BTO 2018.071 | June 2018

BTO report

Reusable Knowledgebases for Hydroinformatics

KWR Watercycle Research Institute

BTO

Reusable Knowledgebases for Hydroinformatics

BTO 2018.071 | June 2018

Project number
400695-047

Project manager
Rosa Sjerps

Client
BTO DO

Quality Assurance
Christos Makropoulos

Author(s)
Mark Morley

Sent to
This report is distributed to BTO-participants. The report is not public.

Year of publishing
2018

More information
Dr. Mark Morley
T 030 6069506
E **mark.morley@kwrwater.nl**

Postbus 1072
3430 BB Nieuwegein
The Netherlands

T +31 (0)30 60 69 511
F +31 (0)30 60 61 165
E info@kwrwater.nl
I www.kwrwater.nl

Keywords
reusable knowlegde base

KWR Watercycle Research Institute

Summary

Knowledgebases vary significantly in complexity. At their most simple they can be considered digital reference documents which might be sensibly represented as printed files or tables but which benefit from the ability of a computer application to guide the user through the data. More sophisticated examples may direct the user to filter the knowledgebase, to "drill-down" through it and to add additional support through analysis such as ranking of alternates.

Past BTO projects have produced a variety of knowledgebase-related outputs. This report investigates options for reducing the development time, complexity and expert developer knowledge required for the creation of new knowledgebases by exploiting synergies between the basic knowledgebase forms.

The report examines the use of the inherent facilities of Database Management Systems to devolve data management as far as possible to be independent of the application. The use of metadatabases to describe a knowledgebase and the use of templatized user interfaces to short-circuit the development of a rich user interface for a knowledgebase are further examined.

Nomenclature

(R)DBMS ..... (Relational) Database Management System – a software system used to manage the implementation and querying of databases. Of these, relational databases are the most common type in which a database comprises a number of *tables* whose *records* are related to each other using *keys*.

Field ..... An individual data element in a database (e.g. “name”, “customer number”) usually with a predefined data type. Fields are aggregated into *records* to associate related information.

Key ..... A *field* that is used to relate one data *record* to one or more others in different *tables*.

Primary Key ..... A *key*, comprised of one or more *fields*, that uniquely identifies a *record* within a table.

Query..... A mechanism for extracting data from a database that establishes relationships between tables and sets criteria for returning a set of results.

Record ..... A collection of related fields that represent a single data item in a database, e.g. “invoice”, “customer”.

SQL ..... Structured Query Language – the most widely-used language for interrogating relational databases using “queries”. SQL is an “English-like” language which allows the composition of complex queries allowing for the dynamic creation of relationships between tables and

Table ..... The basic element of a relational database that acts as a container for related records, such as “invoices”, “customers”.

Contents

1	**Introduction**	**4**
1.1	Motivation	4
1.2	Target	4
2	**Database Management for Knowledgebases**	**5**
2.1	Introduction	5
2.2	File-Based vs Client-Server databases	5
2.3	Devolving Data Management	5
2.4	Spatial Extensions	7
2.5	Lock-in	7
3	**User Interface**	**8**
3.1	Overview	8
3.2	Example Applications	9
4	**Reuse**	**11**
4.1	Metadatabase and Dynamic User Interface	11
4.2	Templatized User Interface	12
5	**Conclusions & Recommendations**	**14**

1 Introduction

1.1 Motivation

In this BTO contract period, a number of “knowledgebase type” applications have been required to be developed by KWR for a broad cross-section of thematic areas. Each Knowledgebase has required separate investment in the software development required to bring it to delivery. Consequently, an investigation was carried out to determine to what extent it was possible to develop core Knowledgebase functionality which could be reused, as necessary, to minimize duplication of effort and to improve maintainability. To this end, this investigation has looked at adapting a common database structure for the Knowledgebase and/or using a redeployable user interface configuration to reduce the costs associated with one of the more time-consuming elements of the software development.

