SQL analysis of NHS big data

Database Systems coursework

Joe Halloran

Executive summary

This report is, in essence, a soliloquy. It reveals thought processes as much as it describes actions. Superficially, it documents an investigation into NHS big data, via the medium of SQL. It records the process of building, populating, and querying a database. However, this report is more about the strategy and decision making that guided the process. It also includes some observations on the importance of coherent and consistent data.

Introduction

Structure is a beautiful thing. Unfortunately, all beautiful things have their price. In this case, the price of structure is foresight.

In this investigation of NHS data, we will learn that imposing structure on large unstructured datasets is difficult. Structure should have been defined and imposed before users (thousands of health professionals working across the NHS) start inputting data in an inconsistent way. As Pandora discovered, once opened, the box is hard to shut.

Section 1 of this report covers the initial set-up of the database. It details the logic behind a number of strategic decisions that shaped the subsequent stages of the investigation. Section 2 explains how the database was populated with NHS data sets. Section 3 describes the execution of SQL queries to answer a range of predetermined questions. It explains how ambiguities in these questions were addressed and how results were refined in light these ambiguities. Section 4 looks at ways in which the data could be cleaned and constrained, it explains how attempts to do so were frustrated by inconsistencies in the data and a lack of medical expertise. Section 4 expands upon the allusion to Pandora in the preceding paragraph; it extols the importance of instituting relational structures before users begin inputting data.

This report also includes appendices that detail the technical steps required to set up the database system. They are as much for my reference, as they are for public consumption. Read them at your peril.

Section 1: Database setup

The data we are handling is historical, and therefore very unlikely to be altered (unless of course some new data was discovered in a dusty filing cabinet in a rarely visited hospital wing). It was, therefore, possible to treat the data as effectively static.

With this in mind, I wanted to create an isolated and automated (where possible) means of setting up the database, so it could be easily rebuilt if problems were discovered in the configuration.

A full description of the process is listed in Appendix 1. In addition, you can find a summary of the process below.

For the reasons outlined above (isolated, replicable, and automated), I created a virtual server using Vagrant *(Vagrant by Hashicorp (2017))* running Ubuntu 14.04 *Ubuntu 14.04.5 LTS (2017)*. I then set up a LAMP server *(LAMP (software bundle) - Wikipedia (2017))* and installed PHPMyAdmin *(How to install PHPMyAdmin on Ubuntu - Liquid Web (2017))* to give me GUI access to the MySQL server from the host machine.

Analysis of the data, lead to a simple strategy for structuring the database tables:

1. Ignore arbitrary division of data into months
    - We were given two sets of spreadsheets from January and February 2016.
    - These spreadsheets were isomorphic and therefore should be included in the same table.
    - In some cases, spreadsheets contained duplicate and unnecessary data. Duplicate data should not be included in the database.
2. Table structure should reflect spreadsheet structure.
    - Table heading should become database fields.

With this strategy, I then wanted to create a script that would set up the database tables. I used the Python SQL Alchemy module *(SQL Alchemy - The Database Toolkit for Python (2017))*, which provides an ORM. This offers a means of visualising the database by creating tables as Python classes. The `database_setup.py` script (see Appendix 3) was created for this purpose.

The following database was created:

```
show tables;
+---------------------------+
| Tables_in_PrescriptionsDB |
+---------------------------+
| chemical                  |
| surgery                   |
```

```
| surgery_data              |
| treatment                 |
+---------------------------+

describe chemical;
+-------------------+-------------+------+-----+---------+----------------+
| Field             | Type        | Null | Key | Default | Extra          |
+-------------------+-------------+------+-----+---------+----------------+
| id                | int(11)     | NO   | PRI | NULL    | auto_increment |
| chemical_sub_code | varchar(20) | NO   | UNI | NULL    |                |
| name              | varchar(100)| NO   |     | NULL    |                |
+-------------------+-------------+------+-----+---------+----------------+

describe surgery;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| gp_id      | varchar(20) | YES  | UNI | NULL    |                |
| name       | varchar(100)| NO   |     | NULL    |                |
| addressOne | varchar(100)| NO   |     | NULL    |                |
| addressTwo | varchar(100)| NO   |     | NULL    |                |
| city       | varchar(100)| NO   |     | NULL    |                |
| county     | varchar(100)| NO   |     | NULL    |                |
| postcode   | varchar(20) | NO   |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+

describe surgery_data;
+-----------------------+-------------+------+-----+---------+----------------+
| Field                 | Type        | Null | Key | Default | Extra          |
+-----------------------+-------------+------+-----+---------+----------------+
| id                    | int(11)     | NO   | PRI | NULL    | auto_increment |
| practice              | varchar(20) | NO   |     | NULL    |                |
| postcode              | varchar(20) | NO   |     | NULL    |                |
| ons_ccg_code          | varchar(20) | NO   |     | NULL    |                |
| ccg_code              | varchar(20) | NO   |     | NULL    |                |
| ons_region_code       | varchar(100)| NO   |     | NULL    |                |
| nhse_region_code      | varchar(100)| NO   |     | NULL    |                |
| ons_comm_rgn_code     | varchar(100)| NO   |     | NULL    |                |
| nhse_comm_region_code | varchar(100)| NO   |     | NULL    |                |
| totalAll              | int(11)     | YES  |     | NULL    |                |
| totalMale             | int(11)     | YES  |     | NULL    |                |
| totalFemale           | int(11)     | YES  |     | NULL    |                |
+-----------------------+-------------+------+-----+---------+----------------+

describe treatment;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| id       | int(11)     | NO   | PRI | NULL    | auto_increment |
| sha      | varchar(10) | NO   |     | NULL    |                |
| pct      | varchar(20) | NO   |     | NULL    |                |
| practice | varchar(20) | NO   |     | NULL    |                |
| bnf_code | varchar(20) | NO   |     | NULL    |                |
| bnf_name | varchar(100)| NO   |     | NULL    |                |
```

```
| items    | int(11)      | NO   |     | NULL    |                |
| nic      | float        | NO   |     | NULL    |                |
| act_cost | float        | NO   |     | NULL    |                |
| quantity | int(11)      | NO   |     | NULL    |                |
| period   | varchar(20)  | NO   |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
```

