

Chapter 14.

SECTION III: WORKING WITH PHP

Integration Between PHP And Oracle

Programmer's pickup a Web server side scripting language like PHP largely to be able to interact with databases. PHP and Oracle is really the best combination for robust data-driven Web sites. PHP understands and supports ANSI SQL, it also compiles on a number of platforms, has multithreading abilities on Unix servers, all of which make for great performance. On Windows NT/XP, Oracle can be run as a service and as a normal process in Windows 95/98 machines.

Till fairly recently MySQL and Postgres, was the only real option for database software run on Linux. Today however, there are a number of commercial database products officially supported on Linux, including the industry heavyweights **Oracle** and **DB2**. These products, combined with Apache Web server and PHP, make Linux a very attractive platform for developing and running Web-enabled database applications of all sizes.

The PHP-Oracle combination is truly cross-platform and free. This means an application can be developed on Windows and then copied to a Unix platform where it will run cheerfully. PHP can be run as an external CGI process, a standalone script interpreter or an embedded Apache module, real flexibility.

The ability to efficiently store and retrieve large amounts of information on demand has contributed enormously to the success of the Internet. This is almost always implemented through the use of a database. Sites such as Yahoo, IndiaTimes and Rediff depend heavily on the reliability of their databases for storing and retrieving enormous amounts of information accurately.

A database is very useful when used with a website. There are a huge variety of things that can be done with this combination, from displaying simple lists to running a complete website from a database.

When a database is properly implemented, it can be used for storing, retrieving and manipulating data. Adding site search and information sorting features really simplify when a database is used. Control over viewing permissions becomes a non-issue because of the access control features in many database systems. Data replication and backup are also simple.

Connecting To AN Oracle Database

Before any interactive HTML form based work can be done, PHP must be able to connect to an Oracle database. If PHP is not connected to the Oracle database, then all commands forwarded to the database via PHP, will fail.

REMINDER



A good practice for using databases is to specify the username, password and connection string name right at the top of the code spec so that if a change is required at a later date, only one line which is at the top of the code spec needs changing.

For example:

```
$username="username";
$password="password";
$connstring="connection_string";
```

HINT



Keeping the database password in a PHP file is not a security risk at all. The Web server processes PHP code spec before anything is sent to a Browser, hence it is impossible for a user to see the PHP script's source.

Getting Started

There are several ways for PHP to talk to an Oracle database, including using ODBC libraries, as well as two types of Oracle library support built into PHP. This chapter concentrates on using the Oracle native libraries specifically the Oracle 8 function calls. The older Oracle libraries still exist in PHP, but are largely deprecated by the OCI8 functions.

Here is the general order of events that take place during the Oracle server communications process:

1. Establish a connection with the Oracle server, by specifying the login information. If the connection attempt fails, display an appropriate message and exit the process
2. Perform necessary queries on the selected database(s)
3. Once querying is complete, close the database server connection

Connecting To the Database

There are several basic steps to running any Oracle SQL command under PHP, which include connecting to the database, parsing the SQL command, executing the command and returning any results. Connecting to the database is accomplished using the **ocilogon** command. The command takes three arguments: database user name, password and the name of the database (Usually specified in form of a connection string).

Issue the following commands to the PHP interpreter to create a connection with the Oracle database:

```
/* Variables used to store Database access information. */
$dbuser = "dba_pm";
$dbpass = "sct2306";
$connStr = "OraLin";    (Only if the Database is on some other
                        machine and communication is via a client)
```

If Oracle Database is on same machine:

```
$rcq = ocilogon($dbuser, $dbpass);
```

If Oracle Database is on some other machine and is communicated with via a client:

```
$rcq = ocilogon($dbuser, $dbpass, $connStr);
```

The above lines tell PHP to connect to the Oracle database engine running in memory (RAM) with a specific username and password, which have been passed as parameters.

The function **ocilogon()** accepts the oracle login information and creates the actual connection to the database. This connection can be made available whenever required by referring to **\$rcq**, which is the connection object created and returned by the **ocilogon()** function. The database name parameter is optional and if it is not specified, the value of **\$ORACLE_SID** (registered in the environment earlier) is assumed.

Executing Commands

Once connected to a database, the next thing is to run some SQL commands, which is accomplished with a combination of functions namely **OCIParse** and **OCIExecute** to generate and return the results in a usable form.

Syntax:

```
int OCIParse(int <Connection>, string <Query>)
```

OCIParse takes the database connection generated earlier and an SQL string as arguments and returns the statement ID of the parsed statement.

Syntax:

```
int OCIExecute(int <StatementID>, int <Mode>)
```

OCIExecute takes the results of **OCIParse** and an optional mode, which tells the database whether to commit the results of the command (The default being **COMMIT**) and returns **TRUE** on success, **FALSE** on failure.

14. INTEGRATION BETWEEN PHP AND ORACLE

Other than the above functions, there are two more functions used often.

Syntax:

```
int OCIFetchStatement(int <StatementID>, array <Variable>)
```

OCIFetchStatement takes the statement ID of an executed statement and an array to populate with the results of the query and returns the number of rows in the result set.

Syntax:

```
int OCIRowCount(int <StatementID>)
```

OCIRowCount takes a statement ID and returns the number of rows altered by an update.

Inserting Data

Add a block of information to the database table **category**: (Assumed that the **category** table already exists. Refer **Chapter 06: Basic Interaction With SQL**)

```
/* Building and executing a SQL query to INSERT the new category
into the category table. */
$query = "INSERT INTO category (category_ID, category_name, category_remarks)
VALUES(1, ".$txtCatName.", ".$txtCatRmrk.")";
/* Parsing the SQL Query. */
$parsed = ociparse($rcq, $query);
/* Executing the SQL Query. */
ociexecute($parsed);
/* Generating and storing a message that indicates the insert
operation was successful. */
$message = '<FONT Color="blue">Entry for <B>'.$txtCatName.'</B> added
successfully.</FONT>';
echo $message;
```

The variable **\$query** is used because the ANSI SQL's **INSERT INTO** statement is being assigned to this variable.

The **INSERT INTO** statement tells the PHP interpreter to append into the **category** table (found within the database) with the values enclosed within parenthesis, **i.e.** the parenthesis contain all the information to be added to the table **category**.

Example: (ctgryInsert.php)

```
<?php
/* Variables used to store Database access information. */
$dbuser = "dba_pm";
$dbpass = "sct2306";
```

```

/* Use only if the Database is on some other machine and
communication is via a client. OraLin will be replaced with the
Service Name created using Net Configuration Assistant. */
$connStr = "OraLin";

/* Connect as, if Oracle Database on same machine */
$rcq = ocilogon($dbuser, $dbpass);

/* Building and executing a SQL query to INSERT the new category
into the category table. */
$query = "INSERT INTO category (category_ID, category_name,
category_remarks) VALUES(6, 'Analyst Designer', 'With 2 Years
Experience)";

/* Parsing the SQL Query. */
$parsed = ociparse($rcq, $query);
/* Executing the SQL Query. */
ociexecute($parsed);

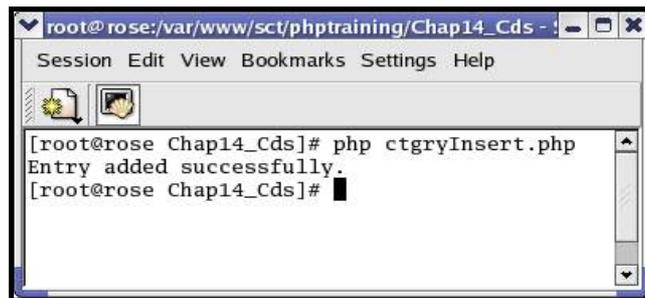
/* Generating and storing a message that indicates the insert
operation was successful. */
$message = 'Entry added successfully.';
/* Displaying the message. */
echo "$message\n";
?>

```

Output: (Refer diagram 14.1)

HTML Input

Inputting data using HTML pages is almost identical to inserting it using a PHP script. The benefit, though, is that the script need not be changed for each block of data that has to be placed into the Category table. Users can also be allowed to input their own data using this technique.



```

root@rose:/var/www/sct/phptraining/Chap14_Cds - :
Session Edit View Bookmarks Settings Help
[root@rose Chap14_Cds]# php ctgryInsert.php
Entry added successfully.
[root@rose Chap14_Cds]#

```

Diagram 14.1: Output for ctgryInsert.php.

The following code spec (saved as **demo.html**) will display an HTML page with textboxes, via which appropriate details that can be entered into the table **category**:

```

<HTML><BODY>
  <FORM Action="insert.php" Method="post">
    Category ID: <INPUT Type="text" Name="txtCategory_ID"><BR>
    Category Name: <INPUT Type="text" Name="txtCategory_Name"><BR>
    Category Remarks: <INPUT Type="text" Name="txtCategory_Remarks"><BR>
    <INPUT Type="Submit" Value="Submit">
  </FORM>
</BODY></HTML>

```

14. INTEGRATION BETWEEN PHP AND ORACLE

Edit the PHP script written earlier, i.e. the **ctgryInsert.php** file. Replace the code block with hard coded information, with the new one where data is input into the database table, using variables:

```

<?php
/* Variables used to store Database access information. */
$dbuser = "dba_pm";
$dbpass = "sct2306";
/* Use only if the Database is on some other machine and communication
is via a client. OraLin will be replaced with the Service Name created
using Net Configuration Assistant. */
$connStr = "OraLin";

/* Connect as, if Oracle Database on same machine */
$rcq = ocilogon($dbuser, $dbpass);

/* Building and executing a SQL query to INSERT the new category into
the category table. */
$query = "INSERT INTO category (category_ID, category_name,
category_remarks) VALUES('".$_POST['txtCategory_ID']."',
".$_POST['txtCategory_Name']."',
".$_POST['txtCategory_Remarks']."'");
/* Parsing the SQL Query. */
$parsed = ociparse($rcq, $query);
/* Executing the SQL Query. */
ociexecute($parsed);

/* Generating and storing a message that indicates the insert operation
was successful. */
$message = '<FONT Color="blue">Entry for
<B>'.$_POST['txtCategory_Name'].'</B> added
successfully.</FONT>';
/* Displaying the message. */
echo $message;
?>

```

This script should then be saved as **insert.php** so that an HTML form can invoke it on demand.

| | |
|---|---|
| <p>Category ID: <input type="text" value="6"/></p> <p>Category Name: <input type="text" value="Data Entry Operator"/></p> <p>Category Remarks: <input type="text" value="Having Typing Skills"/></p> <p><input type="button" value="Submit"/></p> | <p>Entry for Data Entry Operator added successfully.</p> |
|---|---|

Diagram 14.2.1: Entering values into the page generated by demo.html.

Diagram 14.2.2: Message displayed on the page generated by insert.php.

It works because, instead of the data being hard coded, it is being entered via the HTML form and stored in variables, which are then passed to the PHP interpreter for further processing.

Extracting Data

Now that there is at least one record, (if not many more), in the database table Category this data needs to be extracted using PHP.

