University of Glasgow | School of Computing Science

Software for Hypertension Research

Alexandrina Pancheva

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 21, 2017

Abstract

Hypertension contributes to millions of deaths worldwide, and currently researchers are trying to address the limitations of the existing treatment. One such project aims to deliver personalised treatment and the center of the research is investigating how ancestry markers and metabolites can contribute to delivering a better treatment. The research project has data collected from 7 different studies covering approximately 8000 patients who have been prescribed 1 or 2 drugs (out of 9 drugs of interest). The data consists of demographic data and blood pressure measurements, taken before and after the treatment. This paper covers two mini-projects, based on the patient studies, briefly described above. The first sub-project focuses on using a third party tool for loading data into a database and extends the already existing schema to match the requirements of the hypertension research project. The second sub-project discusses the design and evaluation of a tool for visualising metabolites in their pathways and provides charts that can prove useful in comparing the metabolite level across different patients.

Acknowledgements

I would like to thank Dr Simon Rogers for providing me with timely feedback, recognising the challenges of working with customers, enhancing the communication with the various stakeholders, and keeping me sane.

I would like to thank Dr Desmond Campbell for providing his expertise, and taking the time to read drafts of user guides and highlighting any issues that required resolution.

Contents

Chapter 1

Introduction

1.1 General Overview

Hypertension or high-blood pressure is a common condition in which the force of blood pushing against the walls of the arteries is high enough that it may cause a heart condition such as atherosclerosis, heart attack, stroke, and others in the long-term. People can have hypertension for years before it gets detected since there aren't any symptoms but the blood vessels and the heart still get damaged [9].

Hypertension, considered as the "biggest contributor to the global burden of disease and to global mortality", contributes to 9.4 million deaths worldwide [1]. Currently, treatment in the UK is stratified according to age and self-defined race. However, this poses significant limitations such as individual-specific response "despite the existing stratification by age and ethnicity", "uncertainty in stratification of ethnic minorities and uncertainty in stratification to combination treatment in most patients"[10].

Metabolites are "products of reactions that occur within cells"[18]; metabolites should be able to enter into subsequent reactions and they have different biological functions. Metabolites can be primary, synthesized by the cell, and secondary,"not required for the primary metabolic processes" [18] but they are still vital, for example antibiotics. A metabolic pathway is the series of chemical reactions within the cell that the metabolite goes through, and it is the basis of metabolite classification [16][18].

1.2 AIM HY and the Project

Ancestry and biological Informative Markers for stratification of Hypertension, AIM HY, MRC/BHF funded, involves 26 investigators over 11 institutions in the UK and US such as the British Heart Foundation, the Medical Research Council (MRC), Mayo Clinic, the University of Cambridge, the University of Glasgow, Queen Mary University of London, UCL, King's College London, the University of Kentucky, the University of Florida, the University of Manchester, and the University of Nottingham.

The AIM HY Consortium recognises the problems surrounding treating hypertension and the 3 main aims of the project include:

- Investigate how ancestry informative markers and metabolites can be used to personalise treatment
- Determine the best treatment plan based on the evidence
- "Determine mechanisms that define response to existing treatment" [10]

The ultimate goal of the project is to deliver personalised treatment, "based on a single blood test"[17] and improve the management of high-blood pressure in the UK [17][10].

Currently, AIM HY have access to clinical trial data sets in the US, over 8 000 subjects in a multi-ethnic cohorts, that have been exposed to different drugs. For each patient across the 7 studies, there's demographic data (age, gender, BMI, ethnicity), measurements (blood pressure, systolic and diastolic, taken before and after the beginning of the treatment) and drug information. Some patients haven't been treated for hypertension before the beginning of the study, while others are under combinational treatment. Some of the studies have multiple blood pressure measurements taken after the beginning of the treatment, while others contain only 2 (pre and post). Additionally, one of the studies is sub-divided into 2 studies (same patients but different drug treatment). For 2 of the studies in addition to the demographics and the measurements, there is metabolomic data outlining the level of each metabolite per patient.

Currently, the data, stored in CSV files (some of the original data files are in other results formats, as the data comes from different studies but the customer has converted those files to CSV format), reside on the Aridhia system, discussed in details in Chapter 2.

The project aims to make the data analysis much easier and simplify some of the tasks that need to be done with regards to getting access to some statistics and visualising items of interest.

1.3 Report Outline

The rest of this report is structured as follows:

- Chapter 2 discusses the different components of the existing system, developed by Aridhia such as Greenplum, the custom data loading tool, the structure of the YAML files used with the data loading tool, etc.
- Chapter 3 focuses on the requirements for the project and discusses the risks such as technology and customer risks.
- Chapter 4 provides an overview of the first aspect of this project, the data loading, discusses the design decisions and the challenges of the work.
- Chapter 5 looks at Django's integration with Greenplum, why this was a significant limitation to the work, what options were available and how the issue was resolved.
- Chapter 6 provides a detailed discussion of the design, implementation and evaluation of the data visualisation aspect of the project.
- Chapter 7 discusses the possible improvements such as adding new features, implementing other visualisation options, testing and refining some software engineering practices.

Chapter 2

Domain Overview

The following section discusses the different components of the system used by Aridhia, a software development company in the domain of biomedical research and healthcare [4], that currently works on the AIM HY project and hosts the confidential patient data. The company offer a platform for visualising and analysing data with primarily relying on R-shiny, a web application framework for R, integrated applications. For handling the large patient data sets for various projects, they use Greenplum which would be discussed further in this section.

2.1 Greenplum

One of the tools used by Aridhia is Greenplum, an "open-source parallel data warehouse" [15]. Greenplum uses massively parallel processing, also known as MPP, to distribute the load and to process a query in parallel [15].

Greenplum is based on PostgreSQL and "it's essentially several PostgreSQL database instances acting together as one cohesive database management system" [15]. Specifically, it is based on PostgreSQL 8.2.15 and aspects such as functionality, configurations, user interaction are meant to be very similar to PostgreSQL. Greenplum consists of:

- Greenplum Master
- Greenplum segments
- Interconnect
- Query optimiser framework

The Greenplum master is addition to being an entry point by authenticating client connections and accepting SQL queries, distributes the work to the Greenplum segments.The master doesn't contain any user data [15].

The Greenplum segments are separate instances and they each store a portion of the data and that's where the query processing is done. When a user defines a table, the data is distributed across the segments. The distribution key that might or might not be specified (when the distribution key is not explicitly specified, then the primary key is chosen to be the distribution key) when the user creates a table is being used to distribute the data across the segments.Segments run on segment hosts which execute between 2 to 8 Greenplum segments. Typically the segments have the same configuration which is a prerequisite for distributing the work across a large number of equally capable segments [15].

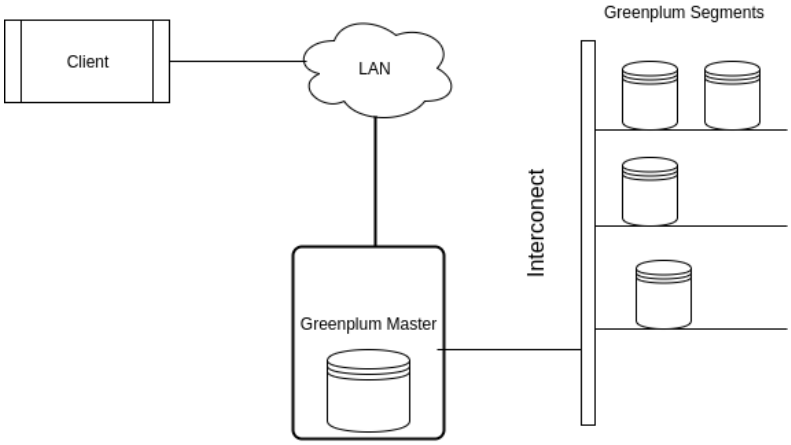


Figure 2.1: Greenplum Architecture [15]

Another element of the Greenplum architecture is the Greenplum interconnect responsible for the communication between segments. It uses UDP but on top of it, packet verification is added [15].

When discussing Greenplum is vital to note that foreign keys can be defined but referential integrity can't be enforced since the data is distributed across segments and checks can't be performed [13]

2.2 Database Schema

Since Aridhia have worked on other health projects, they've created a database schema that can capture information about patients, staff, measures, treatment, and source of the data. Figure 2.2 below presents a general overview of the database schema used by Aridhia. However, for clarity the tables ha_updates and ha_concepts are not includes as they are referenced by all tables.

The table ha_concepts is essentially a lookup table for defining terms appearing in other tables [5]. For example, when defining events for patients (an event for patient can be considered: visit before treatment and visit after treatment), those events can be defined in the ha_concepts table.

The ha_updates provides a useful way of tracking changes that have been made to a table. Additionally, ha_updates is linked to a source table that keeps track of the CSV files used to populate the database.

