Developing and Deploying Oracle and PHP

Purpose

This tutorial describes how you can build highly functional PHP applications for your enterprise.

Time to Complete

Approximately 1 hour

Topics

This tutorial covers the following topics:

Viewing Screenshots

Place the cursor over this icon to load and view all the screenshots for this tutorial. (Caution: This action loads all screenshots simultaneously, so response time may be slow depending on your Internet connection.)

Note: Alternatively, you can place the cursor over an individual icon in the following steps to load and view only the screenshot associated with that step. You can hide an individual screenshot by clicking it.

Overview

PHP is a popular Web scripting language, and is often used to create database-driven Web sites. If you want to develop your Web application using PHP and an Oracle database, this tutorial helps you get started by giving examples on using PHP against Oracle. If you are new to PHP, review the Appendix: PHP Primer to gain an understanding of the PHP language.

Back to Topic List

Prerequisites

Before you perform this tutorial, you should:

1. Install Oracle Database 11g or Oracle Database XE.
2. Install PHP 5.2.4.
3. Configure the Linux Apache Server.
4. Download and unzip the php.zip files into the directory where the Apache Server finds the files (i.e. $HOME/public_html).

Installation instructions for steps 1-3 in this prerequisites section can be found on OTN.

Note: <localhost> is the name of the hostname throughout this tutorial. Change this value to your hostname if different. In addition, `HR` is used throughout the tutorial and `HRPWD` is the assumed pasword. If you plan to use XE instead of Oracle Database 11g, you need to change the SID to `localhost/orcl` to `localhost/XE` throughout.

Back to Topic List

1. Creating a Standard Connection

To create a connection to Oracle that can be used for the lifetime of the PHP script, perform the following steps.

1. Review the code as follows that is contained in the **connect.php** file in the **$HOME/public_html** directory.

```
<?php
// Create connection to Oracle
$conn = oci_connect("hr", "hrpwd", "//localhost/orcl");
```

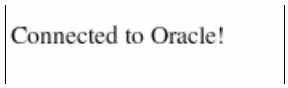
```
if (!$conn) {
   $m = oci_error();
   echo $m['message'], "\n";
   exit;
}
else {
   print "Connected to Oracle!";
}
// Close the Oracle connection
oci_close($conn);
?>
```

The oci_connect() function contains the connection information. In this case, an abbreviated connection string is used.

The oci_close() function is not required as the connection is automatically closed when the script ends.

2. Open a Web browser and enter the following URL to display the output:

   http://localhost/~phplab/connect.php



"Connected to Oracle!" is displayed if the connection succeeds.

The error "Error connecting to Oracle" is displayed if there are problems creating the database connection.

Back to Topic List

2. Creating a Simple Query

A common task when developing Web applications is to query a database and display the results in a Web browser. There are a number of functions you can use to query an Oracle database, but the basics of querying are always the same:

1. **Parse** the statement for execution.
2. **Bind** data values (optional).
3. **Execute** the statement.
4. **Fetch** the results from the database.

To create a simple query, and display the results in an HTML table, perform the following steps.

1. Review the code as follows that is contained in the **query.php** file in the **$HOME/public_html** directory.

```
<?php
// Create connection to Oracle
$conn = oci_connect("hr", "hrpwd", "//localhost/orcl");

$query = 'select * from departments';
$stid = oci_parse($conn, $query);
$r = oci_execute($stid);

// Fetch the results in an associative array
print '<table border="1">';
while ($row = oci_fetch_array($stid, OCI_RETURN_NULLS+OCI_ASSOC)) {
   print '<tr>';
   foreach ($row as $item) {
      print '<td>'.($item?htmlentities($item):' ').'</td>';
   }
   print '</tr>';
}
print '</table>';

// Close the Oracle connection
oci_close($conn);

?>
```

The oci_parse() function parses the statement.

The oci_execute() function executes the parsed statement.

The oci_fetch_array() function retrieves the results of the query as an associative array, and includes nulls.

2. From your Web browser, enter the following URL to display the output:

   http://localhost/~phplab/query.php

10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		1700
140	Control And Credit		1700
150	Shareholder Services		1700
160	Benefits		1700
170	Manufacturing		1700

The results of the query are displayed in a Web browser.

Back to Topic List

3. Creating a Persistent Connection

A persistent connection to Oracle can be reused over multiple scripts. Changes made to the Oracle environment are reflected in all scripts that access the connection. This topic demonstrates this by creating a persistent connection, and then changing the Oracle environment with another script.

To create a persistent connection that can be reused over multiple PHP scripts, perform the following steps:

1. Review the code as follows that is contained in the **pconnect.php** file in the **$HOME/public_html** directory.

```
<?php

// Create a persistent connection to Oracle
// Connection will be reused over multiple scripts
$conn = oci_pconnect("hr", "hrpwd", "//localhost/orcl");
if (!$conn) {
   $m = oci_error();
   echo $m['message'], "\n";
   exit;
}
else {
   print "Connected to Oracle!";
}

// Close the Oracle connection
oci_close($conn);

?>
```

The oci_pconnect() function creates a persistent connection to Oracle.

Using the oci_close() function does not close persistent connections and is redundant in this script.

2. From your Web browser, enter the following URL to display the output:

http://localhost/~phplab/pconnect.php

Connected to Oracle!

A persistent connection has now been created. This connection is still available to scripts that use the same login credentials and that are served by the same http process.

3. Review the code as follows that is contained in the **usersess.sql** file in the **$HOME/public_html** directory.

```
column username format a30
column logon_time format a18
set pagesize 1000 feedback off echo on
alter session set nls_date_format = 'DD-MON-YY HH:MI:SS';
select username, logon_time from v$session where username is not null;
```

You have now created a SQL*Plus (Oracle's command-line SQL scripting tool) script file that you run in SQL*Plus. This SQL*Plus script changes the National Language Character date format of the database, and shows the current database sessions. The date format change only relates to the SQL*Plus session, and is used to format the output of the logon times.

4. Open a terminal window and enter the following commands. Note that you could also exeucte the script in SQL Developer.

```
cd $HOME/public_html
sqlplus system/oracle@//localhost/orcl
@usersess.sql
```

```
SQL> @usersess
SQL> alter session set nls_date_format = 'DD-MON-YY HH:MI:SS';
SQL> select username, logon_time from v$session where username is not null;

USERNAME                       LOGON_TIME
------------------------------ ------------------
HR                             18-OCT-07 11:29:25
SYSMAN                         18-OCT-07 06:52:00
SYSTEM                         18-OCT-07 11:30:38
SYSMAN                         18-OCT-07 06:51:47
SYSMAN                         18-OCT-07 06:51:45
SYSMAN                         18-OCT-07 06:51:44
SYSMAN                         18-OCT-07 06:51:43
SYSMAN                         18-OCT-07 06:51:46
SYSMAN                         18-OCT-07 06:51:28
SYSMAN                         18-OCT-07 06:51:17
DBSNMP                         18-OCT-07 06:51:02
SYSMAN                         18-OCT-07 06:51:42
DBSNMP                         18-OCT-07 06:51:29
DBSNMP                         18-OCT-07 06:50:54
SQL>
```

The SQL*Plus script lists the current database sessions. The session created by the PHP script is still active and shown in the first line of the results as the username HR. Even though the oci_close() function was called, this does not close persistent connections, and the connection is available for other scripts.