The first command will be a query as shown below:

```
SELECT * FROM category;
```

This is a basic command, which tells the PHP script to select all the records from the Category table.

```
$query="SELECT * FROM category";  
$parsed = ociparse($rcq, $query);  
ociexecute($parsed);
```

Setting Up The Loop

Set up a loop to take each row of the **\$parsed** array and print out the data held there to the VDU.

```
<?  
  do {  
    echo($line['CATEGORY_ID'] . " || ". $line['CATEGORY_NAME'] . " || " .  
        $line['CATEGORY_REMARKS'] . "<BR>");  
  } while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM))  
?>
```

This is a basic PHP loop, which executes the code for the number of records retrieved.

Combining The Script

A full PHP script (saved as **select.php**), which outputs data:

```
<?php  
/* Variables used to store Database access information. */  
$dbuser = "dba_pm";  
$dbpass = "sct2306";  
/* Use only if the Database is on some other machine and  
communication is via a client. OraLin will be replaced with the  
Service Name created using Net Configuration Assistant. */  
$connStr = "OraLin";
```

14. INTEGRATION BETWEEN PHP AND ORACLE

```

/* Connect as, if Oracle Database on same machine */
$rcq = ocilogon($dbuser, $dbpass);
// Declare query
$query="SELECT * FROM Category";
$parsed = ociparse($rcq, $query);
ociexecute($parsed);

// Generating Output
echo "<B><CENTER>Database Output</CENTER></B><BR><BR>";
// Setting up a loop
while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM)) {
    echo($line['CATEGORY_ID'] . " || ". $line['CATEGORY_NAME'] . " || " .
        $line['CATEGORY_REMARKS'] . "<BR>");
}
?>

```

In this script the data is not formatted when it is rendered. Refer diagram 14.3

Designing A Login Module With Database Support

```

Database Output
1 || Continental Cook || Specilised in continental preparations
2 || Arabic Cook || Specilised in Irani preparations
3 || Analyst Programmer || Specilised in Irani preparation
4 || Scaffolding Foreman || Well experienced personnel in construction and development
5 || Cabin Steward ||
6 || Analyst Designer || With 2 Years Experience
7 || Data Entry Operator || Having Typing Skills

```

Diagram 14.3: Output for select.php.

The functionality of the Login module built in the earlier chapter, ([Chapter 13: Filehandling In PHP](#) section on *Designing A Login Module With Flat File Support*) is a full-fledged login authentication system for visitors to a Website. The module is capable of comparing a login name and password, submitted by a visitor with a valid set of login information stored in a flat file on the Web server. Only when a match for both the login name and its corresponding password is found, a visitor is allowed access to other modules in the Website.

Although better than using hard coded values in the script itself, the technique of using flat files has security drawbacks. Contents of a normal file are easily accessible via an ASCII editor. Even though a file can be locally protected by a user login system, (as found in UNIX, Linux, Windows Servers and so on) yet the risk of external access to confidential file data is very high.

The introduction of databases reduced security risk considerably. Higher end RDBMS like Oracle, MS SQL, MySQL and so on provide an additional layer of security to confidential data. The example, which follows is a Login module, which refers to data stored in table controlled by the Oracle database server.

Since Oracle is the database in use, access to a table is permitted only after logging in to Oracle. A user of the Oracle database system is given part or whole access to resources under Oracle depending on the login used to access the system. Every table under Oracle is bound to a Tablespace and a user login. Hence, every user login is in turn automatically bound to a Tablespace.

Database Structure For The Login Module

Since the example to follow demonstrates the technique of accessing data stored in Oracle table. It is necessary to create the underlying tablespace, users bound to the tablespace and data tables within the tablespace.

REMINDER



All objects and resources under Oracle are created, accessed and manipulated via SQL statements. These SQL statements get processed and executed by the Oracle Database Engine, which listens to commands passed by other tools. SQL *PLUS is the simplest interactive SQL tool used for passing SQL statements to the Oracle Database Engine.

The first step is to create a Tablespace. The following SQL statement, when passed via SQL *PLUS, will create a Tablespace named **PM_SYS**:

```
CREATE TABLESPACE PM_SYS
  DATAFILE 'PM_SYS.dat' SIZE 50M
  DEFAULT STORAGE(INITIAL 10K Next 50K MINEXTENTS 1 MAXEXTENTS 999
                  PCTINCREASE 10)
  ONLINE;
```

The second step is to create a user login. The following SQL statement will create a user login under Oracle named **DBA_PM**, (The user is bound to the **PM_SYS** tablespace):

```
CREATE USER "DBA_PM"
  PROFILE "DEFAULT"
  IDENTIFIED BY "sct2306"
  DEFAULT TABLESPACE "PM_SYS"
  TEMPORARY TABLESPACE "TEMP"
  ACCOUNT UNLOCK;
```

The creation of the user login is followed by setting permissions. The following SQL statement converts the user **DBA_PM** into an Oracle superuser or Database Administrator:

```
GRANT "DBA" TO "DBA_PM" WITH ADMIN OPTION;
```

The final step is to create a table which will store valid login information used/referred by the website.

14. INTEGRATION BETWEEN PHP AND ORACLE

The structure of the Oracle table will be as follows:

Table Definition:

Table Name : **USERS**
Primary Key : **USER_ID**
Foreign Key : - -

Column Definition:

| Column Name | Data Type | Width | Allow Null | Default |
|-------------------|-----------|-------|------------|-------------------|
| USER_ID | VARCHAR2 | 7 | NOT NULL | |
| USER_USERNAME | VARCHAR2 | 15 | NOT NULL | |
| USER_PASSWORD | VARCHAR2 | 20 | NOT NULL | |
| USER_TYPE | VARCHAR2 | 5 | NOT NULL | adm / cand / clnt |
| FOR_PASS_QUESTION | VARCHAR2 | 100 | | NULL |
| FOR_PASS_ANSWER | VARCHAR2 | 50 | | NULL |

Table Description:

| Column Name | Description |
|-------------------|---|
| USER_ID | Auto Generated unique user number beginning with US , followed by 5 digits |
| USER_USERNAME | Username for the user to login |
| USER_PASSWORD | Password for the user |
| USER_TYPE | User types: candidate (cand), client (clnt), administrator (adm) |
| FOR_PASS_QUESTION | The question for Forgot Password Option |
| FOR_PASS_ANSWER | The answer for forgot password option |

Explanation:

The USERS table stores authentication information of system users (clients and candidates).

The SQL statement used to create the table with the above description is as follows:

```
CREATE TABLE USERS (  

  USER_ID VarChar2(7) NOT NULL,  

  USER_USERNAME VarChar2(15) NOT NULL,  

  USER_PASSWORD VarChar2(20) NOT NULL,  

  USER_TYPE VarChar2(5) CHECK (USER_TYPE IN('cand', 'clnt', 'adm')),  

  FOR_PASS_QUESTION VarChar2(100) DEFAULT NULL,  

  FOR_PASS_ANSWER VarChar2(50) DEFAULT NULL,  

  PRIMARY KEY (USER_ID)  

);
```

The creation of the table is followed by populating it with sample data. The following SQL statements will insert four valid login for the website:

```
INSERT INTO USERS VALUES ('US00000', 'admin', 'sct2306', 'adm',  

  'admin', 'password');  

INSERT INTO USERS VALUES ('US00001', 'sharanam', 'sct2306', 'cand',  

  'Whats up', 'nothing');
```

```

INSERT INTO USERS VALUES ('US00002', 'hansel', 'sct2306', 'cand',
'Whats my moms name?', 'mom');
INSERT INTO USERS VALUES ('US00003', 'ivan', 'sct2306', 'cand',
'what is your wife name?', 'cynthia');

```

Converting The Login Module To Support A Database Management System

Having created the tables for the Login System, changes in PHP code can commence. The example used in **Chapter 13: Filehandling In PHP**, is used as a base, hence do the following:

- ❑ Create a directory named **Chap14_Cds** under **sct.phptraining.com** domain
- ❑ Create a copy of the files **login.html**, **loginscript.js** and **logincheck.php** into the **Chap14_Cds** directory
- ❑ Since the **logincheck.php** file performs the authentication of the visitor's login information. Open the file in an ASCII editor and change its contents as show below:

```

<?
/*
Date :- 29/05/05
Author :- Hansel Colaco.
Filename :- logincheck.php
Purpose :- Reads database to authenticate the login information
provided by the visitor.
*/
/* A variable holding the developers name. */
$author = 'Hansel Colaco';

/* Storing the system date in the format 1st Jan, 2000. */
$sysdt = date("jS M, Y");
/* Storing the system time in the format 00:00. */
$system = date("H:i");

/* Variables used to store Database access information. */
$dbuser = "dba_pm";
$dbpass = "sct2306";

/* Checking if the variable either username or password is empty.
This means that the visitor has managed to submit form
information with empty fields. */
if(empty($txtUserName) || empty($txtPassword)) {
    $Message = '<FONT Color="red"><B>Login Information Not
                Submitted!</B></FONT>';
}

```

14. INTEGRATION BETWEEN PHP AND ORACLE

```

else {
/* The function ociLogon() accepts the oracle login information
to create the actual connection to the database. This
connection made available whenever required by referencing
$rcq, which the connection object created by the ociLogon()
function. */
    $rcq = ociLogon($dbuser, $dbpass);
/* Checking whether the creation of the connection object
failed or not. If failed, an error message is generated. */
    if(!$rcq){
        $Message = '<FONT Color="red"><B>Database connection error!</B>
<BR>Please contact the Oracle Database
Administrator.</FONT><BR>';
    }
/* If the creation of the connection object succeeded. */
    else {
/* Building a SQL query to retrieve values held in USER_ID
and USER_TYPE column from the USERS table. The data
retrieved is filtered by finding the record which satisfies
the following conditions completely: a) the USER_USERNAME
column matches the value captured by the User Name field;
and b) the USER_PASSWORD column matches the value captured
by the password field. */
        $query = "SELECT user_ID, user_type FROM users WHERE
                user_username='$txtUserName' AND
                user_password='$txtPassword'";
/* Parsing the SQL query. */
        $parsed = ociParse($rcq, $query);
/* Executing the SQL query. */
        ociExecute($parsed);
/* Checking if any record was retrieved by the above SQL
query. If a valid user is found, a message is displayed,
which indicates a Valid login attempt. */
        if(ociFetchinto($parsed, $rslt_arr, OCI_NUM)) {
            $Message = '<FONT Color="red"><B>Welcomes To The
Website!</B><BR>A Valid Login information has been
submitted.</FONT><BR>';
        }
/* If a valid user is not found, an error message indicating
the Incorrect login attempt is displayed with a link to the
login page. */
        else {
            $Message = '<FONT Color="red"><B>Incorrect
Login!</B><BR>Please <A
HRef="/Chap14_Cds/login.html">re-log</A> with
valid information.</FONT><BR>';
        }
    }
}
?>

```

```

<!-- Generating a page to read contents of a database using a
PHP's Oracle function. -->
<HTML><HEAD>
  <TITLE>Login Authentication Through A Database Table</TITLE>
</HEAD><BODY><?=$Message;?><BR><HR>
<!-- Presenting the login inform in a tabular layout. -->
  <H4>Details of the login attempt are:</H4>
  <TABLE Border="1" CellSpacing="1" Width="200"><TR>
    <TD Align="right" Width="75">Username :</TD>
    <TD Align="left" Width="100"><?="$txtUserName";?></TD>
  </TR><TR>
    <TD Align="right" Width="75">Password :</TD>
    <TD Align="left" Width="100"><?="$txtPassword";?></TD>
  </TR><TR>
    <TD Align="right" Width="75">Server Date :</TD>
    <TD Align="left" Width="100"><?="$sysdt";?></TD>
  </TR><TR>
    <TD Align="right" Width="75">Server Time :</TD>
    <TD Align="left" Width="100"><?="$system";?></TD>
  </TR></TABLE><HR>
</BODY></HTML>

```

Save the above changes made to **logincheck.php** and run **login.html** via a Web Browser. When the login form appears enter the login information and click **Login** to submit data. The resulting page will appear as one of the following possibilities:

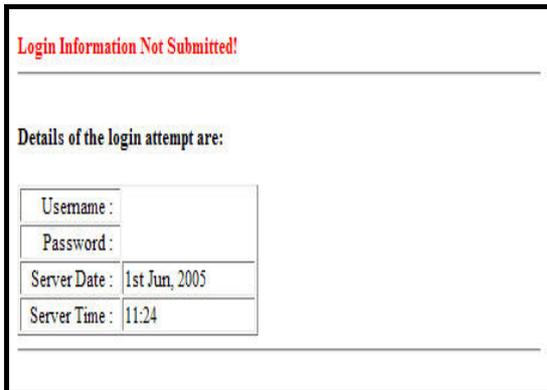


Diagram 14.4.1: Message by the logincheck.php, displayed when the login information is not submitted.

