# Lab # 12: Web Development using Database

**Objective:**

- PHP Arrays, Loops
- PHP Functions
- PHP connection with oracle database

**Scope:**

The student should know the following:

PHP connection with oracle
Select data from database
Display data using PHP

## 1. PHP Arrays

An array stores multiple values in one single variable.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1="Toyota";
$cars2="Hyundai";
$cars3="BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The best solution here is to use an array!

An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

Each element in the array has its own index so that it can be easily accessed.

In PHP, there are three kinds of arrays:

- **Numeric array** - An array with a numeric index
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

## 1.1 Numeric Arrays

A numeric array stores each array element with a numeric index.

There are two methods to create a numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

2. In the following example we assign the index manually:

```
$cars[0]="Saab";
```

```
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

## Example 1.1

In the following example you access the variable values by referring to the array name and index:

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

The code above will output:

```
Saab and Volvo are Swedish cars.
```

## 1.2 Associative Arrays

An associative array, each ID key is associated with a value.

With associative arrays we can use the values as keys and assign values to them.

## Example 1.2

In this example we use an array to assign ages to the different persons:

```
$ages = array("A"=>32, "B"=>30, "C"=>34);
```

## Example 1.3

This example is the same as example 2.2, but shows a different way of creating the array:

```
$ages['A'] = "32";
$ages['B'] = "30";
$ages['C'] = "34";
```

The ID keys can be used in a script:

```
<?php
$ages['A'] = "32";
$ages['B'] = "30";
$ages['C'] = "34";

echo "A is " . $ages['B'] . " years old.";
?>
```

The code above will output:

```
A is 32 years old.
```

## 1.3 Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

## Example 1.4

In this example we create a multidimensional array, with automatically assigned ID keys:

```
$shops = array
  (
  "flowers"=>array( "rose", "daisy", "orchid"  ),
  "prices"=>array  (  "1.5",  "0.75",  "1.15"  )
  );
```

The array above would look like this if written to the output:

```
Array
(
    [flowers] => Array
        (
            [0] => rose
            [1] => daisy
            [2] => orchid
        )

    [prices] => Array
        (
            [0] => 1.5
            [1] => 0.75
            [2] => 1.15
        )

)
```

## Example 1.5

Lets try displaying a single value from the array above:

```
echo $shops['flowers'][0] .' costs '. $shops['prices'][0];
```

The code above will output:

```
rose costs 1.5
```

# 1. PHP Loops

Loops execute a block of code a specified number of times, or while a specified condition is true.
Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.
In PHP, we have the following looping statements:

- **while** - loops through a block of code while a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

## 1.1 The while Loop

The while loop executes a block of code while a condition is true.
**Syntax**

```
while (condition)
  {
  code to be executed;
  }
```

## Example 1.1

The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>
<?php
$i=1;
while($i<=5)
  {
  echo "The number is " . $i . "<br />";
  $i++;
  }
?>
</body>
</html>
```

Output:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

## 1.2 The do...while Statement

The do...while statement will always execute the block of code once, it will then check the condition, and repeat the loop while the condition is true.
**Syntax**

```
do
  {
  code to be executed;
  }
while (condition);
```

## Example 1.2

The example below defines a loop that starts with i=1. It will then increment i with 1, and write some output. Then the condition is checked, and the loop will continue to run as long as i is less than, or equal to 5:

```
<html>
<body>
<?php
$i=1;
do
  {
  $i++;
  echo "The number is " . $i . "<br />";
  }
while ($i<=5);
?>
</body>
</html>
```

Output:

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

## 2.4 The for Loop

The for loop is used when you know in advance how many times the script should run.