6. To show that the persistent connection is being reused by other PHP scripts, and that the session settings are the same, review the code as follows that is contained in the **pconnect2.php** file in the **$HOME/public_html** directory.

```
<?php

// Function to execute a query
function do_query($conn, $query)
{
  $stid = oci_parse($conn, $query);
  oci_execute($stid);
  oci_fetch_all($stid, $res);
  echo "<pre>";
  var_dump($res);
  echo "</pre>";
}

// Create a persistent connection to Oracle
$c = oci_pconnect("hr", "hrpwd", "//localhost/orcl");

// Query the database system date
do_query($c, "select sysdate from dual");

// Change the NLS Territory
$s = oci_parse($c, "alter session set nls_territory=germany");
$r = oci_execute($s);

// Query the database system date again
```

```
do_query($c, "select sysdate from dual");

?>
```

This script creates a new persistent connection, or reuses an existing one with the same login credentials.

The script then uses the do_query() function to query and fetch the database system date. It uses the var_dump debugging function to print the value and structure of the PHP variable containing the date query result.

The script then changes the National Language Territory setting to display the output in the format for Germany, and calls the do_query() function again to display the database system date a second time.

7. From your Web browser, enter the following URL to display the output:

`http://localhost/~phplab/pconnect2.php`

Note that the date format of the two queries differs as the ALTER SESSION command changed to another locale in between queries.

```
array(1) {
  ["SYSDATE"]=>
  array(1) {
    [0]=>
    string(9) "18-OCT-07"
  }
}

array(1) {
  ["SYSDATE"]=>
  array(1) {
    [0]=>
    string(8) "18.10.07"
  }
}
```

To see the effect of using a persistent connection reload the script. You may need to do this a few times until the original PHP session (Apache process) is reused. The date format for both queries is now using the same, new format. This shows that the connection has been reused and the date format set in the inital script is still set when the later script runs. The connection has remained alive (is persistent) for reuse by other PHP scripts that use the same login credentials. If the script was changed to use a standard connection, it would always print two different time formats.

You should be aware of environment changes you make during a persistent session as they may also affect other scripts. But transactions do not span PHP scripts, and uncommitted data will be rolled back at the end of a script.

8. Run the SQL*Plus script **usersess.sql** again to see which connections are open.

```
SQL> @usersess
SQL> set pagesize 1000 feedback off echo on
SQL> alter session set nls_date_format = 'DD-MON-YY HH:MI:SS';
SQL> select username, logon_time from v$session where username is not null;

USERNAME                       LOGON_TIME
------------------------------ ------------------
HR                             19-JAN-07 12:27:23
DBSNMP                         19-JAN-07 12:00:54
HR                             19-JAN-07 12:52:41
HR                             19-JAN-07 12:51:50
DBSNMP                         19-JAN-07 12:00:40
SYSMAN                         19-JAN-07 12:00:35
SYSMAN                         19-JAN-07 12:00:35
SYSMAN                         19-JAN-07 12:00:46
SYSMAN                         19-JAN-07 12:00:41
SYSMAN                         19-JAN-07 12:00:45
SYSMAN                         19-JAN-07 12:00:35
SYSMAN                         19-JAN-07 11:59:12
SYSMAN                         19-JAN-07 11:59:11
SYSMAN                         19-JAN-07 11:59:10
SYSTEM                         19-JAN-07 12:49:37
SQL>
```

There are now a number of database sessions open created by the HR user. This shows the persistent sessions that are currently available. On Linux, Apache runs as multiple independent processes. PHP does not share any information, including connections, between processes. Because each time you run a script it might be executed by a different httpd process, when you use oci_pconnect() you can end up with multiple database connections open.

Back to Topic List

4. Creating Transactions

When you manipulate data in an Oracle database (insert, update, or delete data), the changed or new data is only available within your database session until it is committed to the database. When the changed data is committed to the database, it is then available to other users and sessions. This is a database transaction.

Committing each change individually causes extra load on the server. In general you want all or none of your data committed. Doing your own transaction control has performance and data-integrity benefits.

By default, the oci_execute() function commits changes immediately.

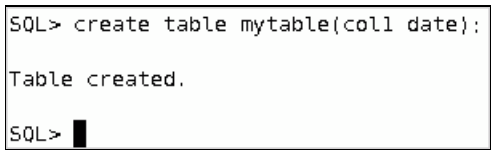
The use of the OCI_DEFAULT parameter in the means that data is not automatically committed, and is not available to other sessions until you explicitly commit it to the database using oci_commit(). You can also rollback with oci_rollback().

Oracle recommends the use of OCI_DEFAULT as a transaction normally consists of multiple database interactions (i.e. DML).

To learn about transaction management in PHP with an Oracle database, perform the following steps.

1. In your SQL*Plus session, enter the following commands to log in to the database as the user HR and create a new table:

```
connect hr/hrpwd@//localhost/orcl
create table mytable (col1 date);
```



2. Review the code as follows that is contained in the **trans1.php** file in the **$HOME/public_html** directory.

```
<?php
    echo "<pre>";

    // Execute a query
    function do_query($conn)
    {
      $stid = oci_parse($conn,
          "select to_char(col1, 'DD-MON-YY HH:MI:SS') from mytable");
      oci_execute($stid, OCI_DEFAULT);
      oci_fetch_all($stid, $res);
      foreach ($res as $v) {
        var_dump($v);
      }
    }

    // Create a database connection
    function do_connect()
    {
      $conn = oci_new_connect("hr", "hrpwd", "//localhost/orcl");
      return($conn);
    }
    $d = date('j:M:y H:i:s');

    // Create a connection
    $c1 = do_connect();

    // Insert the date into mytable
    $s = oci_parse($c1,
        "insert into mytable values (to_date('"
        . $d . "', 'DD:MON:YY HH24:MI:SS'))");

    // Use OCI_DEFAULT to execute the statement without committing
    $r = oci_execute($s, OCI_DEFAULT);

    // Query the current session/connection
    echo "Query using connection 1<br>\n";
    do_query($c1);

    // Create a new connection and query the table contents
    $c2 = do_connect();
    echo "<br>Query using connection 2<br>\n";
    do_query($c2);
    echo "</pre>";
    ?>
```

There are two connections used in this script.

This script uses oci_new_connect() to create a unique, non-persistent database connection, then inserts the date into the mytable table and queries it back.

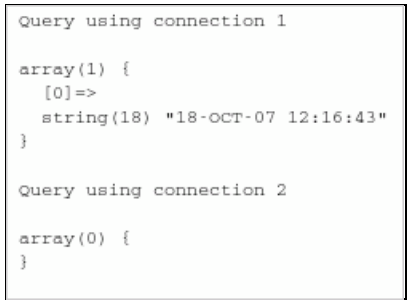
The script then creates a second unique database connection, and queries the table again to show the contents visible to the second connection.