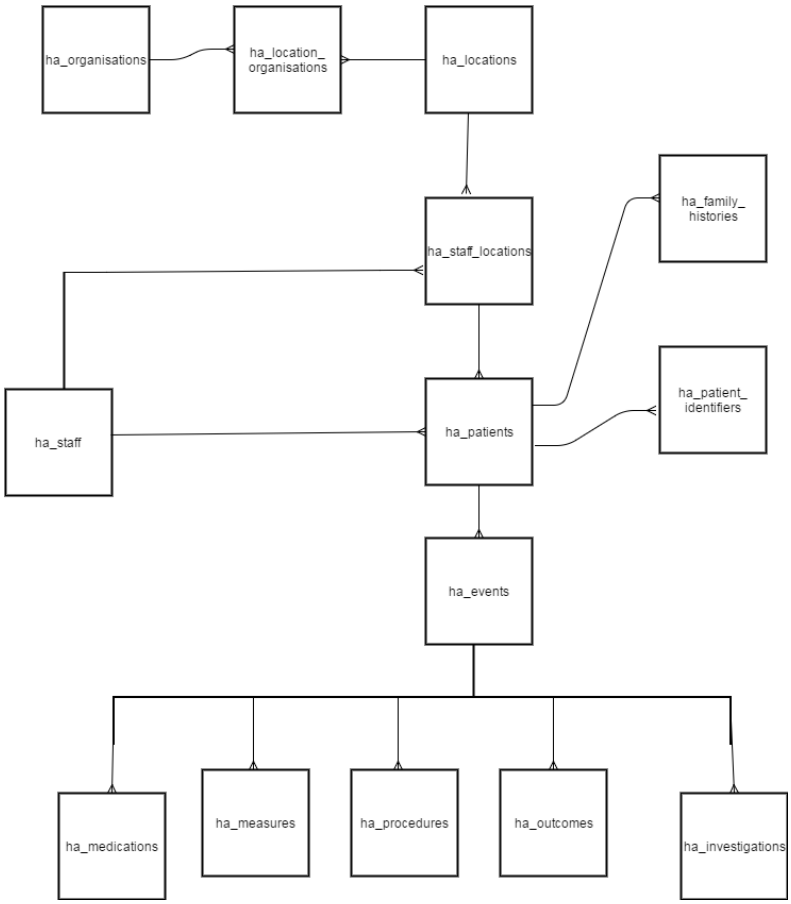


Figure 2.2: Health Analytics Schema [5]

2.3 The data_loader Tool and the YAML Files

As part of their work on various health projects, Aridhia have developed a Python based tool, called data_loader, which takes data files, usually in CSV format, containing the data, and a configuration YAML file that defines the destination tables [7]. A detailed explanation of the structure of the YAML files is provided in this section.

When the data_loader is ran with those files, a SQL script is generated, and this SQL script can be used to populate the database [7].

The data_loader can be used with different sub-commands but the focus here is on the "FILE" subcommand. The data is loaded into a staging table, which has the same fields as the CSV file but the data fields are being trimmed. Once the loading into the staging table is done, the destination tables defined in the configuration file are populated [7].

The YAML configuration file contains 4 sections: identification, input tables, mappings, and destination [7].

The **identification** section is where names and brief description for the data set used are provided [7].

```
---
meta:
 section: identification
 dataset_name:  demographics
 feed_name: demographics-study1
 feed_description: demographics data from study
```

Listing 1: YAML Identification Section

The **input tables** section provides a symbolic name, in the example it's 'study1.csv' for the CSV file that's going to be used for data loading. When the data_loader is ran the symbolic name must be linked to the actual file, the path to the actual CSV file should be specified [7].

```
---
meta:
  section: input tables
stage_study1:
   source: study1.csv
```

Listing 2: YAML Input Table Section

The **map definition** section is used to map namings of one string to another: how something is written in the CSV file vs how it should be inserted in the database.

```
---
meta:
  section: map definitions
gendermap:
    Male: male
    Female: female
```

Listing 3: YAML Mapping Section

The **destination** section specifies where the data from the staging table should be stored. Here is where explicit casting of values is done if necessary [7].

```
---
meta:
  section: destination
  table: ha_patients
  keys:
   - patientid
p:
  age: cast(stage_table1.age as float)
```

Listing 4: Destination Section

In the case of the pilot project, the destination section of the YAML file includes only one destination table while the actual YAML files used with the health analytics schema contain several destination tables in this sections. More information about the structure of the destination section of the YAML files for each of the projects can be found in the next sections.

2.4 Other aspects of the system

Customers of Aridhia get access to the Analytixagility platform, that can be accessed from a browser. Once they are logged in, depending on the project they have permissions for, they can access R-shiny applications that offer data visualisation. Currently, the platform supports only R-shiny but the long-term goal is to extend the functionality and provide support for Django or AngularJS. However, a time-frame for completing this hasn't been specified.

Chapter 3

Requirements and Risk Assessment

This section includes an overview of the general requirements of the system, identified in the beginning of the project. However, as it can be seen from the consequent sections of this report and as software engineering projects show in practice, requirements change over time as the developers learn more about the project and as the customers are provided with prototypes.

3.1 Data modeling and Data Loading

The goal of this stage was to identify what data needs to be stored and find an effective way of storing the data from the CSV files into a database. Since Aridhia have been dealing with this aspect and have a schema that has been used with similar projects and a tool that can be used for data loading, their system was considered as one of the options.

As part of this stage, a pilot project had to be completed in order to develop understanding of the existing system, the pilot project is discussed further in Chapter 4.

The requirements for this stage can be summarised the following way:

- R1: Complete a pilot project, aiming to load the data in one table, by using the Aridhia data loading tool
- R2: Extend the already existing schema, created by Aridhia, and load the data from the different studies
- R3: Produce a detailed user guide, explaining how YAML files are created and how the data loader should be run

3.2 Data Visualisation

The basic visualisation aspect of the project should include the following:

- R4: Integration of Greenplum with Django/AngularJS/Other technology of choice
- R5: Based on the type of variable provide basic statistics for the user: number of rows with ‘null’ value, rarest value, commonest value

- R6: Allow the user to filter the values such as age, BMI, ethnicity, and other available demographic indicators
- R7: Based on the type of the variable, an appropriate chart should be generated (bar charts or histograms)

Additionally, if time allows, the following requirements (only briefly explained, at the initial stage of the project) should be considered:

- R8 (Optional): Research and consider possible options for visualisation of metabolites in their super and sub-pathways. As a starting point, the Kegg pathway should be considered
- R9 (Optional): Based on the chose approach, visualise the metabolites

3.3 Risk Assessment

Often software development involves dealing with potential issues that may lead to unexpected outcome. When those issues are considered from the beginning of the project, the developers involved can prepare to tackle the problems and mitigate the risk. Thus, risk assessment is becoming a key aspect of agile project management. There are various guidelines [8][21], largely based on the work done by Barry Boehm, that aim to make risk identification and resolution easier.

The first step is usually to identify the possible risks (often this stage includes classifying the risk, assessing the likelihood of it occurring and the impact on the project).[8] The risk identification is usually followed by risk-management planning in order to prepare for the risk resolution tasks. Lastly, the risk elements that might affect the project progress need to be monitored continuously and "corrective actions should be taken" [8]. For this particular project, there are 3 possible risks that can affect the delivery of the product.

The first one is the so called technology risk, since the project involves using an already existing system that the developer should understand. The access, given to this system, can be considered as possible risk as well because usually work with third parties requires granting access to tools. Another challenge, if not a risk per se, is the communication overhead since the developer may need additional assistance if there's lack of extensive and up-to-date documentation. Assuming this is the case, identifying the right point-of-contact on the external technical team can also prove time consuming. There's a high probability this would affect the project since Greenplum's integration with web-frameworks is not particularly well-documented. Furthermore, Aridhia have their own tool for inserting data into Greenplum and the documentation in place on how this tool should be used provides examples that seem quite basic and are only a starting point.

Additionally, the environment used by Aridhia is quite limiting since in order to access the database, a user should connect to several VMs: firstly, to a Windows one and from there to a Linux one and then access to the database should be possible. It is important to convey that the limiting from a developer's point of view environment is there for confidential reasons since it's on Aridhia's system where the patient data files are hosted. Another issue, that contributes to the range of technologies being considered as a risk, is the lack of detailed knowledge the developer of this project has regarding the range of technologies and services Aridhia offer and how they handle similar projects. In order to mitigate those risks, a meeting with Aridhia was held in the beginning of the project and since then the developer has been in touch with the lead-software engineer of the company and to an extent some of the issues outlined earlier have been resolved.

Finally, since the project involves a customer this is another risk element since often it comes to customer availability, change and negotiation of requirements, and effective communication. In order to mitigate this risk, frequent, weekly meetings when possible will be scheduled with the customer; requirements will be gathered continuously and any work that requires customer input will follow closely an incremental and iterative approach.

Chapter 4

Sub-project 1

The following section covers the process of resolving the first main customer requirements (R1, R2, and R3), namely the data loading, discusses why the chosen approach is the already existing tool, and outlines any challenges that have been encountered.

4.1 Design Decisions

Since Aridhia already have developed the data_loader and this tool has been tested, trying out the tool on a pilot project was the first step. While getting used to something developed by a third party takes time, the initial assumption was that using this existing system would save time in the long-term. Furthermore, in the future the customer would have the option to get technical support from Aridhia. Finally, even if the customer decides to work with another company, they should be able to use the data_loader as this seems to be the agreement with Aridhia.

While previous projects done by Aridhia are developed in R-shiny, for the purposes of this project R didn't seem like the right option considering its scalability and learning curve. On the other hand, Django is lightweight, scalable and last but not least it's a familiar option that performs well for medium and large-scale projects.

While Django was the familiar choice, something that needed to be considered was Django's integration with Greenplum, since there is no documentation available covering this case.

Furthermore, since currently there is not an automated tool for generating YAML files, the possibility of creating one has been considered. However, after careful consideration this task has been given a lower priority since the creation of such YAML files requires some knowledge and understanding of future project requirements and creating a generic YAML generator is not considered effective.