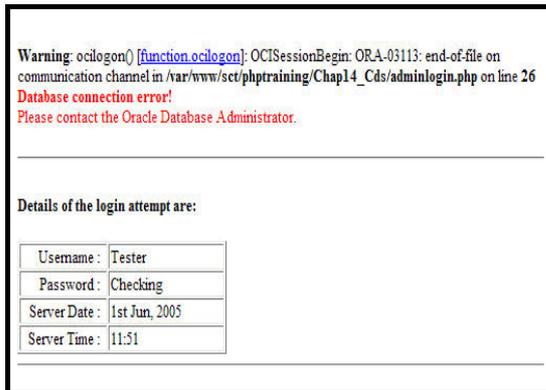


Diagram 14.4.2: Message by the logincheck.php, displayed when the connection object to the Oracle database cannot be established.

14. INTEGRATION BETWEEN PHP AND ORACLE



Diagram 14.4.3: Message by the logincheck.php, displayed for an invalid login attempt.

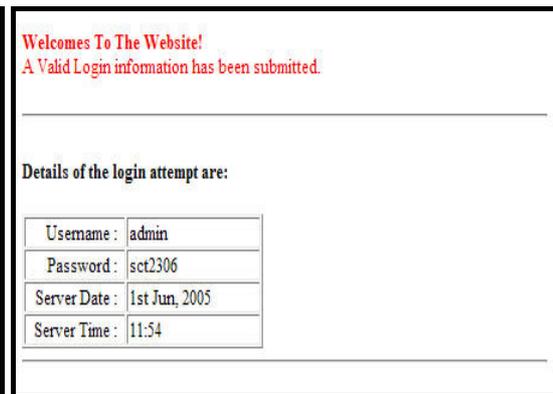


Diagram 14.4.4: Message by the logincheck.php, displayed for a valid login attempt.

The **logincheck.php** file retrieves data passed to it via **login.html**. PHP then begins processing the instructions contained within the file. The processing is as follows:

- Variables **\$dbuser** and **\$dbpass** are initialized and used to store the oracle user name and its corresponding password respectively
- The username and password fields are verified to see if they hold data. If either one is found empty, an error message indicating an invalid login is generated and stored into a variable named **\$Message** which will be rendered later. Refer diagram 14.4.1
- If a value has been passed for both the username and password, an attempt is made to connect to and access the **Oracle Database**. A PHP function **ocilogon()** is used to perform the database connection based on **\$dbuser** and **\$dbpass** variables passed as parameter to create the connection object. This connection object obtained is stored in the variable **\$rcq**
- A check is made whether the variable **\$rcq** holds a valid connection object. If not, an error message indicating an error in database connectivity is generated and stored into a variable named **\$Message** which will be rendered later. Refer diagram 14.4.2
- If creation of the connection object succeeds:
 - A SQL query is built and executed to validate if the username and password captured exists in the database
 - If the query retrieves the record, a message is displayed, which indicates a **Valid** login attempt. Refer diagram 14.4.4
 - If the query does not retrieve the record, an error message indicating **Incorrect** login attempt is displayed with a link to the login page. Refer diagram 14.4.3
- Finally, an HTML page is generated to display the value held by the variable **\$Message** along with the details of the login information

Changing Category Master Module To Support A Database

Retrieving information from a database is done using a SELECT statement. Similarly INSERT, UPDATE and DELETE statements are required to manipulate table information held within the Oracle database.

Having dealt with reading from a database, the same technique can be used to display data in tabular layout for the Category Master form. Consider the following table structure for storing information bound to the Category Master:

Table Definition:

Table Name : CATEGORY
Primary Key : CATEGORY_ID
Foreign Key : - -

Column Definition:

| Column Name | Data Type | Width | Allow Null | Default |
|------------------|-----------|-------|------------|---------|
| CATEGORY_ID | NUMBER | 2 | NOT NULL | |
| CATEGORY_NAME | VARCHAR2 | 35 | NOT NULL | |
| CATEGORY_REMARKS | VARCHAR2 | 255 | | NULL |

Table Description:

| Column Name | Description |
|------------------|---------------------------------|
| CATEGORY_ID | Internal ID |
| CATEGORY_NAME | Name of the category |
| CATEGORY_REMARKS | Remarks for the category if any |

Explanation:

The CATEGORY table is a lookup table, which records the job categories available.

The SQL statement used to create the table with the above specifications is as follows:

```
CREATE TABLE category (
  category_ID Number(2) NOT NULL,
  category_name VarChar2(35) NOT NULL,
  category_remarks VarChar2(255) DEFAULT NULL,
  PRIMARY KEY(category_ID)
);
```

The following SQL statements will populate the **CATEGORY** table with sample data:

```
INSERT INTO category (category_ID, category_name, category_remarks)
VALUES (1, 'Continental Cook', 'Specilised in continental preparations');
INSERT INTO category (category_ID, category_name, category_remarks)
VALUES (2, 'Arabic Cook', 'Specilised in Irani preparations');
```

14. INTEGRATION BETWEEN PHP AND ORACLE

```

INSERT INTO category (category_ID, category_name, category_remarks)
VALUES (3, 'Analyst Programmer', 'Specilised in Irani preparations');
INSERT INTO category (category_ID, category_name, category_remarks)
VALUES (4, 'Scaffolding Foreman', 'Well experienced personnel in
construction and development');
INSERT INTO category (category_ID, category_name, category_remarks)
VALUES (5, 'Cabin Steward', '');

```

To understand the development of the Category master form, glance through the pseudo structure for the same as shown below:

```
<?

```

| | |
|----------|--|
| 1 | Commented statements providing a description and purpose of the file. |
|----------|--|

```

/*
Date :- <Creation Date >
Author :- <Developer's Name>
Filename :- <Filename>
Purpose :- <Brief Description About File Functionality>
*/

```

| | |
|----------|------------------------------|
| 2 | Common PHP variables. |
|----------|------------------------------|

```

/* A variable holding the developer's name. */
$author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
$title = "Category Master Form";

```

| | |
|------------|--|
| 3.1 | PHP variables bound to <u>Database</u>. |
|------------|--|

| | |
|------------|---|
| 3.2 | PHP variables bound to HTML page aesthetics. |
|------------|---|

| | |
|----------|--|
| 4 | PHP code block to establish <u>Database connection</u>. |
|----------|--|

| | |
|----------|---|
| 5 | PHP code block for data manipulation on establishing connection. |
|----------|---|

| | |
|------------|--|
| 5.1 | PHP code block to perform the INSERT operation. |
|------------|--|

```

/* Verifying that the form is currently in the INSERT mode.
*/
if ($hidMode == "I") {

```

| | |
|--------------|--|
| 5.1.1 | PHP code block controlling INSERT operation based on duplication. |
|--------------|--|

5.2 PHP code block to perform the UPDATE operation.

```

/* Verifying that the form is currently in the UPDATE mode.
*/
if ($hidMode == "U") {

```

5.2.1 PHP code block controlling UPDATE operation based on duplication.**5.3 PHP code block to perform the DELETE operation.**

```

/* Checking if the form is in the delete mode. */
if ($hidMode == 'D') {

```

5.3.1 PHP code block to retrieve multiple records selection for deletion.**6 PHP variables re-initialization bound to the page processing/layout.**

```

/* A variable named $hidMode is initialised to 'I'. This variable
holds the form status in terms of I - Insert, U - Update and D -
Delete. */
$hidMode = 'I';
/* A variable $count is initialised to hold zero. This is used in
while loops to generate unique checkbox names for the tabular
layout. */
$count = 0;
?>

```

7 HTML code block rendering the actual page begins.

```

<HTML>
<HEAD>
<TITLE><?=$title;?></TITLE>

```

7.1 META tags bound to the HTML page.**7.2 SCRIPT tag holding JavaScript under the HEAD section.**

```

</HEAD>

```

8 BODY section.

```

<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0" TopMargin="0">

```

9 SCRIPT tag holding JavaScript unique to the current page under the BODY section.

| | |
|------------|---|
| 9.1 | JavaScript function to check contents of form field before submitting data contained in them. |
| | <pre> /* The chkBlanks() function verifies that the form field of category name is not left empty. If the field is empty or contains blank spaces, a message indicates to enter a valid value. If the field is not empty, the data is submitted. */ function chkBlanks() { </pre> |
| 9.2 | JavaScript function to validate contents of form field before submitting data contained in them. |
| | <pre> /* The function vldtFrmFlds() is called when the form data is about to be submitted. This function verifies the presence of changes in the form fields during the update mode. If changes are not encountered, then a indicating the same is displayed and any further processing gets terminated. */ function vldtFrmFlds() { </pre> |
| 9.3 | JavaScript function to prepare the form to accept a new set of data. |
| | <pre> /* The function setNewMode() is called to prepare the form to except a new set for data. This function is called directly when the Clear button is clicked. */ function setNewMode() { </pre> |
| 9.4 | JavaScript function to prepare the form to accept changes in the selected set of data. |
| | <pre> /* The function setEditMode() is called to prepare the form to update data held in the tabular layout. This function is called directly when a link in the tabular layout is clicked. The function also sets the hidden field form mode to update and populates the text fields. */ function setEditMode(id, name, rmrk) { </pre> |
| 9.5 | JavaScript function to prepare the form to delete selected set(s) of data. |
| | <pre> /* The function setDelMode() is called to prepare the form to remove data from the tabular layout. This function is called directly when the Delete button is clicked. */ function setDelMode() { </pre> |
| 9.6 | SCRIPT tag holding JavaScript under the BODY section ends. |
| | <code></SCRIPT></code> |

| | |
|---|-----------------------------|
| 10 | Parent Table begins. |
| <pre><!-- Outer Table Code Begins. --> <TABLE Align="center" BgColor="<?=\$gl_mstr_frst_bar_color;?>" Border="0" CellPadding="0" CellSpacing="0" Name="TlbOuter" Width="90%"></pre> | |

| | |
|-------------|---|
| 10.1 | Parent Table → First Row: Columns to hold page title and a message header. |
|-------------|---|

| | |
|----------|---------------------------------------|
| Category | <Message from the previous operation> |
|----------|---------------------------------------|