3. From your Web browser, enter the following URL to display the output:

   http://localhost/~phplab/trans1.php

   This script inserts a row into the table using connection $c1.

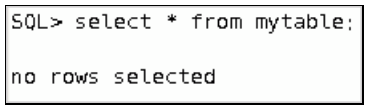
   The data is not been committed to the database because each oci_execute() call uses OCI_DEFAULT and no oci_commit() was called. No other database user can yet see this row. The query using the second connection $c2 returns an empty array.



```
Query using connection 1

array(1) {
  [0]=>
  string(18) "18-OCT-07 12:16:43"
}

Query using connection 2

array(0) {
}
```

4. Because there is no commit, the data is rolled back by PHP when the script finishes. To see that the data has not been committed, query the table to see if there are any inserted rows. From your SQL*Plus session, enter the following commands to select any rows from the mytable table:

   select * from mytable;



```
SQL> select * from mytable;

no rows selected
```

5. Review the code as follows that is contained in the **trans2.php** file in the **$HOME/public_html** directory.

```
<?php
    echo "<pre>";

    // Execute a query
    function do_query($conn)
    {
      $stid = oci_parse($conn,
          "select to_char(col1, 'DD-MON-YY HH:MI:SS') from mytable");
      oci_execute($stid, OCI_DEFAULT);
      oci_fetch_all($stid, $res);
      foreach ($res as $v) {
        var_dump($v);
      }
    }

    // Create a database connection
    function do_connect()
    {
      $conn = oci_new_connect("hr", "hrpwd", "//localhost/orcl");
      return($conn);
    }
    $d = date('j:M:y H:i:s');

    // Create a connection
    $c1 = do_connect();

    // Insert the date into mytable
    $s = oci_parse($c1,
        "insert into mytable values (to_date('"
        . $d . "', 'DD:MON:YY HH24:MI:SS'))");

    $r = oci_execute($s);  // no OCI_DEFAULT means automatically commit

    // Query the current session/connection
    echo "Query using connection 1<br>\n";
    do_query($c1);
```

```
// Create a new connection and query the table contents
$c2 = do_connect();
echo "<br>Query using connection 2<br>\n";
do_query($c2);
echo "</pre>";
?>
```

This script differs from trans1.php in that there is no OCI_DEFAULT when the data is inserted. This means the new data is commited.

6. From your Web browser, enter the following URL to display the output:

http://localhost/~phplab/trans2.php

```
Query using connection 1

array(1) {
  [0]=>
  string(18) "18-OCT-07 12:18:37"
}

Query using connection 2

array(1) {
  [0]=>
  string(18) "18-OCT-07 12:18:37"
}
```

The data is now committed, so both queries return the new row in the table.

Reload the page. Each time you reload you will see more rows added to the table.

```
Query using connection 1

array(2) {
  [0]=>
  string(18) "18-OCT-07 12:18:57"
  [1]=>
  string(18) "18-OCT-07 12:18:37"
}

Query using connection 2

array(2) {
  [0]=>
  string(18) "18-OCT-07 12:18:57"
  [1]=>
  string(18) "18-OCT-07 12:18:37"
}
```

7. From your SQL*Plus session, enter the following commands to delete any rows from the **mytable** table:

```
delete from mytable;
commit;
```

```
SQL> delete from mytable;

2 rows deleted.

SQL> commit;

Commit complete.

SQL>
```

8. You can compare the performance difference between committing each row individually versus at the end of the transaction.

To test the difference, review the code as follows that is contained in the **trans3.php** file in the **$HOME/public_html** directory.

```
<?php
function currTime()
{
```

```
  $time = microtime();
  $time = explode(' ', $time);
  $time = $time[1] + $time[0];
  return $time;
}
function elapsedTime($start)
{
  return (currTime() - $start);
}
function do_query($conn)
{
  $stid = oci_parse($conn,
      "select count(*) c from mytable");
  oci_execute($stid, OCI_DEFAULT);
  oci_fetch_all($stid, $res);
  echo "Number of rows: ", $res['C'][0], "<br>";
}
function do_delete($conn)
{
  $stmt = "delete from mytable";
  $s = oci_parse($conn, $stmt);
  $r = oci_execute($s);
}
function do_insert($conn)
{
  $d = date('j:M:y H:i:s');
  $stmt = "insert into mytable values (to_date('"
          . $d . "', 'DD:MON:YY HH24:MI:SS'))";
  $s = oci_parse($conn, $stmt);
  $r = oci_execute($s);
}
$c = oci_connect("hr", "hrpwd", "//localhost/orcl");
$start = currTime();
for ($i = 0; $i < 10000; $i++) {
  do_insert($c);
}
$et = elapsedTime($start);
echo "Time was ".round($et,3)." seconds<br>";
do_query($c);  // Check insert done
do_delete($c); // cleanup committed rows
?>
```

Run this several times and see how long it takes to insert the 10,000 rows.

```
Time was 35.012 seconds
Number of rows: 10000
```

```
Time was 32.359 seconds
Number of rows: 10000
```

9. Now run the **trans4.php** script. The only difference in this script is that in the do_insert() function OCI_DEFAULT has been added so it doesn't automatically commit, and an explicit commit has been added at the end of the insertion loop:

```
...

function do_insert($conn) {
  $d = date('j:M:y H:i:s');
  $stmt = "insert into mytable values
    (to_date('" . $d . "', 'DD:MON:YY HH24:MI:SS'))";
  $s = oci_parse($conn, $stmt);
  $r = oci_execute($s, OCI_DEFAULT);
}

$c = oci_connect("hr", "hrpwd", "//localhost/orcl");
$start = currTime();
for ($i = 0; $i < 10000; $i++) {
  do_insert($c);
}
oci_commit($c);
$et = elapsedTime($start);

...
```

Rerun the test. The insertion time decreases.

```
Time was 18.285 seconds
Number of rows: 10000
```

In general you want all or none of your data committed. Doing your own transaction control has performance and data-integrity benefits.

Back to Topic List

5. Data Fetching Functions

There are a number of ways to fetch array data from an Oracle database. You can fetch arrays as associative arrays, numeric arrays, or as both.

To learn how to use the array fetching parameters, perform the following steps.

1. The first part shows fetching arrays using the default output of oci_fetch_array(), which is to fetch the array with both associative and numeric indices.

   Review the code as follows that is contained in the **fetch.php** file in the **$HOME/public_html** directory. Review the code as follows:

   ```
   <?php
   echo "<pre>";
   $conn = oci_connect("hr", "hrpwd", "//localhost/orcl");
   $query = 'select * from employees where employee_id = 101';
   $stid = oci_parse($conn, $query);
   oci_execute($stid);
   while ($row = oci_fetch_array($stid)) {
     var_dump($row); // display PHP's representation of $row
   }
   oci_close($conn);
   echo "</pre>";
   ?>
   ```

2. From your Web browser, enter the following URL to display the output:

   http://localhost/~phplab/fetch.php

   ```
   array(11) {
     ["EMPLOYEE_ID"]=>
     string(3) "101"
     ["FIRST_NAME"]=>
     string(5) "Neena"
     ["LAST_NAME"]=>
     string(7) "Kochhar"
     ["EMAIL"]=>
     string(8) "NKOCHHAR"
     ["PHONE_NUMBER"]=>
     string(12) "515.123.4568"
     ["HIRE_DATE"]=>
     string(9) "21-SEP-89"
     ["JOB_ID"]=>
     string(5) "AD_VP"
     ["SALARY"]=>
     string(5) "17000"
     ["COMMISSION_PCT"]=>
     NULL
     ["MANAGER_ID"]=>
     string(3) "100"
     ["DEPARTMENT_ID"]=>
     string(2) "90"
   }
   ```

   The output shows that the results contain both associative and numeric indices. While this may provide more flexibility with how you want to handle the results, it is a bigger network and memory overhead.