4.2 Pilot Project

The first step of the project was based on some work already done by the customer. The idea of this piece of work was to learn more about the data_loader and the Health Analytics Schema, developed by Aridhia, and re-create something done by the customer. Since Greenplum and the data are only accessibly from the Aridhia system, all development has been completed there.

Additionally, another requirement, as outlined by the customer, was to create detailed documentation, explaining how to use the tool, created by Aridhia, and how to make sure the data loading has been successful. The work done on the documentation, and the evaluation done on its suitability are discussed at the end of this chapter.

Essentially, this pilot project consisted of creating a table with demographic patient data such as age, ethnicity, bmi and the respective measurements (blood pressure before and after the treatment) and populating it with the data from the 7 studies (genhat, gera1, gera2, pear1, pear2, invest, invest_ibc).

The table below outlines the name of the columns, the data types and some additional information that influenced the design of the final project database:

Column	**Type**	**Additional Notes**
subjectid	Char varying	
studyid	Char varying	Default value, depending on the study
substudyid	Char varying	(used in the case of pear 2 where there are 2 phases) Default value, depending on the study
drug	Char varying	csv data or default value
age	Double precision	95%
race	Char varying	
bmi	Char varying	
meds_pre	Char varying	csv data for one of the studies, default value for the rest
nofmeds_post	Integer	data for one of the studies,default value for the rest
dbp_pre	Double precision	
dbp_post	Double precision	
sbp_pre	Double precision	
sbp_post	Double precision	
dbp_delta	Double precision	Simple calculation field
sbp_delta	Double precision	Simple calculation field
uds_id	Integer	
uds_rowid	Integer	

Table 4.1: Pilot Project Table Fields and Notes

Note: required field that has to be provided if the data_loader is going to be used to populate the table.

As it can be see from the table, some of the fields have to contain default values (for example, “substudyid” is a valid field only for pear2 as this study has 2 sub-studies. Additionally, the CSV files for some studies specify the drug treatment, while for others this is a default value that has been added to the YAML file. As it’s been mentioned earlier, since for this project there is only one table that needs to be populated and the data is coming from one CSV file (in most cases), the destination section is relatively simple as in fact it is just a mapping of fields.

```
meta:
 section: destination
 table: dataingest
 keys:
   - subjectid
   - studyid
   - substudyid
p:
  subjectid: stage_genhat.GENHATID
  age: cast(stage_genhat.age as float)
  bmi: cast(nullif(stage_genhat.BLBMI,`NA') as float)
  studyid: "'genhat'"
```

Listing 5: Pilot Project: sample destination section

The sample YAML file, Listing 5, shows only some of the fields as inserting the rest is done in a similar way. In addition to the explicit casting, it's important to note the use of the single and double quotes when inserting a constant string. The single quotes are SQL delimiters and the outer, double quotes prevent YAML from parsing the inner quotes [7]. Another useful thing to note here is the way of handling null fields, marked as "NA" in the CSV files.

4.2.1 Findings and Ensuring Correctness

The pilot project turned particularly useful for the purposes of understanding the data sets and the tools already created by Aridhia.

Tasks such inserting a batch of data in the database and ensuring foreign key integrity, something Greenplum itself doesn't support, are simplified by the data_loader. This is not necessarily a tool that a user without any technical experience would find easy to use. In order to be able to write YAML files and use them as a configuration file for the data_loader, the user should have a good understanding of how the system works or at least should have access to more examples as the guide that has been provided by Aridhia only covers simple cases. The other reason sometimes the system can be challenging is how errors are handled and displayed. As long as the YAML file doesn't contain any parsing errors (spaces, field names, etc) running the data_loader and generating executable SQL script won't be a problem. However, at times the error messages are not very detailed and an unexperienced user might struggle to resolve them.

When considering the data not included in the CSV files, some of the data as it could be seen in the notes column of the table above was either default values or simple calculations. Since the simple calculations can be derived easily, that data is not stored in the final database. Default values can easily be specified in the YAML file, so the original CSV files do not require any modifications.

Since the pilot project was relying on using an already existing tool, that has been well-tested, the main assumption was that the data_loader works. Furthermore, since there were no exceptions raised, the task was marked as successful.Since the scope of the work involved only a single table, the testing hasn't gone in depth, and several queries were executed for sanity check to make sure all data has been inserted in the database.

4.3 Health Analytics Project

Another aspect of this mini-project was extending the Health Analytics Schema (HAS) and populating the tables with the required data from the patient studies. Since Aridhia work with various health projects, modifying their schema by adding new tables and fields is straightforward since they built this schema to be extensible as outlined in their design principles [5].

4.3.1 Changes to the HAS

In order to satisfy the requirements of the AIM HY project, the following modifications were made:

1. Adding new fields to the ha_patients table: age, bmi and a reference to the newly created ha_studies table. The documentation specifies two ways of extending the patients table: by adding fields to the ha_patients or adding fields to ha_patients_attributes [6]. The benefits of adding new fields to the ha_patients is minimising the JOIN operations, and in the case when there aren't many fields to be added, this is the preferred options [6]. Furthermore, the two options were discussed with the customer and using only the ha_patients table was the desired solution
2. Extending the database schema:
    - A new table to store the study_id and the substudy_id has been created
    - 2 events were created per patients: pre treatment and post treatment (this is extensible enough and further options could be added)
    - For each event, there could be several measures such as: systolic blood pressure, diastolic blood pressure, number of medications, etc.
    - A new table containing drug type data has been created

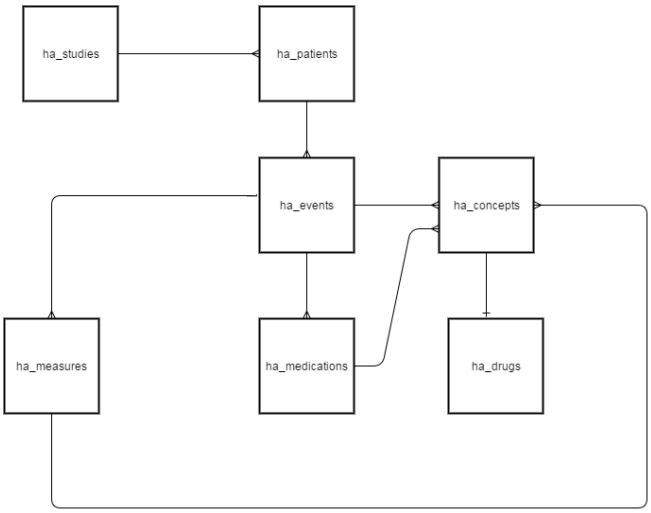


Figure 4.1: Updated schema

4.3.2 YAML Files

Compared to the YAML files used as part of the pilot project, the YAML files that were used to populate several tables were more complex. The identification, input tables, and map definition sections were similar to those

outlined previously but the destination part of the YAML file changed significantly. The ha_concept table was used to define several terms such as the event types (pre and post), the measurements (systolic blood pressure, diastolic blood pressure, medications before the start of the treatment), and the medications used. The code sample below outlines only one example but the same would be done for the rest of the terms.

```
---
meta:
   section: destination
   table: ha_concept
   keys:
   - code
c_d:
  code: "'/measurement sbp'"
  label: "'systolic blood pressure measurement'"
```

Listing 6: Inserting data to ha_concepts

The next code sample demonstrates how related items are inserted in multiple tables: once the patient information is inserted, an event for this patient is created.

```
---
meta:
 section: destination
 table: ha_patients
 keys:
  - health_id
p:
  health_id: stage_gera.id
  age: cast(stage_gera.AGE as float)
  ethnicity: _map(enthicitymap, stage_gera.ethnicity)
  bmi: cast(stage_gera.bmi as float)
---
meta:
  section: destination
  table: ha_events
  keys:
    - patient_id
    - event_type_id
e:
  #the event types have been defined in the concepts table earlier
  event_type_id: m.concept_id
  patient_id: p.patient_id
```

Listing 7: Referencing keys from other tables

Finally, it is interesting to note the format of one of the CSV files and how the mapping section of the YAML file can be used to resolve similarly structured CSV files. Table 4.2 presents blood pressure measurements, taken over different visits. Mapping a column from a CSV file to a column in the database has been explained earlier,

but for this particular case, the values of interests are decided based on a row value.

PatientID	VISIT	SBP	DBP	visit Date
15	1	140	80	22/10/15
15	2	110	90	12/12/15
15	3	120	80	03/03/16

Table 4.2: Sample CSV file data

As it has been discussed with the customer, only 2 measurements par patients are required to be stored in the database: pre and post treatment. For this particular study, the only measurement values needed for the database would correspond to visit 1 (measurement, taken before the start of the treatment) and visit 2 (measurement, taken after). The mapping section of the YAML file can be used to resolve this and to populate the database with the necessary values.

```
---
meta:
  section: map definitions
visitmap:
  '1': 'pre'
  '2': 'post'
```

Once this mapping is defined, the destination section for the concepts table would look like this:

```
---
meta:
  section: destination
  table:   ha_concepts
  keys:
    - code
c_e:
  code: _concat( '/event/', _map( visitmap, stage_table.visit ))
  label: _concat( 'event for visit ', _map( visitmap, stage_table.visit ))
```

Finally, in order to insert an event item (of type pre or post) for each patient with a timestamp the following destination reference mapping needs to be done, as described previously.

```
---
meta:
  section: destination
  table:   ha_events
  keys:
    - patient_id
    - event_type_id
    - start_date
e_visit:
  patient_id: p.patient_id
  event_type_id: c_e.concept_id
  start_date:   stage_table.date
```

As it can be seen from the code samples described earlier, while creating YAML files to be used with the data_loader follows a convention, there are some slightly more complicated cases where additional practical knowledge is required. However, once the developer is used to how the YAML files should be written, writing them is straightforward. Certainly, the process can be improved by providing more detailed documentation than what Aridhia have in place just now.