1.2 Target

Knowledgebases naturally have many target users and this has been reflected in the diversity of Knowledgebase implementations that have been undertaken in the past. The principle deliverables have tended to either be standalone Microsoft Windows applications or web applications that deliver data to the end user via a web browser client with the Knowledgebase application itself running on a web server. For the purposes of this report, the latter configuration will be considered as this arrangement can be adapted to deliver local applications without great difficulty, whereas it is more involved to restructure a native Windows application as a web application.

2 Database Management for Knowledgebases

2.1 Introduction

As data repositories, the selection of an appropriate database design is a key consideration for the development of an effective Knowledgebase. Existing Knowledgebases developed by KWR have not been constrained by guidelines on database usage and this has resulted in a plethora of database systems being adopted for each of these. Existing applications have used a mixture of file-based or client-server database systems.

2.2 File-Based vs Client-Server databases

Many Knowledgebases have adopted file-based database systems because of their ease of use. Examples of these approaches include SQLite or the use of Microsoft Access Database files. These are easy to embed in Knowledgebase applications because they do not require the installation or configuration of a heavyweight database service. This is a considerable attraction with the developer, instead, needing only supply the SQLite access software with the application or, in the case of Access, the necessary software to manage the database is included in Microsoft Windows.

The principal downside of file-based databases is, however, relative inflexibility and lack of support for a number of powerful database management tools which are standard amongst client-server databases – many of which are well suited to reducing the workload necessary for developing a Knowledgebase.

Many client-server databases are available and those employed in Knowledgebase applications have tended to be those which use free-to-use software such as MySQL and PostgreSQL rather than commercial implementations such as Oracle and Microsoft SQL Server.

Configuration of client-server database systems may be daunting as these are very powerful, scalable systems with a myriad of performance-related options that can be used to tweak the behaviour of the system for a particular application. At their most basic, however, they can be pared down to a simple system, suitable even for running on a standalone machine, that is well suited for the development of a Knowledgebase which can be deployed as either as a internet-based application or for local machines.

2.3 Devolving Data Management

One of the complexities apparent in Knowledgebase applications that permit input from the user is that a substantial constituent of the software development is associated with validating data input and managing the behaviour of related data tables to ensure the integrity of the Knowledgebase.

If using a client-server database as the backbone of the Knowledgebase, large parts of the implementation may, and as far as possible, *should* be devolved to the database management system (DBMS) itself. It is possible to greatly simplify the implementation of Knowledgebase applications by incorporating as much of the business logic of the application in the database itself.

In this fashion it is possible to anticipate many of the conditions that need to be accommodated when employing a database without having to specifically add the software code to validate the database's state for each action. This leaves the user interface code much more straightforward to implement as the user interface developer does not need to consider the state of the database to the same extent. Whilst these concepts are straightforward, there is a tendency to overlook their use when designing and implementing a knowledgebase.

DBMS and specifically *Relational* DBMS implement a number of functional concepts which are useful to Knowledgebase applications:

2.3.1 Referential Integrity

Within an RDBMS there are several mechanisms that can optionally be employed to ensure that the references between records in the database remain valid at all times.

Ordinarily, relational databases are designed so that each record has a *primary key*. The primary key field in a database is that (or those, by using a combination of fields) that can be used to uniquely identify a record in the table. For example, an "Invoice" table might use an Invoice number as a primary key to identify each invoice. Foreign Keys are used to represent dependent relationships between different tables. For example, an invoice table might include a foreign key field to link to the primary key of a customer table to associate the invoice with an individual water customer. The removal or changing of a customer record may inadvertently "orphan" an invoice, losing its link to the relevant customer. Use of the foreign key concept prevents this link being lost. The database can be configured to prevent the customer record from being removed or changed while it continues to have dependent invoice records.

2.3.2 Cascading Changes

Preventing the deletion of related records is a rather blunt tool and requires some involvement on the part of the user interface developer to report when a deletion or change has been refused. When deleting or changing records in the database it is sometimes necessary to ensure that records in related tables are also deleted automatically or updated accordingly.

