Oracle® Database Express Edition

2 Day Plus PHP Developer Guide

10*g* Release 2 (10.2)

B25317-01

February 2006

ORACLE®

Oracle Database Express Edition 2 Day Plus PHP Developer Guide, 10*g* Release 2 (10.2)

B25317-01

Primary Author: Simon Watt

Contributors: Christopher Jones, Simon Law, Glenn Stokol

Contents

Preface

Oracle Database Express Edition 2 Day Plus PHP Developer Guide introduces developers to the use of PHP to access Oracle Database Express Edition.

This preface contains these topics:

Audience

Oracle Database Express Edition 2 Day Plus PHP Developer Guide is an introduction to application development using Zend Core for Oracle and Oracle Database Express Edition.

This document assumes a basic understanding of SQL, PL/SQL, and PHP.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Related Documents

For more information, see these Oracle resources:

- *Oracle Database Express Edition Licensing Information*
- *Oracle Database Express Edition Installation Guide for Linux*
- *Oracle Database Express Edition Installation Guide for Microsoft Windows*
- *Oracle Database Express Edition Getting Started Guide*
- *Oracle Database Express Edition 2 Day DBA Guide*
- *Oracle Database Express Edition 2 Day Developer Guide*
- *Oracle HTML DB 2 Day Developer*
- *Oracle Database Express Edition ISV Embedding Guide*
- *Oracle Database SQL Reference*
- *Oracle Database PL/SQL User's Guide and Reference*
- *SQL*Plus User's Guide and Reference*
- *Oracle Database Globalization Support Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Introducing PHP with Oracle Database XE

PHP is a popular scripting language that can be embedded in HTML, which makes it particularly useful for Web development. Zend Core for Oracle enables application development using PHP.

Oracle Database Express Edition (Oracle Database XE) is a free relational database that you can use to store, use, and modify data.

This chapter has the following topics:

Zend Core for Oracle

Zend Core for Oracle, developed in partnership with Zend Technologies, provides a stable, high performance, easy-to-install, and supported PHP development and production environment that is fully integrated with Oracle Database Express Edition.

Purpose

This guide is a tutorial that shows you how to use Zend Core for Oracle to connect to Oracle Database XE, and demonstrates how to use PHP to access and modify data.

Overview of the Sample Application

This document guides you through the development of a simple Human Resources (HR) application for a fictitious company called AnyCo Corp.

The application manages departmental and employee information stored in the `DEPARTMENTS` and `EMPLOYEES` tables in the HR schema provided with Oracle Database XE. See *Oracle Database Sample Schemas* for information about this schema.

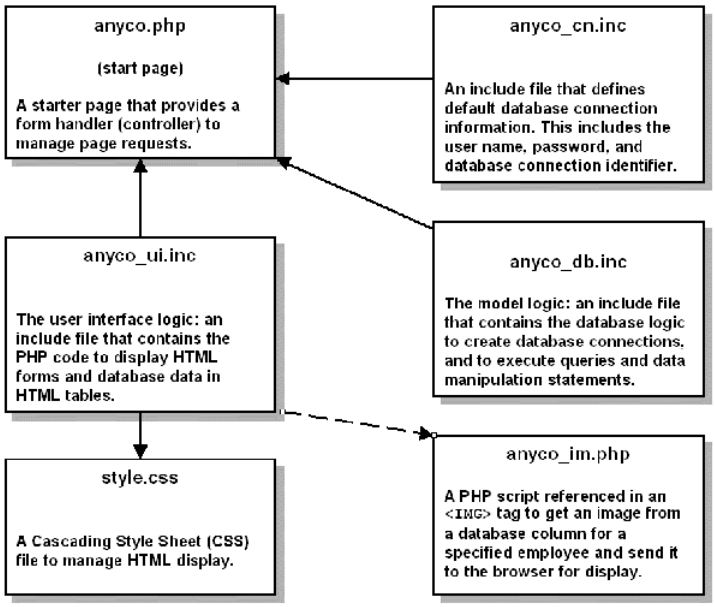
The complete sample application:

- Establishes a connection to the database using the PHP OCI8 extension
- Queries the database for department and employee data
- Displays and navigates through the data
- Shows how to insert, update, and delete employee records

- Handles data exceptions
- Uploads and displays employee photographs

Figure 1-1 shows the relationship between the files developed for this application.

Figure 1–1 Functionality in the Sample PHP Application



The sample application files are:

anyco.php This file has the main logic for the AnyCo application. It contains control logic to decide which page is displayed. It manages session data for navigation. The functionality in `anyco_cn.inc`, `anyco_db.inc`, and `anyco_ui.inc` is used by it.

anyco_ui.inc This file contains the functions used to present data and forms in an HTML page.

anyco_cn.inc This file contains definitions for database connection information, the database user name, password, and database connection identifier.

anyco_db.inc This file contains the database logic to create database connections, execute queries, and execute data manipulation statements.

anyco_im.php This file contains logic to retrieve an image from a database column and send it to a browser for display as a JPEG image.

style.css This file contains Cascading Style Sheet (CSS) definitions for various HTML tags generated by the application. It manages the look and feel of the application.

Files with the suffix `.inc` are PHP code files included in other PHP files.

Files with the suffix `.php` can be loaded in a browser.

You can create and edit the PHP application source files in a text editor or any tool that supports PHP development.

The code for each chapter builds on the files completed in the previous chapter.

Resources

The following Oracle Technology Network Web sites provide additional information you may find useful.

- Oracle Database Express Edition Developer Center at

  http://www.oracle.com/technology/xe

- PHP Developer Center at

  http://www.oracle.com/technology/tech/php/index.html

- Zend Core for Oracle Developer Center at

  http://www.oracle.com/technology/tech/php/zendcore/index.html

- Oracle Database Express Edition documentation at

  http://www.oracle.com/technology/xe/documentation

- The Oracle Database Documentation Library at

  http://www.oracle.com/technology/documentation

2

Getting Started

This chapter explains how to install and test your Oracle Database Express Edition (Oracle Database XE) and PHP environment. It has the following topics:

What You Need

To install your Oracle Database XE and PHP environment, you need:

- Oracle Database 10*g* Express Edition
- Apache Web server 1.3.x or later
- Zend Core for Oracle
- A text editor for editing PHP code

Installing Oracle Database Express Edition

Oracle Database Express Edition is available from the Oracle Technology Network at

`http://www.oracle.com/technology/xe`

Install Oracle Database XE following the instructions in *Oracle Database Express Edition Installation Guide for Linux* or *Oracle Database Express Edition Installation Guide for Microsoft Windows*.

See Also:

- `http://www.oracle.com/technology/xe/forum`

  for the Oracle Database XE discussion Forum
- `http://www.oracle.com/technology/xe/documentation`

  for the Oracle Database XE documentation

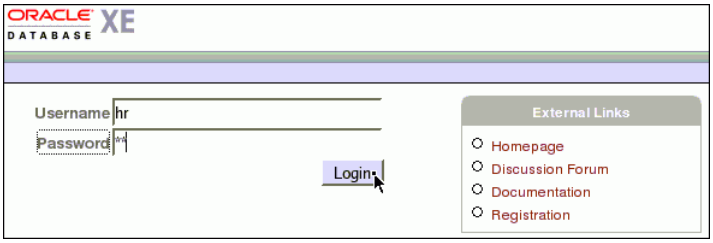
Testing the Oracle Database XE Installation

Perform the following steps to test that you can connect to Oracle Database XE.

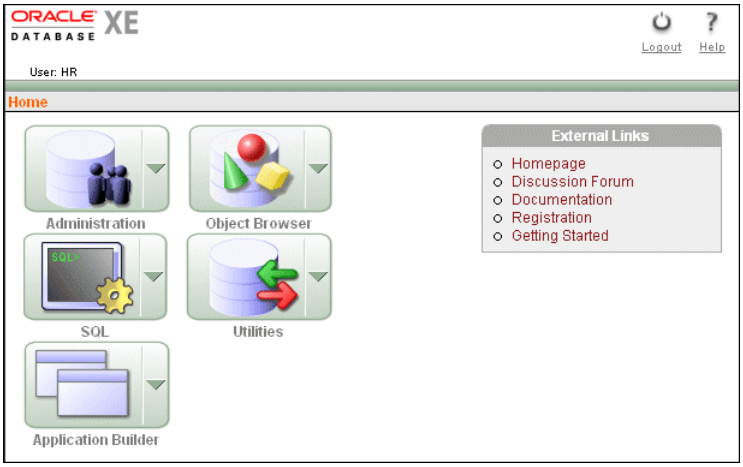
1. To test that Oracle Database XE is accessible, connect to the database using the Oracle Database XE home page:
2. In a browser, enter the URL for your Oracle Database Express Edition home page:

   ```
   http://localhost:8080/htmldb
   ```

3. In the Oracle Database XE Login page, enter `hr` in the Username and Password fields. Click **Login**:



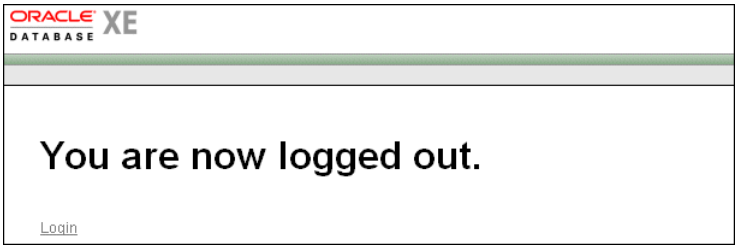
   The Oracle Database XE Home page appears:



   If the message, Invalid Login Credentials appears, you may need to unlock the HR user. See Unlocking the HR User following.

