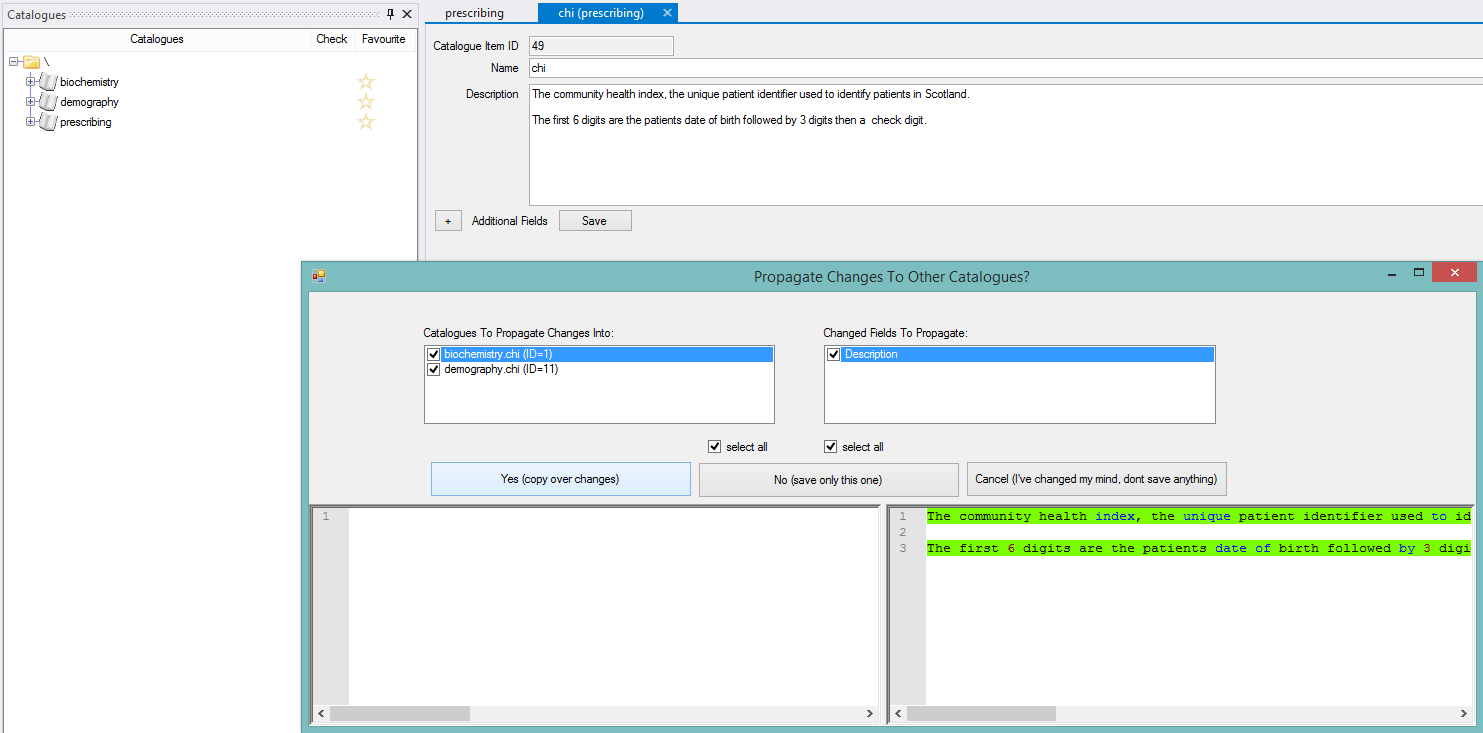


Figure 57 - Synchronising CatalogueItem descriptions by name between Catalogues

Add descriptions for the other columns in prescribing. If you are unsure what a column contains, right click the linked ColumnInfo and click ‘View Aggregate’ or just make up a description (since this is test data).

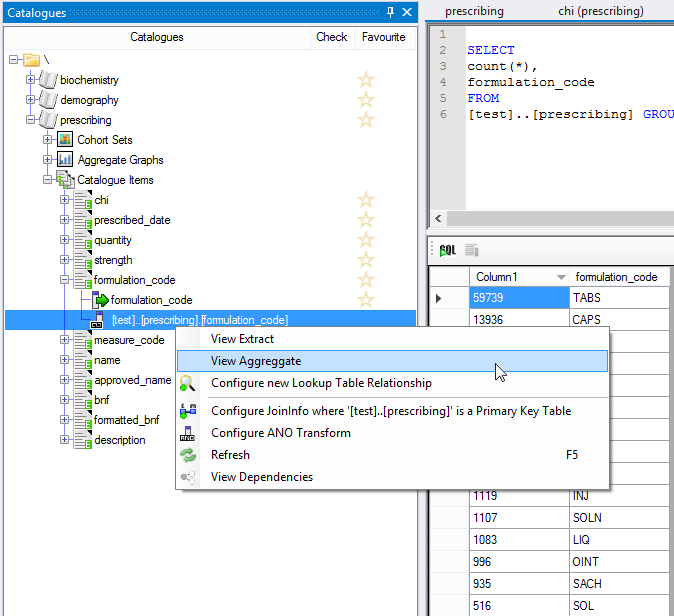


Figure 58 - Aggregate of values in the formulation\_code column

### Generating a Metadata Report

Once you have typed descriptions for all the CatalogueItems in prescribing, select Reports=>Generate=>Metadata Report. And select Only prescribing.

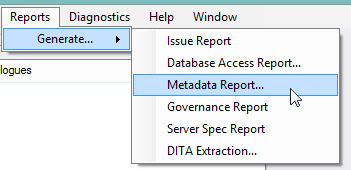


Figure 59 - Generating a Metadata Report

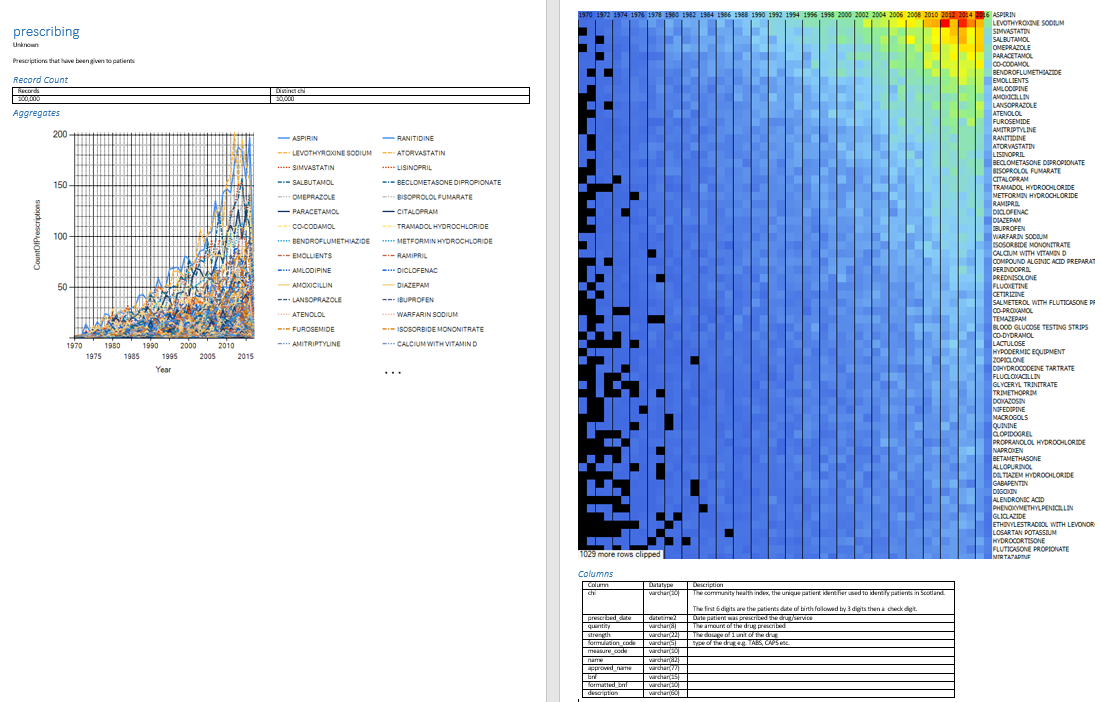


Figure 60 - Resultant Metadata document describing prescribing dataset

The metadata report shows all the descriptive metadata for columns you have described as well as a rowcount, distinct patient identifier count and any Aggregate Graphs that are marked as Extractable.

A variation on this report is automatically generated with each extraction (adjusted to show only those records that are part of the extract).

## Supporting Documents

RDMP allows you to add Attachments to Catalogues, these are files which help either the researcher or data analysts to understand the dataset. They might be PDFs of data collection forms, third party metadata documents or even word/excel documents you have produced yourself.

Drag and drop a file onto the prescribing dataset.

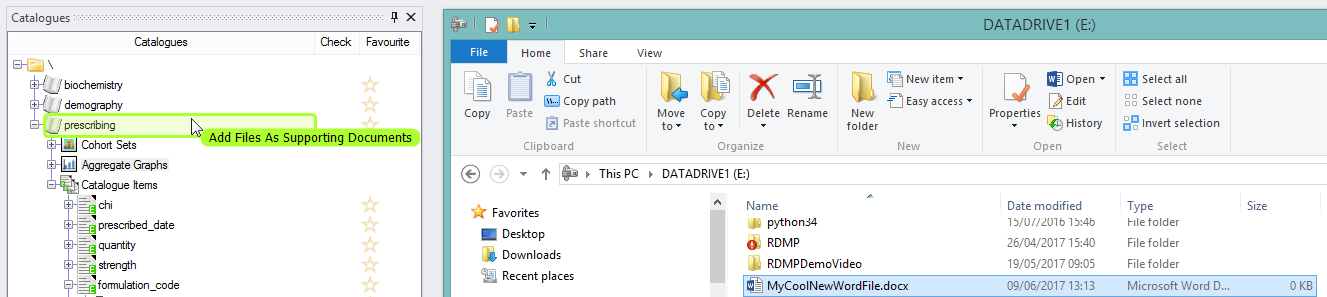


Figure 61 - Adding a Supporting Document via drag and drop

Now expand the Documentation node and double click the file you just added. Here you can tick Extractable flag (which will result in the file being copied into every research extract containing the prescribing dataset). You can also provide a short description of the file and what is in it.

