

**Class Note 7**  
**PHP Oracle Development**  
**(Updated 6/4/2015)**

**Oracle Query and Bind Variables**

The “class note” is the typical material I would prepare for my face-to-face class. I am sharing the notes with everyone in the class.

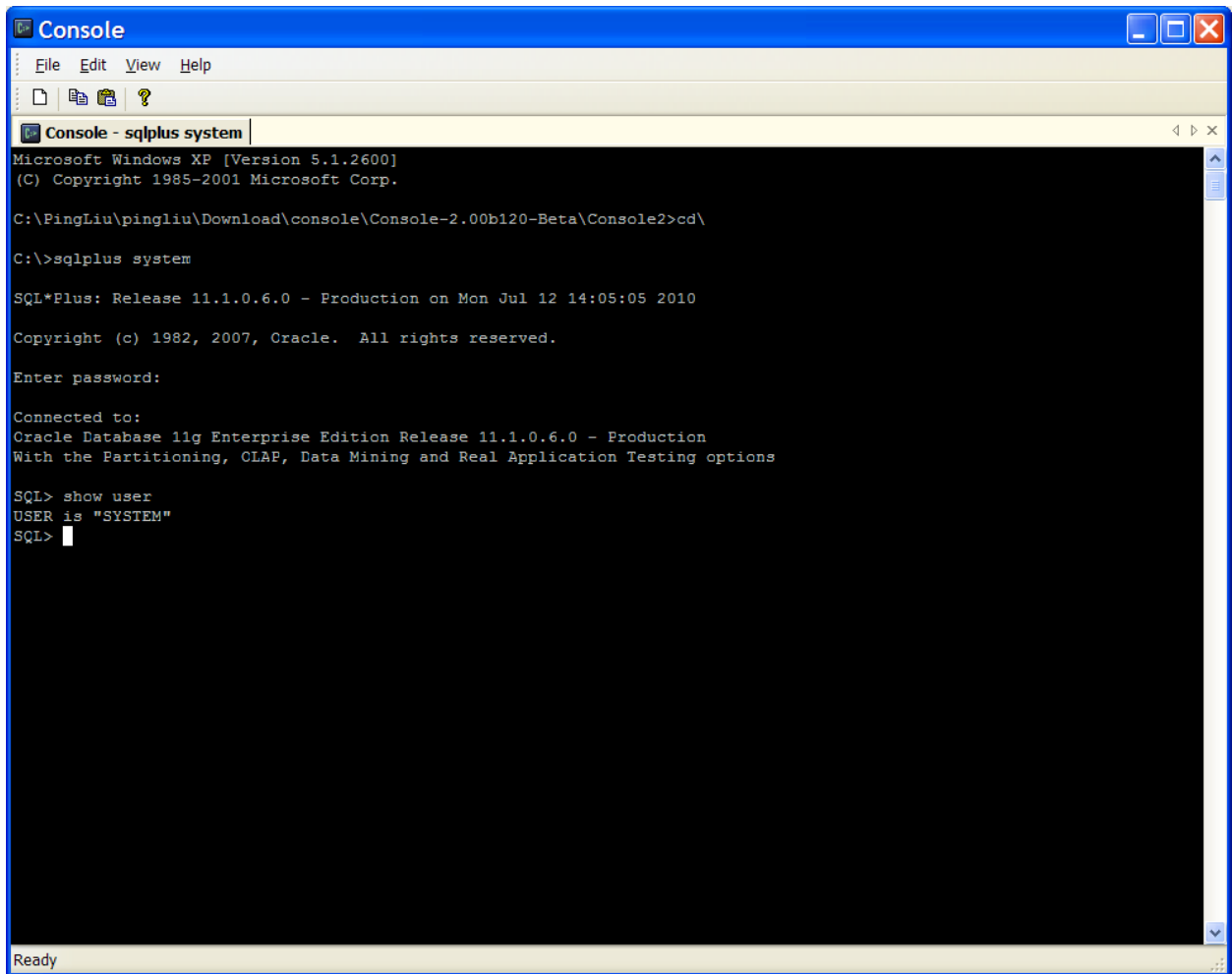
For the best result of your study, please make sure you do the following:

1. Read the chapter(s) in your textbook, corresponding to the week.
2. Read this class note. Please understand that I will only highlight the important points from your chapters in the textbook. I do not intend to repeat it.
3. Study the “Review Questions” for each week. I intend to combine the chapter content and class notes in the “Review Questions” section. Many of the concepts can be better understood if you fully understand the “Review Questions.”

**I. Prepare Your Database for the Class**

In order for us to do the exercises and your project for this week, you need to set up your database so that it has the tables to be used in this class. Some of the details may be trivial if you have already taken a class on Oracle database. But, I am going to do it any way for those students who are fresh to Oracle.

- a. The first step is you need to log in your Oracle SQL Plus in your computer, using user name: SYSTEM and the password you entered while installing your Oracle database.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\PingLiu\pingliu\Download\console\Console-2.00b120-Beta\Console2>cd\

C:\>sqlplus system

SQL*Plus: Release 11.1.0.6.0 - Production on Mon Jul 12 14:05:05 2010

Copyright (c) 1982, 2007, Oracle. All rights reserved.

Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> show user
USER is "SYSTEM"
SQL>
```

After you log in as SYSTEM, you may issue the following SQL command:

```
SQL>show user;
```

You will confirm that you have logged in as SYSTEM.

If you do not remember it, you may ask others on Geeks Corner.

- b. You need to create the user “php” in your account. If you follow the class notes last week and you did your project successfully, you have already accomplished this step. Just in case, it is not a bad idea to refresh your database anyway.

To create the “php” user, you may use the SQL script that comes with your textbook. The script you need is from Appendix G. Please note that the CD has another file with the same file name. But, that file is not for this purpose.

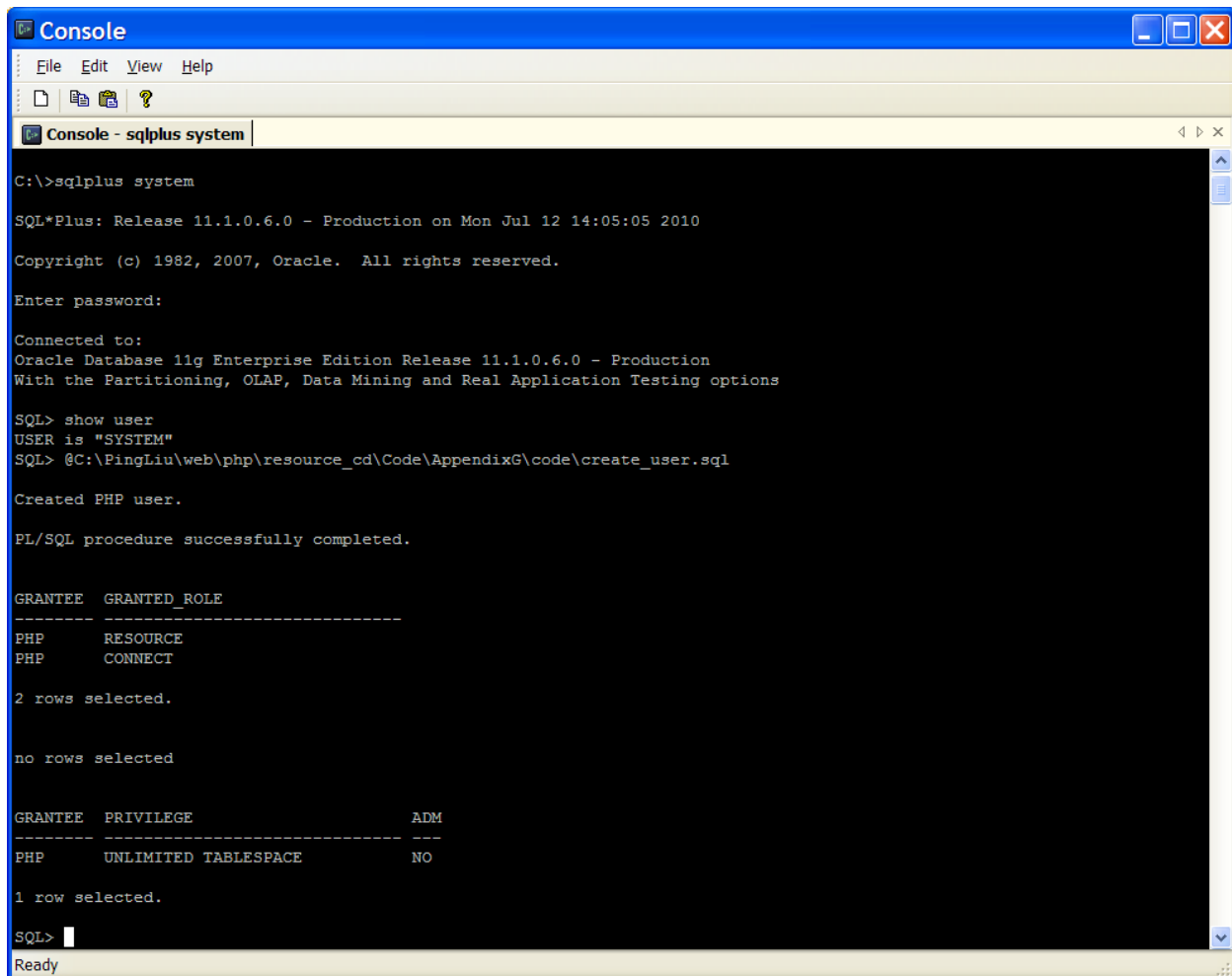
Now, to run the user creation script ([create\\_user.sql](#) in Appendix G), you may issue a SQL statement, similar to the following:

```
SQL> @C:\PingLiu\web\php\resource_cd\Code\AppendixG\code\create_user.sql
```

Please note that your file path may be different from mine above.

(@C:\PingLiu\web\php\resource\_cd\Code\AppendixG\code is the folder in my computer where the script was kept.)

If everything goes well, you will see something similar to my screen shot, as follows:



```
Console
File Edit View Help
Console - sqlplus system
C:\>sqlplus system
SQL*Plus: Release 11.1.0.6.0 - Production on Mon Jul 12 14:05:05 2010
Copyright (c) 1982, 2007, Oracle. All rights reserved.
Enter password:
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL> show user
USER is "SYSTEM"
SQL> @C:\PingLiu\web\php\resource_cd\Code\AppendixG\code\create_user.sql
Created PHP user.
PL/SQL procedure successfully completed.

GRANTEE  GRANTED_ROLE
-----
PHP      RESOURCE
PHP      CONNECT

2 rows selected.

no rows selected

GRANTEE  PRIVILEGE          ADM
-----
PHP      UNLIMITED TABLESPACE  NO

1 row selected.
SQL>
Ready
```

That means, the “php” user was created. The password for the “php” user is “php.”

- c. At this point, your “php” user (schema) has nothing (neither tables nor data). You need to create tables and populate the tables with data.

**It is super critical that you log in as “php” before you create the tables.** You need to issue:

```
SQL> connect php/php;
```

After the “php” is connected to the database, you may run the script ([create\\_store.sql](#))

```
SQL> @C:\PingLiu\web\php\resource_cd\Code\AppendixJ\code\create_store.sql
```

Please note if the script was kept in Appendix J from the textbook CD.

(Note for Geeks: If you are trained as a system administrator, you should modified the above script for its spooling function as follows:

```
SPOOL c:\temp\create_store.log
```

I like to add c:\temp\ so that the file will be spooled to the temp directory instead of C:\. If you do not care, that is fine.)

Now, if you issue the following statement to SQL Plus:

```
SQL>select * from tab;
```

You will see those tables you have just created. Nonetheless, those tables are still empty with no data. You may insert (populate) data into those tables using the following [script](#):

```
SQL>@C:\PingLiu\web\php\resource_cd\Code\AppendixJ\code\seed_store.sql
```

Please note if the script was kept in Appendix J from the textbook CD.

If you examine the output (either on screen or the spooled file), you will see there is one view that was not created, as intended by the script. I will wait if someone in the class likes to resolve the problem. For the time being, it does not affect our work.

The following SQL statements will be handy if you want to see the content of tables:

```
SQL> desc item;
```

You will see the table structure including column names from the “item” table.

```
SQL> select * from item;
```

You will see the content of “item” table. It may not be pretty if the table is huge.

```
SQL> select item_id, item_type, item_title from item;
```

You will see only the selected columns such as item\_id, item\_type, and item\_title.

Now, the database is ready for action.

## II. Query on Oracle Database

To learn how to query a database, we are going to start with the example in your textbook, [SelectItemRecords.php](#). But, we will make some changes based upon what we have learned from previous week.

We will use the file oracle\_parameters.php to define your connection parameters. As we recall, the oracle\_parameters.php is shown below, using my computer as an example:

```
<?php
//The schema defines the user name you like to access in Oracle database
$schema='php';

//It defines the password of the above schema/user
$password='php';

//It defines the database (instance) name
$database='(DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST =
154862EIU.eiuad.eiu.edu)(PORT =1521))) (CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME =
oracle) (INSTANCE_NAME = oracle)))';
?>
```

Your “\$database” string may be different depending upon your settings. Refer the notes from previous week.

Thus, the code will be modified as follows:

```
<?php
include ('oracle_parameters.php');

// Connect to the database.
if ($c = @oci_connect("$schema","$password","$database")) {
    // Parse a query to a resource statement.
    $s = oci_parse($c,"SELECT    i.item_id id
                                ,      i.item_barcode as barcode
                                ,      c.common_lookup_type as type
                                ,      i.item_title as title
                                ,      i.item_rating as rating
                                ,      i.item_release_date release_date
FROM      item i inner join common_lookup c
ON        i.item_type = c.common_lookup_id
WHERE     c.common_lookup_type = 'XBOX'
ORDER BY i.item_title");

    // Execute query without an implicit commit.
    oci_execute($s,OCI_DEFAULT);

    // Open the HTML table.
    print '<table border="1" cellspacing="0" cellpadding="3">';