4. Click the **Logout** link to terminate the HTMLDB session.

5. The Logout Confirmation page appears:



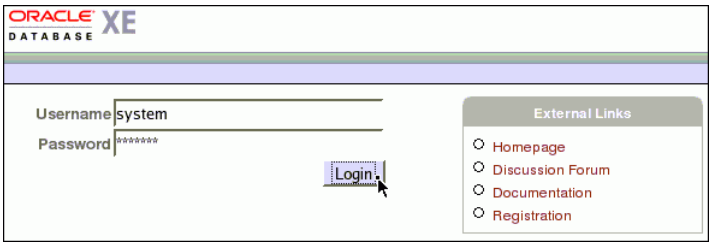
Unlocking the HR User

The PHP application connects to the database as the HR user. You may need to unlock the `HR` account as a user with DBA privileges. To unlock the `HR` user:

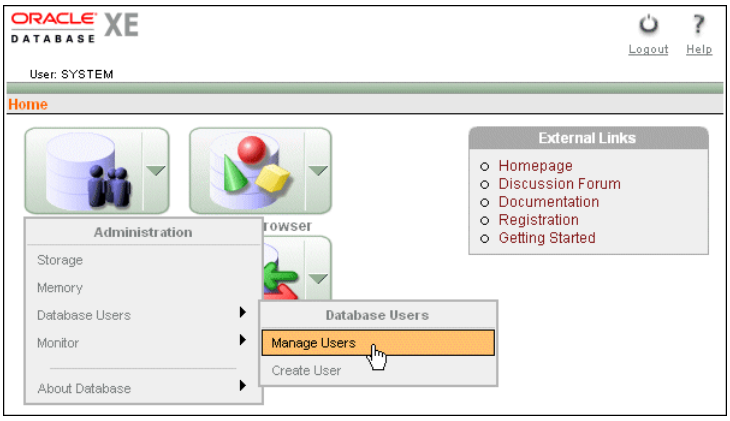
1. In a browser, enter the URL for your Oracle Database Express Edition home page:

   `http://localhost:8080/htmldb`

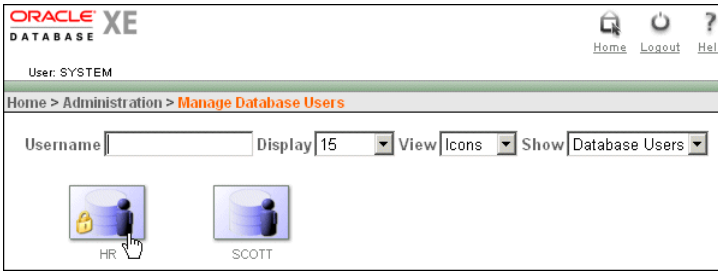
2. At the login screen, in the Username field enter `system`, and in the Password field enter `manager` (or the password you entered at the prompt during configuration of Oracle Database XE). Click **Login**.



3. In the Home page, click the arrow on the **Administration** icon, move the mouse over **Database Users**, and click **Manage Users**:



4. In the **Manage Users** page, click the **HR** user icon:



5. In the **Manage Database User** page, select Unlocked from the Account Status list and then click the **Alter User** button. The message User Altered appears below the navigation breadcrumb toward the top of the page.

6. Click the **Logout** link to terminate the HTMLDB session.

For further information about unlocking an Oracle Database account, see Chapter 6, "Managing Users and Security," in the *Oracle Database Express Edition 2 Day DBA* guide.

Installing Apache

You can download Apache for Windows or Linux from:

http://httpd.apache.org/

Apache is normally a standard part of a Linux environment.

Install Apache following the instructions available from the same site.

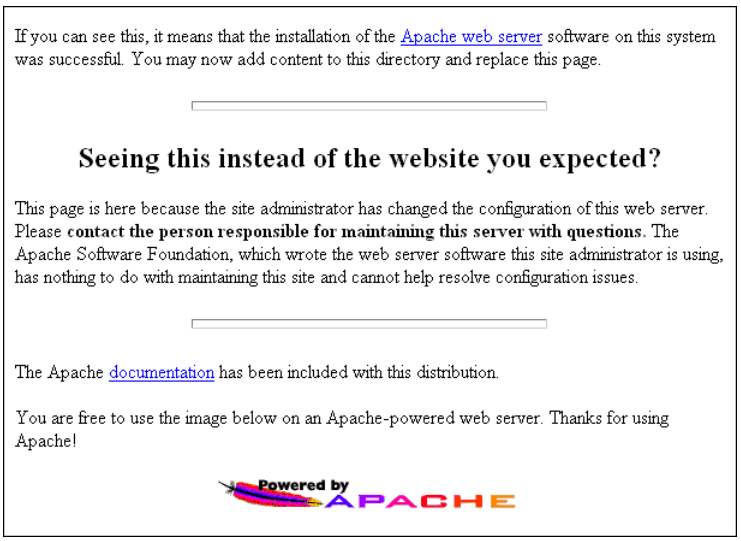
Testing the Apache Installation on Windows

To test the Apache Web server installation:

1. Start your Web browser on the host on which you installed Apache.
2. Enter the following URL:

   ```
   http://localhost/
   ```

   Your browser should display a page similar to the following:



   If this page does not appear check your Apache configuration. Common problems are that Apache is not running, or that it is listening on a non-default port.

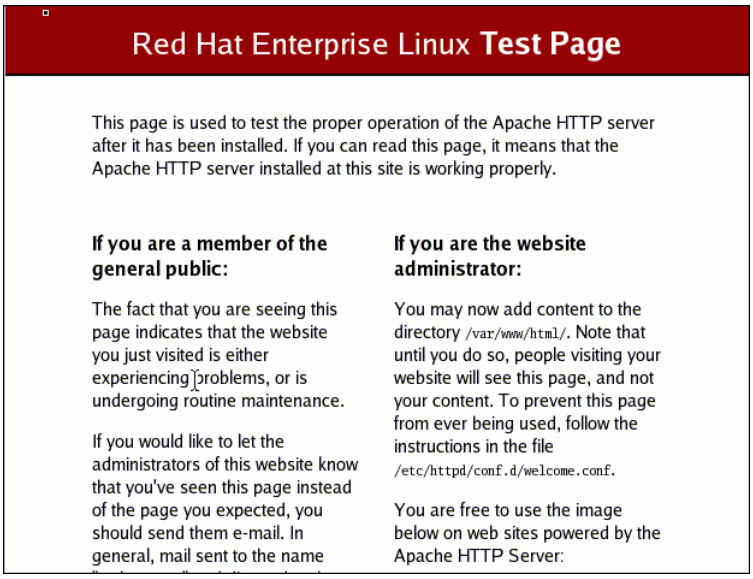
Testing the Apache Installation on Linux

To test the Apache Web server installation:

1. Start your Web browser on the host you installed Apache on, and enter the following URL:

   ```
   http://localhost/
   ```

   Your browser should display a page similar to the following:



If this page does not appear check your Apache configuration. Common problems are that Apache is not running, or that it is listening on a non-default port.

2. In the default Apache Web server configuration file, set up a public virtual directory as `public_html` for accessing your PHP files. Use your preferred editor to open the Apache configuration file `/etc/httpd/conf/httpd.conf` (the directory may be different in your installation of Linux) and remove the "#" character at the start of the following line:

   ```
   #UserDir public_html
   ```

   This enables the browser to make an HTTP request using a registered user on the system and to serve files from the `$HOME/public_html` directory of the user. For example:

   ```
   http://localhost/~user/
   ```

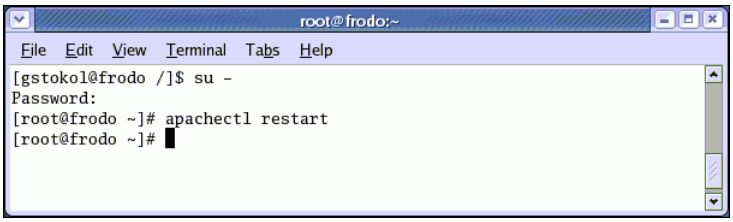
   In this example, your Apache `httpd.conf` file should contain the following lines:

   ```
   <IfModule mod_userdir.c>
       #
       # UserDir is disabled by default since it can confirm the presence
       # of a username on the system (depending on home directory
       # permissions).
       #
       #UserDir disable

       #
       # To enable requests to /~user/ to serve the user's public_html
       # directory, remove the "UserDir disable" line above, and uncomment
       # the following line instead:
       #
       UserDir public_html
   </IfModule>
   ```

3. In a command window, to use the new Apache configuration file, restart Apache by entering the following commands:

   ```
   su -
   Password: <enter your su (root) password>
   apachectl restart
   ```



If the Apache Web server does not start, check the error log files to determine the cause. It may be a configuration error.

