

Transforming the CoderDojo Foundation into a Data Driven Organization Through Data Warehousing

A report submitted as per the requirements for the reward of

BSc (hons)

In

Computing (Specialization in Data Analytics)

By

Adam Horrigan

Declaration Cover Sheet for Project Submission

Name: Adam Horrigan
Student ID: x13735825
Supervisor: Simon Caton

SECTION TWO Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarized overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature: Adam Horrigan.

Date: May 6th, 2017.

Abstract

In 2017, data more data is being collected than ever before. Processing and interpreting large volumes of data can be a difficult business process. The CoderDojo Foundation is an open-source charity with the goal to teach children programming skills. The primary use case of this project is to serve the Foundation with a platform which allows the organization to make sense of the data they have stored.

Through providing a well-designed data warehouse architecture, the Foundation will have quick access to primary metrics which are required for reporting. By using open-source Python libraries and top visualization tools such as Tableau, this project can take multiple sources of data from the Foundation and present in a meaningful and informative format to grow the organization.

Keywords; Data Warehouse, Visualization, Modelling, Transformation.

Table of Contents

Transforming the CoderDojo Foundation into a Data Driven Organization Through Data Warehousing	1
Abstract	3
Introduction	7
Motivation for this project	7
Challenges.....	8
Background and literature review	9
Architecture	9
Design	11
Technologies	13
Alternative technologies explored.....	13
Methodology.....	14
Early Stages.....	14
Requirements gathering.....	15
User requirements definition	16
Environmental requirements	22
Use case diagram.....	22
Architectural view.....	23
Data warehouse dimensions	24
Implementation	25
Setting up the environment.....	26
Connecting to the sources	29
Main method	30
Transformation functions	32
Insert functions	33
Starting the main method	33
Evaluation and testing.....	34
Client feedback	38
Testing methodology.....	38
Non-Trivial Queries	45
Conclusions and future work	49
Appendix.....	50
September monthly report.....	50
Achievements	50
Reflection.....	50
Supervisor meetings	50
October monthly report	51
Achievements	51
Reflection.....	51
Supervisor meetings	51
November monthly report	52

Achievements	52
Reflection.....	52
Supervisor meetings	52
December monthly report.....	53
Achievements	53
Reflection.....	53
Supervisor meetings	53
January monthly report.....	54
Achievements	54
Reflections	54
Supervisor meetings	54
February monthly report.....	55
Achievements	55
Reflections	55
Supervisor meetings	55
March monthly report.....	56
Achievements	56
Reflections	56
Supervisor meetings	56
Bibliography.....	57

Table of Figures

Figure 1 - Data Warehouse Architecture.....	10
Figure 2 - Inmon Data Warehouse Model	11
Figure 3 - Kimball Data Warehouse Model	12
Figure 4 - Iterative Life Cycle Approach.....	14
Figure 5 - System Use Case Diagram	22
Figure 6 - System Architecture	23
Figure 7 - Data Warehouses Cube	24
Figure 8 - Visual Studio Code.....	25
Figure 9 - Sources of Data	25
Figure 10 - Installation of Required Tools	26
Figure 11 - Right Click on Databases	27
Figure 12 - Click on New Database	27
Figure 13 - Name the Database and Click ok.....	27
Figure 14 – Clone Repository	28
Figure 15 - Import Data Dump	28
Figure 16 - Configuration Details.....	29
Figure 17 - Connecting to Sources.....	29
Figure 18 - Gathering Source Data	31
Figure 19 - Users Source Data	32
Figure 20 - Transformation Function.....	32

Figure 21 - Insert Function	33
Figure 22 - Starting the script.....	33
Figure 23 - Events Schema	34
Figure 24 - Dojos Schema	35
Figure 25 - Users Schema	36
Figure 26 - Data Warehouse Schema	37
Figure 27 - Unit Tests	44
Figure 28 - Gender Breakdown by Country.....	45

Introduction

A data warehouse is a collection of transactions, structured to best aid business decisions within an organization. Bill Inmon formally defined a data warehouse as follows:

“A data warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process.”

Typically, a data warehouse is not normalized, meaning that the data is not structured such that data redundancy does not occur. The primary purpose of a data warehouse is to have optimal read times, rather than optimal write times. Joins are computationally expensive, by reducing the number of joins required, there will be a lower cost to reading. A lower cost, leads to an optimal read. By not normalizing the data warehouse, the number of joins is reduced. This is a design pattern which is suggested by Ralph Kimball. Kimball says:

“We are so used to thinking of n-way joins as “hard” that a whole generation of DBAs doesn't realize that the n-way join problem is formally equivalent to a single sort-merge. Really.”

Data visualization is a key field when exploring data. It is easier for a human to interpret data when charted than when represented through tables. Not all organizational metrics can be visualized based on the structure on the source data. Queries can be simply impossible or not feasible. Through a transformation of the data and reorganization, it can become possible to begin visualizing metrics which were beforehand not possible. The primary queries that exhibit this behavior are those which span several sources. At the end of the project, the initial sources will be transformed and structured such that they will be suited by use for the CoderDojo Foundation.

Motivation for this project

Motivation for this project stems from the CoderDojo Foundation needing a frequent and reliable process in which they can gather business facts, in conjunction with having limited resources. The CoderDojo Foundations source data is housed in a badly designed schema, which was rushed in development. This has led to poorly formatted data, without relationships between appropriate tables. Reorganizing the data to an architecture which can be used for analysis, proves a difficult task. The project was formally established through work placement.

The CoderDojo Foundation faces several problems with using their data.

1. Currently the data is stored in three production databases. Several of the key business measures involve queries which span across the three sources. There is no default method for cross database queries.
2. The relational database management system which is being used is PostgreSQL. Despite the nature of the data being relational, a lot of it is saved as JSON within relational columns. This further makes executing queries difficult. PostgreSQL does have several JSON functions built in but they cannot achieve everything.

3. In the current schema, there is no primary keys on tables and as such, no foreign keys linking table dependencies. This means that there is no referential integrity between tables which slows down performance. It also leads to data being removed in one table but not in another. This can lead to issues when joining.
4. Viewing the data. When the team does gather the statistics, there is no visually appealing way to view the data. Instead it is simply imported into an excel sheet where it can be read from.
5. Non-technical team members need to perform SQL queries. Only the development team should need to execute queries against databases, however as the reporting lead most frequently need statistics – they find themselves executing queries.

The CoderDojo Foundation is a small team, consisting of two senior software developers. By completing this project, the developers will be free to focus on user facing issues such as bugs in the platform or new features. The project will also allow the organization to be data-driven and have measures based on time and location granularity. The business measures are vital for the CoderDojo Foundation to secure funding as many donors require measures to monitor growth. This project will make these measures accessible at any point in time and improve reporting organizationally wide.

Challenges

There are several key challenges which pertain to this project, they are as follows:

1. Due to the CoderDojo Foundation being an open-source charity. There can be no tools used which require a license for. These rules out a lot of standard data ware-housing tools such a SQL Server. It also means the project must use an open-source management system to house the data warehouse.
2. Missing data. Many of the field which measures are required for are not a required field on the front facing user platform. This leads to a lot of missing data in the source data which can skew results. The missing data will need to be handled appropriately.
3. Security. As the CoderDojo Foundations primarily data source pertains to users under the age of thirteen, the data must be secure such that no one can access it except for administrators.
4. Time and location snapshots. The CoderDojo Foundation requires that measures can be viewed over a certain time and location period.
5. Bulk and unformatted data. The three sources of data are large and often formatted in strange formatting. A lot of time will be spent picking out target data and cleaning it such that it can be used for proper analysis.
6. The project must provide visualization of the business measures such that a non-technical person can make sense of the data. This means no outputs to terminal.
7. No primary key and foreign key relationships exist in the source. These will need to be established for the first time in this project.

Background and literature review

“The key to success in scalable data warehouse development and the single factor that contributes most to data warehousing success is a data warehouse architecture. The architecture and design of an enterprises warehouse should reflect the performance measurement and business requirements of the enterprise.” – Alan Perkins.

In 2003, Alan Perkins published *Critical Success Factors for Data Warehousing Engineering* and argued that the biggest characteristic that factors into a data warehouses success is its architecture. This is agreed with by Michael A. Schiff who published “Data Warehousing – The keys for a successful implementation.” Schiff believes that a successful architecture is a “*continuous journey*” rather than a one-time event. Schiff states “while an organization’s overall data warehouse architecture can encompass a variety of forms, each organization must decide what is right for its own purposes”. There are several data warehousing architectural models but each organization which build a model which is suited to their needs.

Architecture

Data warehousing architecture includes several key components (Weisensee *et al*):

1. Data Sources.
2. Data Warehouses.
3. Data Marts.
4. Publication Services.

An organization is likely to have many data sources, including several databases and third party data accessed through application programming interfaces. These sources may include, marketing, sales and customer support, (Ramsdale, 2001). A data warehouse is where the source data resides after being cleaned and consolidated. It allows measures to be viewed at several different granularities. A data mart can be thought of a subset of a data warehouse, whereby it is limited to a specific branch of the warehouse. A data mart could be a sales dimension which is only available to a finance department. Publication services allow the end user to interact with the data warehouse, an example would be Tableau. Refer to Figure 1 for a typical data warehouse architecture.

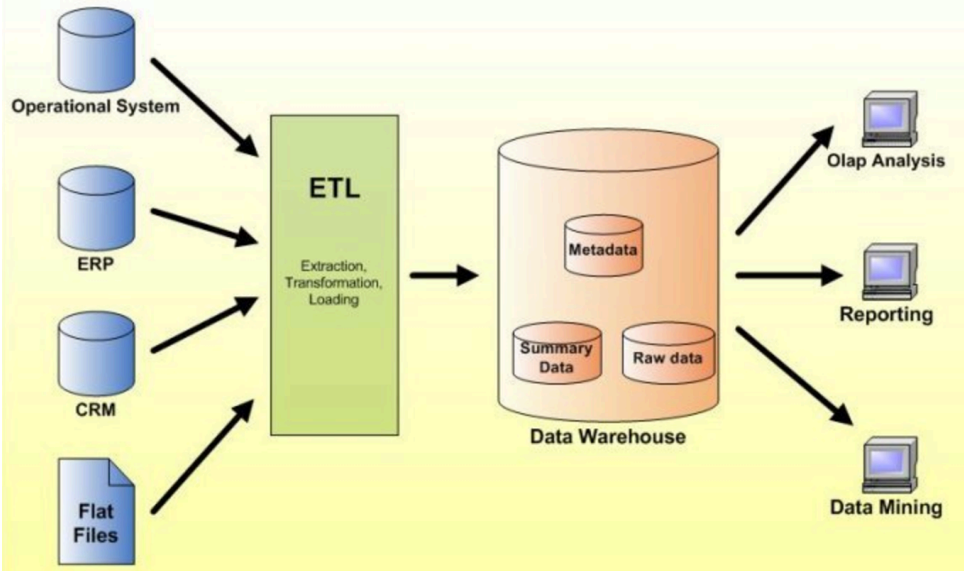


Figure 1 - Data Warehouse Architecture

Shaker H. Ali El-Sappagh agrees with this architecture, who in 2011 published “*A proposed model for data warehouse ETL processes*”. The process of extracting from sources and afterwards transforming the data before finally inserting to the data warehouse is a well-designed approach. El-Shappagh states “*Data is extracted from different data sources, and then propagated to the DSA where it is transformed and cleansed before being loaded to the data warehouse.*”

The above architecture is important as it allows for several key activities to be carried out which are vital to the success of the data warehouse.

1. Specific data can be selected from the source. Not all data in the sources may need to be processed and inserted into the warehouse.
2. The data can be transformed before being inserted to the data warehouse. Plain text is easier to analyze than JSON, for example. During the transformation phase, the data can be prepared appropriately.
3. The process gives an opportunity to re-structure the data which may be more appropriate for consumption. The data does not need to be inserted into the data warehouse with the same conditions and relationships which it was extracted with.

Design

"You can catch all the minnows in the ocean and stack them together and they still do not make a whale." (Inmon).

"The data warehouse is nothing more than the union of all the data marts" (Kimball).

Within data warehousing, there are two main design models which can be implemented. The Inmon model and Kimball model. Each offers their own advantage and disadvantage. Both models use an ETL process to load the data warehouse. The important distinction between the two is how the data structures are modelled, loaded and stored within the data warehouse.

The Inmon approach start by building a corporate data model. The purpose of the corporate model is to identify the key subject areas and more importantly and the key entities that the business operates. From the corporate model, a logical model is then created thereafter. A sales entity would be architected, with all relevant details pertaining to that entity. The logical model is responsible for capturing all business keys, attributes, dependencies and relationships. An important note is that the entities are built in a normalized form and data redundancy is avoided as much as possible. The final step is to build the physical model, which is also normalized. It is the physical model which Inmon considers the 'data warehouse'. The structure makes loading the data less complex with the consequence that querying is difficult as there are many joins and tables involved. To combat this, Inmon suggests building data marts for each entity. The data marts may be de-normalized. The data warehouse must be the only source of data for the data mart, to ensure referential integrity. Figure 2 shows a typical Inmon designed model.

