A Data Warehouse for Storing and Analysing Study Data

Version 3.0. Jan 2025

Paul Watson, Newcastle University, paul.watson@ncl.ac.uk

1 Introduction

This document describes the design of a data warehouse for storing and analysing data collected in healthcare and other studies. The primary aim was to create a general system that enables users to explore and analyse data collected from, or related to, participants in studies. This might include data collected through clinical evaluation forms, data collected from sensors, and features extracted from the analysis of sensor data (e.g., step counts derived from raw accelerometery).

Researchers might wish to slice, dice, visualise, analyse and explore this data in different ways, e.g. all results for one patient, all results for one type of measure in a study, or changes in one participant's measurements over time. Others may wish to build models that can then be used in healthcare applications that make predictions about the participant's future health.

Traditionally, data collected in studies has been stored in a collection of files, often with metadata encoded in the filenames. This makes it difficult, and time consuming, for researchers to explore, interpret and analyse the data. We therefore wished to explore an alternative - exploiting modern database technology to vastly simplify this effort. We have therefore designed and implemented a data warehouse for the storage and analysis of study data. In doing this we have drawn heavily on the best practice for data warehouse design. However, there is more variety in the types of healthcare data to be stored than there is in a typical warehouse, and so we have been forced to deviate from a conventional data warehouse in some aspect of the design.

There are three guiding principles behind the design:

1. The data warehouse must be able to store any type of data collected in a study without modifying the schema. This means that when new types of data are collected in studies (e.g., from a new clinical evaluation form, a new data analysis program, or a new sensor) they can be stored in the warehouse without any changes to its design. This has 3 main advantages: firstly, it enables us to fix and optimise the schema for the tables in which the data is stored; secondly it means that applications and tools (e.g. for analysis and visualisation) built on the warehouse do not have to be updated when new types of data are added; thirdly, a single, multi-tenant database server can support many studies. This reduces the overall costs, the start-up time for a new study, and the overheads of managing the warehouse.
2. Descriptive information about the types of measurement is stored in the warehouse so that tools or humans can interpret the data stored there.
3. The design is optimised for query performance. In several cases, this has led to denormalization (duplication of data) to reduce the need for expensive joins.
4. We must be able to implement a security regime to restrict each user's access to the data collected in studies.

In the rest of this document we describe the warehouse design and how it can be used, using examples drawn from real digital healthcare projects. These examples have all been implemented in a prototype of the warehouse that has been built using an instance of PostgreSQL running in the Microsoft Azure cloud (please contact the author if you want to access this). A Python Client library that simplifies common operations is also available.

2 Types of Data

As was stated in the introduction, a major goal of the design was to be able to store and query any type of data that a user may wish to hold in a Healthcare data warehouse. We have therefore designed the warehouse to be able to store measurements whose values are of the following types:

Table 1. The Types of Values stored in the Data Warehouse

Id	Value Type	Description
0	Integer	
1	Real	
2	Text	
3	Date Time	
4	Boolean	False or True
5	Nominal	The value can be one of a set of categories, e.g., Male, Female, Prefer not to say
6	Ordinal	As nominal, except there is an ordering between the categories, e.g. Every Night, More than once per week, Once per week, Less than once per week
7	BoundedInt	An integer bounded by minimum and maximum values
8	BoundedReal	A real bounded by minimum and maximum values
9	BoundedDateTime	A datetime bounded by minimum and maximum values
10	External	The URI of an external object (e.g., image, file)

Measurements are captured in related groups. For example, the results from a clinical evaluation form would form a group, while accelerometery analytics code may generate a group containing both the average walking speed and a stride length. The warehouse is therefore designed so that measurements can be stored and accessed in groups where that is appropriate.

Through this document we use examples based on 3 different groups of measurements.

2.1 Clinical evaluation form data

This clinical evaluation form (named “Q321”) has the following measurements grouped within it:

Measure	Description	Value Type	Categories (if nominal or ordinal)				
G1	Participant has read PIS	Nominal	Y	N			
G3	Year of Birth	DateTime					
G5	Gender	Nominal	M	F	Prefer not to say		
GC1	Comorbidity: PTT	Nominal	Y	N			
C14.5	KCCQ clinical evaluation form, Item 5	Ordinal	Every Night	3-4 Times per week	1-2 Times per week	Less than once a week	Never over the past 2 weeks
C5	Name of drug	Text					
C5.1	Dosage (mg)	Real					
X1	Biopsy Date	DateTime					

2.2 Measurement Data

We use the following example of the output of an algorithm "GFIT":

Param Id	Description	Value Type
WB1	AWS	Real
WB2	Distance	Real
WB3	Stride Length	Real
WB4	Cadence	Real

3 Database Schema

The schema of the Data Warehouse is shown in Figure 1. In the following sections we describe how each of the tables are used by describing the tasks needed to: i) add new measurement types into the warehouse, ii) add new measurements into the warehouse, iii) query the warehouse.