Supporting documents are linked via the file path therefore it is important that the path be accessible to all data analysts using the software. Therefore it is a good idea to store supporting documents on a network drive.

If you need to move all your files (or a large number of them) to another piece of physical storage and you don’t want to break the RDMP links then you can use ‘Locations=>Adjust File Locations’ to do a find and replace across all physical file locations that RDMP knows about (useful if you want to say replace a network drive mapping of E:\ with J:\ etc).

## Supporting SQL

In addition to file attachments you can set up an SQL Query as an attachment. This is very dangerous as it circumvents the RDMP Query Building / Anonymisation / versioning proceedures in that it results in an absolute query that is executed and the results directly streamed into extracts (if marked extractable).

Ideally this should only be used for things such as global lookup tables etc where you do not want or cannot configure an RDMP column Lookup directly.

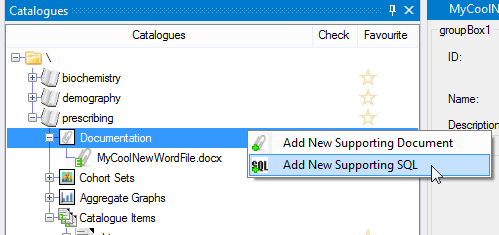


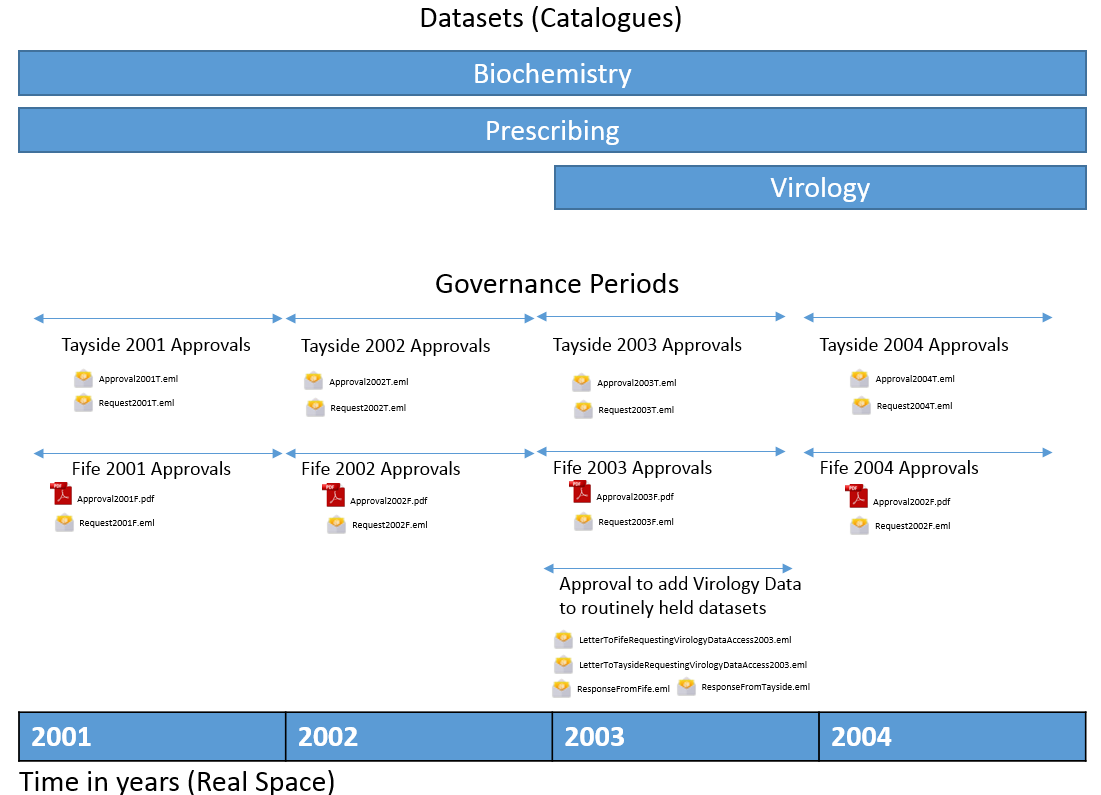
Figure 62 - Adding a verbatim SQL query as Supporting Documentation

# Functionality – Governance

## Introduction

The RDMP was designed to hold sensitive health datasets generated by third party data controllers in anonymous form for reasearch purposes. To this end it has a fully featured Governance system for tracking governance permission to hold datasets. This usually takes the form of a collection of documents (e.g. PDFs and Emails) betweeen your agency and your data providers which grants you permission to hold the datasets (possibly only for a limited periods). It may be that you don’t need permission to hold particular datasets or that some permissions are granted once and last forever (never expire) – RDMP supports both of these scenarios too.

Consider the following scenario. Your agency holds 2 datasets Biochemistry and Prescribing. These datasets are sent to your agency by two healthboards (Tayside and Fife) both of which require annual approval letters. Each year the RDMP alerts you to the fact that Governance has expired on your datasets, your governance manager sends an email to the relevant healthboards and recieves new approval letters. He then updates the system to have a new governance period for the new year and attaches the relevant documents to this new year.

Now consider that a new dataset becomes available (Virology in 2003). Rather than just appending this dataset into the approval request list, a separate email chain is sent to both Tayside and Fife for approval to add the dataset and to integrate it into the main approvals list. In this case there is a special one off governance period for 2003 which covers only Virology and reflects the permission to add this dataset to the repositry of data you hold. In subsequent years Virology is covered by the normal annual approvals.

## Configuring a Governance Period

## Setting up Governance

The RDMP is capable of storing lots of diverse information about your database tables including column descriptions, dataset descriptions, attachments, validation rules, governance rules etc. The novel thing about the RDMP though is it’s division of this metadata (both technical and descriptive) into a relational database structure. This allows you to keep a simple database model for your data and have RDMP drive transformation / extraction:

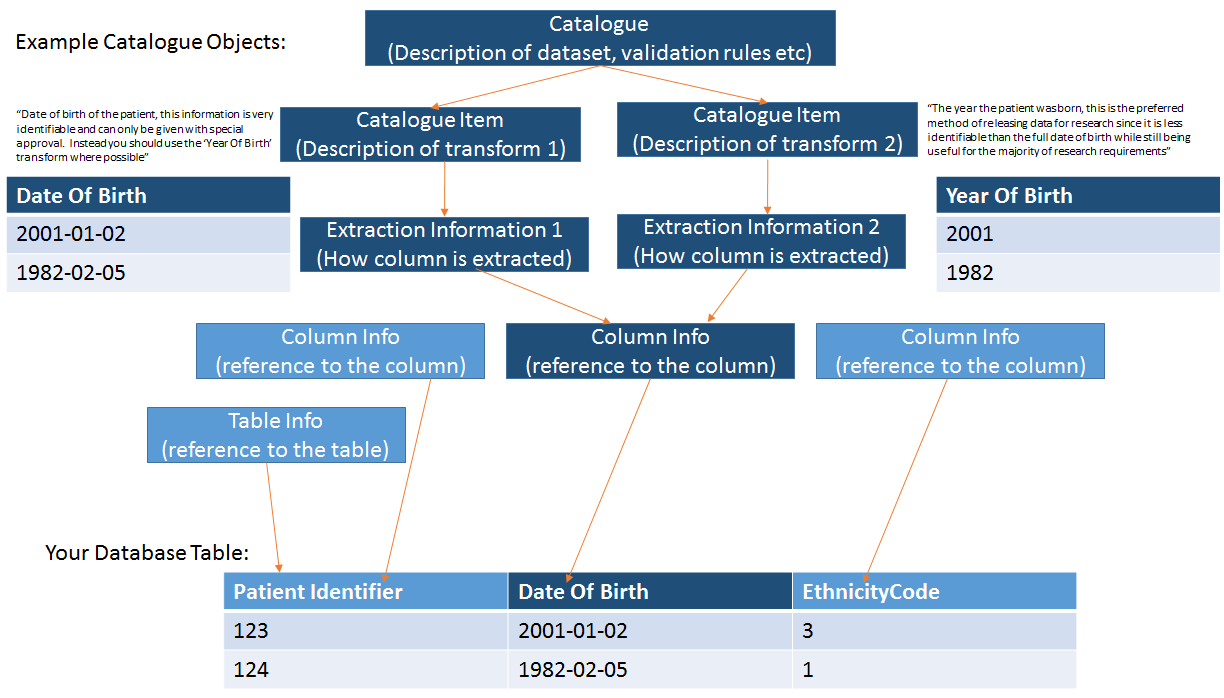


Figure 63 - Example data table and the Catalogue objects that model the table. Notice how there is 1 column in your repository (DateOfBirth) but 2 options for extraction each with their own descriptions.