It is possible to enforce within the database relationships between entities which can only be deleted or changed under certain circumstances. The most common integrity check is ensuring that records with a particular key value which may be referred to by other records in other tables may not be deleted while those records still exist. The example above preventing deletion of a Customer record when it still has related Invoices ensures that any Invoice will always have a valid Customer to relate to.

An alternative approach, which maintains the validity of the database is to *automatically* cascade the deletion of the customer so that all of the related Invoices are also removed at the same time without any further database queries being required. Judicious use of this capability means that many, complex relationships within the database can be managed automatically without the Knowledgebase developer having to repeated test the state of related tables with queries.

2.3.3 Transactions

Transactions are employed to ensure that the database remains in a coherent state at all times when undertaking complex changes to the data tables. When adding records to multiple related tables, it is possible that for some reason the insertion of a single record

fails. Transactions are used to identify groups of actions on the database that must all succeed in order for the database to be in a coherent state. In the event of a failure, the changes to the database in the transaction are "rolled-back" to return the database to its original state.

2.3.4 Stored Procedures

RDBMS use Stored Procedures to execute predefined SQL queries on the database. These are useful to remove functionality out of the Knowledgebase code itself and into the database. This means that to a greater extent, the behaviour of the Knowledgebase can be encapsulated in the database providing a measure of isolation from the application target. In this way a knowledgebase could be "ported" from a local application to a web application with the minimum amount of coding on the new target platform.

2.3.5 Triggers

Triggers can be used within the database to effect additional changes, i.e. calling Stored Procedures, when a certain condition occurs. They are normally connected to behaviours in tables such as adding, modifying or deleting a record. These are useful in a Knowledgebase for automating, for example, the creation of several new dependent records when a new entry is created in a table.

2.3.6 Views

Predefined queries called Views can be used to perform common aggregations across the database tables which then appear to the Knowledgebase developer as a table in their own right. As with the Stored Procedures, this minimizes the amount of SQL code that has to be kept in the Knowledgebase application.

2.4 Spatial Extensions

A key consideration for many of the Knowledgebase applications that may be developed by KWR in future is the availability of spatial extensions to SQL and to represent geographic entities in the data tables. This would allow SQL queries such as "Select all customers within 100m of pipe burst" to be executed. These extensions are only available with client-server databases or with specific GIS-based file database implementations.

2.5 Lock-in

While SQL is a well-defined standard, many RDBMS implementations employ custom extensions to the language specification to provide easier access to advanced functionality. Care should be taken to avoid implementing Stored Procedures, Views or spatial queries that are dependent on these custom extensions which may be difficult, or impossible, to port to a different RDBMS.

3 User Interface

3.1 Overview

At its most simple, a knowledgebase can be considered as a data repository to which filters can be applied to “drill-down” to the required level of detail.