3. You may want, instead, to just fetch an array as an associative array. This part shows how you fetch only an associative array.

   Change the *oci_fetch_array()* call to the following:

   ```
   oci_fetch_array($stid, OCI_ASSOC)
   ```

   Rerun the following URL:

   http://localhost/~phplab/fetch.php

```
array(10) {
  ["EMPLOYEE_ID"]=>
  string(3) "101"
  ["FIRST_NAME"]=>
  string(5) "Neena"
  ["LAST_NAME"]=>
  string(7) "Kochhar"
  ["EMAIL"]=>
  string(8) "NKOCHHAR"
  ["PHONE_NUMBER"]=>
  string(12) "515.123.4568"
  ["HIRE_DATE"]=>
  string(9) "21-SEP-89"
  ["JOB_ID"]=>
  string(5) "AD_VP"
  ["SALARY"]=>
  string(5) "17000"
  ["MANAGER_ID"]=>
  string(3) "100"
  ["DEPARTMENT_ID"]=>
  string(2) "90"
}
```

As shown in the output, the OCI_ASSOC parameter fetches the array as an associative array.

4. The final option is to fetch an array as a numeric array. This part shows how you fetch only a numeric array.

   Change the oci_fetch_array() call once again to the following:

   ```
   oci_fetch_array($stid, OCI_NUM)
   ```

   Rerun the following URL:

   http://localhost/~phplab/fetch.php

```
array(10) {
  [0]=>
  string(3) "101"
  [1]=>
  string(5) "Neena"
  [2]=>
  string(7) "Kochhar"
  [3]=>
  string(8) "NKOCHHAR"
  [4]=>
  string(12) "515.123.4568"
  [5]=>
  string(9) "21-SEP-89"
  [6]=>
  string(5) "AD_VP"
  [7]=>
  string(5) "17000"
  [9]=>
  string(3) "100"
  [10]=>
  string(2) "90"
}
```

   The output shows the OCI_NUM parameter fetches the array as a numeric array.

   There are other oci_fetch_array() parameters and combinations you can use, such as:

   - oci_fetch_array($stid, OCI_BOTH), which returns both associative and numeric indices
   - oci_fetch_array($stid, OCI_ASSOC+OCI_RETURN_NULLS), which returns an associative index, and includes NULLs.

   The PHP documentation contains the full list of the fetching options.

Back to Topic List

6. Using Bind Variables

Bind variables enable you to re-execute queries with new values, without the overhead of reparsing the statement. Bind variables improve code reusability, and can reduce the risk of SQL Injection attacks.

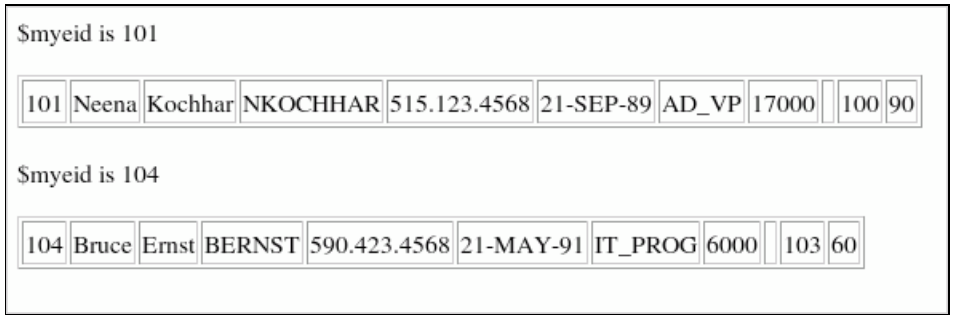
To use bind variables in this example, perform the following steps.

1. Review the code as follows that is contained in the **bind.php** file in the **$HOME//public_html** directory.

```
<?php

   function do_fetch($myeid, $s)
   {
     // Fetch the results in an associative array
     print '<p>$myeid is ' . $myeid . '</p>';
     print '<table border="1">';
     while ($row = oci_fetch_array($s, OCI_RETURN_NULLS+OCI_ASSOC)) {
         print '<tr>';
         foreach ($row as $item) {
             print '<td>'.($item?htmlentities($item):' ').'</td>';
         }
         print '</tr>';
     }
     print '</table>';
   }

   // Create connection to Oracle
   $c = oci_connect("hr", "hrpwd", "//localhost/orcl");

   // Use bind variable to improve resuability, and to
   // remove SQL Injection attacks.
   $query = 'select * from employees where employee_id = :eidbv';
   $s = oci_parse($c, $query);

   $myeid = 101;
   oci_bind_by_name($s, ":EIDBV", $myeid);
   oci_execute($s);
   do_fetch($myeid, $s);

   // Redo query without reparsing SQL statement
   $myeid = 104;
   oci_execute($s);
   do_fetch($myeid, $s);

   // Close the Oracle connection
   oci_close($c);
   ?>
```

2. From your Web browser, enter the following URL to display the output:

   http://localhost/~phplab/bind.php



   The $myeid variable is bound to the :eidbv bind variable, so when the query is re-executed the new value of $myeid is passed to the query. This allows you to execute the statement again, without reparsing it with the new value, and can improve performance of your code.

Back to Topic List

7. Using Stored Procedures

PL/SQL is Oracle's procedural language extension to SQL. PL/SQL stored procedures and functions are stored in the database, so accessing them is incredibly fast. Using PL/SQL stored procedures lets all database applications reuse logic, no matter how the application accesses the database. Many data-related operations can be performed in PL/SQL faster than extracting the data into a program (for example, PHP) and then processing it.

Oracle allows PL/SQL and Java stored procedures. In this tutorial, you will create a PL/SQL stored procedure and call it in a PHP script. Perform the following steps:

1. Start SQL*Plus and create a new table, ptab with the following command:

```
sqlplus hr/hrpwd@//localhost/orcl
create table ptab (mydata varchar(20), myid number);
```

2. In SQL*Plus, create a stored procedure, myproc, to insert data into the ptab table, with the following commands:

```
create or replace procedure
myproc(d_p in varchar2, i_p in number) as
begin
  insert into ptab (mydata, myid) values (d_p, i_p);
end;
/
```

```
SQL> create or replace procedure
  2  myproc(d_p in varchar2, i_p number) as
  3  begin
  4     insert into ptab(mydata, myid) values (d_p, i_p);
  5  end;
  6  /

Procedure created.

SQL>
```

3. Review the code as follows that is contained in the **proc.php** file in the **$HOME/public_html** directory. Review the code as follows:

```
<?php

$c = oci_connect('hr', 'hrpwd', '//localhost/orcl');
$s = oci_parse($c, "call myproc('mydata', 123)");
oci_execute($s);
echo "Completed";

?>
```

4. From a Web browser, enter the following URL to display the output:

http://localhost/~phplab/proc.php

The PHP script has created a new row in the ptab table by calling the stored procedure myproc. The table ptab has a new row with the values "mydata" and 123.