A TableInfo contains a record of the server/database/table name of one of your tables, it also has an optional reference to some encrypted DataAccessCredentials (username/password) to connect to the data table (which can be configured to be available only under specific usage contexts e.g. DataExtraction vs DataLoading). It also serves as a reference point for tasks such as annonymisation of columns and configuring extraction join logic (Left / Right / Inner etc). Finally a TableInfo can be edited/rewired without affecting all the descriptive,transformation,filtering and validation logic you have built up in the Catalogue, this allows you to change your underlying implementation in your database without having to re-write all your extraction / validation logic.

The easiest way to create a TableInfo (and all the other objects shown above) is to import and forward engineer them from your raw data table. Go to the TableInfo tab of CatalogueManager.exe and enter the location of your data table. You can press F1 for help (see Figure 8). Note that if you specify a username and password it will be stored encrypted in the Data Catalogue, alternatively if you omit the username/password (or do not want to create one) then the RDMP will always use Integrated Security to access it with the currently logged on windows user account.

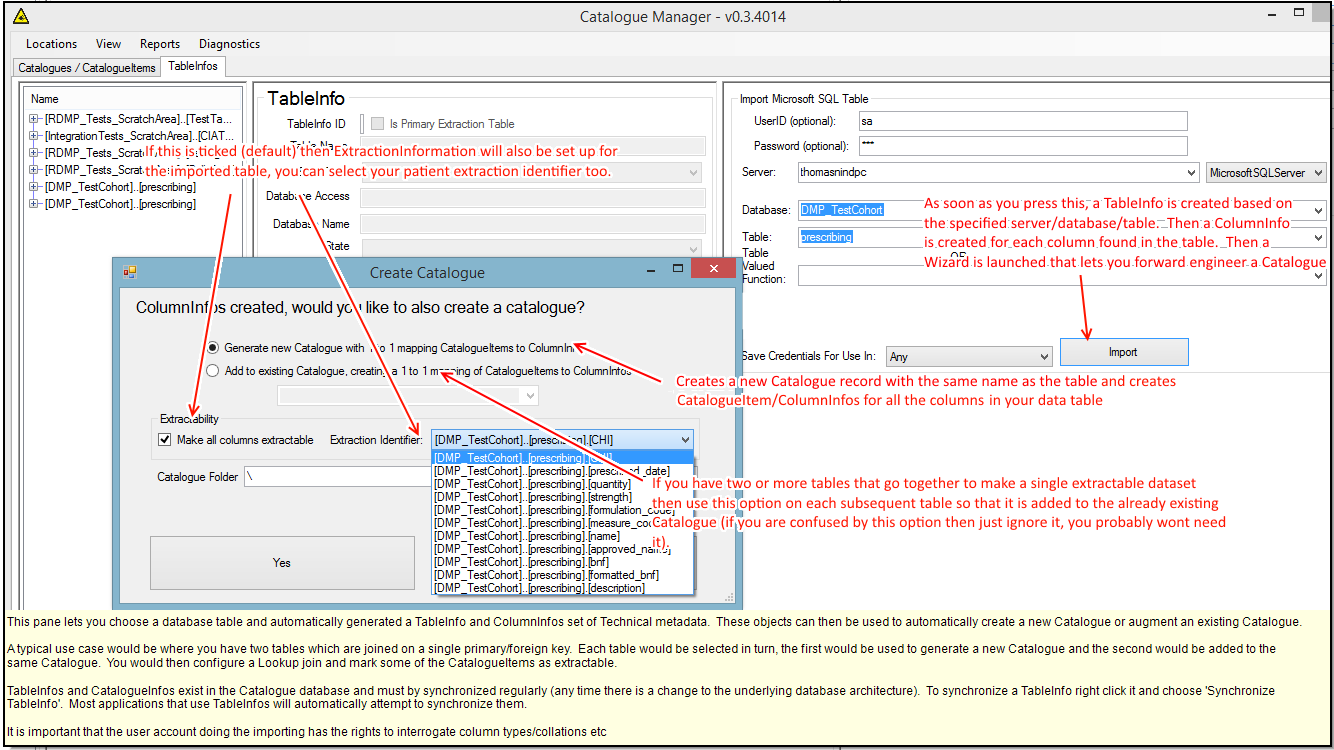


Figure 64 - Help text for TableInfo importing

Alternatively you can create a new Catalogue by importing a flat file (see Quick Start – Generating & Importing Test Data).

# Functionality – Audit

## Introduction

RDMP provides extensive auditing of activities. Audit is integral to many processes and many process include their own audit features (e.g. see Record Versioning). But there is a central logging database for all RDMP activities. This relational database is more easily navigable than a traditional flat text log file and allows for processes such as a error resolution audit.

## Logging Concepts

Logging is divided into the following concepts

|  |  |
| --- | --- |
| Name | Purpose |
| Task | High level category which describes the repeatable goal. For example ‘Data Extraction’ a task under which all activities relating to data extraction occur. By default there is a task for each load configuration (LoadMetadata) you create. |
| Run (DataLoadRun) | Each discrete execution of the task. This includes a start time, end time, description etc. If you load biochemistry monthly then you should have a single Task ‘Loading Biochemistry’ and a new Run audit every month |
| Progress Message | Noteworthy events that occur during a given Run e.g. ‘Archiving to c:\temp\LoadingBiochem\Data\ForArchiving\4049.zip’ |
| Fatal Errors | Messages that resulted in the failure of the Run. If there are fatal errors then the process did not complete succesfully. |
| Table Load (TableLoadInfo) | Each time records move from one place to another during a Run a Table Load audit is created. This includes records moving into a flat file etc as part of extraction. A Run can load many tables e.g. a Table Load could be created for populating RAW and another for populating STAGING and a final one for merging records with LIVE.  This includes counts of the number of inserts, updates and deletes that occurred. |
| Data Source | Each Table Load can have one or more data sources which are logs of what contributed rows to the table. For example if a load has a DelimitedDataFlowAttacher then the ‘Loading RAW’ Table Load log might have a list of Data Source files that matched the regex \*.csv in the forLoading directory. Data Sources can be SQL queries that were run on a database or other isoteric concepts too (they don’t have to be files). |

You can access the RDMP global log viewer through ‘View=>Log Viewer…’. This will let you search and sort through all activities data analysts have been using RDMP for (data load, data quality engine runs, data extraction etc).

You can also view logs for a specific load by right clicking the LoadMetadata and selecting ‘View Load Metadata Logs’. If you have errors you can double click them to see the full message (including stack trace if available).



Figure 65 - Viewing the logs for a given data load

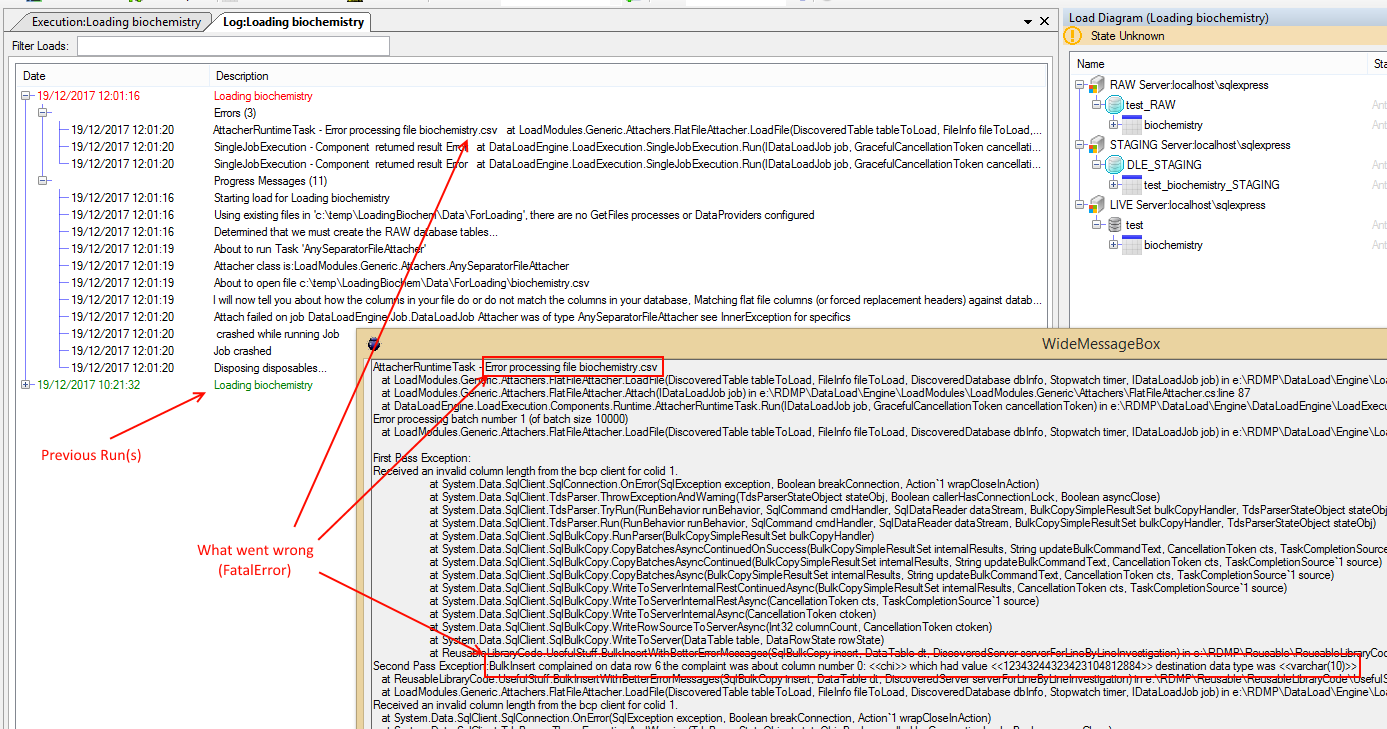


Figure 66 - Example failed load log

You can right click Errors and select ‘Resolve Fatal Error(s)’. This lets you provide a description of what you have done about the problem, if you had to undertake any manual steps to clean live data etc.