4.3.3 Ensuring Correctness

In order to test whether the data has been properly inserted, several queries were executed aiming to retrieve measurements and treatment information about the patients. Furthermore, a view with the structure of the pilot project table was created and results from running the same query against the pilot project and the HAS extension were compared.

4.4 Documentation: Requirements, Challenges and Evaluation

As it has been mentioned earlier, one of the main requirements for this part of the project was creating a detailed user guide, outlining the tables that have been created, the structure of the YAML files and how the user would run the data_loader.

This aspect of the project happened to be more challenging compared to the initial expectations. The initial specifications consisted of "user guide, outlining how to use the tool". Certainly, the requirements could have been captured in a better way and the structure could have been discussed in details and aspect such as level of detail, audience, purpose could have been clarified in the early stages. Initially, the customer was provided with a guide, covering only the pilot project and several weakness were encountered:

- Structure: The document was considered brief and "not-sufficient" by simply presenting the format of the pilot project (outline of the table and its fields), giving an example of a YAML file for one of the studies and providing a general overview of some caveats when it comes to writing YAML files. More details (as described below) were required.
- Audience: The document was assuming a knowledge of relational databases and a good understanding of the command line, thus it wasn't aimed at the correct audience. It's been specified that the document should be aimed at users with basic understanding of command line who haven't used the Aridhia system in depth before.
- Assumptions: The user guide was relying on the assumption that the user would have a good understanding of the Aridhia system.
- Prerequisites: The document was assumed to be used with the respective Aridhia documentation and was considered as a quick-start information guide, assuming the users would like to create their own YAML files. However, the idea was for the user to actually run the data_loader with the existing YAML files.

Based on the feedback provided by the customer, the following modifications were made to address the issues and enhance creating a self-contained user guide, aimed at the right audience:

- The structure of the document has been altered; more sections such as: motivation, background, system overview (covering Greenplum, YAML files, and the data_loader),ingest procedure and testing have been added

- More information was added about relational databases, and possible issues of using YAML files that can be encountered
- A more detailed step by step guide on how to use the Aridhia system has been added.
- Since the user would like to run the data_loader with the already created YAML files, a build script was provided to facilitate the process and save the user the time to copy-paste commands.

The newly updated user guide was sent to the customer for feedback before creating a similar one for the Health Analytics schema.

Chapter 5

Django and Greenplum

This chapter provides a general overview of Django's integration with databases, identifies the reason why Django can't be integrated with Greenplum, and discusses possible options for completing the visualisation aspect of the project, given the connectivity and time limitations.

5.1 Greenplum setup

Since Aridhia's system is not flexible when it comes to providing a development environment, the best way to work on testing integration and having a flexible environment was setting up Greenplum locally. The Greenplum Sandbox is in OVA format and can easily be imported into an Oracle VM [15].

The comprehensive tutorial on Greenplum includes sections on how to create users and user roles, how to create a database and tables, how to load data, etc.

5.2 Django and PostgreSQL

Assuming there's PostgreSQL installed on the local machine, the only thing that differs from the regular Django setup with sqlite database is that psycopg2 has to be installed as this is the database connector used with PostgreSQL databases. A few minor changes are required in the *settings.py* file.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'databasename',
        'USER': 'username',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '',
    }
}
```

If port is left blank, the default value is 5432.

5.3 Django and Greenplum

Greenplum was running on the VM and the settings file was similar to the one for PostgreSQL. The command *python manage.py runserver* worked. However, trying to migrate the database resulted into the following error: "UNIQUE index must contain all columns in the distribution key of relation "django_content_type".

As it has been mentioned earlier, when creating a table in Greenplum, a distribution key could be specified. If it's not specified, the first integer column is used as a distribution key. The distribution key should be unique for each record [12].

When the table contains a unique value, the table must be distributed based on that value and the primary key must be the same as (or a superset of) the table's distribution key columns [12].

The error occurs when Django tries to create the django_content_type table, since no distribution key is specified, the id is chosen. Later the unique constraint django_content_type_app_label is being added and the distributed key is redefined to match this. The distribution key is (app_label, model) but the primary key (id) is not a subset of the distribution key.

5.4 General Note on Django Integration

Django provides the flexibility of being integrated with legacy databases. Just by running the command [2]:

```
$ python manage.py inspectdb > models.py
```

The structure of the database tables and the information about the fields are exported in the models.py file.A few minor changes need to be made to this file to make sure the tables are managed. However, the core Django tables still need to be created and this is done when the migration command is run:

```
$ python manage.py migrate
```

While Django provides an option of being integrated with several databases of the same or different types (for example, as described in the Django documentation MySQL and PostgreSQL). However, even if both databases can be integrated with Django properly (which is not the case with Greenplum, if an option of integrating Django with Greenplum and PostgreSQL is considered) this model presents limitations at referential integrity level and contrib apps behaviour[3].

5.5 Solution

After researching possible workarounds and not being able to identify a way to integrate Greenplum and Django, the following options were considered:

Option 1: Since the project is not dealing with a lot of data and the parallel processing of Greenplum is not strictly necessary, using PostgreSQL with Django is possible since it won't affect the performance of the system. Furthermore, the data loading tool developed by Aridhia can be used with PostgreSQL. However, the only way to run the application would be inside the Windows VM, provided by Aridhia, since the visualisation would be based on sensitive patient data. Since currently, they are not using Django whether the customer would be able to get the necessary support from the developers from Aridhia is unclear. However, as it's been noted earlier the

company are planning to integrate their Analytixagility platform with Django and AngularJS, so depending on their list of priorities, the customer might be able to access the tool from the platform.

Option 2: Aridhia are working with R-shiny applications and Pivotal, the company behind Greenplum, have created a connector for R. However, this option poses a significant time constraint, since R and R-shiny are new tools and the problem with integrating Django and the lack of a workaround have been identified quite late, mid-January.

Given the time constraints and considering the problem domain, Django with PostgreSQL were chosen for this project, and while it's unclear whether the visualisation project will actually get the necessary support in the future, at the very least this is a useful proof of concept of what needs to be done.

Chapter 6

Sub-project 2

After clearly defining what the data visualisation should be and discussing the necessary features with Aridhia and the customer, it became evident they had tools that could generate basic statistical data about a table in a database, filter the table based on a field, and display graphs. Furthermore, they will be extending their R-shiny application in the near future and this will meet the requirements that have been identified with the customer and described here in Chapter 3, respectively R5, R6, R7.

Thus, the visualisation requirements were changed to visualising metabolites in their super and sun pathways which would have been the optional aspect of the project. The rest of this section covers the choice of technology, discusses the possible designs and the evaluations done.

6.1 Choice of technology

As it has been outlined earlier, due to the technical and time constraints, PostgreSQL and Djnago were chosen. Since most of the work is based on visualisation, another significant aspect was choosing a suitable visualisation library. There are many Javascript libraries and certainly, D3 is a popular choice since it offers flexibility and support for large data sets, and it allows code reuse. Furthermore, it is very well-documented and there is a large community behind it, a crucial requirement when something needs to be developed quickly. Additionally, previous exposure to D3 was another reason for choosing this library over others at this stage when time is a significant constraint.

6.2 Possible Designs and Further Requirements

Once the problem domain has been identified, several possible designs were considered:

- Option 1: Nested visualisation. Visualise the metabolites in their super and sub-pathways by displaying them as nested nodes.
- Option 2: Kegg pathway visualisation. A Kegg network is a "collection of pathway maps that represent molecular interactions" [14]. Based on the generic human metabolome data, the specific metabolites appearing in the studies can be mapped on the Kegg pathway.

Option 1 was identified as a starting point with the possibility of extending the visualisation to Option 2 if time allows.

At this stage, further requirements gathering has been done:

- R10: Metabolites size should be based on a p-value of interest. When choosing between area or radius to represent data, studies suggest that the area should be chosen over the radius, since it is more effective [19].
- R11: The colour of the metabolites should range, based on an estimate value.
- R12: When the user clicks on a metabolite, a scatter plot should be displayed showing the change in blood pressure for each person. Each patient is represented by a dot; on the y-axis is the change in blood pressure (measurement, taken after treatment minus measurement, taken before the start of the treatment), and the x-axis is $\log_{10}$ of the valye of this metabolite per patient.

6.3 Design and Customer Evaluation

The work done on the visualistion aspect of the project followed an iterative and incremental approach. There were weekly meetings with the customer, aiming to gather feedback and identify further requirements.

6.3.1 Iteration 1: Data Visualisation Model and Initial Work

During this iteration, a work has been started on visualising the metabolites in their super and sub-pathways. After reviewing existing research on way of visualising data, the circle packing model was chosen as it fits the project requirements.

The circle packing visualisation as an effective technique for presenting a large hierarchical data set has been discssed in details in a paper, presented in 2006 [22]. The research paper discusses the various advantages of circle packing such as ease of use, efficiency, "very clear bird view " [22], and readability. The experiment described in the paper is based on users comparing a traditional representation of a file system and the circle packing variant. The paper outlines the algorithm for packing circles as follows [22]:

- Determine the radius of the internal circles
- Construct a circle to represent the root node (current node as it's done recursively)
- Child nodes to be packed (layers should be zoomable)
- Until there are children nodes, pack and repeat previous steps

Overall, the findings were in favor of the circle packing model as a way to represent data that is usually presented as a tree diagram. The research describes the option of turning the circle packing into a 3D model instead of a 2D one [22] but at this stage for the purposes of simplicity, the visualisation is constraint to 2D.

By the end of Iteration 1, a small mock data set has been visualised and two zoom options have been presented, zoom on click and zoom on mouse scroll. Additionally, each metabolite was linked to a line graph, presenting blood pressure results. Since at this stage the visualisation was based only on mock data, the size of the metabolites didn't correspond to the p-value and the colour of the metabolite wasn't of any significance.

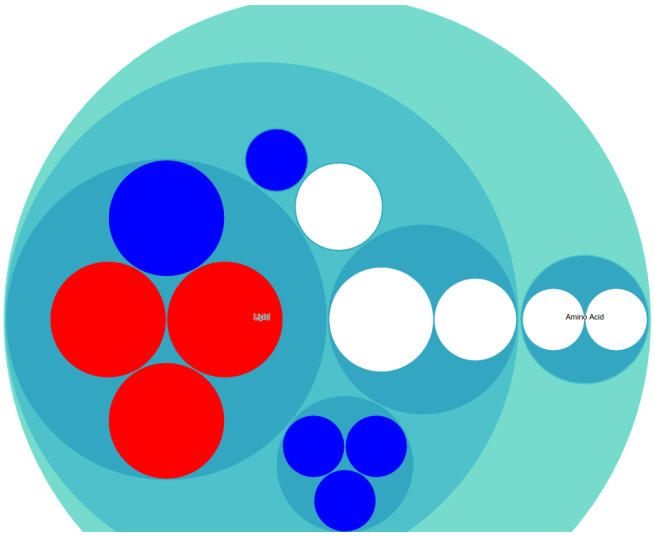


Figure 6.1: Iteration 1

During the customer meeting, the work done over this iteration was discussed, some of the requirements have been elicited. The findings of the meeting with the customer at this stage include:

- Zooming on click should be used as it seemed to be the more usable option
- Instead of a line graph, there should be 2 scatter plots for each metabolite: one for the systolic blood pressure and another one for the diastolic blood pressure
- There should be a help button to explain the interaction with circle packing visualisation
- The original data set includes 4 p-values (reflected as radius of the circles) for each metabolite and different estimates (reflected as colour) since there are 2 types of blood pressure and 2 possible drug treatments, the user should be able to select appropriate values for those
- Actual data should be used for next iteration to see how the visualisation of metabolites scales

6.3.2 Iteration 2: Further Changes and Evaluation

The focus of this iteration was using actual metabolomic data for the circle packing and implementing the filtering based on type of blood pressure and type of drug treatment(Figure 6.2).

Figure 6.2: Iteration 2: Blood Pressure and Drug Selection

Figure 6.3 presents visualisation of metabolites using actual data and only metabolites that have pathways are shown.

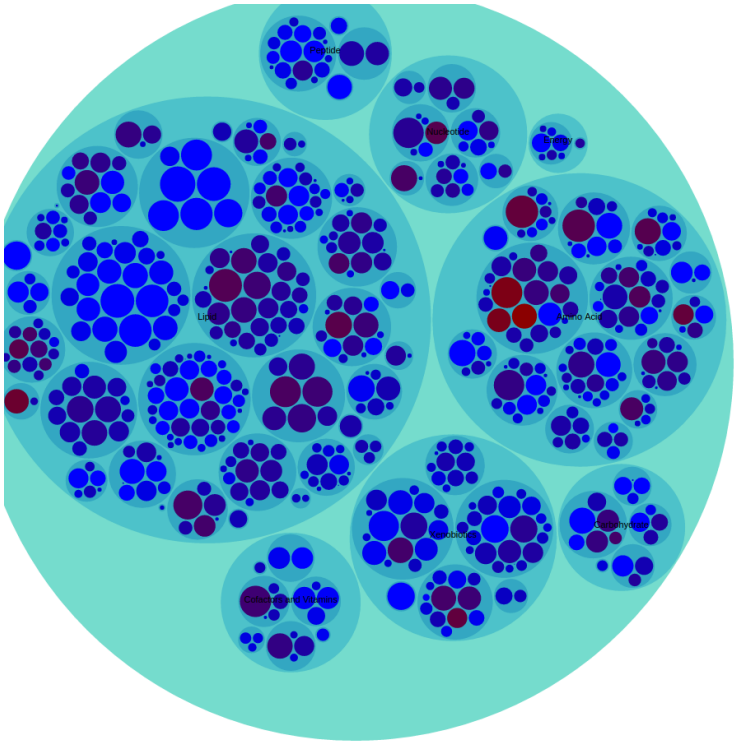


Figure 6.3: Iteration 2

Similarly to the previous meeting, the visualisation has been discussed in details with the customer to identify the following:

- Legend should be included for colour and size of metabolites to make the visualisation more readable and easier to understand
- The colour range should include 3 values: red (approximately negative), white (approximately 0), and blue (positive values) to help identify significant metabolites since at the moment as it can be seen from figure 6.3, the circles lack this
- Labels should be made slightly more readable by including a tooltip that shows up on hover
- A way of making the pathways slightly more visible would be helpful: currently, if the user hovers over a pathway, the boundaries of the pathway are selected.
- Actual data would be provided for the scatter plots during the next iteration to identify whether it scales well.
- The scatter plots should appear on click only if the user has reached the actual metabolites