Section 2: Data upload

Having set up the database, I decided to continue using SQL Alchemy to parse the requisite CSV files and populate the database.

This extra layer of abstraction allowed me to merge spreadsheets separated by month and, where appropriate, ignore duplicates. For instance, the two files that listed all GPs in the UK were more than 99% duplicated. This lead to the creation of the `populate_db.py` script (see Appendix 4).

The one great drawback of this approach was the time taken to run the `populate_db.py` script. It took over 8 hours!! to complete the process of populating the tables with data. This is for a number of reasons:

1. **The sheer scale of the data:**
    - Two of the CSV files where 1.4GB.
    - The final database contained c. 18 million rows.
2. **Limited hardware resources**
    - The virtual MySQL server only had access to 512MB of ram on the host machine.
3. **The extra cost of using an ORM tool:**
    - SQL Alchemy added an extra layer of abstraction.
4. **Some data integrity vs speed trade-offs in the `populate_db.py` code**:
    - For instance, the `session.commit()` command, which commits changes to the database, is run after every line in the CSV file is parsed. This could have been run only at the end of the file, or after *x* lines are parsed, to reduce the number of commits and therefore execution time. However, this would create a risk of data loss, if the program terminated before reaching the end of the file or *x* lines.

Despite this huge time delay, this process provided a reliable, automated, and replicable means of populating the databases.

Section 3: Database queries

a) How many practices and registered patients are there in the N17 postcode Area?

Final Answer

There are 7 surgeries in the N17 area with 52248 patients

Queries

This could be answered using the following queries.

```
SELECT COUNT(postcode)
FROM surgery
WHERE postcode LIKE "n17%";
+-----------------+
| COUNT(postcode) |
+-----------------+
|               7 |
+-----------------+
1 row in set (0.01 sec)
```

```
SELECT sum(totalAll)
FROM surgery_data
WHERE postcode LIKE "N17%";
+---------------+
| sum(totalAll) |
+---------------+
|         52248 |
+---------------+
1 row in set (0.01 sec)
```

b) Which practice prescribed the most beta blockers per registered patients in total over the two month period?

Final answer

Burrswood Nursing home.

Queries

This question requires a definition of "beta-blockers". In lieu of genuine medical expertise, a list was found on the NHS Choices website *(Beta-blockers - NHS Choices (2017))*.

The term "prescribed the most" also requires some consideration. If we are counting the number of times a doctor issues a prescription, we should look at the 'items' column. If we were looking at the sheer amount of drugs that was given to patients, we should look at 'quantity'. I felt focusing on the sum of 'items' would best reflect the question.

I also decided to return the top 10 practices, to see if the top item is an outlier.

```
SELECT treatment.practice AS "GP id",
       sum(treatment.items) AS "Total beta-blockers",
       surgery_data.totalAll AS "total patients",
       (sum(treatment.items)/surgery_data.totalAll) AS "Average"
FROM surgery_data
INNER JOIN treatment ON treatment.practice = surgery_data.practice
WHERE bnf_name LIKE "%atenolol%"
  OR bnf_name LIKE "%Tenormin%"
  OR bnf_name LIKE "%bisoprolol%"
  OR bnf_name LIKE "%Cardicor%"
  OR bnf_name LIKE "%Emcor%"
  OR bnf_name LIKE "%carvedilol%"
  OR bnf_name LIKE "%toprolol%"
  OR bnf_name LIKE "%Betaloc%"
  OR bnf_name LIKE "%Lopresor%"
  OR bnf_name LIKE "%nebivolol%"
  OR bnf_name LIKE "%Nebilet%"
  OR bnf_name LIKE "%Inderal%"
GROUP BY treatment.practice
ORDER BY Average DESC
LIMIT 10;
+--------+---------------------+----------------+---------+
| GP id  | Total beta-blockers | total patients | Average |
+--------+---------------------+----------------+---------+
| G82651 |                  12 |              1 | 12.0000 |
| Y04786 |                 609 |            316 |  1.9272 |
| B86674 |                  12 |             20 |  0.6000 |
| E87711 |                 167 |            284 |  0.5880 |
| Y02511 |                 339 |            710 |  0.4775 |
| B81683 |                 662 |           1727 |  0.3833 |
| A89040 |                1002 |           2902 |  0.3453 |
| Y02625 |                 378 |           1095 |  0.3452 |
| C88626 |                 538 |           1599 |  0.3365 |
| Y00081 |                  12 |             37 |  0.3243 |
+--------+---------------------+----------------+---------+
10 rows in set (2 min 8.14 sec)
```