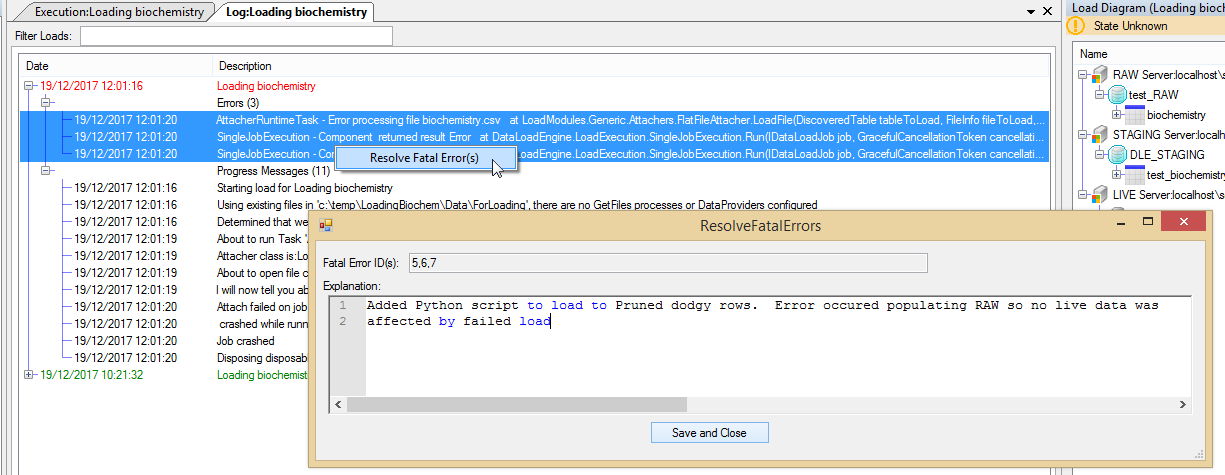


Figure 67 - Resolving fatal errors

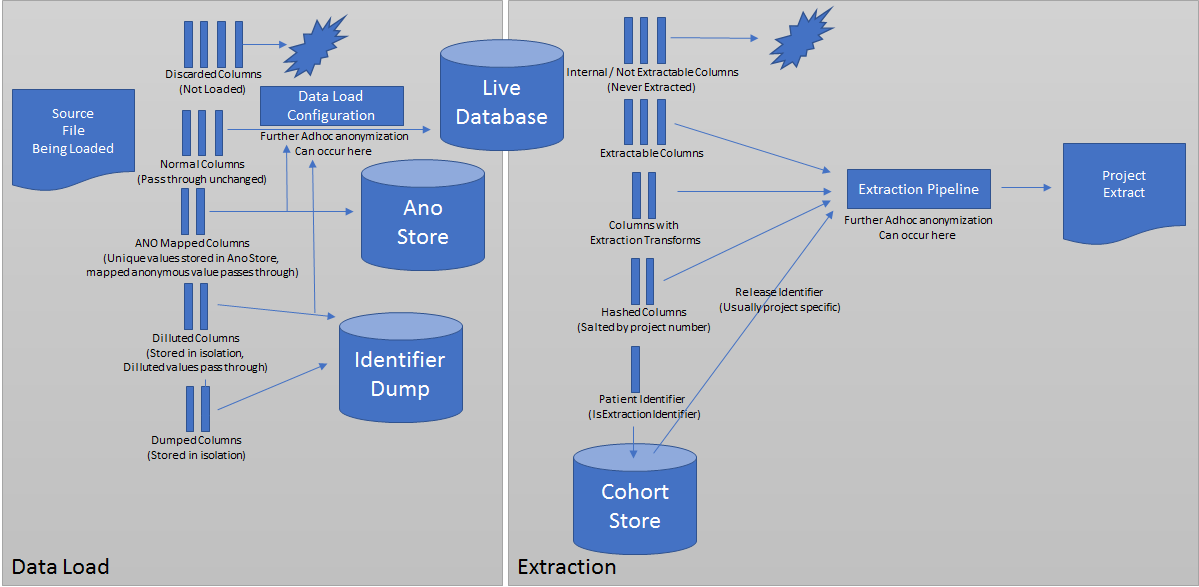
# Data Export Manager

# Anonymisation and Mapping Tables

## Introduction

Identifier mapping is when unique identifiers are substituted with new identifiers as part of an anonymisation strategy. This is usually done to prevent identifiers such as PatientID / LabNumber etc from being used to link with other data sources or for looking up a patient on a live clinical system. Identifier mapping is usually combined with other anonymisation operations such as dropping sensitive columns (e.g. patient name, surname etc) and/or dilluting data (e.g. rounding dates of birth to nearest quarter). Often anonymisation takes place in multiple stages with different rules/strictness e.g. phase 1 anonymisation occurs during loading your live database and phase 2 anonymisation occurs when generating a project extract for release.

RDMP supports anonymisation throughout the data lifecycle.



## Data Load Anonymisation

### Background

Data loading in RDMP takes place using the RAW=>STAGING=>LIVE model (See RAW Bubble, STAGING Bubble, LIVE Model). In this model the RAW database always contains the fully identifiable data (unless you have some custom attacher). Once in the RAW database you can implement any ad hoc anonymisation you want through SQL scripts / plugins components. Between RAW and STAGING columns will undergo load anonymisation (dumping, dropping, identifier substitution etc).

### Discarded Columns

Data is loaded into RAW primarily through Attachers (e.g. DelimitedFlatFileAttacher for csv). Most Attachers will reconcile columns in RAW with the file being loaded and may report a warning/error when expected columns are missing (this allows you to identify when a provided data source gains new columns). Therefore you can declare columns which will only appear in RAW and not your live dataset. This lets you have a live column ‘First Initial’ and declare a discarded column ‘Forename’ then in your AdjustRAW you can issue an UPDATE to set the ‘First Initial’ field based on the ‘Forename’.

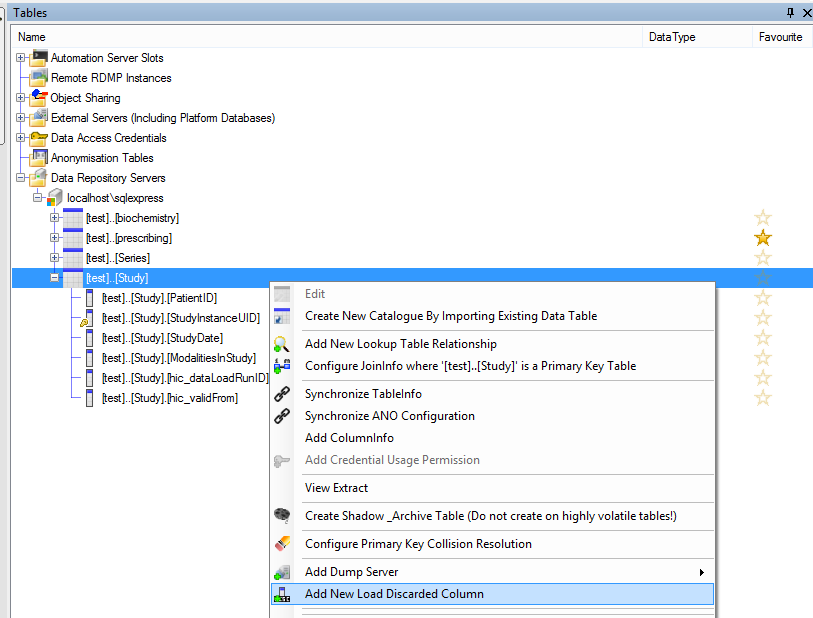


Figure 68 - Adding a new PreLoadDiscardedColumn

## Extraction Time Anonymisation

### Background

Once your live database is loaded with data you can tailor secondary anonymisation on a column by column database. This includes specifying which columns can be extracted, which require hashing, those include an extraction transform (e.g. LEFT 4 of column ‘postcode’). In addition columns (usually only 1 per dataset) can be marked IsExtractionIdentifier. When a cohort is saved to a Project all the patient identifiers are assigned a ‘ReleaseId’ (See Functionality – Release Identifier Allocation). The extraction itself occurs through an extraction pipeline which can contain further adhoc anonymisation / substitutions.

# Automation / Command Line

## Overview

RDMP supports command line execution of both data loads, data quality evaluations and other tasks. This allows you to ensure that your datasets quality is monitored on a regular basis and that routine loads do not get forgotten or take up data analyst time through manual execution etc.

The actual automation executable is called RDMPAutomationService.exe and can be found in install directory of RDMP.

