Designing an Extract Transform Load (ETL) Tool for Environmental Health Studies, v 1.0

May 23, 2016

Kevin Garwood and Peter Hambly

Small Area Health Statistics Unit

Imperial College

1 Introduction

In the broad areas of Big Data and Data Science, there is a growing interest in reusing and combining large data sets drawn from different sources to provide new insights in the fields of healthcare research. The Small Area Health Statistics Unit at Imperial College, set up in 1987 with a remit to investigate potential environmental exposures affecting health, has developed and honed this interest over 29 years. It uses routinely collected health, environmental, population, geographical and lifestyle data to support research and help inform government policy.

SAHUS's remit is unusual in that it is driven by both research *and* service imperatives, in terms of policy and public health support. It is this second characteristic that means it has had to make significant investments in the general field of health informatics as well as epidemiology, statistical methodology and exposure assessment. Fortunately, for Computer Scientists who want to know more about how this field works, SAHSU is also encouraged to share its informatics expertise when it is appropriate to do so.

We present here a discussion about how we have been developing a bespoke Extract Transform Load (ETL) tool to load geospatial and health data into a database that is used to help scientists perform environmental health studies. Although we describe what the scientist tools do, the emphasis of discussion here is on the sequence of design decisions we've made to support the ETL. Its target audience are data scientists and software engineers, and we assume both will have a limited knowledge of the scientific domain the ETL supports.

This document is intended to show an engineering narrative that formalises a kind of experience report. It is not intended to be prescriptive in tone. Instead, it is meant to share with the data science community a sequence of design decisions we made in response to the needs of the use case and project constraints. The aim of discussion is to contribute to the wider body of knowledge about how to create tools to support activities in the field of environmental health.

Kevin covers most of the discussion related to ETL features for health data, whereas Peter has provides insights about the processing geospatial data for the RIF. If anyone has questions, please get in contact with us through our GitHub site at: https://github.com/smallAreaHealthStatisticsUnit/rapidInquiryFacility.

If you'd like to follow along more closely, keep watch of the `./rifDataLoaderTool` and `./rifNodeServices` parts of the code repository. Please note that we are still implementing minor changes for some of the design decisions, in particular (e.g.: ETL-42, ETL-43, and ETL-64).

The Authors

Kevin Garwood

Kevin is a data scientist and software developer who has spent over a decade creating scientific software to support fields of immunology, bioinformatics, and epidemiology. His past experience has been in the use of model-driven software to generate scientific software, the management of metadata and making software that can foster repeatable scientific protocols.

Code Links:

- https://github.com/kgarwood
- https://sourceforge.net/u/kgarwood/profile/

Contact: kevin_garwood@hotmail.co.uk

Peter Hambly

Peter is a developer operations engineer who spent over a decade managing SAHSU's health data and its information governance policies. Originally trained as a software engineer in BT's research division, he served as a database administrator in the airfreight industry before becoming the main Oracle and PostgreSQL database administrator for the Small Area Health Statistics Unit. Much of his past experience has related to: data loading and cleaning, performance tuning large-scale databases, preferably by design; anonymisation and pseudonymisation of sensitive data; and practically implementing information governance policies set by providers of health data.

Code Links:

- https://github.com/peterhambly

Contact: phambly@fastmail.co.uk

Project Links

- **Coding Project Web Site**: https://github.com/smallAreaHealthStatisticsUnit/rapidInquiryFacility/wiki/What-is-the-RIF
- **Imperial College RIF Web Site**: http://www.sahsu.org/content/rapid-inquiry-facility

2 The Role of the ETL in the Scientific Use Cases

The Rapid Inquiry Facility is a software suite that supports disease mapping and risk analysis activities in the field of environmental health. It uses geospatial and routinely collected health data to visualise health outcomes on maps.

Disease mapping can visualise mortality and morbidity rates and risks across an area. For example, a disease map may be able to answer a specific question such as:

"How many males in the district of Birmingham, between the ages of 20-24, were admitted to hospitals between 2000 and 2004, where there was a diagnosis was asthma?"

The maps can answer broader questions such as:

"How do the rates of asthma outcomes vary from one geographic area to another?"

Risk analysis can be used to explore whether a source of some particular exposure is having an impact on health in the local population. For example, a risk analysis could answer a question such as:

"Is there a statistically significant relationship between the distance people live with respect to a smoke stack and their likelihood of developing a specific type of cancer?"

From these questions, we can work backwards to determine what kinds of data the ETL will need to process. The RIF needs to match health data with population data at varying geographical resolutions and for specific demographics. For example, to answer the first question, it would have to compare the number of males aged 20-24 who were diagnosed with an asthma-related health outcome with the total number of males aged 20-24 who lived in the same area. That area could have varying resolutions of detail such as super-output area, output area, ward, district, region, or nation.

The health data are managed in **numerator data** tables and will include routinely collected hospital data that describe episodes of care for patients. An episode of care would contain information about a patient's demographic, as well as **health codes** that describe diseases, diagnoses or treatments.

For example, consider a numerator record that contains: "`Male,20-24, B32 3PP, J45`". It would describe a person who lived in the Adams Hill area, which is in the district of Birmingham, which is in the region of West Midlands in the nation of England. At some point in the man's hospital visit, administrative staff would have recorded the ICD-10 health code for "asthma". ICD-10 is a classification of diseases that is maintained by the World Health Organisation. It is one of many classification systems that are used to standardise the way health records are marked-up to suit large-scale analysis.

The population data would be managed in **denominator data** tables. The tables would be able to indicate the total number of males aged 20-24, who lived in the district of Birmingham, the region of West Midlands, or the nation of England. The RIF would link numerator and denominator data for the same resolution to provide a rate expressed for the district of Birmingham or the nation of England.

Environmental studies may have to consider **covariates**, which are any variables that may be predictive of the outcome under study. In the RIF, covariate data are linked with numerator and denominator data at varying levels of geographical resolution. Common types of covariates include:

- exposures of various pollutants
- socio-economic level
- ethnicity

Covariate data may be available at certain geographical resolutions but not others. For example, pollution data may only be available for very specific administrative areas, and ethnicity data may only be available for broad ones.

The results of linking numerator, denominator and covariate data would be difficult to interpret unless the locations could be set in geographical maps. The RIF uses **map data** to allow scientists to define areas that either define their study area or some comparison area. For example, they may want to compare the results they observe in one district of England with the results they may observe for the same demographics and health outcomes at a national level. When studies are processed, the tool can overlay information layers from the results onto areas of interactive maps.

Before we move onto discussing specific design decisions that relate to the ETL, we need to stand back and consider design forces that influenced the RIF tool suite in general. In Section 3, we will discuss some general design decisions that influenced the way the ETL would be developed.

3 A Review of Decisions about the General RIF Project which Influenced ETL Development

The ETL is a component in a larger system, so the design forces which shaped the project in general have had a direct bearing on the way the tool has been developed. There were several major factors that influenced development before we even began to properly prototype the ETL:

- We had to rewrite the previous version of the RIF because the code base was entirely reliant on working within ArcGIS.
- We had to design the tool suite so that it could be supported with SQL Server or PostgreSQL databases.
- In reviewing SAHSU's legacy data processing workflows, it was clear we needed to support ETL activities through both batch and interactive modes
- We had to decide what kind of IT skill we would expect RIF data managers to have
- We decided to break the monolithic tool structure of RIF v3.0 into multiple tools and to adopt a three-tier architecture to help organise the code base for all of them.

3.1 Understanding the Business Constraints that Affected the ETL Development

Although an in-depth project history of the RIF would merit its own detailed discussion, we review some of it here to set the context of how the ETL had to evolve. RIF v3.0 was developed as a Visual Basic plugin that worked within the ArcGIS framework. That plugin could be made to work with an underlying database that was Microsoft Access or Oracle. Development was focused on features for the scientists, and it depended on the ArcGIS framework and underlying database to handle data management issues.

Eventually, ESRI, the makers of ArcGIS, decided it would no longer support Visual Basic extensions. Please see https://blogs.esri.com/esri/arcgis/2011/01/11/vba-licensing-at-arcgis-10/ for more information about this issue. SAHSU was faced with one of two decisions: either continue to evolve RIF v3.0 using deprecated versions of the ArcGIS tool chain, or rewrite the entire code base. We chose the latter, and decided to rewrite the RIF using free, open-source technologies. In many ways, rewriting the code base allowed us to treat RIF v4.0 as a new project. However, it would retain two legacy characteristics:

- Its feature set would have to retain most of the same features that were provided in version v3.0
- It would need to support the underlying RIF database schema, which had evolved in response to SAHSU needs.

The stability of scientist features allowed us to shift the emphasis of development from being user-driven to being machine-driven. Although the basic features for supporting disease mapping and risk analysis had not changed much over a decade, the needs for supporting larger and more complex data sets had. As well, in light of the previous release's vulnerability to deprecated technologies, we had to consider how to make the software respond better to the prospect of obsolescence.

> **ETL-1**: *Whereas the main emphasis of development in RIF v3.0 was on user-driven features, the main emphasis for RIF v4.0 is on data-driven features and the need to have a system can better respond to changes in tool-chain support.*

The legacy database schema had evolved specialised permission features, views and procedures to help it support data management activities within the context of SAHSU. Although many of its aspects have since been generalised or even removed, we still recognise that the schema is a legacy artefact that must be supported by whatever ETL tool is developed. This is an important consideration to bear in mind, because many off-the-shelf data processing solutions tend to work best when you create a database in the way that they imagined it should have been designed.

> **ETL-2**: *The ETL must be able to extract, transform and load data to a target production database that uses a legacy database schema.*

Given our concern about being too reliant on a single licensed technology, and given our concerns of how off-the-shelf tools would transform our legacy data sets to support our legacy schema, we decided to build a bespoke ETL.

> **ETL-3**: *The RIF will use a bespoke ETL developed in-house rather than depend on other off-the-shelf ETL alternatives.*

Our project has attracted the interest of other projects besides SAHSU's. Many research groups want to use the RIF but also want to retain the ways that they use to process data. Their data processing workflows can vary depending on factors such as the computing environment, feature priority, the skill of data administrators and the complexity of data sets.

In crafting version 4.0, we have had to become more aware of making a data processing solution which can be generalised to meet the needs of other organisations, and yet remains fit to support our own. For example, we intend to use the ETL to replace much of the bespoke code we've developed to clean CSV files. However, other

projects will likely retain their legacy scripts for doing the same thing. We must include support for data cleaning to achieve our own in-house goals, but we can't overemphasise it in design at the expense of not developing other features that would support their needs.

> **ETL-4**: *SAHSU needs to generalise its own bespoke approach to processing data sets for environmental health so that the ETL can support a broader community.*

As another example, SAHSU chose to use PostgreSQL to be the underlying database that would support the tool suite. However, our partners at the Centers for Disease Control (CDC) needed the RIF to be able to use SQL Server instead. In order to accommodate both use cases, the ETL needs to be generic enough to support both back end technologies.

