

UMEÅ UNIVERSITET
Institutionen för Datavetenskap

13th September 2004

Extracting Relational Data From Web Sites

NAME: Araz Mustaffa

E-mail: ens98ama@cs.umu.se

Abstract

Many web sites contain interesting information that could be put into relational databases. Unfortunately such sites are heterogeneous and are in HTML. Thus they need to be structured before they are inserted into a database. The way this can be done is by using a program to extract information from web sites that have some common layout. The technique used here is to use regular expressions to find attributes and their values within web-pages

Contents

1	Introduction	1
2	Approach	2
2.1	Perl	2
2.2	DBI	2
2.3	Creating Database Tables	3
2.3.1	Planetary Table	3
2.3.2	World Table	4
2.4	Web sources	5
2.5	Extracting from web sources	7
2.5.1	Open file	8
2.6	Pattern matching	9
2.6.1	Planets pattern matching	9
2.6.2	Countries pattern matching	16
3	Results	18
3.1	Planets Result	18
3.2	Countries results	19
3.3	Querying the extracted data	19
4	Discussion	21
4.1	Limitation of searching	21
4.2	Compares to other projects	23
4.3	Mining Data Records in Web Pages	23
5	Conclusion	24
6	Acknowledgments	25
7	Appendix	27
7.1	Perl planets program	27
7.2	Perl countries program	30

1 Introduction

It is not always possible to get exact results when searching the web with keywords. For example in the case of searching for the *highest population country or minimum mass planet*, many documents will be found. Such documents must then be skimmed to obtain the exact answer. But when using relational databases (*structured information*) for searching *highest population country or the minimum mass planet* the correct results is returned if the relational data base is populated with correct information. By *structured information* we mean relational databases. A question however is how to obtain relational database from web. There are three possibilities *insert data manually, use natural language understanding or use extracting techniques*. It is expensive to insert data manually because one needs to hire somebody to do it. *Natural Language understanding* means that there is AI that builds data bases through reading documents or through reading different web sites. Finally data could be extracted from web but these *data* are semi structured or non structured in the form of hyper text *HTML forms*. *Data* needs to convert it to structured ways to be able to make query of it. The way to do that is by using program to handle this program. This program could be done in many languages but in this case *Perl* is used.

2 Approach

The way which is used to build data bases is by extracting information from web. Such information is semi or non structured in the form of hyper text *HTML documents* and the language which is used to extract information is perl.

2.1 Perl

Perl was introduced in 1987 by Larry Wall. The reason for its creation was that Mr. Wall was unhappy with the functionality that Sed, C, Awk and the Bourne Shell offered him. He looked for a language that will combine all of their best features, while having as few disadvantages of its own. Since then, Perl has seen several versions, each adding additional functionality. Perl is stable language cross platform programming language and is open source, licensed under its Artistic License, or the GNU General Public License. Perl support operating system such as Unix systems, Macintosh - (OS 7-9 and X), Windows and so on. The feature about Perl that it takes best features from other language such c and basic among them and it's database integration interface (DBI) supports third-party databases including Oracle, Sybase, Postgres, MySQL and others. Perl works with HTML, XML, and other mark-up languages. It is the most popular web programming language due to its text manipulation capabilities and rapid development cycle and it's DBI package makes web-database integration easy [1]. As well as Perl support the regular expression which help to find the pattern in efficient way without writing too much code and that depends on how familiar one with regular expression

2.2 DBI

DBI stand for Database independent interface for Perl [2]. The purpose behind that DBI is used is to connect the Perl program to databases which in their turn makes insertion to and select attributes from the databases that is created.

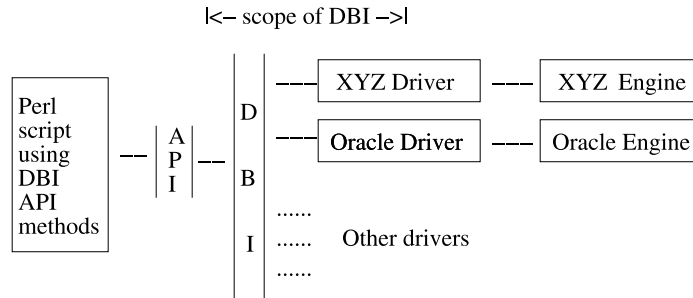


Figure 1: Perl DBI and the scope of DBI

The syntax of the connection could be like:

```
$dbh = DBI->connect($mydb, $dbuser, ' ') or die "Cannot connect to
database $DBI::errstr";
```

```
$mydb = 'dbi:Pg:dbname=world;host=postgres';
```

```
$dbuser = 'world'
```

\$dbh Database handle object

Pg means that Database independent interface is Postgres.

'world' is the user.

After the *DBI* is connected, it is possible to insert values to the table (planets). The syntax of the insert statement:

```
"insert into planets(attribute1, attribute2,..., attribute(n)) values (value1,
value2,..., value(n));"
```

2.3 Creating Database Tables

We assume that tables must be created for the data we intend to extract. For our examples two types of tables are created depending on the site the information is extracted.

2.3.1 Planetary Table

The tables which is created is called planets and all attributes are declared as real except for name which is declared varchar(string).

```

CREATE TABLE planets (
    Name          varchar,
    mass          real,
    vol           real,
    equa         real ,
    polar        real,
    radius       real,
    ellip        real,
    density      real,
    gravit       real,
    accel        real,
    primary key(Name));

```

Primary key is the name of the planet, because we must avoid inserting the same planet two times to the same database.

Name planet name (Earth, Jupiter, Mars, Mercury, Neptune, Pluto, Saturn, Uranus and Venus)

mass planet mass ($10^{24}kg$)

vol planet volume ($10^{10}km^3$)

equa planet equatorial radius (km)

polar planet polar radius (km)

radius Volumetric mean radius (km)

ellip planet Ellipticity (Flattening)

density planet Mean density (kg/m^3)

gravit planet Surface gravity (m/s^2)

accel planet Surface acceleration (m/s^2)

primary key(Name) to avoid inserting databases with the same name.