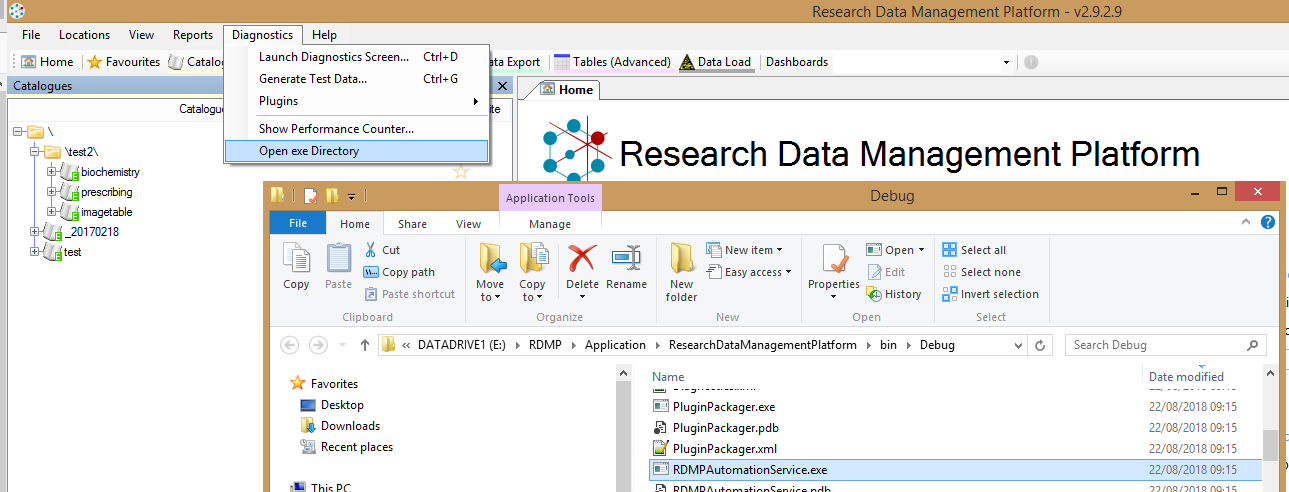


Figure 69 - Locating the RDMPAutomationService executable

You can run the process from the command line directly, set up a Windows Scheduled Task, bat file or use your own automation system.

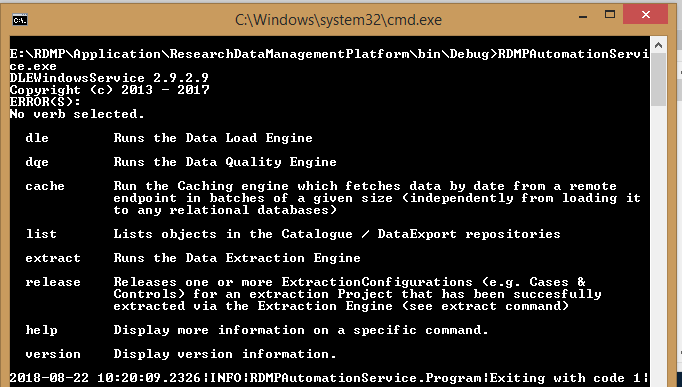


Figure 70 - Running RDMP from the command line

## Command Generation

Since many of the commands are quite complex and involve knowing the IDs of the objects you want to run (e.g. the ID of the load you want to run) RDMP can generate the command for you from the user interface.

For example if you launch the DQE execution screen for a given Catalogue (see Figure 69) then you can get the command line instruction copied directly to your clipboard. After running checks click ‘Copy’ and paste the results into notepad.



Figure 71 - DQE Execution Tab for Catalogue Prescribing

In the above example the resulting command would be:

|  |
| --- |
| RDMPAutomationService.exe dqe --Catalogue 2 --cataloguedatabasename RDMP\_Catalogue --dataexportdatabasename RDMP\_DataExport --servername localhost\sqlexpress run |

# Data Access Credentials / Password Encryption

The RDMP takes authentication very seriously and considerable effort has been put into ensuring malicious users cannot use information stored in the Data Catalogue database to access resources to which they do not have a mandate for.

The RDMP software requires access to secure databases (e.g. for loading data) with highly sensitive information on them. This access often has to happen in an automated manner without user intervention (e.g. a data load service periodically loading your repository at 2am). SQL Server (and most other database providers) allow two ways to authenticate:

* Integrated / Windows authentication - Where a user/services windows logon is granted access rights and Windows handles the communication of authentication tokens.
* Username / Password based authentication – Where a textual username and password combination is sent to the database server.

Both methods have advantages and limitations and so the RDMP supports both (See Figure 17). Storing free text usernames/passwords in the Data Catalogue would be disastrous as it would lead to a single point of failure for all data assets (repositories). A typical solution for this problem is password hashing but this is not possible in our case since usernames/passwords must be sent to the remote data repository at execution time (not hashes). The RDMP solves this problem by using 4096-bit RSA public/private key encryption to encrypt passwords that are stored in the Data Catalogue. The location only of the private key is stored in the Data Catalogue. This key should be on a shared network drive with tightly controlled (Windows) access rights configured on it. When the RDMP software needs to connect to a server with a username/password (Option 2) it will download the encrypted password from the Catalogue database and then attempt to open the private key file (which will fail if the executing Windows user account does not have access to the key store location).

If there is no username/encrypted password in the Catalogue Database then the software will attempt to access the Data Repository using Integrated Security (Option 1).

This approach ensures that both options rely on Windows user account management as the gateway to authentication (either adding user accounts directly to the database or by controlling which users have access to the decryption key file location).

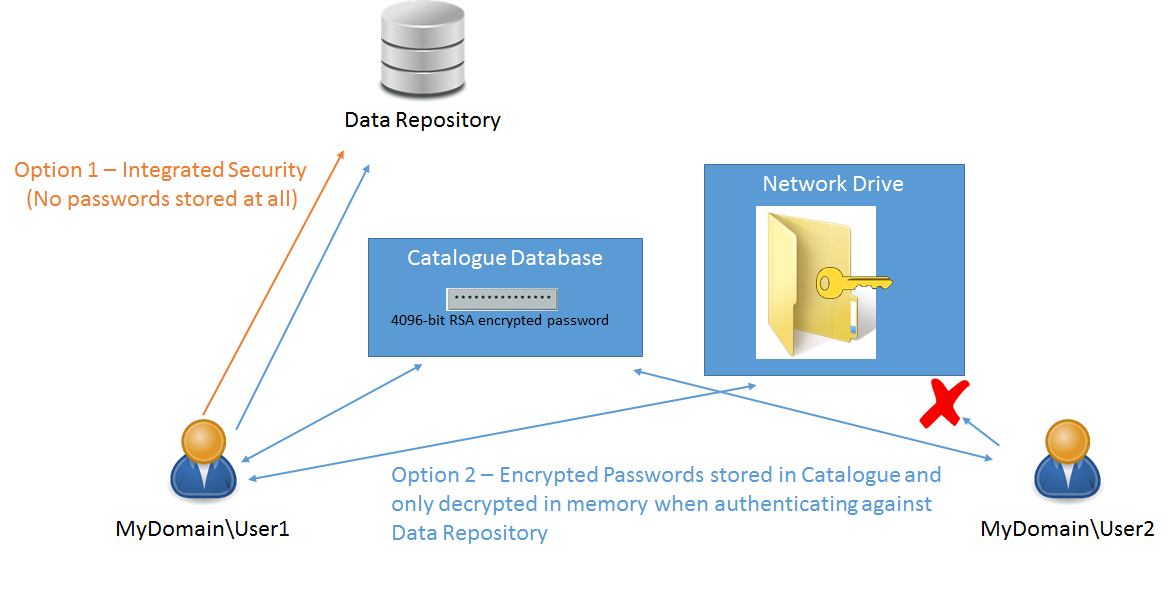


Figure 72 - Authentication model used by RDMP which uses Windows user accounts either directly through Integrated Security or indirectly through access to the private decryption key file.

# Pipeline Components

## Background

# Software Development

## RDMP Design Philosophy

The RDMP was written with a design philosophy informed by the highly sensitive nature of it’s operating environment. The most catastrophic risk events that can occur in the provision of research datasets are (in order of severity):

1. Accidentally releasing of identifiable data to researchers
2. Corrupting datasets with duplication or data error
3. Datasets becoming unusable due to knoweldge loss (e.g. only data analyst who understand a dataset leaving)
4. Inconsistent data provision (e.g. being unable to provide a researcher a consistent data extract that matches the first extract he got at the start of his 5 year research project).

All design decisions and features of the RMDP are intended to limit the risk of each of these occurring. This began right at the begning of the project in the selection of technology for storing RDMP data (Platform databases – Microsoft SQL) and should always be borne in mind when developing new features.

Core design descisions that have been made in the RMDP development include:

### Checkability

RDMP provides comprehensive support for checking planned operations ahead of executing them (Reduces R1 and R2). It lets you define runtime rules for validating whether your class is in a fit state to run ahead of execution. This is a similar to unit testing or code contracts (which the RDMP also makes use of) but occurs at ‘check time’ when the software is running. It lets the data analyst know when he has misconfigured or not met the requirements of a piece of functionality. For example the CatalogueConstraintReport check functionality checks that there is a DataQualityEngine results database ready to receive the final results of the report, that the dataset has syntactically valid validation rules and that the data repository it is pointed at is accessible under data access context InternalDataProcessing.

|  |
| --- |
| To support checking simply inherit your object from ICheckable. To support it in user interfaces you can add ChecksUI, PopupChecksUI or ChecksUIIconOnly.  All messages (successes,warnings and failures) in your Check method will go via ICheckNotifier which requires an argument NotifyEventArgs, the constructor of this transparently generates a stack trace of where the message originated. This combined with a build action which embedds the entire RDMP source code into each build executable allows the user to visualise immediately in the codebase why given messages are appearing without having to have visual studio or other traditional debugging tools (See Data Load Engine Error Messages).  Implementation |

### Primary Keys And Load Bubbles

The RDMP includes it’s own dedicated Data Load Engine (See Dataset Loader). Several design decisions have been made to reduce R2 and R3. All datasets must have a uniquely identifying primary key which comes from the source data (i.e. it cannot be a database autonum generated locally). Primary keys can be composite and should always make logical sense (e.g. a biochemistry result record primary key should be the patient identifier and date of sample collection or a collection ID if it is generated at the source of data collection).