Switch to your SQL*Plus session and query the table to show the new row:

select * from ptab;

```
SQL> select * from ptab;

MYDATA                         MYID
-------------------- ----------
mydata                        123

SQL>
```

5. Extend proc.php to query from the table to check the data has been inserted. Change **proc.php** to the following:

```
<?php

$c = oci_connect('hr', 'hrpwd', '//localhost/orcl');
$s = oci_parse($c, "call myproc('mydata', :bv)");
$v = 123;
oci_bind_by_name($s, ":bv", $v);
oci_execute($s);
echo "Completed";

?>
```

Use *oci_bind_by_name()* to bind a PHP variable $v to ":bv" and experiment changing the value inserted by changing the value in $v.

Rerun the following URL:

http://localhost/~phplab/proc.php

```
Completed
```

Query the table again to show the new row:

select * from ptab;

```
SQL> select * from ptab;

MYDATA                         MYID
-------------------- ----------
mydata                          123
mydata                          123

SQL>
```

6. Apart from stored procedures, PL/SQL stored functions are also commonly used. In SQL*Plus, create a PL/SQL stored function myfunc() to insert a row into the ptab table, and and returns the inserted double the myid value:

```
create or replace function
   myfunc(d_p in varchar2, i_p in number) return number as
   begin
     insert into ptab (mydata, myid) values (d_p, i_p);
     return (i_p * 2);
   end;
   /
```

```
SQL> create or replace function
  2    myfunc(d_p in varchar2, i_p in number) return number as
  3    begin
  4      insert into ptab (mydata, myid) values (d_p, i_p);
  5      return (i_p * 2);
  6    end;
  7    /

Function created.

SQL>
```

7. Review the code as follows that is contained in the **func.php** file in the **$HOME/public_html** directory. Review the code as follows:

```
<?php

   $c = oci_connect('hr', 'hrpwd', '//localhost/orcl');
   $s = oci_parse($c, "begin :bv := myfunc('mydata', 123); end;");
   oci_bind_by_name($s, ":bv", $v, 10);
   oci_execute($s);
   echo $v, "<br>\n";
   echo "Completed";

   ?>
```

Because a value is being returned, the optional length parameter to oci_bind_by_name() is set to 10 so PHP can allocate the correct amount of memory to hold up to 10 digits

Rerun the following URL:

http://localhost/~phplab/func.php

```
246
Completed
```

Back to Topic List

8. Using Collections

A PL/SQL collection is an ordered group of elements of the same type, for example, of the type array.

To work with PL/SQL collections in PHP, perform the following steps:

1. You first will create a simple table and new procedure myproc(). The procedure accepts an array and uses Oracle's fast bulk insert "FORALL" statement to insert all the elements of the array. Review the code in the **proc2.sql** file in the **$HOME/public_html** directory.

```
drop table ptab;

create table ptab(name varchar2(20));

create or replace package mypkg as
  type arrtype is table of varchar2(20) index by pls_integer;
  procedure myproc(p1 in out arrtype);
end mypkg;
/

create or replace package body mypkg as
  procedure myproc(p1 in out arrtype) is
  begin
    forall i in indices of p1
      insert into ptab values (p1(i));
  end myproc;
end mypkg;
/
```

From a terminal window, execute the following commands:

```
sqlplus hr/hrpwd@//localhost/orcl
@proc2
```

```
Table dropped.


Table created.


Package created.


Package body created.

SQL>
```

2. Review the code as follows contained in the **coll.php** file in the **$HOME/public_html** directory.

```
<?php
 function do_query($conn)
 {
    echo "<pre>";
    $stid = oci_parse($conn, "select * from ptab");
    oci_execute($stid, OCI_DEFAULT);
    oci_fetch_all($stid, $res);
    var_dump($res);
    echo "</pre>";
 }
 for ($i = 0; $i < 10; $i++) {
    $a[] = 'value '.$i;
 }
 $c = oci_connect("hr", "hrpwd", "//localhost/orcl");
 $s = oci_parse($c, "BEGIN mypkg.myproc(:c1); END;");
 oci_bind_array_by_name($s, ":c1", $a, count($a), -1, SQLT_CHR);
 oci_execute($s);
 do_query($c)
 ?>
```

This creates an array of strings in $a. The array is then bound to the PL/SQL procedure's parameter.

3. From your Web browser, enter the following URL to display the output:

```
http://localhost/~phplab/coll.php
```

```
array(1) {
  ["NAME"]=>
  array(10) {
    [0]=>
    string(7) "value 0"
    [1]=>
    string(7) "value 1"
    [2]=>
    string(7) "value 2"
    [3]=>
    string(7) "value 3"
    [4]=>
    string(7) "value 4"
    [5]=>
    string(7) "value 5"
    [6]=>
    string(7) "value 6"
    [7]=>
    string(7) "value 7"
    [8]=>
    string(7) "value 8"
    [9]=>
    string(7) "value 9"
  }
}
```

The values are queried back from the PTAB table to verify that they have been inserted.

Back to Topic List

9. Error Handling

The PHP function oci_error() is useful when working with Oracle database error handling.

oci_error() connection errors return FALSE if no error is found, and do not require a parameter to be passed in. If a connection error occurs, oci_error() returns the Oracle error as an associative array. This applies to all connection functions (oci_connect(), oci_pconnect(), and oci_new_connect()).

When working with parsing or execution errors, pass in the resource handle to oci_error().

To practice some simple error handling, perform the following steps.

1. Review the code as follows contained in the **errors.php** file in the **$HOME/public_html** directory.

```
<?php
//Create connection to Oracle
$conn = oci_connect("hr", "hrpwd", "//localhost/orcl");
if (!$conn) {
   // No argument needed for connection errors.
   // To generate an error here, change the connection parameters to be invalid.
   $e = oci_error();
   print "There was a database connection error: " . htmlentities($e['message']);
   exit;
}
// To generate an error here, change the * to an another character, such as %.
$query = "select * from departments";
$stid = oci_parse($conn, $query);
if (!$stid) {
   // For parsing errors, pass the connection resource
   $e = oci_error($conn);
   print "There was a statement parsing error: " . htmlentities($e['message']);
   exit;
}
$r = oci_execute($stid);
if (!$r) {
   // For execution and fetching errors, pass the statement resource
   // To generate an error here, change $query to be an invalid query.
   $e = oci_error($stid);
   echo "<p>";
   print "There was a statement execution error: <strong>" . htmlentities($e['message']).
     "</strong><br>";
   print "The error is located at character " . htmlentities($e['offset']+1) ."
     of the query:
     <strong>". htmlentities($e['sqltext']). "</strong><br>";
   echo "</p>";
exit;
}
// Fetch the results in an associative array
print '<table border="1">';
while ($row = oci_fetch_array($stid, OCI_RETURN_NULLS+OCI_ASSOC)) {
```

```
  print '<tr>';
  foreach ($row as $item) {
  print '<td>'.($item?htmlentities($item):' ').'</td>';
  }
  print '</tr>';
}
print '</table>';
// Close the Oracle connection
oci_close($conn);
?>
```

2. From your Web browser, enter the following URL to display the output:

   http://localhost/~phplab/errors.php

10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		1700
140	Control And Credit		1700
150	Shareholder Services		1700
160	Benefits		1700
170	Manufacturing		1700

3. To generate a connection error, edit errors.php to change the login information to a connection string that will fail, for example an invalid password for the HR user.