4 Adding a new Study

Each measurement is associated with a study. The **study** table holds information on the study:

- **id**: an identifier for the study that is unique within the data warehouse
- **studyid**: this text field holds the identifier of the study allocated by those running the study. It does not therefore have to be unique in the data warehouse (two different clinical teams, working for different organisations, may have allocated the same **studyid** to different studies, e.g. "Study 1")

For the running examples, we have chosen to create one study:

*Table 2. Example **study** table entries*

id	studyid
1	Test Data

5 Adding a new Trial

A Study can consist of 0 or more trials. A *trial* consists of a set of measurements taken under the same conditions. For example, within a study we could use trial = 0 for baseline measurements, trial = 1 for a 6-month follow-up, trial = 2 for the 12-month follow-up etc. The columns in the **trial** table are:

- **study**: the id of the study of which the trial is a part
- **trialid**: a unique id for the trial within the study
- **description**: a textual description of the trial

*Table 3 Example **trial** table entries*

study	trialid	description
5	0	Baseline
5	1	6-month follow-up
5	2	12-month follow-up

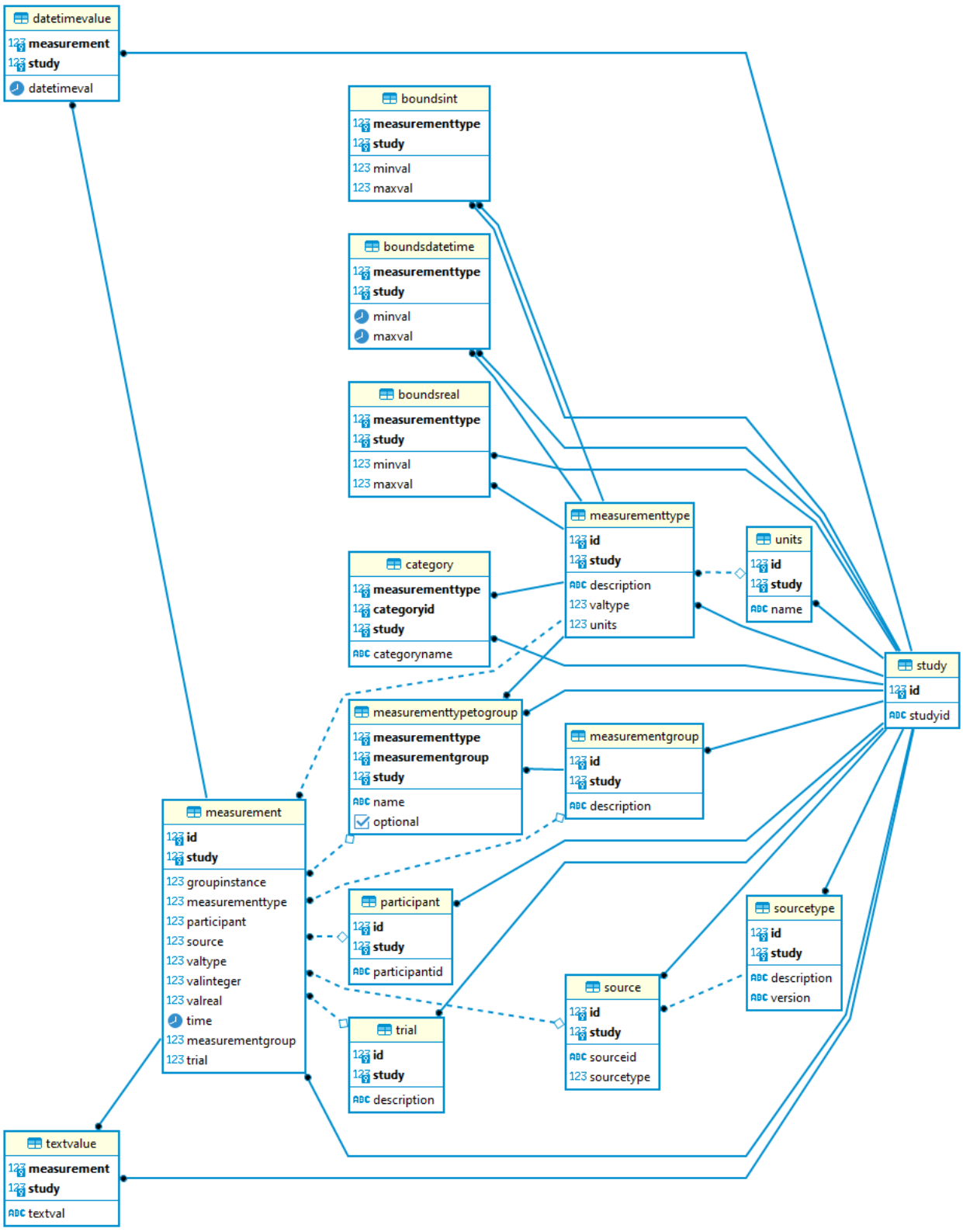

Figure 1. Data Warehouse Schema

6 Adding new Types of Measurements

All measurements are captured in groups of 1 or more measurements. When a new measurement group is to be added to the warehouse, a new entry is first made in the **measurementgroup** table. This has three columns:

- **id**: a unique identifier for that group
- **description**: a textual description of the measurement group
- **study**: the **id** of the relevant entry in the **study** table

For example, the entries in the warehouse for the three groups of measurements described in Section 2 could be:

*Table 4. Example **measurementgroup** table entries*

id	description	study
1	Q321	1
2	GFIT	1
3	Unilever Temperature Sensor	1

The next step is to add a new entry in the **measurementtype** Table for each of the types of measurements found within the new measurement group. This table has the following columns:

- **id:** a unique id for the measurement type
- **study**: the **id** of the relevant entry in the **study** table
- **description**: a textual description of the measurement type
- **units**: (optional) the identifier of the units of measurement (this is a foreign key into the **unit** table described below)
- **valtype**: the type of the value that will be stored for this type of measurement; this is the **id** shown in Table 1