    // Read fetched headers.
    print '<tr>';
    for ($i = 1;$i <= oci_num_fields($s);$i++)
        print '<td class="e">'.oci_field_name($s,$i).'</td>';
    print '</tr>';
```

```

// Read fetched data.
while (oci_fetch($s))
{
    // Print open and close HTML row tags and columns data.
    print '<tr>';
    for ($i = 1;$i <= oci_num_fields($s);$i++)
        print '<td class="v">'.oci_result($s,$i).'</td>';
    print '</tr>';
}

// Close the HTML table.
print '</table>';

// Disconnect from database.
oci_close($c);
}
else
{
    // Assign the OCI error and format double and single quotes.
    $errorMessage = oci_error();
    print htmlentities($errorMessage['message'])."<br />";
}
?>

```

Please note the changes:

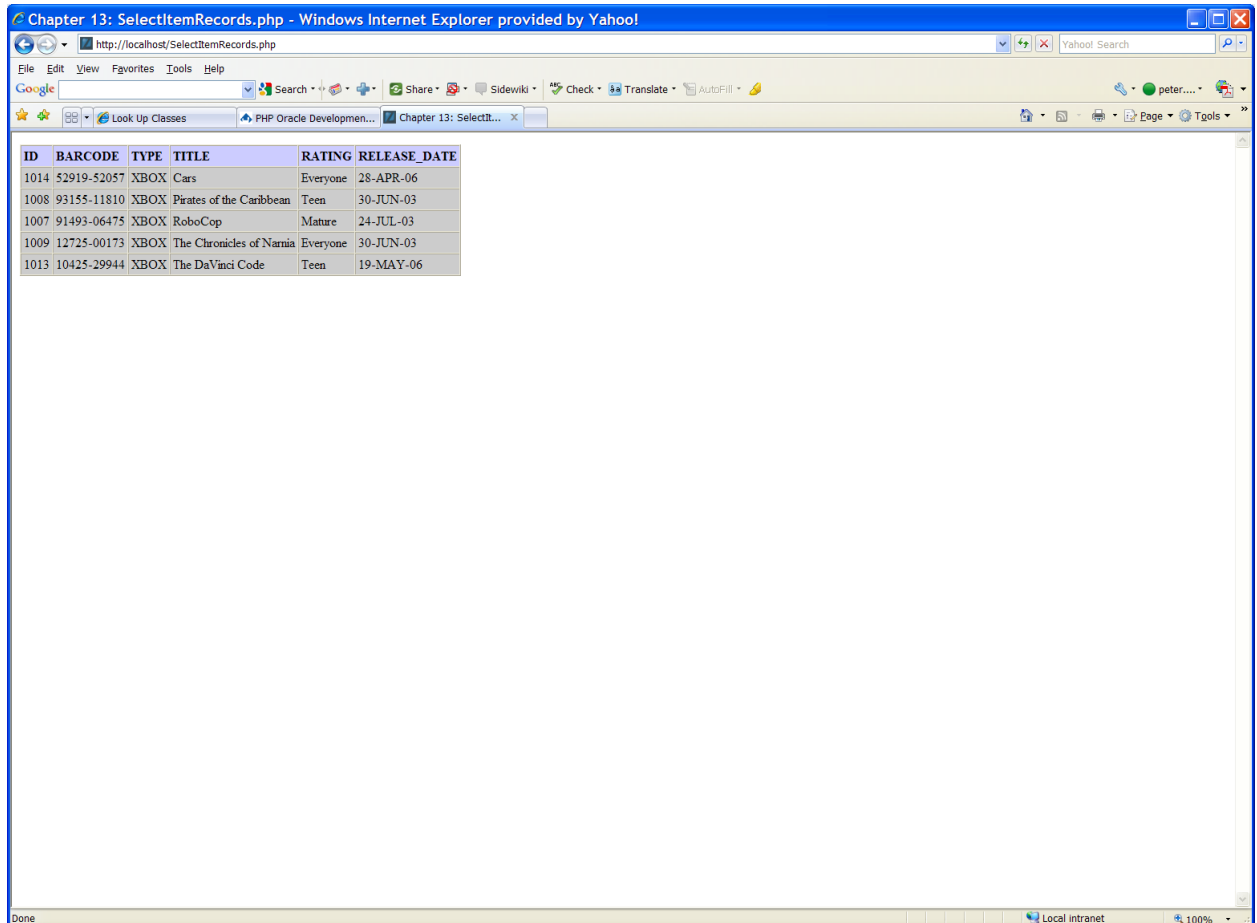
a. Added on include statement:

```
include ('oracle_parameters.php');
```

b. Changed the database connection:

```
if ($c = @oci_connect("$schema", "$password", "$database"))
```

The output of the above code will look something similar to the following:



The screenshot shows a web browser window titled "Chapter 13: SelectItemRecords.php - Windows Internet Explorer provided by Yahoo!". The address bar shows "http://localhost/SelectItemRecords.php". The page content displays a table with the following data:

ID	BARCODE	TYPE	TITLE	RATING	RELEASE_DATE
1014	52919-52057	XBOX	Cars	Everyone	28-APR-06
1008	93155-11810	XBOX	Pirates of the Caribbean	Teen	30-JUN-03
1007	91493-06475	XBOX	RoboCop	Mature	24-JUL-03
1009	12725-00173	XBOX	The Chronicles of Narnia	Everyone	30-JUN-03
1013	10425-29944	XBOX	The DaVinci Code	Teen	19-MAY-06

My point was just to show you that I tested everything before I say it. That is a trait of being trained as an engineer. Just kidding, but seriously.

Now, let us get serious and spend a few seconds to understand what does the above code do, step-by-step.

- (\$c = @oci\_connect("\$schema", "\$password", "\$database")): It connects to the Oracle database defined by user name (\$schema), password for user (\$password), and the database connection string (\$database).

You now have seen the word oci\_ in the statement since it is an OCI8 function.



- b. `$s = oci_parse($c,"SELECT ...)`: It puts the SQL (Structured Query Language) statement (stuff within the double quote) in the computer memory. The fancy term is to parse the SQL statement. Just imagine that the SQL statement was checked to see it has any syntax problems and then store the SQL statement for action.

Do not be intimidated by the SQL statement within the double quote. They are beyond the scope of this course. If you still want to know, you may ask those who took TEC 5323 Advanced Database Technology. (See more details below. In the meantime, focus on the OCI logic flow to get a solid understanding on it.)

- c. `oci_execute($s,OCI_DEFAULT)`; It runs the above SQL statement, query in this case.
- d. `“for ($i = 1;$i <= oci_num_fields($s);$i++)”` and `“print '<td class="e">'.oci_field_name($s,$i).”` The first for loop print out the column names. Here it uses `“oci_num_fields ()”` and `“oci_field_name”`. You can see here the roles of the two functions are pretty self-explanatory.
- e. While loop: It prints out the results from the above SQL statement. It uses `oci_result ()` function.
- f. `oci_close()`: It closes the connection after the query is done.
- g. Error message was handled by the following code:

```
$errorMessage = oci_error();  
  
print htmlentities($errorMessage['message'])."<br />";
```

You may test it by making some change in the `SelectItemRecord.php` code to see how the error message is displayed. It will help you to debug your program in the future.

## Structured Query Language (SQL) Statement

In practical database programming, once you have done once and fully understand the sequence such as the above, you will find it pretty straight forward to work with the database through your programming language (PHP in this case). Since most programmers know the language well, they sometimes have difficulty in debugging and performing quality assurance of the software system. Most of the visible and invisible errors come from the SQL statement used with PHP program.

My only suggestion for the above challenge is to test the SQL statement before you put it in the PHP program. Let us use the above PHP code as one example. The SQL statement in the `oci_parse ()` statement is:

```
SELECT    i.item_id id
          ,      i.item_barcode as barcode
          ,      c.common_lookup_type as type
          ,      i.item_title as title
          ,      i.item_rating as rating
          ,      i.item_release_date release_date
FROM      item i inner join common_lookup c
ON        i.item_type = c.common_lookup_id
WHERE     c.common_lookup_type = 'XBOX'
ORDER BY i.item_title
```

Please note that I have stripped off the double quotes and other stuff outside the SQL statement. Moreover, I kept the original spaces, which do not affect the actual programming.

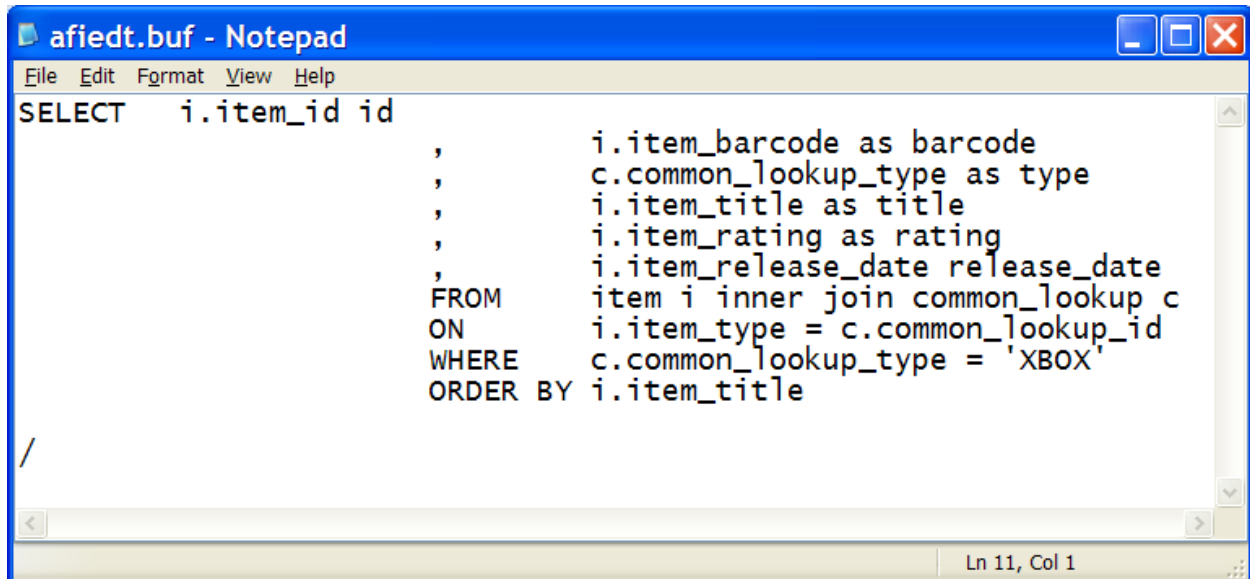
The most important thing you need to make sure is that the SQL statement does exactly what you want it to do. The main suggestion I have is the same as I always give: **test it**. Here I am talking about testing with the database (not your PHP program).

To test the SQL statement, of course, you need to log in your SQL Plus. One simple way is to issue a SQL command as follows:

```
SQL>ed (enter)
```

After you enter the above command, SQL Plus will invoke a text editor in your computer. In Windows, the default text editor is notepad. Vi editor is the default for Unix system. Those default settings can be changed or customized as you wish.

You may then copy your SQL statement to the text editor and make whatever changes you like. After the changes, you may save it and exit from the text editor.

A screenshot of a Notepad window titled "afiedt.buf - Notepad". The window has a menu bar with "File", "Edit", "Format", "View", and "Help". The main text area contains the following SQL query:

```
SELECT  i.item_id id
        ,      i.item_barcode as barcode
        ,      c.common_lookup_type as type
        ,      i.item_title as title
        ,      i.item_rating as rating
        ,      i.item_release_date release_date
FROM    item i inner join common_lookup c
ON      i.item_type = c.common_lookup_id
WHERE   c.common_lookup_type = 'XBOX'
ORDER BY i.item_title
```

The status bar at the bottom right shows "Ln 11, Col 1".

Upon exit from the text editor, you will be going back to Oracle SQL Plus. Then you can issue another command to run the SQL statement, as follows:

```
SQL>/
```

You will see the results on your SQL Plus screen.

```

Console
File Edit View Help
Console - sqlplus system
1 row selected.

SQL> ed
Wrote file afiedt.buf

 1 SELECT  i.item_id id
 2          ,          i.item_barcode as barcode
 3          ,          c.common_lookup_type as type
 4          ,          i.item_title as title
 5          ,          i.item_rating as rating
 6          ,          i.item_release_date release_date
 7 FROM    item i inner join common_lookup c
 8 ON      i.item_type = c.common_lookup_id
 9 WHERE   c.common_lookup_type = 'XBOX'
10*
11 ORDER BY i.item_title

-----
ID BARCODE          TYPE
-----
TITLE              RATING  RELEASE_D
-----
1014 52919-52057   XBOX
Cars              Everyone 28-APR-06

1008 93155-11810   XBOX
Pirates of the Caribbean Teen    30-JUN-03

1007 91493-06475   XBOX
RoboCop           Mature  24-JUL-03

1009 12725-00173   XBOX
The Chronicles of Narnia Everyone 30-JUN-03

1013 10425-29944   XBOX
The DaVinci Code  Teen    19-MAY-06

5 rows selected.