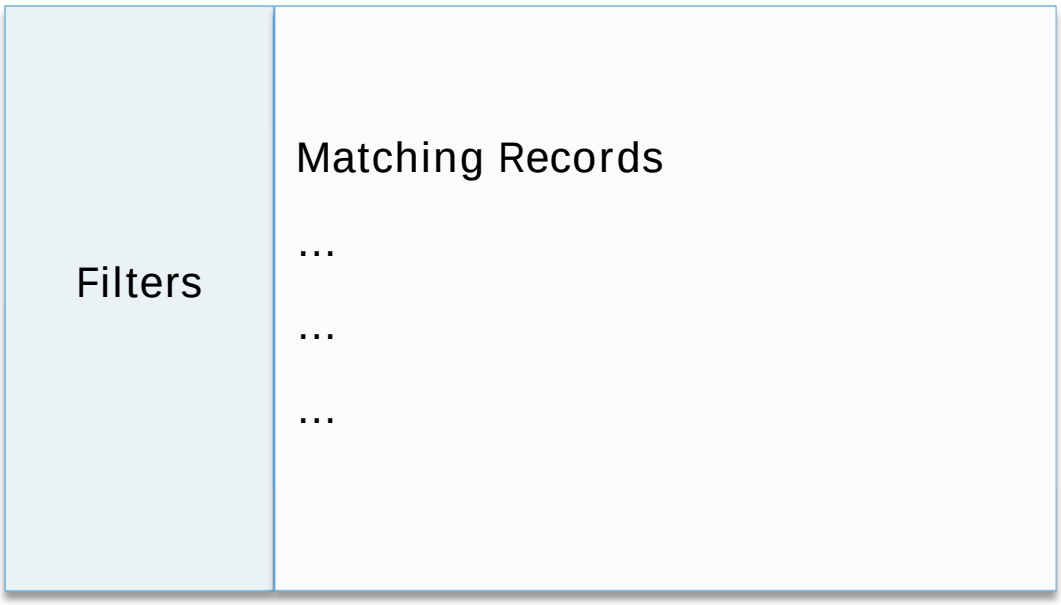

FIGURE 1 - BASIC USER INTERFACE LAYOUT FOR A KNOWLEDGEBASE WITH FILTER + MATCHING RECORDS

Figure 1 represents the simplest form of interface that can be applied to such a Knowledgebase. On the left-hand side are a set of filters that restrict the scope of the matching records that are shown as results in the main pane of the Knowledgebase application. Depending on the computational expense of applying a query to the database, these filters may either be applied in real-time as the user makes changes to the filter selections or in one go through the use of an “Apply” button in the filter pane.

The arrangement in Figure 1 is appropriate for records from the Knowledgebase that are either simple in structure and can easily be displayed in a tabular format or which are sufficiently complex that they require a secondary window to be displayed to further investigate the data.

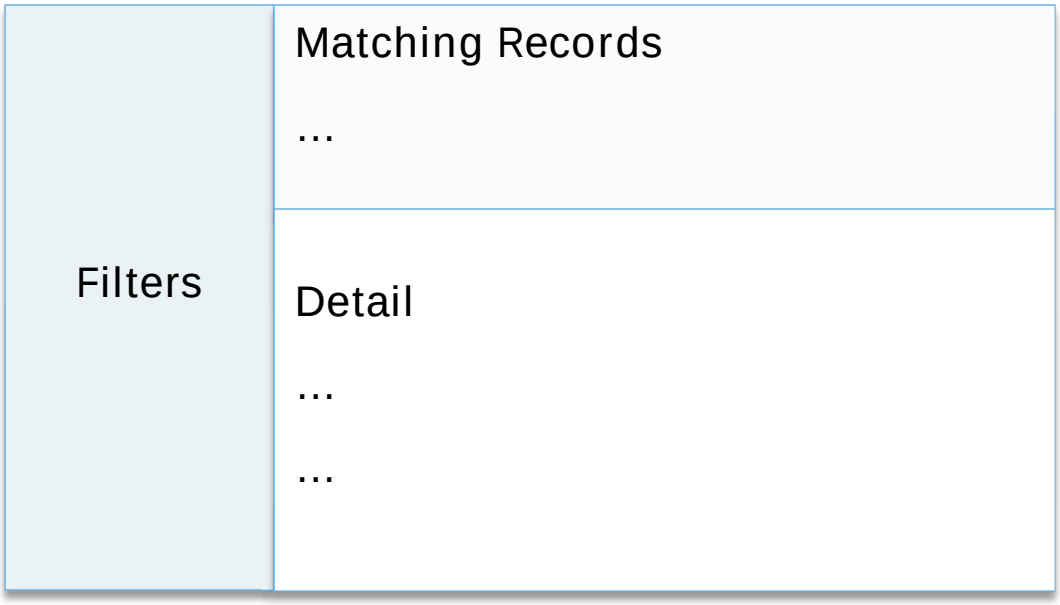

FIGURE 2 – FILTER/MATCHING RECORDS + DETAIL INTERFACE

A more complex interface example is illustrated in Figure 2. Here the filters pane with the dynamic list of matching records is retained and a detail pane is added. The content of the detail pane is updated depending on which (if any) of the matching records has been selected.

The detail pane can be adapted to contain a variety of information including charts and maps as well as textual information.