- For the value/change in blood pressure scatter plots there are several options: only one set of graphs to appear on click and to be replaced when another metabolite is clicked or the scatter plots should appear in a separate scrollable area
- Both of the scatter plots should disappear on click, since at this stage, they only appear together but one can be deleted on click but the other remains. However, it seems to be more intuitive to have them disappear together

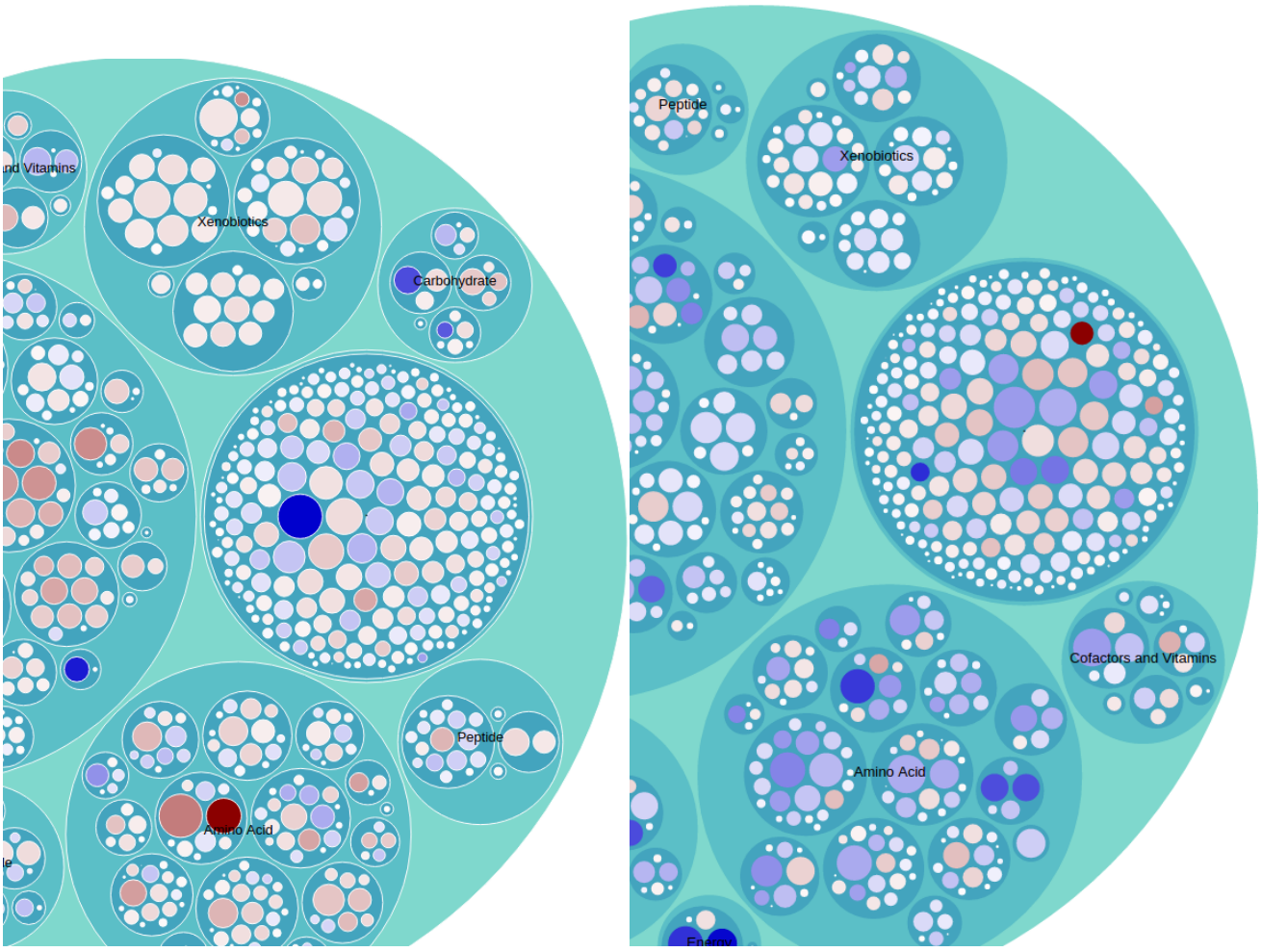
Most of the development of the visualisation has been done locally, and the end product is expected to be deployed on the Aridhia system, so it can be accessed from within their virtual machine until their platform is extended to work with other types of web-applications, not just R-shiny. Thus, this is a task that should be considered for the next iteration.

Due to the significant time pressure, the identified tasks had to be prioritised and completed as follows: integrating the scatter plots with the database (and using actual data), completing the most necessary changes to the UI (labels on hover, scatter plots replacing each other on new click, legend for colour and circles), and starting work on setting up the PostgreSQL and Django on the Aridhia system.

6.3.3 Iteration 3

The aim of this iteration was to use actual data for the scatter plots and finalise the development as much as time allows.

Since the metabolite classification didn't seem visible enough, a padding border has been added to the nodes.

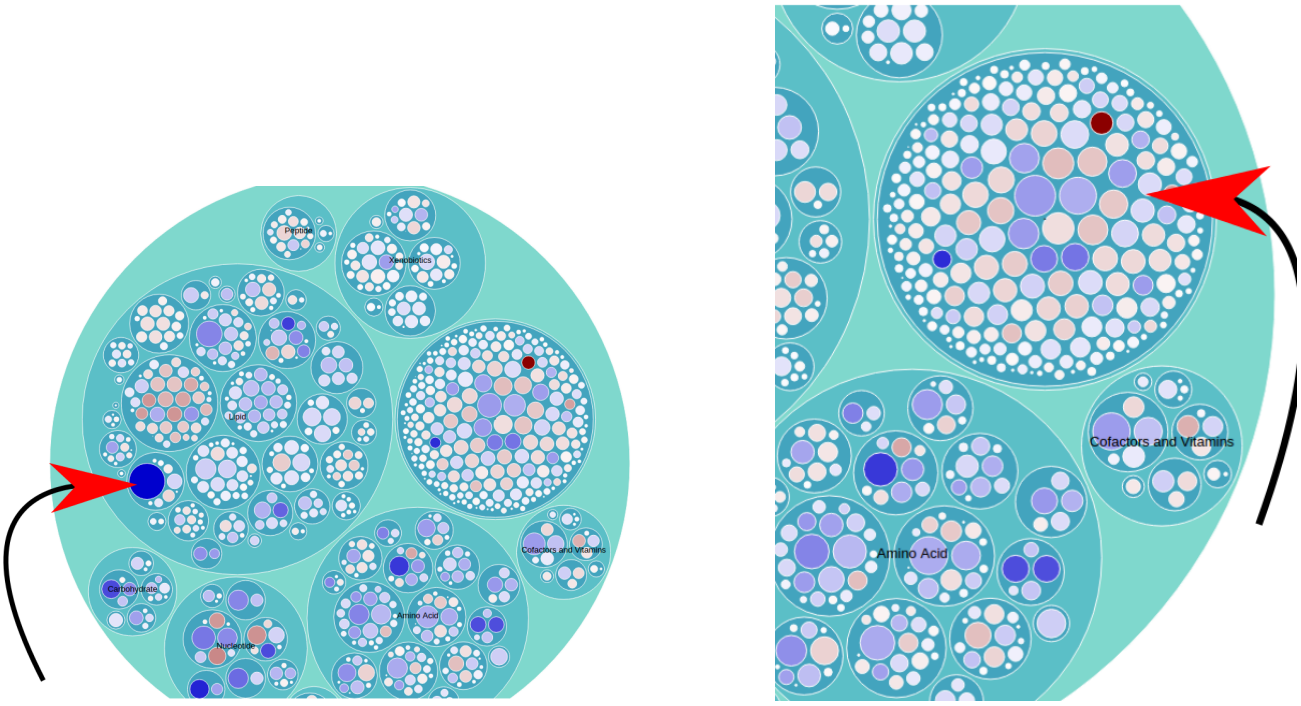


(a) Border

(b) No border

As part of the research of AIM HY, earlier this year, a significant metabolite has been identified. Thus, a good usability and accuracy test would be to check from the circle packing visualisation whether this metabolite can be spotted easily.

Furthermore, it is interesting to note the metabolites with no pathways, and the variation in the size and colour.



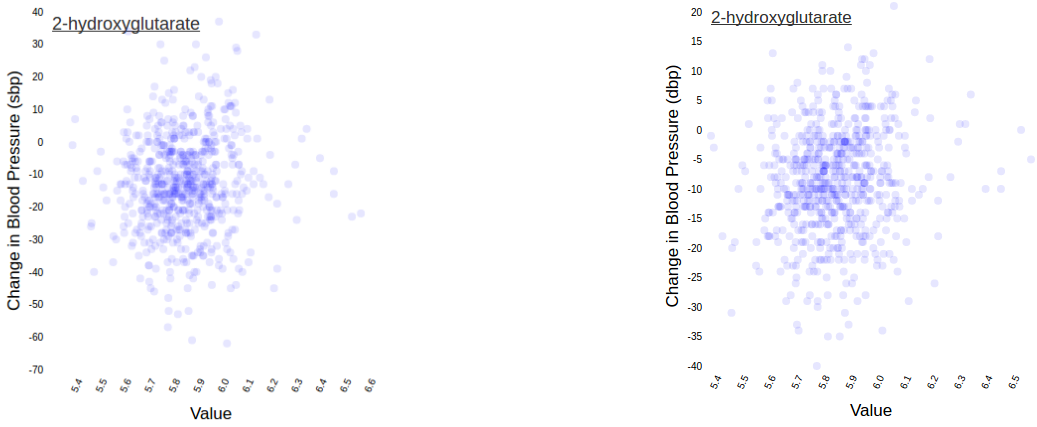
(a) Analysing Visualisation: significant metabolite

(b) Analysing visualisation: metabolites with no pathways

Figure 6.4: Scatter plots.

Since at this stage there were over 600 patients presented as dots, the scatter plots weren't readable, so an alpha channel was added to the colour of the dots and now a pattern can be seen.

The results of the scatter plots correspond to the scatter plots that have been generated by the customer, earlier this year when the recent findings of the project have been presented. Again, this could be used as another usability and correctness test.



(a) Systolic Blood Pressure

(b) Diastolic Blood Pressure

Figure 6.5: Scatter plots.

The implementation of the circle packing, largely relies on the default d3.pack() layout for packing the

children of each node, sorting the nodes, setting the area of each circle proportionally to the value, adding padding to the adjacent circles, etc [23]. Furthermore, as it has been discussed previously when choosing value to be proportional to a circle radius or area, it is better to choose the area [19].

At this stage, the scatter plots appear only if the user is at the corresponding sub-pathway. Any other clicks on them have zooming effect. There is only one set of scatter plots appearing at a time, and they replace as the user clicks on another metabolite. On hover, the name of the node (metabolite, sub-pathway, or super-pathway) appears as requested. There is a "Help" button overlay that gives a brief description of what the visualisation is, and explains how one should interact with the zooming in and out (ie double-click to zoom-out).

Finally, a colour bar has been provided to make the colour-coding of the metabolites more readable. The legend requirement should be extended to including a legend of the circle sizes since the implementation of d3.pack() allows leaf nodes to be compared [23].

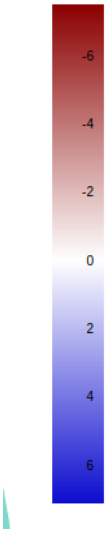


Figure 6.6: Colour Legend

A task that remains after the end of this iteration is making the tool available as part of the Aridhia system, so even if not integrated with their Analytixagility it can be used. This would require setting up and populating the PostgreSQL database, and downloading the required Javascript libraries since the Aridhia system doesn't allow network access. Furthermore, the customer should be provided with a user-friendly way of starting the Django application but this could be solved easily by writing a shell script that could start the Django server. Since this application hasn't been developed by Aridhia, it remains unclear whether using it in the long-term would be possible.

Chapter 7

Conclusion

This chapter provides an overview of the aims that have been completed and explains why some of the project requirements haven't been satisfied. Furthermore, there is a discussion of possible improvements from addition of new features to making changes to the software process.

7.1 Summary

Looking back at the requirements, identified during the earlier stages of the project, work has been done on using the tool provided by Aridhia and completing the pilot project, suggested by the customer. The Health Analytics Schema has been extended to meet the requirements of the AIM HY project. The customer has been provided with the requested documentation for the pilot project. Overall, the goals of the first stage of the project have been met. However, as previously explained the visualisation aspect led to numerous challenges, caused both by incompatibility of Django with Greenplum and lack of information on the range of tools provided by Aridhia. Those issues were resolved by finding a work around the Django-Greenplum integration issue and changing the visualisation requirements. However, those issues caused a significant delay and while some of the requirements have been satisfied, the visualisation of metabolites is still work in progress.

This project can be considered as an excellent introduction to the challenges involved in working with customers and building on top of existing systems. Often in industry, companies rely on organisational memory and their documentation is not detailed, sometimes out of date, which to an extent describes the quality of resources, provided by Aridhia.

Furthermore, while from agile point of view working with customers can be extremely beneficial, and in the case where the customer is actually the main user of the system, the interaction certainly facilitates building software that meets its requirements, the main drawbacks are related to availability, communication overhead, and managing expectations.

Overall, considering the list of requirements identified in the beginning some of the goals of this project have been completed. However, there is more that could be done to improve the current product.

7.2 Future Improvements

- Design of the visualisation: While the nested super and sub-pathways circle packing model seems like an appropriate solution, trying the Kegg pathway model might be considered. Comparing the usability of the

two can establish which one is more useful for the purposes of the AIM HY project.

- Testing: while it can be argued that user acceptance testing is more important than regression testing, more work can be done in this area. For example, Selenium can be used to test the front-end.
- Refactoring: While using a framework, tends to provide low coupling and high cohesion, the overall code quality can be improved and some time can be spend on refactoring.
- Security: The initial assumption was that the application would only be accessibly via the Aridhia platform. Thus, focusing too much on the security aspect wasn't identified as a priority. However, this might be something to consider in the future once there is more information from Aridhia on how they are going to extend their platform
- Making the application available on the Aridhia system: This has been mentioned earlier as a task for Iteration 2 and 3. However, due to time constraints it hasn't been completed. Discussion with the customer on how to make the application as usable (on start-up) as possible should take place. Perhaps, speaking to Aridhia and seeing whether they would like to take ownership of the application would prove beneficial as this might lead to actually have them supporting it
- Deployment: As it has been explained earlier in this report, while properly deploying the Django application and making it visible via the Analyticagility platform it's something that should be done by Aridhia, liaising with them and discussing the progress further might be desirable since there seemed to be aspects of this project that lacked detailed understanding of some of their tools and the primary focus was the data_loader
- Visualisation and wider context (research): Since this packing model and the scatter plots provided a good way of visualising statistics, and in general since the packing model seems to be a useful way of displaying hierarchical data, another future improvement might be researching the needs to similar health project. It would be interesting to see how important visualisation is for them; what kind of tools they would like to use, and what kind of visualisation models would fit their requirements. If this particular visualisation model proves useful, producing a more general but easily customisable version could be considered.
- Functionality additions: exporting the scatter plots as an image or a PDF; being able to see patient demographics data on hover when looking at the scatter plots; loading animation while the user is waiting for the value/blood-pressure graphs to appear since it takes some time before the graphs are displayed