   $conn = oci_connect("hr", "hrxx", "//localhost/orcl");

4. Reload the following URL:

   http://localhost/~phplab/errors.php

   Warning: oci_connect() [function.oci-connect]: ORA-01017: invalid username/password; logon denied in **/home/oracle/public_html/errors.php** on line **4**
   There was an error connecting to Oracle: ORA-01017: invalid username/password; logon denied

   The connection error handling code catches the connection error and displays the error in the output.

   Note: The first error is an error generated by PHP, and can be suppressed by turning off error reporting in the php.ini configuration file.

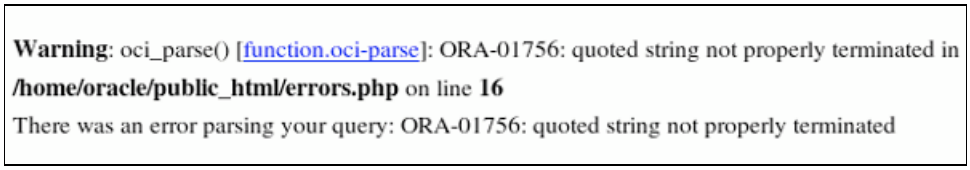
5. Edit **errors.php** to change the login information to the original login so the login and connection will succeed.

6. To generate a parsing error, edit the $query variable to an invalid query structure, for example:

   $query = "select ' from departments";

7. From your browser reload the following URL:

   http://localhost/~phplab/errors.php

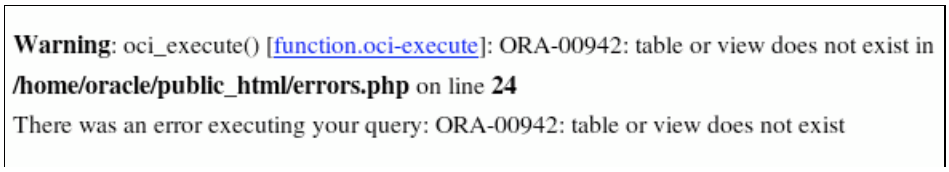
   The parsing error handling code catches the parsing error and displays the error in the output.

8. To generate a fetching error, edit the $query variable to an invalid query, for example:

   ```
   $query = "select * from sometable";
   ```

9. Rerun the following URL:

   http://localhost/~phplab/errors.php

   The fetching error handing code catches the fetching error and displays the error in the output.

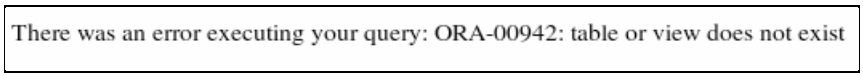
   The offset parameter contains the location of the character at which the parsing error beings, and the sqltext parameter contains the SQL statement that caused the parsing error.

10. The @ function prefix suppresses all PHP errors. This is the same as setting the php.ini file to not display errors, but it is only relevant to the function on which you've used it. Using the @ prefix removes the PHP errors that have been displayed in the previous error-handling examples. To demonstrate this, change oci_execute() to:

    ```
    $r=@oci_execute($stid);
    ```

11. Rerun the following URL:

    http://localhost/~phplab/errors.php

    The PHP errors have been suppressed, but the Oracle errors are still displayed by the error handling code in the script.

Back to Topic List

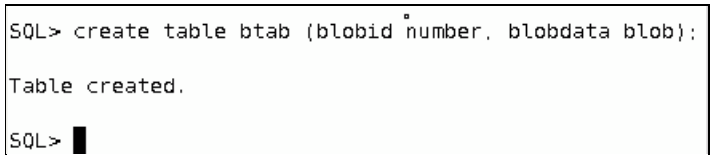
10. Using LOBs: Uploading and Querying Images

Oracle Character Large Object (CLOB) and Binary Large Object (BLOB) columns (and PL/SQL variables) can contain very large amounts of data. There are various ways of creating them to optimize Oracle storage. There is also a pre-supplied package DBMS_LOB that makes manipulating them in PL/SQL easy.

To create a small application to load and display images to the database, perform the following steps.

1. Before doing this section create a table to store a BLOB. In SQL*Plus logged in as HR, execute the following commands:

   create table btab (blobid number, blobdata blob);

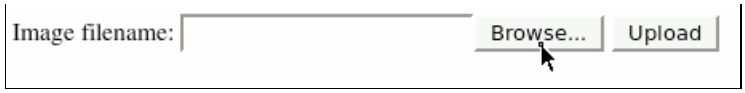
   

2. Review the code as follows contained in the **blobins.php** file in the **$HOME/public_html** directory.

```
<?php
$myblobid = 1; // should really be a unique id e.g. a sequence number
if (!isset($_FILES['lob_upload'])) {
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>"
      method="POST" enctype="multipart/form-data">
Image filename: <input type="file" name="lob_upload">
<input type="submit" value="Upload">
</form>
<?php
}
else {
  $conn = oci_connect("hr", "hrpwd", "//localhost/orcl");
  // Delete any existing BLOB
  $query = 'DELETE FROM BTAB WHERE BLOBID = :MYBLOBID';
  $stmt = oci_parse ($conn, $query);
  oci_bind_by_name($stmt, ':MYBLOBID', $myblobid);
  $e = oci_execute($stmt, OCI_COMMIT_ON_SUCCESS);
  if (!$e) {
    die;
  }
else {
  $conn = oci_connect("hr", "hrpwd", "//localhost/orcl");
   // Delete any existing BLOB
  $query = 'DELETE FROM BTAB WHERE BLOBID = :MYBLOBID';
  $stmt = oci_parse ($conn, $query);
  oci_bind_by_name($stmt, ':MYBLOBID', $myblobid);
  $e = oci_execute($stmt, OCI_COMMIT_ON_SUCCESS);
  if (!$e) {
    die;
  }
  oci_free_statement($stmt);
  // Insert the BLOB from PHP's temporary upload area
  $lob = oci_new_descriptor($conn, OCI_D_LOB);
  $stmt = oci_parse($conn, 'INSERT INTO BTAB (BLOBID, BLOBDATA) '
  .'VALUES(:MYBLOBID, EMPTY_BLOB()) RETURNING BLOBDATA INTO :BLOBDATA');
  oci_bind_by_name($stmt, ':MYBLOBID', $myblobid);
  oci_bind_by_name($stmt, ':BLOBDATA', $lob, -1, OCI_B_BLOB);
  oci_execute($stmt, OCI_DEFAULT);
  if ($lob->savefile($_FILES['lob_upload']['tmp_name'])) {
    oci_commit($conn);
    echo "BLOB uploaded";
  }
  else {
    echo "Couldn't upload BLOB\n";
  }
  $lob->free();
  oci_free_statement($stmt);
}
}
?>
```