2.3.2 World Table

The table which is created called world which indicates the countries in the world.

```
CREATE TABLE world (  
    Name          varchar,  
    Population    real,  
    Hivrate       real,  
    primary key(Name));
```

The table has three attributes Name, Population and Hivrate.

Name is the country name.

Population is the population of the country.

Hivrate is the percentage of *HIV* in the country.

primary key(Name) to avoid inserting same country more than once to the database.

2.4 Web sources

Two sites have been used to extract information from depending on what kind of information are needed:

- Extracting information for planets: The site which is used to extract information from is "<http://nssdc.gsfc.nasa.gov/planetary/factsheet/planetfact.html>", where **planetfact** stand for earthfact, jupiterfact, marsfact, mercuryfact, neptunefact, plutofact, saturnfact, uranusfact and venusfact. The first ten properties from each planet are chosen to make database. Those properties are the same although in some cases it has a different name in one planetfact than another planetfact, for example in earthfact and jupiterfact are surface gravity and gravity the same property with different names.

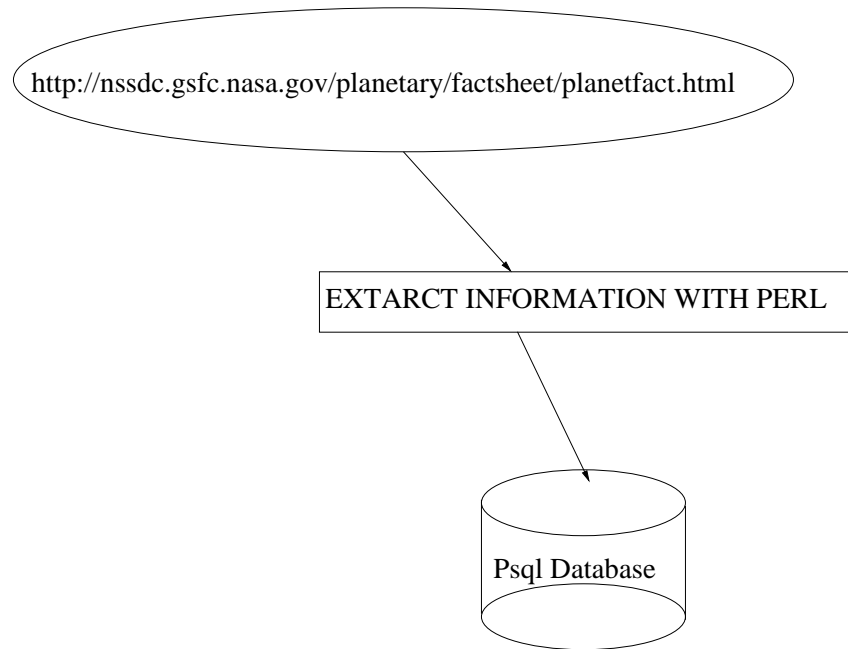


Figure 2: Extracting information from planetfact sites

- Extracting information for countries
the site which is used to extract information from could be found in this link "<http://www.cia.gov/cia/publications/factbook/geos/country.html>" where *country* stand for the first to letters from each country, some examples of that:
 - Afghanistan = af.
 - Albania = al.
 - sweden = sw.
 - United States = us.
 - and so on..

Only three attributes have been chosen from this site and the are *name*, *population* and *hiv rate*.

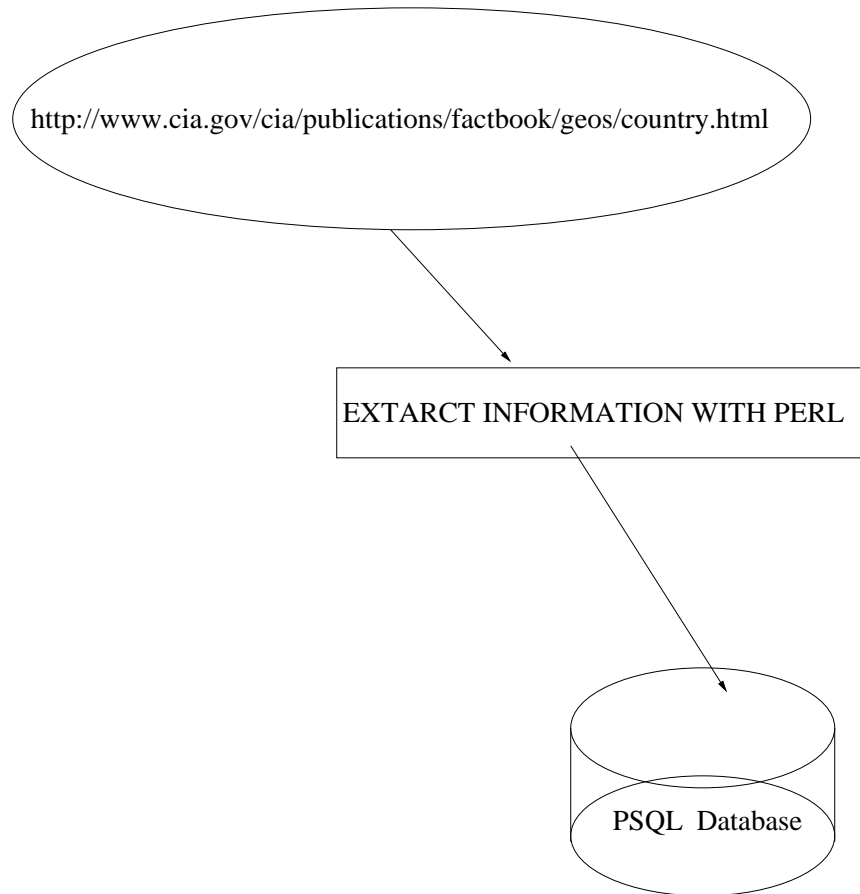


Figure 3: extracting information from countries sites

2.5 Extracting from web sources

To connect the program to the web service Perl provides **LWP** which stands for The World-Wide Web library for Perl. Because getting information from the web the program must be connected to the web. There are many types of **LWP** one of them is **LWP::Simple** which is stand for Simplified procedural interface for common functions. **LWP::Simple** provides functions such as **getstore** (*url*,*file*) gets a document identified by a URL and stores it in the. The return value is the HTTP response code