For the three groups described in Section 2, the **measurementtype** Table entries are:

*Table 5 Example **measurementtype** table entries*

id	study	description	units	valtype
1	1	participant has read PIS		4
2	1	Date of Birth		3
3	1	Gender		5
4	1	Comorbidity: PTT		4
5	1	Name of Drug		2
6	1	Dosage	1	1
7	1	Biopsy Date		3
8	1	KCCQ clinical evaluation form, Item 5		6
9	1	AWS	2	1
10	1	Distance	2	1
11	1	Stride Length	2	1
12	1	Cadence	3	1
13	1	Temperature	4	1

The **units** table has the columns:

- **id**: a unique identifier for the unit
- **name**: the name of the unit
- **study**: the **id** of the relevant entry in the **study** table

For the running examples, the entries in the **units** table are:

*Table 6 Example **units** table entries*

id	name	study
1	mg	1
2	metres	1
3	steps per minute	1
4	Centigrade	1

For categorical (ordinal and nominal) types, entries are made in the **category** table for each specific category. This table has the fields:

- **measurementtype**: this is the relevant id field from the **measurementtype** table
- **categoryid**: each different category has a unique id. As will be seen, this is the value that is stored in the warehouse when a measurement is taken. For *ordinal* types (e.g., C14.5 in the running example), the order of the **categoryid** values is significant, whereas it is not for nominal types (whose values can only be compared for equality).
- **categoryname**: a textual description of the category.
- **study**: the **id** of the relevant entry in the **study** table

For the running example, the **category** table holds:

*Table 7 Example **category** table entries*

measurementtype	categoryid	categoryname	study
1	0	Y	1
1	1	N	1
3	0	Male	1
3	1	Female	1
3	2	Prefer not to say	1
4	0	Y	1
4	1	N	1
8	0	Every Night	1
8	1	3-4 times per week	1
8	2	1-2 times per week	1
8	3	Less than once per week	1
8	4	Never over the past 2 weeks	1

For integers with bounds (e.g. a clinical evaluation form asking "How hard is it to walk without an aid – give a number from 1 (easy) to 6 (impossible)") then the **valtype** is set to 7 (see Table 1), and the minimum and maximum bounds are stored in the **boundsint** table, which has the following columns:

- **measurementtype** is a foreign key to the **id** field of the **measurementtype** table
- **minval**: the minimum value
- **maxval**: the maximum value
- **study**: the **id** of the relevant entry in the **study** table

There is an equivalent **boundsreal** table for real numbers: e.g., to set the minimum and maximum values for a percentage as 0.0 and 100.0 respectively. For bounded reals, the **valtype** is set to *8* (see Table 1).