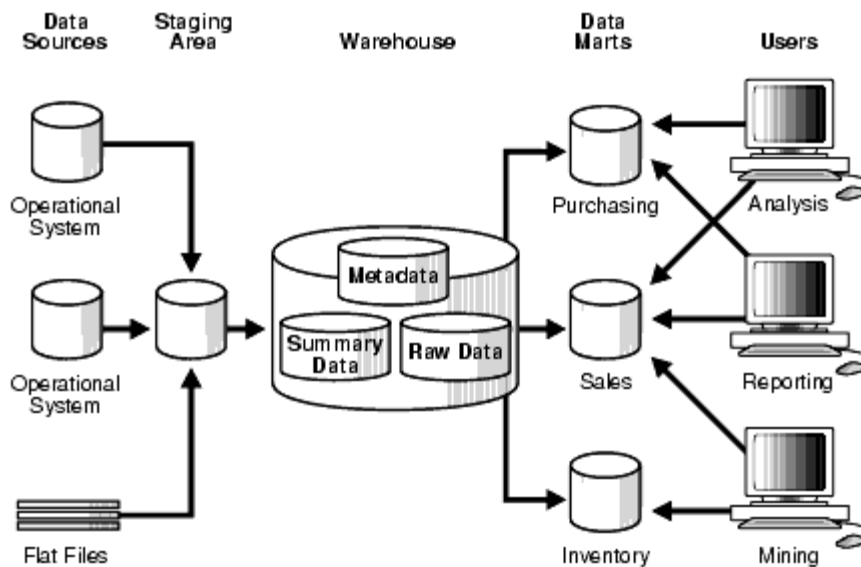


Figure 2 - Inmon Data Warehouse Model

Kimball's approach to data warehouse model differs from that of Inmon. The Kimball approach starts by identifying the key business questions which need to be answered by the data warehouse. Importantly, the sources of data for the data warehouse are well documented and analyzed. Using an ETL process, the data is extracted from the sources and inserted into a staging area. By using a staging area, the dimension tables can be loaded more efficiently. It is at this stage that there is a stark contrast to the Inmon design, the dimensions are not normalized. Kimball proposes a star schema design. Within a star schema, there is a fact table which has the purpose of holding all business measures. Surrounding the fact table there are dimension tables which are used to describe the measures. It's important to note that the fact table holds all the foreign keys to the dimension tables. By having the dimensions de-normalized, the user can explore the data without the need for joining tables. Figure 3 shows how a typical Kimball designed model looks.

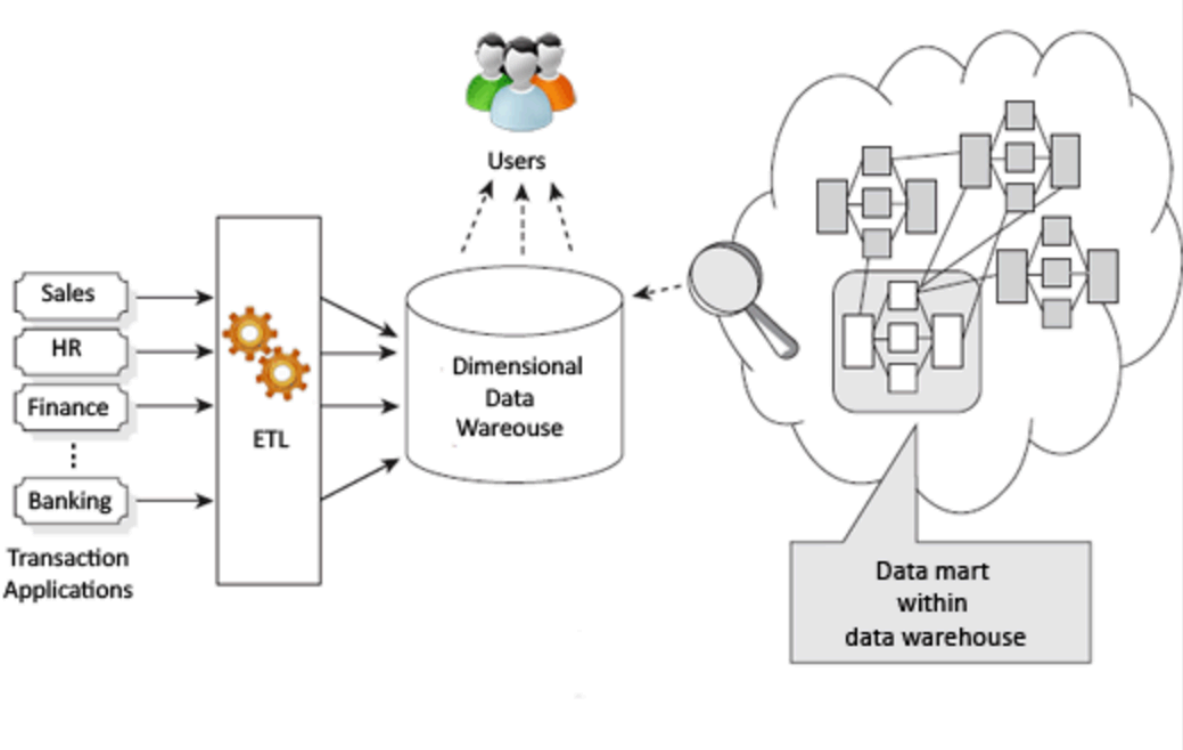


Figure 3 - Kimball Data Warehouse Model

Technologies

Most work for this project has been completed using Python. Python is a general-purpose programming language, developed by Guido Van Rossum, it was released in 1991. Python is also open source under the OSI-approved open-source license, meaning it adheres to the requirements set out by the CoderDojo Foundation. Python was used to for the ETL process which makes up the bulk of the project.

As per Python.org: *“Python is powerful... and fast; plays well with others; runs everywhere; is friendly & easy to learn; is Open.”*

Open source Pythons were also used to reduce complexity. By using existing libraries, focus could be put on the architecture of the model rather than tasks which have an already well established solution.

Additionally, technologies used include SQL. SQL is a structured query language which is used against databases to perform operations. SQL was used to extract data from sources and insert into the data warehouse.

For visualization, Tableau was used. Tableau is a visualization client which can connect to data sources and process the data into visuals which are easy to interpret. Tableau is not open-source but does have a license which charities can avail of.

As per Tableau.com: *“Tableau can help anyone see and understand their data. Connect to almost any database, drag and drop to create visualizations, and share with a click.”*

Alternative technologies explored

Several alternative technologies were explored for the execution of the project. In the early stages of the project a data warehouse was not considered. Instead, the project would execute SQL queries and graph the results through D3, an open-source Node.JS library. The CoderDojo Foundation platform, Zen, is built in Node.JS. The dashboard would have been built into Zen’s administration section.

Once it was established that several queries would not be possible with the current source schema, it was decided that a data warehouse is the best approach for the project. Several well accepted data warehousing technologies such as SQL Server, SiSense and PowerBI. However, none of these would be used for the following reasons.

1. SQL Server does not have a charitable license and would be too expensive for the CoderDojo Foundation to purchase a standard license.
2. A license for SiSense would also prove too expensive and does not have a charitable license.
3. PowerBI could have been used as it is free. However, it requires the data source to be hosted in Azure. Azure does not support PostgreSQL and as such could not be used.

Methodology

The project will be built using an iterative approach. This allows for requirements to grow and change, which is a very real possibility since organizational use cases can evolve. The iterative approach was also suitable as the nature of the project goes against everything that has been thought regarding structuring of databases. By using an iterative approach, there was scope to learn data warehousing as the project progressed.

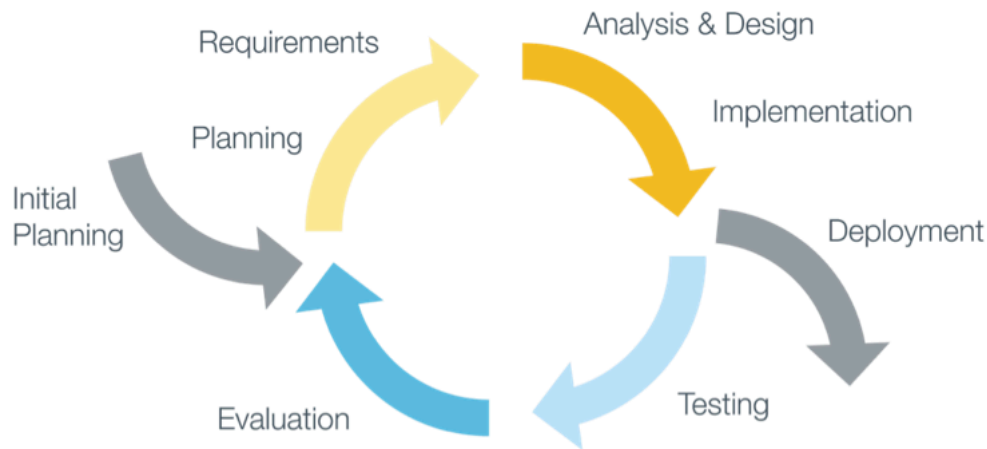


Figure 4 - Iterative Life Cycle Approach

Ihris.org explain that the process builds in frequent and regular cycles. This allows for feedback from stakeholders based on reactions from end-users to a working, although incomplete system. The approach allows the development team to further their understanding of the system as development progresses. The ultimate gain is that the final product will be one which meets the current needs of the stakeholders. Figure 4 showcases that the methodology is a cycle, right up until deployment. Each cycle showcases new features and fixes based on feedback from the previous cycle.

Early Stages

Due to the project domain being a completely new concept, early stages of planning underestimated the length of time each requirement would take to complete. It was through frequent meetings with Dr. Caton that this was cleared up. Once development began, it became clearer the length of time which would be required. Furthermore, difficult aspects of the project could be identified and time was set aside accordingly for these requirements.

Requirements gathering

Initial requirements were gathered while the student was still on work placement with the CoderDojo Foundation. The requirements were gathered through meetings to see which business measures are of importance. At this stage of requirement gathering, final measures were only gathered. No requirements of implementation or architecture design were agreed upon at this stage. It was in the next iteration of that the project started to shape up from an architectural point of view and deciding on the technologies to be used. Guillaume Feliciano, the CoderDojo Foundation Technical Lead, had the following to say regarding architecture (November 2016).

“Well, you need a process to transform the data from Zen into the new DB, that's what the ETL is for, or do you expect to simply transform the SQL dump/"flow" through triggers? If you need to consolidate with other datasources, i'd consider that writing providers w/ a pre-processing in R/python/whatever may be beneficial in term of maintenance or plug one of those ETL thingy”

Additionally, Feliciano had the following to say regarding visualization (November, 2016).

“Hey Adam, I went through your links and a couple of documents regarding BI & OSS. I don't believe any of those links you sent will suite considering i don't believe we'd go for thousands of € to get a licence/support and expose our users data.”

It was through further meetings in which a technological stack was decided upon. As mentioned in the technologies section, the stack consists primarily of Python, with use of SQL and Tableau.

User requirements definition

The CoderDojo Foundations community platform, Zen, is of vital importance to business needs. As such, key business measures come from this domain. Requirements were gathered based on the measures required by funding partners. The project has a direct link to helping the Foundation secure funding through reporting the key business measures which must be showcased as a funding requirement. As such the below preliminary statistical requirements have been set out for this project ~ July. The statistics requirements were documented by CoderDojo based on metrics required by funding partners.

Statistical Requirement 1

Statistic Name	Dojos by region, language, frequency or number.	Data Sources Required	Cp-dojos-development database.
Importance	Critical.	Target columns from source	Country, languages, count of rows.
Significance of statistic to the CoderDojo Foundation	Allows the CoderDojo Foundation to target specific regions for growth incentives.	Target dimension within data warehouse.	dimDojos dimension.

Statistical Requirement 2

Statistic Name	Users by type, region, age, events booked.	Data Sources Required	Cp-users-development database.
Importance	Critical.	Target columns from source	Type, country, dob.
Significance of statistic to the CoderDojo Foundation	Allows the CoderDojo Foundation to target specific user demographics for growth incentives.	Target dimension within data warehouse.	dimUsers dimension.

Statistical Requirement 3

Statistic Name	Tickets by user type, availability, acquisition, cancelation, checked in.	Data Sources Required	Cp-events-development database.
Importance	Important.	Target columns from source	Ticket id.
Significance of statistic to the CoderDojo Foundation	Allows the CoderDojo Foundation to see the most popular ticket types.	Target dimension within data warehouse.	dimTickets dimension.

On top of the statistical requirements, the below functional requirements have also been set out.