As suspected the top item is an anomaly, with only one patient. Further investigate revealed G82651 is a private nursing home that presumably accepts a limited number of NHS patients *(Burrswood nursing home (2017))* . We can assume that only 1 NHS patient was resident at the time.

```
SELECT gp_id,
       name,
       postcode
FROM surgery
WHERE gp_id = "G82651";
+--------+------------------------+----------+
| gp_id  | name                   | postcode |
+--------+------------------------+----------+
| G82651 | BURRSWOOD NURSING HOME | TN3 9PY  |
+--------+------------------------+----------+
1 row in set (0.00 sec)
```

c) Which was the most prescribed medication across all practices?

Final answer

Simvastatin, and related drugs with chemical code 0212000Y0, were prescribed 5116027 times.

Queries

As with question b, the term "most prescribed" requires technical definition. For consistency, I stuck with summing values in the 'item' columns.

The term "prescribed medication" also requires unpicking. I first grouped by "bnf_code".

```
SELECT bnf_code,
       bnf_name,
       sum(items) AS total
FROM treatment
GROUP BY bnf_code
ORDER BY total DESC
LIMIT 1;
+-----------------+-------------------------+---------+
| bnf_code        | bnf_name                | total   |
+-----------------+-------------------------+---------+
| 0103050P0AAAAAA | Omeprazole_Cap E/C 20mg | 4269629 |
+-----------------+-------------------------+---------+
1 row in set (26.55 sec)
```

However, a BNF code describes a particular drug at a particular dosage and in a particular form. This seemed too narrow, as the same drug, when prescribed in different forms and at quantities, is counted separately.

For this reason, and after looking at the Chemical table, I decided the look at the first 9 characters of the BNF code. This seemed to reflect the drug genus, as opposed to exact name and dosage. For example:

```
SELECT *
FROM chemical
LIMIT 1;
+----+-------------------+-----------------+
| id | chemical_sub_code | name            |
+----+-------------------+-----------------+
|  1 | 0101010A0         | Alexitol Sodium |
+----+-------------------+-----------------+
1 row in set (0.00 sec)
```

Searching for prescriptions with this code gave me a result of:

```
SELECT left(bnf_code,9) AS sub_code,
       bnf_name,
       sum(items) AS total
FROM treatment
GROUP BY sub_code
ORDER BY total DESC
LIMIT 1;
+-----------+----------------------+---------+
| sub_code  | bnf_name             | total   |
+-----------+----------------------+---------+
| 0212000Y0 | Simvastatin_Tab 40mg | 5116027 |
+-----------+----------------------+---------+
1 row in set (0.00 sec)
```

Digging a little further, I found that this drug came in a variety of names and dosages:

```
SELECT left(bnf_code,9) AS sub_code,
       bnf_name,
       count(items)
FROM treatment
WHERE left(bnf_code,9) = "0212000Y0"
GROUP BY bnf_name;
+-----------+-------------------------------------+--------------+
| sub_code  | bnf_name                            | count(items) |
+-----------+-------------------------------------+--------------+
| 0212000Y0 | Simvador_Tab 10mg                   |          409 |
| 0212000Y0 | Simvador_Tab 20mg                   |          757 |
| 0212000Y0 | Simvador_Tab 40mg                   |         1020 |
| 0212000Y0 | Simvador_Tab 80mg                   |           70 |
| 0212000Y0 | Simvastatin_Liq Spec 40mg/5ml       |            1 |
```

```
| 0212000Y0 | Simvastatin_Oral Susp 20mg/5ml S/F |          631 |
| 0212000Y0 | Simvastatin_Oral Susp 40mg/5ml S/F |          946 |
| 0212000Y0 | Simvastatin_Tab 10mg               |        14733 |
| 0212000Y0 | Simvastatin_Tab 20mg               |        15650 |
| 0212000Y0 | Simvastatin_Tab 40mg               |        15697 |
| 0212000Y0 | Simvastatin_Tab 80mg               |        10203 |
| 0212000Y0 | Zocor_Tab 10mg                     |          301 |
| 0212000Y0 | Zocor_Tab 20mg                     |          534 |
| 0212000Y0 | Zocor_Tab 40mg                     |          452 |
| 0212000Y0 | Zocor_Tab 80mg                     |           38 |
+-----------+------------------------------------+--------------+
15 rows in set (12.65 sec)
```

d) Which practice spent the most and the least per patient?