3.2 Example Applications

A number of Knowledgebases have been created over the duration of this report which have both informed and been influenced by the evolving understanding of database and UI design within KWR. These have taken the form of Web Applications that are underpinned by a client-server relational DBMS, in this case PostgreSQL.

3.2.1 QMRA

The Quantitative Microbial Risk Assessment (QMRA) application, as shown in Figure 3, uses the basic filter/matching records interface from Figure 1. In this case, the matching records retrieved from the database are represented on the chart as an X-Y coordinate plot rather than the more regular tabular output from a database. Additional data on individual data points is available by clicking on them on the chart.

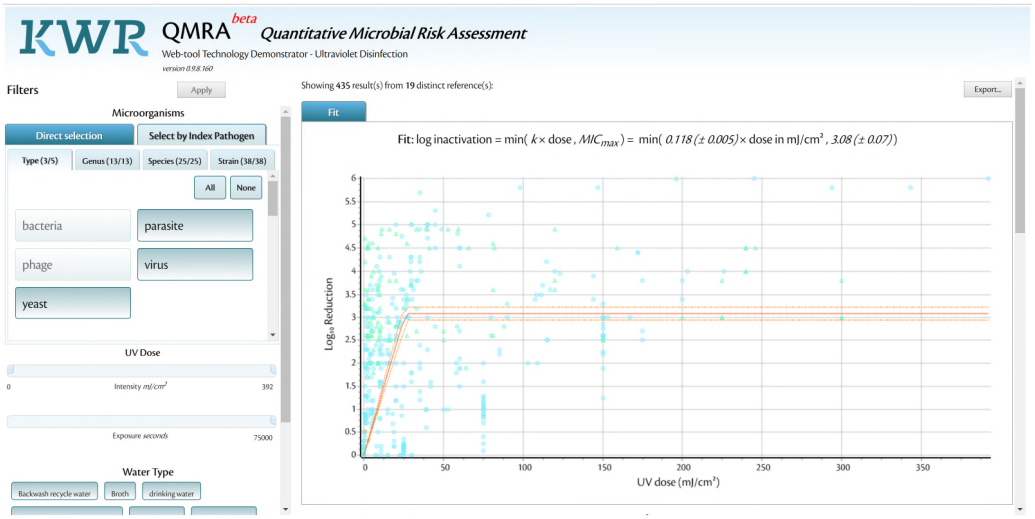

FIGURE 3 - QMRA APPLICATION

3.2.2 PIPE-works

The PIPE-works application (Figure 4) is an example of the user interface design (Figure 2) with the detail records displayed in the main pane of the Knowledgebase application. Here, the user select filters on the left hand side, views matching records in the top right pane and specific information in the detail pane.

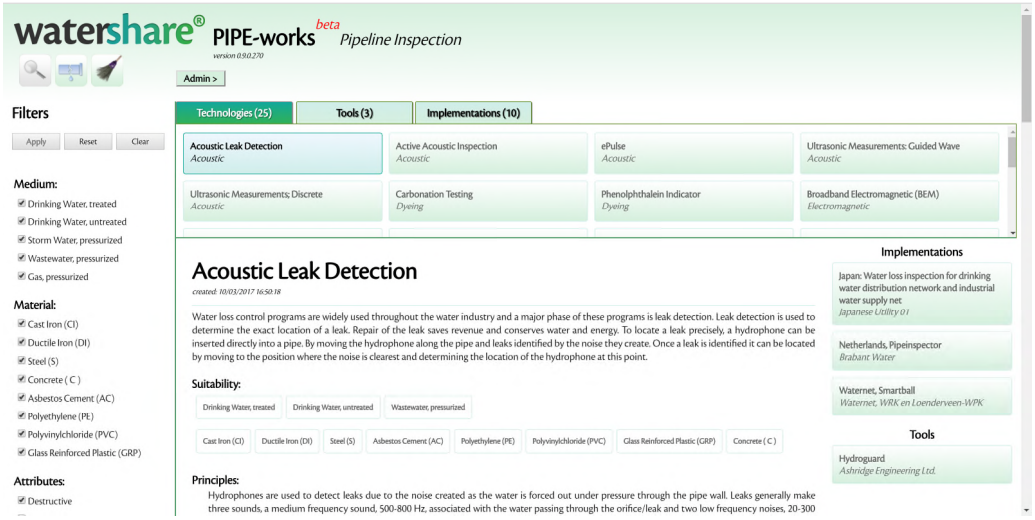

FIGURE 4 – PIPE-WORKS WEB APPLICATION

One departure from the interface is the addition to the right-hand side of the detail pane of links to other related areas of the knowledgebase, allowing the user to drill into related detail in the knowledgebase with ease.