Secondly there exists a design methodology of always being able to backout of operations through the use of transactions and backups. This extends throughout the entire codebase from the backing up of platform databases (e.g. the Catalogue database) before patching it during software updates to the loading of new datasets through ‘super transaction’ load bubbles (See RAW Bubble, STAGING Bubble, LIVE Model).

The RDMP must always try to provide as much information about errors as possible and where practical leave runtime artifacts (such as load bubbles) available for inspection by the data analyst performing the operation. This is vital for long term management of datasets where the original dataset load configuration creator may no longer work at your institution when his load (which might have been working fine for years) suddenly stops working (Reduces R3). Data load logic also follows the mono-atomic design philosophy (see Mono-Atomic Record Storage in Platform Databases) which helps reduce R3 by ensuring that rather than one massive poorly documented load script (or bespoke software), there is a clearly divided series of plugin tasks and documented specific scripted steps (Adjust RAW etc).

This design implementation acknowledges that data loading / management of complex and volatile medical research datasets over long periods of time (R4) is a fundamentally different process from traditional static/managed ETL processes for which tools such as SSIS, Talend etc already exist.

|  |
| --- |
| See Chapter: Dataset Loader  Implementation |

### Dynamic SQL Generation

The RDMP is designed to be used by a team of data analysts with varying levels of SQL knowledge. It is anticipated that each data analyst will be an expert in specific datasets but have a poor understanding of others. The RDMP was designed to allow expert knoweldge to be entered in the form of mono-atomic SQL blocks and documented by the creator such that even users with no knowledge of SQL can use them by simply dragging and dropping appopriate blocks to the task they are attempting to execute (e.g. generating an aggregate or performing a data extraction). The idea is that as long as each separate component is properly documented and thouroughly tested the combination of these components into increasingly complex queries can be handled without concern by the software (Reduces R1 and R4). An added advantage to this design philosophy is that when a data anlayst with expert knowledge leaves he leaves the documented artifacts required for data set filtering/extraction/loading etc.

|  |
| --- |
| For example a piece of expert knowledge might be ‘how to filter hospital admissions by ICD code list’. This would be stored as a WHERE SQL block ‘LEFT(AdmissionCode,4) in (@list)’ attached to the extractable column AdmissionCode which is modeled as a SELECT SQL ‘LEFT(AdmissionCode,4)’ with governance of ‘Core’, column order 10 and a SQL Parameter of @list. Each block has full documentation of appropriate values, pecularities (e.g. why is only Left 4 characters considered?!) and advice on usage.  In the user interface a data analyst can select the columns they want (including transformations) and combine filters in AND/OR containers. The system will generate the appropriate SQL as readonly text which it will use at extraction time. This entire configuration will itself be stored in the DataExportManager platform database with references to the original blocks so that the system knows when implementations have changed since the configuration was last used.  Implementation |

### Auditing

Another core philosophy of the development of the RDMP is that all operations which result in the movement of data from one place to another (including extraction to flat files) be fully audited in a central location (The relational platform database). This includes the user performing the operation, any SQL used/generated by the system which fetched data, the number of rows writen to each data destination and MD5 checksums of any flat files generated. A relational database (Managed by Logging.Database.dll) is provided for this functionality. This provides a more structured and navigable representation of activities going on within the data repository than would be possible with console output logging (Reduces R3) .

Furthermore where possible metadata configurations should be frozen rather than deleted (e.g. an ExtractionConfiguration in data export manager is frozen after it has been executed and the resultant files released). This allows configurations to be cloned and executed later for reproducability or to provide a refresh/update of a given extraction (Reduces R4).

### Plugins and Reusable Pipelines

The RDMP began as an in house tool to support HIC Services but it rapidly became apparent that the software could be used more widely by other agencies that performed a similar research data repository management role (and extraction to anonymous safehaven). A decision was made early on to restrict all agency specific implementational logic to ‘plugins’.

This prevented the codebase from becoming unmaintanable with cryptic HIC specific data management patterns while still allowing for agencies to develop their own functionality to fit their unique data management requirements (Reduces R2 and R3). An example of such logic would be HIC’s handling of CHI (Community Health Index) identifiers, these are 10 digit numbers in which the first 6 digits are the patients date of birth. Often data is provided to HIC in Excel format which can strip leading zeros from these identifiers. A plugin exists which looks for columns called CHI and appends 0 to the front of 9 digit values where this would result in a valid CHI. The plugin can be used in any use case where data flows from one place to another (E.g. data loading, data extraction etc). It would not make any sense for this plugin to be part of the core RMDP codebase.

|  |
| --- |
| Plugins can be used at any point in a DataFlowPipeline (see A brief overview of what a pipeline is). A plugin can be as simple as enforcing a table naming convention to performing identifier substitution against a remote web API.    Plugins must be written in C# and contain a reference to the nuget RDMP reditributable. MEF is used to expose your plugin classes from your dll, to achieve this all you need to do is inherit from the appropriate interface (e.g. IPluginDataFlowComponent<DataTable>). Your compiled plugin dll is uploaded directly into the DataCatalogue database via CatalogueManager with optionally a pdb file for debugging at runtime.  Once you have uploaded your plugin to your local Catalogue database, you can use MEF Checker to confirm that your classes appear and can be constructed succesfully. Theceforth you can drag the plugin into any Pipeline you are creating. Plugins can specify contextual requirements e.g. implement IPipelineRequirement<CohortCreationRequest> if you only want your plugin to be used for data extraction (lets imagine you want to send an email to someone every time a data extraction is performed saying who the cohort is for, the date of execution etc).  Implementation |

# Licenses

RDMP makes use of many libraries, the license terms of these libraries are included below

## GPLv2 Licensed Libraries

The following libraries are used under license of the GPL V2:

MySql.Data,ObjectListView,SharpZipLib

## Apache Licensed Libraries

The following libraries are used under the Apache license

Common.Logging, CsvHelper, GraphX, RestSharp, Roundhouse

## MIT Licensed Libraries

The following libraries are used under the MIT license

Command Line Parser, NancyFX

## BSD Licensed Libraries

The following libraries are used under the BSD license

RhinoMocks, SSH.NET

## FAM FAM FAM Icons

Some icons from FAM FAM FAM are used under the Creative Commons Attribution 2.5 License. [ http://creativecommons.org/licenses/by/2.5/ ]

FAM FAM FAM Icons are hosted at

http://www.famfamfam.com/lab/icons/silk/

## Libraries with custom licenses

The following libraries are used with their own unique custom licenses which are compatible with the GPL.

### Scintilla.Net

(<http://www.scintilla.org/>)

License for Scintilla and SciTE

Copyright 1998-2002 by Neil Hodgson <neilh@scintilla.org>

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its

documentation for any purpose and without fee is hereby granted,

provided that the above copyright notice appear in all copies and that

both that copyright notice and this permission notice appear in

supporting documentation.

NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS

SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY

AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY

SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES

WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,

WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER

TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE

OR PERFORMANCE OF THIS SOFTWARE.

### Nunit

(<http://nunit.org/nuget/license.html> )

Portions Copyright © 2002-2012 Charlie Poole or Copyright © 2002-2004 James W. Newkirk, Michael C. Two, Alexei A. Vorontsov or Copyright © 2000-2002 Philip A. Craig

### Oracle ODP.NET Managed Driver

<http://www.oracle.com/technetwork/licenses/distribution-license-152002.html>

## Public Domain Images



<https://www.wpclipart.com/smiley/glossy_smiley/glossy_green_smileys/glossy_smiley_green_wink.png.html>



<http://www.freestockphotos.biz/stockphoto/15766>



<https://pixabay.com/en/smiley-angry-furious-unhappy-face-155846/>



<https://pixabay.com/en/database-storage-data-storage-152091/>

# Quick Start – Descriptive Metadata

This guide will show how to import an existing table from your database, document the dataset/columns and produce a report containing the descriptive metadata.

Ensure you have followed the steps in section ‘Quick Start - Test Environment Setup’. Ensure that you do not have a Catalogue called TestHospitalAdmissions visible in the main screen of Catalogue Manager.

## Synchronizing descriptions (Optional)

Make a change to the description of the CHI column and save it (either by pressing Ctrl+S or by selecting a different CatalogueItem). This will launch the Propagate Changes wizard which allows you to ensure that there is only one definition for CHI. In this case it is quite a trivial example but if you have 200+ datasets with many overlapping columns, ensuring a single description can be very useful.



Figure 74- The Propagate Changes Wizard

Choose to copy over the changes to DMPTestCatalogue CHI column (tick all boxes). Select Catalogue DMPTestCatalogue and the CatalogueItem CHI, notice how it’s description now matches the new description.

## Fill in descriptions for the remainder of the CatalogueItems

Type in descriptions for each of the remaining CatalogueItems in TestHospitalAdmissions. Then mark them all as Extractable by selecting Extraction->Catalogue Items -> ‘Mark All Associated Columns Extractable’. This will ensure that other parts of the RDMP software suite know that the dataset can be extracted from.

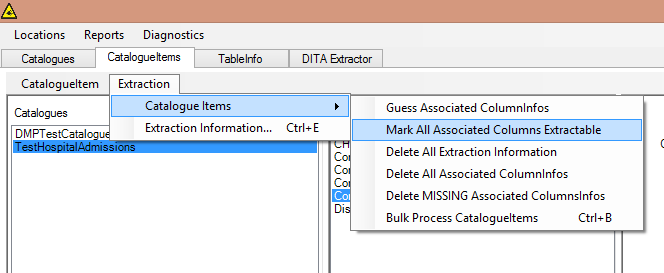


Figure 75 - Mark all the CatalogueItems extractable

Double click TestHospitalAdmissions to see the systems understanding of how the data is extracted.

