EX.NO:10b

DATE:

DATABASE CONNECTIVITY USING PERL AND MYSQL

Aim:

To create a database in Mysql and accessing and updating the table using Perl.

Introduction to PERL:

Structure of perl program

Perl program is a text file. You can use any text editor to create the program. Normally following line will be the first line

#! /usr/bin/perl

This tells Linux to use /usr/bin/perl executable to interpret rest of the lines in the program. This line may vary depending on the location of perl binary. Sometimes it may be /usr/local/bin/perl or some other place.

Commonly .pl extension is used, however you can write without extension also.

Starting with customary hello world program.

Use any editor and create a file with following contents

#! /usr/bin/perl

print "hello world\n";

Assuming that you have saved it as prog.pl, we will see how to run the program.

Running perl program

Perl programs can be run in two ways. Assuming prog.pl is your file then Method a:

$ perl prog.pl

Method b: Grant executable permission using chmod command and then invoking program name

$ chmod u+x prog.pl

$ ./prog.pl

or use full path name like

$ /home/ram/prog.pl

Note: For method b to work #! line is compulsory and ensure that #! occupies first and second character in the file.

If everything goes well you will see hello world in your prompt.

Now that we know how to create and run let us go into language details.

1.Comments:

symbol is used for comments. All text from # till end of line is treated as comment.

e.g

 # This is a full line comment

print "hello"; # This is statement+comment

Note: There is no multiline comment.

2. Statement terminator ;

All Statements end with ; like C language.

3.print

print is simple function to display/output something on monitor/stdout. e.g

print "hello world";

4.Variables

Variables are memory location to store information.

Variables are type less i.e there is no data type like int,char.

Every variable is a string and depending on the context will be treated as int, float etc.

There are 4 kinds of variables namely **scalars,lists,arrays,hashes.**

5 .Scalars

Scalar variables contain singular value like 10,hello etc

Name of scalar variable is prefixed with $ symbol.

eg.

$name=”ram”;# in string context

$age=30; # in numerical context

$age=$age+1; #treated as numeric

$age1=$age.$age; #treated as string. ‘.’(dot) is a concatenate operator

6.Handling quotes

” (double quote) is used when interpolation/substitution is required.

e.g

$name=”Raman”;

print “hello $name”;

will substitute $name with its value and output ‘hello Raman’.

‘ (single quote) is used when it is a literal string. Special characters will not be interpreted.

e.g

$name=’Raman’;

print ‘hello $name’;

The above line will print ‘hello $name’.

7.Lists

List variables are noted by symbol (). List is just a list of values – may be constants, scalars etc

e.g

(a,b,c) or ($name,$age,$sex)

They can be referred with index also. The index are specified inside a square bracket [ ].

e.g

$first=(a,b,c)[0];

print "$first\n";

will output a.

List variables can be assigned like this

($name,$age)=('Raman',20);

8.Conditionals – IF

The syntax of if statement is

if (condition) {

}

elsif (condition){

}

else {

}

The if statement is similar to if in C language, except

* flower brace is required even for single statement

* else if is noted by elsif (note missing e).

e.g

$mark=40;

if ($mark>75){

print “passed with distinction\n”;

}

elsif ($mark<35){

print “failed\n”;

}

else {

print “passed\n”;

}

Alternate form of *if* statement is

print “variable a is >10″ if ($a>10);

9.Accepting input

Keyboard inputs can be accepted using <STDIN>.

e.g

print “enter your name “;

$name=<STDIN>;

print “Welcome $name\n”;

Exercise:

Accept age. Type child if age below 12, type senior citizen when age above 60,otherwise type adult

10.Loops

for

for loop syntax is similar to c. It can also be used for iterating on a list. *foreach* is same as *for*. Both *for* and *foreach* are used interchangeably. for readability.