Final answer

Lowest: Y01690 - £0.01 per patient

Highest: G82651- £7609.05 per patient

Queries

This seemed to be relatively more straightforward to interpret. Find the total spent, find the total patients and divide to find an average.

I began looking at who spent the least, again getting the top 10 to check for anomalies. This query took 1hr 38mins!! to return results.

```
SELECT treatment.practice,
       sum(treatment.act_cost) AS total,
       surgery_data.totalAll,
       (sum(treatment.act_cost)/surgery_data.totalAll) AS "Average"
FROM surgery_data
INNER JOIN treatment ON treatment.practice = surgery_data.practice
GROUP BY treatment.practice
ORDER BY Average
LIMIT 10;
+----------+--------------------+----------+----------------------+
| practice | total              | totalAll | Average              |
+----------+--------------------+----------+----------------------+
| Y01690   | 50.740000396966934 |     3777 | 0.013433942387335699 |
| N82647   |   33.8699996471405 |     1883 | 0.017987254193914233 |
| E87718   |  2.569999933242798 |       90 | 0.028555554813808864 |
| E87055   | 14.989999771118164 |      285 | 0.052596490424976015 |
| L85602   | 101.61999946832657 |     1499 |  0.06779186088614181 |
| M82612   |  277.7500001192093 |     2308 |  0.12034228774662448 |
```

```
| N81634   | 2728.3400220274925 |    13716 |  0.19891659536508402 |
| F84727   |  389.9899954199791 |     1847 |  0.21114780477529999 |
| Y02988   |  84.33999973535538 |      325 |  0.25950769149340114 |
| N85643   |  644.0400002002716 |     1779 |   0.3620236088815467 |
+----------+--------------------+----------+----------------------+
10 rows in set (1 hour 38 min 38.25 sec)
```

Remarkably, surgery Y01690 only spent just over £0.01 per patient. This seemed bizarre, so I looked at the number of items prescribed.

```
SELECT sum(items),
       sum(act_cost),
       period
FROM treatment
WHERE practice = "Y01690"
GROUP BY period;
+------------+------------------+--------+
| sum(items) | sum(act_cost)    | period |
+------------+------------------+--------+
|         21 | 46.29000046849251 | 201601 |
|          2 | 4.449999928474426 | 201602 |
+------------+------------------+--------+
2 rows in set (13.11 sec)
```

Strangely, Y01690 only prescribed 21 items in January 2016 and 2 items in February 2016. This suggests extenuating circumstances (maybe the surgery was closed for refurbishment) would give such low prescription numbers and therefore such low total spend.

I then looked for the highest value, again returning the top 10.

```
SELECT treatment.practice,
       sum(treatment.act_cost) AS total,
       surgery_data.totalAll,
       (sum(treatment.act_cost)/surgery_data.totalAll) AS "Average"
FROM surgery_data
INNER JOIN treatment ON treatment.practice = surgery_data.practice
GROUP BY treatment.practice
ORDER BY Average DESC
LIMIT 10;
+----------+--------------------+----------+--------------------+
| practice | total              | totalAll | Average            |
+----------+--------------------+----------+--------------------+
| G82651   |  7609.050047278404 |        1 |  7609.050047278404 |
| Y02045   | 19572.239992558956 |        4 |  4893.059998139739 |
| Y01924   | 147981.80988019705 |      112 | 1321.2661596446164 |
| Y02873   | 10717.569992467761 |       20 |   535.878499623388 |
| Y02507   | 475.95999723672867 |        2 | 237.97999861836433 |
| Y02508   |  3333.070020824671 |       15 | 222.20466805497804 |
| Y04786   |  68229.09005251527 |      316 | 215.91484193833946 |
| Y02625   | 209576.74016997218 |     1095 | 191.39428326024856 |
```

```
| Y02511   |  131495.9803173244 |      710 |  185.2056060807386 |
| E87711   |  34552.28004068136 |      284 |  121.6629578897231 |
+----------+--------------------+----------+--------------------+
10 rows in set (1 hour 40 min 47.78 sec)
```

As with the lowest spent, we have found an anomalous value. G82651 spent over £7,000 on 1 patient.

G82651 is the same Burrswood Nursing home from question *b*. We can assume that the sole patient has extensive healthcare needs. Therefore, creating a very high average spend.

e) What was the difference in selective serotonin reuptake inhibitor prescriptions between January and February?