4. In the command window, log in as a normal (not root) user and create a `public_html` subdirectory in the `$HOME` directory with the following command:

```
mkdir $HOME/public_html
```

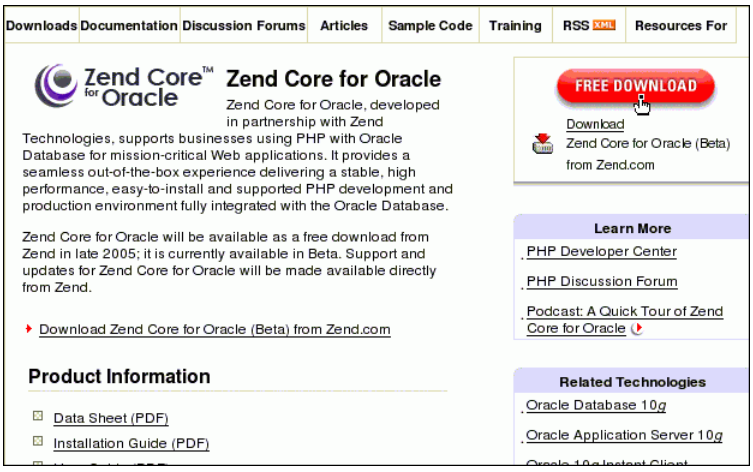
Installing Zend Core for Oracle

To obtain Zend Core for Oracle for Windows or Linux:

1. Enter the following URL in your Web browser

   `http://www.oracle.com/technology/tech/php/zendcore/index.html`

2. To the right of the "Zend Core for Oracle" Web page, click the **Free Download** button:



3. Save the downloaded file in a temporary directory, such as `c:\tmp` on Windows, or `\tmp` on Linux.

Installing Zend Core for Oracle on Windows

This section describes how to install Zend Core for Oracle on Windows.

This tutorial is specific to PHP in Zend Core for Oracle.

For detailed setup information for Zend Core for Oracle, see the Installation Guide under Product Information on the Zend Core for Oracle Web page at

`http://www.oracle.com/technology/tech/php/zendcore/index.html`

This procedure assumes you downloaded the Zend Core for Oracle software to `c:\tmp`. If not, in step 1 you must `cd` to the directory containing the downloaded software, and not to `c:\tmp`.

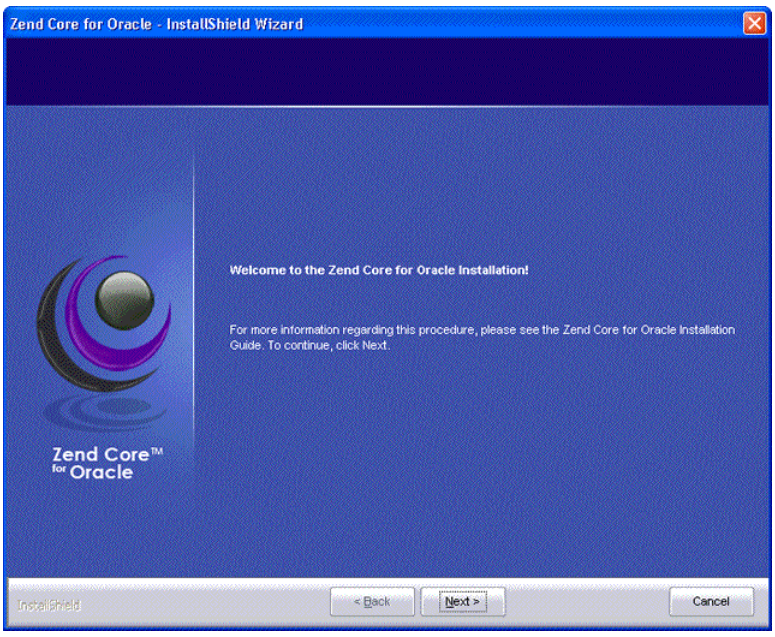
The file name and extraction directory are based on the current version. Throughout this procedure, ensure you use the directory name for the version you are installing.

You must be the administrator user to install Zend Core for Oracle. To install Zend Core for Oracle, perform the following steps:

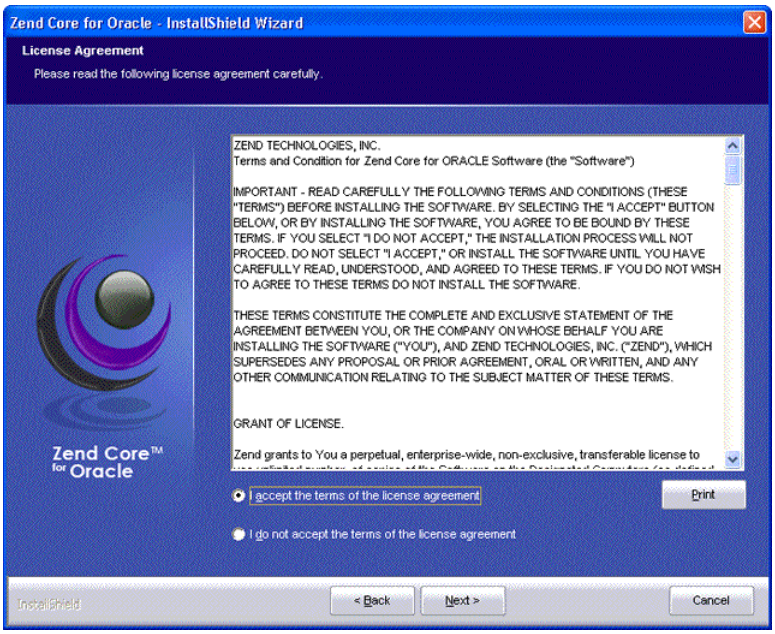
1. In Windows Explorer, navigate to the directory where you downloaded the Zend Core for Oracle software.
2. To start the Zend Core for Oracle installation process, double click the `.exe` file.

   Review the README and installation documentation distributed with Zend Core for Oracle.

   Use the tab or arrow keys, or use your mouse to navigate between input fields and buttons in the Zend installer. Press Enter or click with the mouse to select a button.
3. In the initial Zend Core for Oracle Installation page, click **Next**.

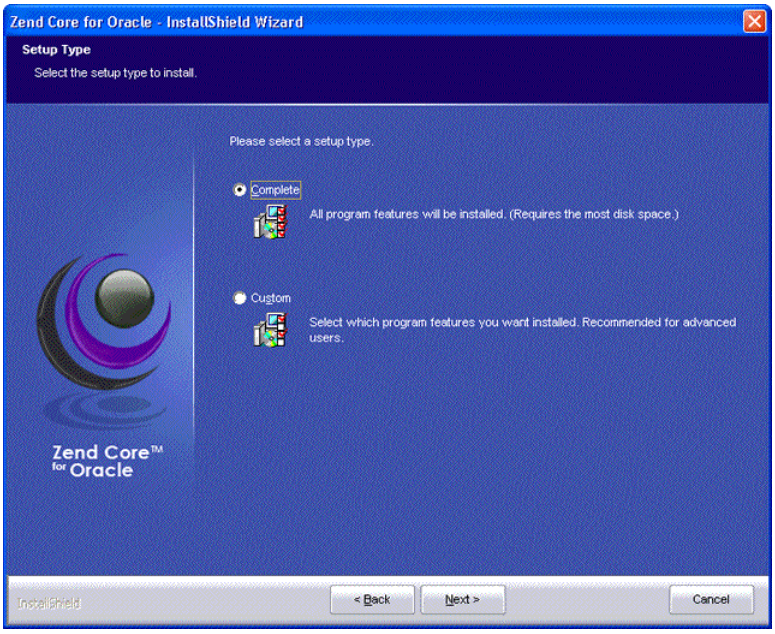


4. In the Zend Core for Oracle License Agreement page, read the license agreement. To continue with the installation, select **I accept the terms of the license agreement** and then click **Next**.



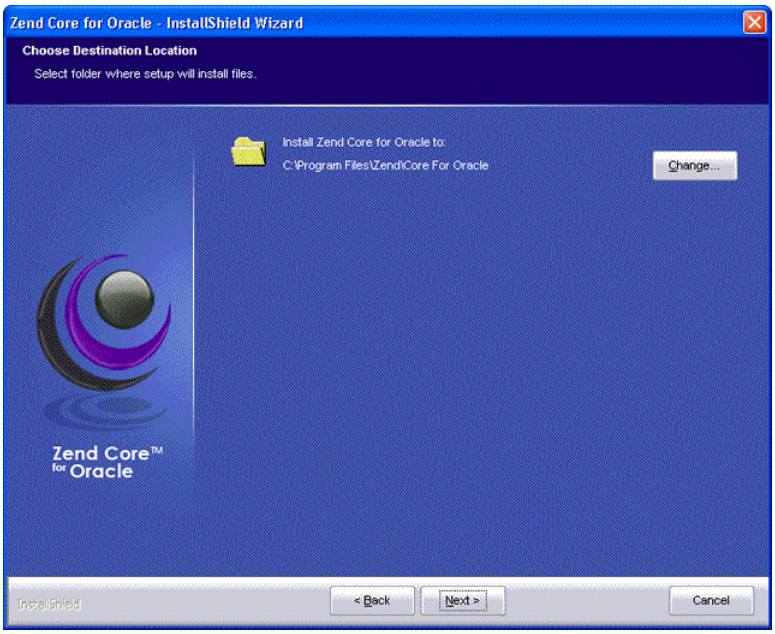
Copyright, 2005, Zend Technologies Ltd.

5. You are prompted to select the type of installation you want. Select **Complete** and then click **Next**.

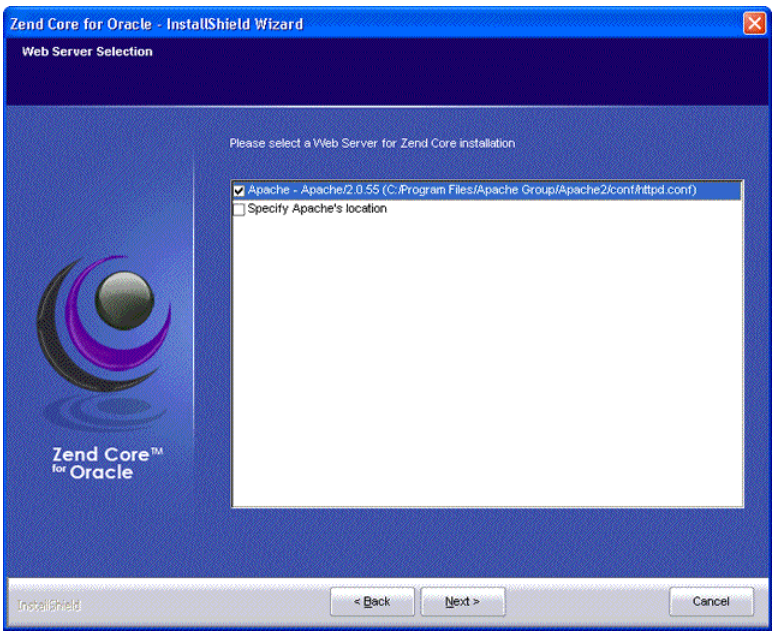


Copyright, 2005, Zend Technologies Ltd.