> **ETL-5**: *SAHSU needs to generalise its own bespoke approach to processing data sets for environmental health so that the ETL can support a broader community of projects.*

> **ETL-6**: *The RIF needs to be designed so that it can operate in environments that use either PostgreSQL or SQL Server databases.*

3.2 Supporting SQL Server and PostgreSQL Databases

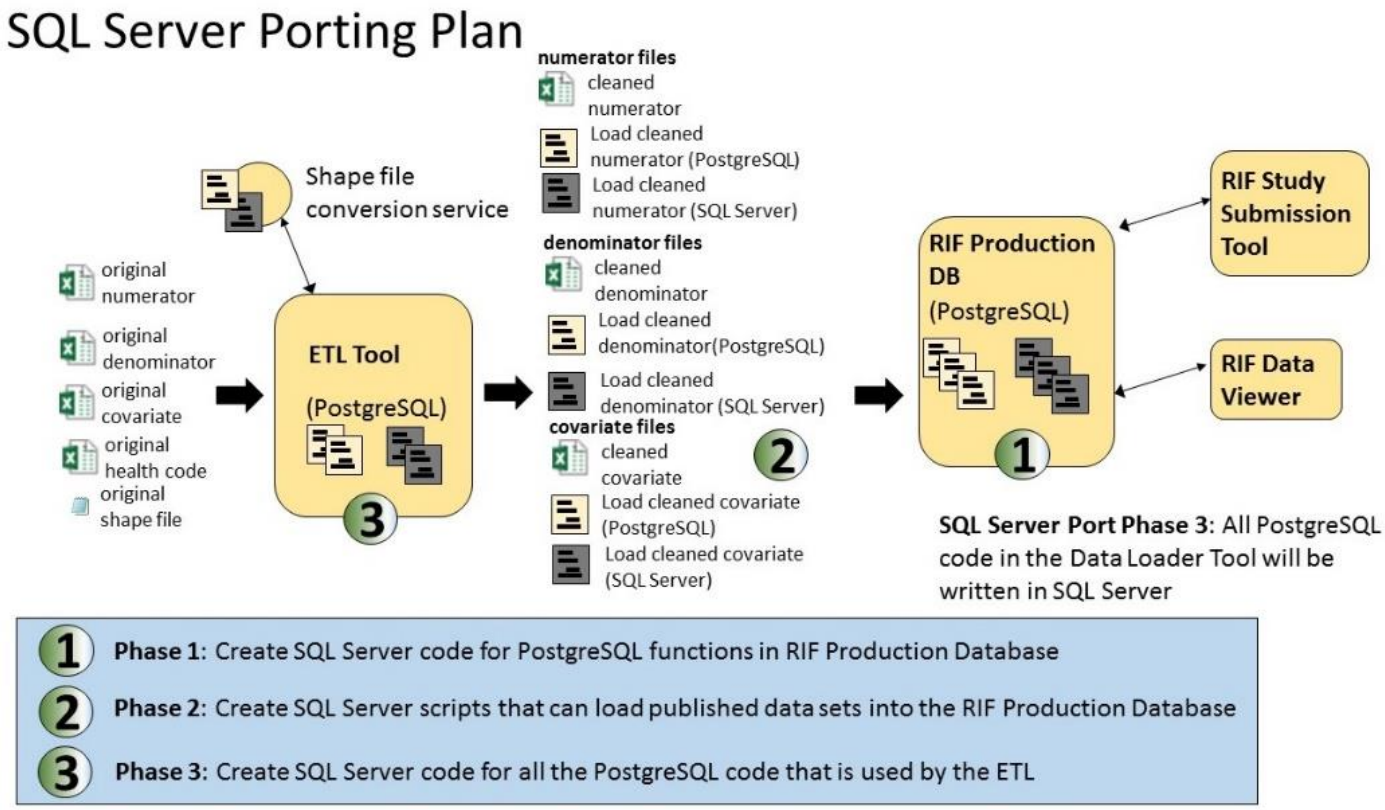
The ETL has a scope of development that could quickly complicate its design. Apart from its need to support two kinds of databases, it also needs to transform imported data sets so they can work within a target database that has special table views, security-based user roles and a legacy schema.

In order to simplify the tool and to manage database portability issues, we decided to isolate the activity of the ETL from the rest of the RIF tool suite. Instead of publishing a data set directly in the production database, it will instead generate a processed data set and scripts that can be used to load it in either kind of target database.

For example, it will process a CSV file of cancer health events by producing a cleaned version of the same file and two database script files: one which would load it for PostgreSQL and another script which would load it for SQL Server.

> **ETL-7**: *In order to isolate the complexity of a generic ETL system from the rest of RIF code base, ETL design will focus on interoperating rather than being integrated with the RIF database. Instead of making database calls that directly load finished data sets into the production database, the ETL will produce a cleaned version of an imported data set and programming scripts which will load it into PostgreSQL or SQL Server databases.*

The ETL is isolated to help limit the overall complexity of the RIF code base. These stages are shown in the following diagram.



Phase 1: Porting the RIF production database. All of the PostgreSQL functions in the production database will have corresponding functions written in SQL Server. The database will come with its own fake data sets, and testing will assume that some process has correctly loaded that data into the RIF database.

Phase 2: ETL will generate load scripts for both databases. The ETL will create PostgreSQL and SQL Server scripts that can correctly load a processed version of an imported data set. For testing purposes, we will assume that once the data sets have been loaded into the correct target areas of the RIF production database, the scientist applications will work correctly.

Phase 3: ETL is ported. Any database queries that the ETL uses to do its work will be written to support both PostgreSQL and SQL Server. For testing purposes, we assume that it will correctly create database load scripts.

Isolating the ETL from the rest of the tools and the RIF production database allows the RIF tool suite as a whole to decouple concerns about how the data are transformed from how the data are used in scientific applications. The ETL's emphasis on being interoperable with the RIF database rather than being tightly integrated with it makes it easier to divide the code-base into processing stages that simplify testing.

The stages let us economise on resources needed to do database porting. Porting database code from one system to another (e.g.: PostgreSQL to SQL Server) can require a lot of time from database staff who may be deployed on many activities. The stages allow us to create short, well defined work periods that can be more easily mixed with other development priorities than to schedule a single long block of committed activity.

Apart from simplifying testing and work scheduling, the stages also make it easier to make future substitutions in the tool chain. For example, other groups may decide to use another ETL tool that can be guaranteed to generate the same database scripts, or that can load the data directly into the production database. Supporting interoperability of system components supports substitution better than integrating them because in the former case, one component does not require as intimate a knowledge of others.

ETL-8: *Although initially the RIF will support only a PostgreSQL database, support for SQL Server will occur in three phases.*

3.3 Supporting Interactive and Batch Modes of Data Processing

We expect that data managers will use the ETL in two phases. Initially, they may want to go through multiple iterations of using an interactive, graphical tool to define their data loading workflows. However, once they have created them, they may want them to run in a batch mode. The batch mode requires the ETL to be able to save its workflow descriptions in a way that can be run by a command-line version of the tool.

ETL-9: *The ETL must support both interactive and batch modes of running a data processing work flow.*

ETL-10: *The ETL needs to be able to express the way it transforms imported data sets as an XML file that can be run by the batch-mode version of the tool.*

3.4 The Challenge of Appealing to Technical and Non-Technical RIF Managers

One of the ongoing design issues we have faced in creating the ETL is: what should we assume about the technical skill level of the RIF data manager? We have imagined two types of manager, who may be tasked with maintaining the RIF database for scientific projects. The first type is what we would call 'non-technical', and would describe epidemiologists who have great knowledge about the studies they want to do but who may not have much skill doing complex database management knowledge. We envision this kind of manager to rely heavily on a graphical application.

The second type of manager is what we'll call a 'technical manager', who has in-depth experience writing database queries that can do complex data management activities. We assume these people would prefer to write scripts rather than use GUIs and that they may not have a great knowledge of epidemiology.

We base these types on observations of how people often get hired in research settings. Domain scientists are recruited from the ranks of researchers and many would likely use graphical applications that masked the complexity of tasks. Database managers, who are often are recruited from the ranks of engineers, often prefer to use command-line tools or text editors to make and run complex database tasks. We don't claim these stereotypes apply to all projects, but they are useful in imagining how a typical user may interact with ETL features.

For now, we present the idea that the ETL could be designed to appeal to both groups, with non-technical managers preferring to use an interactive tool and with technical managers preferring to edit scripts. However, as our discussion expands, it will become clear that we need to be consistent when we decide to favour making the ETL cater to non-technical or technical managers.

In a graphical application, it can often present more work to mask underlying complexity for instructions that are meant to inform a machine-based activity. If the underlying complexity of instructions is simplified to suit a non-technical manager, there is a risk that the data processing activities will not be sophisticated enough to support the needs of the RIF database. Given that the ETL is more of a tool driven by the needs of data rather than the needs of scientist end-users, and given that we have an interest in minimising costs, we have decided in some cases to favour exposing the data manager to complexity rather than incur the expense of hiding it.

ETL-11: *The design will attempt to appeal to both non-technical and technical RIF data managers. However, we will opt to expose data managers to the underlying complexity of data processing actions when the effort needed to hide it through GUI features proves expensive.*

3.5 General Architecture Decisions from the Wider RIF Project

In this section, we review technology choices which were made in the broader RIF project and which had an effect on the design of the ETL. When we decided to rewrite RIF v3.0, we chose to develop a suite of interactive applications rather than one large monolithic application that did everything. The tools would communicate indirectly via a shared RIF database. Developing multiple smaller tools provided at least three benefits:

- it allowed each tool to have a minimal set of features that were targeted for different user types
- it allowed tools to use technologies which suited different needs for security and access
- it provided a hedge against obsolescence by allowing one tool to be replaced while having little effect on the others

ETL-12: *RIF v4.0 will be made of a collection of interoperating tools rather than of a single monolithic application that does everything.*

By having multiple tools, the epidemiologist who knew little about shape file data formats would not have to filter, avoid or feel compelled to understand features that targeted data managers. Whereas it was likely that the scientist applications would be used by many remotely-located users, it seemed that the ETL would be used in a secure restricted computing environment by one or two people at most. The difference in the number and access requirements of different user groups compelled us to render scientist tools as web applications and the data manager application (the ETL) as a desktop application.

ETL-13: *The ETL will be developed as a desktop application that we assume will be run within a secure isolated network that hosts sensitive health data.*

Along with the idea of having a collection of tools, a feature of end-user training would help limit problems with multiple tools trying to access the database at once. Scientist tools would not be used when the ETL was updating the database and the ETL would not be used when scientist tools were being used. There are other aspects of the RIF architecture which would probably make it unlikely these collision problems (called concurrency access problems in Computer Science) would happen. However, we felt it was a good precautionary measure to take.

ETL-14: *When the ETL is updating the RIF database, the scientist applications will not be used. When the scientist applications are in use, the ETL will not be used.*

3.6 Using a Three Tiered Architecture Design

Once we opted to make a tool suite rather than a single monolithic application, we then organised the code in ways that would affect design of the system as a whole. The code for each tool, including the ETL, is organised into three layers:

- A **presentation layer**, which contains all the code used to render the user interface screens. It is unaware of the data storage layer
- A **business concept layer**, which contains all the definitions of domain concepts, their validation routines and definitions of software services that used them. The business layer is meant to be agnostic towards both front and back end technologies.
- A **data storage layer**, which contains all the code that created queries that would exercise the RIF database. It is unaware of the presentation layer.

This way of organising code has provided three main benefits:

- it uses the Three Tiered Architecture, a design used prolifically in IT projects that involve client and server components
- the layers corresponded well with the expertise areas of different developers on the team
- the theme of each layer can provide guidance to future developers about where in the code base they would add new code

By using a widely known architecture, we made it more likely that the code base would be easily understood by a potentially large pool of future developers. The layers helped separate concerns which matched skills and technologies of people on the team. We had someone who was good at creating graphical user interfaces, another who was good at creating Java-based middleware, and a third person who was good at database management issues and did a lot of work with PostgreSQL and Oracle.