2.5.1 Open file

After storing URL to file, it is possible now to open the file in HTTP code. Some times the opening files does not seem to be clear as one open the source code(*HTTP code*) from a site to recover from that problem the following function `local $/ = "\r";` could be used which it means to separate the HTTP code in return.

```
<html>^M<head>^M<title>Earth Fact Sheet</title>^M</head>^M<body>^M<p$
Dr. David R. Williams, <a href="mailto:dwilliam@nssdc.gsfc.nasa.gov"$
NSSDC, Mail Code 633<br>
NASA Goddard Space Flight Center<br>
Greenbelt, MD 20771<br>
+1-301-286-1258<br>^M</address>^M<br clear=left>^M<hr>^M<h6>NASA Off$
^MLast Updated: 27 October 2003, DRW</h6>^M^M</body>^M</html>
```

By using the function `local $/ = "\r";` to get into more clear form, which separate the file in the end of line $\wedge M$ see bellow.

```
<html>
<head>
<title>Earth Fact Sheet</title>
</head>
<body>
<p>
<hr>
<H1>Earth Fact Sheet</H1>
SRC="/planetary/banner/earthfact.gif"></A>
<h3>Bulk parameters</h3>
<pre>
Mass (10<sup>24</sup> kg)                5.9736
Volume (10<sup>10</sup> km<sup>3</sup>)          108.321
Equatorial radius (km)              6378.1
Polar radius (km)                   6356.8
Volumetric mean radius (km)         6371.0
Core radius (km)                    3485
Ellipticity (Flattening)            0.00335
```

```

Mean density (kg/m3)          5515
Surface gravity (m/s2)        9.798
Surface acceleration (m/s2)    9.780
</body>
<html>

```

So that it is understandable what pattern to look for. Without the function `local $/ = "\r"`; it would be hard to look for the pattern that needed. Because of looking for something in the line and the whole file comes in the same line it would be very hard to make pattern searching in the form of technique that is used to solve the problem.

2.6 Pattern matching

2.6.1 Planets pattern matching

The patterns to be searched are the following:

1. Name

The name of the planets could be found by searching the html tag `title`. The name is between `<title> planet Fact Sheet </title>`. By removing `<title> Fact Sheet </title>`, we get the name of the planet in the form name equal to planet which is equal to one of the planets (Earth, Jupiter, Mars, Mercury, Neptune, Pluto, Saturn, Uranus and Venus). The Perl description to remove `<title> Fact Sheet </title>` is `(/(\>.*?)(.*?)\s/)` which means to skip `<title>` and the rest after the planet, then printing `$2 = (.*?)` which is equal to the planet's name

2. Mass ($10^{24}kg$)

There are two ways to search of the mass of the planets depending on what planet is taken:

- searching for the mass of the earth it looks like this in the html formats:

```

Mass (1024 kg)          5.9736

```

Only the last figure is of interest the rest to be skipped. The way to skip `Mass (1024 kg)` is the following:

```

(/(\>.*?)(\d.*?)\s/)

```

```

(\>.*?) = $1

```

```

(\d.*?)\s/ = $2

```

Which means to take away until the) and print \$2 which is equal to 5.9736

- Searching the masses of the rest planets
 Mass (10²⁴ kg) $number_1$ $number_2$ $number_3$
 $number_1$ is the planet mass.
 $number_2$ is the Earth mass.
 $number_3$ is the Ratio (Planet/Earth)

What is interesting is $number_1$ because it indicate the mass of the planets, $number_2$ and $number_3$ to be ignored. Perl description to get the result in the form mass = $number_1$ is the following:

```
(/(\).*?) (\d.*?)\s/)
```

```
\$1 = (/(\).*?)
```

```
\$2 = (\d.*?)\s/)
```

```
$planetmass = $2;
```

Which means to ignore "Mass (10²⁴ kg) = (/(\).*?) = \$1" and to insert planet mass to \$2 which is equal to the occurrence of first number($number_1$).

3. Volume (10¹⁰ km³)

There are two ways to search of the *Volume* of the planets depending on what *planet* is taken:

- Searching for the *Volume* of the *earth* it looks like this in the html formats:

```
Volume (1010 km3) 108.321
```

Only the last figure is of interest, the rest will be skipped. The way to skip (10¹⁰ km³) :

```
(/(\).*?) (\d.*?)\s/)
```

```
(\).*?) = \$1
```

```
(\d.*?)\s/ = \$2
```

Which means to take away until the) and print \$2 which is equal to 108.321

- Searching the *volumes* of the rest planets
 Volume (10¹⁰ km³) $number_1$
 $number_2$ $number_3$

$number_1$ is the planet volume.

$number_2$ is the earth volume.

$number_3$ is the Ratio (Planet/Earth)

What is interesting is $number_1$ because it indicates the mass of the planet, $number_2$ and $number_3$ to be ignored. Perl description to get the result in the form volume = $number_1$ is the following:

```
(/(\.*) (\d.*?)\s/) $planetvolume = $2;
```

Which means to ignore "Volume (10¹⁰ km³) = $number_1$ " and to insert planet volume to \$2 which is equal to the occurrence of first number($number_1$).

4. Equatorial radius (km)

By searching the pattern Equatorial in all these sites,

"<http://nssdc.gsfc.nasa.gov/planetary/factsheet/planetfact.html>",

where the **planetfact** are earthfact, marsfact, mercuryfact, venusfact and plutofact. their *HTML* is in the form:

Equatorial radius (km) $number_1$ $number_2$ $number_3$.