4 Reuse

As part of this investigation the use of a wholly generic knowledgebase was examined to determine whether it was possible, or desirable, to promote reuse of Knowledgebase constituents in this way. An alternative proposition was that new Knowledgebases might be created more easily if there was a template method for producing them.

Accordingly, a specialized database was created to model the metainformation required to create a generic knowledgebase which could be configured entirely by changing the content of the database and which would require no additional code to be developed by the end user. This would be coupled with a dynamic user interface that would be able to use the metainformation to manage the knowledgebase, again without any additional coding.

4.1 Metadatabase and Dynamic User Interface

Figure 5 illustrates the conceptual design of the metadatabase that can be used to define the operation of a basic Knowledgebase without additional user coding.

The bottom section of Figure 5 (highlighted in blue) defines the basic relationships between datasets – effectively SQL views into the database – and the filters that can be used upon them. A filtered dataset creates a new dataset which can itself then be filtered. This permits an arbitrary number of levels of filtering for the datasets although in all of the applications that have been examined, a single level of filtering is all that is required.

The section above (highlighted in orange) defines the relationships between the results extracted from the datasets and indicators that can be used to perform some sort of automated analysis on the data returned, e.g. ranking.

Finally, the top section (highlighted in green) defines specialized views of the data results such as charts and tables that can be referenced by the user interface.