Having a clarity of theme in the design layers made it easier to define APIs and for modules in one layer to make assumptions about modules in another. For example, in order for a GUI component in the presentation layer to get data, it would use service API methods and make requests in terms of objects created from classes in the business concept layer. As long as the GUI client used the correct method names and supplied the correct parameters, it didn't have to know how the method was being implemented. The last main benefit was that future developers could use the layers as a kind of scaffolding to make future features easier to maintain.

> **ETL-15**: *The ETL application will use a Three Tiered Architecture to organise the code into presentation, business concept and data storage layers.*

3.7 Using Service APIs to Support Security, Testing and Planned Obsolescence

Once we began to rewrite the entire codebase for RIF v3.0, we decided to make it a design priority to help insulate the project from future obsolescence of the technologies on which it depended.

We made a heat map of our design architecture to identify the least and most volatile parts that would change. Our assessment of volatility was based on likely areas of future development, the number of technological dependencies and how fast we thought those technologies would age. We concluded that the different parts of the code base could be ranked in the following order of increasing volatility: the business concept layer, the data storage layer, the RIF database, and finally the presentation layer.

The business concept layer should prove to be the most stable part of the code base. Much of its code relates to definitions of domain concepts (e.g.: health codes, numerators, investigations, covariates) that have not changed much in meaning over past RIF versions. In the code base, it would be written entirely in Java.

The next most volatile layer will likely be the data storage layer. In the RIF code base, the layer mainly comprises Java code whose job is to use business objects to assemble database queries that it then sends to the RIF database to execute. Much of the work is dryly predictable as it tries to translate business concepts into database queries. Wherever possible, the language of those queries relies on aspects of the SQL standard that are implemented by both PostgreSQL and SQL Server.

The RIF Database will likely be the next most volatile part of the RIF code base. Although most of its database schema features pre-date RIF v3.0, its potential volatility owes to its large number of stored database procedures. The stored procedures are written in an SQL-based scripting language, and they are meant to help make some database queries execute more efficiently in the database. It is this collection of procedures, initially written for PostgreSQL, which must also work in a SQL Server database. As we do more of the SQL porting activity, we will find out whether some database procedures have to be rewritten so that they may both work equally well on the respective types of database.

The most volatile layer of the RIF will be the presentation layer, which holds all the code used to create the electronic forms and GUI features that scientists and data managers will use. In particular, it will be the code that supports secure web applications which will change the most rapidly. Differences in the way web applications work from browser to browser, the great variety of vendors supplying tools to support web services, and the library dependencies that come with those services make the presentation layer an area that will experience significant change over the coming years.

> **ETL-16**: *In order to design the RIF to anticipate forms of future obsolescence, we assessed volatility of the code base based on the expected amount of future development, and the volatility and number of technologies used to support specific features. We concluded that parts of the code base could be ranked in order of increasing volatility: the business concept layer, the data storage layer, the RIF database and the presentation layer.*

Ranking parts of the code base based on their likely future volatility allowed us to isolate the least volatile components from the effects of the most volatile ones. Across all tools, including the ETL, we decided to isolate the RIF database, data storage layer and business concept layer by hiding their access within a service. GUI features in the presentation layer would not access the rest of the code base directly. Instead, it would call service API methods that were parameterised by business objects.

Using service APIs provides a number of benefits:

- The ways that one part of the code base can use another become limited and well constrained, thereby limiting the scenarios we need to consider for testing
- Limiting interaction of front-end applications to service APIs also limits the visibility of code to malicious code threats.
- Most of the code can remain unchanged when front-end applications are rewritten with newer, more rapidly changing web application technologies

The implications for the ETL are that its front-end application could be easily replaced in the future.

ETL-17: *Code in the presentation layer will create or access RIF data via service APIs which hide details of the data storage layer and RIF database. The front-ends should not care what kind of underlying database is being used to support its operations. Conversely, the business concept and data storage layers should not care about how the presentation layer works.*

3.8 The Motives for using Java Swing to Create an ETL Desktop Application

We now try to refocus our discussion from design issues which relate to the whole RIF project to issues that specifically deal with the ETL. Java Swing is a very mature, almost dated Java-based technology that is used to produce electronic forms. However, we chose to use Java Swing for the presentation layer of the ETL. There were several factors which account for this decision:

- existing developer skill
- stability of the technology
- complexity of underlying software library dependencies
- anticipation of obsolescence

Given the limited time we've had to create such a large software project, we turned first to the technologies we already had skill in using. I've used Java Swing for years. In my view, it's old, it produces dull electronic forms, and it has some problems and short comings. However, it is also a stable part of the standard release of the Java Software Developer Kit (JSK), which means it does not add new dependencies in our tool chain. Tutorials, bug fixes and examples of Java Swing are widely available on the Internet, thereby lowering the maintenance cost of servicing the electronic forms. Our decision to encapsulate our most important business logic into services means that we are already able to cope with having the presentation code replaced with some other technology. We expect that in future, the ETL will be supported through web-based forms. But in the initial part of the RIF project, we would focus on making an ETL work first, and allocate scarce resources to aesthetics later.

ETL-18: *The front-end for the ETL will use Java-Swing. It is convenient to use, stable, does not add new software dependencies and we expect that in the future it will eventually be replaced by web-based forms.*

4 Specific Design Issues for the ETL

Now that we have introduced the scientific use cases that the RIF needs to support, and have discussed the general design issues which have affected the RIF tool suite as a whole, we turn specifically to the challenge of developing a domain-specific ETL whose underlying work flows can be generalised to support the environmental health studies that both SAHSU and its partners want to do.

The ETL will transform imported data sets into cleaned files that can then be loaded into the RIF production database using simple scripts it makes for both PostgreSQL and SQL Server databases. The ETL itself will use a temporary database to provide a means of iterating through transformation steps that use temporary tables. Because the output of the ETL includes both a finished version of an imported data set and scripts that could load it into the RIF production database, the ETL's database can itself be viewed as a temporary artefact. Once data managers have processed all the files they want to load into the RIF database, they can elect to delete both the ETL and its underlying database.

There are several benefits for assuming that the ETL may be deleted after it has been used:

- it can improve security by deleting temporary tables that may contain sensitive information
- it provides a hedge against obsolescence by making it easier to substitute the ETL with some alternative that can still produce the same target scripts
- it makes the phased development of SQL porting activities more meaningful

Many projects may have to demonstrate to funders or health data providers that they have minimised the amount of sensitive data they retain for their studies. One source of sensitive data are the intermediate data files that may be produced from activities such as data cleaning. By assuming the ETL database is temporary, data managers can choose to retain only the processed data sets and the loading scripts.

If it can be assumed that an entire tool can be deleted, it will help design enforce the idea that the ETL must interoperate rather than be integrated with its target production database. That assumption can help make it easier to one day use or develop another ETL which is capable of producing the files that are destined for the RIF database. Promoting future substitution in the tools provides a way for the RIF tool suite as a whole to better weather the forces of future obsolescence.

In the first phase of database porting, we will load the RIF database with fake data from SAHSULand using a bespoke set of scripts. Eventually those scripts will be able to be replaced by the set of scripts which are generated by the ETL. In the final phase of the database porting, the SQL queries used internally by the ETL itself can be rewritten.

ETL-19: *Data managers will have the option of regarding the ETL as a tool that they can delete after they use.*

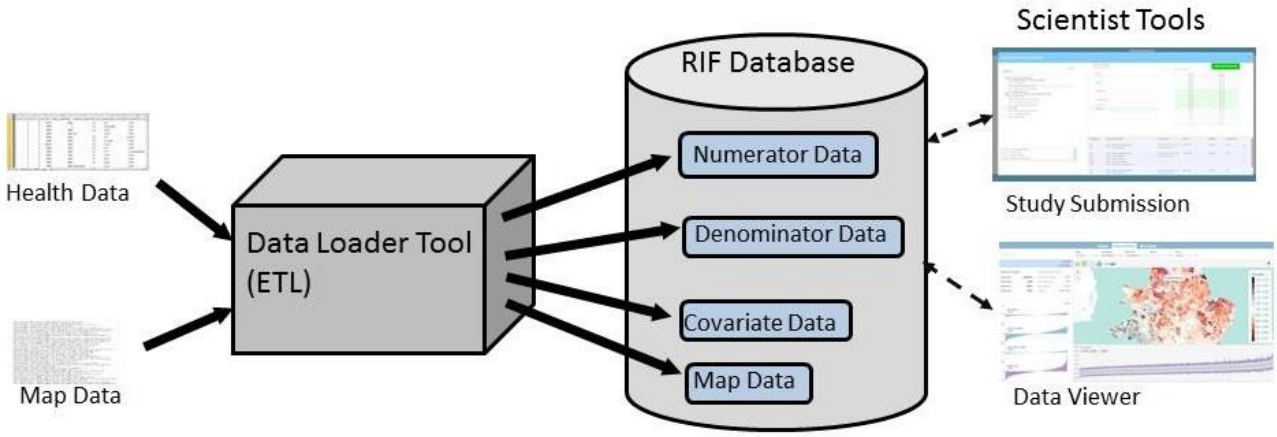
ETL-20: *The ETL will use its own temporary database in order to manage temporary data sets associated with transforming imported data sets into finished ones that can be loaded into the RIF production database.*

Our discussion of how to design ETL processes will be done in five parts:

1. describe what the final data sets must look like for them to support the RIF study design tools
2. describe the nature of the imported data sets that are destined to end up as the final data sets
3. develop a process for transforming health data
4. develop a process for transforming map data
5. special consideration for a specialised service to process map data for the ETL

4.1 Identifying Destination Areas in the RIF Production Database

For its most basic purpose, the role of the ETL is to import data sets so they can support functionality in the RIF database. The features that are developed to support that journey will vary depending on which specific part of the RIF database an imported data set is meant to serve. Therefore, it is useful to examine how data will appear in each of the following target areas of the database. The figure below shows the four destinations for data that are imported into the RIF:



The following sections describe how the characteristics of data sets that are published for these target areas.

4.1.1 Target Area: Map Data

Map data include three parts that are significant: projection information, definitions of geographical resolutions and the data that can be used to render maps. The projection data describes how the Earth's three dimensional shape is represented in a two dimensional space. Geographical resolutions describe administrative area classifications and serve two purposes:

- They define the areas that will be used to summarise health data
- They will determine the boundaries of selectable shapes in the RIF's interactive map features.

Geographical resolutions express maps at different levels of detail, and correspond to different types of administrative area classifications. For example, in the UK, geographical resolutions would show an increasing level of detail as follows: nation, region, district, ward, output area, super output area. In the US, the relevant resolutions could be nation, regions, divisions, counties, census tracts.

The geographical resolutions help define the level at which health data will be aggregated and determine the boundaries for the minimal selectable in interactive map features. They define the spatial "glue" that links health and map data together, and must be defined before other data sets are loaded. The names of these resolutions will also appear as fields in the health data sets, and will contain area identifiers that label a location in the context of broadening administrative classifications.