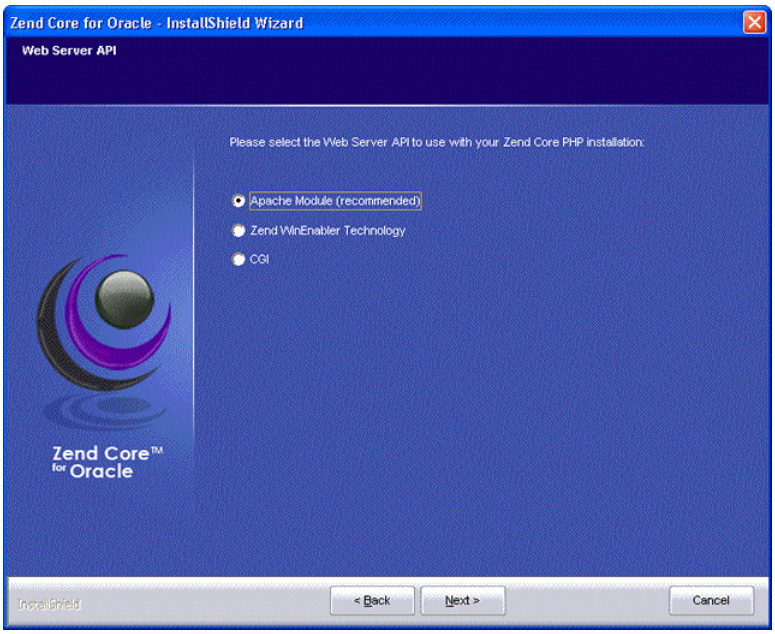
6. When prompted to specify the location for installing Zend Core for Oracle, accept the default (or enter your preferred location), and click **Next**.



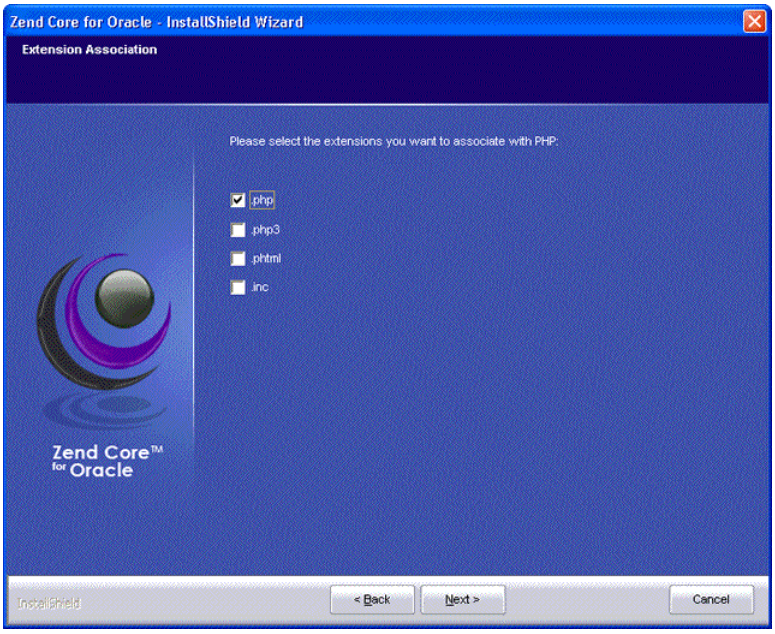
7. The next page prompts you to select the Web server for Zend Core installation. Accept the Apache installation and click **Next**.



8. You are then prompted to select the Web Server API to use. Select **Apache Module** and then click **Next**.

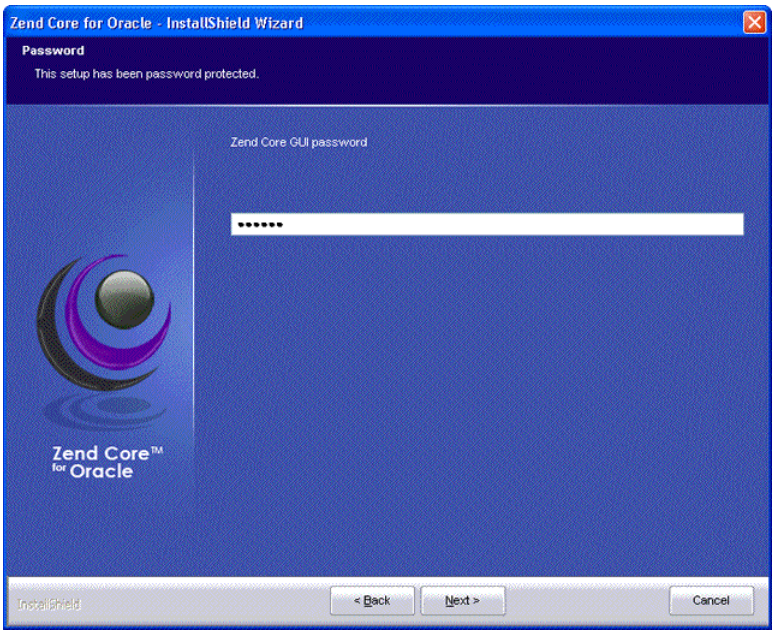


9. When prompted to select extensions to associate with your Zend Core for Oracle installation, select **php** and click **Next**.

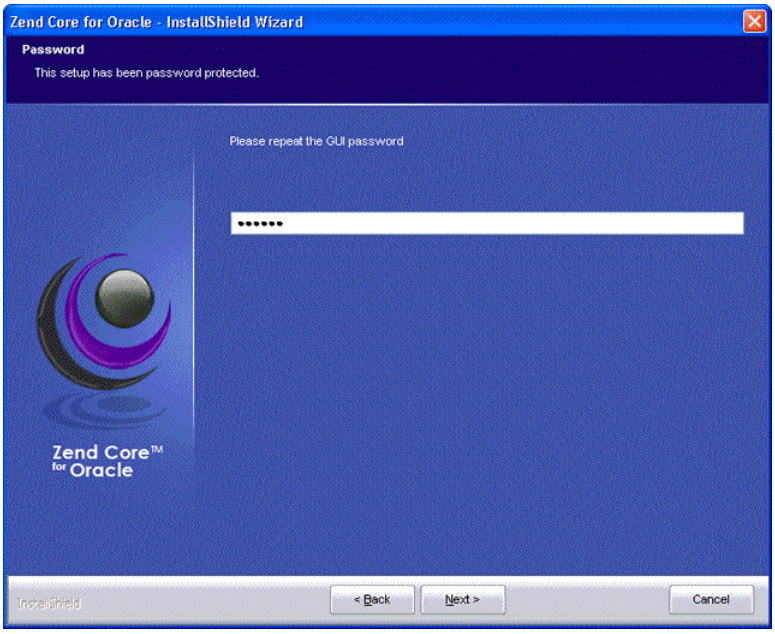


10. You are now prompted to enter a Zend Core GUI password. This password enables you to log in to the Zend Core Console to configure directives or property values.

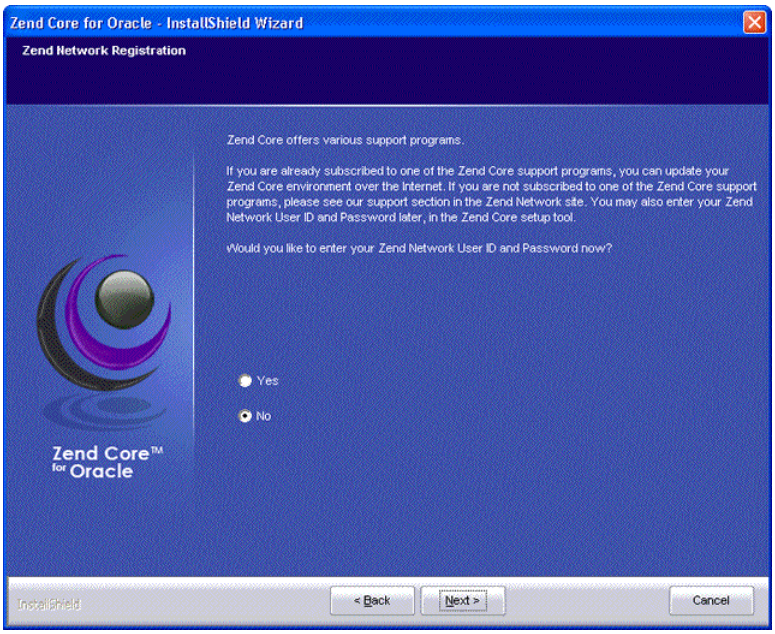
    Enter the password you want to use when accessing the Zend Core Console, and click **Next**.