Functional Requirement 1

Requirement Name	Data only accessible to admins.	Data Sources Required.	N/A.
Importance	Critical.	How requirement will be demonstrated.	Only those with database login can access data. Only those with Tableau account can access KPI dashboard.
Key Stakeholders	Development team.	Description.	The system must not allow users other than CoderDojo Foundation members to access the KPI dashboard.

Functional Requirement 2

Requirement Name	Filtering of data.	Data Sources Required.	<ol style="list-style-type: none"> 1. Dojos database. 2. Events database. 3. Users database.
Importance	Critical.	How requirement will be demonstrated.	Applying different descriptive from dimension onto a fact from the fact table.
Key Stakeholders	Reporting Lead.	Description.	The dashboard must be filterable using dimensions.

Functional Requirement 3

Requirement Name	Prediction modelling.	Data Sources Required.	<ol style="list-style-type: none"> 1. Dojos database. 2. Events database. 3. Users database.
Importance	Low.	How requirement will be demonstrated.	Demonstrated through the Tableau dashboard.
Key Stakeholders	Reporting Lead.	Description.	The dashboard should be able to predict future attribute values based on historical data.

Functional Requirement 4

Requirement Name	Adding external data sources.	Data Sources Required.	<ol style="list-style-type: none"> 1. Dojos database. 2. Events database. 3. Users database.
Importance	Low.	How requirement will be demonstrated.	Create a new dimension and insert data from an API.
Key Stakeholders	Reporting Lead.	Description.	The data warehouse should have the ability to create additional dimensions from external data sources.

Functional Requirement 5

Requirement Name	Display static statistics.	Data Sources Required.	<ol style="list-style-type: none"> 1. Dojos database. 2. Events database. 3. Users database.
Importance	Critical.	How requirement will be demonstrated.	Demonstrated through the Tableau dashboard.
Key Stakeholders	Reporting Lead.	Description.	The dashboard should display descriptives such as count and average.

Environmental requirements

The data warehouse must be able to be hosted on an Amazon EC2 instance. Additionally, the visualization software must be compatible with Mac OS. There are no Windows requirements as the CoderDojo Foundations uses MacOS and Linux variants.

Use case diagram

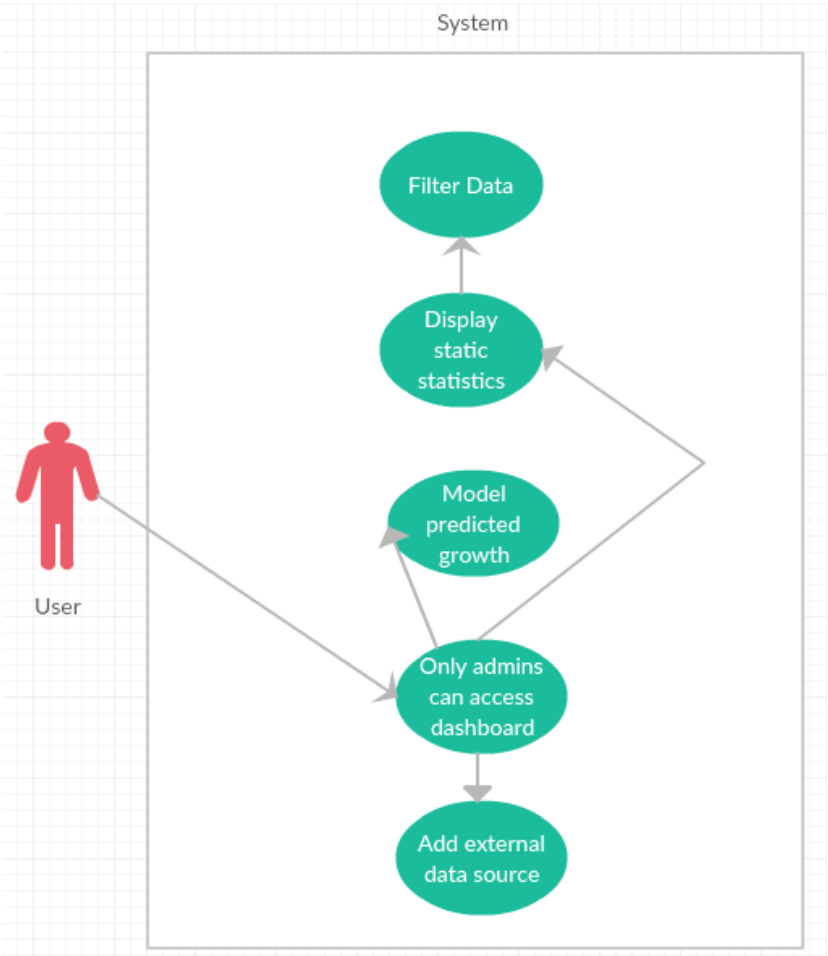


Figure 5 - System Use Case Diagram

Architectural view

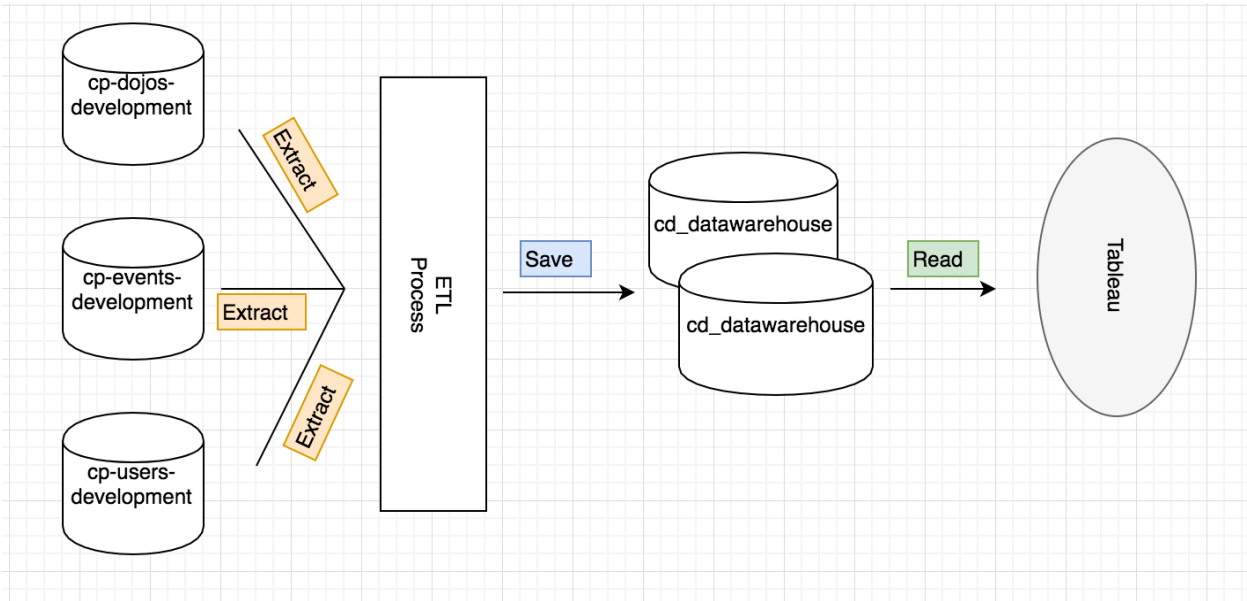


Figure 6 - System Architecture

Figure six depicts the system architecture. There are three sources databases. The project starts by reading the target data from each source on a row by row basis. Each row is passed into a transformation function. This is known as the ETL process. The purpose of the transformation function is to modify the data to make it more suitable for persistence within the data warehouse. An example of this is extracting values out of the source columns which are saved in JSON format. Once the data warehouse is loaded, a connection is made to it from Tableau. Tableau consumes the data warehouse such that the data can be used for visualization.

Data warehouse dimensions

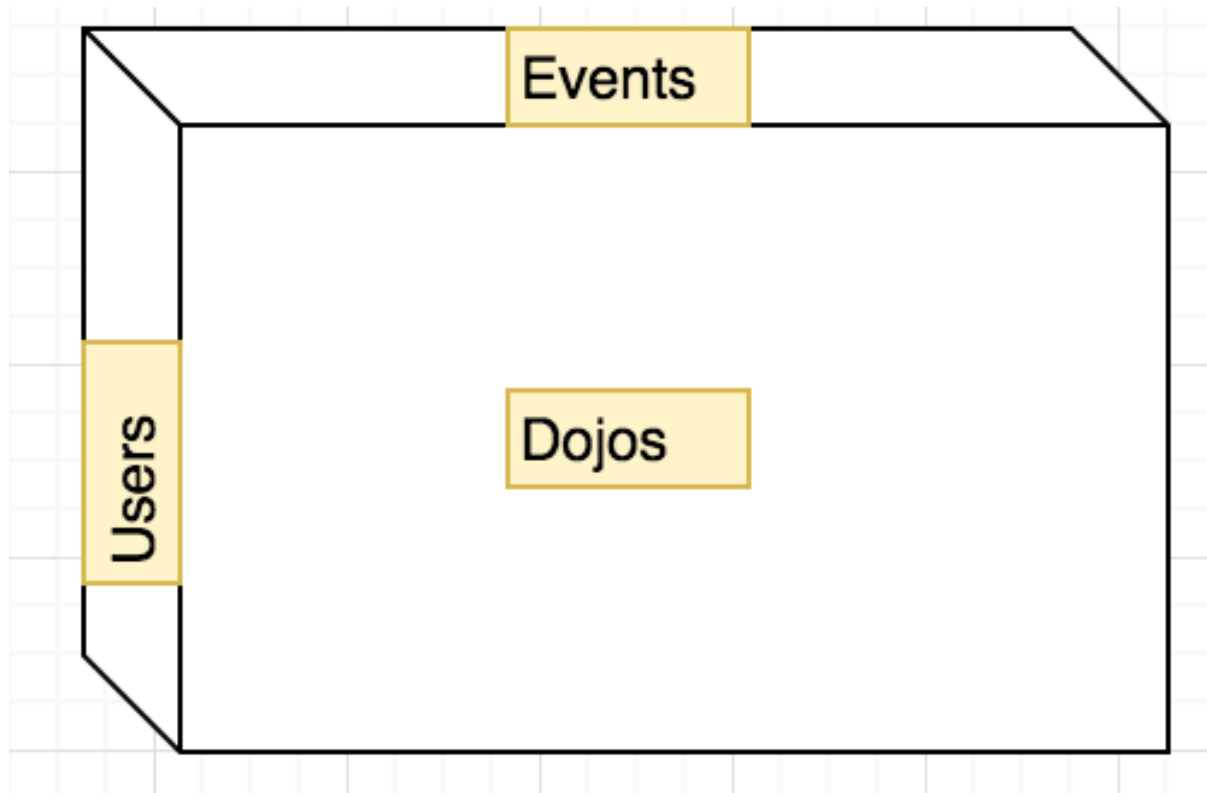


Figure 7 - Data Warehouses Cube

Figure 7 shows the data warehouse model. The three primary dimensions are users, events and dojos. These can be represented in a cube model with each side taking up a different dimension. By using this model, it allows statistics to be gathered by slicing the cube up, per say. It allows for more complex querying which spans across multiple dimensions. An example would which user attended how many events pertaining to Dojos which are in Ireland. Previously this would be impossible as the data was spread across three different databases, with no relationships between them.

Implementation

Primary aspects of the project will be described through code snippets. All the code was developed through Visual Studio Code, a lightweight open source text editor.



Figure 8 - Visual Studio Code

The primary use case of the project involves reading data from the three source databases.

1. Cp-Events-Development.
2. Cp-Dojos-Development.
3. Cp-Users-Development.

Figure 8 shows the three databases from PgAdmin, an open-source client for PostgreSQL. As can be seen, having the data persisted in three different databases is not ideal for analysis. It makes gathering measures which span several dimensions extremely difficult. Additionally, PostgreSQL does not support cross database queries. Cross schema queries would work but that is not how the source is saved.




- ▶  cp-dojos-development
- ▶  cp-events-development
- ▶  cp-users-development

Figure 9 - Sources of Data

Setting up the environment

The first step is to install the required tools for the project. These tools are:

1. Homebrew.
2. PostgreSQL.
3. PgAdmin3.
4. Python virtual environment.

All the tools can be installed by executing the following commands:

```
#Install homebrew
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

#Install PostgreSQL
brew install postgresql

#Install PgAdmin
brew cask install pgadmin3

#Install python virtual environment
sudo easy_install pip
sudo pip install virtualenv
```

Figure 10 - Installation of Required Tools

Once these tools have been installed, three empty databases need to be created in PostgreSQL. It is important to name these three databases as follows. The databases can be created through the terminal or through PgAdmin. For convenience, these instructions show how to create the databases through PgAdmin.

1. cp-events-development.
2. cp-dojos-development.
3. cp-users-development.