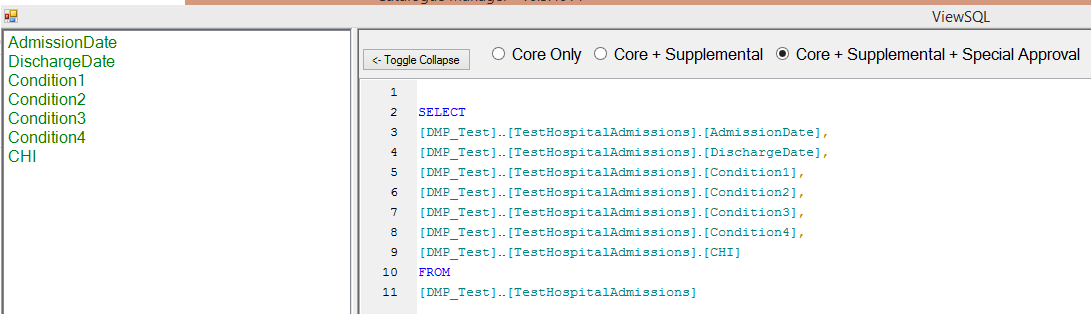
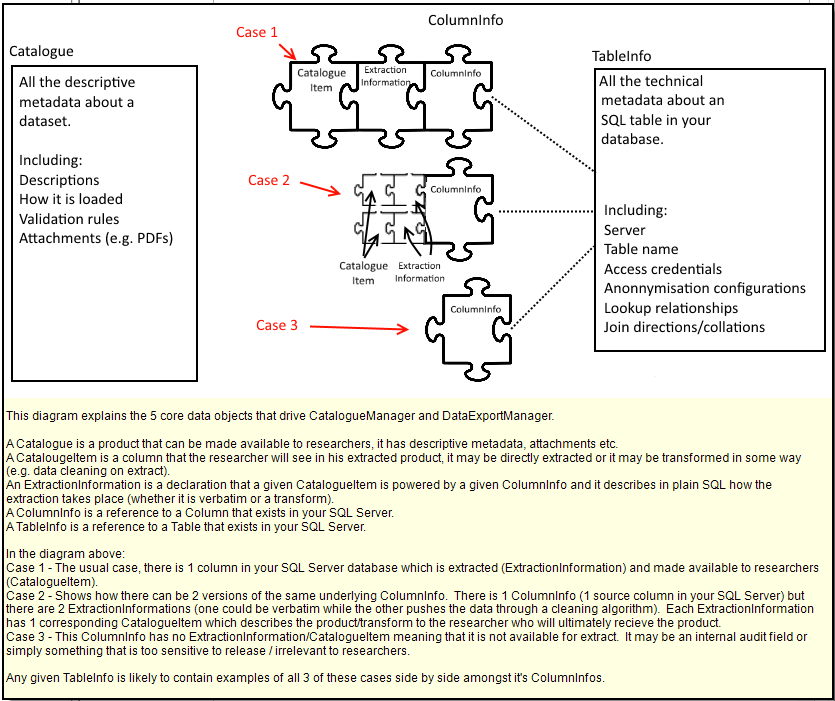


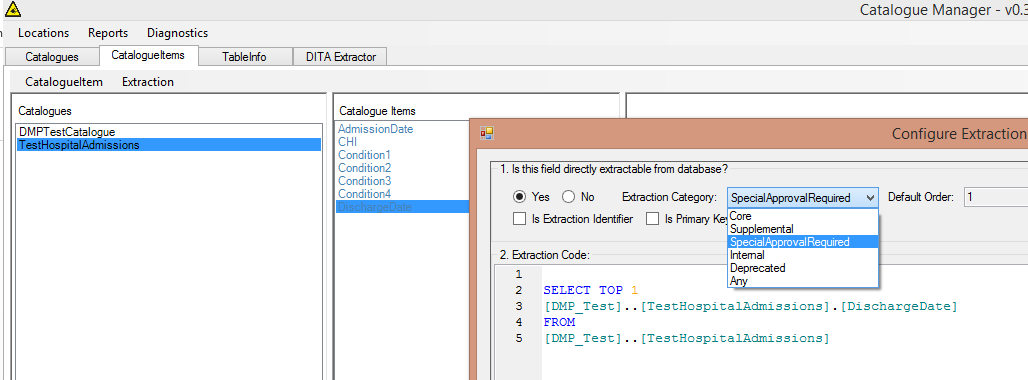
Figure 76- Catalogue Managers understanding of how the dataset is extracted

## Understanding Catalouge Manager data model

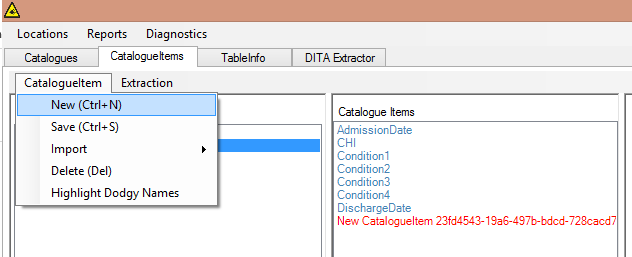


## Marking Discharge Date as ‘Special Approval’

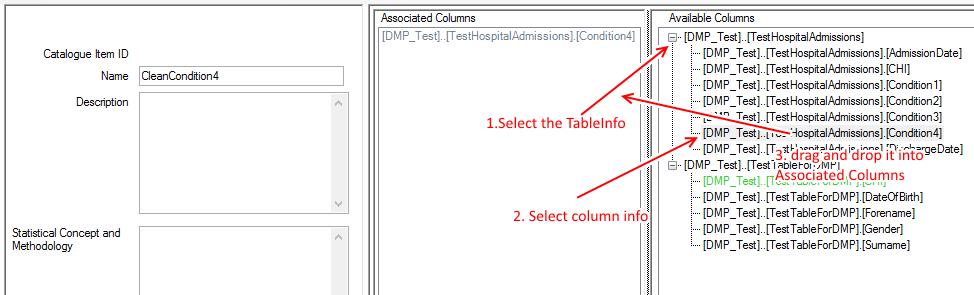
Return to Catalogue Manager and select the CatalogueItem tab. Select TestHospitalAdmissions Catalogue and double click DischargeDate. This should launch the ‘Configure Extraction Information’ dialog. Select Extraction Category and change it to ‘SpecialApprovalRequired’



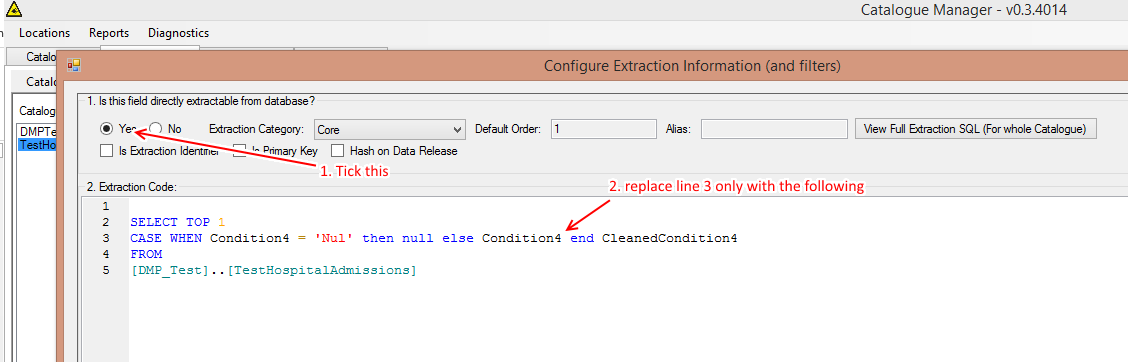
Next create a new CatalogueItem (for our cleaned version of condition 4)



Select New CatalogueItem and change it’s name to CleanCondition4. Now comes the difficult bit, we need to associate our new ‘virtual’ column with the same underlying column (Condition4). To do this expand and locate Condition4 in the ‘Available Columns’.



Then double click the grey Condition 4 Associated Column and paste in the transform line:



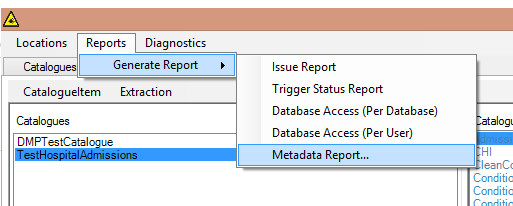
Open the old Condition4 column and mark it as ‘Supplemental’.

Double click TestHospitalAdmissions and look at the new representation of the dataset. Try switching between the radio buttons to see the various views of the data available.