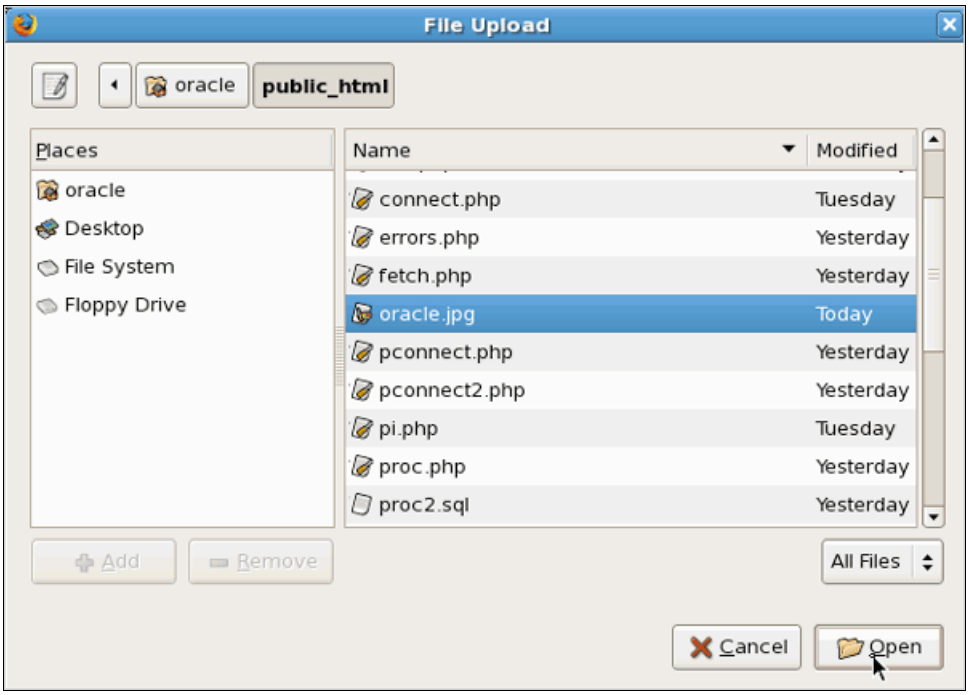
3. From your Web browser, enter the following URL to display the output:

   http://localhost/~phplab/blobins.php



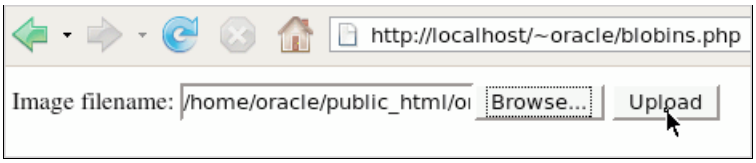
   It shows a Web form with Browse and Upload buttons. Click **Browse.**

4. Select the **oracle.jpg** from the /home/oracle/public_html directory and click **Open**.



4. Click **Upload**.

   The form action calls the script a second time, but now the special variable $_FILES['lob_upload'] is set and the echo statement is executed.



   The image has been uploaded to the Web server.

5. To show the image, review the code as follows contained in the **blobview.php** file in the **$HOME/public_html** directory.

```
<?php
$myblobid = 1;
$conn = oci_connect("hr", "hrpwd", "//localhost/orcl");
// Now query the uploaded BLOB and display it
$query = 'SELECT BLOBDATA FROM BTAB WHERE BLOBID = :MYBLOBID';
$stmt = oci_parse ($conn, $query);
oci_bind_by_name($stmt, ':MYBLOBID', $myblobid);
oci_execute($stmt);
$arr = oci_fetch_assoc($stmt);
$result = $arr['BLOBDATA']->load();
header("Content-type: image/JPEG");
echo $result;
oci_free_statement($stmt);
oci_close($conn);
?>
```

6. From your Web browser, enter the following URL to display the output:

   http://localhost/~phplab/blobview.php

   Make sure there are no echo statements in the script or any whitespace before "<?php", because, otherwise the wrong HTTP header will be sent and the browser won't display the image properly. If you have problems, comment out the header() function call and see what is displayed.

Back to Topic List

11. Using XML

PHP5 has excellent XML capabilities. This tutorial covers the basics of returning XML data from Oracle to PHP.

1. You can fetch relational rows as XML. In this case, you will use the SQL XMLELEMENT function to retrieve the Name and ID of the Employees table where employee_id < 115. Review the code in the **xml1.php** file in the **$HOME/public_html** directory.

2. From your Web browser, enter the following URL to display the output:

   http://localhost/~phplab/xml1.php

```
<tree EMPLOYEE_ID="101" LAST_NAME="Kochhar" DEPARTMENT_ID="90"></tree>
<tree EMPLOYEE_ID="102" LAST_NAME="De Haan" DEPARTMENT_ID="90"></tree>
<tree EMPLOYEE_ID="103" LAST_NAME="Hunold" DEPARTMENT_ID="60"></tree>
<tree EMPLOYEE_ID="104" LAST_NAME="Ernst" DEPARTMENT_ID="60"></tree>
<tree EMPLOYEE_ID="105" LAST_NAME="Austin" DEPARTMENT_ID="60"></tree>
<tree EMPLOYEE_ID="106" LAST_NAME="Pataballa" DEPARTMENT_ID="60"></tree>
<tree EMPLOYEE_ID="107" LAST_NAME="Lorentz" DEPARTMENT_ID="60"></tree>
<tree EMPLOYEE_ID="108" LAST_NAME="Greenberg" DEPARTMENT_ID="100"></tree>
<tree EMPLOYEE_ID="109" LAST_NAME="Faviet" DEPARTMENT_ID="100"></tree>
<tree EMPLOYEE_ID="110" LAST_NAME="Chen" DEPARTMENT_ID="100"></tree>
<tree EMPLOYEE_ID="111" LAST_NAME="Sciarra" DEPARTMENT_ID="100"></tree>
<tree EMPLOYEE_ID="112" LAST_NAME="Urman" DEPARTMENT_ID="100"></tree>
<tree EMPLOYEE_ID="113" LAST_NAME="Popp" DEPARTMENT_ID="100"></tree>
<tree EMPLOYEE_ID="114" LAST_NAME="Raphaely" DEPARTMENT_ID="30"></tree>
```