Classical *for* as in 'C'

e.g.

```
for ($i=0;$i<10;$i++){

print "i=$i\n";

}
```

The other way of using *for* is below.

```
foreach $i (a,b,c) {

print uc $i;

}
```

Explanation:

foreach will execute the body once for every element in the list – 3 times in this case. Each time the variable $i will get the value it is iterating ie. $i will be 'a' first time 'b' second time and 'c' the third time. uc – is a perl function to change a string into upper case.

You can combine functions like 'print uc $i' instead of print(uc($i)). Also note that brackets are optional for passing arguments to functions like uc, print.

The output will be ABC.

while

while loop is used to iterate and has syntax similar to C. e.g

```
$i=0;

while ($i<10){

print "i=$i\n";

$i++;

}
```

11. Default scalar variable $_

$_ is called default variable. It will be used if no other variable is specified. We will see this by an example.

e.g

foreach (a,b,c){

print uc ;

}

The above foreach is similar to what is given under section10, however $i is omitted. Still perl will output same i.e 'ABC'. This is because perl uses default variable $_ to store and expands the lines as

foreach $_ (a,b,c){

print uc $_;

}

Similarly $_ is used in the following case where '..' the generator function is used.

foreach (1..10){

print ;

}

12.Arrays

Arrays are used to store multiple ordered values. Array variables should have prefix @. The size of array need not be specified beforehand. Each element of the array is scalar. Index starts with zero.

Whenever the whole array is required @array will be used. Suppose we want only an element then $array[ ] will be used as every element is a scalar, thus the prefix $.

e.g

@array=(1,2,3);

print @array;

Operations on Array

Assignment – whole array using list

@array=(1,2,3);

print "@array";

Assigning element

$array[3]=4;

print "@array \n";

New element can be appended at the end using *push* function

e.g.

push @array,'4′;

print "@array \n";

Last element can be removed using *pop* function

e.g.

$last=pop @array;

print "last=$last\n";

First element can be removed using *shift* function

e.g

@array=(1,2,3);

$first=shift @array;

print "first=$first\n";

An element can be inserted at the beginning using *unshift*.

e.g

@array=(3,4,5);

unshift @array,'2′;

print "array=@array\n";

Looping contents of an array using foreach

e.g

@array=(1,2,3);

foreach $i (@array){

print $i;

}

$#array is a special variable containing index of last element.It will be -1 for an empty array. In the above example *$#array* will be 2.

scalar(@array) is function to return the size of array.

Classical for can be used for iterating on array like this

e.g

@array=(1,2,3);

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

print "i=$i array element=$array[$i]\n";

}

for ($i=0; $i<=$#array; $i++) {

print "i=$i array element=$array[$i]\n";

}

13.Hashes

Hash is associative/named array.They are key-value pairs. It is similar to array, except that we can use strings as index instead of 1..n. Hash variables will have **%** as prefix. The contents of hash are called values and index is called key.

e.g

%fruits= (

'apple' =>'red',

'banana'=>'yellow',

'grape' =>'black'

);

Other way of populating a hash

e.g

%fruits =('apple','red','banana','yellow','grape','black');

Here the list should contain even number of values. First element will be treated as key, second element value, third element key, fourth value and so on and so forth. In short odd elements will be keys, even elements will be values.

Individual elements of hash are accessed by means of *$hash{key}*

e.g.

print "colour of apple is $fruits{apple}\n";

Adding new element

e.g.

$fruits{'orange'}='orange';

Note the { } instead of [ ] as in the case of array;

Looping on hashes – keys function

e.g

foreach $f (keys %fruits) {

print “Color of $f is $fruits{$f}\n” ;

}

Explanation: keys is a function which return a list of key values.The list ( ) will contain apple,banana,grape while running.

14. Subroutines

Subroutines can be defined using sub keyword. The arguments passed will be in a default array @_;

e.g

$v1=10;$v2=20;

add($v1,$v2);

sub add {

($a,$b)=@_;

print $a+$b;

}

This should give output 30.

You can return value using return statement.