The first step is to open PgAdmin. Once it is open:

1. Right click on databases.
2. Click on 'new database'.
3. Name the database and click 'ok'.

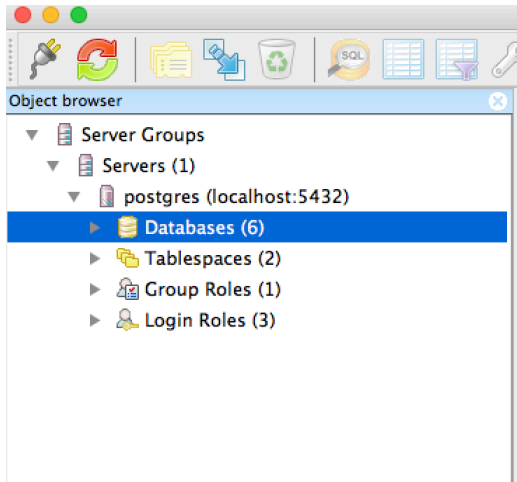


Figure 11 - Right Click on Databases

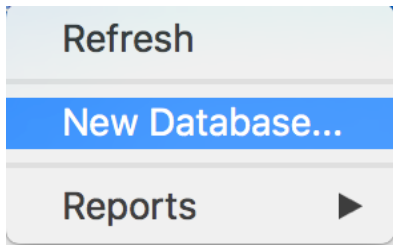


Figure 12 - Click on New Database

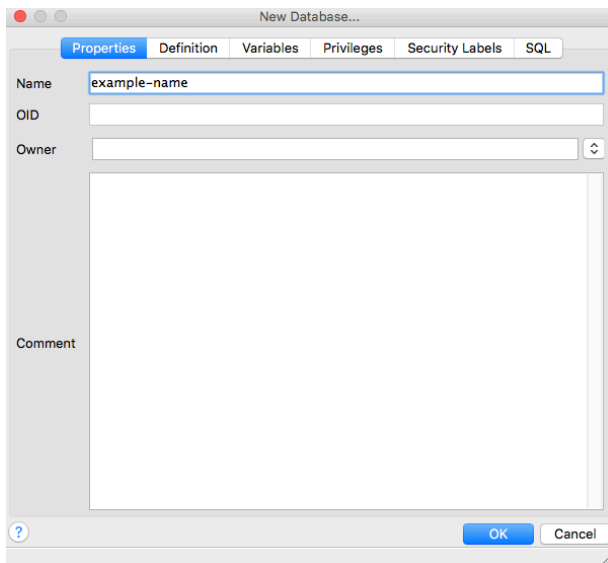
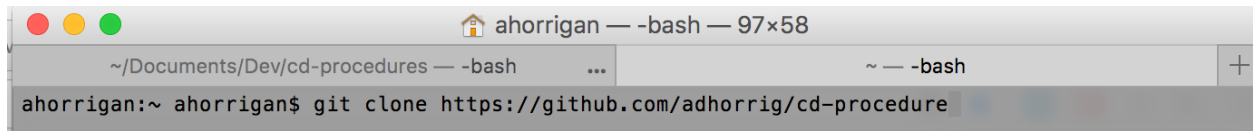


Figure 13 - Name the Database and Click ok

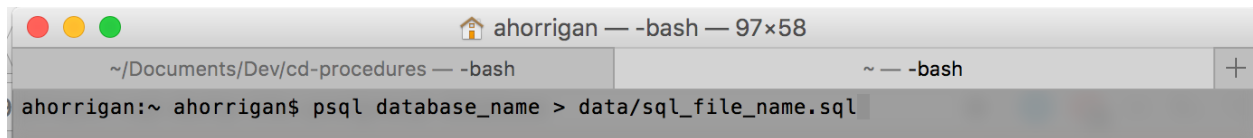
Once the three empty databases have been created, the project repository will need to be cloned from Github by running the command in Figure 14 from the terminal.



```
ahorrigan — -bash — 97x58
~/Documents/Dev/cd-procedures — -bash ... ~ — -bash
ahorrigan:~ ahorrigan$ git clone https://github.com/adhorrig/cd-procedure
```

Figure 14 – Clone Repository

Cloning the repository will create a folder named ‘cd-procedures’. This folder contains all project files and folders. Within this folder there is a ‘data’ folder which contains the three CoderDojo Foundation source databases along with content. These three database dumps need to be imported into the three newly created databases from the previous steps. This can be achieved using PSQL on the command line. Execute the three commands in Figure 15 from the terminal.



```
ahorrigan — -bash — 97x58
~/Documents/Dev/cd-procedures — -bash ~ — -bash
ahorrigan:~ ahorrigan$ psql database_name > data/sql_file_name.sql
```

Figure 15 - Import Data Dump

It is important to note that ‘database_name’ must match one of the empty database names which was created earlier. Sql_file_name must correspond to that database. Consider the below examples:

```
psql cp-dojos-development > data/dojosdbdump.sql
psql cp-events-development > data/eventsdbdump.sql
psql cp-users-development > data/usersdbdump.sql
```

As the users’ database is the largest, it will take the longest to import. At this stage, all local databases have been populated with the CoderDojo Foundation source data. The next step is to activate the python virtual environment and install the project requirements. This is done by executing the following commands from the terminal.

```
Virtualenv env
Source env/bin/activate
Pip install uuid
Pip install psychopg2
Pip install isodate
```

The final step is to create another empty database named ‘cd_datawarehouse’ and import the sql/dw.sql file.

Connecting to the sources

The script needs to connect to the data sources to perform the ETL process. All connection details are stored in a .JSON configuration file. This means additional sources can be easily added in the future and changes can easily be made as it does not need to be done multiple times.

```
1  [
2  "databases": {
3    "dojos": "cp-dojos-development",
4    "users": "cp-users-development",
5    "events": "cp-events-development",
6    "dw": "cd_datawarehouse"
7  },
8  "config": {
9    "user": "postgres",
10   "password": "",
11   "host": "localhost"
12 }
13 ]
```

Figure 16 - Configuration Details

The configuration details are then consumed by the main script and used to connect to each data source. This is done one by one.

```
10 with open('./config.json') as data:
11     data = json.load(data)
12
13     dojos = data['databases']['dojos']
14     users = data['databases']['users']
15     events = data['databases']['events']
16     dw = data['databases']['dw']
17     user = data['config']['user']
18     password = data['config']['password']
19     host = data['config']['host']
20
21     #connection strings
22     dojos_string = "host="+host+" dbname="+dojos+" user="+user+" password="+password
23     users_string = "host="+host+" dbname="+users+" user="+user+" password="+password
24     events_string = "host="+host+" dbname="+events+" user="+user+" password="+password
25     dw_string = "host="+host+" dbname="+dw+" user="+user+" password="+password
26
27     # print connection strings
28     print "Connecting to database\n    ->%s" % (dojos_string)
29     print "Connecting to database\n    ->%s" % (users_string)
30     print "Connecting to database\n    ->%s" % (events_string)
31     print "Connecting to database\n    ->%s" % (dw_string)
32
33     # get a connection
34     dojos_conn = psycopg2.connect(dojos_string)
35     users_conn = psycopg2.connect(users_string)
36     events_conn = psycopg2.connect(events_string)
37     dw_conn = psycopg2.connect(dw_string)
38
39     # use cursor with queries
40     dojos_cursor = dojos_conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
41     users_cursor = users_conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
42     events_cursor = events_conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
43     dw_cursor = dw_conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
44     print "Connected to all databases!\n"
```

Figure 17 - Connecting to Sources

Referring to figure 17, line 10-12 opens the .JSON configuration file from figure sixteen and stores the data in variables named data. Lines 13-19 enter the .JSON and assign each key from the file to its own independent variable. Line 22-25, connection strings are setup using the previously declared variables. These strings are printed to the console to let the user know the script is connecting to the database. Line 34-43, connects to each individual database using the connection details which are stored in the previous strings. Using the connection, a cursor is setup. The cursor allows the script to execute a query and iterate over the results row by row. Finally, a success message is logged to let the user know all connections were successful.

Main method

The very first thing that the main method does is truncate all existing tables. The reason for this is that each time the script is ran, it will error out as the primary key being inserted already exists in each table. Truncating all tables before performing the inserts solves this issue. Alternative solutions include checking the primary key on a row by row basis and only performing the insert if there is no conflict.

```
#Truncate all tables before fresh insert from sources
dw_cursor.execute('TRUNCATE TABLE "factUsers" CASCADE')
dw_cursor.execute('TRUNCATE TABLE "dimDojos" CASCADE')
dw_cursor.execute('TRUNCATE TABLE "dimUsers" CASCADE')
dw_cursor.execute('TRUNCATE TABLE "dimEvents" CASCADE')
dw_cursor.execute('TRUNCATE TABLE "dimLocation" CASCADE')
dw_cursor.execute('TRUNCATE TABLE "dimTime" CASCADE')
dw_cursor.execute('TRUNCATE TABLE "dimTickets" CASCADE')
dw_cursor.execute('TRUNCATE TABLE "zen-source"."staging" CASCADE')
```

The main method is used to query the source databases to get the target data.

```
48 | def main():
49 |
50 |     #Queries - Dojos
51 |     dojos_cursor.execute('SELECT * FROM cd_dojos where verified = 1 and deleted = 0 and stage != 4')
52 |     for row in dojos_cursor:
53 |         transformDojo(row)
54 |     print("Inserted all dojos")
55 |
56 |     #Queries - Events
57 |     events_cursor.execute('SELECT * FROM cd_events')
58 |     for row in events_cursor:
59 |         transformEvent(row)
60 |     print("Inserted all events and locations")
61 |
62 |     #Queries - Users
63 |     users_cursor.execute('select * from cd_profiles inner join sys_user on cd_profiles.user_id = sys_user.id')
64 |     for row in users_cursor:
65 |         transformUser(row)
66 |     print("Inserted all users")
67 |
68 |     #Queries - Tickets
69 |     events_cursor.execute('select cd_sessions.id as session_id, event_id, status, cd_tickets.id as ticket_id, type, quantity from cd_sessions inner join cd_tickets
70 |     for row in events_cursor:
71 |         transformTicket(row)
72 |     print("Inserted all tickets")
73 |
74 |     #Queries - Staging
75 |     events_cursor.execute('select cd_applications.id, cd_applications.ticket_id, cd_applications.session_id, cd_applications.event_id, cd_applications.dojo_id, cd_
76 |     for row in events_cursor:
77 |         staging(row)
78 |     print('Populated staging')
79 |
80 |     #Populate Measures
81 |     dw_cursor.execute('select staging.dojo_id, staging.ticket_id, staging.session_id, staging.event_id, staging.user_id, staging.time_id, staging.location_id from
82 |     for row in dw_cursor.fetchall():
83 |         measures(row)
84 |     print("Inserted measures")
85 |
```

Figure 18 - Gathering Source Data

Each block selects target data from the source databases. The script then iterates over each row in the result set. Each row is fed into a transformation function. After all rows have been transformed, a string is printed to let the user know.

Transformation functions

The purpose of the transformation functions is to transform the data such that they become more appropriate for analysis. Figure 19 shows example rows from the CoderDojo Foundation Users source data.

badges json[]	countryname character varying	countrynumber character varying	continent character varying
	Taiwan, Province of China	158	AS
{"id":45,"slug":"my-1st-dojol","name":"My 1st Dojo!","strapline":\			EU
	Portugal	620	EU
	Greece	300	EU
	Ireland	372	EU
{"id":45,"slug":"my-1st-dojol","name":"My 1st Dojo!","strapline":\			EU
			EU
			NA
			EU
	Spain	724	EU
{"id":45,"slug":"my-1st-dojol","name":"My 1st Dojo!","strapline":\			NA
			NA
			EU
			OC
	Martinique	474	NA
	Ireland	372	EU

Figure 19 - Users Source Data

The data is not ideal for analysis. There are missing values from a lot of the rows and some columns are being saved in JSON arrays. The transformation functions make the data more useable and readable. It also deals with missing and null values. Figure 20 shows one of the transformation functions.

```

86 def transformDojo(row): #Transform / Load for Dojo Dimension
87     dojo_id = row['id']
88     created_at = row['created']
89     verified_at = row['verified_at']
90     stage = row['stage']
91     country = row['country'] if (row['country'] is not None) and (len(row['country'])) > 0 else 'Unknown'
92     city = row['place'] if (row['city'] is not None) and (len(row['city'])) > 0 else 'Unknown'
93     county = row['county'] if (row['county'] is not None) and (len(row['county'])) > 0 else 'Unknown'
94     state = row['state'] if (row['state'] is not None) and (len(row['state'])) > 0 else 'Unknown'
95     continent = row['continent']
96     tao_verified = row['tao_verified']
97     expected_attendees = row['expected_attendees'] if (row['expected_attendees'] is not None) else 0
98     verified = row['verified']
99     deleted = row['deleted']
100
101     #For fields which zen prod dbs are storing as json
102     if country is not 'Unknown':
103         country = country['countryName']
104
105     if city is not 'Unknown':
106         city = city['name']
107
108     if county is not 'Unknown':
109         county = county['toponymName']
110
111     if state is not 'Unknown':
112         state = state['toponymName']
113
114     insertDojo(dojo_id, created_at, verified_at, stage, country, city, county, state, continent, tao_verified, expected_attendees, verified, deleted)

```

Figure 20 - Transformation Function

Line 87-99 extracts out values for each column and assigns the data to a variable. Depending on how the variable is saved in the source, the data may need to be cleaned. For columns which are not mandatory, instead of saving a blank field, the word 'Unknown' is used. This allows to CoderDojo Foundation to see proportionally, how much of their dataset is unknown. This could lead to a decision to enforce more fields on the front end of the applications.