There is also an equivalent **boundsdatetime** table for datetime value: e.g., to set the minimum and maximum values for attendance at a medical clinic. For bounded datetimes, the **valtype** is set to 9 (see Table 1).

The **measurementtypetogroup** table combines the measurement types into groups. This table has the fields:

- **measurementtype** is a foreign key to the **id** field of the **measurementtype** table, **measurementgroup** is a foreign key to the **id** field of the **measurementgroup** table
- **name**: an optional name for the measurement type when it is contained within that measurement group (for example "date of birth" may be collected in many different clinical evaluation forms, but it may have a different question identifier in each – these can be held in the **name** field).
- **optional** should be set to true if the measurementtype is optional in this measurementgroup. Otherwise, it can be set to false or Null.

For the running example, it has the following entries:

Table 8 Example measurementtypetogroup table entries

measurementtype	measurementgroup	name	study	optional
1	1	G1	1	false
2	1	G3	1	false
3	1	G5	1	false
4	1	GC1	1	false
5	1	C5	1	false
6	1	C5.1	1	false
7	1	X1	1	false
8	1	C14.5	1	false
9	2	WB1	1	false
10	2	WB2	1	false
11	2	WB3	1	false
12	2	WB4	1	false
13	3	TS1	1	false

Relating the measurement types and measurement groups in this way (through a many-to-many relationship) enables the same measurement type to be a part of multiple measurement groups. For

example, a participant's temperature may have been collected, at different times, in a set of different measurement groups (including measurements taken when they attended clinics, and measurements taken at home using a smart sensor). This design allows all the measurements in a measurement groups to be extracted together, or individual measurements can be retrieved – for example all temperature measurements, irrespective of the measurement group.

7 Adding new Data Sources into the Warehouse

All data added to the warehouse is collected from a source, e.g., a clinical evaluation form, an algorithm, or a sensor. The warehouse holds information on the source in two tables.

The **sourcetype** Table holds generic information about a type of source. There are the following columns:

- **id**: an id for the type of source that is unique within the warehouse
- **description**: a textual description of the source
- **version**: many types of sources evolve through different versions (especially for software and hardware); the version can be held in this table (as text)
- **study**: the **id** of the relevant entry in the **study** table

The entry for the 3 running examples is:

*Table 9 Example **sourcetype** table entries*

id	description	version	study
1	q321	1	1
2	GFIT	27	1
3	Unilever Temperature Sensor	12	1

In some cases, we might also want to identify a specific source (e.g., the unique identifier of a sensor). For example, this can be useful if there are concerns that a sensor has become inaccurate. This information is held in the **source** table which has the columns:

- **id:** this is an id that is unique within the data warehouse
- **sourceid**: a text identifier for the source. It may, for example be the serial number of a sensor, or another form of id. However, it does not have to be globally unique within the data warehouse: two different types of sensors may number instances of that sensor in the same way (e.g., from 1 upwards).
- **sourcetype**: the id of an entry in the **sourcetype** table
- **study**: the **id** of the relevant entry in the **study** table

For the running example, the entries in the **source** table might be:

*Table 10 Example **source** table entries*

id	sourceid	sourcetype	study
1	1	1	1
2	1	2	1
3	3267	3	1

8 Adding a new Participant

Measurements made in a study can be associated with a participant. The **participant** table holds the following information:

- **id**: a unique id for the participant
- **participantid**: this text field holds the identifier of the participant as allocated by those running the study. It does not therefore have to be unique in the data warehouse (two different clinical teams, working for different organisations, may have allocated the same **participantid** to different patients, e.g., "Patient 27")
- **study**: the **id** of the relevant entry in the **study** table

For the running examples, we might have one participant:

*Table 11 Example **participant** table entry*

id	participantid	study
1	P123456	1

9 Adding new Measurements

Measurements are added one instance of a measurement group at a time. Each measurement within the measurement group instance is stored in a separate row in the **measurement** table.