SQL>
Ready

```

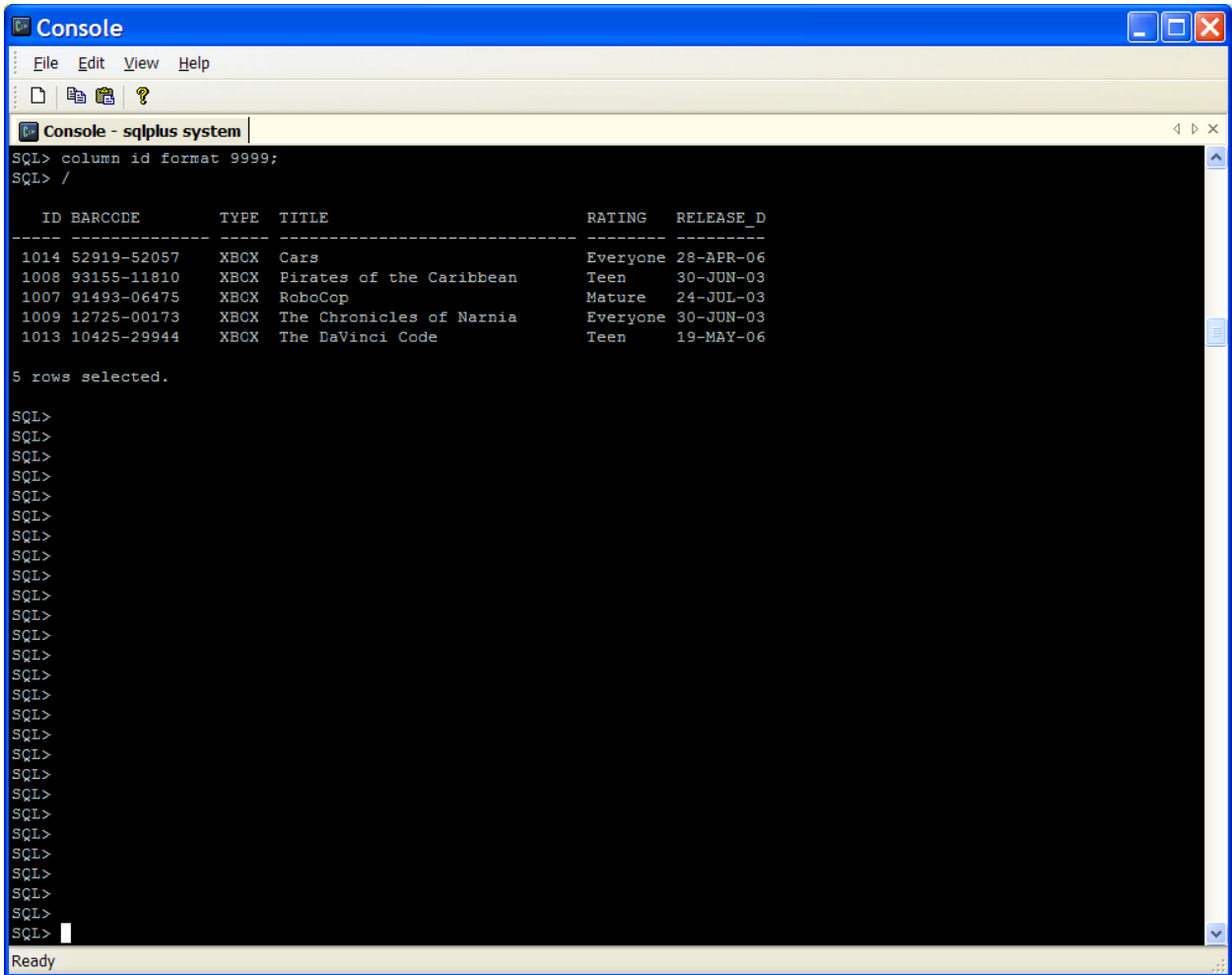
The above display seems to be out of place a little. That is only the format issue. If you like to have a nice display of the results. You may try the following SQL commands:

```

SQL> column type format a5;
SQL> column id format 9999;

```

You may see something a little better organized.



```
SQL> column id format 9999;
SQL> /

  ID BARCODE      TYPE TITLE                                RATING  RELEASE_D
-----
1014 52919-52057   XBOX Cars                                Everyone 28-APR-06
1008 93155-11810   XBOX Pirates of the Caribbean            Teen     30-JUN-03
1007 91493-06475   XBOX RoboCop                              Mature   24-JUL-03
1009 12725-00173   XBOX The Chronicles of Narnia            Everyone 30-JUN-03
1013 10425-29944   XBOX The DaVinci Code                     Teen     19-MAY-06

5 rows selected.

SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
```

Ready

The point is that you need to test the SQL statement, not only for the syntax errors but also for the accuracy of the programming. If the SQL statement gives you incorrect results, there is no way you can tell. I have seen this multiple times.

## Simple User Authentication Application

Now, I am going to use a simpler example, hoping that you may be able to create your own web application in near future.

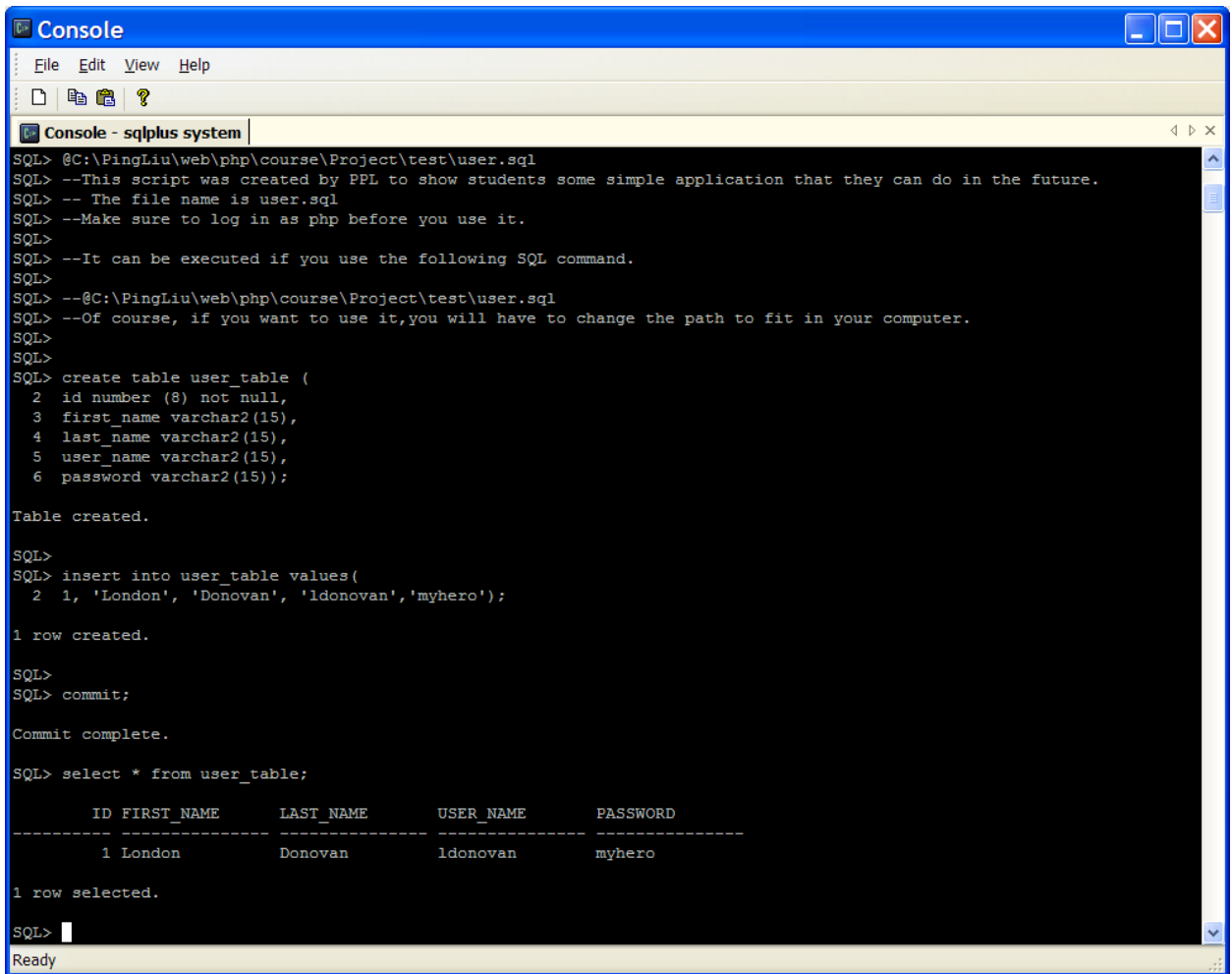
Let us create a table called “user\_table”, in which we like to store some basic user information including first name, last name, user name on the system, and the password. As an example, we use the following [script](#) to create the table and insert data for one user.