Diagram 14.5.1

| | |
|---|--|
| 11 | Form Section holding Data and Control objects begins. |
| <pre><!-- Initialising a form object, which will submit data captured on the form to the processing file. --> <FORM Action="<FileName>" Method="post" Name="frmMstrCat" onSubmit="return chkBlanks();"></pre> | |

| | |
|--|---|
| 11.1 | Form Hidden Variable to identify the current page. |
| <pre><!-- Declaring a hidden form field used to identify the current (i.e. category master) page. --> <INPUT Name="hidPage" Type="hidden" Value="<Page Name>"></pre> | |

| | |
|--|--|
| 11.2 | Form Hidden variables to maintain a copy of the data held by the form fields just before changes attempted. This is useful to confirm change(s) made to the form fields before an update is fired. (To avoid database or flat files concern as the case may be) |
| <pre><!-- Declaring hidden form fields required for data validation. --> <INPUT Name="hidCatID" Size="2" Type="hidden" Value=""> <INPUT Name="hidCatName" Size="2" Type="hidden" Value=""> <INPUT Name="hidCatRmrk" Size="2" Type="hidden" Value=""></pre> | |

| | |
|---|---|
| 11.3 | Form Hidden Variable to determine the form operation. Default, being Insert operation. |
| <pre><!-- Declaring a hidden form field used for determining the form mode. It holds 'I' for Insert, 'U' for Update and 'D' for Delete. It will hold 'I' when the page is rendered for the 1st time. --> <INPUT Name="hidMode" Type="hidden" Value="<?=\$hidMode;?>"></pre> | |

14. INTEGRATION BETWEEN PHP AND ORACLE

| | |
|-------------|--|
| 11.4 | Form Hidden Variable holding entries (checkbox values) selected for deletion. |
|-------------|--|

```
<!-- Declaring a hidden form field used for identifying records
which have been selected for deletion. -->
<INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
```

| | |
|-----------|--|
| 12 | Parent Table → Second Row: Columns to hold the actual d/e form. |
|-----------|--|

```
<TR Height="300" VAlign="top">
<TD Align="center" Border="1" ColSpan="10"><BR>
```

| | |
|-----------|--|
| 13 | First Child Table holding d/e Form objects, to <u>Capture</u> / <u>Control</u> data begins. |
|-----------|--|

```
<!-- Form Table Code Begins. -->
<TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"
Name="TlbInner" Width="90%">
```

| | |
|-------------|---|
| 13.1 | First Child Table → First and Second Rows: Columns to hold form title. |
|-------------|---|



Diagram 14.5.2

| | |
|-------------|--|
| 13.2 | First Child Table → Third & Forth Rows: Columns holding label and data capture objects to capture data. |
|-------------|--|

The screenshot shows a web form with a title bar containing 'Category' and '<Message from the previous operation>'. Below the title bar is the heading 'Master Setup - Category'. There are two input fields: 'Category:' followed by a text box, and 'Remarks:' followed by a larger text box.

Diagram 14.5.3

13.3 First Child Table → Last Row:
Columns holding data control objects.

This screenshot is identical to Diagram 14.5.3, but it includes two buttons, 'Save' and 'Clear', positioned below the 'Remarks' input field.

Diagram 14.5.4

13.4 First Child Table ends.

<?

14 PHP code block retrieving data to create the tabular layout by Database connection. The tabular layout is generated only after data gets retrieved.

?>

14. INTEGRATION BETWEEN PHP AND ORACLE

15 **Second Child Table holding the tabular layout, to display / control data captured begins.**

```
<!-- Data Table Code Begins. -->
<TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
Width="90%">
```

16 **Second Child Table → First Row: Columns to header row.**

The screenshot displays a web form titled "Master Setup - Category". At the top, there is a message: "<Message from the previous operation>". Below this, the form contains two input fields: "Category:" and "Remarks:". To the right of the "Remarks:" field are two buttons: "Save" and "Clear". Below the form is a table with a dark purple header row. The first cell of the header row contains a "Delete" button. The second and third cells of the header row are labeled "Category" and "Remarks" respectively.

Diagram 14.5.5

<?

17 **PHP code block to populate data in the tabular layout using a loop.**

```
/* A variable named $rowBgColor hold the hexadecimal colour
value. This variable is used decided the background colour of
rows for data in the tabular format. */
$rowBgcolor = $gl_mstr_scnd_bar_color;
/* An iterative code performed as long as a single row of
category data is available. */
do {
/* Incrementing the value of the variable used for assigning
unique names to check boxes display of each row of data. */
$count = $count + 1;
```

17.1 **PHP code block to control the background colour for the current row.**

```
/* If the colour code in for the current row matches the
value for the second row. */
if ($rowBgcolor == $gl_mstr_scnd_bar_color) {
```

```

/* Colour code for the current row set to match the value
for the colour code for the first bar. */
    $rowBgcolor = $gl_mstr_frst_bar_color;    }
/* If the colour code in for the current row matches the
value for the first row. */
    else {
/* Colour code for the current row set to match the value
for the colour code for the second bar. */
    $rowBgcolor = $gl_mstr_scnd_bar_color;    }
?>

```

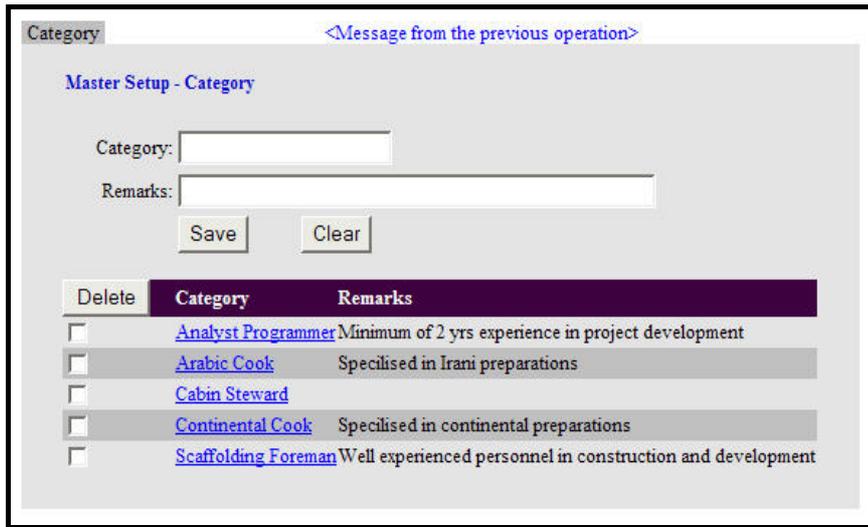


Diagram 14.5.6

- | | |
|--|--|
| 17.2 | Loop and the Second Child table ends. |
| <pre> <? } while(ocifetchinto(\$parsed, \$line, OCI_ASSOC+OCI_NUM)) ?> </TABLE> </pre> | |
| 18 | Form holding Data and Control objects ends. |
| <pre> </FORM> </pre> | |
| 19 | Parent Table ends. |
| <pre> </TABLE> <!-- Outer Table Code Ends. --> </pre> | |
| 20 | HTML code block along with BODY section rendering the actual page ends. |
| <pre> </BODY></HTML> </pre> | |

14. INTEGRATION BETWEEN PHP AND ORACLE

To begin building database support to the Category Master Module created earlier, the **ctgry_mstr.php** file will be modified as specified below. To avoid rewriting the entire page create a copy of the **ctgry_mstr.php** and **mstrscript.js** files from the **Chap12_Cds** directory and store it in the **Chap14_Cds** directory.

Open the **ctgry_mstr.php** file in an ASCII editor and perform the following change in the PHP code block:

- Delete the following block of PHP code, which verifies the form delete mode (i.e. \$hidMode holds D). This is done as the code spec for the delete operation will be built later. The block of PHP code to be deleted is found in the declaration section. (Refer point **5.3** under the sub-title **Skeleton Of The Code Undertaken For Development.**)

```

...
/* A variable holding the title for the page. */
$title = "Category Master Form";

/* If the form mode is set to Delete. */
if ($hidMode == 'D') {
    /* A variable $Message is initialised to hold a message,
    which has occurred during the previous (delete) operation.
    */
    $Message = '<BR><FONT Color="RED">* Details for ".$hidDelRcrdLst.' has
    been deleted.</FONT>'; }

/* A variable named $hidMode is initialised to 'I'. This variable holds the form
status in terms of I - Insert, U - Update and D - Delete. */
$hidMode = 'I';
?>
...

```

- Insert the code block mentioned below, before the initialization of the **\$hidMode** variable. The inserted code will create an Oracle connection object and then check whether it was really created. If the creation fails an error message is generated and stored in the variable **\$Message**. (Refer points **3.1** and **4** under the sub-title **Skeleton Of The Code Undertaken For Development.**)

```

...
/* A variable holding the title for the page. */
$title = "Category Master Form";

/* Variables used to store Database access information. */
$dbuser = "dba_pm";
$dbpass = "sct2306";
/* The function ocilogon() accepts the oracle login information
to create the actual connection to the database. This connection
made available whenever required by referencing $rcq, which the
connection object created by the ocilogon() function. */
$rcq = ocilogon($dbuser, $dbpass);
/* Checking whether the creation of the connection object failed
or not. If failed, an error message is generated. */
if(!$rcq){

```

```

$Message = '<FONT Color="red"><B>Database connection
            error!</B><BR>Please contact the Oracle Database
            Administrator.</FONT><BR>'; }
/* A variable named $hidMode is initialised to 'I'. This variable holds the form
status in terms of I - Insert, U - Update and D - Delete. */
$hidMode = 'I';
?>
...

```

- Scroll to the section where the HTML code block defining the table holding the data capture form ends. This block is immediately followed by PHP code spec, which checks if the form field holds a value. (Refer point 14 under the sub-title **Skeleton Of The Code Undertaken For Development.**) Replace this code block with the following which retrieve data from the category table and stores the same in to a record set:

```

...
        <INPUT Name="cmdReset" onClick="setNewMode();" Type="button" Value="Clear">
    </TD>
</TR></TABLE><BR>
<!-- Form Table Code End. -->
<?
/* Building and executing a SQL query to retrieve records from
the CATEGORY table after sorting the category names in alphabetic
order. */
$query = "SELECT * FROM category ORDER BY category_name";
$parsed = ociparse($rcq, $query);
ociexecute($parsed);
/* Checking if records have been retrieved from the Category
table. */
if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {
?>
    <!-- Data Table Code Begins. -->
    <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Width="90%"><TR
        BgColor="#400040">
        <TD Width="15%"><INPUT Name="cmdDelete" onClick="setDelMode();" Type="button"
            Value="Delete"></TD>
...