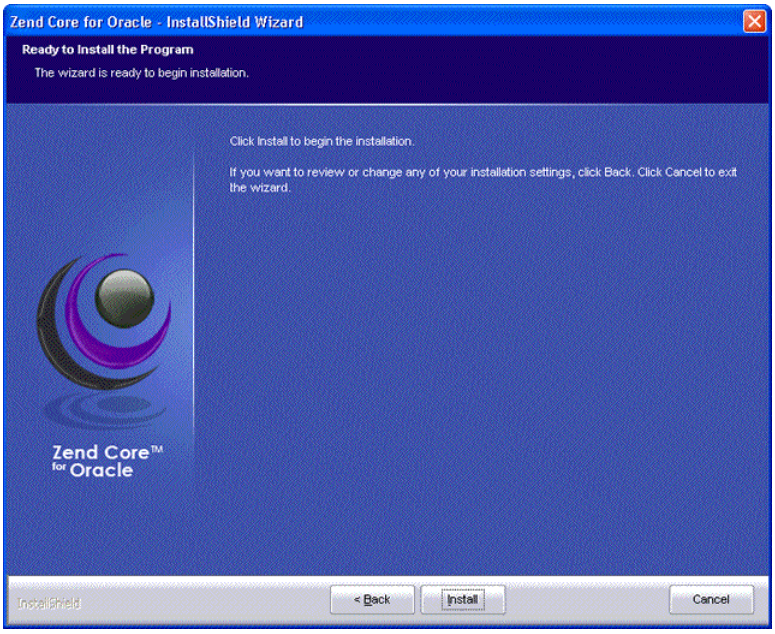
11. You are prompted to reenter your Zend Core GUI password.



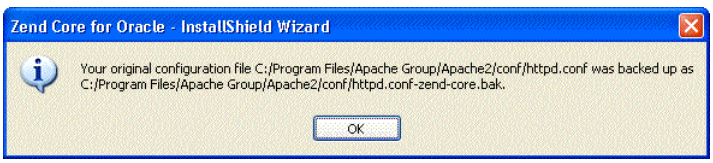
12. You may optionally enter your Zend network user ID and password to be able to use the Zend Core Console to track when updates to Zend Core and PHP components are available. If you have not registered, or do not want to track updates, select **No** and then click **Next**.



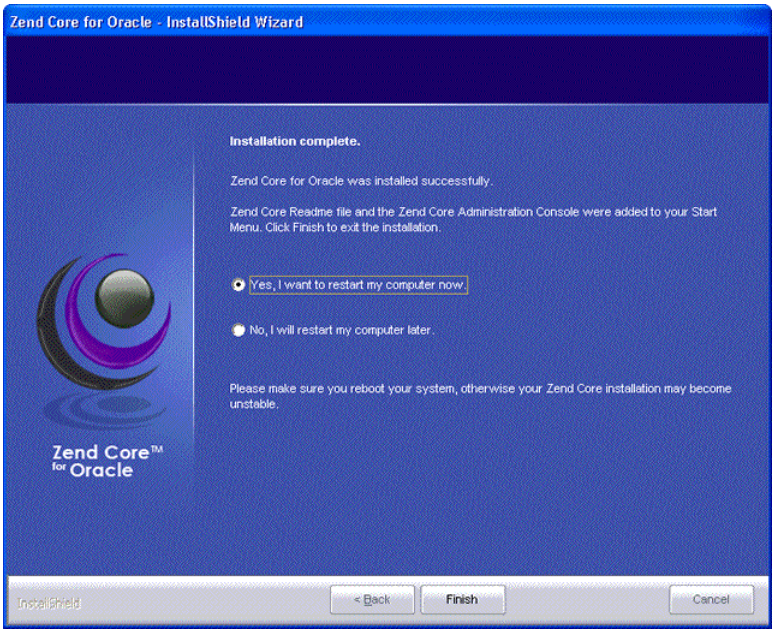
13. The installation wizard is now ready to begin installing Zend Core for Oracle on your computer. Click **Install** to start the installation wizard.



14. You are notified that the Apache configuration file has been modified, and where the original Apache configuration file has been stored. Click **OK** to continue.



15. When the installation completes, you are prompted to restart your computer. Select **Yes, I want to restart my computer now** and then click **Finish**.



The Zend Core for Oracle installation is now complete.

Installing Zend Core for Oracle on Linux

This section describes how to install Zend Core for Oracle on Linux.

This tutorial is specific to PHP in Zend Core for Oracle.

For detailed setup information for Zend Core for Oracle, see the Installation Guide under Product Information on the Zend Core for Oracle Web page at

http://www.oracle.com/technology/tech/php/zendcore/index.html

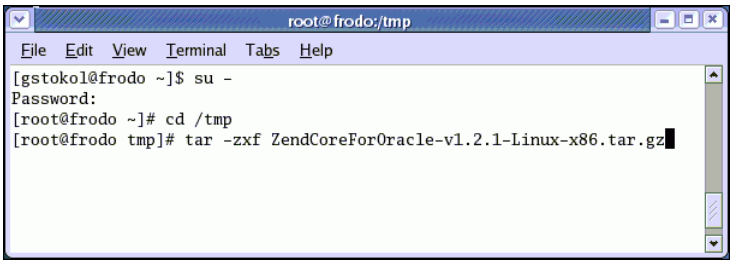
This procedure assumes you downloaded the Zend Core for Oracle software to `/tmp`. If not, in step 1 you must `cd` to the directory containing the downloaded software, and not to `/tmp`.

The file name and extraction directory are based on the current version. Throughout this procedure, ensure you use the directory name for the version you are installing.

You must be the root user to install Zend Core for Oracle. To install Zend Core for Oracle, perform the following steps:

1. Enter the following commands in a command window to extract the contents of the downloaded Zend Core for Oracle software:

```
su -
Password: <enter the root password>
cd /tmp
tar -zxf ZendCoreForOracle-v1.2.1-Linux-x86.tar.gz
```

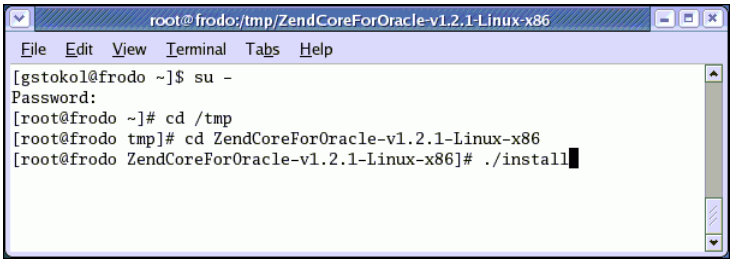


By default, files are extracted to a subdirectory called `ZendCoreForOracle-v1.2.1-Linux-x86`.

Review the README and installation documentation distributed with Zend Core for Oracle.

2. To start the Zend Core for Oracle installation process, enter the following commands:

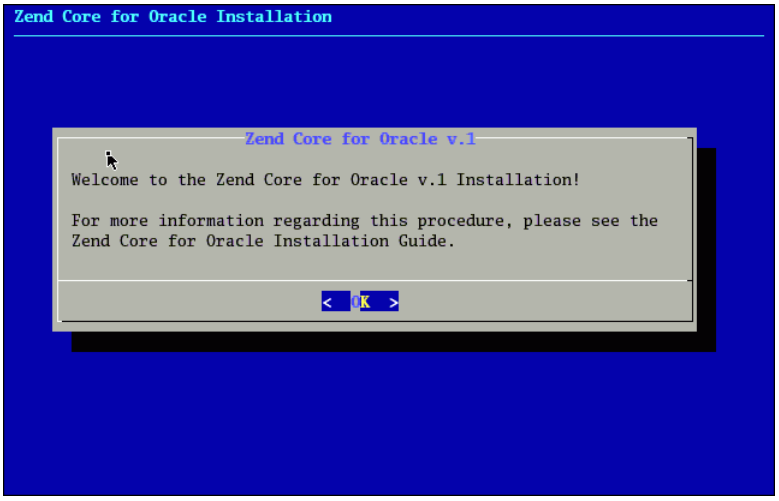
```
cd ZendCoreForOracle-v1.2.1-Linux-x86
./install
```



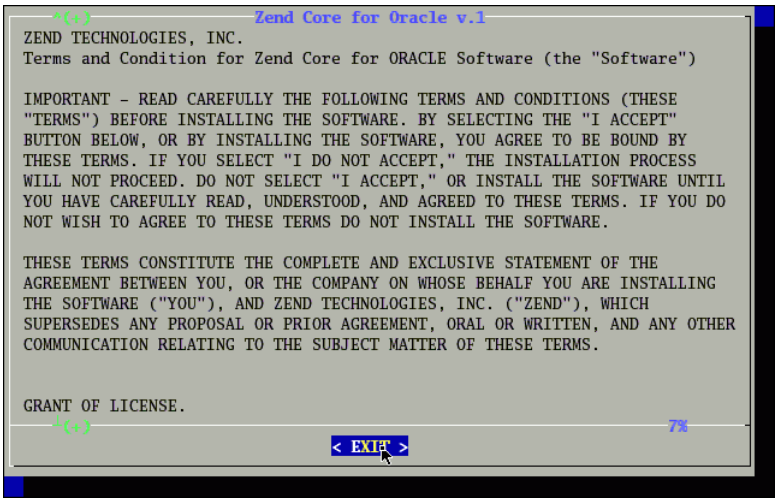
The `install` command must be executed with root user privileges. After you enter the `./install` command, the installation process begins, as documented in subsequent steps.

Use the tab or arrow keys, or use your mouse to navigate between input fields and buttons in the Zend installer. Press Enter or click with the mouse to select a button.