Final answer:

2742049 - 2725157 = 16892

Queries

Here we must define "serotonin reuptake inhibitor". The NHS provides the following list *(SSRIs - NHS (2017))*:

Types of SSRIs currently prescribed in the UK:

- citalopram (Cipramil)
- dapoxetine (Priligy)
- escitalopram (Cipralex)
- fluoxetine (Prozac or Oxactin)
- fluvoxamine (Faverin)
- paroxetine (Seroxat)
- sertraline (Lustral)

Which lead to this query.

```
SELECT sum(items) AS "Serotonin prescriptions",
       period AS selective_serotonin
FROM treatment
WHERE bnf_name LIKE "%citalopram%"
  OR bnf_name LIKE "%Cipramil%"
  OR bnf_name LIKE "%dapoxetine%"
  OR bnf_name LIKE "%Priligy%"
  OR bnf_name LIKE "%escitalopram%"
```

```
  OR bnf_name LIKE "%Cipralex%"
  OR bnf_name LIKE "%fluoxetine%"
  OR bnf_name LIKE "%Prozac%"
  OR bnf_name LIKE "%Oxactin%"
  OR bnf_name LIKE "%fluvoxamine%"
  OR bnf_name LIKE "%Faverin%"
  OR bnf_name LIKE "%paroxetine%"
  OR bnf_name LIKE "%Seroxat%"
  OR bnf_name LIKE "%sertraline%"
  OR bnf_name LIKE "%Lustral%"
GROUP BY period;
+-------------------------+---------------------+
| Serotonin prescriptions | selective_serotonin |
+-------------------------+---------------------+
|                 2742049 | 201601              |
|                 2725157 | 201602              |
+-------------------------+---------------------+
2 rows in set (27.81 sec)
```

Subtracting on value from the other gives the final answer.

f) Visualise the top 10 practices by number of metformin prescriptions throughout the entire period.

As above, The term "prescriptions" needs interpretation. I have again looked at the 'items' value for sake of consistency between answers.

```
SELECT practice,
       sum(items) AS "metformin prescriptions"
FROM treatment
WHERE bnf_name LIKE "%metformin%"
GROUP BY practice
ORDER BY sum(items) DESC
LIMIT 10;
+----------+-------------------------+
| practice | metformin prescriptions |
+----------+-------------------------+
| M85063   |                    3192 |
| K83002   |                    2848 |
| C82024   |                    2810 |
| F84006   |                    2739 |
| C83019   |                    2652 |
| C83064   |                    2545 |
| D82044   |                    2264 |
| F84087   |                    2183 |
| J82155   |                    2108 |
| Y01008   |                    2083 |
+----------+-------------------------+
10 rows in set (12.93 sec)
```

Section 4: Observations of Database

Enforcing more structure on such a large data set would allow for relationships between tables and therefore generate a more efficient database.

The GP Practice ID (e.g. "M85063") is a key identifier that could be used a primary / foreign key to link the Surgery, Surgery_data, and Treatment tables together. This would also avoid duplicating data, such as the appearance of the postcode in the Surgery and the Surgery_data tables.

The Chemicals table and the Treatment table could also be linked using the chemical_sub_code as a primary / foreign key. This link would render the repeated (and somewhat clunky) use of the "SELECT **LEFT(bnf_code,9)** FROM etc.." in the queries above redundant. It would, therefore, make it much easier to identify the prescription drugs that belong to the same chemical genus, but have distinct names for branding, legal, or other reasons.

Attempts to build this relational structure were frustrated by inconsistencies in the data, showing how important it is to institute these policies before users input data, and enforcing them with validation. For instance, there were discrepancies in the practice ids in the surgery and surgery_data table (both had 7 GP practices in N17 but only 6 were identical).

```
+--------------------+--------------------+
| Surgery_data table | Surgery table      |
+--------------------+--------------------+
| F85017             | F85017             |
| F85019             | F85019             |
| F85028             | F85028             |
| F85030             | F85030             |
| F85615             | F85615             |
| F85628             | F85628             |
| F85699             | Y04848             | // Does not match
+--------------------+--------------------+
```

There is also a danger in non-subject specialists (i.e. me) making these sorts of decisions. For instance, the assumption that bnf_codes in the Treatments table could be linked to those in the Chemical table is not supported by medical expertise. This further highlights the utility of making these decisions in the design phase, where expert advice can be sought.

Section 5: Conclusion

This report (hopefully) provides a thorough account of the process of building and querying a database containing NHS data. There are two key themes that emerged throughout the process. The first is that of instituting structure before users begin inputting data. It would be easier to answer the questions above if the data had been given a relational structure from the very beginning. This point was expounding upon in Section 4; trying to impose this structure once the data input has begun is very difficult.

The second theme, which was alluded to, but never subject to explicit and direct analysis is that of vagueness, ambiguity, and interpretation. The SQL queries supplied in Section 3 were all coloured by this issue. In each case, there was a phrase that needed picking apart and second guessing. All answers, therefore, lack precision as they are dependent on these imperfect deductions. This problem is stark in computing; logic machines are characterised by exactitude. But this is not a computer problem, it is a human one; it is a direct consequence of language and meaning.

Therefore, I fear this problem is insoluble, as the philosopher Timothy Williamson says: "we can make language less vague, we cannot make it perfectly precise" *(Williamson, pp. 1)*. With this eternal imperfection in mind, I submit this report for your judicious consideration.