```

- The above block is followed by the HTML code block, which defines the header row of the tabular layout (grid). Replace the code block that follows immediately after the declaration of the header row upto the end of the column (i.e. the </TD> tag) before the </FORM> tag. (Refer points 17, 17.1 and 17.2 under the sub-title **Skeleton Of The Code Undertaken For Development.**) The new code spec that will appear will populate the grid i.e. the tabular layout with the records retrieved using a loop:

```

...
    <!-- Data Table Code Begins. -->
    <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Width="90%"><TR
        BgColor="#400040">
        ...
    </TR>

```

14. INTEGRATION BETWEEN PHP AND ORACLE

```

<?
/* An iterative code performed as long as a single row of
category data is available. */
$count = 0;

do {
    $count = $count + 1;
?>
    <TR BgColor="#E4E4E4">
        <TD><INPUT Name="chk<?=( $count);?>" Type="checkbox"
            Value="<?=( $line['CATEGORY_ID'];?>"></TD>
        <TD><A
            HRef="JavaScript:setEditMode('<?=$line['CATEGORY_ID'];?>,
                '<?=$line['CATEGORY_NAME'];?>',
                '<?=$line['CATEGORY_REMARKS'];?>')"><?
                echo($line['CATEGORY_NAME']); ?></A></TD>
        <TD><? echo($line['CATEGORY_REMARKS']); ?></TD>
    </TR>
<?
    } while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM)) ?>
</TABLE><BR>
<?
} ?>
</FORM>
</TR></TABLE>
<!-- Outer Table Code Ends.-->
</BODY>
</HTML>

```

Save the changes made to **ctgry_mstr.php** and test the changes by calling the file in a Web Browser.

When the page appears, as shown in diagram 14.6.1, the d/e form along with the populated tabular layout is displayed. The data displayed in the tabular layout is retrieved from the Oracle table named CATEGORY.

| Delete | Category | Remarks |
|--------------------------|-------------------------------------|--|
| <input type="checkbox"/> | Analyst Programmer | Specilised in Irani preparations |
| <input type="checkbox"/> | Arabic Cook | Specilised in Irani preparations |
| <input type="checkbox"/> | Cabin Steward | |
| <input type="checkbox"/> | Continental Cook | Specilised in continental preparations |
| <input type="checkbox"/> | Scaffolding Foreman | Well experienced personnel in construction and development |

Diagram 14.6.1: Output for ctgry_mstr.php.

REMINDER



In addition to retrieving data from the CATEGORY table the SELECT statement will sort the records on the basis of CATEGORY_NAME column.

Enhancement To Page Aesthetics

Having managed to display data for the Category Master module, an attempt to improve the appearance of the page rendered by **ctgry_mstr.php** can be made. The simplest change would be to display a different background colour for alternate rows in the tabular layout. Refer diagram 14.6.2.

To achieve this open the file in an ASCII editor and add the following code blocks:

- Assign hexadecimal colour code to variables by mentioning them in the declaration section of the file, (Refer point 3.2 under the sub-title **Skeleton Of The Code Undertaken For Development.**):

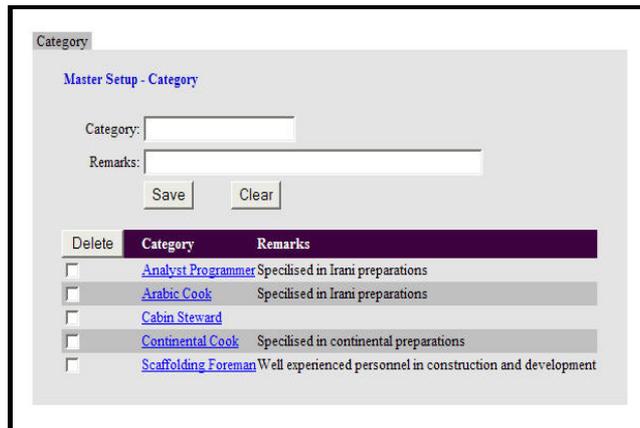


Diagram 14.6.2: New Tabular Layout for ctgry_mstr.php.

```

...
/* A variable holding the title for the page. */
$title = "Category Master Form";

/* Variables used to store colours for the master page. */
$gl_mstr_top_bar_color = '#400040';
$gl_mstr_frst_bar_color = '#E4E4E4';
$gl_mstr_scnd_bar_color = '#C0C0C0';

/* Variables used to store Database access information. */
$dbuser = "dba_pm";
$dbpass = "sct2306";
...
    
```

- In the outer table's **<TABLE>** tag, change value of the **BgColor** attribute as mentioned below, (Refer point 10 under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```

...
</SCRIPT>
<!-- Outer Table Begins. -->
<TABLE Align="center" BgColor="<?=$gl_mstr_frst_bar_color;?>"
Border="0" CellPadding="0" CellSpacing="0" Name="TlbOuter" Width="90%"><TR>
<TD Align="center" Border="1" BgColor="#C0C0C0" Width="10%">Category</TD>
<TD Align="center" BgColor="#FFFFFF" ColSpan="9" Width="90%"><?=$Message;?></TD>
</TR>
...
    
```

- In the **<TR>** tag of the Header row in the tabular layout, change the value for the **BgColor** attribute as mentioned below, (Refer point 16 under the sub-title **Skeleton Of The Code Undertaken For Development.**):

14. INTEGRATION BETWEEN PHP AND ORACLE

```

...
    <!-- Data Table Code Begins. -->
    <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Width="90%">
        <TR BgColor="<?=$gl_mstr_top_bar_color;?>">
            <TD Width="15%"><INPUT Name="cmdDelete" onClick="setDelMode();" Type="button"
                Value="Delete"></TD>
...

```

- Replace the PHP code block immediately after the declaration of the header row upto the declaration of the data row (i.e. the second row in the tabular layout). (Refer points **17** and **17.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**) The following code spec actually alternates between the two colors defined:

```

...
        <TD><FONT Color="#FFFFFF"><B>Category</B></FONT></TD>
        <TD><FONT Color="#FFFFFF"><B>Remarks</B></FONT></TD>
    </TR>
<?
    /* A variable named $rowBgColor hold the hexadecimal colour
    value. This variable is used decided the background colour for
    rows of data in the tabular format. */
    $rowBgcolor = $gl_mstr_scnd_bar_color;
    /* An iterative code performed as long as a single row of
    category data is available. */
    do {
        /* Incrementing the value of the variable used for assigning
        unique names to check boxes display of each row of data. */
        $count = $count + 1;
        /* If the colour code in for the current row matches the
        value for the second row. */
        if ($rowBgcolor == $gl_mstr_scnd_bar_color) {
            /* Colour code for the current row set to match the value
            for the colour code for the first bar. */
            $rowBgcolor = $gl_mstr_frst_bar_color;
        }
        /* If the colour code in for the current row matches the
        value for the first row. */
        else {
            /* Colour code for the current row set to match the value
            for the colour code for the second bar. */
            $rowBgcolor = $gl_mstr_scnd_bar_color;
        }
    }
?>
    <TR BgColor="#E4E4E4">
        <TD><INPUT
            Name="chk<?=( $count);?>"
            Type="checkbox"
            Value="<?=( $line['CATEGORY_ID']);?>"></TD>
...

```

- In the **<TR>** tag of the data row in the tabular layout, change the value of the **BgColor** attribute as mentioned below, (Refer point **17.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```

...
    else {
        /* Colour code for the current row set to match the value for the colour
        code for the second bar. */
        $rowBgcolor = $gl_mstr_scnd_bar_color;    }
?>
<TR BgColor="<?=$rowBgcolor;?>">
    <TD><INPUT Name="chk<?=( $count);?>" Type="checkbox"
    Value="<?=( $line['CATEGORY_ID']);?>"></TD>
...

```

This completes the changes for the new appearance of the tabular layout in the Category Master page.

Using PHP To Insert Data Into The Database

The change brought in by the above code block is limited to retrieving data from the database and displaying with aesthetics. Apart from this the form does not allow any data entry. This means that data entered into the form fields will be lost unless it is saved into the database.

The illustration below improvises the Category master page and allows it to use the Oracle Database to store data (captured by the form fields). To continue with the illustration create a copy of the **ctgr_y_mstr.php** file.

Insert the following PHP code block immediately after the code spec that verifies the connection object at the beginning of the file, (Refer point **5, 5.1 and 5.1.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```

...
/* Checking whether the creation of the connection object failed or not. If
failed, an error message is generated. */
if (!$rcq){
    $Message = '<FONT Color="red"><B>Database connection error!</B><BR>Please contact the Oracle
    Database Administrator.</FONT><BR>';
}

/* If the creation of the connection object succeeded. */
else {
    /* If Category name has been passed through the variable
    $txtCatName, (i.e. the $txtCatName is not empty). */
    if (!empty($txtCatName)) {
        /* Verifying that the form is currently in the INSERT mode.
        */
        if ($hidMode == "I") {

```

14. INTEGRATION BETWEEN PHP AND ORACLE

```

/* Building and executing a SQL query to retrieve records
for the Category table such that the contents of the
Category_Name matches the value entered in the category
name field in the form. */
$query = "SELECT * FROM Category WHERE
        TRIM(LOWER(category_name))=TRIM(LOWER('"
        .$txtCatName."'))";
$parsed = ociparse($rcq, $query);
ociexecute($parsed);

/* Checking if any record was retrieved by the above SQL
query. If the same category name exists in the database,
an error message is displayed to the user, which
indicates that the Insert operation has failed. */
if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {
/* Generating and storing an error message which
indicates a duplication of category. */
    $Message = '<FONT Color="red">Entry for
                <B>'.$txtCatName.'</B> already exists. </FONT>';
}
/* When the same category name does not exist in the
databases, the new category is stored into the database.
*/
else {
/* Building and executing a SQL query to INSERT the
new category into the category table. The new category
is assigned a unique ID, which is the highest ID
assigned to a category incremented by one. */
    $query = "INSERT INTO category (category_ID, category_name,
        category_remarks) VALUES((SELECT
        COALESCE(MAX(category_ID), 0)+1 FROM
        category),'.$txtCatName.', ".$txtCatRmrk."");
    $parsed = ociparse($rcq, $query);
    ociexecute($parsed);
/* Generating and storing a message that indicates the
insert operation was successful. */
    $Message = '<FONT Color="blue">Entry for
                <B>'.$txtCatName.'</B> added
                successfully.</FONT>';
    }
}
}
}
}