For example, consider a health event that describes a health care episode of asthma for a female in the age bracket of 30-34. Suppose this person lived in the Adam's Hill area of Birmingham. In the numerator table, the event could describe the person living in the ward of Bartley Green, which is in the district of Birmingham, which is in the region of the West Midlands, which is in the nation of England. If scientists wanted to look at cases of asthma by district, the health event would contribute to a summary statistic. Each of Bartley Green, West Midlands, Birmingham and England would refer to different geometric outlines that could appear as scientists moved their mouse cursors over maps of the UK.

Map data also include geometric information that can be used to render maps and the way the RIF manages it needs to takes into consideration performance issues that are related to drawing them on the screen for the users.

When the scientist web applications render a map, they do not draw all of it in one action. Instead, they initiate dozens of actions, each of which attempts to draw a part of the map called a tile. This approach of fetching multiple map tiles from the database has the effect of letting the applications draw the map quickly. This aspect is particularly important when users move within a map because each move may cause the map to be redrawn.

Working backwards from the scientist applications, the data loader needs to be able to load map data, and break it apart into individual tile records. Whereas much of the structure in health data files is preserved in the finished data sets, in map data the structure is broken down in a way that can be more easily processed by a relational database.

4.1.2 Target Area: Numerator Data

A numerator data appear in RIF database tables that contain the following fields:

Field	Meaning
`year`	the year of a health event
`age_sex_group`	A numeric field that contains codes which combine age and sex attributes.
e.g.: `ward`, `district`, `region`, `nation`	one column for each geographical resolution
`icd1`, `icd2`, etc.	one or more health code fields that describes a disease, and is typically an ICD-10 or ICD-9 code
`total`	the total number of cases

4.1.3 Target Area: Denominator Data

Denominator tables have exactly the same minimal set of fields as numerator data, except that they do not contain any health code fields.

Field	Meaning
`year`	the year of a population assessment
`age_sex_group`	A numeric field that contains codes which combine age and sex attributes.
e.g.: `ward`, `district`, `region`, `nation`	one column for each geographical resolution
`total`	the total number of cases

4.1.4 Target Area: Covariate Data

In the RIF database, covariate attributes are represented as table columns and grouped in tables based on the geographical resolution. For example, consider the case where the RIF is made to use the geography of England and geographical resolutions that include ward, district and region. It would then have three tables called `england_covariates_ward`, `england_covariates_district` and `england_covariates_region`. Each table would have a field for `year`, and then a column for the name of each covariate that had data available for the table's geographical resolution. Data for a covariate may be available for some resolutions but not others. For example, `england_covariates_region` could have an ethnicity field but `england_covariates_ward` may not. Whereas CSV files for numerator and denominator data sets will end up as corresponding tables, parts of a CSV file that contains covariate data will appear in multiple tables.

Now that we have discussed what characteristics the data sets must have at the end, we can now focus on identifying characteristics that imported data sets will have at the start. Between the start and end states of data sets, we can define a data loading process.

4.2 Characteristics of Imported Data Sets

The data processing workflows supported by the ETL also depend on the nature of the data sets that it imports. There are many differences in the characteristics of health and map data, some of which include:

Format. Health data (numerator, denominator, and covariate) is typically stored in CSV files, whereas map data will be contained in shape files.

Vulnerability to human error. Although many health data sets are mechanically aggregated, the results can often be traced back to a human-based data entry activity. Map data records are generated mechanically from sources such as satellite images, aerial photography or GPS sensors.

Variation in data quality. Map data and some kinds of health data are centrally managed. In the UK, map data are managed by the Ordnance Survey. Denominator data come from the Office of National Statistics and many covariates are based on well-curated census data. However, numerator data can be produced and managed in a far more decentralised way and produce health event records of varying quality. For example, for the UK's Hospital Episode Statistics (HES), data quality can vary between people who do data entry, vary between hospitals and vary between groups of hospitals that may use different types of patient record software applications.

Rate of record updates. In the UK, new maps are produced each decade, along with health data that are based on national censuses. HES records can be made available as data sets covering a financial year or updates that are created on a monthly basis.

Effects of optimising data. When detailed health data are summarised to suit broad geographical resolutions, few if any errors would be introduced because the process involves relying on a simple action of creating record totals. However, when map data are optimised to suit end-user viewer requirements, the process can introduce distortions that appear as errors in visualisation.

Storing data in the target database. In most cases, the structure of health records will be preserved in the RIF production database because both the source and destination data structures are tables. However, most map data that would describe an area are not tabular in nature.

Another way to consider the differences between the types of data is to compare them in terms of attributes of Big Data: volume, velocity, variety and veracity.

	numerator	**denominator**	**covariate**	**map**
Velocity	HES is yearly or monthly	ONS once a decade to correspond with major updates annual updates for minor tends to correspond with census	once every census or year for government surveys	ONS once a decade to correspond with major updates, again derived often census
Volume	up to 15 million records per year (and 200+ fields)	up to 236,000 areas with 22 age and sex groups, 40 years modelled 236,000x22x2x40 records 1 field - total	covariates never reported below ward, typically district in the UK	236,000 areas x however many points in each area, depends on original

			look up number of districts n districts x 22 x 2 n fields can also be annual with a year field	simplification of map, 1.5G file
Variety	CSV	CSV	CSV	Shape File
Veracity	not all that clean	tends to be clean because of centralised source or derived from one	very clean derived from census data or things from Dept of Transport	Very clean, again because it's been centrally maintained by Ordnance Survey

A similarity in the characteristics of data sets would suggest that there could also be a similarity of characteristics of workflows that would process those data sets. At a more concrete level, a shared process would invite the prospect of shared code, thereby helping to minimise the maintenance costs.

However, as we can see, there are enough differences in the characteristics of map and health data to assume that they won't share code. This is an important assumption because it means that we would be better to define a separate workflow for each rather than to try and find an abstraction level that could accommodate both.

> **ETL-21**: *Assume that the data loading process and the code supporting it will not be shared for handling map data and health data. Develop features to support each kind of data without thought that they will be consolidated through shared code*.

Our discussion about defining a work flow will now split into two parts, one for each kind of data. As we will see in the next section, in some ways it is more useful to view the work flows in terms of the file format of a data set rather than its meaning.

4.3 Defining a Data Loading Process for Health Data

We have already shown examples of how we've managed the complexity of the RIF by organising the code base according to various themes which separate concerns. In the broader discussion about the RIF, we have split up the code based on a number of things: end-user (i.e.: scientist, data manager); the feature area (i.e.: study submission, analysing results, data loading); and categories of system behaviour (i.e.: data storage, business concept, presentation layers).

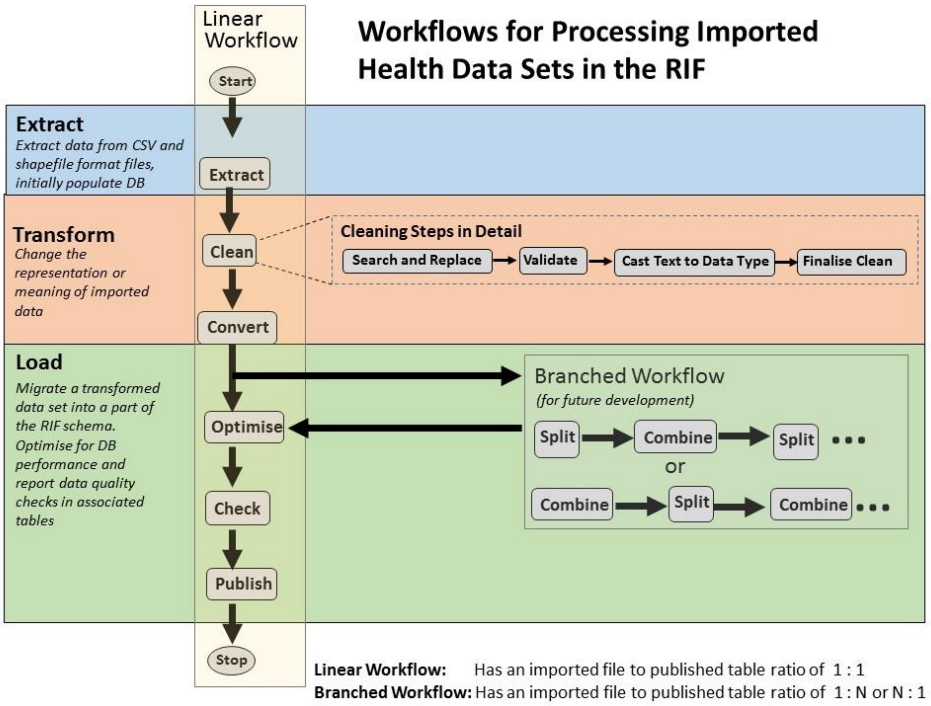
In the ETL, we also divide the code based on processing steps. These serve not just to help organise the code base, but when they are arranged into a sequence, we can use them to drive a state machine. A state machine defines a finite number of system states, and uses explicitly defined conditions to determine how the machine can go from being in one state to being in another. Using a state-machine to guide the behaviour of a data loading process has a number of benefits:

- the explicit determination of the next step can help limit the number of execution paths in the code and thereby reduce the likelihood the software will exhibit unexpected behaviour
- each state comes with a set of assumptions that makes the software easier to test
- the states can help organise pieces of code based on the kinds of activity they support
- the failure to go from one state to the next makes it easier to code rollback-and-recovery procedures that prevent unexpected results

> **ETL-22**: *The workflow steps for processing health data will be used to create a state machine that will help manage the sequence of events that turn imported data sets into finished ones.*

SAHSU has extensive experience process routinely collected health data, most of which begins as CSV files. Processing CSV files is much easier than processing map data because data in the CSV files, the temporary database used by the ETL, and the target tables in the RIF production database are all tabular in nature.

The generic workflow we've used in the ETL is shown below; we will use it to develop a discussion about linear and branched work flows that are meant to transform imported CSV files into output files that can exist within the RIF production database.



4.3.1 The Linear Workflow for Importing CSV Files

We begin with the simplest work flow – the Linear Workflow – which can be used to transform a single imported CSV file into a corresponding cleaned data set. The linear workflow is likely to be the most common work flow to support project groups, and its stages fit within the Extract-Transform-Load paradigm:

Extract. The database will read a CSV as a basic table, where all the columns are of type text and all the original field data are preserved.

Clean. Data cleaning and data validation actions are applied to the original data table to improve data quality.

Convert. Table fields are renamed and mapped to corresponding fields the table is expected to have when it is used in the production database. The expectations of mapping depend on whether a data set is destined to behave as numerator data, denominator data or covariate data in the RIF production database.

Optimise. Tables are indexed to facilitate more rapid data linking and searching activities when they are used in the RIF production database.

Check. An auxiliary table is created which can record the percentage of field values that are blank or blank for a given year.

Publish. The finished table is documented with informative comments and is ready to be used in the RIF production database.

The definition of clean can be clarified through a sequence of three sub-steps. First, **search-and-replace** will substitute field values, either to improve their meaning or standardise its representation. For example, a specialised ICD code "`J45.3`" could be replaced by "`J45`" to generalise its meaning from "`mild persistent asthma`" to "asthma". Alternatively "`J45.3`" could be changed to "`J453`" to change the way the term is

represented. Note that in the UK, 5th and 6th character positions are sometimes added to reflect local bespoke codes.

Validate will check whether cleaned values are valid and **cast-to-data-type** may attempt to treat a text value as some other data type such as an integer, date or floating point number.