15. Scope of variables

By default all variables are global i.e available throughout the file. You can limit scope to a block/sub by using my.

e.g

$v1=10; $v2=30; #v1,v2 global

$v3=30;

$v3=add($v1,$v2);

```
sub add{

my ($i,$j)=@_;

print “inside add sub value of i=$i j=$j\n”;

print “inside add sub value of globals v1=$v1 v2=$v2 v3=$v3\n”;

return $i+$j;

}

print ” Value of globals v1=$v1 v2=$v3\n”;

print ” Value of scoped variables v3=$v3\n”;

print ” Value of variables inside sub i=$i j=$j\n”;
```

You can limit scope to a block also

e.g

```
for (my $i=0; $i<10; $i++ ) {

print “inside for i=$i\n”;

}

print “outside for i=$i\n”;
```

16.use strict

In perl you need not define variables before using. By default all variables are global. However, this may lead to errors due scope conflict or errors in naming. ‘*use strict*‘ is a pragma which will help in avoiding it. Once *use strict* is used, every variable has to be declared with proper scope using **my.**

e.g

```
use strict;

$v1=10;$v2=20;

add($v1,$v2);
```

```
sub add {

($a,$b)=@_;

print $a+$b;

}
```

The above code will not run and produce error. The corrected one will be like this

e.g

```
use strict;

my $v1=10;

my $v2=20;

add ( $v1,$v2 ) ;

sub add {

my ($a, $b)=@_;

print $a+$b;

}
```

17. References

References are address of the variable, similar (but not exactly) to pointers in c. You can take a reference by using **\.** It can be dereferenced by using **$$**;

e.g

```
$a=10;

$ref_toa=\$a;

print "value of a using reference = $$ref_toa\n Value of using directly=$a\n Reference of a= $ref_toa";
```

18.File handling

File handling can be done after opening a file and getting handle similar to C.

e.g

open($fh, “<”, “data.txt”);

here *$fh* – file handle which is a scalar variable. Also it is common to uppercased variable like FH.

< – open read only

data.txt – name of the file. Full path name should be given it it is not in current directory

File reading line by line can be done like

$line=<$fh>;

File writing

print $fh “hello”;

Example Problem: Open data.txt file. Copy contents to udata.txt duly converted into upper case

e.g

open ($fh, “<”, “data.txt”); #open file read only

open ($fh1,”>”,”udata.txt”); #Open file write mode

while ($line = <$fh>) { #read line by

print “line=$line”; #display content on screen

print $fh1 uc($line); #write upper cased content to new file

}

close($fh);

close($fh1);

19.Some common functions

uc :uc is used to convert string into all upper cases

e.g

$name=’raman’;

$name=uc ($name);

print “name=$name\n”;

Similarly lc will convert into lowercase.

split: split is function which will split a string into components based on delimiter specified, and return a list of values.

eg. $line=”20/12/2010";

($day,$month,$year)=split /\//,$line;

print “day=$day month=$month year=$year\n”;

In the above example split will use / as delimiter. Note that / is special character. If we want literal / then we should use \/.

length: length returns the number of characters in a string.

20.Context

In perl many operations/functions perform differently based on context. For example @array will have different meaning like

@array=(‘a’,'b’,'c’);

print “array=@array\n”;

In the above line perl will print contents of the array.

$size=@array;

print “array = @array and size=$size\n”;

In the above example perl will put 3 into $size i.e the number of elements in @array because it is used in scalar context (singular value).

Similarly split will return different values according to context. In the previous split returned a list of values. If used in scalar context split will return number of values i.e count

$line=”hello how are you”;

print “number of words=” . split / /,$line . “\n”;

CREATING A TABLE IN PERL THAT WILL BE AUTOMATICALLY UPDATED IN DATABASE USING MYSQL

Procedure:

1.1 First create the database as follows in mysql

[linuxpert@localhost ~]$ mysql -u root -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 2

Server version: 5.1.45 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement. mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information_schema |

| mysql |

| test |

+--------------------+

3 rows in set (0.25 sec)

mysql> create database student; Query OK, 1 row affected (0.02 sec)

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| information_schema |

| mysql |

| student |

| test |

+--------------------+

4 rows in set (0.00 sec)

mysql> use student; Database changed mysql> connect; Connection id: 3

Current database: student

mysql> show tables; Empty set (0.00 sec)

now the table is empty .