- Integration with the ChemSpider API: ChemSpider provides detailed chemical data about compounds and it can be used to draw the molecules. There is a Python wrapper for the ChemSpider API which makes it easy to integrate with a Django application, for example. This would add another level of visualisation which might be interesting to some users. However, this feature relies on properly deploying the application and making it accessible from a web-browser.
- Process improvement: Some aspects of the project such as making small incremental changes and working closely with a customer were following well-established agile practices. However, there are certain aspects such as communication with Aridhia and the customer that can be improved further. While communicating with Aridhia via email proved useful for answering questions regarding the data_loader, having meetings in person with a detailed introduction of their tools would have been beneficial (a meeting of that format has taken place earlier in October, however it was focused on the data loading tool) in terms of understanding what the customer actually needs that's not offered by them. Furthermore, better planning techniques need to be considered since there were some cases of agreeing on what to do for the an iteration and then those tasks remained in the backlog for about 2 weeks, so some features weren't delivered on time. A possible way of resolving this issue would be for the developer to assign story points to the tasks that need to be completed and reflect on the workload per iteration. After several iterations, the developer should be able to establish an understanding of the project velocity and this would help focus on the "right" amount of tasks.

7.3 Lessons Learned

Working on a project based on an existing system and involving several stakeholders can pose significant challenges to the software development process; from getting access to all required tools and being granted the necessary permissions on time to being able to gather requirements and put the software project into perspective. From a software engineering point of view, the main lessons relate to the importance of good documentation, detailed understanding of an existing system, and ability to communicate effectively with stakeholders and negotiate requirements. Furthermore, the developer has gained understanding of challenges that can arise when dealing with confidential data and the lack of flexibility of the development environment in that case. Finally, the developer was exposed to some new technologies such as Greenplum, and had the chance to do some interesting investigative work in order to discover the cause of the unsuccessful integration of Django with Greenplum.

Bibliography

[1] Hypertension: an urgent need for global control and prevention. *Lancet*, 383, 2014.

[2] Django. Integrating Django with a legacy database. https://docs.djangoproject.com/en/1.10/howto/legacy-databases/. Accessed: 2017-01-15.

[3] Django. Multiple databases. https://docs.djangoproject.com/en/1.10/topics/db/multi-db/. Accessed: 2017-01-15.

[4] Aridhia. Analytixagility. http://www.aridhia.com/.

[5] Aridhia. Health analytics schema - principles and design strategy. 2016.

[6] Aridhia. Strategies for extending the health analytics schema. 2016.

[7] Aridhia. Using the data_loader tool. 2016.

[8] B. Boehm. Software risk management:principles and practices. *IEEE Software*, 8:32–41, 1991.

[9] Mayo Clinic. High blood pressure (hypertension). http://www.mayoclinic.org/diseases-conditions/high-blood-pressure/basics/symptoms/con-20019580. Accessed: 2017-03-14.

[10] AIM HY consortium. Stratified medicine:full stage case for support form. 2014.

[11] D3. D3 Data-Driven Documents. https://d3js.org/.

[12] Pivotal Documentation. CREATE TABLE. https://gpdb.docs.pivotal.io/4360/ref_guide/sql_commands/CREATE_TABLE.html. Accessed: 2016-10-12.

[13] Pivotal Documentation. Summary of Greenplum Features. https://gpdb.docs.pivotal.io/43120/ref_guide/feature_summary.html. Accessed: 2016-10-14.

[14] Genome. KEGG PATHWAY Database. http://www.genome.jp/kegg/pathway.html. Accessed: 2017-03-05.

[15] Greenplum. Greenplum Database Tutorials. http://greenplum.org/gpdb-sandbox-tutorials/. Accessed: 2016-10-20.

[16] E. Harris. Biochemical facts behind the definition and properties of metabolites.

[17] AIM HY. About aim hy. http://www.aimhy.org.uk/about-aim-hy/. Accessed: 2017-01-18.

[18] T. Mestrovic. What are metabolites? http://www.news-medical.net/life-sciences/ What-are-Metabolites.aspx. Accessed: 2017-03-10.

[19] J. Steele and N. Iliinsky, editors. *Beautiful Visualization: Looking at Data through the Eyes of Experts*. O'Reilly Media, Canada, 2010.

[20] M. Swain. ChemSpiPy. http://chemspipy.readthedocs.io/en/latest/. Accessed: 2017-02-15.

[21] R.L. Van Scoy. Software development risk: Opportunity, not problem. Technical report, Software Engineering Institute: Carnegie Mellon University, 1999.

[22] Weixin Wang, Hui Wang, Guozhong Dai, and Hongan Wang. Visualization of Large Hierarchical Data by Circle Packing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06)*, pages 517–520, 2006.

[23] D3 wiki. Pack layout. http://d3-wiki.readthedocs.io/zh_CN/master/Pack-Layout/. Accessed: 2017-02-24.

Appendices

Pilot Data Ingest Project

1 Motivation

The purpose of this document is to illustrate how the *data_loader.py*, a tool developed by Aridhia for inserting data into a Greenplum database, works; provide a general description of the structure of the yaml files, configuration files used with the tool. Finally, the report aims to provide an outline of how to run the data_loader and insert data from several CSV studies into one database table.

2 Background

The project contains patient data such as age, BMI, nationality, measurements (systolic and diastolic blood pressures) and medications. Currently, the data is stored into CSV files. In order to be able to query the data and analyse it, those patient records should be inserted into a database. Since the approach of inserting data with the *data_loader* is new, a pilot project to demonstrate how the tools work is being developed.

3 System Overview

3.1 Greenplum and some database concepts

A database is data structured in a logical manner. In this specific case, the definition of a database is restricted to a relational database, one which as part of its modelling of the data recognizes how the objects are related.

A database **schema** is a collection of all database objects, such as database tables and users, and it defines the relations and the attributes of the tables in the database.

Greenplum is parallel database, based on PostgreSQL. The data is distributed among several Greenplum segments, that's why when the user creates a table they can specify a **distribution key**.

If the user doesn't specify the distribution key, then by default the **primary key** is chosen. The primary key's function is to uniquely identify all table records. The primary key is unique and it cannot be null.

In databases, 'null' signifies a missing or unknown value.

When discussing Greenplum is important to note that foreign keys can be defined but referential integrity is not enforced.

A **foreign key** is a field in a table that uniquely identifies a field in another table. Consider the following two tables: a study table and a patients table. The relation between the two tables is one-to-many, as there are many patients per study. The unique identifiers are the *study_id* and the *patientID*, respectively. *study_id* in the patients table references a particular study from the study table, thus *study_id* (in the patients table) is a foreign key.

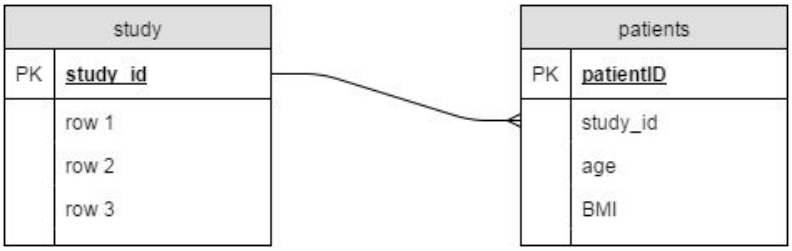


Fig 1: Example of a foreign key relation.
The ***study_id*** in study is a primary key in that table
and it is a foreign key in the patients table.

Referential integrity is a feature of most database management systems have and the aim is to make sure the data is consistent. Considering the 2 tables above, an example of referential integrity that won't be enforced (the case of Greenplum) is deleting the study table without a warning message. In most databases, this operation won't be allowed since *study_id* in the patients table references the study table.

3.2 What is data_loader.py?

The file *data_loader.py* is a Python-based tool which generates SQL script that can be used to load data into the Health Analytics Schema, a database schema developed by Aridhia. The *data_loader* can be ran with two different commands, the example discussed later on uses only the FILE subcommand that loads data into a staging table, a table that is essentially the CSV data table.

When executing the SQL script generated from *data_loader*, the user will see command prompt output of created and updated records.

3.3 YAML files

The loader execution takes 2 input arguments, one is the CSV data file and the other a YAML configuration file. YAML is human-readable language, used for creating configuration files for applications dealing with data storing. The YAML files used with the *data_loader* contain 4 sections: identification, input tables, map definitions, destinations.

Below, there's a short description of each section:

- **Identification**: specifying the name of the dataset and providing a short description.
- **Input tables**: provides a symbolic name for the CSV file that's going to be used for data loading. When *data_loader* is run the symbolic name must be linked to the actual file- the path to the actual CSV file should be specified.

- **Mapping**: mapping from one string to another, eg:

```
meta:
    section: map definitions
gendermap:
    Male: male
    Female: female
```

- **Destination**: specifies where the data from the staging table should be stored. In the case of

the pilot project, there is only one table specified in the destination section. The extended has_schema document will present a more complicated example. Eg:

```
meta:
  section: destination
  table: dataingest
  keys:
    - subjectid
    - studyid
    - substudyid

p:

    subjectid: stage_genhat.GENHATID

    age: cast(stage_genhat.age as float)
```

There are a few caveats to note:

1. Default values insert: the following format should be used when inserting a default value, eg:

meds_pre: " 'Currently untreated' " (the single-quotes: needed for SQL strings; double quotes: prevent the yaml parser from eating the inner quotes)

2. Explicit casting of values is required. By default all fields in the CSV are populated with string values and those values don't correspond to the type of the table fields, that's why casting is necessary, eg:

bmi: cast(stage_pear1.bmi as float)
dbp_delta: cast (stage_pear1.AC5_CDBP as float) – cast(stage_pear1.AC3_CDBP as float)

3. Handling null values in the dataset; when there's a value labelled with 'NA' in the CSV, this value is replaced with 'null', eg:

dbp_pre: cast(nullif(stage_genhat.V2DBP, 'NA') as float)

4 Ingest Procedure

The following section describes the prerequisites of the data ingest, running a simple example to populate a table, aggregating data from several sources and testing whether the data ingest has been successful.

4.1 Prerequisites

4.1.1 Accounts and permissions

- Remote Desktop Access
- GitLab account to be created: the project requires the git repository, containing the *data_loader.py* and some sample YAML files and examples to be cloned
- Access to the *aim_hy* database to be granted

4.1.2 Login

Note: When outlining the steps how to login, the assumption is that Windows is used on the user's local machine.

- Go to: https://access.stratmed.co.uk/RDWeb/Pages/en-US/login.aspx?ReturnUrl=/RDWeb/Pages/en-US/Default.aspx

There should be a login page. The "Domain\username" should be: "SMS-IC\username". The password is the standard password for this user. If there are any issues, Aridhia's support team should be able to address any enquiries.

- Once the user details are completed, an RDP connector should be downloaded.
- Double-click the RDP connector and provide the login details as requested.
- In order to connect to the Linux VM, double-click on the Putty application. Specify the host name and choose "Open". A login option would be given, provide your username.

4.1.3 Ensuring prerequisites are installed:

- Once the user is logged in, the prerequisites should be checked:

```
$ sudo yum install postgresql93  #tool for accessing postgres
$ sudo yum install git           #necessary since the git repository with the tools needs to be cloned
$ sudo yum install PyYAML              #yaml parser for python
$ sudo yum install pyparsing     #parsing module for python
```

- Access to the database:

```
$ /usr/pgsql-9.3/bin/psql -d aim_hy_data -h smspopsvip -U username
```

The flag ***"-d"*** is used to specify the database, in this case it's "*aim_hy_data*". The flag ***"-h"*** is used to specify the hostname for the database, ***"smspopsvip"***. The reason for giving the whole absolute path to psql is that the path variable hasn't been set.

The password will be requested and if the user has the necessary permissions, the login will be successful:

```
aim_hy_data=>
```

- Cloning the Health Analytics Schema repository:

```
$ mkdir ~/git
$ cd ~/git
$ git clone https://username@collab.aridhia.net/open-source/health-analytics-schema.git
```

As mentioned earlier, a prerequisite is having a GitLab account, created by Aridhia.

4.1.4 Installing the database objects (optional step)

NOTE: This and the next step should be **skipped** if the particular database has been setup for this user, because the "*ha_schema_core.sql*" script drops the already existing tables and creates

them again.

$ cd ~/git/health-analytics-schema/src/core

$ echo "set search_path to aim_hy_data_sandbox;" | cat - ha_schema_core.sql ha_eventstream.sql | /usr/pgsql-9.3/bin/psql -d aim_hy_data -h smspopsvip -U username

- The command above sets the path to *aim_hy_data_sandbox*, the schema that's relevant to the particular database. It executed the SQL scripts that create the health analytics schema tables (*ha_patients, ha_staff*, etc).

4.2 Ingest procedure:

4.2.1 Files and directories required

- *loadData.sh*: a bash script that creates the SQL scripts, used for populating the database, and executes the SQL scripts. Should be placed under:
  - *~/git/health-analytics-schema/src/data_loader/*

- A directory, called *DataFiles*, containing the CSV files with the data. This directory should be placed under:
  - *~/git/health-analytics-schema/src/data_loader/features/data/*

- A directory, called *dataingest*, containing the YAML configuration files. The directory should be placed under:
  - *~/git/health-analytics-schema/src/data_loader/features/config/*

4.2.2 Executing the shell script

- Before executing the script, the necessary privileges should be given to the user.
  $ chmod +x loadData.sh

- Running the script:
  $ ./loadData.sh

  In the command line will be printed several things and then, the username will be requested. Once this is provided, a password request will follow. There will be more command line output such as creation and updates of entries.

4.2.3 The structure of the shell script

The main purpose of this script is to reduce the number of commands the user is required to input. The shell script does the following:

- Creates a directory for the SQL scripts that the *data_loader* generates.
- Runs the *data_loader* command FILE for each of the configuration files with the corresponding CSV files. This generates the SQL scripts.

- The username is requested and the SQL scripts are ran against the database and the table gets populated.

  The command used for that is the following:

echo "set search_path to aim_hy_data_sandbox" | cat - ~/sqlScripts/.sql | /usr/pgsql-9.3/bin/psql –d aim_hy_data –h smspopsvip –U $varname*

"*echo*" is used to get the output of a command printed in the command prompt. It's useful for debugging purposes.

"set search_path to aim_hy_data_sandbox" is a database command that saves the user the need to refer to each table in the particular schema by: *aim_hy_data_sandbox.dataingest* (dataingest is a table, created as part of that schema).

The command "*cat*" concatenates the SQL scripts that can be found in that directory. This particular directory, *sqlScripts*, is created at the beginning of the bash script.

Once this command is executed, all SQL scripts are ran against the database and the table is populated.

4.3 Testing

- Login to the database:
  $ /usr/pgsql-9.3/bin/psql -d aim_hy_data -h smspopsvip -U username
- Make sure you set the path:
  aim_hy_data=>set search_path to aim_hy_data_sandbox;
- Once the path is set, the following queries can be executed:
  *aim_hy_data=>select * from dataingest where studyid='gera2';*
  *aim_hy_data=> select * from dataingest where drug='Atenolol';*
  *aim_hy_data=> select * from dataingest where sbp_post is null and dbp_post is null;*
  aim_hy_data=> select count (distinct subjectid) from dataingest where studyid= 'genhat';

  The result should be: 11602

  aim_hy_data=> select count (distinct subjectid) from dataingest where nofmeds_post=1 and studyid='genhat';

  The result should be: 5663

5 Description of the table

The pilot data ingest project consists of all studies and the data is aggregated in one table. The table contains the following fields:

Column	Type	Notes
subjectid	Character varying	Not null
studyid	Character varying	Not null
substudyid	Character varying	Not null
drug	Character varying	
sex	Character varying	
age	Double precision	
race	Character varying	
bmi	Double precision	
meds_pre	Character varying	
Nofmeds_post	integer	
dbp_pre	Double precision	
dbp_post	Double precision	
sbp_pre	Double precision	
sbp_post	Double precision	
sbp_delta	Double precision	
sbp_delta	Double precision	
drugclass	Character varying	Hasn't been populated
bkeep	boolean	Hasn't been calculated
trial	Character varying	Hasn't been populated
uds_id	integer	
uds_rowid	inte*ger*	

Table 1: Description of the fields of the dataingest table, the table used for the pilot project

Constraints: primary key (*subjectid, studyid, substudyid*)

Foreign key: *uds_id references ha_update_data_sources(uds_id)*

.