```
create table user_table (  
id number (8) not null,  
first_name varchar2(15),  
last_name varchar2(15),  
user_name varchar2(15),  
password varchar2(15));  
  
insert into user_table values(  
1, 'London', 'Donovan', 'ldonovan','myhero');  
  
commit;
```

Since I created and saved the above code in my  
“C:\PingLiu\web\php\course\Project\test\” folder. I can run it as follows:

```
SQL>@C:\PingLiu\web\php\course\Project\test\user.sql.
```

After running the above script, you will be able to see the data using SQL Plus.



```
SQL> @C:\PingLiu\web\php\course\Project\test\user.sql
SQL> --This script was created by PPL to show students some simple application that they can do in the future.
SQL> -- The file name is user.sql
SQL> --Make sure to log in as php before you use it.
SQL>
SQL> --It can be executed if you use the following SQL command.
SQL>
SQL> --@C:\PingLiu\web\php\course\Project\test\user.sql
SQL> --Of course, if you want to use it,you will have to change the path to fit in your computer.
SQL>
SQL>
SQL> create table user_table (
  2 id number (8) not null,
  3 first_name varchar2(15),
  4 last_name varchar2(15),
  5 user_name varchar2(15),
  6 password varchar2(15));

Table created.

SQL>
SQL> insert into user_table values(
  2 1, 'London', 'Donovan', 'ldonovan','myhero');

1 row created.

SQL>
SQL> commit;

Commit complete.

SQL> select * from user_table;

-----
      ID FIRST_NAME      LAST_NAME      USER_NAME      PASSWORD
-----
         1 London          Donovan          ldonovan          myhero

1 row selected.

SQL>
Ready
```

Now, if we change the above PHP code (I saved it as [SimpleSelection.php](#)) with the following SQL statement.

```
select * from user_table;
```

The code becomes:

```
<?php
include ('oracle_parameter.php');

// Connect to the database.

if ($c = @oci_connect("$schema","$password","$database"))
{
    // Parse a query to a resource statement.
    $s = oci_parse($c,"SELECT * from user_table");
```

```

// Execute query without an implicit commit.
oci_execute($s,OCI_DEFAULT);

// Open the HTML table.
print '<table border="1" cellspacing="0" cellpadding="3">';

// Read fetched headers.
print '<tr>';
for ($i = 1;$i <= oci_num_fields($s);$i++)
    print '<td class="e">'.oci_field_name($s,$i).'</td>';
print '</tr>';

// Read fetched data.
while (oci_fetch($s))
{
    // Print open and close HTML row tags and columns data.
    print '<tr>';
    for ($i = 1;$i <= oci_num_fields($s);$i++)
        print '<td class="v">'.oci_result($s,$i).'</td>';
    print '</tr>';
}

// Close the HTML table.
print '</table>';

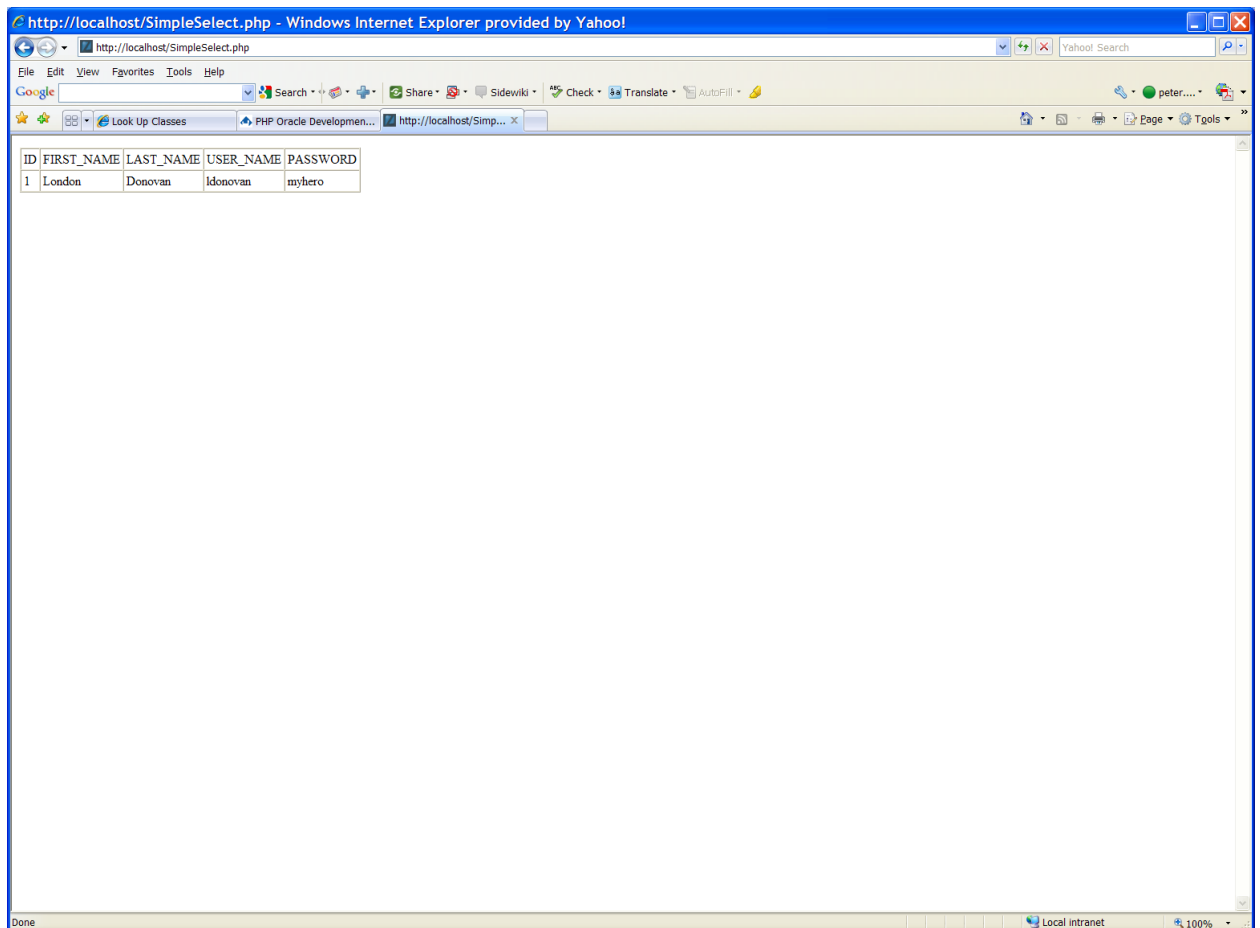
// Disconnect from database.
oci_close($c);
}
else

```