$number_1$ is the planet equatorial.

$number_2$ is the earth equatorial.

$number_3$ is the Ratio (Planet equatorial/Earth equatorial)

Except for the earth it's *HTML* looks like that:

Equatorial radius (km) $number_1$

Only one sort of pattern matching to be made for all the sites above and it is in the form of `(/(\.*) (\d.*?)\s/)` where $\$1 = (/(\.*)$ and $\$2 = (\d.*?)\s/)$. By $\$1$ means to find every thing until the right brackets that comes after *km*. $\$2$ is to find the first occurrence of number and skip the rest by using `\s`. When the case are other **planetfact** jupiterfact, neptunefact, uranusfact and saturnfact. Their *HTML* formats is:

Equatorial $number_1$ $number_2$ $number_3$.

The pattern matching is `(/(\.*) (\d.*?)\s/)`, where $\$1 = (.\d.*?)$ = Equatorial and $\$2 = (\d.*?)\s/$ to pick up $number_1$ and the rest to be skipped by using `\s` which means to ignore every thing after $number_1$.

5. Polar radius (km)

Like Equatorial radius there are also two cases to make pattern matching.

Case 1 By searching the pattern Polar in all these sites,

"<http://nssdc.gsfc.nasa.gov/planetary/factsheet/planetfact.html>",

where the **planetfact** are earthfact, marsfact, mecuryfact, venusfact and plutofact. their *HTML* is in the form

Polar radius (km) *number*₁ *number*₂ *number*₃.

*number*₁ is the planet polar radius.

*number*₂ is the earth polar radius.

*number*₃ is the Ratio (Planet Polar/Earth Polar)

Except for the earth it's *HTML* looks like this

Polar radius (km) *number*₁

Only one sort of pattern matching to be made for all the sites above and it is in the form of `(/(\.)*?)(\d.*?\s)/)` where

\$1 = `(/(\.)*?)` and \$2 = `(\d.*?\s)/)`. By \$1 means to find every thing until the right brackets that comes after *km*. \$2 is to find the first occurrence of number and skip the rest by using `\s`.

Case 2 When the case are other **planetfact** jupiterfact, neptunefact, uranusfact and saturnfact. Their *HTML* formats is:

Polar *number*₁ *number*₂ *number*₃.

The pattern matching is `(/(\.)*?)(\d.*?\s)/)`, where

\$1 = `(.)*?)` = Polar and \$2 = `(\d.*?\s)` to pick up *number*₁ and the rest to be skipped by using `\s` which means to ignore every thing after *number*₁.

6. Volumetric mean radius (km)

The pattern matching is the same for all planets(**planetfact**). Their *HTML* formats looks like:

Case earthfact

Volumetric mean radius (km) 6371.0

Case other planets

Volumetric mean radius (km) *number*₁ *number*₂ *number*₃

*number*₁ is the planet Volumetric mean radius radius.

*number*₂ is the earth Volumetric mean radius .

*number*₃ is the Ratio (Planet Volumetric mean radius/Earth Volumetric mean radius)

There is only one pattern matching to be made. `(/(\.)*?)(\d.*?)\s/)` where:

\$1 = (\).*?) to find until the right bracket that come after *km*.
 \$2 = (\d.*?\s) means to find *number₁* and the rest to be skipped by using \s which means to ignore every thing after *number₁*.

7. Ellipticity (Flattening)

there are two cases for making pattern matching when looking for attribute *Ellipticity*

Case 1 when *planetfact* are earthfact, marsfact, mecuryfact, venusfact and plutofact.

The *HTML* formats for Ellipticity (Flattening)is:

- When the *planetfact is earthfact*
 Ellipticity (Flattening) 0.00335
- When the *planetfact* marsfact, mecuryfact, venusfact and plutofact.
 Ellipticity (Flattening) *number₁* *number₂* *number₃*.

number₁ is the planet Ellipticity (Flattening).

number₂ is the earth Ellipticity (Flattening) .

number₃ is the Ratio (Planet Ellipticity (Flattening)/Earth Ellipticity (Flattening))

Perl description to find *Ellipticity* for the planets above is

(/(\.*?)(\d.*?\s)/) where:

\$1 = (\).*?) means to find until the right bracket that com after *Flattening*

\$2 = (\d.*?\s) mean to find *number₁* and skip the rest after *number₁*.

Case 2 when *planetfact* are jupiterfact, neptunefact, uranusfact and saturnfact. The *HTML* description is:

Ellipticity *number₁* *number₂* *number₃*.

To convert it to Perl, it would be: (/(\.*?)(\d.*?\s)/)

\$1 = (.*) = Ellipticity.

\$2 = (\d.*?\s) means to get *number₁* and skip every thing after *number₁* in the same line. \$2 is of interest to print it to get *Ellipticity* of the planets.

8. Mean density (*kg/m³*)

To find *Mean density* of each planet, the *HTML* for that pattern is:

Mean density (kg/m^3) **number**.
 where **number** indicate density of planet in case the planet is earthfact.
 Mean density (kg/m^3) *number₁* *number₂* *number₃*
 in the case of the planet is other than Earth where:

number₁ is the planet Mean density (kg/m^3).

number₂ is the Earth Mean density (kg/m^3).

number₃ is the Ratio (Planet Mean density (kg/m^3)/Earth Mean density (kg/m^3))

The pattern matching in Perl form is `(/(\.)*?(\d.*?\s)/)` where \$1 = `(/(\.)*?)` to the right bracket of (kg/m^3) and \$2 = `(\d.*?\s)/)` mean to skip every thing after the first number.

When printing \$2 the first number that come after right bracket (kg/m^3) it shows.

9. Surface gravity (m/s^2)

To find *surface gravity* or just *gravity*, two ways suggested:

Case 1

when the *planetfact* are earthfact, marsfact, mecuryfact, venusfact and plutofact. The *HTML* for these sites are:

Surface gravity (m/s^2) 9.798 . when the case is earthfact.