Line 101-112 is for rows which the source is saving as JSON or inside a JSON array. JSON is not very efficient for analysis and as such, the values for the interesting keys are extracted out and saved to variables.

Finally, line 114 calls an insert function with each variable which was extracted out from the row. The transformation is of utmost important as it cleans the data. Without this, the data would not be able to be analyzed.

Insert functions

The purpose of the insert function is to receive data from the transformation function and insert it into the appropriate table inside the data warehouse.

```
174 def insertDojo(dojos_id, created_at, verified_at, stage, country, city, county, state, continent, tao_verified, expected_attendees, verified, deleted):
175     sql = '''
176         INSERT INTO "public"."dimDojos"(
177             id,
178             created_at,
179             verified_at,
180             stage,
181             country,
182             city,
183             county,
184             state,
185             continent,
186             tao_verified,
187             expected_attendees,
188             verified,
189             deleted)
190         VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
191     '''
192     data = (dojos_id, created_at, verified_at, stage, country, city, county, state, continent, tao_verified, expected_attendees, verified, deleted)
193     try:
194         dw_cursor.execute(sql, data)
195         dw_conn.commit()
196     except (psycopg2.Error) as e:
197         print(e)
198
```

Figure 21 - Insert Function

Line 172-192 is the SQL command which will be executed to insert the data into the data warehouse table. The values are '%s' are for security reasons. This is a prepared statement and prevents SQL injection. Line 194-198 attempts to execute the query through a try-catch. If for any reason, the query generates an error – the error will be caught and logged to the user. Otherwise, the query will successfully execute.

Starting the main method

Figure 22 shows the starting point when the application is run.

```
490 if __name__ == '__main__':
491     main()
```

Figure 22 - Starting the script

Evaluation and testing

The solution to this project is rather unique and very manual. Having the constraint of using open source software made the project tenfold more difficult. This was especially true when it came to populating the fact table. In any case, I believe the project can be considered based on looking the schemas to the original three sources of data. The below images showcase the schema for the three sources of data.

cd_events	
id	CHARACTER VARYING
name	CHARACTER VARYING
country	JSON
city	JSON
address	CHARACTER VARYING
created_at	TIMESTAMP(6) WITH TIME ZONE
created_by	CHARACTER VARYING
type	CHARACTER VARYING
description	CHARACTER VARYING
dojo_id	CHARACTER VARYING
position	JSON
public	BOOLEAN
status	CHARACTER VARYING
recurring_type	CHARACTER VARYING
dates	JSON[]
ticket_approval	BOOLEAN

cd_applications	
id	CHARACTER VARYING
name	CHARACTER VARYING
date_of_birth	DATE
event_id	CHARACTER VARYING
status	CHARACTER VARYING
user_id	CHARACTER VARYING
ticket_name	CHARACTER VARYING
ticket_type	CHARACTER VARYING
session_id	CHARACTER VARYING
created	TIMESTAMP(6) WITH TIME ZONE
deleted	BOOLEAN
attendance	TIMESTAMP WITH TIME ZONE[]
dojo_id	CHARACTER VARYING
ticket_id	CHARACTER VARYING
notes	CHARACTER VARYING

cd_tickets	
id	CHARACTER VARYING
session_id	CHARACTER VARYING
name	CHARACTER VARYING
type	CHARACTER VARYING
quantity	INTEGER
deleted	SMALLINT
invites	JSON[]

cd_sessions	
id	CHARACTER VARYING
name	CHARACTER VARYING
description	CHARACTER VARYING
event_id	CHARACTER VARYING
status	CHARACTER VARYING

schemaversion	
version	INTEGER
name	TEXT
md5	TEXT

Figure 23 - Events Schema

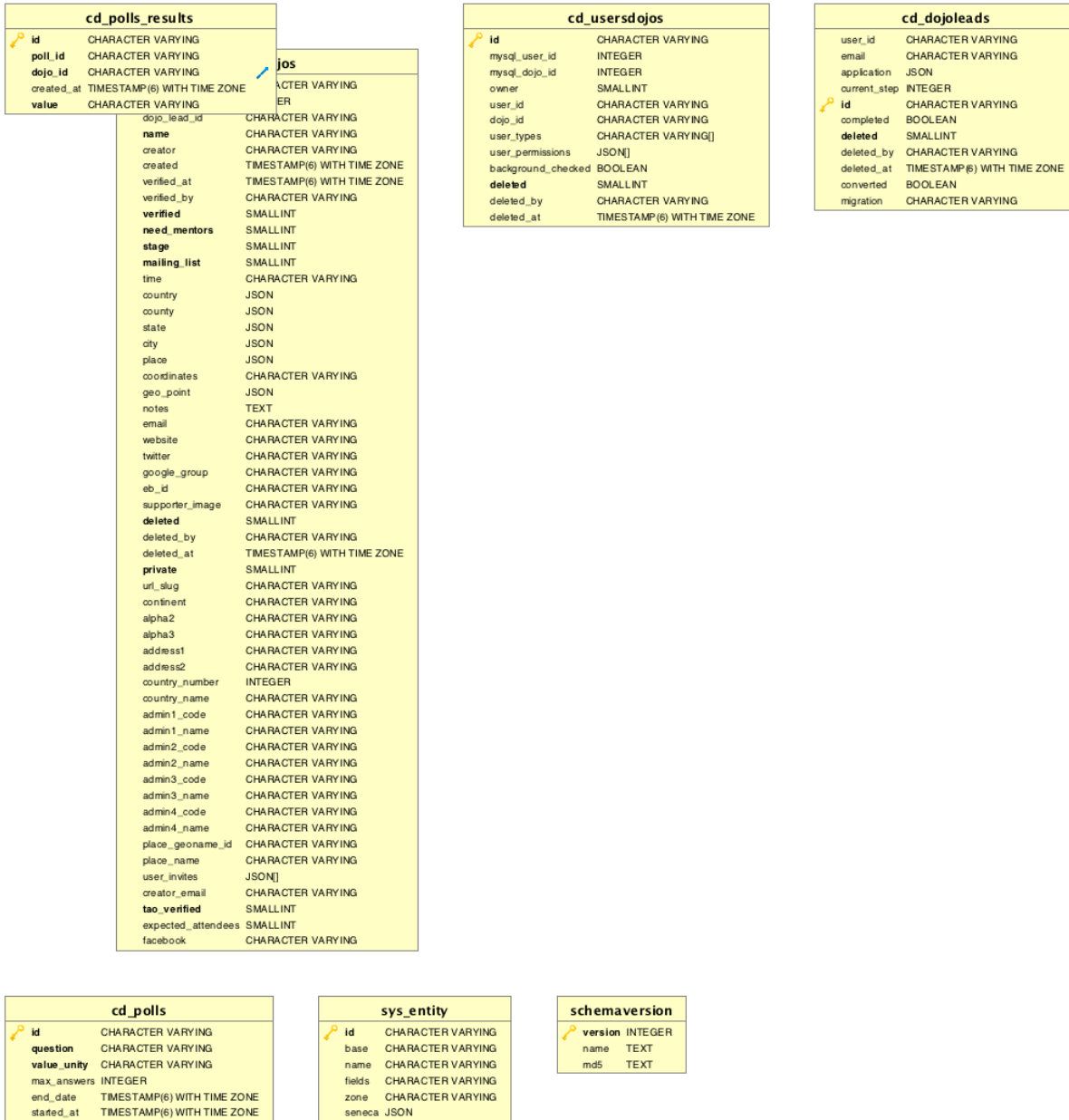


Figure 24 - Dojos Schema

cd_profiles	
id	CHARACTER VARYING
name	CHARACTER VARYING
user_id	CHARACTER VARYING
alias	CHARACTER VARYING
dob	TIMESTAMP(6) WITH TIME ZONE
country	JSON
city	JSON
private	BOOLEAN
place	JSON
gender	CHARACTER VARYING
address	CHARACTER VARYING
last_edited	TIMESTAMP(6) WITHOUT TIME ZONE
email	CHARACTER VARYING
phone	CHARACTER VARYING
parents	TEXT[]
children	TEXT[]
linkedin	CHARACTER VARYING
twitter	CHARACTER VARYING
languages_spoken	TEXT[]
programming_languages	TEXT[]
notes	CHARACTER VARYING
projects	CHARACTER VARYING
badges	JSON[]
countryname	CHARACTER VARYING
countrynumber	CHARACTER VARYING
continent	CHARACTER VARYING
alpha2	CHARACTER VARYING
alpha3	CHARACTER VARYING
admin1_code	CHARACTER VARYING
admin1_name	CHARACTER VARYING
admin2_code	CHARACTER VARYING
admin2_name	CHARACTER VARYING
admin3_code	CHARACTER VARYING
admin3_name	CHARACTER VARYING
admin4_code	CHARACTER VARYING
admin4_name	CHARACTER VARYING
state	JSON
county	JSON
place_geoname_id	CHARACTER VARYING
place_name	CHARACTER VARYING
user_type	CHARACTER VARYING
optional_hidden_fields	JSON
avatar	JSON
required_fields_complete	BOOLEAN
ninja_invites	JSON[]
parent_invites	JSON[]

sys_user	
id	CHARACTER VARYING
nick	CHARACTER VARYING
email	CHARACTER VARYING
name	CHARACTER VARYING
username	CHARACTER VARYING
activated	SMALLINT
level	SMALLINT
mysql_user_id	INTEGER
first_name	CHARACTER VARYING
last_name	CHARACTER VARYING
roles	CHARACTER VARYING[]
active	BOOLEAN
phone	CHARACTER VARYING
mailing_list	SMALLINT
terms_conditions_accepted	BOOLEAN
when	TIMESTAMP(6) WITH TIME ZONE
confirmed	BOOLEAN
confirmcode	CHARACTER VARYING
salt	CHARACTER VARYING
pass	CHARACTER VARYING
admin	BOOLEAN
modified	TIMESTAMP(6) WITH TIME ZONE
accounts	CHARACTER VARYING[]
locale	CHARACTER VARYING
banned	SMALLINT
ban_reason	CHARACTER VARYING
int_user_type	CHARACTER VARYING
join_requests	JSON[]
last_bgin	TIMESTAMP(6) WITH TIME ZONE
ims_id	CHARACTER VARYING

sys_login	
id	CHARACTER VARYING
nick	CHARACTER VARYING
email	CHARACTER VARYING
user	CHARACTER VARYING
when	TIMESTAMP(6) WITH TIME ZONE
why	CHARACTER VARYING
token	CHARACTER VARYING
active	BOOLEAN
auto	BOOLEAN
ended	CHARACTER VARYING

cd_agreements	
mysql_user_id	INTEGER
full_name	CHARACTER VARYING
ip_address	CHARACTER VARYING
timestamp	TIMESTAMP(6) WITH TIME ZONE
agreement_version	SMALLINT
user_id	CHARACTER VARYING
id	CHARACTER VARYING

sys_reset	
id	CHARACTER VARYING
nick	CHARACTER VARYING
user	CHARACTER VARYING
when	TIMESTAMP(6) WITH TIME ZONE
active	BOOLEAN
token	CHARACTER VARYING

sys_account	
id	CHARACTER VARYING
orignick	CHARACTER VARYING
name	CHARACTER VARYING
origuser	CHARACTER VARYING
active	BOOLEAN
users	CHARACTER VARYING

sys_entity	
id	CHARACTER VARYING
base	CHARACTER VARYING
name	CHARACTER VARYING
fields	CHARACTER VARYING
zone	CHARACTER VARYING
seneca	JSON