The **measurement** table has the following columns:

- **id:** an identifier that is unique within the data warehouse
- **time**: the date and time when the measurement was taken
- **measurementgroup**: the **id** of the measurement group within which this measurement was taken
- **groupinstance**: a unique identifier for a set of measurements captured within a group. Each measurement within a group will be stored in a separate row of the **measurement** table, and so this field is used to link together all those related measurements. By convention, we use the **id** of the first row in the **measurement** table as the value of the **groupinstance**. Not all measurement types registered as being within a measurement group (in **measurementtypetogroup**) need to be represented within each measurement group instance – for example, some types may be optional and so will not appear in each instance.
- **measurementtype**: the **id** of the entry for this type of measurement in the **measurementtype** table
- **participant**: the **id** of the relevant entry in the **participant** table. This is set to null for measurements that are not related to a specific participant
- **study**: the **id** of the relevant entry in the **study** table
- **trial:** the **id** of the relevant entry in the **trial** table. This is set to *null* for measurements made outside of a trial.
- **valtype**: the **id** of the type of value, taken from Table 1
- **valinteger**: this field holds the value for integer measurements, as well as Booleans (as 0 and 1), nominals and ordinals (stored as the **categoryid** from the **category** table) - see Table 16 for more information on how values are represented
- **valreal**: holds the value for real number measurements

To balance space efficiency with query performance, the **measurement** table only holds integers and real values: other types of values – datetimes and text - are stored in other tables with a link to the **measurement** table. This enables the selection and aggregation of numeric values without the need to perform expensive join operations. Booleans, nominals, and ordinals are stored as integers in the **measurement** table, enabling them to be selected without the need for joins (see Table 16).

For the running examples, some example measurements are:

*Table 12 Example **measurement** table entries*

id	time	measurement group	group instance	measurement type	participant	study	trial	source	valtype	val integer	val real
1	08/03/2020 14:05	1	1	1	1	1		1	4	1	
2	08/03/2020 14:05	1	1	2	1	1		1	3		
3	08/03/2020 14:05	1	1	3	1	1		1	5	2	
4	08/03/2020 14:05	1	1	4	1	1		1	4	0	
5	08/03/2020 14:05	1	1	5	1	1		1	2		
6	08/03/2020 14:05	1	1	6	1	1		1	1		2.50
7	08/03/2020 14:05	1	1	7	1	1		1	3		
8	08/03/2020 14:05	1	1	8	1	1		1	6	3	
9	11/05/2020 11:03	2	9	9	1	1		2	1		4.30
10	11/05/2020 11:03	2	9	10	1	1		2	1		1.03
11	11/05/2020 11:03	2	9	11	1	1		2	1		22.00
12	11/05/2020 11:03	2	9	12	1	1		2	1		5.30
13	16/06/2020 01:02	3	23	13	1	1		3	1		37.50
14	11/05/2020 13:03	3	45	13	1	1		3	1		36.40
15	11/05/2020 17:05	3	77	13	1	1		3	1		35.80

For those measurements that hold datetime or text values, an entry is made in the **datetimevalue** or **textvalue** tables respectively:

textvalue Table:

Table 13 Example textvalue table entry

measurement	textval	study
5	The patient was confused	1

datetimevalue Table:

Table 14 Example datetimevalue table entries

measurement	datetimeval	study
2	1962-07-24 00:00:00	1
7	2012-09-07 06:10:00	1

In both cases, the **measurement** field holds the **id** of the relevant row in the **measurement** table.

In this example, we omit to show the **trial** column as it would be set to Null for this example. However, the trial and participant columns can be used to add measurements in four different scopes:

- if **trial** is Null and **participant** is Null then the measurement is relevant to the whole of the study. This might, for example be the medical condition affecting all participants in the study.
- If **trial** is Null but the **participant** is not Null then the measurement is relevant to that participant for all trials in the Study. For example, this might be the participant's height.
- If **trial** is not Null but **participant** is Null then the measurement is relevant to all participants in the trial. For example, this might be the condition of the ground, if it was the same for all participants.
- If **trial** is not Null, and **participant** is not Null then the measurement is relevant to that specific participant in that specific trial. For example, this might be the average walking speed of the participant during a trial.