> **ETL-23**: *Cleaning actions are meant to increase the value of data by improving their meaning meant for a human audience or the representation meant for a machine audience.*

> **ETL-24**: *Apply search-and-replace cleaning actions and validation activities independently. Assume that a search and replace action may miss some erroneous values.*

Now that we've identified the steps for a Linear Work Flow, we need to consider the data manager features that will be associated with them. In the following subsections, we introduce the configurable options which will be associated with a given field in a CSV file.

In order to facilitate testing and auditing, CSV data will be imported into an original table and copies of it will be retained to reflect the changes made by each transformation step. The temporary tables that are created will be included in the exported results, so that data managers have an audit trail of how data rows progressed through the data loading process.

> **ETL-25**: *The ETL will retain temporary tables that can show the changes made by each successive transformation step. These tables will be exported along with the original and finished data sets to provide data managers a way of tracing how records are changed.*

4.3.1.1 Features for Extract

When the ETL first reads a CSV file, it needs to know two key things:

- whether the first row describes data or a column names and
- what character is used to delimit columns

In order to simplify the process of reading data, we have made a header row mandatory in imported CSV files:

> **ETL-26**: *CSV files are required to have a header row that describes the field names.*

We have also adopted the approach that popular spreadsheet application such as Excel have used to show how a choice of delimiter can influence how the column data are parsed:

> **ETL-27**: *When data managers import a CSV file, they will be presented with a preview feature that shows how their choice of a delimiter, by default a comma, would cause the first few rows to be parsed.*

Data managers will need the ability to mark the importance of each CSV field. In some cases, a CSV file may contain fields which have no relevance to the RIF or a project's interest. These will appear in the original imported data table but shouldn't be promoted in subsequent transformation steps. Some fields may be required by the RIF and others may just be fields that are of interest to a project.

For example, a numerator data table may contain fields including `patient_id`, `age`, `sex` and `ethnicity`. Data managers may choose to omit the `patient_id` field from finished data sets. Age and sex will be required by the RIF database, and ethnicity could be marked as a useful field for numerator tables to contain.

> **ETL-28**: *A CSV field may be marked with importance levels of required, extra or ignore. Required fields are those that the data manager believes will be required by the RIF. Extra fields will appear in the finished data sets but serve no purpose in operations of the RIF database. Ignore means that the field will not appear in the finished data set.*

During the extract phase, each record from the CSV file will also be assigned a `row_number` value, in order to help data managers trace a transformed record back to its original location in the imported file.

> **ETL-29**: *When the ETL initially reads a CSV file into a table, that table will contain all the CSV fields and a row number field that preserves the location of a record in the original imported file.*

4.3.1.2 Features for Clean

Data managers may want to rename field names that were derived from the CSV files. For example, the CSV file may have been created with a different purpose in mind, other than that it would be used in the RIF.

> **ETL-30**: *Data managers will be able to change the names of CSV field names when the ETL creates a copy of the extract table to be transformed by the data cleaning steps.*

One of the most important design features of the ETL is the support of "RIF Data Types" that define the policies for the search-and-replace and the validating steps of data cleaning. A RIF Data Type does not correspond to the way a data type is understood in the RIF production database, or how a scientist might view it. Instead, it is really a name assigned to a reusable bundle of procedures that will be used to do data cleaning in a particular way. It comprises: a name, a description, a cleaning policy and a validating policy.

Where supported by the database, domains (i.e.: a direct implementation of the data type with verification in the database) will be used. However, the ETL will not assume cross-database support exists until it is proven. Domains have the advantage of speeding up data inserts since the relatively slow procedures are not required. For example, it takes the RIF production database 3 to 4 seconds to create a study without domains. With domains we expect it to be about a second because the domains reduce the amount of procedural code that needs to be executed.

The cleaning policy, which governs how a field value can be replaced, has three options: no cleaning, cleaning using a set of user-defined rules, or cleaning using a database function. No cleaning means the original values will be preserved. Each cleaning rule comprises a search expression and a replace expression, which can both be written using the syntax of regular expressions. The rules are applied in order and the one which provides the first match will be used to change the search expression into the replace expression. Database functions are stored in the ETL's database and are meant to have only a single parameter that is reserved for the field value.

The validating policy is designed in a similar manner to the cleaning policy. No validating means that all values that are transformed by the cleaning policy will be acceptable and remain unchanged. Each validation rule comprises a regular expression that indicates a valid value. Rules are applied in sequence, and a value is considered valid if at least one of the validation rules provides a match. Like the database functions used for cleaning, validation functions accept a single parameter for the field value and return true or false. If a field is considered invalid, then its values becomes null.

Some RIF data types may use cleaning, validating or both. Here are some examples:

- **US Zip code**: has no cleaning policy but uses a validation policy rule that asserts whether the field value is a sequence of exactly five numbers.
- **Sex**: has a cleaning policy of rules to convert "`M`" and "`f`" to "`1`" and "`2`" respectively, and uses a validation policy rule that asserts whether the value is either a 1 or a 2.
- **ICD**: has a cleaning policy that removes any dot "." character from an ICD code and has no validation policy
- **Birth Date**: has a cleaning function that imputes partial dates and a validation function that asserts whether the date format matches: "`dd/mm/yyyy`".
- **Birth weight**: has minimum and maximum values for validating. Sometimes the fields are designed to measure pounds but are recorded in kilograms.
- **Maternal age**: has a minimum and maximum ((get in concept of plausibility)

We therefore put forth the following design decisions:

ETL-31: *The ETL will allow data managers to define custom RIF data types, which describe how a CSV field should be cleaned. A RIF Data Type comprises a name, description, a cleaning policy and a validation policy. The cleaning policy defines the way values will be searched and replaced, whereas validation policies define the way that field values are considered valid or invalid. Cleaning policies and validating policies are applied independently of one another.*

ETL-32: *A cleaning policy will have three options: do nothing, use rules or use a database function. Each rule will comprise regular expressions for search and replace values, and the rules will be applied until the first one that produces a match is found. Functions are database procedures that feature a single parameter for the field text value and return a cleaned text value.*

ETL-33: *A validating policy will have three options: do nothing, use rules or use a database function. Each rule comprises a regular expression that indicates validity, and a field value is valid if any of the rules produce a match. Functions are database procedures that feature a parameter for the field text value and return a Boolean result.*

ETL-34: *The rule and database function options for the validation policy will be able to consider whether a blank value should be considered valid. Data managers will be able to specify whether a data field is considered required.*

Some RIF Data Types will have special meaning for the RIF. For example, fields of Sex, Age will have a very specific meaning for numerator and denominator tables. Age might use a cleaning function to standardise an age band category (i.e.: age 22 converted to "4", an identifier for representing the 20-24 age group). Sex may have cleaning routines that are different for US-based data sets and UK-based data sets. The ETL will allow data managers to change the cleaning and policy features for reserved RIF data types, but they will be warned that their actions could affect key RIF operations.

ETL-35: *The ETL will warn data managers when they attempt to change cleaning and validating policies for reserved RIF data types such as Age and Sex.*

Data managers will need an audit feature that can record the nature of changes that the data cleaning activity makes to imported data. Auditing ceases to be useful when either too little or too much information is recorded about a change. If a cleaning routine is simply changing the format of a field, then it may not be useful to record that it has done so if it manages to change every field value. For example, it probably isn't useful to record that a postal code was stripped of spaces when all of them would be changed. Some audit entries may only need to indicate that a field was changed. The detail could be used to help compile statistics on changes made per record field. For important fields, it may be necessary to record both the original value and the value it was changed to become.

To support these needs, the ETL lets data managers associate three levels of auditing with a field.

- **None**. Any change in the field value will not be recorded.
- **Include field name only**. Changes will be recorded in the form "Field X was changed".
- **Include detailed change description**. Changes will be recorded in the form "Field X was changed from Y to Z".

ETL-36: *Each CSV field can be associated with one of three levels of change auditing: none, include field name only, and include detailed change description.*

For each imported CSV file, the ETL will create a corresponding change table that describes how field values were transformed. In some cases, it is desirable to have a detailed description of what was changed. In others, less information may be needed and for some fields, the data manager may not want to record changes at all.

4.3.1.3 Features for Convert

Convert will map a cleaned data field to some field that is expected to appear in the RIF production database. In order to do this, the data manager will have to specify what part of the RIF database the data set is meant to support:

> **ETL-37**: *The ETL will require data managers to indicate which of three target areas describe the end-use for an imported data set: numerator data, denominator data, and covariate data.*

The convert step was developed mainly in response to a need for age and sex fields to be combined in the numerator and denominator tables that appear in the RIF database. The ETL needs a way of being able to map two cleaned fields to parameters of a function that converts age and sex into a single integer value. The conversion function has two fields which are specifically named `age` and `sex`. A mapping feature needs to consider that fields of type Age or Sex may not have the names of their data types. For example, the name of a cleaned CSV field may be `patient_sex`, but it would map to the `sex` parameter of the function.

> **ETL-38**: *If a CSV field is marked as required, and has a data type of Sex or Age, the ETL will prompt the data manager to ensure that the cleaned field maps to a specific field that needs to be called "age" or "sex".*

4.3.1.4 Features for Optimise

The Optimise step relates to features that help make database queries perform efficiently. The three main feature areas we considered for end-user features include: assigning primary keys, creating database table partitions and applying database indices.

The first two features are either handled automatically by the RIF or are outside the scope of the ETL's functionality. Primary keys are assigned to CSV data sets based on the needs of the RIF production database, and are not based on the decisions of the RIF managers. Therefore, no features are provided which allow them to choose their own primary keys.

Database partitioning is a technique used to help make large database tables easier to maintain and query. It is often used to help break a large index into smaller ones that are based on some attribute. SAHSU often partitions many of its data sets by year. The main benefit of partitioning an index by year is that in queries that filter by year, the database can rapidly find the data sets for relevant years without having to look for records whose years are not relevant. Another benefit is that it can be easier to remove and archive old data that belong in partitions that become rarely used in queries.

Partitioning is a complex topic that is dealt with by the RIF production database and not the ETL. The RIF database may try to partition a table after it has been loaded, but it does not assume the ETL will generate scripts that support the partitioning activity.

The remaining area is the only one of the three we considered that resulted in end-user features. Creating an index for a table field makes most sense if it is likely that there will be some SQL query that uses it in the WHERE, ORDER BY or GROUP BY clauses. We can use this guideline and the importance level of each field to make some intelligent decisions about when indexing should be done automatically or be something left to the decision of RIF managers.

If a field is marked as `ignore`, then it will obviously not need indexing because it will never be used. If a field is marked as `required`, then it is guaranteed that the field will be used by one or more queries that are used either within the ETL or in the RIF production database it supports. Therefore, we can automatically index required fields and save the RIF manager the work of specifying that the index should exist

If a field is marked as `extra`, then it is possible that in future development, the field may be exposed to end-users as part of a filter or search and retrieve activity. In this case, it makes most sense to let the RIF manager decide whether a field should be indexed.

> **ETL-39**: *Fields marked "required" will automatically be indexed. If they are marked "extra", data managers will be able to indicate whether they want an index created for them.*

4.3.1.5 Features for Check

Check will cover features for creating data quality assessments of CSV fields. We included the two most common data quality checks that SAHSU has used over the years: the percentage of blank values for a field and the same check done per on a per-year basis. For each data set, the ETL will create a corresponding table which holds the results of these checks, if they are wanted by the data managers.