3. In the initial Zend Core for Oracle Installation page, click **OK**.

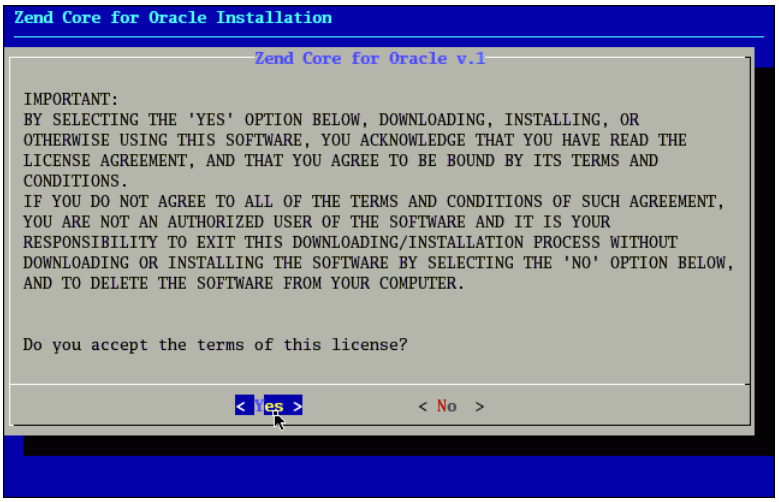


4. In the Zend Core for Oracle V.1 page, read the license agreement. To continue with the installation, click **Exit**.



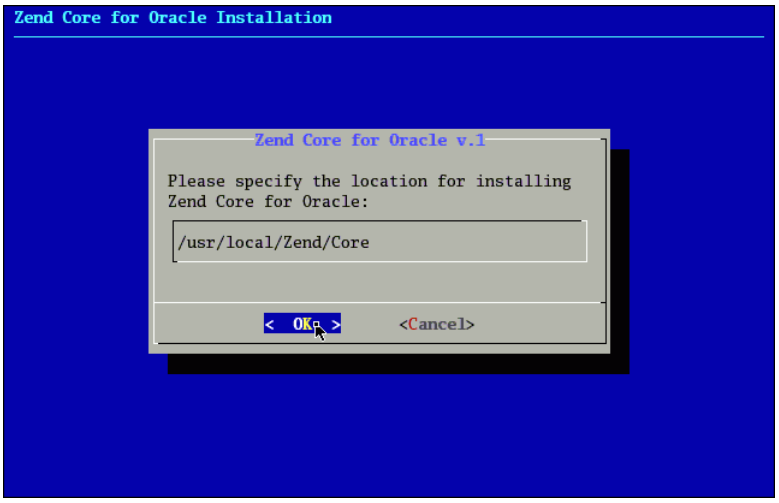
Copyright, 2005, Zend Technologies Ltd.

5. When prompted to accept the terms of the license, click **Yes**.

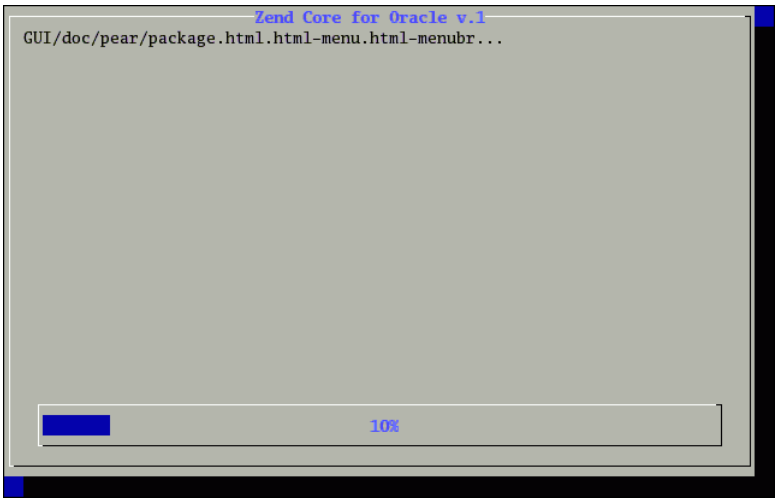


Copyright, 2005, Zend Technologies Ltd.

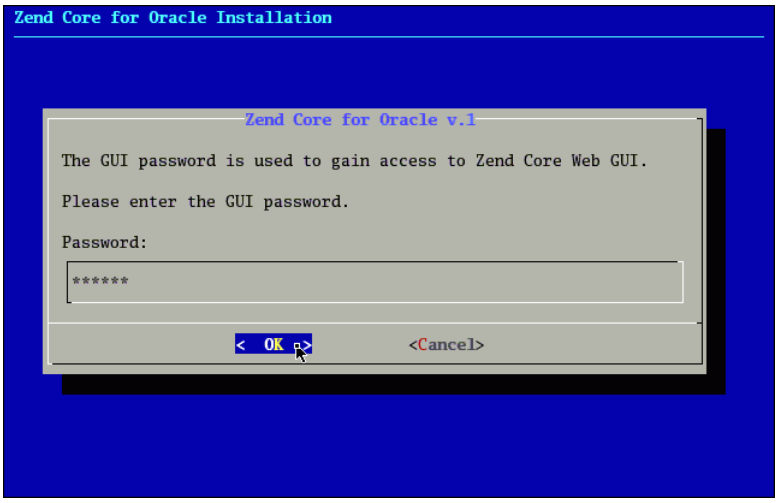
6. When prompted to specify the location for installing Zend Core for Oracle, accept the default (or enter your preferred location), and click **OK**.



The installer begins extracting the files required for the installation. The following progress screen is visible during the installation process:

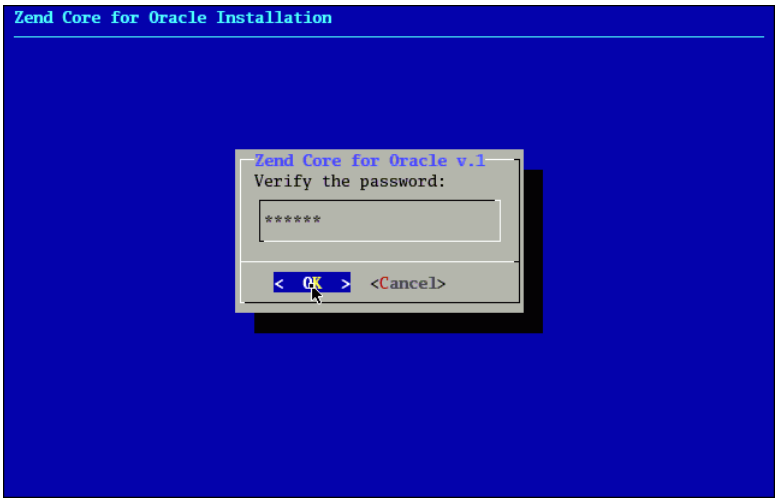


7. When the progress window indicates that all the software has been installed, you are prompted to "Please enter the GUI password." In the Password field, enter the password you want to use when accessing the Zend Core Console, and click **OK**.

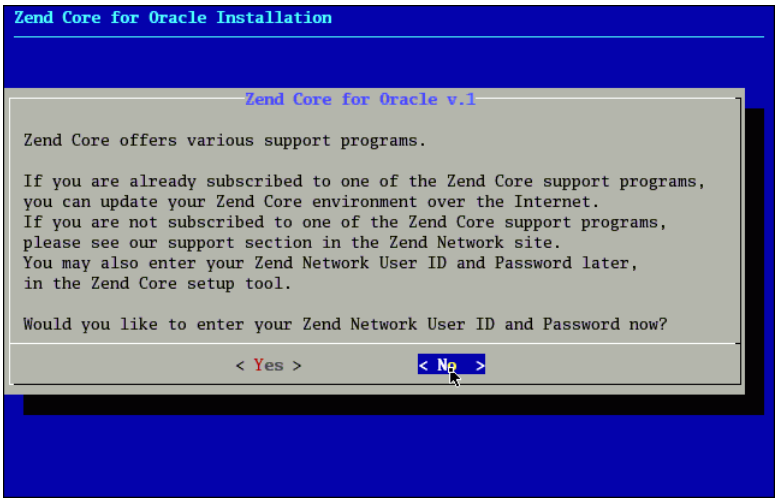


The password specified here allows you to log in to the Zend Core for Oracle administration Web pages to enable configuration of Zend Core for Oracle engine directives or property values.

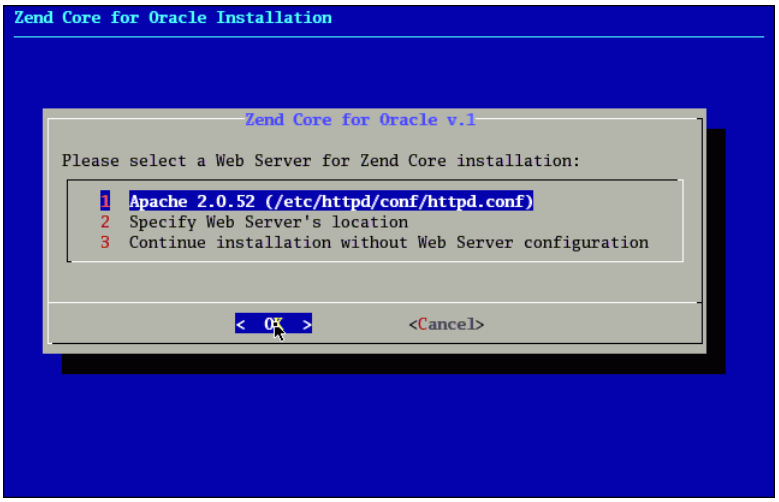
8. When prompted to "Verify the password," enter the same password as specified in Step 7 and click **OK**.