Surface gravity (m/s^2) *number₁* *number₂* *number₃*. when *planetfact* are marsfact, mecuryfact, venusfact and plutofact.

number₁ is the planet Surface gravity (m/s^2).

number₂ is the earth Surface gravity (m/s^2).

number₃ is the Ratio (Planet Surface gravity (m/s^2)/Earth Surface gravity (m/s^2))

The Perl description to find 9.798 when *planetfact* is earthfact, or to find *number₁* when planetfact are marsfact, mecuryfact, venusfact and plutofact is following:

`(/(\.)*?(\d.*?\s)/)` where: \$1 = `(\.)*?` means to find until the right bracket). \$2 = `(\d.*?\s)/)` to find *number₁* or 9.798 and skip the rest of the line.

Case 2

In this case *planetfact* are jupiterfact, neptunefact, uranusfact and

saturnfact. The *HTML* for these sites are following:

Gravity (eq., 1 bar) (m/s²) *number*₁ *number*₂
*number*₃

where *number*₁, *number*₂ and *number*₃ are the same as above.

Perl description in this case is `(/)(.*)\.(.*)\d.*?s/)` where:

`$1 = (/)(.*)` means to find the first right bracket.

`$2 = (\).*)` means to find the second right bracket.

`$3 = (\d.*?s/)` to find *number*₁ and skip all that come after *number*₁ in the line.

In **Case 1** `$2` indicate the gravity of planets and in **Case 2** `$3` indicate the gravity of planets.

10. Surface acceleration (*m/s*²)

To find *surface acceleration* or just *acceleration*, two ways suggested:

Case 1

when the *planetfact* are earthfact, marsfact, mecuryfact, venusfact and plutofact. The *HTML* for these sites are:

Surface acceleration (m/s²) 9.780 . when the case is earthfact.

Surface acceleration (m/s²) *number*₁ *number*₂
*number*₃

when *planetfact* are marsfact, mecuryfact, venusfact and plutofact.

*number*₁ is the planet Surface acceleration (*m/s*²).

*number*₂ is the earth Surface acceleration (*m/s*²) .

*number*₃ is the Ratio (Planet Surface acceleration (*m/s*²)/Earth Surface acceleration (*m/s*²))

The Perl description to find 9.780 when *planetfact* is earthfact, or to find *number*₁ when planetfact are marsfact, mecuryfact, venusfact and plutofact is following:

`(/)(.*)\d.*?s/)` where: `$1 = (\).*)` means to find until the right parenthesis).

`$1 = (\d.*?s/)` to find *number*₁ or 9.798 and skip the rest of the line.

Case 2

In this case *planetfact* are jupiterfact, neptunefact, uranusfact and saturnfact. The *HTML* for these sites are following:

Acceleration (eq., 1 bar) (m/s²) *number*₁ *number*₂

number₃

where *number₁*, *number₂* and *number₃* are the same as above. Perl description in this case is:

`(/\)(.*?)\)(.*?)(\d.*?\s)/)` where:

`$1 = (/\).*?)` means to find the first right bracket.

`$2 = (\).*?)` means to find the second right bracket.

`$3 = (\d.*?\s)/)` to find *number₁* and skip all that come after *number₁* in the line.

In **Case 1** `$2` indicate the acceleration of planets and in **Case 2** `$3` indicate the acceleration of planets.

2.6.2 Countries pattern matching

Before extracting data from these sites, it would be wiser to have look to the html for these sites to give us an idea how to make pattern matching. Because only three patterns are required to insert it to database and they are:

1. Country Name
2. Population.
3. Hiv rate.

The *html* for these three patterns by taking *Afghanistan* as ¹ an example are:

```
<html>
<head>
<title>CIA - The World Factbook - - Afghanistan</title>
<a href=" ../rankorder/2119rank.html"></a>
```

```
<br>
```

28,717,213 (July 2003 est.)

```
<a href=" ../rankorder/2155rank.html"><img
```

¹the rest of countries have the same html formats except those which have no population

```
src="../../../graphics/rankorder.jpg" alt="Rank Order"></a>
```

```
<br>
```

```
0.01\% (2001 est.)
```

```
</html>
```

The rest of *html* is not of interest in this case. After opening each link "http://www.cia.gov/cia/publications/factbook/geos/country.html" , now it is available to make pattern matching for the attributes that wanted to insert it to the database.

Before making any pattern matching let first the program read each line of the *html* because it makes easier to search for the wanted attributes.

- **The name of the country**

The name of the countries are between the two *html* tags <title>... </title>. by searching for *title* to find the name of countries. Perl description to find each country name is:

```
/(\--.*?)(\w.*?)(\<.*?)title(\>.*?)/
```

\$1 = (\--.*?) means to find (-) that comes before the name of the country.

\$2 = (\w.*?) to find the name of country.

\$3 = (\<.*?) to find < that comes after the name of country and before title.

\$4 = (\>.*?) to find > that comes after title

only \$2 is of interest to inserted to the database because it indicate the name of the country.

- **Finding population of the countries**

Note many countries are with out population so their population are indicated to zero (0). Because of the different text in the line that population of the countries appear [?], it would be hard to find population of countries by searching that line. The easiest way is by searching for *2119rank.html* because it comes in all the sites and in the same place(four steps before the line where population digits come). Note the fourth line that comes after *2119rank.html*. The Perl translation to find population is:

- **When counties have no population**

The searched pattern is `/(.*?)uninhabited\s/` or `/(.*?)no\s/`

or `/(.*?)uninhabited,\s/`. When those pattern are founded indicate their population to 0. The figure 0 are inserted to the database.

– **When countries have population**

There is digit to looks for that is equal to the population of the country

`/(\d+.*?\s)/`

`$1 = /(\d+.*?\s)/` which means to find the digit that comes in the beginning of the line and skip the rest of the line.