**Syntax**

```
for (init; condition; increment)
  {
  code to be executed;
  }
```

Parameters:

- *init*: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
- *condition*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment*: Mostly used to increment a counter (but can be any code to be executed at the end of the loop)

**Note:** Each of the parameters above can be empty, or have multiple expressions (separated by commas).

## Example 2.6

The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>
<?php
for ($i=1; $i<=5; $i++)
  {
  echo "The number is " . $i . "<br />";
```

```
  }
?>
</body>
</html>
```

Output:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

## 2.5 The foreach Loop

The foreach loop is used to loop through arrays.

**Syntax**

```
foreach ($array as $value)
  {
  code to be executed;
  }
```

For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.

## Example 2.7

The following example demonstrates a loop that will print the values of the given array:

```
<html>
<body>
<?php
$x=array("one","two","three");
foreach ($x as $value)
  {
  echo $value . "<br />";
  }
?>
</body>
</html>
```

Output:

```
one
two
three
```

Displaying array (from example 1.5) using for loop;

## Example 2.8

```
<html>
<body>
<?php
```

```
echo "<h2>Manual access to each element from associative array</h2>";

for ($row = 0; $row < 3; $row++)
{
  echo $shops['flowers'][$row] .' costs '. $shops['prices'][$row];
   echo "<br />";
}
?>
</body>
</html>
```

# PHP Functions

In this lecture we will show you how to create your own functions.

To keep the script from being executed when the page loads, you can put it into a function.

A function will be executed by a call to the function.

You may call a function from anywhere within a page.

## 3.1 Create a PHP Function

A function will be executed by a call to the function.

**Syntax**

function *functionName*()
{
*code to be executed*;
}

PHP function guidelines:

- Give the function a name that reflects what the function does
- The function name can start with a letter or underscore (not a number)

### Example 3.1

A simple function that writes my name when it is called:

```
<html>
<body>
<?php
function display_name()
{
echo "My name is Ali";
}
display_name ();
?>
</body>
</html>
```

Output:

```
My name is Ali
```

### 3.2 PHP Functions - Adding parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable.

Parameters are specified after the function name, inside the parentheses.

### Example 3.2

The following example will write different first names:

```
<html>
<body>
<?php
function display_name($fname)
{
echo "My name is ". $fname ;
}
display_name (“Ali”);
?>
</body>
</html>
```

Output:

```
My name is Ali
```

### Example 3.3

The following function has two parameters:

```
<html>
<body>
<?php
function display_name($fname,$lname)
{
echo "My name is ". $fname . ' '. $lname;
}
display_name ("Ali","Ahmed");
?>
</body>
</html>
```

Output:

```
My name is Ali Ahmed
```

### 3.3 PHP Functions - Return values

To let a function return a value, use the return statement.

```
<html>
<body>
<?php
function add($x,$y)
{
$total=$x+$y;
return $total;
}
 echo "1 + 16 = " . add(1,16);
?>
</body>
</html>
```

Output:

```
1 + 16 = 17
```

# 4. Using Databases

To use a database or query some table in a database, following are the typical steps to do;

1. Create a connection to the database.
2. If connection successful, write a query statement.
3. Parse this query using connection handler and query statement (note that this step is specific to ORACLE only).
4. Execute query.
5. Fetch the result set generated by the query.
6. Loop over the result set and display data using html and PHP.

## 4.1 Step 1 (Create a Connection to the database)

In PHP, this is done with the [db type]_connect() function.

[db type]_connect(servername, username, password);

| Parameter | Description |
|---|---|
| servername | Specifies the server to connect to. |
| username | Specifies the username to log in with. Default value is the name of the user that owns the database server |
| password | Specifies the password to log in with. |

**Note:** oracle 11g servername can be found in:
(oraclebase)\app\Your_username\product\11.2.0\dbhome_1\NETWORK\ADMIN\ tnsnames.ora

```
ORACLR_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
    (CONNECT_DATA =
      (SID = CLRExtProc)
      (PRESENTATION = RO)
    )
  )

ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = Turab-PC)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )
```

### Example 4.1

Create a web page (blank page) with the name as dbcon.php in your 'htdocs' directory.

Type the following code in the file and run it in the browser.