To create the table we use the following procedure

1.2 The following packages should be used to connect PERL with Mysql

(use new terminal)

[linuxpert@localhost ~]$ rpm -q perl-DBI

perl-DBI-1.609-4.fc13.i686

[linuxpert@localhost ~]$ rpm -q perl-DBD-MySQL

perl-DBD-MySQL-4.013-3.fc13.i686

1.3 Write the PERL script to connect with mysql as follows

```
#!usr/bin/perl
use DBI; #to use the build in package we use "Use", DBI is the build in package in perl
my $dbh=DBI->connect("dbi:mysql:student","root",""); #connect to database
if(!$dbh)
{
die("error:$!");
}
$sth=$dbh->prepare("create table students(rollno int,sname varchar(50))");
# create the table
```

$sth->execute();

$dbh->disconnect;

1.4 Run the Perl script [linuxpert@localhost ~]$ perl connect.pl **now see the tables in database ("student")**

mysql> show tables;

+------------------+

| Tables_in_student |

+------------------+

| students |

+------------------+

1 row in set (0.00 sec)

2. insert the values in perl that will be automatically updated in database using mysql as follows

#!usr/bin/perl

use DBI; #to use the build in package we use "Use", DBI is the build in package in perl

my $dbh=DBI->connect("dbi:mysql:student","root",""); #connect to database

if(!$dbh)

{

die("error:$!");

}

$sth=$dbh->prepare("insert into students values(100,'thamarai')"); # create the table

$sth->execute();

$dbh->disconnect;

2.1 compile the perl [linuxpert@localhost ~]$ perl dbinsert.pl **now the output is**

mysql> select * from students;

+--------+----------+

| rollno | sname |

+--------+----------+

| 100 | thamarai |

+--------+----------+

1 row in set (0.00 sec)

2.2 insert the values in perl using execute statement

#!usr/bin/perl

use DBI; #to use the build in package we use "Use", DBI is the build in package in perl

$rollno=200;

$sname="selvi";

my $dbh=DBI->connect("dbi:mysql:student","root",""); #connect to database

if(!$dbh)

{

die("error:$!");

}

$sth=$dbh->prepare("insert into students values(?,?)"); # create the table

$sth->execute($rollno,$sname);

$dbh->disconnect; **compile the program as** [linuxpert@localhost ~]$ perl dbinsert1.pl **now the output is**

mysql> select * from students;