3. An alternative way of creating XML form relational data is to use the PL/SQL paclage DBMS_XMLGEN(), which returns a CLOB. The code in the file **xml2.php** does the following:

   a) retrieves the first name of employees in department 30 and stores the XML marked-up output in $mylob

```
$q = "select dbms_xmlgen.getxml('

        select first_name
        from employees
        where department_id= 30') xml
        from dual";

        $s = oci_parse($c, $q);
        oci_execute($s);
        $res = oci_fetch_row($s);
        $mylob = $res[0]->load(); // treat as LOB descriptor
```

   b) dislays the content of $mylob

```
        echo htmlentities($mylob);
```

   c) turns the CLOB into an XML Array using PHP's SmpleXML function.

```
        $xml = (array) simplexml_load_string($mylob);
```

4. From your Web browser, enter the following URL to display the output:

   http://localhost/~phplab/xml2.php

XML Output from Oracle

```
<?xml version="1.0"?>
<ROWSET>
 <ROW>
  <DEPARTMENT_NAME>Marketing</DEPARTMENT_NAME>
  <EMPS>
   <EMPS_ROW>
    <FIRST_NAME>Michael</FIRST_NAME>
   </EMPS_ROW>
   <EMPS_ROW>
    <FIRST_NAME>Pat</FIRST_NAME>
   </EMPS_ROW>
  </EMPS>
 </ROW>
 <ROW>
  <DEPARTMENT_NAME>Accounting</DEPARTMENT_NAME>
  <EMPS>
   <EMPS_ROW>
    <FIRST_NAME>Shelley</FIRST_NAME>
   </EMPS_ROW>
   <EMPS_ROW>
    <FIRST_NAME>William</FIRST_NAME>
   </EMPS_ROW>
  </EMPS>
 </ROW>
</ROWSET>
```

Convert to XML Array with PHP's SimpleXML

```
array(1) {
  ["ROW"]=>
  array(2) {
    [0]=>
    object(SimpleXMLElement)#3 (2) {
      ["DEPARTMENT_NAME"]=>
      string(9) "Marketing"
      ["EMPS"]=>
      object(SimpleXMLElement)#2 (1) {
        ["EMPS_ROW"]=>
        array(2) {
          [0]=>
          object(SimpleXMLElement)#5 (1) {
            ["FIRST_NAME"]=>
            string(7) "Michael"
          }
          [1]=>
          object(SimpleXMLElement)#6 (1) {
            ["FIRST_NAME"]=>
            string(3) "Pat"
          }
        }
      }
    }
    [1]=>
    object(SimpleXMLElement)#4 (2) {
      ["DEPARTMENT_NAME"]=>
      string(10) "Accounting"
      ["EMPS"]=>
      object(SimpleXMLElement)#7 (1) {
        ["EMPS_ROW"]=>
        array(2) {
          [0]=>
          object(SimpleXMLElement)#8 (1) {
            ["FIRST_NAME"]=>
            string(7) "Shelley"
```

Back to Topic List

Summary

In this tutorial, you learned how to:

- Create a Connection
- Create a Simple Query
- Create a Persistent Connection

- Create transactions
- Fetch data functions
- Tune data Prefetching
- Use bind variables
- Use PL/SQL
- Use Collections
- Implement error handling
- Upload and query images
- Use XML

Back to Topic List

Appendix: PHP Primer

PHP is a dynamically typed scripting language. It is most often seen in Web applications but can be used to run command-line scripts. Basic PHP syntax is simple to learn. It has familiar loops, tests, and assignment constructs. Lines are terminated with a semi-colon.

Strings can be enclosed in single or double quotes:

```
'A string constant'
"another constant"
```

Variable names are prefixed with a dollar sign. Things that look like variables inside a double-quoted string will be expanded:

```
"A value appears here: $v1"
```

Strings and variables can also be concatenated using a period.

```
'Employee ' . $ename . ' is in department ' . $dept
```

Variables do not need types declared:

```
$count = 1;
$ename = 'Arnie';
```

Arrays can have numeric or associative indexes:

```
$a1[1] = 3.1415;
$a2['PI'] = 3.1415;
```

Strings and variables can be displayed with an echo or print statement. Formatted output with printf() is also possible.

```
echo 'Hello, World!';
echo $v, $x;
print 'Hello, World!';
printf("There is %d %s", $v1, $v2);
```

The var_dump() function is useful for debugging.

```
var_dump($a2);
```

Given the value of $a2 assigned above, this would output:

```
array(1) {
  ["PI"]=>
  float(3.1415)
}
```

Code flow can be controlled with tests and loops. PHP also has a switch statement. The if/elseif/else statements look like:

```
if ($sal > 900000) {
  echo 'Salary is way too big';
}
elseif ($sal > 500000) {
  echo 'Salary is huge';
}
  else {
  echo 'Salary might be OK';
}
```

This also shows how blocks of code are enclosed in braces.

A traditional loop is:

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

This prints the numbers 0 to 9. The value of $i is incremented in each iteration. The loop stops when the test condition evaluates to false. You can also loop with while or do while constructs.

The foreach command is useful to iterate over arrays:

```
$a3 = array('Aa', 'Bb', 'Cc');
```

```
foreach ($a3 as $v) {
  echo $v;
}
```

This sets $v to each element of the array in turn.

A function may be defined:

```
function myfunc($p1, $p2) {
  echo $p1, $p2;
  return $p1 + $p2;
}
```

Functions may have variable numbers of arguments, and may or may not return values. This function could be called using:

```
$v3 = myfunc(1, 3);
```

Function calls may appear earlier than the function definition.

Sub-files can be included in PHP scripts with an include() or require() statement.

```
include("foo.php");
require("bar.php");
```

A require() will generate a fatal error if the script is not found.

Comments are either single line:

```
// a short comment
```

or multi-line:

```
/*
  A longer comment
*/
```

PHP scripts are enclosed in <*?php* and *?*> tags.

```
<?php
  echo 'Hello, World!';
?>
```

When a Web server is configured to run PHP files through the PHP interpreter, loading the script in a browser will cause the PHP code to be executed and all output to be streamed to the browser.

Blocks of PHP code and HTML code may be interleaved. The PHP code can also explicitly print HTML tags.

```
<?php
  require('foo.php');
  echo '<h3>';
  echo 'Full Results';
  echo '</h3>';
  $output = bar(123);
?>
<table border="1">
  <tr>
    <td>
     <?php echo $output ?>
    </td>
  </tr>
</table>
```

Many aspects of PHP are controlled by settings in the php.ini configuration file. The location of the file is system specific. Its location, the list of extensions loaded, and the value of all the initialization settings can be found using the phpinfo() function:

```
<?php
  phpinfo();
?>
```

Values can be changed by editing phpl.ini or using the Zend Core for Oracle console, and restarting the Web server. Some values can also be changed within scripts by using the ini_set() function.

A list of the various oci_xxx functions include the following:

oci_fetch_all	Fetches all rows of result data into an array
oci_fetch_array	Returns the next row from the result data as an associative or numeric array, or both
oci_fetch_assoc	Returns the next row from the result data as an associative array
oci_fetch_object	Returns the next row from the result data as an object
oci_fetch_row	Returns the next row from the result data as a numeric array

Back to Topic List

Move your mouse over this icon to hide all screenshots.

E-mail this page
Printer View

ORACLE IS THE INFORMATION COMPANY

About Oracle | RSS | Careers | Contact Us | Site Maps | Legal Notices | Terms of Use | Privacy