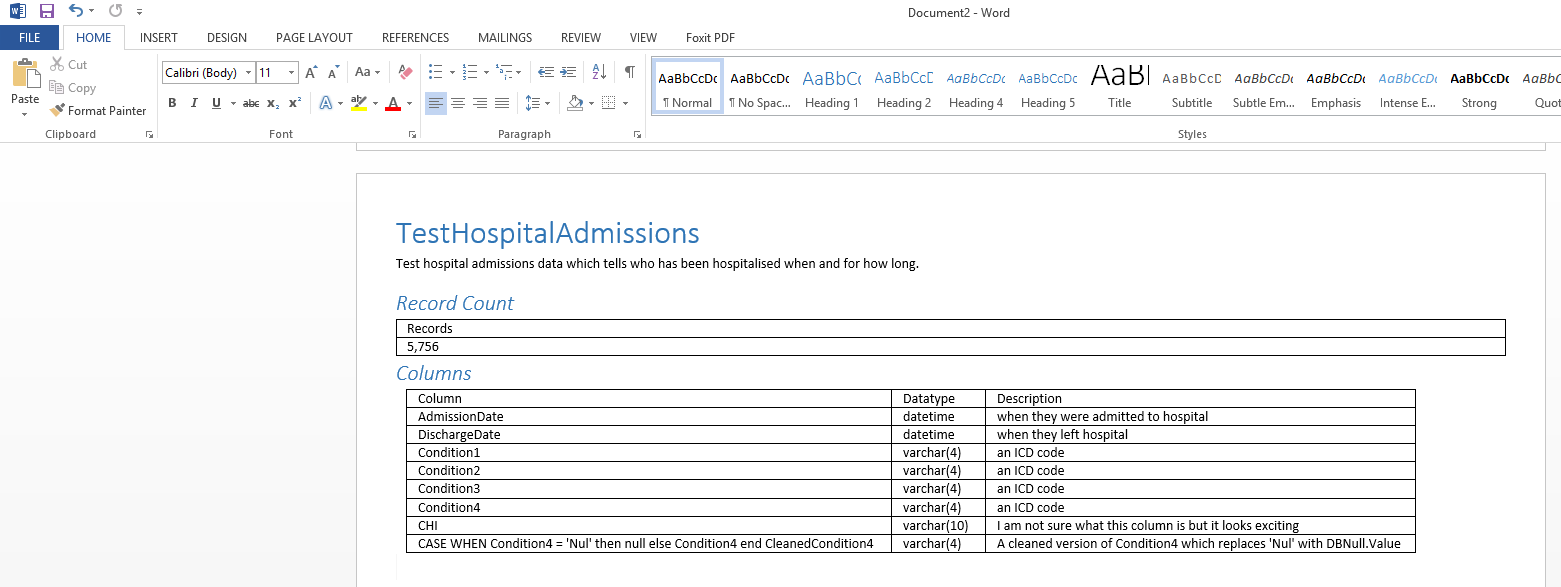


## Generate metadata report

Put in an appropriate description for CatalogueItem CleanedCondition4 and then select ‘Generate Report’ -> Metadata Report…



Assuming Microsoft Word is installed you should see a nice summary of the datasets you hold.



# Technical

## A brief overview of what a pipeline is

### Pipelines

Pipelines are how RDMP manages batch/mulit stage operations in which files or data moves from one place to another. They consist of a source component generating data (e.g. reading a file) and a series of components operating on the data then a destination component (e.g. writing final data into a database).

Each Pipeline is configured once and then reused for the task any time it must be undertaken. All pipelines are shown in the main RDMP user interface (See Figure 80).

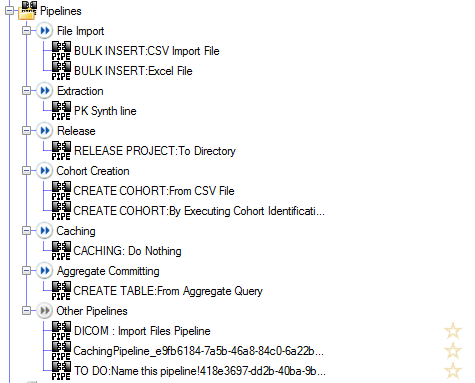
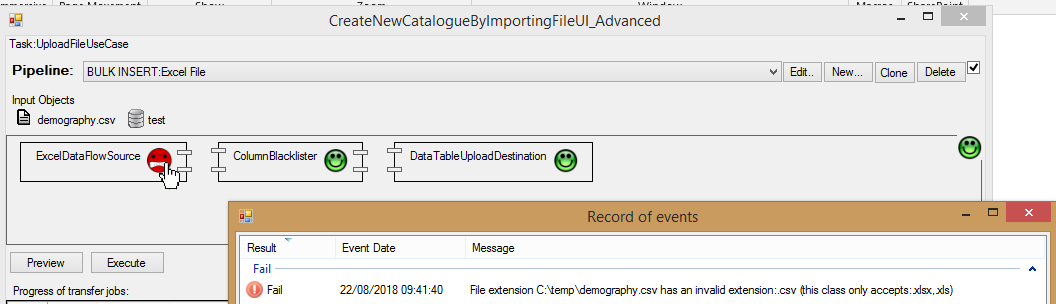


Figure 79 - Pipelines in RDMP

### Using a Pipeline

When RDMP is asked to perform an activity (e.g. import a CSV file) that requires the use of a Pipeline it will ask you to select one. Only Pipelines that are compatible with the task will be shown. Compatibility is decided based on the task and selected objects (See Pipeline Use Case).

The image below shows the (advanced) user interface for importing a file. The input objects are shown (a file called demography.csv and the target database the user has picked). When you select a Pipeline each component is shown the input objects and given a chance to object. In this case the source (ExcelDataFlowSource) compains that it is incompatible with the file demography.csv because it is not a Microsoft Excel File. This problem can be corrected by selecting a different pipeline (e.g. ‘BULK INSERT: CSV’ instead).



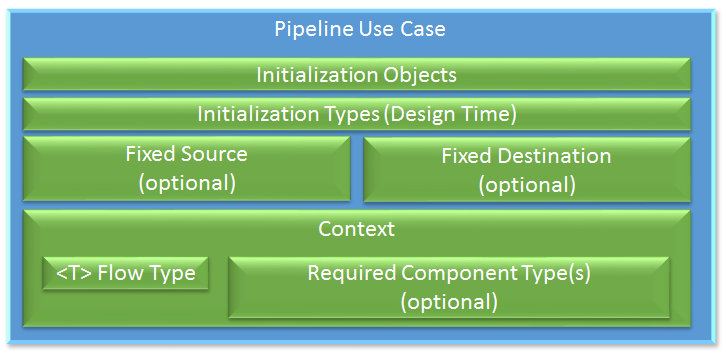
As you become more profficient in the use of RDMP you can customise and produce multiple Pipelines for a given task. For example you could create:

|  |  |
| --- | --- |
| Example Pipeline | Purpose |
| BULK INSERT: CSV Identifiable | Load CSV files and crash if any columns contain identifiable data |
| BULK INSERT: CSV Anonymous | Load CSV files but anonymise the data mid pipeline |
| BULK INSERT: CSV Dropping null PatientIds | Load only rows in which the PatientId column has data |

### Pipeline Use Case

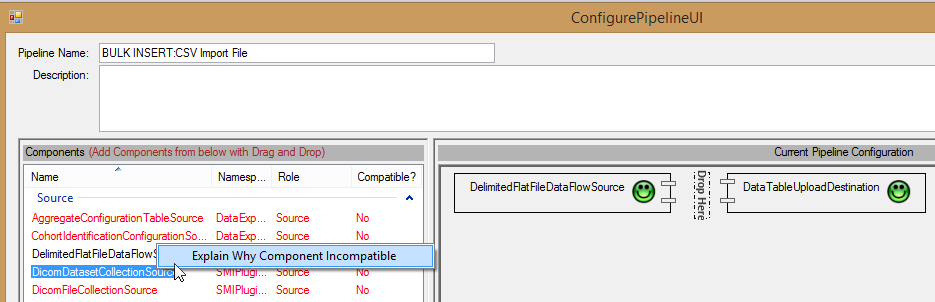
Pipeline compatibility and checking is dependant on the activity being undertaken (an extraction pipeline can’t be used to load data!). In code this is represented by the PipelineUseCase class.

The PipelineUseCase describes what components are allowed, what objects are being operated on and what the source/destination must be (e.g. if your uploading a file the destination must be (or be derrived from) DataTableUploadDestination.

****

* Initialization objects
  + Any Pipeline containing components with IpipelineRequirement<X> are only compatible if X is one of the use case initialization objects (e.g. a FlatFileToLoad)
  + Are passed to pipeline components at check time so they can decide if they are compatible or not during checking
* Initialization Types (Design Time)
  + If the user just wants to change a pipeline (e.g. file import) then he doesn’t want to have to pick initialization objects (e.g. select a file and a database). Therefore the use case also knows what Types are available
* Fixed Source/Destination
  + Some use cases do not let the user pick or configure the source and/or destination instead use an explicit fixed instance at runtime.
* Flow Type
  + The C# Type that is operated on by each component e.g. a DataTable or a ReleaseAudit (in the case of ReleaseUseCase)
* Required Component Types
  + Some use cases require the user to pick a particular kind of source/destination e.g. UploadFileUseCase requires a destination component of (or inherited from) DataTableUploadDestination
  + Use cases can also list components that are not allowed
  + Remember that any component requiring an initialization object that isn’t available will automatically be incompatible (See IpipelineRequirement)

When building a Pipeline, if you get confused as to why a component isn’t compatible then you can right click it. In the example below we are editing the CSV import pipeline and want to know why the plugin component DicomDatasetCollectionSource isn’t compatible (because its for loading dicom files not CSV!).



The message is int his case is:

|  |
| --- |
| Type **DicomDatasetCollectionSource is not a IPipelineRequirement<FlatFileToLoad> (which is the required type for all sources** within the current context) and the following types are required by the component but **not available as input objects to the pipeline SMIPlugin.PipelineComponents.DicomSources.Worklists.IDicomWorklist** |

# Glossary

ETL – Extract, Transform, Load

Cohort – A group of people often selected for common traits e.g. ‘People in Tayside over 30 with diabetes’.

Linkage – The process of reducing a large dataset by restricting data extracted to only those matching a given cohort.