Bibliograph

- Vagrant by Hashicorp (2017). Available at: https://www.vagrantup.com/ (Accessed: 13th June 2017).
- Ubuntu 14.04.5 LTS (2017). Available at: http://releases.ubuntu.com/14.04/ (Accessed: 13th June 2017)
- LAMP (software bundle) - Wikipedia (2017). Available at: https://en.wikipedia.org/wiki/LAMP_ (Accessed: 13th June 2017)
- How to install phpmyadmin on ubuntu - Liquid Web (2017). Available at: https://www.liquidweb.com/kb/how-to-install-and-configure-phpmyadmin-on-ubuntu-14-04/ (Accessed: 13th June 2017)
- Welcome! - The Apache HTTP Server Project (2017). Available at: https://httpd.apache.org/ (Accessed: 13th June 2017)
- SQL Alchemy - The Database Toolkit for Python (2017). Available at: https://www.sqlalchemy.org/ (Accessed: 13th June 2017)
- Beta-blockers - NHS Choices (2017). Available at: http://www.nhs.uk/conditions/beta-blockers/pages/introduction.aspx (Accessed: 13th June 2017)
- Burrswood | Burrswood Hospital, Kent. Available at: http://www.burrswood.org.uk/ (Accessed: 22nd June 2017)
- SSRIs - NHS (2017). Available at: http://www.nhs.uk/conditions/SSRIs-(selective-serotonin-reuptake-inhibitors)/Pages/Introduction.aspx (Accessed: 15th June 2017)
- Williamson, Timothy. Vagueness. London [u.a.]: Routledge, 2005. Print.

Appendices

Appendix 1 - How to setup a LAMP server with Vagrant.

Set up vagrant

- Install vagrant
- `mkdir [folder_name]`
- `cd [folder_name]`
- `mkdir public` (this will become the webserver root)
- `vagrant init`
- `vagrant box add ubuntu/trusty64` (other boxes available on vagrant website)
- Open Vagrantfile in [folder_name]. Add this:

```
Vagrant.configure("2") do |config|
config.vm.box = "ubuntu/trusty64"
end
```

- `vagrant up`
- Add this to the Vagrantfile:

```
PORT FORWARDING
Vagrant.configure("2") do |config|
config.vm.box = "ubuntu/trusty64"
config.vm.network :forwarded_port, guest: 80, host: 4567 //Choose host num
end
```

- `vagrant reload --provision`

Configure server

- `vagrant ssh` (Note, when you vagrant ssh into your machine, you're in/home/vagrant. /home/vagrant is a different directory from the synced /vagrant directory.)
- `sudo apt-get update`
- `sudo apt-get upgrade`
- `sudo apt-get -y install php5 libapache2-mod-php5 php5-mcrypt`
- `sudo apt-get -y install mysql-server libapache2-mod-auth-mysql php5-mysql`
- `sudo mysql_secure_installation` (complete set up)
- `cd /etc/apache2/sites-enabled`

- `Sudo nano 000-default.conf` . Change document root:

```
DocumentRoot /vagrant/public
# /var/www/html - Comment this out
```

- `cd ..`
- `sudo nano apache2.conf` . Add this (scroll down to find relevant section near bottom of document)

```
<Directory /vagrant/public >
      Options Indexes FollowSymLinks
      AllowOverride All
      Require all granted
</Directory>
```

- `sudo a2enmod rewrite`
- `sudo service apache2 restart`

Set up PHP My admin

- `sudo apt-get install phpmyadmin php-mbstring php-gettext`
- `sudo nano apache2.conf` Add this to bottom line

```
Include /etc/phpmyadmin/apache.conf
```

- `sudo service apache2 restart`

Create database

- `mysql -u root -p`
- `mysql> create database PrescriptionsDB;`
- `exit` (you can exit MySQL server and use PHP My Admin to access DB)
- `logout` (leave virtual machine)
- Add index.html (see appendix 2) to `[folder_name]/public` on host machine
- In your browser, navigate to localhost: [PORT_NUMBER_YOU_SELECTED_EARLIER]
- Log in to PHP My Admin

Appendix 2 - index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
  <body>
    <h1>Joe's PHP My Admin Testing server</h1>
    <a href="/phpmyadmin">Go to php my admin</a>
  </body>
</html>
```

Appendix 3 - Build database with Python & SQL Alchemy

Add script

- Navigate to [folder_name] on host machine
- Add `database_setup.py` (see below)

Install Python modules & run script

- `vagrant ssh`
- `sudo apt-get install python-dev python-pip python-setuptools build-essential`
- `pip install pymysql`
- `cd /vagrant`
- `python database_setup.py`

database_setup.py

```
import os
import sys
from sqlalchemy import Column, ForeignKey, Integer, String, Float
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy import create_engine