Table 15. Summary of the use of trial and participant fields in a measurement

trial	participant	scope of the measurement
Null	Null	the whole of the study
Null	not Null	the specified participant in all trials within the study
Not Null	Null	all participants in the trial
Not Null	Not Null	the specified participant in the specified trial

Table 16. How values are stored, by valtype

valtype	Description	Where values are stored	Encoding
0	Integer	`measurement.valinteger`	-2147483648 to +2147483647
1	Real	`measurement.valreal`	Double Precision (8 bytes): 15 decimal digits
2	Text	`textvalue.textval`	Up to 500 characters
3	DateTime	`datetimevalue.datetimeval`	date and time (no time zone) from 4713 BC to 294276 AD, with 1 microsecond resolution
4	Boolean	`measurement.valinteger`	False = 0; True =1
5	Nominal	`measurement.valinteger`	The category table maps the integer stored in `valinteger` into a name for the category
6	Ordinal	`measurement.valinteger`	The category table maps the integer stored in

			`valinteger` into a name for the category
7	BoundedInt	`measurement.valinteger`	The boundsint table holds the maximum and minimum values
8	BoundedReal	`measurement.valreal`	The boundsreal table holds the maximum and minimum values
9	BoundedDateTime	`datetimevalue.datetimeval`	The boundsdatetime table holds the maximum and minimum values
10	External	`textvalue.textval`	A URI, up to the length of a Text value.

10 Security

We may wish to restrict the data that users can access. To ensure that a user can only see data from a study that they have been given permission to access, all tables have a study field as part of the primary key. Client functions all include a study id so that the client can limit the data returned to a study that the client is allowed to access.

11 Python Client Library

A Python client library is provided to allow users who can program to retrieve, filter, and analyse data from the warehouse without them needing to understand the structure of the data warehouse and be fluent in SQL. Later, we will build graphical user interfaces to provide this level of access for those who cannot program. The main functions are now described (the SQL behind them is covered in the next Section (12).

The code is made available as a python package `data-warehouse-client` that can be installed from PyPi.

Before using any of the functions in the client library, an instance of the client must be created using the DataWarehouse constructor:

```
dw = data_warehouse.DataWarehouse("db-credentials.json","datawarehouse")
```

The credentials file is not held in the repo for security reasons: it is available from the platform system administrator on request.

11.1 Accessing and Filtering Measurements

```
dw.get_measurements(study, [optional keyword arguments])
```

This function returns all measurements in the data warehouse that meet the optional criteria specified in the keyword arguments. These criteria are:

```
participant,measurement_type,measurement_group,
group_instance, trial,start_time,endTime.
```

The result is a list of measurements. Each measurement is held in a list with the following fields:

```
id, time, study, participant, measurement_type, type_name,
measurement_group, group_instance,trial, val_type, value
```

For example:

```
dw.get_measurements(1, measurement_group=5)
```

returns all the measurements in measurement group 5, in study 1.

Sometimes it is useful to find all measurement of a particular type whose value meets some criteria. This is done with the function:

```
dw.get_measurements_with_value_test(  study,
                                      measurement_type,
                                      criteria
                                      [optional keyword arguments])
```

The optional keyword arguments are:

```
participant,measurement_group,group_instance,
trial,start_time,end_time.
```

The `criteria` parameter is a string holding the condition against which the value in each measurement is compared. For example:

```
dw.get_measurements_with_value_test(3, 155, "> 9")
```

returns all measurements of type `155` in study `3` that are greater than `9`.

N.B. for nominals and ordinals, the integer `category_id` is used, rather than the category name. Similarly, for Booleans, 0 and 1 are used rather than “F” and “T” respectively.

Any condition that is allowable in an SQL `WHERE` clause can be used (e.g., `<`, `>`, `<>`, `=`). The result is in the same format as for `get_measurements`.

11.2 Aggregation over Measurements

```
dw.aggregate_measurements(study, measurement_type, aggregation_type
                          [optional keyword arguments])
```

This function applies an aggregation operator (from Table 17Table 1) to all of the measurements in the data warehouse that are of the specified type, and that meet the optional criteria specified in the keyword arguments. These are:

```
participant, measurement_group,group_instance,
trial,start_time,end_time.
```

For example:

```
dw.aggregate_measurements(3, 155, "avg")
```

returns the average of all measurements of type 155, in study 3.

Table 17. Aggregation Functions

Aggregation	Result
`avg`	the mean of the measurements
`count`	the number of measurements
`max`	the maximum measurement

min	the minimum measurement
sum	the sum of all measurements
stddev_samp	sample standard deviation
stddev_pop	population standard deviation
var_samp	sample variance (square of the sample standard deviation)
var_pop	population variance of the input values (square of the population standard deviation)

11.3 Filtering Measurement Groups

We can use `get_measurements` to retrieve all measurements in a specific measurement group, e.g.:

```
dw.get_measurements(2, measurement_group=15)
```

returns all the measurements in measurement group 15, in study 2.

