**Installing your own Web Server**

There are multiple web servers that you may use. A comparison of different web servers is available at Wikipedia at <http://en.wikipedia.org/wiki/Comparison_of_web_server_software>

One web server is Apache HTTP Server. You can obtain basic instructions, including instructions for installation at <http://httpd.apache.org/docs/2.2/>

Main things to note: How to start/stop/restart the web server, if you install the web server as a service using the httpd executable found in the /bin directory.

You can keep your html files in \htdocs directory, or you can specify alternate configurations in the file \conf\httpd.conf

Check that you are set by trying <http://localhost>. Also create a html file in \htdocs directory, suppose I call it test1.html, open the web page <http://localhost/test1.html>

Because we want to run Perl, we need to install Perl (for Windows). A Perl executable can be obtained from <http://www.activestate.com/activeperl/downloads>. When you install the software, the default location for installation is C:\Perl – see the \bin\Perl.exe executable. You can check that it is installed by running the Perl script at \cgi-bin\printenv.pl in Apache HTTP server installation.

Also, you can check <http://localhost/cgi-bin/printenv.pl> -- see that the page runs.

Now you need to ensure that you have the Oracle Instant Client <http://www.oracle.com/technetwork/database/features/instant-client/index-097480.html>

Now, for connecting to Oracle database, we need two things:

DBI module (that provides the interface for connecting to databases)

DBD::Oracle module (that is the Oracle driver implementing the DBI module).

You will see that these are already present in your Perl installation in the \lib directory.

Let us now write our first program, which we will call as will call as helloWorld.pl

#!c:\perl\bin\perl.exe

use CGI::Carp qw(fatalsToBrowser warningsToBrowser);

print "Content-type: text/html\n\n";

print ("<html><head><title>Hello World Script</title></head>\n");

print ("<body>\n");

print ("<h1>Hello World !!!!</h1>\n");

print ("</body></html>\n");

The first line is a directive as to where the perl executable is located. There are different modules within Perl, and we will use CGI::Carp module, which allows us to print statements to the browser window (i.e., the HTTP client). It also helps in debugging, which is difficult when we are using HTTP. We need the first print statement with the two new line characters (or more than two new line characters) according to the HTTP protocol. The rest of these are print statements, which get printed to the HTTP client (browser).

We will save the helloWorld.pl in your Apache Server’s \cgi-bin directory. Now check the web page: <http://localhost/cgi-bin/helloWorld.pl>

The helloWorld.pl could have been also written as helloWorldHTML.pl –

#!c:\perl\bin\perl.exe

use CGI::Carp qw(fatalsToBrowser warningsToBrowser);

print "Content-type: text/html\n\n";

print <<ENDHTML;

<html><head><title>Hello World Script</title></head>

<body>

<h1>Hello World !!!!</h1>

</body></html>

ENDHTML

Here everything within the block of PRINT <<ENDHTML – ENDHTML are printed to the HTTP client. **BE CAREFUL** – There **SHOULT NOT** be any white space characters before ENDHTML on the last line.

Let us look at one more scipt – that prints out the environment variables (this is the sample code called printenv.pl already present in your cgi-bin directory). Try running this from both your shell and from your browser, and notice the difference in the environment variables.

#!c:/perl/bin/perl.exe

##

## printenv -- demo CGI program which just prints its environment

##

print "Content-type: text/plain; charset=iso-8859-1\n\n";

foreach $var (sort(keys(%ENV))) {

$val = $ENV{$var};

$val =~ s|\n|\\n|g;

$val =~ s|"|\\"|g;

print "${var}=\"${val}\"\n";

}

Let us look at some things in this script – %ENV has the set of environment variables, as a hash. A hash is a data structure that consists of a set of (key, value) pairs. Look at the way we interate over the hash. keys%ENV returns the array of keys of the hash, and sort sorts the array. We then interate over this array with $var taking the value of the key every time. $ENV{$var} gives the value in the hash corresponding to that key. (Ignore the 2nd and 3rd lines. They are replacing some characters so that they appear correctly when displayed).

**Perl + Oracle**

Now, we are ready to connect to Oracle server using Perl. We will keep our passwords in a separate file, or Perl module. I will call this file as configOracle.pm and it looks like –

#!c:/perl/bin/perl.exe

package configOracle;

use Exporter;

@ISA = ('Exporter');

@EXPORT = qw($host $sid $port $userName $passwd);

$host = "db.csep.umflint.edu";

$sid = "csep";

$port = "1521";

$userName = "mmani";

$passwd = "mmani";

Note – the package line defines this module. It should be the same as the name of the file. We are defining a few variables like $host, $sid etc. In order to make these variables visible to other modules/programs, we export them.

Now we will write a Perl script – testOracle1.pl – that reads in the password etc from the above file, will drop a table, create another one, insert two rows, and then retrieve them and display them.

#!c:/perl/bin/perl.exe

use CGI::Carp qw(fatalsToBrowser warningsToBrowser);

use DBI;

use configOracle;

print "Content-type: text/html\n\n";

print ("<html><head><title>Test Oracle - 1</title></head>\n");

print ("<body>\n");

$dbh = DBI->connect("DBI:Oracle:host=$host;sid=$sid;port=$port", $userName, $passwd) || die "Database connection not made: $DBI::errstr";

$dropTable = $dbh->do ("DROP TABLE studentTemp");

if (!defined ($dropTable)) {

print ("error in dropping table studentTemp $DBI::errstr<br>\n"); }

$crTable = $dbh->do ("CREATE TABLE studentTemp (num int, name varchar (10))");

if (!defined ($crTable)) {

print ("error in creating table studentTemp $DBI::errstr<br>\n"); }

$rows = $dbh->do ("INSERT INTO studentTemp VALUES (1," . $dbh->quote ("Matt") . ")");

$rows = $dbh->do ("INSERT INTO studentTemp VALUES (2," . $dbh->quote ("Greg") . ")");

$st = $dbh->prepare("SELECT \* from studentTemp");

$st->execute();

print ("<table>\n");

while ($data = $st->fetchrow\_hashref()) {

print "<tr><td> $data->{NUM} </td><td> $data->{NAME} </td></tr>\n"; }

print ("</table></body></html>\n");

$st->finish();

$dbh->disconnect();

Note that we are now using two modules – DBI (Perl Data Base Interface) module provides the functions to connect to any database. It will automatically load the driver specific to the database system we are using (provided it is available). The configOracle module is the one that we defined just now, that holds our Oracle configuration parameters.

We then have the DBI->connect statement, that loads the Oracle driver as specified, and returns a handle to the driver, this handle is referred to as $dbh. Now we can execute statements that do not return rows (non-SELECT SQL queries) using $dbh->do (…). We drop a table, check if the drop was success or not, then we create a table, insert two rows. Also notice the usage of $dbh->quote for strings.

You must notice the way we issue SELECT statements. First we prepare a statement using $dbh->prepare that returns a statement handle $st. We then can execute this statement as $st->execute. When we execute the statement, it returns a set of rows. We can iterate over these rows and fetch them one by one. One way of doing it is using $st->fetchrow\_hashref () that returns a reference to a hash corresponding to the current row, with keys as the column names. We deference it using -> for example, we say $data->{NUM}. **BE CAREFUL** – Oracle is case-insensitive to names of tables, columns etc. So when it returns, it returns them as caps. Even though we defined the column as num, we retrieve it back as NUM !!