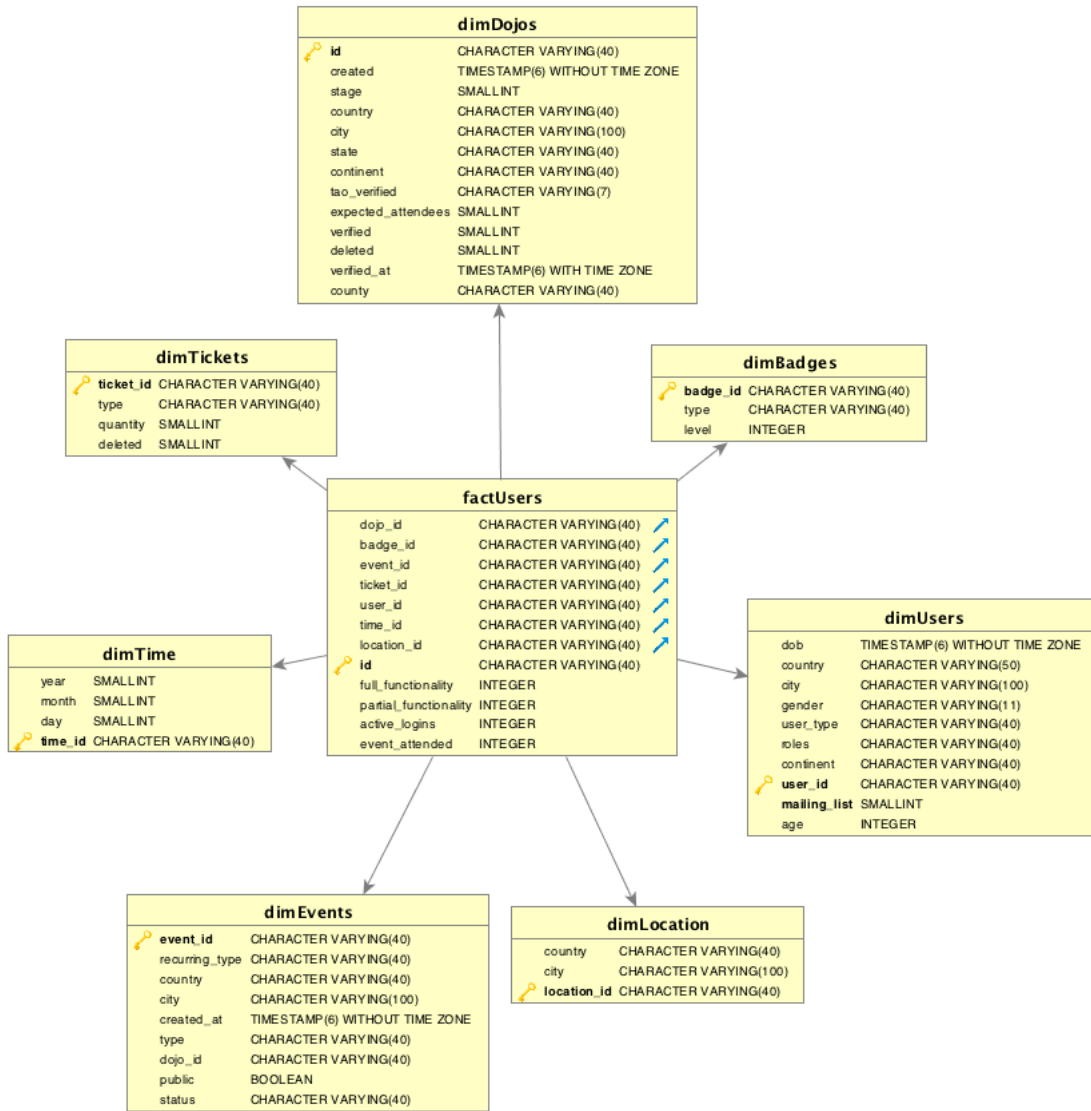
cd_oauth2	
id	CHARACTER VARYING
userid	CHARACTER VARYING
code	CHARACTER VARYING
token	CHARACTER VARYING
created	TIMESTAMP(6) WITH TIME ZONE

schemaversion	
version	INTEGER
name	TEXT
md5	TEXT

Figure 25 - Users Schema

To go from these three source schemas, to the data warehouse schema in Figure 26, would be considered a success. The previous schemas have little to no relationships set up, contains a lot of unwanted data and contains data in formats which is bad for analyzing.

Figure 26 - Data Warehouse Schema



Client feedback

The following feedback is directly from Rosa Langhammer, the reporting lead of the CoderDojo Foundation.

“CoderDojo built a Community Platform, Zen, in 2015 which serves to help our community members to organize and run their Dojo from one central place as opposed to using a variety of free tools as was happening previously. A central platform was also needed to help the movement assess its impact as the movement has now scaled to over 1200 clubs in 70 countries with 10's of thousands of young people attending Dojos globally. Unfortunately, with limited internal resources and a higher priority of user facing issues we were never able to create a specific statistics dashboard for our internal team to check in on our progress regularly. Adams specification from CoderDojo was to build something reliable which we could use to visualize statistics for the entire Foundation team. It was also to build something scalable which we could add to when we want to track new metrics in the future. Adam's final solution has met the entirety of the specification and will be incredibly useful for us as a team to analyze our impact, allowing us to become more effective at scaling, engaging and enabling our community members and consequently growing the CoderDojo movement worldwide.”

Testing methodology

In the final iterations of the project, testing was implemented. Two primary testing methodologies were used.

1. Unit testing.

Unit testing involves breaking the script into smaller pieces, or units and running a set of tests against that specific unit. Unit testing is used to confirm whether a unit of code is working as expected, or not. Unit tests are relatively straight forward and are typically written as functions. The unit test, tests weather the returned values equals that which was expected.

The following pages show the results of the tests which were carried out.

Unit Test 1.

Test ID	UT1.
Purpose of Test	<p>When reading from the source databases, many columns are encoded in JSON.</p> <p>The purpose of this test is to ensure that the application can extract the values from the specified key.</p> <p>This ensures that no JSON is inserted into the data warehouse.</p>
Test Environment	<p>The test was carried out in the following environment:</p> <p>Operating System: OS X El Capitan, Version 10.11.6. Python Version: 2.7.10. Test data: Source data from the CoderDojo Foundation production database.</p>
Test Steps	<p>From terminal, the tester should:</p> <p>Navigate to the directory where the project has been downloaded. Execute 'python test.py' The results will be logged to the terminal.</p>
Expected Result	<p>After the tests have been completed by running the above steps, the data assertEquals() method should pass.</p> <p>This ensures that that the data was correctly extracted out of the JSON.</p>
Actual Result	<p>The data was successfully extracted and the test passed.</p>

Unit Test 2.

Test ID	UT2.
Purpose of Test	<p>When reading from the source databases, several fields are empty. This is due to the CoderDojo platform not making the field required, on the front-end application.</p> <p>The purpose of this test is to ensure that the application handles these empty fields correctly.</p> <p>This ensures that there are no empty fields within the data warehouse dimensions.</p>
Test Environment	<p>The test was carried out in the following environment:</p> <p>Operating System: OS X El Capitan, Version 10.11.6. Python Version: 2.7.10. Test data: Source data from the CoderDojo Foundation production database.</p>
Test Steps	<p>From terminal, the tester should:</p> <p>Navigate to the directory where the project has been downloaded. Execute 'python test.py' The results will be logged to the terminal.</p>
Expected Result	<p>After the tests have been completed by running the above steps, the data assertEquals() method should pass.</p> <p>This ensures that that the empty field was replaced with the value 'Unknown'. This gives the CoderDojo Foundation team more insight.</p>
Actual Result	<p>The empty field was successfully updated and the test passed.</p>

Unit Test 3.

Test ID	UT3.
Purpose of Test	<p>The purpose of this test is to ensure that the script can successfully connect to the local PostgreSQL server.</p> <p>Without being able to connect to the initial data sources, no extraction, transformation and loading of the data would be possible.</p>
Test Environment	<p>The test was carried out in the following environment:</p> <p>Operating System: OS X El Capitan, Version 10.11.6. Python Version: 2.7.10. Server: PostgreSQL running locally with password set on admin account.</p>
Test Steps	<p>From terminal, the tester should:</p> <p>Navigate to the directory where the project has been downloaded. Execute 'python test.py' The results will be logged to the terminal.</p>
Expected Result	<p>After the tests have been completed by running the above steps, the data assertFail() method should pass.</p> <p>This means that the application is successfully able to connect to the initial data source.</p>
Actual Result	<p>The application successfully connected to the source database.</p>

Unit Test 4.

Test ID	UT4.
Purpose of Test	<p>The purpose of this test is to ensure that the script can successfully read from the data source after making a successful connection.</p> <p>Without being able to read from the sources, the transformation and loading would not be possible.</p>
Test Environment	<p>The test was carried out in the following environment:</p> <p>Operating System: OS X El Capitan, Version 10.11.6. Python Version: 2.7.10. Server: PostgreSQL running locally with password set on admin account.</p>
Test Steps	<p>From terminal, the tester should:</p> <p>Navigate to the directory where the project has been downloaded. Execute 'python test.py' The results will be logged to the terminal.</p>
Expected Result	<p>After the tests have been completed by running the above steps, the data assertFail() method should pass.</p> <p>This means that the application is successfully able to read from the initial data source after connecting to it.</p>
Actual Result	<p>The application successfully read from the source database.</p>

Unit Test 5.

Test ID	UT5.
Purpose of Test	<p>The purpose of this test is to ensure that the script can successfully insert to the local PostgreSQL server.</p> <p>Without being able to insert to a database, no data would be able to be loaded into the data warehouse.</p>
Test Environment	<p>The test was carried out in the following environment:</p> <p>Operating System: OS X El Capitan, Version 10.11.6. Python Version: 2.7.10. Server: PostgreSQL running locally with password set on admin account.</p>
Test Steps	<p>From terminal, the tester should:</p> <p>Navigate to the directory where the project has been downloaded. Execute 'python test.py' The results will be logged to the terminal.</p>
Expected Result	<p>After the tests have been completed by running the above steps, the data assertFail() method should pass.</p> <p>This means that the application is successfully able to insert to the database</p>
Actual Result	<p>The application successfully inserted to the database.</p>

The below Figure shows the code which is used to execute the unit tests.

```
def makeConnection():
    dojos_string = "host=localhost"+" dbname=cp-dojos-development"+" user=postgres"+" password="
    dojos_conn = psycopg2.connect(dojos_string)
    dojos_cursor = dojos_conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
    return dojos_cursor

def readFromTable():
    cursor = makeConnection()
    return cursor.execute('select * from cd_dojos limit 10')

def insertRow():
    dojos_string = "host=localhost"+" dbname=cp-dojos-development"+" user=postgres"+" password="
    dojos_conn = psycopg2.connect(dojos_string)
    dojos_cursor = dojos_conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
    sql = 'insert into cd_dojos (id, stage, deleted, verified, name) values (\test-test-1-1-1-test\, 4, 1, 0, \test\)'
    dojos_cursor.execute(sql)
    return dojos_conn.commit

class EqualityTest(unittest.TestCase):

    #Testing extracting value from json column
    def testJson(self):
        json_data = {"countryName":"Ireland","countryNumber":"372","continent":"EU","alpha2":"IE","alpha3":"IRL"}
        country = json_data['countryName']
        self.assertEqual('Ireland', country)

    #Testing fields which are None
    def testNone(self):
        data = None
        test = data['test'] if (data is not None) else 'Unknown'
        self.assertEqual('Unknown', test)

    #Test database connection
    def testConnection(self):
        try:
            makeConnection()
        except (psycopg2.Error) as e:
            self.fail("makeConnection() raised an error and could not connect to the database")

    #Test reading from connection
    def testRead(self):
        try:
            readFromTable()
        except (psycopg2.Error) as e:
            self.fail("readFromTable() rasied an error and could not read from the dojos table")

    #Test inserting to database
    def testInsert(self):
        try:
            insertRow()
        except (psycopg2.Error) as e:
            self.fail("insertRow() rasied an error and could not insert to the dojos table")

if __name__ == '__main__':
    unittest.main()
```

Figure 27 - Unit Tests

Non-Trivial Queries

The below images showcase some of the non-trivial queries that have come because of this project. Due to the nature of the source data schemas, the below queries would simply be impossible.