```
<html>
<body>
<?php
// example 2.1 ..creating a database connection
$db_sid = "(DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST = Turab-PC)(PORT = 1521)) )
(CONNECT_DATA = (SID = orcl) ) )";

$db_user = "scott";
$db_pass = "1234";

$con = oci_connect($db_user,$db_pass,$db_sid);
if($con)
{
            echo "connection successful.";
}
else
{
             die('Could not connect: ');
}
?>
</body>
</html>
```

In this example we store the connection in a variable ($con) for later use in the script. The "die" part will be executed if the connection fails.

So now if the connection is successful, we'll now proceed to query the EMP table as follows;

## 4.2 Step 2 (Write a query)

### Example 4.2

Write the bold line in your file as follows;

```
<html>
<body>
<?php
// example 2.1 ..creating a database connection
  $db_sid = "(DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST = Turab-PC)(PORT = 1521)) )
(CONNECT_DATA = (SID = orcl) ) )";

$db_user = "scott";
$db_pass = "1234";
```

```
$con = oci_connect($db_user,$db_pass,$db_sid);
if($con)
{
            echo "connection successful.";
}
else
{
             die('Could not connect: ');
}

$q = "select * from emp";

?>
</body>
</html>
```

### 4.3 Step 3 (Parse a query)

Prepares SQL (Oracle) statement for execution.

resource **oci_parse** ( resource connection, string query );

| Parameter | Description |
|---|---|
| connection | An Oracle connection |
| identifier. query | The SQL query. |

Prepares the `query` using `connection` and returns the statement identifier on success, or **`FALSE`** on error.

**Note** that this function *does not* validate query. The only way to find out if query is valid SQL statement is to execute it.

### Example 4.3

Write the bold line in your file as follows;

```
<html>
<body>
<?php
// example 2.1 ..creating a database connection
$db_sid = "(DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST = Turab-PC)(PORT = 1521)) )
(CONNECT_DATA = (SID = orcl) ) )";

$db_user = "scott";
$db_pass = "1234";

$con = oci_connect($db_user,$db_pass,$db_sid);
if($con)
{
            echo "connection successful.";
}
else
{
             die('Could not connect: ');
```

```
}

$q = "select * from emp";

$query_id = oci_parse($con, $q);

?>
</body>
</html>
```

### 4.4 Step 4 (Execute a query)

Executes a statement.

bool **oci_execute** ( resource statement [,mode] );

| Parameter | Description |
|---|---|
| Statement | A valid OCI statement identifier. |
| mode | Allows you to specify the execution mode (default is OCI_COMMIT_ON_SUCCESS). If you don't want statements to be committed automatically, you should specify OCI_DEFAULT as your mode.<br><br>When using OCI_DEFAULT mode, you're creating a transaction. Transactions are automatically rolled back when you close the connection, or when the script ends, whichever is soonest. You need to explicitly call oci_commit() to commit the transaction, or oci_rollback() to abort it. |

Returns **TRUE** on success or **FALSE** on failure.

### Example 4.4

Write the bold line in your file as follows;

```
<html>
<body>
<?php
// example 2.1 ..creating a database connection

$db_sid = "(DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST = Turab-PC)(PORT = 1521)) )
(CONNECT_DATA = (SID = orcl) ) )";

$db_user = "scott";
$db_pass = "1234";

$con = oci_connect($db_user,$db_pass,$db_sid);
if($con)
{
            echo "connection successful.";
}
else
{
             die('Could not connect: ');
}
```

```
$q = "select * from emp";

$query_id = oci_parse($con, $q);

$r = oci_execute($query_id);


?>
</body>
</html>
```

### 4.5 Step 5 (Fetch the result set)

Returns the next row from the result data as an associative or numeric array, or both.

array **oci_fetch_array** ( resource statement [,mode]);

| Parameter | Description |
|---|---|
| Statement | A valid OCI statement identifier. |
| mode | An optional second parameter can be any combination of the following constants:<br>OCI_BOTH - return an array with both associative and numeric indices (the same as OCI_ASSOC + OCI_NUM). This is the default behavior.<br>OCI_RETURN_NULLS - create empty elements for the NULL fields. |

Returns an array with both associative and numeric indices, or FALSE if there are no more rows in the statement.

### Example 4.5

Write the bold line in your file as follows;