• **Finding hiv rate**

Here too the line that the percentage of the *hiv rate* of each country comes has different text , and when it comes to those countries that have no population there is of course nothing about their hiv rate so their hiv rate are 0. Because of different text in the line that hiv rate comes, it would be better to looks for a pattern that is equal for all the site to avoid many cases. The pattern that is looked for is *2155rank.html* and then to search for the fourth line that comes after that pattern.

– **When hiv rate is either NA or NA%**

The Perl description is:

`/(\w+.*?\s)/` and it is equal to `$1` which means to find the word that come in the line and skip the rest of the line. `$1` is indicated to 0 because of easiness when making selection and deletion in the database.

– **When hiv rate is digit**

Perl translation is:

`/(\d+.*?\s)/` which means to find the digit that comes in that line and skip the rest of line. By inserting that digit to the database to get hiv rate for those countries.

3 Results

3.1 Planets Result

After running the program in the form of: `perl programname.pl ARGV[i]`. where `ARGV[i]` are one of the following *planetfact* *earthfact*, *jupiterfact*,

marsfact, mercuryfact, neptunefact, plutofact, saturnfact, uranusfact and venusfact and by inserting the data from the running program to the table *planets* the following result is returned.

name	mass	vol	equa	polar	radius	ellip	density	gravit	accel
Earth	5.9736	108.321	6378.1	6356.8	6371	0.00335	5515	9.798	9.78
Jupiter	1898.6	143128	71492	66854	69911	0.06487	1.326e+06	24.79	23.12
Pluto	0.0125	0.715	1195	1195	1195	0	1750	0.58	0.58
Saturn	568.46	82713	60268	54364	58232	0.09796	687	10.44	8.96
Venus	4.8685	92.843	6051.8	6051.8	6051.8	0	5243	8.87	8.87
Uranus	86.832	6833	25559	24973	25362	0.02293	1.27e+06	8.87	8.69
Mercury	0.3302	6.083	2439.7	2439.7	2439.7	0	5427	3.7	3.7
Neptune	102.43	6254	24764	24341	24624	0.01708	1.638e+06	11.15	11
Mars	0.64185	16.318	3397	3375	3390	0.00648	3933	3.71	3.69

Table 1: The nine planets and their properties

3.2 Countries results

After running the program in the form of:
 perl program_name.pl ARGV[i].
 where ARGV[i] are one of the following countries see *country table*. Notes not the whole country name is taken as an ARGV[i] only two letters from each country are taken as an argument and those letters indicate the country.

3.3 Querying the extracted data

Now data is available in both *planets* and *world* tables. So it is possible to make queries over them in the form of selection.

- **The selection form**
SELECT <attribute list>
FROM <table list>
WHERE <condition>
- **Queries over planets table**
 - **Selecting the highest mass from planets**
query1:
 world=> select max(mass) from planets; max —— 1898.6 (1 row)

name	population	hivrate
Sweden	8.9864e+06	0.1
United States	2.93028e+08	0.6
China	1.29885e+09	0.1
Anguilla	13008	0
Baker Island	0	0
.	.	.
.	.	.
.	.	.
Cook Islands	21200	0
Coral Sea Islands	0	0
Costa Rica	3.95651e+06	0.6
Croatia	4.49687e+06	0.1

Table 2: Countries with their population and hiv rate

- **selecting name, mass, volume from planets who has a gravity more than 8.0.**

query2:

select name, mass, vol from planets where gravit > 8.0; and the returning results as follow: And many others select queries could

name	mass	vol
Earth	5.9736	108.321
Jupiter	1898.6	143128
Saturn	568.46	82713
Venus	4.8685	92.843
Uranus	86.832	6833
Neptune	102.43	6254

Table 3: Results from running the query2

be made.

- **Queries over world table**

Some queries example:

– **Selecting country with highest population**

query3:

```
select max(population) from world ;
```

```
max
```

```
-----
```

```
1.29885e+09
```

To know which country has such population another selection must be made and it is:

query4:

```
select name from world where population ilike '1.29885e+09';
```

```
name
```

```
-----
```

```
China
```

– **Selecting country with highest hiv rate**

query5:

```
select max(hivrate) from world;
```

```
max
```

```
-----
```

```
38.8
```

To know which country has such *hiv rate* another selection must be made and it is:

query6:

```
select name from world where hivrate = '38.8';
```

```
name
```

```
-----
```

```
Botswana
```

4 Discussion

There are several points that have to be discussed to make our view clear for the reader of this report and specially when it comes to those web sites that are unstructured.

4.1 Limitation of searching

Because when searching for some patterns in files or web sites it comes in their turn to looks for those lines that include these pattern and then make

pattern matching in the form of regular expression to find values of patterns or patterns themselves. Suppose there are many sites that information must be extracted from and all these sites have the same links except when it comes the word before *html* in other words when those sites are different in *word.html*, so they have different *word*. And suppose the word is looked for is *percentage* and it means to look for *middle age percentage* where *word.html* is *country.html*. The question is what would be done if the *html* sites for all those sites are different for the attribute *percentage*?

- **Same pattern in different places**

There is another percentage which indicate the *hiv percentage* of each country that comes either before or after the *middle age percentage* so when making pattern matching how would it be possible to know which percentage it means? In this case it would not be possible to make program that generate all sites. Many cases must be made for those sites. Sites with some conditions must be separated from other sites with other conditions.

- **Same pattern with different values**

When looking for pattern percentage, the perl description is to look for that word, but the syntax of the line that include percentage is different in different web sites so how would it be possible to make pattern matching? the syntax could be in one of the following cases:

1. Percentage followed by number
Percentage 2%.
2. Just a number
3.
3. Number followed by percentage symbol
4%
4. Percentage followed by number and the year when the statistics is taken
Percentage 2.5% (2003 st.)