```
{  
    // Assign the OCI error and format double and single quotes.  
    $errorMessage = oci_error();  
    print htmlentities($errorMessage['message'])."<br />";  
}  
?>
```

The result of the PHP code will be similar to the following:



Now, what if we change the SQL statement within the above PHP code to:

```
SELECT user_name, password from user_table
```

You will be able to find out the user name and password only.

Certainly, at this point, we feel we can use PHP program to talk to Oracle database and get data out of the database.

Let us go a little further, to make the following change:

```
SELECT password from user_table WHERE user_name='ldonovan'
```

That is, if we specify the user\_name (for example, "ldonovan"), we will be able to find and compare with its password. That is a part of login or authentication. It gets more exciting.

### III. Oracle Bind Variables

In the above section, by using the following SQL statement:

```
SELECT password from user_table WHERE user_name='ldonovan'
```

We could get the password for the user "ldonovan".

If you try the following SQL statement,

```
SELECT password from user_table WHERE user_name='thoward'
```

It will give us nothing since the user "thoward" did not exist in the table.

What the above actions tell us is that in our PHP program, we could SPECIFY the SQL condition to get the information we want, in the above examples, to get the passwords of two users ("ldonovan" and "thoward").

The trick now is how do we SPECIFY the condition? Do you want to your user to go in your PHP code and make similar changes as we did? Of course, the above activities were meant only for learning purpose. Oracle's answer to the challenge is to use "bind variables." You may read more descriptions about the bind variables from your textbook on pages 379-388.

We will use the above simple PHP code, to illustrate the concept. With the introduction of bind variable, the above PHP code becomes something as follows:

```
<?php
//include the Oracle connection parameter file.
include ('oracle_parameters.php');
```

```

//define the PHP input variable ($username)
$username='ldonovan';

// Connect to the database.
if ($c = @oci_connect("$schema","$password","$database"))
{
    // Parse a query to a resource statement. An Oracle bind variable is used
    (:username).
    $s = oci_parse($c,"SELECT password from user_table WHERE user_name=:username");

//Connect (Bind) the PHP input variable ($username) with Oracle bind variable
(:username).
oci_bind_by_name($s, ":username", $username, -1, SQLT_CHR);

// Execute query without an implicit commit.
oci_execute($s,OCI_DEFAULT);

// Open the HTML table.
print '<table border="1" cellspacing="0" cellpadding="3">';

// Read fetched headers.
print '<tr>';
for ($i = 1;$i <= oci_num_fields($s);$i++)
    print '<td class="e">'.oci_field_name($s,$i).'</td>';
print '</tr>';

// Read fetched data.
while (oci_fetch($s))
{
    // Print open and close HTML row tags and columns data.
    print '<tr>';

```

```

    for ($i = 1;$i <= oci_num_fields($s);$i++)
        print '<td class="v">'.oci_result($s,$i).'

```

For the sake of discussion, I renamed the PHP code as [SimpleBind.php](#). Let us look at the steps of using bind variable.

- a. The first step is you need to define a PHP input variable. In the above program, we used the following:

```

//define the PHP input variable ($username)
$username='ldonovan';

```

Here, we defined a PHP variable called \$username. In web programming, this variable may be defined and passed from previous HTML form, as we did in the chapter dealing with function. The above definition is just the simplest example of defining a PHP variable.

- b. The second step is to use (define) an Oracle bind variable within your SQL statement. In the above example, we used:

```
// Parse a query to a resource statement. An Oracle bind variable is used
(:username).

    $s = oci_parse($c,"SELECT password from user_table WHERE
user_name=:username");
```

Here we used a bind variable called :username. Please note the difference among those variable names we have used to refer to the same thing including:

\$username is a PHP variable.

:username is an Oracle bind variable.

user\_name is the column name (data field) in Oracle database table "user\_table"

Clear understanding on the above three things is very essential for your actual programming.

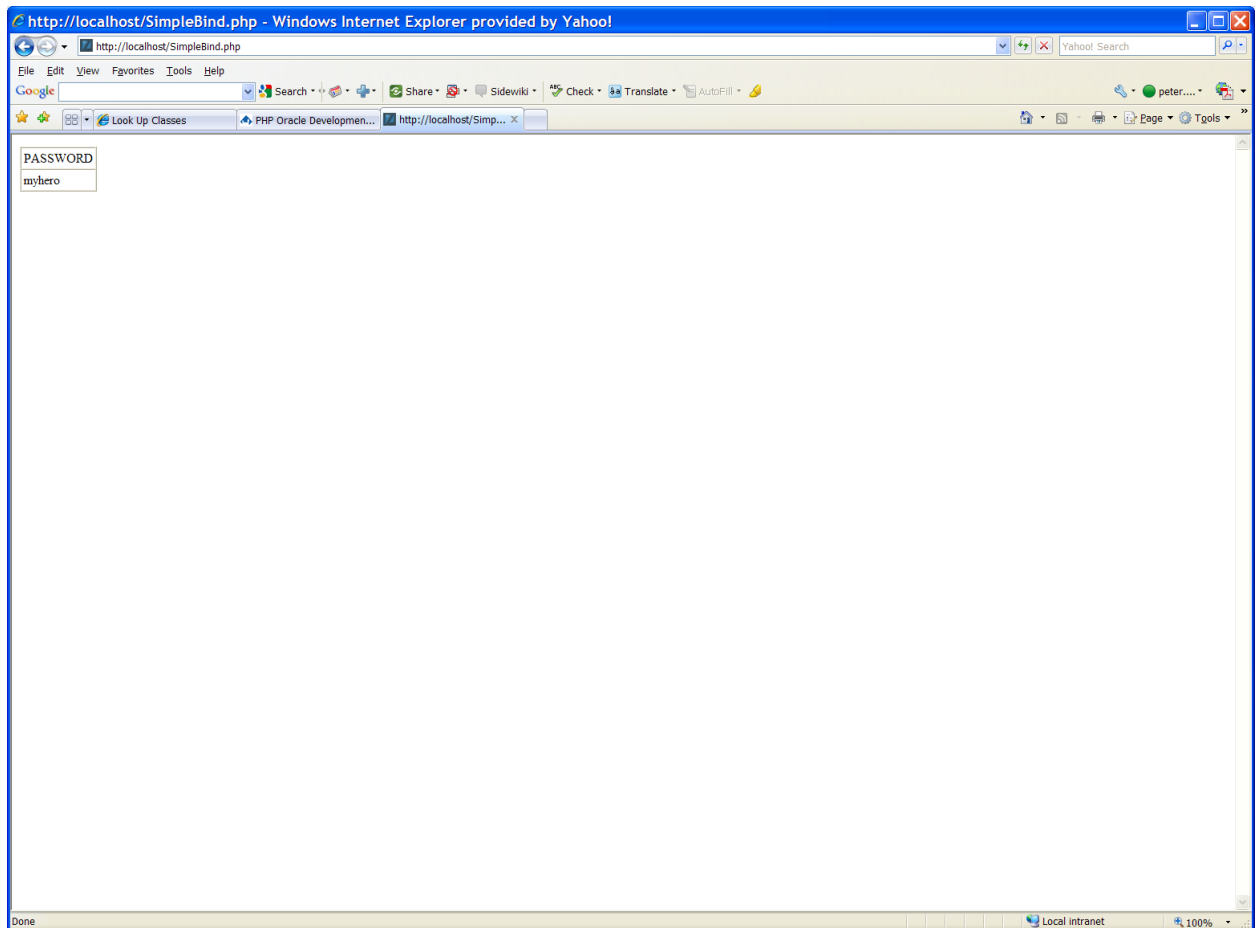
- c. The third step is connect the above two variables together.