/* A variable named $hidMode is initialised to 'I'. This variable holds the form
status in terms of I - Insert, U - Update and D - Delete. */
$hidMode = 'I';
?>
...

```

Save the changes made to the **ctgry_mstr.php** file and run it within the web browser. When the page appears enter a new category and click **Save**. Refer diagram 14.7.1.

When a call is made to the **ctgry_mstr.php** file with new category information the following processes occur:

- ❑ A check is made to ensure that **\$rcq** holds a reference to the Oracle connection object
- ❑ A check is made to ensure that data is submitted by the previous instance of the page:
 - A check is made to ensure that the form is in the insert mode (i.e. **\$hidMode** holds the value **I**). If yes, then:
 - An SQL query is built and executed to check for duplication in the record submitted. This is done by retrieving the data filtered on the basis of the new category submitted
 - If the above SQL query retrieved any record(s) i.e. the new category name submitted already exists in the database:
 - An error message, indicating an Insert operation failure due to duplication of category name is generated and stored into **\$Message**
 - If no records are returned by the above SQL query:
 - An SQL query is built and executed to INSERT the new category into the category table. The new category is assigned a unique ID
 - A message indicating a successful insert is generated and stored into **\$Message**
- ❑ Finally, the functionality of rendering a HTML page for **ctgry_mstr.php** is processed. This will include:
 - Displaying the value stored in **\$Message**
 - Laying out the form for capturing category information
 - Laying out the grid for displaying category information stored in category table

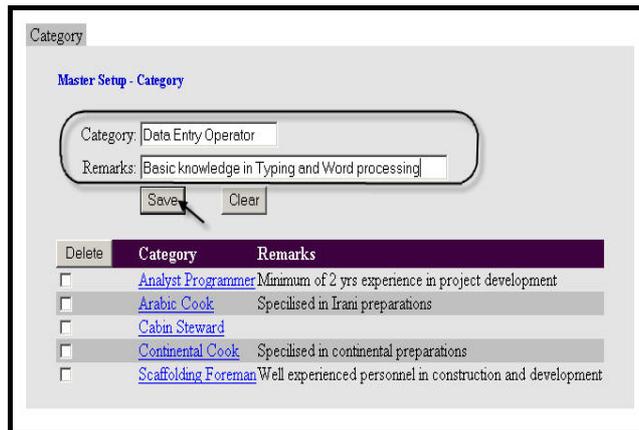


Diagram 14.7.1: Adding a new entry with category information via the **ctgry_mstr.php** file.

14. INTEGRATION BETWEEN PHP AND ORACLE

The new category will be listed as an entry in the tabular layout / grid. Refer diagram 14.7.2.

Using PHP To Change Data Held In The Database

A successful Insert operation is followed by building the update operation for the form. This means that data held in the database will be accessed and changed. The changes will then be reflected in the table by updating the data held within them.

The screenshot shows a web form titled "Master Setup - Category". At the top, a message box says "Entry for Data Entry Operator added successfully!". Below the message are two input fields: "Category:" and "Remarks:". There are "Save" and "Clear" buttons. Below the form is a table with columns "Delete", "Category", and "Remarks". The table contains several entries, with "Data Entry Operator" highlighted and circled in red. The table data is as follows:

| Delete | Category | Remarks |
|--------------------------|-------------------------------------|--|
| <input type="checkbox"/> | Analyst Programmer | Specilised in Irani preparations |
| <input type="checkbox"/> | Arabic Cook | Specilised in Irani preparations |
| <input type="checkbox"/> | Cabin Steward | |
| <input type="checkbox"/> | Continental Cook | Specilised in continental preparations |
| <input type="checkbox"/> | Data Entry Operator | Basic knowledge in Typing and Word processing |
| <input type="checkbox"/> | Scaffolding Foreman | Well experienced personnel in construction and development |

Diagram 14.7.2: Entry added by ctgry_mstr.php.

Clicking on the hyper-linked text (in the second column in the tabular layout) will invoke the form's update mode. The values bound to the entry, whose hyperlink is clicked, is passed to the form fields permitting data modification.

To illustrate the Update technique using database tables, the code in **ctgry_mstr.php** will be used as base. Insert / modify PHP code block as mentioned below:

- After the PHP code block checking for form's insert mode ends, (Refer points 5.2 and 5.2.1 under the sub-title **Skeleton Of The Code Undertaken For Development.**) place the following:

```

...
        /* Generating and storing a message that indicates the insert
        operation was successful. */
        $Message = '<FONT Color="blue">Entry for <B>'.$txtCatName.'</B> added
        successfully.</FONT>';
    }
}

/* Verifying that the form is currently in the UPDATE mode.
*/
if ($hidMode == "U") {
    /* Building and executing a SQL query to retrieve records
    for the category table such that the contents of the
    Category_Name matches the value entered in the category
    name field in the form, while the unique record number
    does not match the unique record number held in the
    hidden form field $hidCatID. */
    $query = "SELECT * FROM category WHERE
    TRIM(LOWER(category_name))=TRIM(LOWER('".$txtCat
    Name."')) AND NOT (category_ID='".$hidCatID."");
    $parsed = ociparse($rcq, $query);
    ociexecute($parsed);
}

```


When a call is returned to the **ctgry_mstr.php** file with the modified category information the following processes occur:

- A check is made to ensure that **\$rcq** holds a reference to the Oracle connection object
- A check is made to ensure that category name field holds data. If it holds data:
 - A check is made to ensure that the form is in the update mode, (i.e. **\$hidMode** holds the value **U**). If yes, then:
 - The newly entered record details, in this case the category name, is checked for duplication against the data retrieved from the file. Additionally it is ensured that the category name duplication check is avoided against the same record.
 - If the category name is being duplicated
 - An error message, indicating an Update operation failure due to duplication of category name is generated and stored into **\$Message**
 - If there is no duplication taking place:
 - A SQL query is built and executed which will perform the update operation on the category table
 - A message, indicating a successful update, is generated and stored into **\$Message**

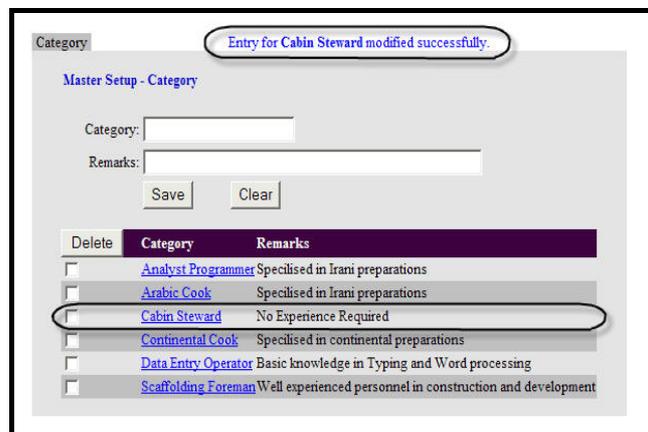


Diagram 14.8.2: Entry changed by ctgry_mstr.php.

- Finally, the functionality of rendering a HTML page for **ctgry_mstr.php** is processed. This will include:
 - Displaying the value stored in **\$Message**
 - Laying out the form for capturing category information
 - Laying out the grid for displaying category information stored in category table

The modified category will be listed as an entry in the tabular layout / grid. Refer diagram 14.8.2.

Using PHP To Remove Data Held In The Database

Having completed the data insert and the data manipulation operations using database tables, it is time to move on to the delete operation. Although, the delete functionality at the HTML level will remain the same, the change occurs in the PHP processing.

When is clicked, a list of record(s) marked for deletion is created and submitted to the next instance of the same page.

As the database holds category data, when a list of records selected for deletion is received from the page's previous instance, the delete operation requires the following modification in the PHP code block:

- After the PHP code block, which checks for captured data submitted via **\$txtCatName** ends, replace the check for the page in delete mode with the following, (Refer point **5.3** and **5.3.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```

...
/* If Category name has been passed through the variable $txtCatName, (i.e. the
$txtCatName is not empty). */
if (!empty($txtCatName)) {
    /* Verifying that the form is currently in the INSERT mode. */
    if ($hidMode == "I") {
...
        }
    }

    /* Verifying that the form is currently in the UPDATE mode. */
    if ($hidMode == "U") {
...
        }
    }
}

/* Checking if the form is in the delete mode. */
if ($hidMode == 'D') {
    /* Building and executing a SQL query to retrieve category
names from the category table such that the unique record
number matches the entries selected for deletion, (i.e.
value held in $hidDelRcrdLst). */
    $query = "SELECT category_name FROM category WHERE category_ID
            IN('.$hidDelRcrdLst.)";
    $parsed = ociparse($rcq, $query);
    ociexecute($parsed);

    /* An iterative code performed as long as a single row of
category data is available. */
    while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM)) {
        /* Checking if the variable $delEntry is empty, to
determine the method for storing the current category
into the variable $delEntry. */
        if(empty($delEntry)) {
            $delEntry = $line['CATEGORY_NAME'];    }
        else { $delEntry = $delEntry.", ".$line['CATEGORY_NAME']; }
    }
    /* Checking if the variable $delEntry is not empty. */
    if(!empty($delEntry)) {

```

14. INTEGRATION BETWEEN PHP AND ORACLE

```

/* Building and executing a SQL query to delete record(s)
for the category table. The condition of deleting
record(s) is that the unique record number should exists
in the list of entries selected for deletion, (i.e. value
held in $hidDelRcrdLst). */
$query = "DELETE FROM category WHERE category_ID
        IN('.$hidDelRcrdLst.')";
$parsed = ociparse($rcq, $query);
ociexecute($parsed);
/* A variable $Message is initialised to hold a message,
which list the record(s) deleted. */
$Message = '<BR><FONT Color="RED">Details for
           <B>'.$delEntry.'</B> has been deleted.</FONT>';
}
/* If the variable $delEntry is empty. */
else {
/* A variable $Message is initialised to hold an error
message, which indicates that records for deletion do not
exists in the database. */
$Message = '<BR><FONT Color="RED">Records selected for deletion
           do not exists. </FONT>';
}
}
}

/* A variable named $hidMode is initialised to 'I'. This variable holds the form
status in terms of I - Insert, U - Update and D - Delete. */
$hidMode = 'I';
?>
...

```

Save the changes made to the **ctgry_mstr.php** file and run it within the web browser. When the page appears, select the checkbox(es) bound to the records desired to be deleted from the tabular layout and click . Refer diagram 14.9.1.

When a call is returned to the **ctgry_mstr.php** file with list of categories selected for deletion the following processes occur:

- A check is made to ensure that **\$rcq** holds a reference to the Oracle connection object
- A check is made to ensure that the form is in the delete mode, (i.e. **\$hidMode** holds the value **D**). If yes, then:

| Delete | Category | Remarks |
|-------------------------------------|---------------------|--|
| <input type="checkbox"/> | Analyst Programmer | Minimum of 2 yrs experience in project development |
| <input type="checkbox"/> | Arabic Cook | Specilised in Iran preparations |
| <input type="checkbox"/> | Cabin Steward | No Experience Required |
| <input type="checkbox"/> | Continental Cook | Specilised in continental preparations |
| <input checked="" type="checkbox"/> | Data Entry Operator | Basic knowledge in Typing and Word processing |
| <input type="checkbox"/> | Scaffolding Foreman | Well experienced personnel in construction and development |

Diagram 14.9.1: Deleting an entry with category information via the **ctgry_mstr.php** file.

- An SQL query is built and executed to retrieve category names from the category table for the unique record number held in **\$hidDelRcrdLst**
- A loop traverses through the record(s) retrieved by the above query to perform the following:
 - The category names retrieved by the above query is stored in a comma separated form in the **\$delEntry** variable. This variable will hold the category names to be displayed as the records deleted after the actual delete operation is performed
- A check is made to ensure that the variable **\$delEntry** holds a value. If yes, then:
 - An SQL query is built and executed to delete record(s) from the category table based on the unique record values held in **\$hidDelRcrdLst**
 - A message indicating a successful delete along with the category names deleted is generated and stored into **\$Message**
- If the variable **\$delEntry** does not hold a value, a message indicating that records selected for deletion do not exist, is generated and stored into **\$Message**

□ Finally, the functionality of rendering a HTML page for **ctgry_mstr.php** is processed. This will include:

- Displaying the value stored in **\$Message**
- Laying out the form for capturing category information
- Laying out the grid for displaying category information stored in the **category** table found in the Oracle database

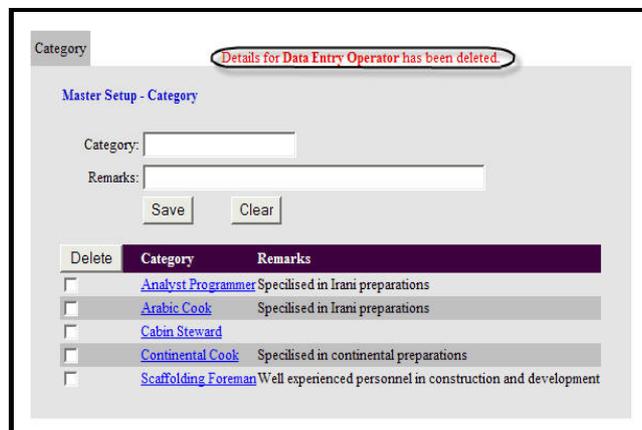


Diagram 14.9.2: Entry deleted by ctgry_mstr.php.