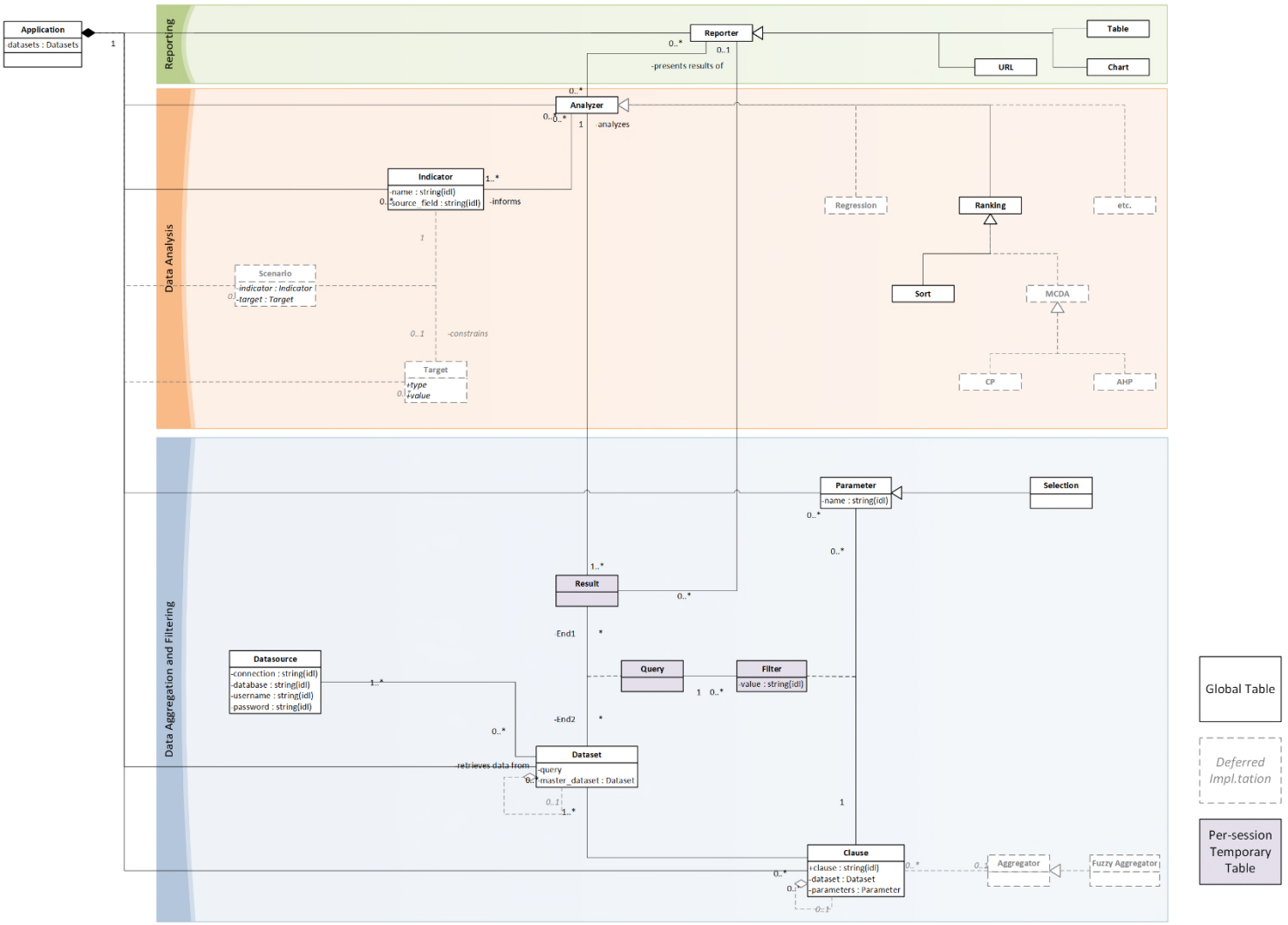

FIGURE 5 - OVERVIEW OF METADATABASE FOR GENERIC KNOWLEDGEBASE

The PIPE-works application (Figure 4) was used as the test-bed for the metadatabase. Modelling the basic relationships within the knowledgebase proved straightforward using the schema presented in Figure 5.

However, the development of a dynamic user interface which could make use of this was much more challenging. While a basic interface was relatively easy, it proved to require disproportionate effort to handle the additional user interface requirements – in PIPE-work's case, for example, the additional links to allow the user to drill-down into related data: a key function of the knowledgebase.

4.2 Templatized User Interface

As an alternative to a fully generic Knowledgebase, a template user interface was derived from the QMRA application (Figure 3). The template allows the developer to reuse all of the basic user interface constituents such as the filtering, matching records and detail panes and to easily repurpose the database queries to the relevant datasets for the new knowledgebase. The template is developed for web applications that are written in PHP, Delphi or C++ and

uses JavaScript on the client-side to manage interactivity with the browser. AJAX is used to maintain communication between the client-side and server-side components.

The application represented in Figure 6, the Portfolio of Adaptation Measure developed for the EU project BINGO is an example of a Knowledgebase application developed rapidly using the template user interface. Since this project was developed as a read-only Knowledgebase there are no stored SQL procedures in the application and the knowledgebase is implemented as a single stored SQL view which is filtered as required.