```
//Connect (Bind) the PHP input variable ($username) with Oracle bind variable
(:username).

oci_bind_by_name($s, ":username", $username, -1, SQLT_CHR);
```

where we used an Oracle function known as "oci\_bind\_by\_name()" to do the trick. "\$s" refers the query we parsed earlier. We map the two variables (":username" and "\$username") together by using the oci\_bind\_by\_name() function.

The rest of the story is the same as before. Your resulting output will be like the following:



To make use of the above script for authentication purpose, I am going to modify the above the PHP program by adding some control structure. For example, we can define a PHP variable called `$userpwd` and compare the variable with the query result from the database table (`user_table`). After stripping off some of the printing format tags, here is the [sample code](#) for fun.

```
<?php
//include the Oracle connection parameter file.
include ('oracle_parameter.php');

//define the PHP input variables ($username and $userpwd)
//The variables can be defined from a HTML file or in other ways.
$username='ldonovan';
$userpwd='myhero';

// Connect to the database.
if ($c = @oci_connect("$schema","$password","$database"))
{
// Parse a query to a resource statement. An Oracle bind variable is used
(:username).
```

```

    $s = oci_parse($c,"SELECT password from user_table WHERE
user_name=:username");

//Connect (Bind) the PHP input variable ($username) with Oracle bind variable
(:username).
oci_bind_by_name($s, ":username", $username, -1, SQLT_CHR);

// Execute query without an implicit commit.
oci_execute($s,OCI_DEFAULT);

// Read fetched data.
while (oci_fetch($s))
{
    for ($i = 1;$i <= oci_num_fields($s);$i++)

//print '<td class="v">'.oci_result($s,$i).'</td>';
if ($userpwd==oci_result($s,$i))
{
print "The user matches his/her password.";
}
}

// Disconnect from database.
oci_close($c);
}
else
{

// Assign the OCI error and format double and single quotes.
$errorMessage = oci_error();
print htmlentities($errorMessage['message'])."<br />";
}
?>

```

What we did was we simply utilize the query results from the database table, as in the following block:

```

//The variables can be defined from a HTML file or in other ways.
$username='ldonovan';
$userpwd='myhero';

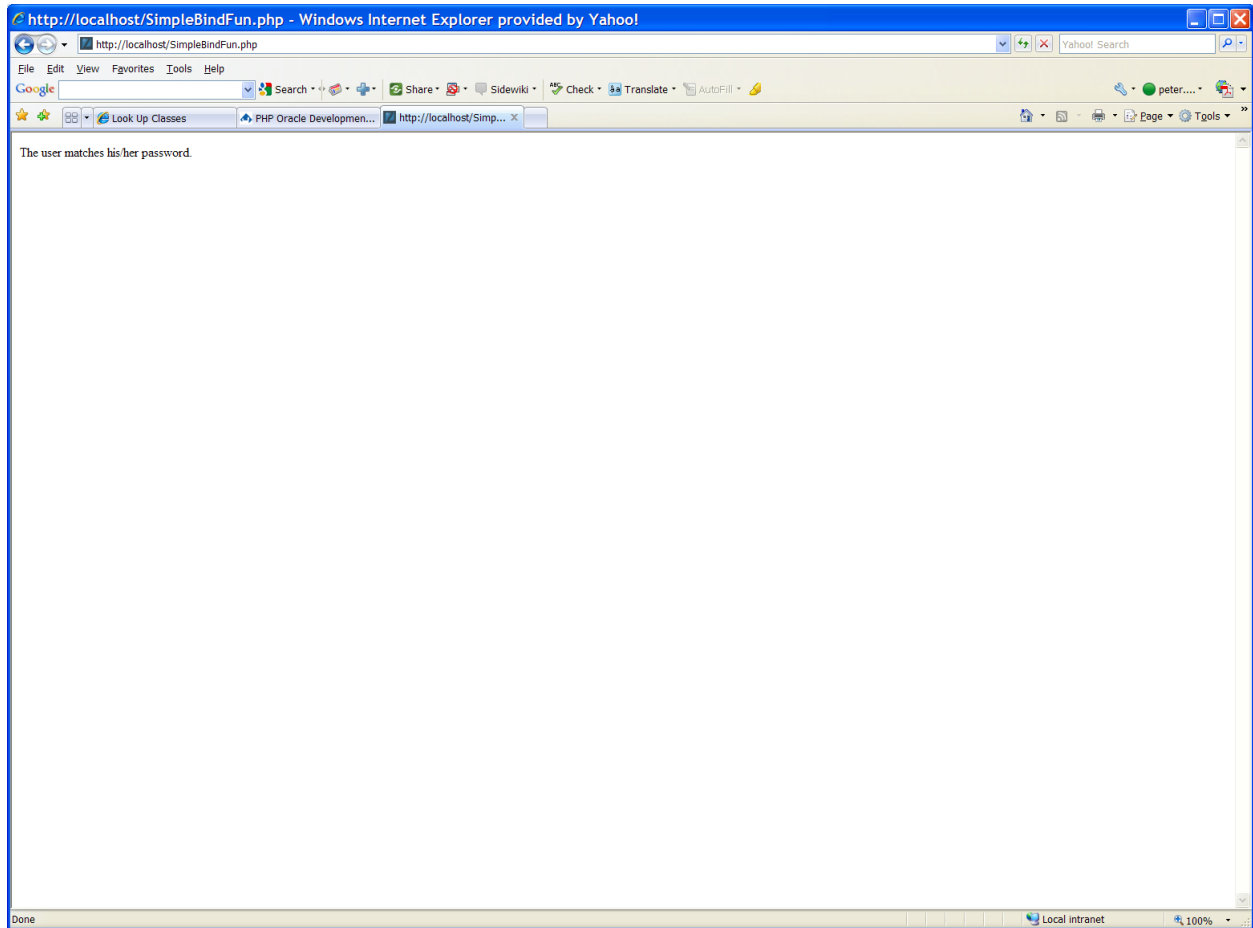
...

if ($userpwd==oci_result($s,$i))
{
print "The user matches his/her password.";
}

...

```

The result of the code will be as follows:



The above simple example is not to the professional grade of program yet. But, it points out the potential and direction for future applications. Sky is the limit for future. We will learn how to fly.

Have fun.