9. In the Zend Core support page, you may optionally enter your Zend network user ID and password to be able to use the Zend Core Console to track when updates to Zend Core and PHP components are available. If you have not registered, or do not want to track updates, click **No**.

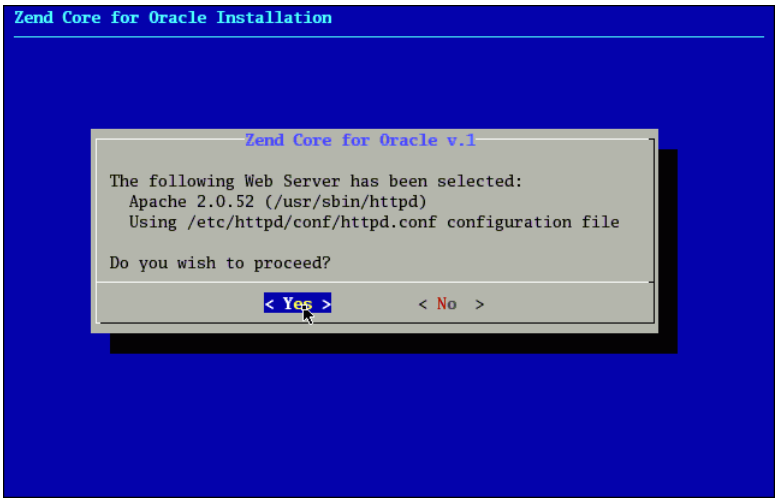


10. The next page prompts you to select the Web server for Zend Core installation. Select the default Apache installed with Linux. Click **OK**.

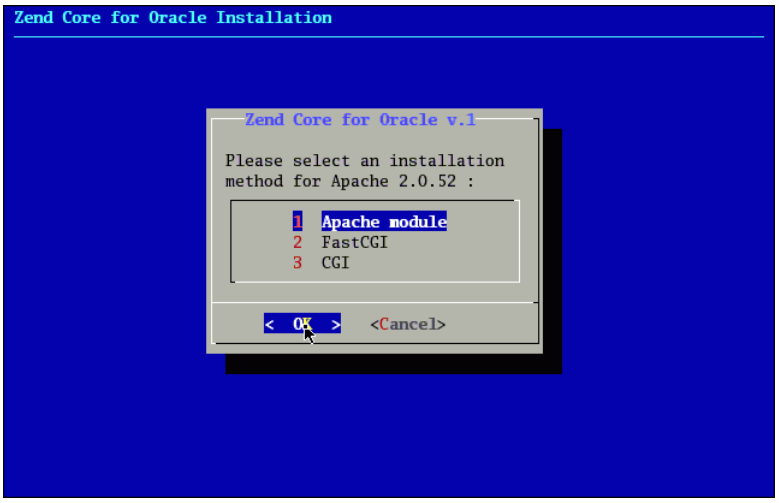


If you choose, you can install Zend Core for Oracle with another supported Web server that is installed on your system.

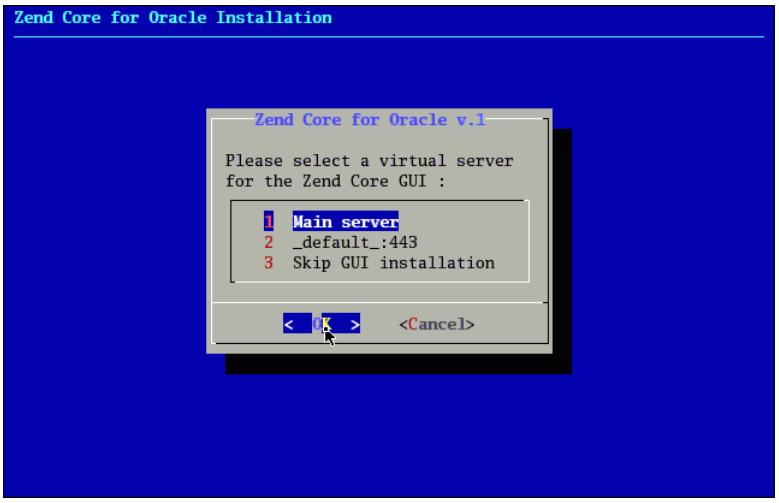
11. In the page confirming your Web server selection, at the "Do you wish to proceed?" prompt, click **Yes**.



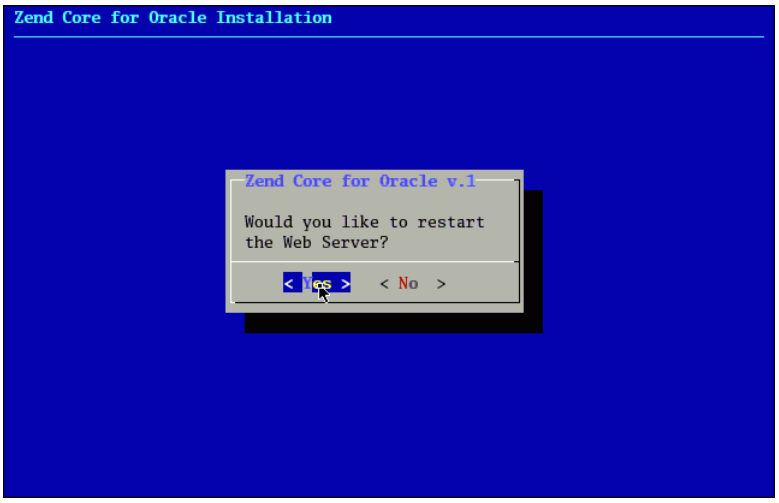
12. In the next installation page, you are prompted to "Please select an installation method for Apache 2.0.52." Select **Apache module** as the method, and click **OK**.



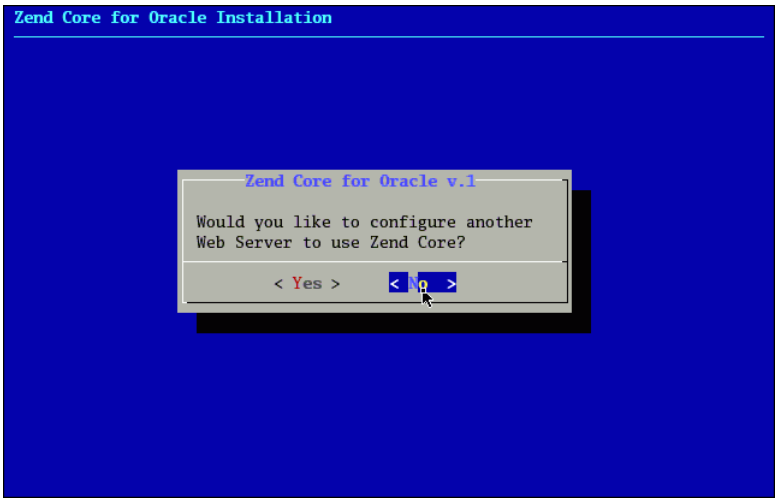
13. In the next installation page, when you are prompted to "Please select a virtual server for the Zend Core GUI," select **Main Server**, and click **OK**.



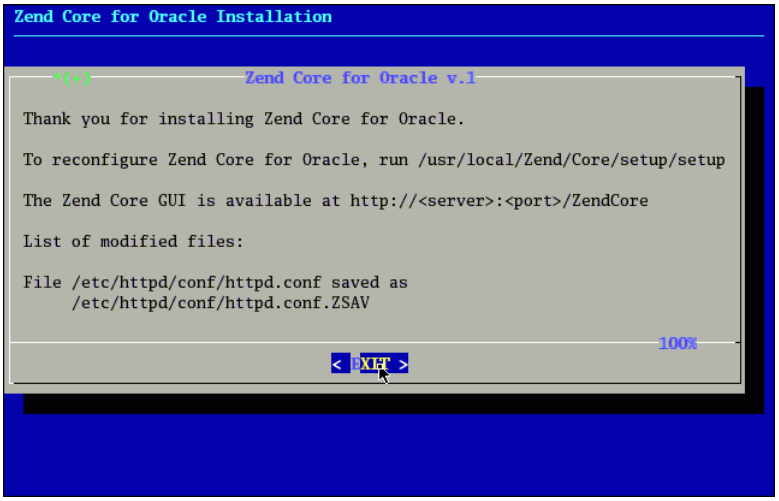
14. In the next installation page, at the "Would you like to restart the Web Server" prompt, click **Yes**.



15. When prompted "Would you like to configure another Web Server to use Zend Core," click **No**.



16. The final installation page (containing "Thank you for installing Zend Core for Oracle") lists useful configuration commands and a Web page for the administration of the Zend Core engine. Take note of the information and click **EXIT**.



17. When the Zend Core installation pages are terminated, a message is displayed in your command window.

```
*******************************************************************************

Thank you for installing Zend Core for Oracle.

To reconfigure Zend Core for Oracle, run /usr/local/Zend/Core/setup/setup

The Zend Core GUI is available at http://<server>:<port>/ZendCore

List of modified files:

File /etc/httpd/conf/httpd.conf saved as
     /etc/httpd/conf/httpd.conf.ZSAV

*******************************************************************************




[root@frodo ZendCoreForOracle-v1.2.1-Linux-x86]#
```


The Zend Core for Oracle installation is now complete.

Configuring Zend Core for Oracle

In this section, you configure environment variables and Zend Core directives that control default error reporting in Web pages.