> **ETL-40**: *Data managers will be able to specify whether a field should be associated with a `percent empty check`, a `percent empty per year check`, neither of these checks or both of them.*

The Check step will also identify duplicate records in a data set. Data managers will be able to mark a data field to indicate whether it should be used in a duplicate check. All fields which are marked as duplicate criterion fields will be used to determine if two rows are considered the same. When the ETL encounters one or more duplicate rows, it will retain the first one.

The published data set will contain two "yes/no" fields that support this feature: `is_duplicate` and `kept_duplicate`. The first field indicates whether the row is some duplicate of another row in the table. The second field indicates whether a duplicate field was kept or marked for deletion.

For example, consider entries for a numerator table shown below. If only `age`, `sex` and `district` are marked as duplicate criteria fields, then the first record would be retained and the second would be marked for deletion.

admission_date	age	sex	district	icd	(other fields)	is_duplicate	keep_duplicate
20/03/2004	26	M	Birmingham	J45	...	Y	Y
15/04/2004	26	M	Birmingham	J45	...	Y	N

If we also marked `admission_date` as a duplicate criterion, then the table would look like:

admission_date	age	sex	district	icd	(other fields)	is_duplicate	keep_duplicate
20/03/2004	26	M	Birmingham	J45	...	N	null
15/04/2004	26	M	Birmingham	J45	...	N	null

The ETL will not actually delete any record. Instead, it will mark the row as deleted so that as much data from the original CSV file can be preserved.

ETL-41: *CSV fields which are marked as `is_duplicate` will be used by the ETL to identify duplicate rows. Two rows are considered duplicates if all of the fields marked in this way have the same value.*

ETL-42: *The ETL will not delete duplicate rows. Instead, finished data sets will show duplicate rows that are marked for deletion using two fields:* `is_duplicate` *and* `duplicate_kept`*.*

4.3.1.6 Features for Publish

The last step in the workflow is Publish, which is not associated with any configurable properties. Publish signals the ETL to begin generating a bundle of results files in some directory that is specified by the data manager

ETL-43: *When the ETL publishes a data set, it will create the following outputs:*

- *A copy of the finished data set*
- *PostgreSQL and SQL Server scripts that can load that data set into the RIF production database*
- *An archive zip file.*

The archive zip file will contain the original CSV file, CSV files for the transformation steps in the work flow, CSV files containing change logs and data quality checks, and the first two items listed above.

In order to support administrative aspects of data management, the ETL will using naming conventions to mark published files.

ETL-44: *Publish will use the following prefixes for finished data sets:*

- `num_` *for numerator*
- `den_` *for denominator*
- `cov_` *for covariate*

These conventions come with their own pros and cons that may lead us to change them in the future. "`num_`" would be a more readable prefix than "`n_`", but the latter choice better economises on the length that table names can have in some databases.

4.3.2 The Branched Workflow for Importing CSV Files

The other work flow for processing CSV files is the Branched Workflow. We define it, but defer its implementation because it will be complicated to support and may cater more for SAHSU-specific operations than for more general use cases.

SAHSU sometimes splits or combines the health files it uses. For example, we may choose to combine HES files that contain records from different financial years. We may also want to split a single file containing all the data fields into multiple files that contain different subsets of fields. At least two considerations can influence how or whether a file should be broken up:

- **frequency of use**: Isolating frequently used column fields allows the table to be smaller and therefore show better performance in table scan operations.
- **sensitivity:** the most sensitive fields could be destined for a database table that has more restricted access than other tables that come from the same file

> ETL-45: *The ETL should be able to split or combine health files, in order to improve database performance or security.*

Early in development we realised that supporting split and combine operations would be much more difficult to implement than a workflow that only applied to a single file. Furthermore, we assumed that such operations would not likely present a high priority for other organisations which processed fewer records than SAHSU does. In an effort to favour making a generic solution, we deferred developing features to support one of our own requirements. For now, we must assume that split and combine occur outside the scope of the RIF:

> ETL-46: *Adding features to support split and combine operations will be deferred. For now, data managers should split or combine input files before they process them with the ETL.*

4.3.3 The Decision not to Support Encryption, Anonymisation or Pseudonymisation

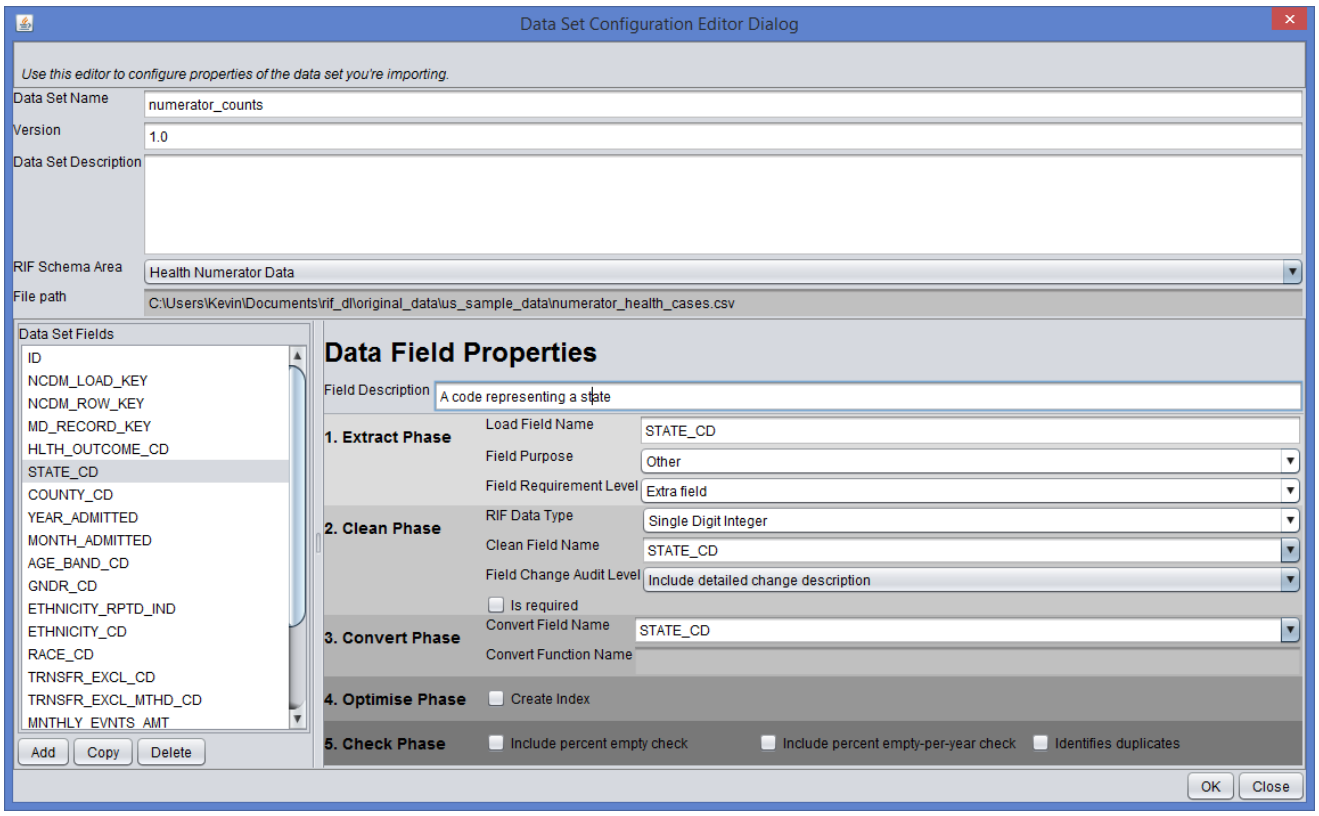
Many of the fields that appear in SAHSU's health databases are encrypted, but we decided that the ETL would not provide features to support this kind of activity. We were concerned that agreeing to support the ability to encrypt fields would add new encryption technologies to the tool chain needed by the tool suite. There was also the security matter of managing keys used to encrypt or decrypt. With respect to algorithms used to anonymise or pseudonymise fields, we decided that we would be at risk of developing feature sets that would be better serviced by other tools.

Given the scope of development for the RIF as a whole, along with consideration for our limited developer resources, we concluded that it would be better to focus our efforts on other areas. We decided that if data managers want to apply encryption, anonymisation or pseudonymisation algorithms to data fields, they should do it prior to using the ETL.

> ETL-47: *Adding features to support split and combine operations will be deferred. For now, data managers should split or combine input files before they process them with the ETL.*

4.4.4 Example Screenshot of ETL Properties for Health Data

The following screen shot indicates the main dialog of the ETL that data managers would use to configure the transformation of imported CSV files. The RIF Schema Area indicates the target area of the RIF database that the data set will support. The basic linear work flow steps are enumerated, along with their corresponding configurable properties.



The GUI features can help disable options to minimise data entry work or to prevent errors. For example, if the Field Requirement Level field is marked as `Required Field`, then all of the options for Optimise and Check will be disabled because ETL will assume that all the options should be ticked. If that same field is marked `Ignore`, then again the fields are disabled because they are not relevant to a field which will not appear in the published data set.

4.4.5 Supporting Configuration Hints

The more configuration options that the system supports, the more work this can cause the data manager. For example, based on the screenshot we showed for configuring CSV properties, there would be 14 configuration options per field. Suppose a numerator CSV file contained the following fields: `year`, `patient_id`, `age`, `sex`, `ethnicity`, `ward`, `district`, `region`, `icd1`, `icd2`.

The data manager would have a possible 140 separate controls to adjust to configure the CSV file. SAHSU tends to use HES files to provide numerator tables and a HES file can contain over 200 hundred fields. The ETL needs features which allow it set intelligent default values for fields.

In order to reduce the burden of data entry for the data manager, the ETL has features for supporting configuration hints that are based on naming conventions of data sets and fields. Data managers can associate regular expression patterns with default values of general data set properties. These include: version, a description and the target area of the RIF production database. Data managers can also define regular expression patterns to match CSV field names with field properties.

As an example of a data set hint, the regular expression `^cancer.*` could be used to set the target area of the CSV file `cancer_data_2012.csv` so that it is set to "Health Numerator Data". As an example of a field hint, `.*year.*` could be used to set the data type to RIF data type "Year" for any field name that contains 'year'. A field hint of `^year$` would be more specific, meaning that the exact field name was `year`. In this case, we may decide that this is a required field for the RIF, whose numerator tables expect to have a field by that name.

ETL-48: *The ETL will allow data managers to define default configuration values based on regular expression patterns that match either the name of the CSV file or the name of one of its fields. For each of data set level and field level hints, only the first hint that matches will be used.*

4.4.5 Decision not Manage Health Outcomes in the RIF Production Database

We include this section to describe an important feature of the RIF that the ETL does *not* include. Early in the project, we envisioned that the ETL would allow data managers to load health code taxonomies such as ICD-9, ICD-10 and OPCS classifications into the RIF production database. To make a further step, we thought we would be able to pre-load distributions of the RIF production database with these collections of health codes.

Once loaded into the database, the codes would be available in the scientist's study submission tool. Users would create their studies by specifying health codes that were provided by the database. When studies were run, the RIF would search numerator data tables for any records that contained those health codes.

Later in the RIF v4.0 development, we decided that the tool suite would delegate to taxonomy services to provide terms rather than rely on the tables of terms that would be stored in the RIF production database.