Base = declarative_base()

class Chemical(Base):
    __tablename__ = 'chemical'
    id = Column(Integer, primary_key = True)
    chemical_sub_code = Column(String(20), nullable=False, unique = True)
    name = Column(String(100), nullable = False, unique = False)

class Surgery(Base):
    __tablename__ = 'surgery'
    id = Column(Integer, primary_key = True)
    gp_id = Column(String(20), unique = True)
    name = Column(String(100), nullable = False, unique = False)
    addressOne = Column(String(100), nullable = False, unique = False)
    addressTwo = Column(String(100), nullable = False, unique = False)
    city = Column(String(100), nullable = False, unique = False)
    county = Column(String(100), nullable = False, unique = False)
    postcode = Column(String(20), nullable = False, unique = False)

class SurgeryData(Base):
    __tablename__ = 'surgery_data'
```

```
    id = Column(Integer, primary_key = True)
    # Should be FK
    practice = Column(String(20), nullable = False)
    postcode = Column(String(20), nullable = False)
    ons_ccg_code = Column(String(20), nullable = False)
    ccg_code = Column(String(20), nullable = False)
    ons_region_code = Column(String(100), nullable = False)
    nhse_region_code = Column(String(100), nullable = False)
    ons_comm_rgn_code = Column(String(100), nullable = False)
    nhse_comm_region_code = Column(String(100), nullable = False)
    totalAll = Column(Integer)
    totalMale = Column(Integer)
    totalFemale = Column(Integer)

class Treatment(Base):
    __tablename__ = 'treatment'
    id = Column(Integer, primary_key = True)
    sha = Column(String(10), nullable = False, unique = False)
    pct = Column(String(20), nullable = False, unique = False)
    # Should eb FK
    practice = Column(String(20), nullable = False, unique = False)
    bnf_code = Column(String(20), nullable = False, unique = False)
    bnf_name = Column(String(100), nullable = False, unique = False)
    items = Column(Integer, nullable = False, unique = False)
    nic = Column(Float, nullable = False, unique = False)
    act_cost = Column(Float, nullable = False, unique = False)
    quantity = Column(Integer, nullable = False, unique = False)
    period = Column(String(20), nullable = False, unique = False)

engine = create_engine(
    'mysql+pymysql://root:root@127.0.0.1:3306/PrescriptionsDB')
Base.metadata.create_all(engine)
```

Appendix 4

- Ensure `populate_db.py` (see below) is in [folder_name] on host machine
- From within `/vagrant` folder on virtual machine
- `python populate_db.py`

populate_db.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from time import gmtime, strftime

from database_setup import Base, Chemical, Surgery, SurgeryData, Treatment


def readChemicals(file):
    duplicates = []
    f = open(file)
    lineCounter = 0
    for line in f:
        items = line.split(",")
        chem_sub, name = stripWhiteSpaces(items[0]), stripWhiteSpaces(items[1])
        chem = Chemical(chemical_sub_code = chem_sub, name=name)
        session.add(chem)
        session.commit()
    f.close()

class FileReader:

    def getSession(self):
        engine = create_engine(
      'mysql+pymysql://root:root@127.0.0.1:3306/PrescriptionsDB')
        Base.metadata.bind = engine
        DBSession = sessionmaker(bind=engine)
        session = DBSession()
        return session

    def readChemicals(self, file):
        duplicates = []
        errors  =[]
        session  = self.getSession()
        f = open(file)
        lineCounter = 0
        first_line = f.readline() ## SKIP HEADERS
        for line in f:
            items = line.split(",")
            chem_sub = self.stripWhiteSpaces(items[0])
      name = self.stripWhiteSpaces(items[1])
            # if chem_sub does not exist in db
            if session.query(
```

```
        Chemical.id).filter(Chemical.chemical_sub_code==chem_sub).count() == 0:
                try:
                    chem = Chemical(chemical_sub_code = chem_sub, name=name)
                    session.add(chem)
                    session.commit()
                except:
                    errors.append(chem_sub)
            else:
                duplicates.append(chem_sub)
            lineCounter += 1
        print("")
        print("UPLOAD REPORT:", file)
        print("Lines parsed: ",lineCounter)
        print("Errors: ", len(errors))
        print("Duplicates: ", len(duplicates))
        f.close()

    def readSurgeries(self, file):
        duplicates = []
        errors = []
        session  = self.getSession()
        f = open(file)
        lineCounter = 0
        for line in f:
            items = line.split(",")
            gp_id = self.stripWhiteSpaces(items[1])
            if session.query(Surgery.gp_id).filter(Surgery.gp_id==gp_id).count() ==
                try:
                    gp = Surgery(
                        gp_id = gp_id,
                        name= self.stripWhiteSpaces(items[2]),
                        addressOne= self.stripWhiteSpaces(items[3]),
                        addressTwo= self.stripWhiteSpaces(items[4]),
                        city= self.stripWhiteSpaces(items[5]),
                        county = self.stripWhiteSpaces(items[6]),
                        postcode = self.stripWhiteSpaces(items[7])
                    )
                    session.add(gp)
                    session.commit()
                except:
                    errors.append(gp_id)
            else:
                duplicates.append(gp_id)
            lineCounter += 1
        print("")
        print("UPLOAD REPORT:", file)
        print("Lines parsed: ",lineCounter)
        print("Errors: ", len(errors))
        print("Duplicates: ", len(duplicates))
        f.close()

    def readSurgeriesData(self, file):
        duplicates = []
        errors = []
        session  = self.getSession()
```

```
        f = open(file)
        lineCounter = 0
        first_line = f.readline() ## SKIP HEADERS
        for line in f:
            items = line.split(",")
            gp_id = self.stripWhiteSpaces(items[0])
            try:
                data = SurgeryData(
                    practice = gp_id,
                    postcode = self.stripWhiteSpaces(items[1]),
                    ons_ccg_code = self.stripWhiteSpaces(items[2]),
                    ccg_code = self.stripWhiteSpaces(items[3]),
                    ons_region_code = self.stripWhiteSpaces(items[4]),
                    nhse_region_code = self.stripWhiteSpaces(items[5]),
                    ons_comm_rgn_code = self.stripWhiteSpaces(items[6]),
                    nhse_comm_region_code = self.stripWhiteSpaces(items[7]),
                    totalAll = self.stripWhiteSpaces(items[8]),
                    totalMale = self.stripWhiteSpaces(items[9]),
                    totalFemale = self.stripWhiteSpaces(items[10]),
                )
                session.add(data)
                session.commit()
            except:
                errors.append(gp_id)
            lineCounter += 1
        print("")
        print("UPLOAD REPORT:", file)
        print("Lines parsed: ",lineCounter)
        print("Errors: ", len(errors))
        print("Duplicates: ", len(duplicates))
        f.close()

    def readTreatment(self, file, run):
        duplicates = []
        errors = []
        session  = self.getSession()
        f = open(file)
        lineCounter = 0
        first_line = f.readline() ## SKIP HEADERS
        for line in f:
            items = line.split(",")
            try:
                treatment = Treatment(
                    sha = self.stripWhiteSpaces(items[0]),
                    pct = self.stripWhiteSpaces(items[1]),
                    practice = self.stripWhiteSpaces(items[2]),
                    bnf_code = self.stripWhiteSpaces(items[3]),
                    bnf_name = self.stripWhiteSpaces(items[4]),
                    items = self.stripWhiteSpaces(items[5]),
                    nic = self.stripWhiteSpaces(items[6]),
                    act_cost = self.stripWhiteSpaces(items[7]),
                    quantity = self.stripWhiteSpaces(items[8]),
                    period = self.stripWhiteSpaces(items[9])
                )
                session.add(treatment)
```

```
                session.commit()
            except:
                errors.append(self.stripWhiteSpaces(items[3])) #record bnf code as a
            lineCounter += 1
        print("")
        print("UPLOAD REPORT:", file)
        print("Lines parsed: ",lineCounter)
        print("Errors: ", len(errors))
        print("Duplicates: ", len(duplicates))
        f.close()

    def stripWhiteSpaces(self, string):
        return str.lstrip(str.rstrip(string))

x = FileReader()
print("BEGIN EXECUTION AT: " + strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime()))
x.readChemicals("T201601CHEMSUBS.CSV")
x.readChemicals("T201602CHEMSUBS.CSV")
x.readSurgeries("T201601ADDRBNFT.CSV")
x.readSurgeries("T201602ADDRBNFT.CSV")
x.readSurgeriesData("gp-reg-patients-prac-quin-age.csv")
print("BEGIN LARGE FILE 1: " + strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime()))
x.readTreatment("T201601PDPIBNFT.CSV", 1) ## Int input is  fudge to make PK
print("BEGIN LARGE FILE 1: " + strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime()))
x.readTreatment("T201602PDPIBNFT.CSV", 2) ## Int input is  fudge to make PK
print("COMPLETED: " + strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime()))
```