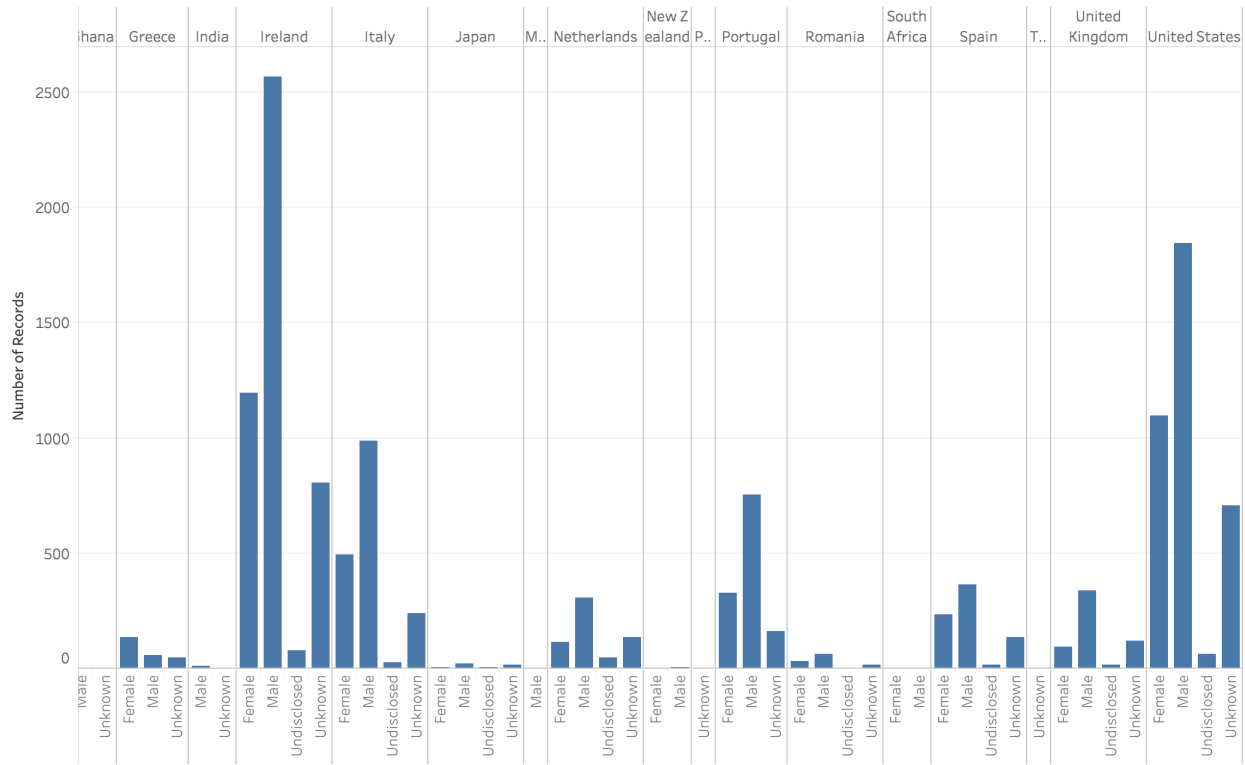


Figure 28 - Gender Breakdown by Country

Figure 28 shows the gender breakdown by country for each Dojo. This query would have been previously impossible as gender and dojo data is stored within different source databases.

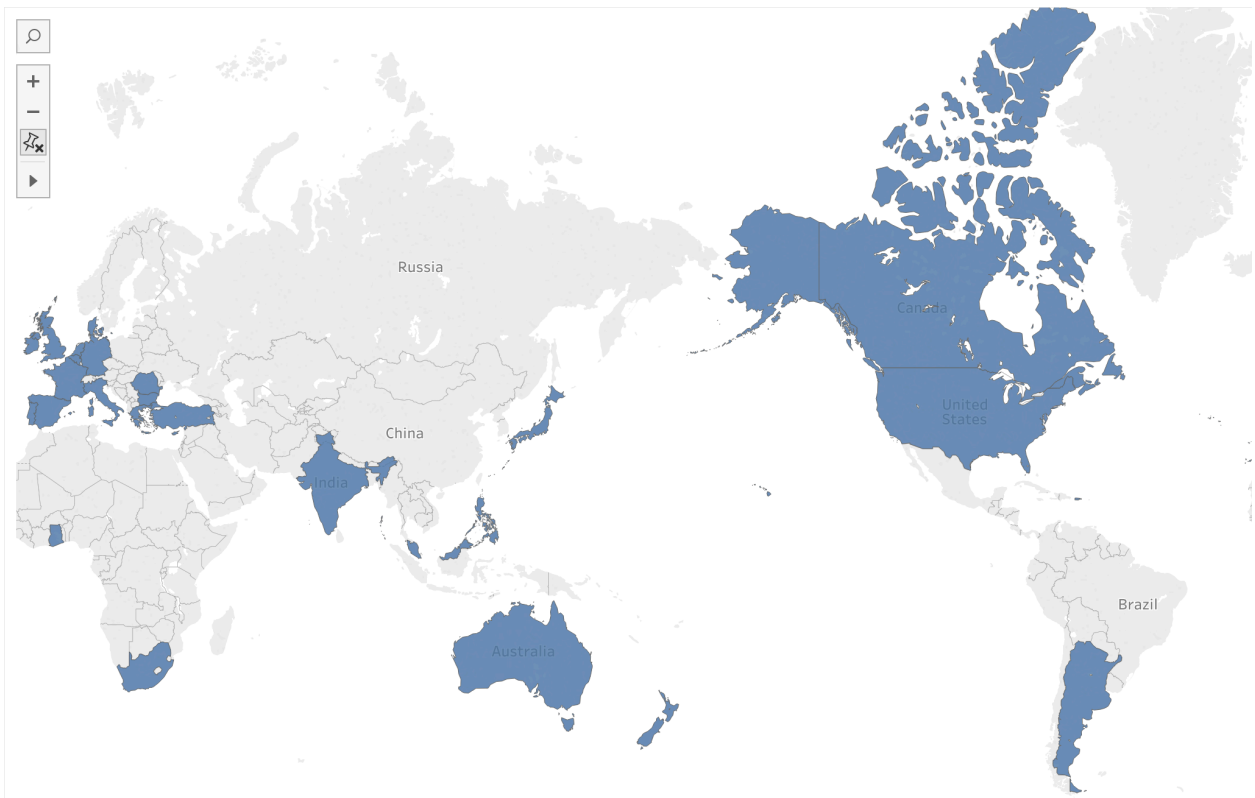


Figure 29 - Visualization of Countries with Active Dojos

Figure 29 shows visualizes which countries have active Dojos. This would have been previously impossible as their country data is saved as JSON within the source Dojos database.

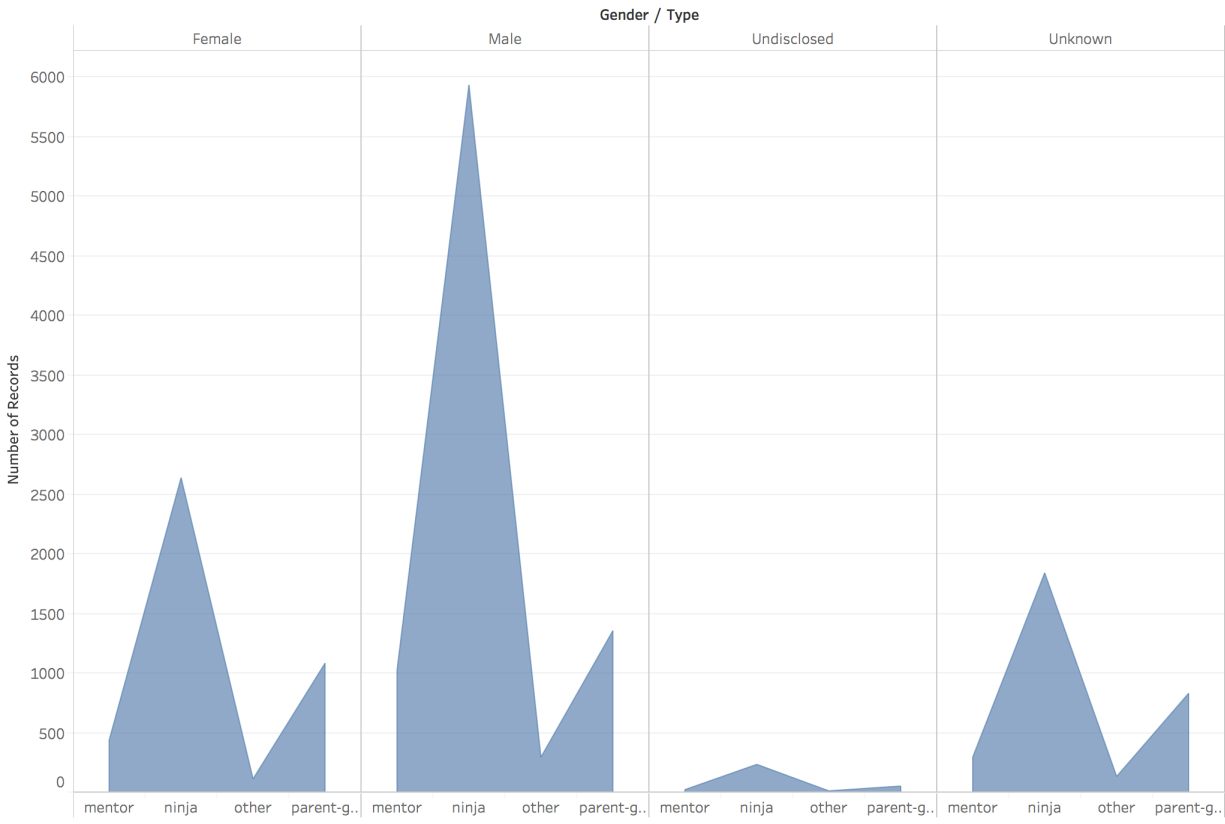


Figure 30 - Total Number of Ticket Types Based on Gender

Figure 30 shows the count of genders for each ticket type. This query would have been previously impossible as tickets were stored in the events database and gender was stored within the user's database.

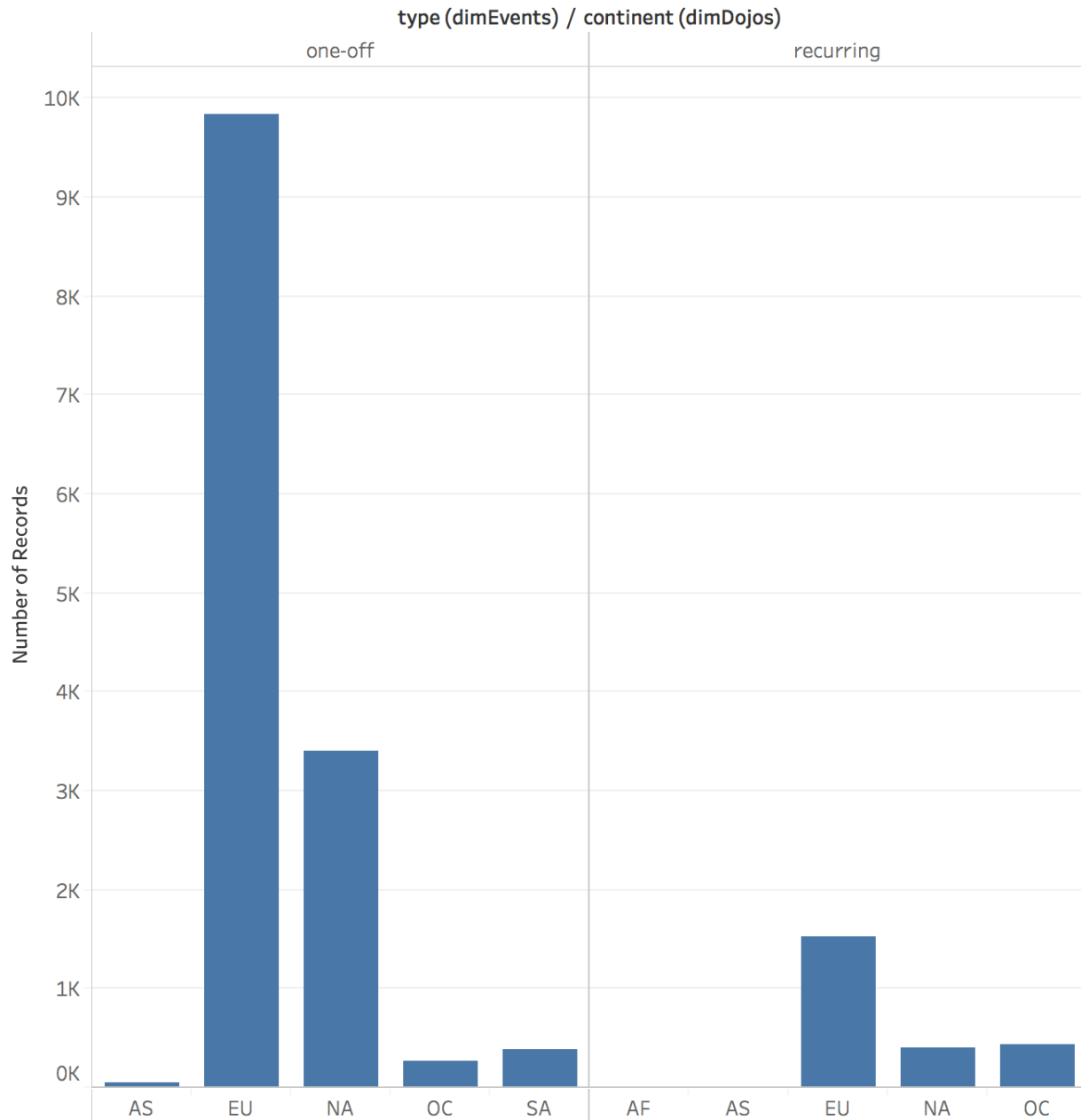


Figure 31 - Which Event Types Are Most Prevalent in Which Countries

Figure 31 the most common Dojo event type by continent. This would have been previously impossible as the event type was stored in the events database and the Dojo location was stored in the Dojos database.