There were two reasons why we made this decision. We were concerned that pre-loading code sets such as ICD-10 might violate the licensing agreements that govern their use. By including all the ICD-10 terms in our own GPL-based distribution, we wondered whether there would be conflicts between the terms of a GPL license and the bespoke license that is managed by the World Health Organisation. For example, if we did ship whole taxonomies with the software, then users could accept the terms and conditions of GPL and never be aware that they also had to accept the terms and conditions of using ICD-10. Our interest in respecting IP laws made us decide not to pre-load health code taxonomies.

Later on, we realised that the complexity of the RIF production database would increase when it tried to manage terms from different taxonomies. For example, we had tables that took advantage of the way ICD-10 codes expressed levels of hierarchy in their naming conventions. As an example of how meaning is embedded in the code name, consider the ICD code `J45.51`. The "J" describes the level of diseases of the respiratory system, J45 describes asthma, `J45.5` describes severe persistent asthma and J45.51 describes severe persistent asthma with (acute) exacerbation. The hierarchy levels could be described in a table with columns for one character, three character and four character versions of the ICD-10 codes.

However, other taxonomies may not follow the same coding conventions. Other code systems may simply have an identifier and a label, which reference some node within a network of related terms. If we were to provide bespoke features for each new taxonomy the community wanted to have available through the RIF, it would overcomplicate the RIF production schema.

Having the scientist tools use software services to supply taxonomy terms simplifies problems of both intellectual property and software complexity. Software services are less likely to violate the problems of distribution in a software license. Community-based taxonomy services could take on their own forms of registration to ensure that RIF project groups have explicitly agreed to the terms and conditions of using ICD codes. As well, with software services, we are not distributing the terms in the sense of bundling them wholesale with the software.

Delegating to services means that the terms could be stored and retrieved by technologies which are better suited to managing them than the technologies that are included in the existing RIF tool chain. For example, an ICD-10 lookup service could be made to use MongoDB as a way to provide very fast access to terms. Taxonomy

services could also leverage various semantic web technologies that would be able to provide intelligent search results based on the kinds of relationships that may exist between terms.

> **ETL-49**: *The RIF Database will not itself manage collections of health codes that provide the filtering criteria for selecting relevant records in numerator data. Instead, the scientist tools will rely on taxonomy services to provide terms. We put the management of health code taxonomies outside the scope of the ETL because of concerns related to violating license terms of shipping terms, and because managing them in the RIF database would add unwanted complexity.*

Although the architecture delegates the task of retrieving health codes, we will need to create default implementations of some of the taxonomy services. For example, we have been developing a default ICD-10 taxonomy service that could be used by different RIF project groups.

> **ETL-50**: *The RIF will include default service implementations of some taxonomies such as ICD-10. Projects will be able to use the services with a copy of the health codes they have obtained through their own efforts to get a license.*

For our design, our decisions about not managing health code collections means that the ETL does not need to support transforming a CSV containing health codes into a "health code" target area of the RIF production database. However, we recognise that there is value in allowing data managers to use the ETL to systematically clean CSV files that they would use to build their own taxonomy services. Therefore, as well as having the target areas for map data, numerator data, denominator data and covariate data, we also have an "other" area. The "other" area tells the ETL that none of the columns are destined to become part of the RIF, but apply cleaning and validating operations anyway. The result of processing a CSV file bound for an "other" target area is that the ETL will generate a cleaned version of a CSV file containing health codes.

> **ETL-51**: *The ETL allows data managers to map an imported CSV to an "other" target area. This means that the tool will process the CSV and produced a cleaned version, but the file is not destined to be used in the RIF production database. Support for an "other" target area allows data managers to use the ETL for purposes other than directly supporting the RIF.*

4.5 Defining the Data Loading Process for Map Data

The design of the map data workflow is heavily influenced by the needs of a niche technology that the RIF must use to support the efficient rendering of high resolution maps in the scientist tools. To best appreciate the decisions for handling map data that we've made in the ETL, we need to first examine the needs of technologies we've used to retrieve and display maps in the web applications.

One of the base requirements of the RIF is that scientists need web-based applications to help them create studies and to view the results. A core feature for supporting both of those activities is to have interactive maps that can be redrawn quickly as the user zooms in and out, or pans right or left.

We found early in the project that the best web-based library for supporting interactive maps was Leaflet.js. Leaflet is able to draw images quickly because it relies on tiled maps, which are displayed in web-browsers by seamlessly joining many map tiles that could be retrieved from all over the Internet. A map tile is a fragment of JSON that is effectively a vector representation boundaries for some geographical area.

In our project, all of the tiles will be retrieved from a web service that is able to return tile data from the RIF production database. When high resolution maps are being rendered quickly, Leaflet's requests for tiles can cause large amounts of data to flow from web service to web page. When both the browser and the service are running on the same host, the performance may not necessarily suffer. However, we expect that in most projects they will run on different hosts and therefore performance will likely cause map rendering to slow down and create an unacceptable end-user experience.

Therefore, it is critically important to minimise the time needed to process tile data. There are three areas of the RIF architecture that could be altered to help reduce processing time. The web client that renders the maps could use various schemes for caching tiles either in memory or in temporary files that reside on the machine on which the client runs. Leaflet does not appear to have its own caching mechanism, but a plugin will likely be available in future versions.

Another feature to consider is a caching mechanism that the web service uses to find a tile before it requests one from the database. Here, the approach would be to either cache map tiles in files or hold them in the memory.

The third and most important way to reduce processing time is to pre-process map tiles that are loaded into the RIF production database. This area can yield significant performance gains because all the work to prepare a tile for viewing in a map would be done once rather than each time the tile was retrieved. Furthermore, additional performance gains are made through database caching mechanisms.

> **ETL-52**: *There are three main candidate areas of the RIF architecture that could be examined to reduce processing time associated with retrieving a map tile: client-side caching; caching done by the RIF web services; and pre-processing tiles in the database. We consider only the last in this discussion because the first two are outside the scope of the ETL discussion.*

The need to pre-process tiles brings this activity within the remit of the ETL rather than the RIF production database, the middleware or the web clients.

> **ETL-53**: *Map tiles will not be created dynamically when web-clients request them. Instead, static, pre-computed map tiles will be retrieved.*

4.5.1 Making Map Tiles that work with Leaflet

Our goal for pre-computing map tiles is to make sure that they can be used by Leaflet. Working backwards from the web-client, we will need to ensure that each map tile has the expected dimensions.

> **ETL-54**: *The ETL will create map tiles that have pixel dimensions of 256x256.*

As well as expecting map tiles to have x and y attributes, Leaflet also expects them to support standard zoom levels that have been defined by the Open Geospatial Consortium. The standard levels begin with a zoom level of 0, which can show the whole world in a single tile. Each successive zoom level doubles in both dimensions, so a single tile is replaced by 4 tiles when zooming in. This means that about 22 zoom levels are sufficient for most practical purposes. Of these levels, the RIF will initially support levels `6`, `8` and `11`.

> **ETL-55**: *Each map tile will be associated with an OGC defined zoom level that includes levels `6`, `8` and `11`.*

These levels seem to be sufficient to provide an acceptable zooming feature for a UK-based geography. However, the RIF may need more zoom levels if the amount of detail in maps causes performance problems.

ETL-56: *In future, the RIF may need to support additional zoom levels if the amount of time needed to render a map at `6`, `8` or `11` becomes too great. This problem may occur in geographies which are large and very detailed.*

ETL-57: *Map tile names will follow the naming conventions of the OpenStreetMap standard, which means they will have an X, Y and zoom level component.*

The use of a zoom level allows the database to economise on the amount of data it sends back in a tile; instead of returning one tile with a great amount of detailed information, it can send back a tile that contains information about fewer geographical features. However, we have to consider the computational overhead of fetching tiles having different zoom levels. Because we are pre-processing all of our tiles, we need to make tiles that have already been created to anticipate each zoom level:

ETL-58: *For any map, the ETL will generate a set of map tiles for each zoom level that is supported in the RIF.*

The tiles must also be rendered using a specific projection. A projection determines how a three-dimensional surface will be rendered as a two-dimensional area. To understand the importance of a projection, imagine the task of flattening a large piece of orange peel – in making it sit flat, you would inevitably have to distort some parts of the curved surface. The same task applies to rendering the three-dimensional character of the Earth's surface. The distortion caused by projection explains why areas like Greenland appear much larger on maps than they actually are. Areas will appear the least distorted near the Equator and the most distorted near the North and South poles.

If a tiled map is meant to seamlessly join map tiles, then it makes sense that they must all have been distorted using the same algorithm. Leaflet requires that its map tiles are created using a specific projection scheme:

ETL-59: *The ETL must produce map tiles which are created using a Web Mercator (WGS84) projection system for coordinates.*

4.5.2 The Need to Store and Retrieve topoJSON map tiles

Apart from needing to minimise the amount of time needed to retrieve map tiles, the RIF also needs to minimise the amount of map data it sends back to the web client for any given map tile request. A major part of the ETL's support for processing map data is dedicated to simplifying large volumes of map data.

Again, we will work backwards from the web client to better understand why we need to have an efficient way of representing map tile data. When a web client fetches data for a map tile, it will render the information as a piece of JavaScript that is written using JSON. JSON stands for JavaScript Object Notation, and it is used as a syntax for storing and exchanging data. For many of the methods in the RIF's web services, what is returned to the web client are results expressed as JSON. For example, in the scientist's study submission tools, the choices

used to populate drop-down menu items are returned to the web application in a JSON format. Likewise, the parts of the interactive map data also use the same format.

The JSON notation provides the means for organising geographic data, but it has no facility for supporting specific geographic constructs. Its main power lies in the flexibility with which developers can express collections of name-value pairs and ordered lists of values. But it does not give special meaning to geometric concepts.

GeoJSON uses the JSON notation, but also supports geometric types such as `Point`, `LineString`, `Polygon`, `MultiplePoint`, `MultiLineString` and `MultiPolygon`. The added support means that when map data are returned using GeoJSON, client-side map libraries can make use of the added semantics to better perform drawing operations. GeoJSON support geometric but not geographical concepts.

topoJSON is an extension of GeoJSON that includes support for topology. The format is designed to consider issues that may be acceptable in the context of just drawing shapes on a screen, but is not acceptable in the context of drawing geographical maps. For example, if two geographical areas are next to each other, then for part of their boundaries they should share exactly the same set of points. In a geometric use case, it may be acceptable for two polygons to slightly overlap with one another but it doesn't make sense if they are map areas.

topoJSON expresses features using a unique collection of line segments (called arcs) that may be referenced by multiple polygons. Its use of non-redundant line segments and its more efficient way of encoding coordinates means that topoJSON files are much smaller than GeoJSON files that would describe the same area. According to the description of topoJSON on GitHub (https://github.com/mbostock/topojson/), topoJSON files can be up to 80% smaller than GeoJSON files. It is this economy of space that makes its use appealing in the RIF.

topoJSON comes with its drawbacks as well. It is a more complex format than GeoJSON, and there are far fewer code libraries that can directly process the former. In some front-end libraries such as D3, topoJSON will need to be converted back to GeoJSON, just because that is the format that D3 features can understand. However, the main overriding goal in considering formats for storing map data is to minimise the amount of data that are returned for each tile.