It could be done by saving those sites and making some changes on those attributes so at those sites would be similar as possible but it is bad idea in applications. But in any way it is better than inserting data to databases

4.2 Compares to other projects

The project which is used to compare with it is Mondial database [3] which is a huge database that includes most of the information about countries, among these information are population, cities, rivers, mountains, ethnicities religions, and so on about each country . Instead of inserting all these information to the data bases , those information is brought from different sites by using *Florid's perl interface* as language to extract information from these sites. Those sites that information are extracted from are:

- Extracting political, economic, social, and geographical about these countries [4].
- Extracting information about administrative division (area and population) and main cities population [5].
- Extracting geographical coordinates of cities around the world [6]

By using pattern matching to look for information that needed with help of *Florid's perl interface* from all these sites and then making integration of attributes from all those sites that information are brought from. many attributes need to be reformatted because they have different names in different sites for example in one site is Denmark and in the other is Denm ark so both must have the same name when making queries from that database. The queries are in the form of *F-logic* which they are not in the form of selection or deletion. the query could be in the form:

: ?- C:country [*name@(cia)* - > *N*; *continent* - > *CT*].

and the result is:

```
cid(cia,"Albania"):country[name@(cia) - > "Albania"; continent - > "Europe"].  
cid(cia,"Andorra"):country[name@(cia) - > "Andorra"; continent - > "Europe"].
```

4.3 Mining Data Records in Web Pages

Another technique which is used to extract information from web is called *MDR* (Mining Data Records in Web Pages).It is a method that used for regular structured objects called data records that are in the form of table and related tags [7]. This technique is based in two observations:

1. A group of same data records that contain description of similar objects are presented in the same region of a page and they have same HTML tags. such

regions are called data regions. With the regard for the same HTML tags. a string matching could be used to define those similar tags.

- Using tag tree so that a group of similar data records are placed in the same region with the fact they are under one parent node.

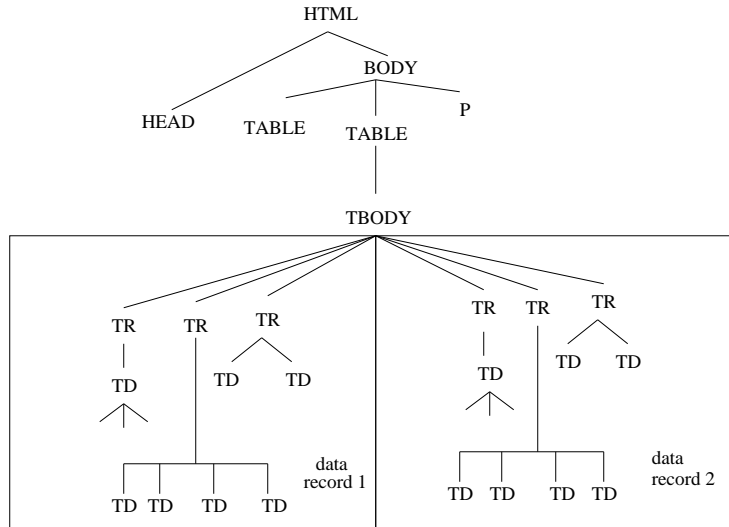


Figure 4: Tag tree of two similar records

Note that a web page may have few data regions. The method is applied with the regard to that data region should have more than two data records.

There are three steps for that method in applied web page:

- Building a HTML tree for the taken page.
- Data mining is used to identify different data regions by using HTML tree tag and string comparison.
- Last step is to identify data records for every data region so that each data region should contain more than two data records.

5 Conclusion

The aim of the project is to extract information from different web sites using perl, and to assess how effective perl is for such task. Many sites are updated all

frequently so why not making program and by just running the program to update those database that depends on these sites. DBI stands for data base independent interface for perl that has many driver among them is *Psql postgres structured query language* where it's driver is *Pg* the two letters from *Postgres*. The aim of DBI is to insert data to databases which is in turn make available the selection and deletion attributes from the databases that are created. Databases are widely used in every days life so it be wise to make program that handle it with out making insertion manually. Although some times it is not easy to make generate program for many similar sites because the bad structures of these site, the idea is still good by extracting information from web sites by using programming language.

6 Acknowledgments

I would like to thanks Michael Minock for choosing subject for me and supporting me during doing my theses. He made me to learn a new language and his idea about the extracting information from web sites is a good one.

References

- [1] Larry Wall. About perl. <http://www.perl.org/about.html>.
- [2] Tim Bunce. Dbi. <http://search.cpan.org/~timb/DBI/DBI.pm>.
- [3] Wolfgang May. Information extraction and integration with FLORID: The MONDIAL case study. *Universität Freiburg, Institut für Informatik*, (131), 1999. <http://dbis.informatik.uni-goettingen.de/Mondial>.
- [4] CIA. World factbook. 1996. http://theodora.com/wfb/wfb_1996.html.
- [5] John van der Heiden. Geohive global statistics. <http://www.stats.demon.nl>.
- [6] Qiblih geographical coordinates. <http://www.bcca.org/misc/qiblih/latlong.html>.
- [7] Robert Grossman Bing Liu and Yanhong Zhai. Conference on knowledge discovery in data archive proceeding of ninth acm sigkdd international conference on knowledge discovery and data mining table of contents washington, D.C. pages : 601–606, 2003. ISBN:1-58113-737-0.