However, we sometimes want to retrieve all instances of a measurement group in which one or more of the measurements within an instance meet some specified criteria, e.g. all instances of a clinical evaluation form from study 5 that collects data about participants (measurement group 15) where the participant's age is greater than 22 and their body mass is less than 55kgs. For this we use:

```
(headers, insts) = dw.get_measurement_group_instances(study,
          measurement_group, value_test_conditions
                                [optional keyword arguments]))
```

This function returns all instances of the measurement group in the data warehouse that are of the specified type, and that meet the optional criteria specified in the keyword arguments. These are:

```
participant,trial,start_time,end_time.
```

The `value_test_conditions` is a list where each element is takes the following form:

```
(measurement_type,condition)
```

where `condition` is in the same format as in `get_measurements_with_value_test`

The result is a pair: (header, instances) where header is a list of the column names, and instances is a list of instances that meet the criteria. The format of each instance is:

```
group_instance, time of first measurement in the instance,
study, participant, measurement_group, trial,value1,
value2....
```

where value n is the value for the nth measurement in the instance (ordered by measurement type)

e.g. the following call returns the header and instances of measurement group 15 in study 2 where the participant's age (measurement type 151) is greater than 22 and their body mass (measurement type 154) is less than 55kgs:

```
dw.get_measurement_group_instances(2, 15,
```

```
                                        [(151,">22"),(154,"<55.0")])
```

11.4 Generating CSV Files

Sometimes users need the data retrieved from the data warehouse using the above functions to be stored in a CSV file that then can be loaded into another application for further analysis.

This is done using the function:

```
csv_io.export_measurements_as_csv(results,file_name)
```

The results must be in the format produced by `get_measurements` or `get_measurements_with_value_test`

After a header row, each row contains one measurement consisting of the following fields:

```
id,time,study,participant,measurement_type,type_name,
measurement_group, group_instance,trial,val_type,value
```

For example, storing all measurements from study 5 into a CSV file `study5.csv` would be done by:

```
results = dw.get_measurements(5)
csv_io.export_measurements_as_csv(results,'study5.csv')
```

Similarly, we can export measurement group instances as follows:

```
(header, instances) = dw.get_measurement_group_instances(2, 15,
                                          [(151,">22"),(154,"<55.0")])
csv_io.export_measurement_groups_as_csv(header, instances,
                                        'results3.csv')
```

11.5 Exporting all, or a subset of, measurements in a study to CSV Files

Creating a CSV file holding the measurements in each measurement group in a study can be achieved by:

```
from data_warehouse_client import study_summary
study_summary.print_instances_in_a_study_to_csv_files(
      dw, study, report_dir,
      select_participants=False, participants=[],
      filename_prefix='')
```

where:

`report_dir` is the directory to hold the csv files

select_`participants` is an optional boolean. When True all instances are included. When False, only those instances containing participants whose ids are in the `participants` list are included. This enables the measurements for a subset of patients (perhaps those with a specific condition in a healthcare study) to be exported.

`filename_prefix` is an optional string that will be the prefix of the names of all the csv files generated.

11.6 Exploratory Data Analysis

The python pandas-profiling library has been integrated into the data warehouse client and can be used for exploratory data analysis of a whole study, or a part of it. This includes missing values, analysis of each field in the measurements, as well as interaction and correlation results and graphs. One html profile is written for each measurement group:

```
from data_warehouse_client import study_profile
study_profile.profile_all_measurement_groups(
            dw, study, report_dir,
            select_participants=False, participants=[],
            select_trials=False, trials=[],
            hide_trial_column=False,
            filename_prefix='')
```

where:

`report_dir` is the directory to hold the profile html files

`select_participants` is an optional boolean. When True all instances are included. When False, only those instances containing participants whose ids are in the `participants` list are included. This enables the measurements for a subset of patients (perhaps those with a specific condition in a healthcare study) to be profiled.
`select_trials` and `trials` are optional parameters used to restrict the profiles to those measurements that are from the subset of trials specified in the `trials` list.
`hide_trial_column` is an optional parameter used to prevent the trial field of each measurement being included in the profile. It is useful when the trial field is not used in a study.
`filename_prefix` is an optional string that will be the prefix of the names of all the csv files generated.