Conclusions and future work

From undertaking this project, a lot was learned. It quickly became clear that data warehousing is a difficult skill to master. It goes against many concepts which are thought in standard database classes. Using Kimballs approach, the idea of not normalizing the data was a strange and took some time to understand. It also became abundantly clear that a data warehouse project isn't entirely all about itself. It is vitally important to understand how the data is stored and saved in the source databases, as well as what kind of relationships exist within the data.

Using a star schema proved an ideal solution to the problem. The schema allows the CoderDojo Foundation to filter their metrics across several dimensions, something which was previously not possible within the structure of the source data. By using a star schema, non-trivial queries can be executed such as those which span across multiple source databases.

Many difficulties were encountered while working on the project. The toughest problem was populating the fact table. Not having access to professional tools made this problem tenfold more difficult. SQL Server, for example would populate the fact table by itself based on the relationships between the dimension tables to the fact table. This was not feasible in this project and the populating the fact table became a manual process. It was solved using a staging table. The abundant lack of data in some key columns also proved a delicate issue. After analyzing these columns, it was seen that it was appropriate to replace the empty fields with 'Unknown'. This gives more insight into the measures and the CoderDojo Foundation can act on this.

Tableau is an extremely powerful tool which made visualizing the data extremely efficient. Without using Tableau, the project would include building a KPI dashboard manually through a separate technological stack to this project. This would drag the project timeline out a lot. It was fortunate that Tableau had a charitable license which the team could avail of. This is an item which can be considered for future work. Without relying on Tableau, the dashboard could be built into the CoderDojo Foundation platform. This would mean the platform provides a one stop shop for everything and limits the number of tools needed by the Foundation.

A final consideration for future work is to bring external APIs in to the data warehouse. Currently, the data warehouse focuses on the community platform and it's three databases as the source of data. However, the schema could be extended to include third party applications so long as there is an appropriate relationship to link the external services to the data held in Zen. Examples include email addresses or user identification strings.

Appendix

September monthly report

Achievements

This was an important month. It was the month in which I had to finalize my project idea. No advancements were made on the project, other than finalizing the idea. My project will be built for the organization in which I completed my internship, CoderDojo Foundation. As my project will be for an actual client, I had back and forth communication with the team, regarding the project. The initial idea was for a KPI (Key Performance Index) admin dashboard to be built. As this would lead to better reports and help secure funding. The pros and cons of taking on this project were outlined:

Pros:

1. Because it's more programming oriented.
2. Because more value on long term.
3. Because cross-database requests.
4. More dynamic filtering for internal analysis.
5. Possibility of analysis post-dashboard
6. Health check and input on strategy
7. Evolution of languages over time.

Cons:

1. Less data oriented

Reflection

I am happy with the month in case. I could finalize my project idea and have it accepted, following communication with the CoderDojo Foundation. I am looking forward to starting the project in the next few days. My steps to complete next are below:

1. Gather business requirements.
2. Map out functional and non-functional requirements.
3. Re-define the database schema.
4. Get the stack setup on my local environment.

Once these steps have been completed, I can begin to start piecing together the project. These steps will involve constant communication with the CoderDojo Foundation, for feedback and design ideas.

Supervisor meetings

At this stage, project supervisors have not been assigned.

October monthly report

Achievements

This month I could gather business requirements for my project. This involved several meetings with the CoderDojo Foundation team and my academic supervisor. CoderDojo layed out their needs through functional requirements, while my supervisor advised the best approach to solve the problem. One of the key achievements was agreeing that the project, for all intents and purposes – is a data warehousing project.

Additionally, I fully completed my requirements specification – subject to review. Once this was completed, I started work on the project itself. I gathered all data sources from CoderDojo which will be used for the data warehouse, and took actions from our meeting. Using the data sources, I could build an initial entity relationship model for the data warehouse, however it will heavily change as it is an early draft.

Reflection

I believe it was a productive month. I completed a lot of objectives. All paperwork is done, such that I can begin the development of the project. This is also aided by the fact that all requirements have been gathered and agreed upon between both parties. By finishing my requirements specification early, it gives me more time to focus on development, which is certainly needed as an expected 50-60 hours is to be put into modelling the data warehouse.

If there was one item I would have liked to complete which I didn't, it would be GUI mock-ups. This is something I will need to do later. However, visualization of the data is not a priority until further down the road. Due to this it is not essential that the mock-ups are completed *right away*.

Supervisor meetings

This month, I was assigned my supervisor – Dr. Simon Caton. I could meet with Caton several times this month to discuss the needs of the project and how to best approach the problem. I passed information between Caton and CoderDojo to best establish requirements and approach.

November monthly report

Achievements

In November, I was rather ill and such got less done than I would have liked. I had more discussions with CoderDojo. I discussed in depth about the strategy for the warehouse and specifically how the ETL process works and what it does. On top of this, I continued working on the data warehouse model and requested more data sources from CoderDojo. One of the additional sources of data is the forums on zen.coderdojo.com. This is another priority to have statistics about and so it will be added to the model early on.

This month the requirements also evolved. There are discussions to build an API alongside the data warehouse. This way Zen can consume the warehouse, but also it will allow the warehouse to be opened to more external applications also. This is not set in stone yet but more of a talking point.

Reflection

I would have liked to have gotten more done this month. There is one item which I really want to finalize as soon as possible and it is which database management system will be used for the data warehouse. SQL Server and MySQL are the two biggest choices, however CoderDojo is open-source and would prefer PostgreSQL. This is my biggest challenge at the minute and something which I want to finalize soon.

In December, I plan to complete and finalize the data warehouse modal ERD with two initial data sources, Zen and Zen Forums. Once the ERD is finished, I will then construct the warehouse in the (yet to be chosen) DBMS. This will be the construct of my prototype which I will present on the final week of the semester. I plan to meet with Simon again and talk more about the project and specifically the prototype. It's an area which is important as it will build the foundation of work down the line. I will also be meeting with CoderDojo again this month to discuss progress.

Supervisor meetings

There were meetings with Caton this month, there was also back and forth emails to discuss progress.

December monthly report

Achievements

During December, I finalized my database entity relationship model for two sets of data.

1. Zen.
2. Forums.

Once the schema was finalized, I wrote a small pipeline script to extract the data from Zen production databased and insert them into the data warehouse. Not all fields were carried across but only those which were relevant to the metrics required by CoderDojo. I also prepared my PowerPoint presentation for my prototype, detailing all the work which has been done to data, work which is left to do and the architecture of the system.

Finally, I presented my prototype to Simon and Vikas. The presentation went well and I got a result of 18/25. There were no real problems, other than the fact that CoderDojo has not yet handed over data. However, I went straight to CoderDojo afterwards about.

Reflection

It was a good month and I was happy with my presentation. Simon and Vikas had no real questions, which I believe means I covered everything in my presentation. Not having the data at this stage was an issue with Vikas and he suggested I get this sorted as soon as possible. It will be difficult to move forward with the project without the data.

Supervisor meetings

I met with Caton early in the month, prior to my presentation. The purpose of this meeting was to go over what key points my prototype presentation should hit well on to get good marks. We also went over the grading rubric.

January monthly report

Achievements

During January, I primarily spent time reviewing what I have got done up until this point and spent a bit more time researching data warehousing to ensure I was on the correct path. I realized that I had one or two errors in my design of the warehouse so I have worked to fix them. This is primarily in the schema of the warehouse. The relationships were not correct in some places and had to be changed. Furthermore, I have spent time investigating other APIs which will be pipelined into the warehouse. I have passed these APIs onto CoderDojo for review and for them to extract the metrics which they want.

Reflections

This was a slow month due to the exams. Although I am happy to have spotted some minor mistakes in my design and have been able to fix them. This has given me confidence going forward.

Supervisor meetings

There were no meetings between Simon and I this month.

February monthly report

Achievements

During February, I focused on my ETL strategy. Up until this point I was simply populating the data warehouse by transferring the data from the source production databases to the data warehouse. There was no real strategy. I worked up a strategy and actively worked on it. It begins with reading data from the source tables row by row and performing a transformation where needed. Such transformations include stripping out data from JSON. An example of this would be a column named country which shows where a user is from, being saved as so in the source. { "nameWithHierarchy" : "Ireland" }. However, only the value of "Ireland" is needed for the data warehouse. Other operations included replacing null values with a value "Unknown".

Finally, the schema changed once more as I began actively working on the ETL process. I enforced relationships between tables and assigned appropriate primary keys. I also wrapped the ETL script in a CronJob such that it would run every X days. This is not needed at the minute but will be needed once the application is deployed.

Reflections

February proved to be a great month for getting work done. I am happy that all the dimension's tables are successfully populated with the data from source tables. The ETL process has also proved a success and all values are now in the correct data type. I would have liked to start working on the fact table this month but ran out of time.

Supervisor meetings

Unfortunately, this month I had no meeting with Simon or CoderDojo. Over the next month, I would love to meet with both. I would like to get feedback from Simon about design and the next stage of the project. From CoderDojo, I would like to see how they think the project is going and to get feedback on metrics being used. The next big step is to populate the fact table.

March monthly report

Achievements

This month, I finalized the software for visualization. The software which will be used is Tableau, although costly – it has a non-profit license which can be used by CoderDojo and a student license which I can personally use. I could meet Simon twice to discuss the next steps. The biggest of which is populating the fact table. This is something which I was not able to wrap my head around straight away and took some time. Simon suggested I populate the fact table with the more “simple” queries and then put them into Tableau. Afterwards, I should get feedback from CoderDojo.

After doing this, I had a product which I could present to CoderDojo. The technical team were happy with the solution and had several questions which they wanted answered. Most of these questions were related to the pipeline between hosting and Tableau receiving the data. At the end of the month, I met Simon again to show the measures in the fact table. It turned out that I populated the fact table in the wrong manner. Some of the measures which I had there can be retrieved by simply performing a count on a dimension column within Tableau itself. This led to me removing the unnecessary measures and identifying the ones which span across multiple sources. These are the ones which will go into the fact table.

Reflections

March was a positive month, I could begin adding real measures to my project and start to visualize them. For the first time in the project life cycle, I had a project which I could show off to the client and get actual feedback. This was an important step to make sure I was on the correct track. As mentioned, the fact table was populated with the wrong measures. This is the final step in the project – correctly populating the fact table with the correct measures which span multiple sources. Once this is done, the dashboard will be updated to be more visually appealing to the team. Afterwards, unit tests will be added and the project will be handed over to the CoderDojo Foundation.

Supervisor meetings

I met with Dr. Caton twice this month to discuss populating the fact table and how to best achieve this. Dr. Caton also helped to conceptually wrap my head around how the queries will work with the relationship between the fact table and the dimension table.

Bibliography

- [1] Ali, Shaker. "A Proposed Model For Data Warehouse ETL Processes". Sciencedirect.com. N.p., 2017. Web. 2 July 2011.
- [2] "Data Warehouse Definition - What Is A Data Warehouse". 1keydata.com. N.p., 2017. Web. 7 May 2017.
- [3] Dhakal, Avesh. "Data Warehouse Architecture". Datasciencecentral.com. N.p., 2017. Web. 20 May 2014.
- [4] Perkins, Alan. "Critical Success Factors For Data Warehouse Engineering". <http://www.visible.com/>. N.p., 2017. Web. 7 May 2003.
- [5] Rangarajan, Sakthi, and View →. "Data Warehouse Design – Inmon Versus Kimball". TDAN.com. N.p., 2017. Web. 7 May 2017.
- [6] Ross, Margy. "The 10 Essential Rules Of Dimensional Modeling - Kimball Group". Kimball Group. N.p., 2017. Web. 7 May 2017
- [8] Schiff, Michael. "Data Warehousing: The Keys For A Successful Implementation". N.p., 2017. Web. 7 May 2011.
- [9] "Postgresql: Downloads". Postgresql.org. N.p., 2016. Web. 24 Oct. 2016.
- [10] Foundation, Node.js. "Node.js". Nodejs.org. N.p., 2016. Web. 24 Oct. 2016.
- [11] "Seneca, A Microservices Toolkit For Node.js". Senecajs.org. N.p., 2016. Web. 24 Oct. 2016.
- [12] "Coderdojo/Community-Platform". GitHub. N.p., 2016. Web. 24 Oct. 2016.
- [13] Caton, Simon. Data Warehousing and Business Intelligence. Slide 27-30. 21 Oct. 2016.
- [14] "Coderdojo/Community-Platform". GitHub. N.p., 2016. Web. 21 Oct. 2016. Development, Software. "HSC Agile Software Development -
- [15] Development, Software. "HSC Agile Software Development - Mahara". Web1.muirfield-h.schools.nsw.edu.au. N.p., 2016. Web. 21 Oct. 2016.