Report from populate_db.py run on 11th June 2017

```
BEGIN EXECUTION AT: Sun, 11 Jun 2017 07:29:17 +0000

('UPLOAD REPORT:', 'T201601CHEMSUBS.CSV')
('Lines parsed: ', 3432)
('Errors: ', 0)
('Duplicates: ', 0)

('UPLOAD REPORT:', 'T201602CHEMSUBS.CSV')
('Lines parsed: ', 3433)
('Errors: ', 0)
('Duplicates: ', 3432)

('UPLOAD REPORT:', 'T201601ADDRBNFT.CSV')
('Lines parsed: ', 9905)
('Errors: ', 0)
('Duplicates: ', 0)

('UPLOAD REPORT:', 'T201602ADDRBNFT.CSV')
('Lines parsed: ', 9916)
('Errors: ', 0)
('Duplicates: ', 9753)

('UPLOAD REPORT:', 'gp-reg-patients-prac-quin-age.csv')
```

```
('Lines parsed: ', 7712)
('Errors: ', 0)
('Duplicates: ', 0)
BEGIN LARGE FILE 1: Sun, 11 Jun 2017 07:30:46 +0000

('UPLOAD REPORT:', 'T201601PDPIBNFT.CSV')
('Lines parsed: ', 10036406)
('Errors: ', 0)
('Duplicates: ', 0)
BEGIN LARGE FILE 1: Sun, 11 Jun 2017 12:00:56 +0000

('UPLOAD REPORT:', 'T201602PDPIBNFT.CSV')
('Lines parsed: ', 10037913)
('Errors: ', 0)
('Duplicates: ', 0)

COMPLETED: Sun, 11 Jun 2017 15:51:15 +0000
```