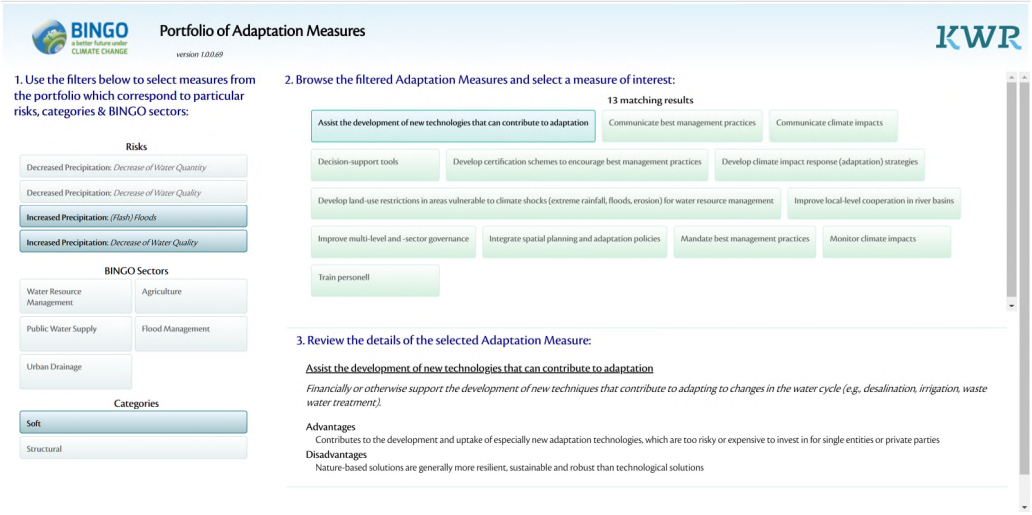

FIGURE 6 - BINGO PORTFOLIO OF ADAPTATION MEASURES

As can be seen, the BINGO application shares the Filter/Matching Records/Detail panes of the other example knowledgebases. In this case, however, the application was derived from a template which contained the basic arrangement for the web application and the connections to the different database tables. These were repurposed with the minimum of effort to create the new Knowledgebase.

Because of the simple user interface for the Portfolio of Adaptation Measures, this application would have lent itself well to implementation using the Generic Knowledgebase. However, the reduction in effort expended using a template as the basis of the new knowledgebase suggests that there would be little overhead in using the template compared to the time taken to define the knowledgebase in the metadatabase. It should be noted that the level of technical capability required of the developer is higher in the case of reusing the template rather than populating the metadatabase.

5 Conclusions & Recommendations

This investigation examined the potential for reusing software and database components to facilitate and enhance the creation of Knowledgebases, which are often a part of Hydroinformatic applications.

Client-Server relational database management systems offer a number of key advantages over their file-based equivalents. The use of Stored Procedures, Triggers and Views can offload a significant proportion, if not all of the database processing functionality to the database itself. This avoids the necessity for large amounts of SQL to be held within the application itself and eases porting the system to other environments. In addition, the use of the referential integrity capabilities of an Relational DBMS can make the development of Knowledgebase interfaces significantly easier by obviating the need for the developer to maintain complete control of the state of related tables in a database. These approaches require a more disciplined approach to the design and implementation of the database but, ultimately, reduce the investment necessary in developing the software to handle database management tasks and to anticipate and correctly respond in the event of malfunctions. It is recommended that future knowledgebase applications are implemented using a Relational DBMS that uses a client-server rather than file-based model in order that more sophisticated facilities are available to the database designer for constraining the behaviour of the knowledgebase without additional coding.

A standardized structure for a user interface for basic knowledgebases was proposed and implemented on a number of applications. This structure uses the common concept of filters to reduce the data being sought by the user to a set of matching records. Depending on the complexity of the matching records, an interface with a detail pane allows the presentation of additional data on the user-selected matching record. Templates have been produced that couple a JavaScript client-side environment to a back-end which can be implemented in C++, Delphi or PHP. Since the beginning of this project, most web-based applications have been implemented in Java. It is recommended that a similar template should be created for this platform, also, to promote the rapid development of Java-based knowledgebases.

The establishment of a Generic Knowledgebase was limited by the visual richness of the user interface. Disproportionate effort would be needed to extend the metadatabase to the extent that it was able to contend with the complexity of behaviours required by the test applications, such as the incorporation of charts (Figure 3) and more complex “drill-down” behaviour (Figure 4) for extracting further detail from the knowledgebase. It is unlikely that such complexity can be accommodated within the budgetary constraints of a conventional project. The use of a template user interface is, however, recommended as this establishes the behavioural norms for the user interface and basic functionality such as implanting the filter, record and detail panes. The resulting basic user interface can then be customized as required and when coupled to a knowledgebase which is largely implemented using the RDBMS facilities rather than in code, as suggested above, can significantly reduce the development input required to produce a new knowledgebase application.