The name(s) of category deleted (using the values held by the **\$delEntry** variable) will be listed as shown in diagram 14.9.2.

This completes the Category Master module using database table. The entire code for the **ctgry_mstr.php** appears as listed below:

```
<?
/*Date :- 28/05/05
 Author :- Hansel Colaco.
 Filename :- ctgry_mstr.php
 Purpose :- Allows display, insert, updates and delete of
 category master table.
*/
```

14. INTEGRATION BETWEEN PHP AND ORACLE

```

/* A variable holding the developers name. */
$author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
$title = "Category Master Form";

/* Variables used to store Database access information. */
$dbuser = "dba_pm";
$dbpass = "sct2306";

/* Variables used to store colours for the master page. */
$gl_mstr_top_bar_color = '#400040';
$gl_mstr_frst_bar_color = '#E4E4E4';
$gl_mstr_scnd_bar_color = '#C0C0C0';

/* The function ocilogon() accepts the oracle login information
to create the actual connection to the database. This connection
made available whenever required by referencing $rcq, which the
connection object created by the ocilogon() function. */
$rcq = ocilogon($dbuser, $dbpass);

/* Checking whether the creation of the connection object failed
or not. If failed, an error message is generated. */
if(!$rcq) {
    $Message = '<FONT Color="red"><B>Database connection
                error!</B><BR>Please contact the Oracle Database
                Administrator.</FONT><BR>';
}
/* If the creation of the connection object succeeded. */
else {
    /* If Category name has been passed through the variable
    $txtCatName, (i.e. the $txtCatName is not empty). */
    if (!empty($txtCatName)) {
        /* Verifying if the form is currently in the INSERT mode. */
        if ($hidMode == "I") {
            /* Building and executing a SQL query to retrieve records
            for the Category table such that the contents of the
            Category_Name matches the value entered in the category
            name field in the form. */
            $query = "SELECT * FROM Category WHERE
                    TRIM(LOWER(category_name))=TRIM(LOWER('".$txtCat
                    Name."'))";
            $parsed = ociparse($rcq, $query);
            ociexecute($parsed);

            /* Checking if any record was retrieved by the above SQL
            query. If the same category name exists in the database,
            an error message is displayed to the user, which
            indicates that the Insert operation has failed. */
            if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {

```

```

        /* Generating and storing an error message which
        indicates a duplication of category. */
        $Message = '<FONT Color="red">Entry for
        <B>'.$txtCatName.'</B> already exists. </FONT>';
    }
    /* When the same category name does not exist in the
    databases, the new category is stored into the database.
    */
    else {
        /* Building and executing a SQL query to INSERT the
        new category into the category table. The new category
        is assigned a unique ID, which is the highest ID
        assigned to a category incremented by one. */
        $query = "INSERT INTO category (category_ID, category_name,
        category_remarks) VALUES((SELECT
        COALESCE(MAX(category_ID), 0)+1 FROM
        category),".$txtCatName.", ".$txtCatRmrk.");";
        $parsed = ociparse($rcq, $query);
        ociexecute($parsed);
        /* Generating and storing a message that indicates the
        insert operation was successful. */
        $Message = '<FONT Color="blue">Entry for
        <B>'.$txtCatName.'</B> added
        successfully.</FONT>';
    }
}

/* Verifying that the form is currently in the UPDATE mode.
*/
if ($hidMode == "U") {
    /* Building and executing a SQL query to retrieve records
    for the category table such that the contents of the
    Category_Name matches the value entered in the category
    name field in the form, while the unique record number
    does not match the unique record number held in the
    hidden form field $hidCatID. */
    $query = "SELECT * FROM category WHERE
    TRIM(LOWER(category_name))=TRIM(LOWER('".$txtCat
    Name."')) AND NOT (category_ID='".$hidCatID."";";
    $parsed = ociparse($rcq, $query);
    ociexecute($parsed);

    /* Checking if any record was retrieved by the above SQL
    query. If the same category name exists in another
    record, an error message is displayed to the user, which
    indicates that the Update operation has failed. */
    if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {

```

14. INTEGRATION BETWEEN PHP AND ORACLE

```

        /* Generating and storing an error message which
        indicates a duplication of category. */
        $Message = '<FONT Color="red">The modified category
        <B>'.$txtCatName.'</B> already exists.</FONT>';
    }
    /* When the same category name does not exist in another
    record, the modified category data is stored into the
    table. */
    else {
        /* Building and executing a SQL query to Update the
        category table with the change in the category data.
        The record to be changed is determined on the bases of
        the unique ID assigned to the category data. */
        $query = "UPDATE category SET
        category_name='".$txtCatName."',
        category_remarks='".$txtCatRmrk.'" WHERE
        category_ID='".$hidCatID;
        $parsed = ociparse($rcq, $query);
        ociexecute($parsed);
        /* Generating and storing a message that indicates the
        update operation was successful. */
        $Message = '<FONT Color="blue">Entry for
        <B>'.$txtCatName.'</B> modified
        successfully.</FONT>';
    }
}

/* Checking if the form is in the delete mode. */
if ($hidMode == 'D') {
    /* Building and executing a SQL query to retrieve category
    names from the category table such that the unique record
    number matches the entries selected for deletion, (i.e.
    value held in $hidDelRcrdLst). */
    $query = "SELECT category_name FROM category WHERE category_ID
    IN('$hidDelRcrdLst)";
    $parsed = ociparse($rcq, $query);
    ociexecute($parsed);
    /* An iterative code performed as long as a single row of
    category data is available. */
    while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM)) {
        /* Checking if the variable $delEntry is empty, to
        determine the method for storing the current category
        into the variable $delEntry. */
        if(empty($delEntry)) {
            $delEntry = $line['CATEGORY_NAME']; }
        else { $delEntry = $delEntry.", ".$line['CATEGORY_NAME']; }
    }
}

```

```

/* Checking if the variable $delEntry is not empty. */
if(!empty($delEntry)) {
/* Building and executing a SQL query to delete record(s)
for the category table. The condition of deleting
record(s) is that the unique record number should exists
in the list of entries selected for deletion, (i.e. value
held in $hidDelRcrdLst). */
$query = "DELETE FROM category WHERE category_ID
IN(".$hidDelRcrdLst.")";


$parsed = ociparse($rcq, $query);



ociexecute($parsed);


/* A variable $Message is initialised to hold a message,
which list the record(s) deleted. */


$Message = '<BR><FONT Color="RED">Details for
<B>'. $delEntry.'</B> has been deleted.</FONT>';


}
/* If the variable $delEntry is empty. */
else {
/* A variable $Message is initialised to hold an error
message, which indicates that records for deletion do not
exists in the database. */


$Message = '<BR><FONT Color="RED">Records selected for deletion
do not exists. </FONT>';


}
}
}
/* A variable named $hidMode is initialised to 'I'. This variable
hold the form status in terms of I - Insert, U - Update and D -
Delete. */


$hidMode = 'I';


/* A variable $count is initialised to hold zero. This is used in
while loops to generate unique checkbox names for the tabular
layout. */


$count = 0;


?>
<HTML>
<HEAD><TITLE><?=$title;?></TITLE>
<META Content="<?=$author;?>" Name="Author">
<META Content="" Name="Keywords">
<META Content="" Name="Description">
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<!-- JavaScript code-spec to trim a string and change string
case. -->
<SCRIPT Language="JavaScript" Src="mstrscript.js" Type="text/javascript">
</SCRIPT>
</HEAD>
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0"
TopMargin="0"><BR><BR>

```

14. INTEGRATION BETWEEN PHP AND ORACLE

```

<!-- JavaScript code-spec unique to the category master form. -->
<SCRIPT Language="JavaScript">
/* The chkBlanks() function verifies that the form field of
category name is not left empty. If the field is empty or
contains blank spaces, a message indicates to enter a valid
value. If the field is not empty, the data is submitted. */
function chkBlanks() {
/* A variable holding a reference to the form object
initialised in this page. */
    frm = document.frmMstrCat;
/* Extracting the contents of the category name field into a
variable named str. */
    var str = frm.txtCatName.value;
/* If the category name field is empty or contains blank
spaces. */
    if(str.trim() == "") {
/* Displaying a message indicates to enter a valid value.
*/
        alert('Please enter category name.');
/* Placing the form cursor on the category name field. */
        frm.txtCatName.focus();
/* Preventing the data in the form field from being
submitted. */
        return false;
    }
/* If the category name field holds a value. */
    else {
/* Calling the function vldtFrmFlds() to validate data
captured by the form. If the function returns TRUE. */
        if (vldtFrmFlds() == true) {
/* Allowing the data in the form field to be
submitted. */
            return true; }
/* If the function vldtFrmFlds() returns FALSE during
validation of data captured by the form. */
            else {
/* Preventing the data in the form field from being
submitted. */
                return false; }
            }
        }
    }

/* The function vldtFrmFlds() is called when the form data is
about to be submitted. This function verifies the presence of
changes in the form fields during the update mode. If changes
are not encountered, then a indicating the same is displayed
and any further processing gets terminated. */
function vldtFrmFlds() {

```

```

/* A variable holding a reference to the form object
initialised in this page. */
frm = document.frmMstrCat;
/* If the form is in the update mode. */
if (frm.hidMode.value == 'U') {
/* Initialising a variable named mNoChng to hold TRUE.
This value of this variable determines whether data in
the form field has been modified or not. */
    var mNoChng = true;
/* Checking the form objects individually for modified
information. If any changes have been made, the value of
the variable named mNoChng is set to hold FALSE. */
    if (frm.hidCatName.value != frm.txtCatName.value) {
        mNoChng = false; }
    if (frm.hidCatRmrk.value != frm.txtCatRmrk.value) {
        mNoChng = false; }
/* If data in the form fields have remained unchanged. */
if (mNoChng) {
    alert('Update failed.\n No changes have been made during
modification.');
/* Calling the JavaScript function to reset values in
the form fields. */
    setNewMode();
/* Placing the form cursor on the category name field.
*/
    frm.txtCatName.focus();
/* Preventing the data in the form field from being
submitted. */
    return false;
}
/* If data in the form fields have been changed. */
else {
/* Allowing the data in the form field to be
submitted. */
    return true; }
}
/* If the form is not in the update mode. */
else {
/* Allowing the data in the form field to be submitted. */
    return true; }
}

/* The function setNewMode() is called to prepare the form to
except a new set for data. This function is called directly
when the Clear button is clicked. */
function setNewMode() {
/* A variable holding a reference to the form object
initialised in this page. */
    frm = document.frmMstrCat;