7 Appendix

7.1 Perl planets program

```
#!/user/local/bin/perl
use LWP::Simple;
use DBI;
$db=dbi:Pg:dbname=world;host=postgres';
$dbuser='world';

for ($i = 0; $i <@ARGV ; $i++) {

    $url =
'http://nssdc.gsfc.nasa.gov/planetary/factsheet/' . $ARGV[$i] . '.html';
    getstore($url, $ARGV[$i] . ".html" );

    open(IN, "< " . $ARGV[$i] . ".html") || die "Can't open $file: $!";
    local $/ = "\r";
    while (<IN>) {

switch:
    if (/\\btitle\\b/) {

        if (/(>.*?)(.*?)\\s/) {
            $pla = $2;

            print 'Name = ' . $pla . "\\n" ;
        }
    }

    if (/\\bMass\\b/) {
        if (/(&).*?)(\\d.*?)\\s/) {
            $planetmass = $2;
            print 'masses = ' . $planetmass . "\\n";
        }
    }

    if (/\\bVolume\\b/) {
        if (/(&).*?)(\\d.*?)\\s/) {
            $planetvolume = $2 ;
        }
    }
}
```

```

    print 'volume = ' . $planetvolume . "\n";
  }
}

if (/\\bEquatorial\\b/) {
  if (/(\.)*(\d.*\s)/) {
    $planetequatorial = $2;
  } elsif (/(\.)*(\d.*\s)/) {
    $planetequatorial = $2;
  }
  print 'equatorial = ' . $planetequatorial . "\n";
}

if (/\\bPolar\\b/) {
  if (/(\.)*(\d.*\s)/) {
    $planetpol = $2;
  } elsif (/(\.)*(\d.*\s)/) {
    $planetpol = $2;
  }
  print 'polar = ' . $planetpol . "\n";
}

if (/\\bVolumetric mean radius\\b/) {
  if (/(\.)*(\d.*\s)/) {
    $planetvolum = $2;
    print 'planetvol = ' . $planetvolum . "\n";
  }
}

if (/\\bEllipticity\\b/) {
  if (/(\.)*(\d.*\s)/) {
    $planetellp = $2;
  } elsif (/(\.)*(\d.*\s)/) {
    $planetellp = $2;
  }
  print 'planetelip = ' . $planetellp . "\n";
}

```

```

    }
    if (/\\bMean density\\b/) {
        if (/(\).*?)(\d.*?\s)/) {
$plamen = $2;
if($plamen =~ /\d,\d/){

                $planetmean = join("", split(/,/,$plamen))*1000;
            }
else
{$planetmean = $2;
}
}

    print 'planetme = ' .$planetmean."\n";
}

if (/\\bSurface gravity\\b/ || /\\bGravity\\b/) {
    if (/(\).*?)(\d.*?\s)/) {

        $planetgrav = $3;
    } elseif (/(\).*?)(\d.*?\s)/) {
        $planetgrav = $2;
    }

    print 'planetgr = ' .$planetgrav . "\n";
}

if (/\\bSurface acceleration\\b/ || /\\bAcceleration\\b/) {
    if (/(\).*?)(\d.*?\s)/) {
        $planetacce = $3;
    } elseif (/(\).*?)(\d.*?\s)/) {
        $planetacce = $2;
    }
    print 'planetac = ' .$planetacce . "\n";

    close(IN);
}

```

```

    }
}

$dbh = DBI->connect($mydb,$dbuser,' ')
    or die "Cannot connect to database $DBI::errstr";

$command = "insert into planets(Name, mass,
vol, equa , polar, radius
,Ellip, density, gravit, accel
) values ('$pla', to_number('$planetmass', 9999999.9999999)
,to_number('$planetvolume', 9999999.9999999),
to_number('$planetequatorial', 9999999.9999999),
to_number('$planetpol', 9999999.9999999)
, to_number('$planetvolum',9999999.9999999) , to_number('$planetellp',
9999999.9999999) ,
to_number('$planetmean', 9999999.9999999) , to_number('$planetgrav',
99.999)
, to_number('$planetacce', 99.999));";
$sth = $dbh->prepare($command);
$sth->execute();

$dbh->disconnect();
print "\n";
}

```

7.2 Perl countries program

```

#!/user/local/bin/perl
use LWP::Simple;
use DBI;
$mydb='dbi:Pg:dbname=world;host=postgres';
$dbuser='world';

for ($i = 0; $i <@ARGV ; $i++) {

```

```

$url =

'http://www.cia.gov/cia/publications/factbook/geos/'.$ARGV[$i].'.html';
getstore($url, $ARGV[$i].".html" );

open(IN, "< ".$ARGV[$i].".html") || die "Can't open $file: $!";
while (<IN>) {
    @array = <IN>;

    for ($j = 0; $j < @array ; $j++) {
        if ($array[$j] =~ /\btitle\b/) {
            $cont = $array[$j];
            if ($cont =~ /(\---.*?)(\w.*?)(\<.*?)title(\>.*?)/) {
                $country1 = $2;
                $country = join("",(split('/',,$country1)));
                print 'Name = ' . $country. "\n" ;
            }
        }
    }

    if ($array[$j] =~ /2119rank.html/) {
        $output = $array[$j+4];
        $output1 = join("", split(//, $output));
        if ($output1 =~ /(.*)uninhabited\s/ || $output1 =~ /(.*)no\s/ ||
            $output1 =~ /(.*)uninhabited,\s/) {
            $pop = 0;
        } elsif ($output1 =~ /(\d+.*?\s)/) {
            $pop = $1;
        } elsif ( $output1 =~ /(\d+.*?\s)/) {
            $pop = $1;
        } else {
            $pop = 0;
        }
        print "$pop\n";
    }

    if ($array[$j] =~ /2155rank.html/) {
        $output3 = $array[$j+4];
        $output2 = join("", split(//, $output3));
        #print "$output3\n";
        if ($output2 =~ /(\d+.*?\s)/) {

```

```

        $hiv = $1;
        close(IN);
    } elsif ( $output2 =~ /(\w+.*?\s)/) {
        $hiv = 0;
        close(IN);
    }
}
}
if ($pop == 0) {
    $hiv = 0;
    #close(IN);
}
print "$hiv\n";
close(IN);

}

$dbh = DBI->connect($mydb,$dbuser,' ') or die
    "Cannot connect to database $DBI::errstr";
$command = "insert into world(Name, Population, Hivrate) values
('$country', to_number('$pop', 999999999999999), to_number('$hiv',99.99));";
$sth = $dbh->prepare($command);
$sth->execute();

$sth->finish();

$dbh->disconnect();
}

```