Perl/CGI

Perl/CGI is a server-side scripting language, used to run arbitrary programs within the web server. The web server is configured to understand the Perl scripts and to execute them. The web server on CCC is configured to connect to both mySQL as well Oracle 9.2 database servers through their respective Perl "modules".

Let us start with a simple Hello World Perl script, which I will call as helloWorld.pl

```
#!/usr/local/bin/perl

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 compiler exists. 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). You need to set the permissions of this to 700 from a unix prompt as:
chmod 700 helloWorld.pl

I can run this script from my shell, and see if it works. I can also put this in my public_html directory. I will create a subdirectory called perlSamples and change permissions of this directory to 755. I put helloWorld.pl in that directory. Then I can access it from my browser using:
http://www.wpi.edu/~mmani/perlSamples/helloWorld.pl

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

```
#!/usr/local/bin/perl

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. Try running this from both your shell and from your browser, and notice the difference in the environment variables. The following envVars.pl script will do the work for you.

```
#!/usr/local/bin/perl -w

use CGI::Carp qw(fatalsToBrowser warningsToBrowser);

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

foreach $key (keys %{ENV}) {

    print ("$key = $ENV{$key}<br>\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. We then interate over this array with $key taking the value of the key every time. $ENV{$key} gives the value in the hash corresponding to that key.

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 –

```
#!/usr/local/bin/perl

package configOracle;

use Exporter;
@ISA = ('Exporter');
@EXPORT = qw($host $sid $port $userName $passwd);

$host = "oracle.wpi.edu";
$sid = "cs";
$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. This file needs to have permissions of 600:

chmod 600 configOracle.pm

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.

```
#!/usr/local/bin/perl

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");

if ($ENV{HTTP_ACCEPT}) {
  $ENV{ORACLE_HOME} = "/usr/local/oracle/product/10.1.0/db_1";
}

$dbh = DBI->connect("DBI:Oracle:host=$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 new 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.

Note the funny looking condition statement that sets $ORACLE_HOME environment variable. That is necessary only for Oracle, it ensures that your ORACLE_HOME stays intact if you connect from your shell, and that you use the Oracle 10 files if you connect from a web server.

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 !!

Perl + mySQL

It is very similar to Perl + Oracle. However, the connection strings are different. Let us first define a configmySQL.pm with permissions 600 as –

```
#!/usr/local/bin/perl

package configmySQL;

use Exporter;
@ISA = ('Exporter');
@EXPORT = qw($schema $server $userName $passwd);

$server = "mysql.wpi.edu";
$schema = "...";
$userName = "...";
$passwd = "...";
```

We will now write a Perl script – testmySQL1.pl – that will connect to mySQL DB, and will mimic the functionality of the previous testOracle1.pl.

```
#!/usr/local/bin/perl

use CGI::Carp qw(fatalsToBrowser warningsToBrowser);
use DBI;
use configmySQL;

print "Content-type: text/html\n\n";
print ("<html><head><title>Test mySQL - 1</title></head>\n");
print ("<body>\n");

$dbh = DBI->connect("DBI:mysql:$schema:$server", $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 can now retrieve the data using $data->{num}, because mySQL is case sensitive.

Let us try to put together one Perl script, that will ask for user input, and will also display the current value in the tables. In short, it combines HTML forms with a Perl script that

will handle the form. We will call this as testOracle2.pl

```
#!/usr/local/bin/perl

use CGI::Carp qw(fatalsToBrowser warningsToBrowser);
use DBI;
use configOracle;
use CGI ":standard";

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

print ("<html><head><title>Test Oracle - 2</title></head>\n");
print ("<body>\n");
print ('<form method=post action=testOracle2.pl>');

if ($ENV{HTTP_ACCEPT}) {
  $ENV{ORACLE_HOME} = "/usr/local/oracle/product/10.1.0/db_1";
}

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

if (defined (param ("submit"))) {
    $rows = $dbh->do ("INSERT INTO studentTemp VALUES (" .
            $dbh->quote (param ("a1")) . ", " .
            $dbh->quote (param ("a2")) . ")");
    if (! defined ($rows)) { print ("error inserting
$DBI::errstr<br>\n"); }
}

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

print ("<table border=1>\n");
print ("<tr><th>number</th><th>name</th></tr>\n");
while ($data = $st->fetchrow_hashref()) {
    print "<tr><td> $data->{NUM} </td><td> $data->{NAME}
</td></tr>\n";
}

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

print <<ENDHTML;
<br><br>
Num: <input type="text" name="a1"><br>
Name: <input type="text" name="a2"><br><br>
<input type="Submit" name="submit" value="Enter Information"/>
ENDHTML

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

```
$st->finish();
$dbh->disconnect();
```

Main things to note – param (“a1”) gets the value corresponding to a1 that is passed from the form. This is available from the CGI module. Further, defined ($varName) checks whether a variable has been defined or not. You might not have to use defined typically.

Open the web page: **http://www.wpi.edu/~mmani/perlSamples/testOracle2.pl**

number	name
1	Matt
2	Greg

Num:

Name:

Enter Information