+--------+----------+

| rollno | sname |

+--------+----------+

| 100 | thamarai |

| 200 | selvi |

+--------+----------+

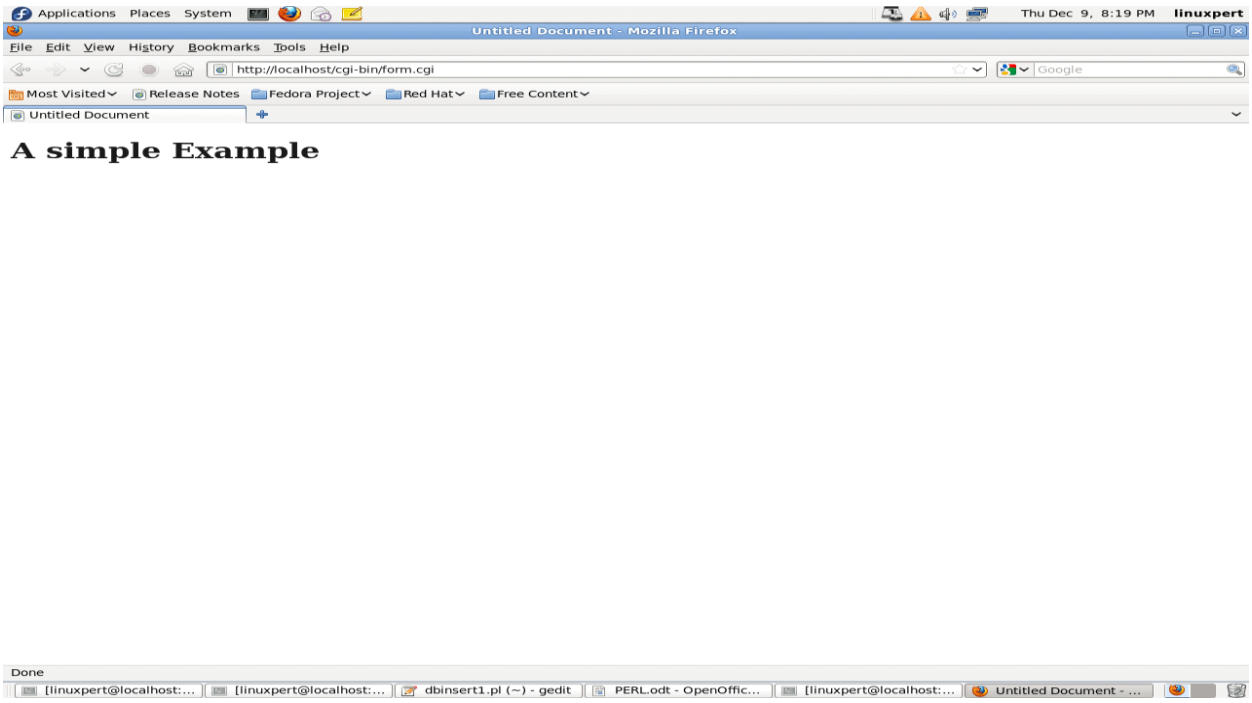
2 rows in set (0.08 sec)

Display the output in the browser using CGI

type the following in terminal [linuxpert@localhost ~]$ su Password:
[root@localhost linuxpert]# cd /var/www/cgi-bin
[root@localhost cgi-bin]# gedit
type the following in gedit

```
#!/usr/bin/perl use CGI;
$cgi=new CGI;
print $cgi->header,
$cgi->start_html,
$cgi->h1("A simple Example"),
$cgi->end_html;
```

type the following in the same terminal
[root@localhost cgi-bin]# chmod +x form.cgi
go to browser and type the following URL
http://localhost/cgi-bin/form.cgi



in terminal [linuxpert@localhost ~]$ su Password:
[root@localhost linuxpert]# cd /var/www/html
[root@localhost html]# gedit
type the following in gedit editor

```
<html>
<head>
<title>LOGIN</title></head>
<body>
<form action="/cgi-bin/form2.cgi" method="post">
<p>
"Enter student roll no"<input type="text" name="rollno"></p>
<p>"enter the student name"<input type="text" name="sname"></p>
<p>"click here to submit"<input type="submit" name="submit"></p>
</form>
</body>
```

save the page as form.html & close it **type the following URL in browser**
http://localhost/form.html