11.7 Printing

`Measurements can be printed with print_measurements(measurements)`
e.g.

```
measurements = dw.get_measurements(2, measurement_group=15)
print_io = print_measurements(measurements)
```

Instances can be printed with

```
print_measurement_group_instances(header,instances)
```

e.g.

```
(header, instances) = dw.get_measurement_group_instances(5, 20, [])
print_io.print_measurement_group_instances(header, instances)
```

11.8 Generating Plots

The data in the warehouse is largely self-describing: for each measurement we know the type, the name, the units, the range (for integers), and the category names (for nominal and ordinal data). This will enable general tools to be written to analyse and visualise the measurements. One basic

example is plotting the value of a measurement over time. This can be done, for any type of measurement using the function:

```
plot.plot_measurements(dw, results, study,
                       measage_group_id, file_name)
```

e.g.

```
mts = dw.get_measurements(3, measurement_type=155)
plot.plot_measurements(mts, 155, 3, 'example155.png')
```

This automatically titles the plot and labels the y axis.

11.9 Inserting Measurements into the Data Warehouse

To insert all the measurements in a measurement group:

```
dw.insert_measurement_group(
      study, measurement_group,[(measurement_type, val_type,
            value)],
      [optional keyword arguments])
```

the optional keyword arguments are: `time, trial, participant, source`

If time is not specified, then the current time is used.

The function returns the id of the new measurement group instance.

For example, the following stores all 9 measurements in an instance of measurement group 22 (in study 6):

```
dw.insert_measurement_group(6,22,
      [(182,0,58),(183,1,99.94),(184,2,"The quick brown fox"),
       (185,3,'2020-03-08 14:05:06'),(186,4,1),(187,5,1),(188,6,2),
       (189,7,4),(190,8,3.142)],participant=36)
```

11.10 Cohort-based Querying

Sometimes we want to specify a cohort of patients to analyse. This can be done in two stages. Firstly, create a list of the ids of all participants in the cohort.

In the mobilise project, this can be done by specifying the conditions and sites of the patients:

e.g. Create a cohort of all participants in UNEW and USFD with HA or CHF in study 26:

```
cohort = mobilise_cohort_selection.get_mobilise_cohort(
                                      dw, # warehouse handle
                                      26,             # study id
                                      ["UNEW","USFD"], # sites
                                      ["HA","CHF"])    # conditions
```

The cohort can then be used in a variant of the function `get_measurement_group_instances` that takes a cohort instead of a single participant:

```
dw.get_measurement_group_instances_for_cohort(
                       26,        # study id
```

```
14,         # measurement group
cohort,     # participants
[])         # valueTestConditions
```

12 Example Query

Users with a knowledge of SQL can write arbitrary queries to extract data from the warehouse. This is done using the following functions:

```
dw.return_query_result(queryText)
```

This executes the SQL in `queryText` and returns the result as a list of rows (each row is represented as a list holding the value for each column). It is used for `SELECT` queries.

As an example, the query that underpins the `get_measurements` and `getMeasurements_with_value_test` functions in the Python client library is based on the following SQL:

```
SELECT
  measurement.id,
  measurement.time,
  measurement.study,
  measurement.participant,
  measurement.measurementtype,
  measurementtypetogroup.name,
  measurement.measurementgroup,
  measurement.groupinstance,
  measurement.trial,
  measurement.valtype,
  measurement.valinteger,
  measurement.valreal,
  textvalue.textval,
  datetimevalue.datetimeval,
  category.categoryname
FROM measurement
INNER JOIN measurementtype
  ON measurement.measurementtype = measurementtype.id
  AND measurement.study = measurementtype.study
INNER JOIN measurementtypetogroup
  ON measurement.measurementgroup = measurementtypetogroup.measurementgroup
  AND measurement.measurementtype = measurementtypetogroup.measurementtype
  AND measurement.study = measurementtypetogroup.study
LEFT OUTER JOIN textvalue
  ON textvalue.measurement = measurement.id
  AND textvalue.study = measurement.study
LEFT OUTER JOIN datetimevalue
  ON datetimevalue.measurement = measurement.id
  AND datetimevalue.study = measurement.study
LEFT OUTER JOIN category
  ON measurement.valinteger = category.categoryid
  AND measurement.measurementtype = category.measurementtype
  AND measurement.study = category.study
```

The conditions in the `WHERE` clause are automatically generated in the client library depending on the parameters to the query. For example, for:

```
getMeasurements(measurementType=155,study=3)
```

the `WHERE` clause conditions are:

```
measurement.measurementType=155 AND measurement.study=3
```