> **ETL-60**: *The ETL will produce map tiles that use the topoJSON notation for describing map data.*

Some formats for map data have native support in the database. For example, PostgreSQL can manage JSON data in table columns that are of type `json` or `jsonb`. Using these data types, PostgreSQL can make sure that stored JSON text data complies with JSON rules. PostgreSQL also has a PostGIS extension which can support operations for geospatial databases. The extension has features that can manage data in GeoJSON format.

However, there is not uniform support for JSON in PostgreSQL and SQL Server. SQL Server 2016 has no JSON-specific data types[1]. Currently, it is unclear whether SQL Server will in future provide matching support for PostGIS's use of GeoJSON.

Our policy with porting is that if one but not both of PostgreSQL and SQL Server supports a feature, we will try not to use it. Instead, we will rely on more basic database features that would be equally supported.

JSON and GeoJSON data types are not supported for both databases, and topoJSON is not supported by either. However, both PostgreSQL and SQL Server support Well Known Text (WKT), which can be used to store a 'blob' of text data. We decided that we would use this format to store the text data. Although the database will attach no meaning to a WKT value, when Leaflet receives the text blob from the database, it will know how to interpret it.

ETL-61: *The ETL will express map tiles using the topoJSON file format but store it in the RIF production database using the Well Known Text (WKT) data type.*

We can now summarise how the map tiles be stored in the RIF production database so that they can be most efficiently retrieved as tiled maps by Leaflet. Given a map data set, the ETL will generate 256x256 pixel map tiles for GSC zoom levels from 0 to 11 inclusively. The map tiles will all use a web Mercator projection. The map data will be expressed using a topoJSON notation but it will be stored as a blob field in the database using the Well Known Text (WKT) data type that is supported by both PostgreSQL and SQL Server.

The design decisions we've made establish the requirements the ETL must meet in order to satisfy conditions of the map target area of the RIF production database. Our design discussion now turns to more closely examining the way shape files are imported into the ETL and turned into map tiles.

4.5.3 Extraction from Shape Files

In the ETL, the Extraction activity begins by combining the map data that are contained in multiple shape files, with each shape file representing a different geographic resolution for the same area. The goal of the extraction activity is to extract data from the zipped, proprietary ESRI shapefile format and produce a collection of GeoJSON features.

As an example of how data from multiple shapefiles would be combined, one shapefile may describe the national boundaries of England, another may show the same area split into regions, and others may divide the area into smaller administrative areas such as districts, wards, output area and super output areas. The data would be initially stored in the shapefile format, but at the end of the extraction activity the result would be a collection of GeoJSON features.

Although data managers can vary the number of shapefiles they use, the RIF requires that at least two geographical resolutions are defined:

ETL-62: *A geography must comprise a minimum of two shapefiles, each having a different geographical resolution.*

Within a shape file, the information about any one geometric features is spread over multiple constituent files that share the same base name. Although a shape file can have more than 10 constituent files, the following ones are relevant to the map data workflow:

- `.PRJ`: describes projection information.
- `.SHP`: contains the geospatial data.
- `.DBF`: contains thematic data attributes for each geometric feature
- `.SHP.EA.ISO.XML`: an optional file that contains important meta-data about the shape file

ETL-63: *The ETL will only consider importing shapefiles that have a *.prj, *.shp and *.dbf components.*

Suppose a geometric feature is the outline for an administrative district in the UK. Information about how that small part of the Earth's surface should be made into a two-dimensional map will be stored in the PRJ file.

The SHP file would contain the polygons that would describe the district. In most cases, a feature will be represented by a single polygon, but geographic anomalies such as islands and lakes sometimes mean that an area is represented by more than one.

Attributes for the district such as covariates or demographic classifications might appear as fields in the attributes table stored in the DBF file. The attribute table will have two fields which are especially important to the RIF: an area identifier and a name field. The area identifier field is the column that describe codes that are used to make geometric features identifiable by machines. The name field contains the name of a geometric feature as it would make sense to a human user.

These two fields are important for when the RIF links data sets together at different levels of geographical resolution. The area identifier for a given geometric feature will later be used in the web application actions. For example, clicking on a map to initiate a zoom action will cause the web client to request more feature data from the RIF web services; the area identifier will be the basis of the query to retrieve more information. The name field will allow users to have a more specific understanding about what an area they're looking at represents.

The ETL can extract the names of available fields from the `*.dbf` file. However, it still requires the RIF manager to decide which of the choices should be used in the two fields. Sometimes the names of the fields are not intuitive and it isn't obvious what field should be used. But if the shapefile has a `.SHP.EA.ISO.XML` file, then that file will contain human-readable descriptions for the fields. RIF Managers can then make their decisions based on descriptions that correspond with each field.

ETL-64: *The ETL will provide a feature that previews the contents of the attribute table stored in a shapefile's `*.dbf` file. Based on their inspection of the previewed rows, data managers will decide which field corresponds to an area identifier field and which field corresponds to a name field.*

ETL-65: *If a shapefile contains an `*.shp.ea.iso.xml` file, then the ETL will extract human-readable descriptions of fields that appear in the DBF file. The descriptions can help RIF managers decide which field names correspond to the area identifier and name fields.*

Apart from the area identifier and name field definitions that are derived from the `*.dbf` file, the other important field needed to process the shapefiles is the Spatial Reference System Identifier (SRID).

The SRID corresponds to a spatial reference system based on the specific ellipsoid used for either flat-earth mapping or round-earth mapping. It is used by geospatial database operations to help calculate results such as the area, distance or length involving coordinates for a map. Both PostgreSQL and SQL Server support facilities for understanding what SRIDs mean for geometric calculations. However, the ETL needs to be able to determine what SRID matches the `*.prj` files found in the set of shape files that make up a geography. One of the ETL validation checks is to make sure that whether a shape file describes a district, a region or some other geographical resolution, that all of the shape files use the same projection:

ETL-66: *All shape files for a given geography must use the same projection system.*

The `*.prj` file of a shapefile contains a project name that will correspond to some SRID. In many cases, there will be a one to one correspondence, but there are cases where the same projection could result in multiple SRIDs (e.g.: as is the case for the UK and Eire).

There are various lookup services such as http://prj2epsg.org/search where you can paste in the contents of a `*.prj` file, which is formatted as Well Known Text, and the service returns the SRID number.

> **ETL-67**: *The ETL will use some lookup facility to determine which SRID corresponds to the projection description that is used in the shapefiles used to create a geography.*

The data from a `*.shp` files is combined iteratively for each geometric feature so that each one is associated with union of all polygons that refer to the same area identifier. As part of this process, each polygon definition will be cleaned so that its start and end points are the same. The result of the extract activity will be a collection of unique area identifiers, each of which is associated with one or more well-formed polygons that can be represented using the GeoJSON notation.

4.5.4 Transforming GeoJSON

The transformation process combines collection of GeoJSON features expressed at different geographical resolutions and creates a collection of topoJSON features that are stored in a geospatial database. It begins by ordering the collections of GeoJSON features based on the number of area identifiers. The lower the total number of identifiers, the broader the geographical resolution of the collection. Once they have been ordered, the features are geometrically intersected to create a RIF database table that shows the same area identifier expressed at different resolutions. For example, the table could describe a ward as being within a district within a region within a nation.

> **ETL-68**: *The ETL order the collections of GeoJSON features from lowest to highest geographical resolution. It will then intersect features from all the resolutions to create tables that place an area identifier within resolutions from highest to lowest.*

4.5.4.1 TopoJSON Conversion

As features from different levels are combined, the line segments they share are identified and represented as topoJSON arcs. If the level of precision in the points of features is high, then they may not appear to overlap enough to conclude they represent a shared boundary. The ETL uses a process called pre-quantisation to reduce the precision of points so that shared arcs become more obvious. Pre-quantisation specifies the maximum magnitude that the minimum X and Y coordinates in a map can differ from the largest.

A typical example of pre-quantisation would be using a threshold of $1x10^6$ for the United States at zoomlevel 11. This setting uses a 1:250,000 scale, and each pixel on the screen is 76m on the ground. Given the threshold, the smallest X or Y co-ordinate must be no more than 1 millionth of the largest.

> **ETL-69**: *In order to more easily identify line segments that may be shared by multiple features, a process of pre-quantisation is used to reduce the precision of points and to make it more likely that they will overlap. The quantisation process will be automated given knowledge of the bounding area of the map.*

4.5.4.2 Simplification of topoJSON

So far, we have applied pre-quantisation to GeoJSON features so that the points are most likely to neatly overlap and thus be identified as part of line segments that can be shared. When an arc can be shared to show the boundary between two features, then two separate arcs can be replaced by one. In this way, topoJSON can help reduce the number of line segments that can represent geometric features that border each other.

But even when redundant arcs are eliminated, the resulting topoJSON map can still contain a large number of points. The next step in transformation is to determine the minimum number of points that are needed to represent map boundaries for a given resolution. There are two common algorithms for doing this task: Douglas-Peucker and Visvalingham.

Mike Bostock provides an excellent discussion about them (See: https://bost.ocks.org/mike/simplify/). According to Bostock's analysis, Visvalingham's algorithm can eliminate up to 95% of the points while still retaining sufficient detail for visualisation. Through our own investigations we have determined that for US census map data, elimination of 25% of the points is visually acceptable and 75% elimination results in over-simplification at lower geographical resolutions.

> ETL-70: *The ETL will rely on* Visvalingham's algorithm to minimise the number of points that are needed to support visualising maps at different resolutions.

Applying Visvalingham's algorithm can help remove the problems of slivers which appear when technologies such as Leaflet attempt to visualise multiple layers of map data at once. Sometimes, when maps having different geographical resolutions but different scales are layered, gaps can appear between features or between layers. When they all use the same scale, the algorithm should produce a map that has no slivers. Although Leaflet can be prone to showing sliver problems, the RIF only returns tiles at one geographical resolution at a time. Therefore, although slivers may exist, the users won't see them.

4.5.5 Decision to have the ETL use a shapefile transformation service

After a lot of investigation, we concluded that the only practical way we could implement many of the steps for processing shape files was to rely on facilities provided in the node.js project.

Node.js was chosen because:

- It is fast, and can support multi gigabyte input data and is only limited by the memory in the machine. Node has an internal limit of 256MB (-1 byte!) for strings; this does need to be carefully coded around to avoid string to JSON and JSON to string conversions.
- It supports the following through reputable third party libraries:
    - Shapefile reading, Visvalingham's simplification and topoJSON conversion;
    - Conversion to and from well known text;
    - Projection conversion to WGS84;
    - Zip file read and write (current the new async version is in test);
    - XML parse and create to and from JSON;
    - A full range of geospatial processing (via turf). This removes the need for a database to pre-process geospatial data (e.g. area calculations, creating multi polygons from polygons (to make the area_id key field unique)

The only major area not supported is a JSON database; without indexing some operations (e.g. tile creation, adjacency analysis) are computationally impracticable; so a geospatially enabled relational database is required.

The disadvantage of Node.js is that the database connectors require native code for performance and this needs a C compiler. Internally Node.js uses Python for building add on modules. This creates a complex tool chain on Windows where neither Python nor a C compiler are available by default. The Microsoft SQL Server database connector is particularly hard to compile in; despite Microsoft being a platinum corporate member of the Node.js foundation.

ETL-71: *The ETL will delegate the process of simplifying shapefiles to a node.js service. The tool will submit collections of shapefiles to a web service that hides the installation issues associated with node.js.*

ETL-72: *As part of a testing feature, the topoJSON conversion service will come with its own test web page for visually verifying that maps have been created correctly.*