then type the following in the terminal

[root@localhost html]# cd /var/www/cgi-bin

[root@localhost cgi-bin]# gedit

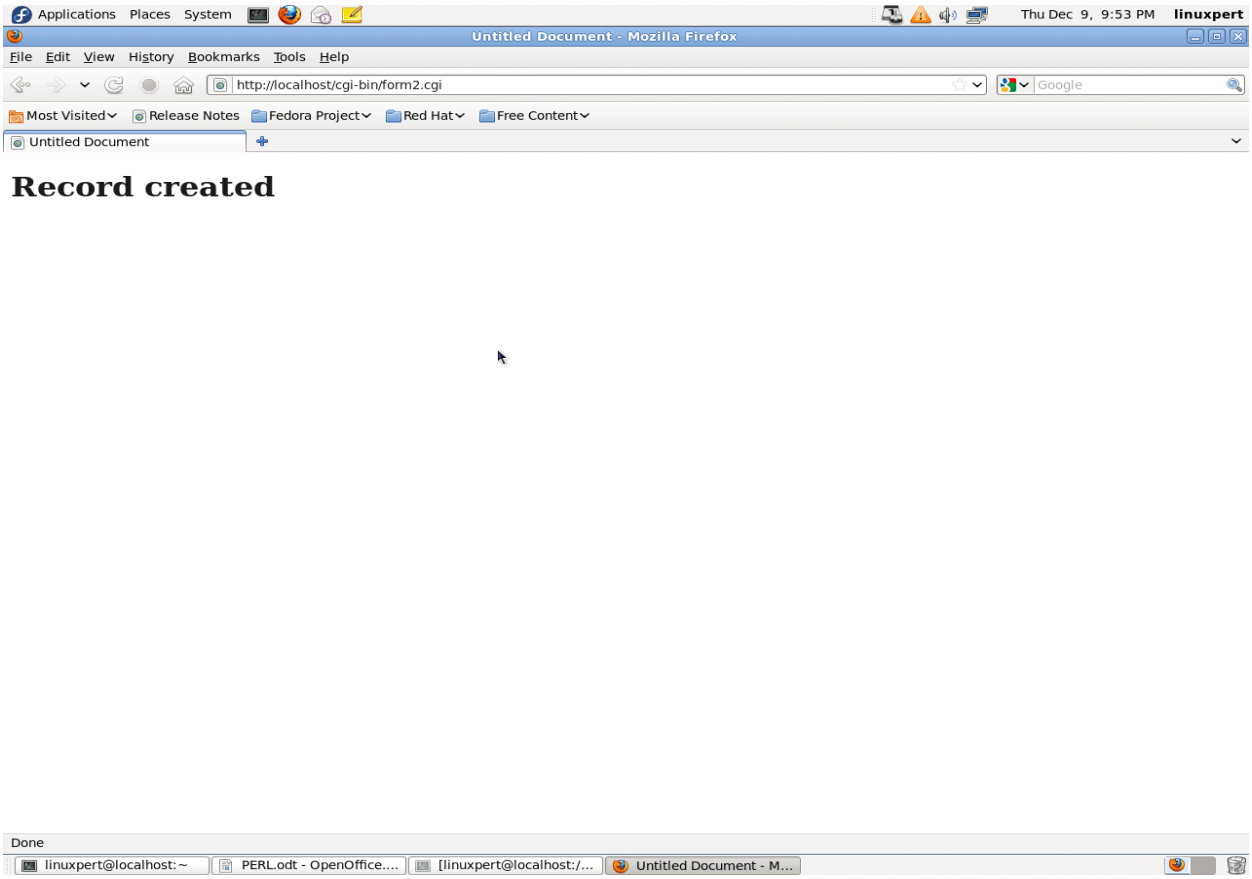
type the following in gedit

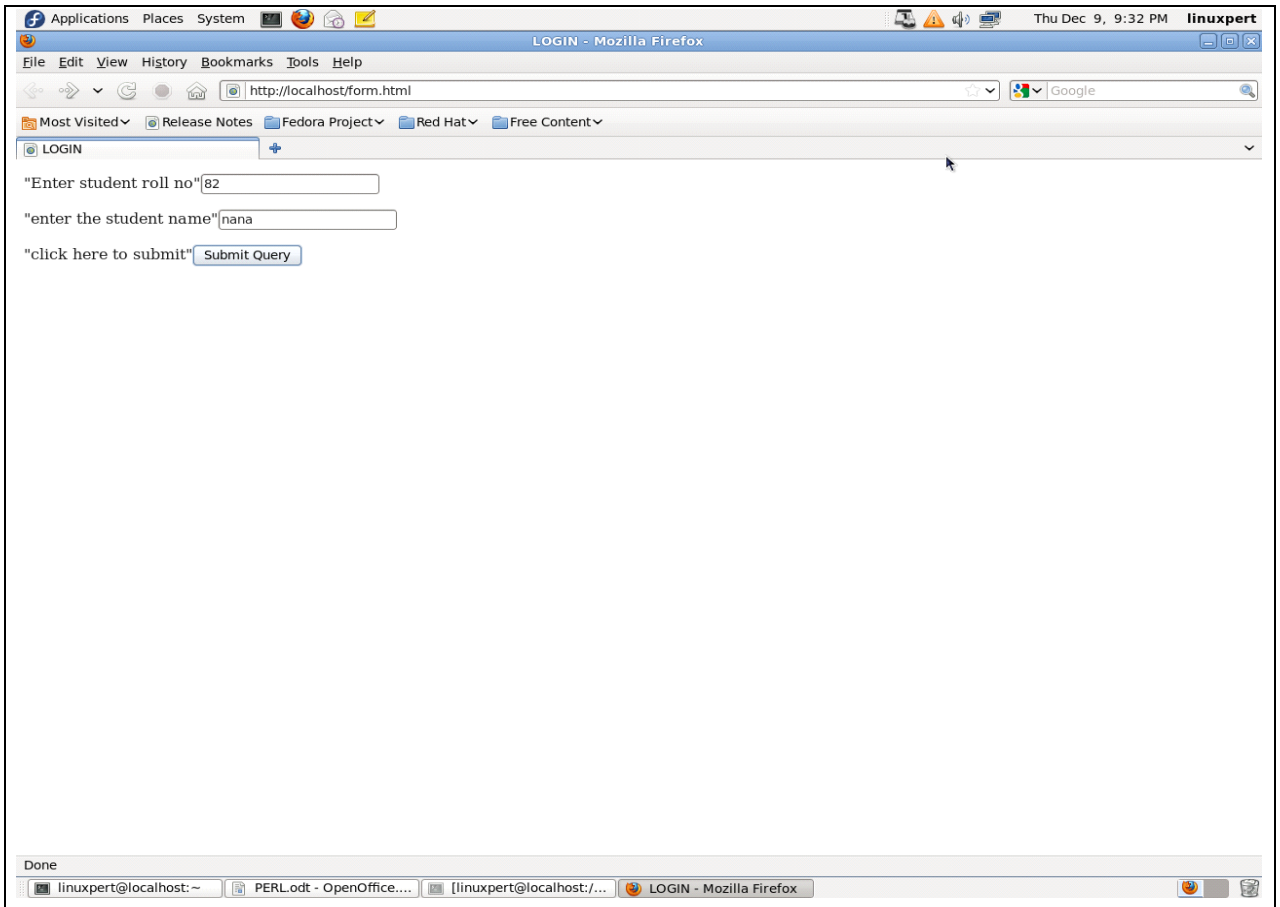
```
#!/usr/bin/perl use CGI;
$cgi=new CGI;
use DBI;
$rollno=$cgi->param('rollno');
$name=$cgi->param('sname');
my $dbh=DBI->connect("dbi:mysql:student","root","");
my $sth=$dbh->prepare("insert into students values(?,?)");
$res=$sth->execute($rollno,$name);
$dbh->disconnect;
if($res)
{
print $cgi->header,
$cgi->start_html,
$cgi->h1("Record created"),
$cgi->end_html;
}
```

save that file as form2.cgi

close that file & open a same terminal type as [root@localhostcgi-bin]# chmod +x form2.cgi **In new browser type the following** http://localhost/form.html

Output:


Applications Places System
Thu Dec 9, 9:53 PM linuxpert
Untitled Document - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/cgi-bin/form2.cgi
Google
Most Visited Release Notes Fedora Project Red Hat Free Content
Untitled Document
Record created
Done
linuxpert@localhost:~
PERL.odt - OpenOffice....
[linuxpert@localhost:/...
Untitled Document - M...



now check the database as

mysql> select * from students;

```
+--------+---------------+
| rollno | sname |
+--------+---------------+
| 100 | thamarai |
| 200 | selvi |
| NULL | abcd |
| 71 | thamaraiselvi |
+--------+---------------+
4 rows in set (0.00 sec)
```

RESULT:

Thus the database connectivity using Perl with MySQL are implemented and executed successfully.