```
<html>
<body>
<?php
// example 2.1 ..creating a database connection
$db_sid = "(DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST = Turab-PC)(PORT = 1521)) )
(CONNECT_DATA = (SID = orcl) ) )";

$db_user = "scott";
$db_pass = "1234";

$con = oci_connect($db_user,$db_pass,$db_sid);
if($con)
{
        echo "connection successful.";
}
else
{
         die('Could not connect: ');
}

$q = "select * from emp";
```

```
$query_id = oci_parse($con, $q);

$r = oci_execute($query_id);

$rs_arr = oci_fetch_array($query_id, OCI_BOTH+OCI_RETURN_NULLS);
?>
</body>
</html>
```

### 4.6 Step 6 (Loop over the result set)

While loop is best to display result using fetch array method.

### Example 4.6

Write the bold line in your file as follows;

```
<html>
<body>
<?php
// example 2.1 ..creating a database connection
$db_sid = "(DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(HOST = Turab-PC)(PORT = 1521)) )
(CONNECT_DATA = (SID = orcl) ) )";

$db_user = "scott";
$db_pass = "1234";

$con = oci_connect($db_user,$db_pass,$db_sid);
if($con)
{
        echo "connection successful.";
}
else
{
         die('Could not connect: ');
}

$q = "select * from emp";

$query_id = oci_parse($con, $q);

$r = oci_execute($query_id);

$rs_arr = oci_fetch_array($query_id, OCI_BOTH+OCI_RETURN_NULLS);
//print_r($rs_arr);
//echo"<hr>";

while($row = oci_fetch_array($query_id, OCI_BOTH+OCI_RETURN_NULLS))
{
        echo $row['EMPNO'] ."<br>";
}
?>
</body>
</html>
```

The output would be;

```
7521
7566
7654
7698
7782
7788
7839
```

**TASK 1:** Create PHP connection with Oracle and run '**emp1.php'** and '**emp2.php'** files. Provide screenshots of the output of these files.

**TASK 2:** Create html file (**search_emp.html**) which takes input from user as shown in Fig. 1.
-Job combo box has following input options: CLERK, SALESMAN, MANAGER, ANALYST, and PRESIDENT.
-Department No. has following input options: 10, 20, and 30.

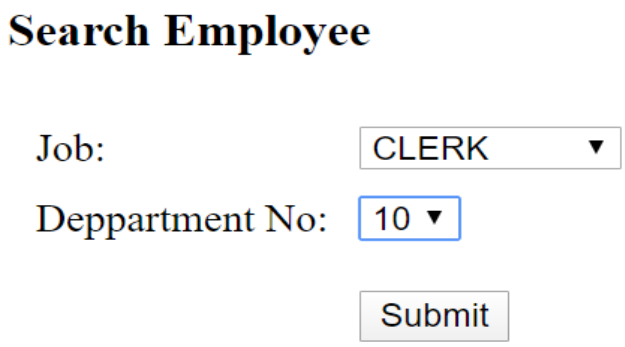


**Figure 1**

**TASK 3:** Create PHP file (**emp_info.php**) which receive inputs from html file (**search_emp.html**) and query the database on basis of received inputs (job and deptno) and output the information of employee/s.

**Submission:**
Each individual student need to submit the task 1 screenshots, 'search_emp.html', and 'emp_info.php' files. Create ZIP to include files. Zip file format should be: Name_RollNo_Sec e.g. Ahmed_18i1234_A.zip

# 5 Resources

## 5.1 websites

http://www.w3schools.com/

## 5.2 Books

Professional PHP Programming by Wrox word press Ltd. **ISBN:** 81-7366-201-0