```

14. INTEGRATION BETWEEN PHP AND ORACLE

```

/* Clearing form fields. */
frm.txtCatName.value = "";
frm.txtCatRmrk.value = "";
/* Clearing hidden variables used for comparison during in
the update operation. */
frm.hidCatID.value = "";
frm.hidCatName.value = "";
frm.hidCatRmrk.value = "";
/* Setting the form mode to insert. */
frm.hidMode.value = "I";
/* If the form contains a Delete button. */
if (frm.cmdDelete) {
    /* Enabling the Delete button. */
    frm.cmdDelete.disabled = false;
}

/* The function setEditMode() is called to prepare the form to
update data held in the tabular layout. This function is called
directly when a link in the tabular layout is clicked. The
function also sets the hidden field form mode to update and
populates the text fields. */
function setEditMode(id, name, rmrk) {
    /* A variable holding a reference to the form object
initialised in this page. */
    frm = document.frmMstrCat;
    /* If the form is not in the update mode. */
    if (frm.hidMode.value != 'U') {
        /* Assigning values passed as parameters to hidden
variables. These are used for comparison during in the
update operation. */
        frm.hidCatID.value = id;
        frm.hidCatName.value = name;
        frm.hidCatRmrk.value = rmrk;
        /* Populating the form fields. */
        frm.txtCatName.value = name;
        frm.txtCatRmrk.value = rmrk;
        /* Setting the form mode to update. */
        frm.hidMode.value = 'U';
        /* Disabling the Delete button. */
        frm.cmdDelete.disabled = true;
    }
    /* If the form is in the update mode. */
    else {
        alert('Update in progress.\n An entry has already been select for
modification.');
```

```

/* The function setDelMode() is called to prepare the form to
remove data from the tabular layout. This function is called
directly when the Delete button is clicked. */
function setDelMode() {
  /* Calling the JavaScript function to reset values in the
form fields. */
  setNewMode();
  /* Setting the form mode for deleting record(s). */
  document.frmMstrCat.hidMode.value = 'D';
  /* Calling the JavaScript function to populate the variable
holding a list of deleted records. The function
formDeleteValues() has been defined in the mstrscript.js
file. */
  formDeleteValues('hidDelRcrdLst', 'frmMstrCat');
}
</SCRIPT>
<!-- Outer Table Code Begins. -->
<TABLE Align="center" BgColor="<?=$gl_mstr_frst_bar_color;?>" Border="0"
CellPadding="0" CellSpacing="0" Name="TlbOuter" Width="90%"><TR>
  <TD Align="center" Border="1" BgColor="#C0C0C0"
Width="10%">Category</TD>
  <TD Align="center" BgColor="#FFFFFF" ColSpan="9"
Width="90%"><?=$Message;?></TD>
</TR>
<!-- Initialising a form object, which will submit data captured
on the form to the processing file. -->
<FORM Action="ctgry_mstr.php" Method="post" Name="frmMstrCat"
onSubmit="return chkBlanks();">
<!-- Declaring a hidden form field used to identify the current
(i.e. category master) page. -->
  <INPUT Name="hidPage" Type="hidden" Value="ctrgy_mstr">
<!-- Declaring hidden form fields required for data validation.
-->
  <INPUT Name="hidCatID" Size="2" Type="hidden" Value="">
  <INPUT Name="hidCatName" Size="2" Type="hidden" Value="">
  <INPUT Name="hidCatRmrk" Size="2" Type="hidden" Value="">
<!-- Declaring a hidden form field, for determining the form
mode. It will holds 'I' for Insert, 'U' for Update and 'D' for
Delete. It will hold 'I' when the page is rendered for the 1st
time. -->
  <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
<!-- Declaring a hidden form field used for identifying records
which have been selected for deletion. -->
  <INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
<TR Height="300" VAlign="top">
  <TD Align="center" Border="1" ColSpan="10"><BR>
  <!-- Form Table Code Begins. -->
    <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"
Name="TlbInner" Width="90%"><TR>

```

14. INTEGRATION BETWEEN PHP AND ORACLE


```

<?
/* A variable named $rowBgColor hold the hexadecimal colour
value. This variable is used decided the background colour of
rows for data in the tabular format. */
    $rowBgcolor = $gl_mstr_scnd_bar_color;

/* An iterative code performed as long as a single row of
category data is available. */
    do {
/* Incrementing the value of the variable used for assigning
unique names to check boxes display of each row of data. */
        $count = $count + 1;

/* If the colour code in for the current row matches the
value for the second row. */
        if ($rowBgcolor == $gl_mstr_scnd_bar_color) {
/* Colour code for the current row set to match the value
for the colour code for the first bar. */
            $rowBgcolor = $gl_mstr_frst_bar_color;        }

/* If the colour code in for the current row matches the
value for the first row. */
        else {
/* Colour code for the current row set to match the value
for the colour code for the second bar. */
            $rowBgcolor = $gl_mstr_scnd_bar_color;        }
?>

<TR BgColor="<?=$rowBgcolor;?>">
  <TD><INPUT Name="chk<?=( $count);?>" Type="checkbox"
    Value="<?=( $line['CATEGORY_ID']);?>"></TD>
  <TD><A
    HRef="JavaScript:setEditMode('<?=$line['CATEGORY_ID'];?>'
      , '<?=$line['CATEGORY_NAME'];?>',
      '<?=$line['CATEGORY_REMARKS'];?>')"><?
      echo($line['CATEGORY_NAME']); ?></A></TD>
  <TD><? echo($line['CATEGORY_REMARKS']); ?></TD>
</TR>

<?      } while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM)) ?>
</TABLE><BR>
<?    } ?>
  </TD>
</FORM>
</TR></TABLE>
<!-- Outer Table Code Ends. -->
</BODY>
</HTML>

```

14. INTEGRATION BETWEEN PHP AND ORACLE

As seen in the above code, the JavaScript functions for the HTML page rendered by **ctgry_mstr.php** is stored in the **mstrscript.js** file. The file will be downloaded by the client's web browser and will contain the following:

```

/*
Date :- 13/05/05
Author :- Hansel Colaco
Filename :- mstrscript.js
Purpose :- Defines Generic JavaScript functions for trimming
strings and changing string case.
*/
/*
Generating the function used for trimming string values. */
function strtrim() {
/* Returns the string passed as a parameter after removing
blank spaces, if any, at both the beginning and the end of the
parameter. */
return this.replace(/^\s+/, "").replace(/\s+$/, "");
}
/* Generating a global alias name for the function used for
trimming string values. */
String.prototype.trim = strtrim;

/* The chngCase () function changes the first alphabet to an upper
case character for the value held in the field passed as the
first parameter. The second parameter is a reference to the form
holding the field of the first parameter. This function is called
when a form cursor moves away from a field. */
function chngCase(pFld, pFrm) {
/* Using the eval () function to extract the value held in the
field passed as a parameter. */
var mFldVal = eval("document." + pFrm + "." + pFld + ".value;");
/* Removing extra spaces for the value extracted above and
restoring into the same variable. */
mFldVal = mFldVal.trim();
/* If the variable containing the extracted value is not empty.
*/
if (mFldVal != "") {
/* Changing the string Proper case. */
mFldVal = mFldVal.charAt(0).toUpperCase() + mFldVal.substr(1,
mFldVal.length);
}
/* Using the eval () function to display the new value in the
field passed as a parameter. */
eval("document." + pFrm + "." + pFld + ".value = mFldVal;");
}

```

```

/* The formDeleteValues() function generates a string of comma
separated identities of the records selected for deletion. If no
records are selected, a message indicates the same. The
parameters for this function are references to a hidden variable
(used for storing the generated string) and the form
respectively. It is called when the DELETE button is clicked. */
function formDeleteValues(pHid, pFrm) {
  /* Initialising variables for later use. */
  var selValues = "";
  var firstSelBox = "";
  /* Using the eval() function to create a variable to hold a
reference to the form object. */
  eval("frm = document." + pFrm);
  /* Iterating through every object on the form. */
  for (i=0; i<frm.elements.length; i++) {
    /* If the current object is a Checkbox. */
    if (frm.elements[i].type == "checkbox") {
      /* If the variable firstSelBox is empty. */
      if (firstSelBox == "") {
        /* Assigning the element number, of the current form
object, to the variable firstSelBox. */
        firstSelBox = i; }
        /* If the current Checkbox has been selected, (i.e.
checked). */
        if (frm.elements[i].checked == true) {
          /* Assigning the element number, of the current form
object, to the variable selValues. */
          selValues = selValues + frm.elements[i].value + ","; }
        }
      }
    }
  }
  /* If the variable selValues does not holds a value. */
  if (selValues.length < 1) {
    /* Displaying a message indicating to select a checkbox. */
    alert('Please choose records you wish to delete.');
    /* Using the eval() function to place the form cursor on the
first checkbox. */
    eval("document." + pFrm + ".elements[" + firstSelBox + "].focus();");}
  /* If the variable selValues holds a value. */
  else {
    /* Removing the last character (i.e. a comma) for the value
held in the variable selValues. */
    selValues = selValues.substring(0, selValues.length-1);
    /* Using eval() function to assign a value to the form's
hidden variable. */
    eval("document." + pFrm + "." + pHid + ".value = " + selValues + "");
    frm.submit();
  }
}

```

14. INTEGRATION BETWEEN PHP AND ORACLE

Hands On Exercises

- Based on the Hands on exercise developed in Chapter 12 previously, modify the form to capture information for Book Dimensions. After modifications, the PHP program should be able to store captured data into a table named **DmsnMstr** held under the Oracle database.
 - The contents held in the table is displayed in a tabular layout, placed immediately after the data entry form. Refer diagram 14.10
 - Besides storing new entries, the program should allow modification of previously stored data
 - Similarly, the PHP program should allow (single / multiple) deletion of entries held in the Oracle table
 - The page (containing the data entry form and the tabular layout) generated, will display a message indicating the previous operation performed by the PHP program

The screenshot shows a web browser window with the title 'Dimensions'. At the top, there is a red message: '<Message from the previous operation >'. Below this is the 'Publishers Management System' header. The main content is a form titled 'Specification for Book Dimensions' with input fields for 'Dimension Name', 'Height', and 'Width'. There are radio buttons for 'Units' with 'Centimeters' and 'Inches' options. Below the form are 'Save' and 'Clear' buttons. At the bottom, there is a table with the following data:

| Delete | DimensionName | Height | Width | Units |
|--------------------------|--------------------------|--------|-------|-------------|
| <input type="checkbox"/> | A4 | 11.69 | 8.27 | Inches |
| <input type="checkbox"/> | A5 | 8.27 | 5.83 | Inches |
| <input type="checkbox"/> | B5 (ISO) | 9.84 | 6.93 | Inches |
| <input type="checkbox"/> | B5 (JIS) | 10.12 | 7.17 | Inches |
| <input type="checkbox"/> | Crown | 19 | 12 | Centimeters |
| <input type="checkbox"/> | Foolscap | 43 | 34 | Centimeters |
| <input type="checkbox"/> | Royal | 63 | 51 | Centimeters |

Diagram 14.10: Output for DmsnMstr.php.