1. Enter the following URL in a Web browser to access the Zend Core Administration page:

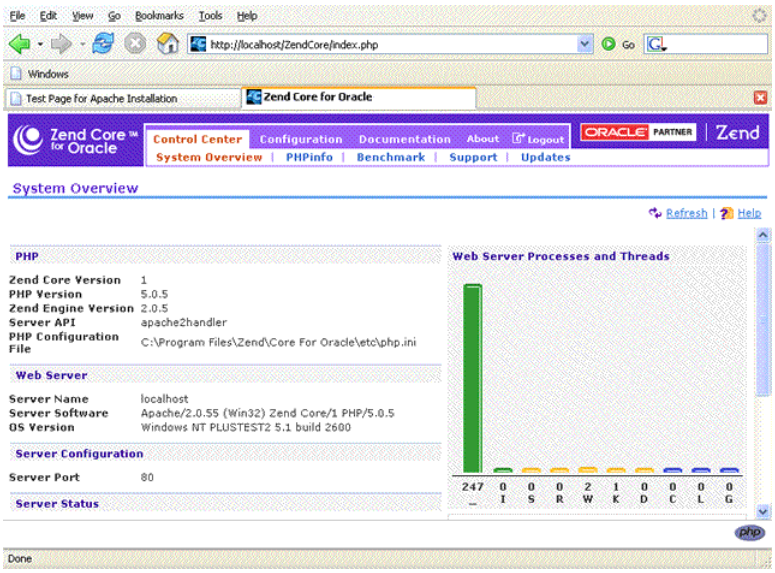
   ```
   http://localhost/ZendCore/
   ```

   The Zend Core for Oracle Welcome page is displayed.

2. Enter the GUI password that you provided during Zend Core for Oracle installation in the Password field. Click the login >>> icon.

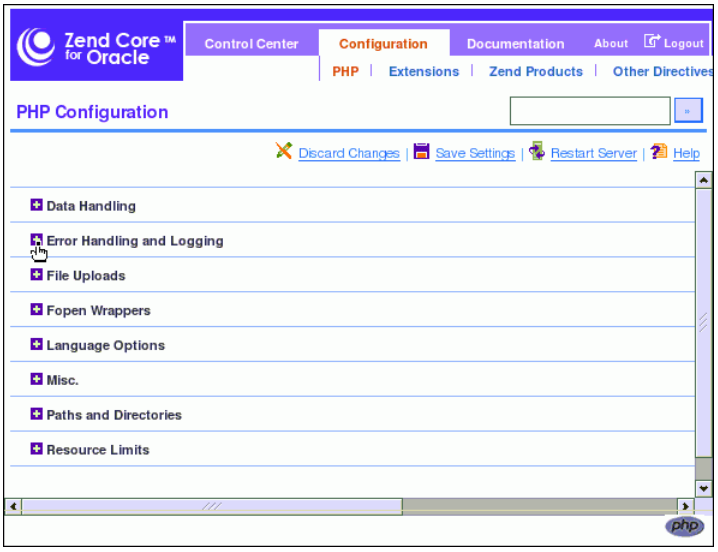
The Control Center System Overview page is displayed.

3. Click the **Configuration** tab to display the configuration options.



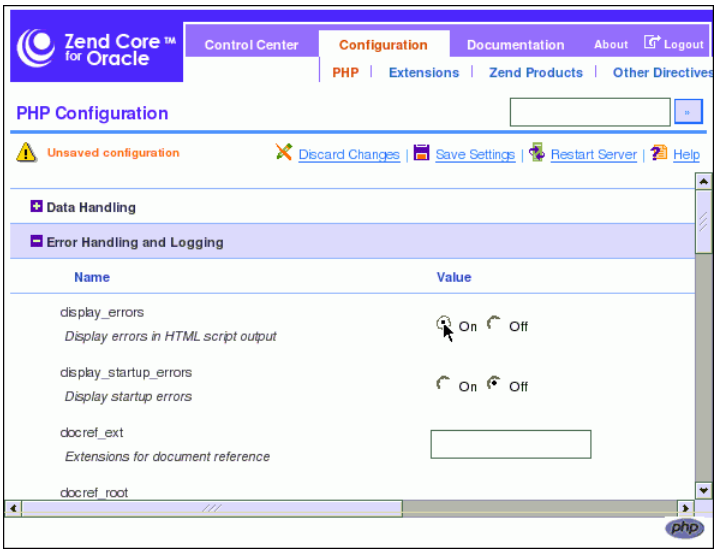
Copyright, 2005, Zend Technologies Ltd.

4. Click the + icon to expand the Error Handling and Logging configuration entry.



Copyright, 2005, Zend Technologies Ltd.

5. Set the `display_errors` directive to `On` to enable the display of errors in the HTML script output during development.



Because there are unsaved changes, the "Unsaved configuration" message appears under the page header.

6. Click **Save Settings** to save the configuration change.

   Because you have made configuration changes, you must restart the Apache Web server. Under the page header notice the "Please Restart Apache" message reminding you to do so.

7. Click **Restart Server** to restart the Apache server.

   The PHP Configuration page is refreshed after the Apache server restarts. Should the page not refresh after a short period of time, manually restart Apache and reload the PHP Configuration page.

8. Click **Logout** to exit the Zend Core for Oracle Administration page.

   Now that you have completed the basic configuration changes, proceed to the next section to test the Zend Core for Oracle installation.

Testing the Zend Core for Oracle Installation

To test the Zend Core for Oracle installation:

1. Create a subdirectory called `chap2`. To create a directory for your application files, and to change to the newly created directory, enter the following commands in a command window:

   On Windows:

   ```
   mkdir c:\program files\Apache Group\Apache2\htdocs\chap2
   cd c:\program files\Apache Group\Apache2\htdocs\chap2
   ```

   On Linux:

   ```
   mkdir $HOME/public_html/chap2
   cd $HOME/public_html/chap2
   ```

   If you create files in a different location, you must change the steps for file editing and execution to match your working directory name and URL.

```
[gstokol@frodo ~]$ mkdir $HOME/public_html/chap2
[gstokol@frodo ~]$ cd $HOME/public_html/chap2
[gstokol@frodo chap2]$
```

2. Create a file called `hello.php` that contains the following HTML text:

```
<?php
  echo "Hello, world!";
?>
```

3. Open a Web browser and enter the following URL in your browser:

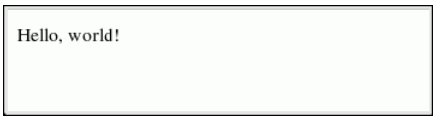
   On Windows:

   `http://localhost/chap2/hello.php`

   On Linux:

   `http://localhost/~<username>/chap2/hello.php`

   The line "Hello, world!" appears in the browser.



3

Getting Connected

In this chapter, you create HR application files that implement PHP functions to connect and disconnect to the Oracle Database. You also develop a PHP function that enables you to execute a query to validate that a database connection has been successfully established.

This chapter also guides you through the creation and modification of PHP files that call a function to produce the header and footer for the Departments page, where the footer section of the page includes a date and time.

This chapter has the following topics:

> **Note:** For simplicity, the user name and password are written into this sample application code. For applications that will be deployed, coding the user name and password strings directly into your application source code is not recommended. Oracle recommends that you use a more secure technique, such as implementing a dialog that prompts the user for the user name and password.
>
> See *Oracle Database Security Guide* and the documentation for your development environment for details on security features and practices.

Building the Departments Page

In this section, you will create the functions and styles for the first screen of your application.

Follow these steps to build the Departments page:

1. To create a directory for your application files, and to change to the newly created directory, enter the following commands in a command window:

   On Windows:

   ```
   mkdir c:\program files\Apache Group\Apache2\htdocs\chap3
   cd c:\program files\Apache Group\Apache2\htdocs\chap3
   ```

   On Linux:

   ```
   mkdir $HOME/public_html/chap3
   ```

```
cd $HOME/public_html/chap3
```

If you create files in a different location, you must change the steps for file editing and execution to match your working directory name and URL.

2. To start developing your application user interface, use your preferred text editor to create a file called `anyco_ui.inc` that contains the two functions `ui_print_header()` and `ui_print_footer()` with their parameters to enable your application Web pages to have consistent header and footer sections:

```
<?php

function ui_print_header($title)
{
  $title = htmlentities($title);
  echo <<<END
  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
     "http://www.w3.org/TR/html4/strict.dtd">
  <html>
  <head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=ISO-8859-1">
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Any Co.: $title</title>
  </head>
  <body>
  <h1>$title</h1>
END;
}

function ui_print_footer($date)
{
  $date = htmlentities($date);
  echo <<<END
  <div class="footer">
    <div class="date">$date</div>
    <div class="company">Any Co.</div>
  </div>
END;
}

?>
```

- This application design uses PHP function definitions to enable modular reusable code.
- The functions in `anyco_ui.inc` use a PHP language construct called a "here document." This enables you to place any amount of HTML formatted text between the following two lines:

  ```
  echo <<<END
  END;
  ```

  Do not put leading spaces in the `END;` line. If you do, the rest of the document will be treated as part of the text to be printed.
- Any PHP parameters appearing inside the body of a "here document" are replaced with their values, for example, the `$title` or `$date` parameters.

- The PHP function `htmlentities()` is used to prevent user-supplied text from accidentally containing HTML markup and affecting the output formatting.

3. The PHP file uses a Cascading Style Sheet (CSS) file called `style.css` to specify the presentation style in HTML in the browser.

   Create a `style.css` file in the `chap3` directory with the following CSS text:

```
body
{ background: #CCCCFF;
  color:      #000000;
  font-family: Arial, sans-serif; }

h1
{ border-bottom: solid #334B66 4px;
  font-size: 160%; }

table
{ padding: 5px; }

td
{ border: solid #000000 1px;
  text-align: left;
  padding: 5px; }

th
{ text-align: left;
  padding: 5px; }

.footer
{ border-top: solid #334B66 4px;
  font-size: 90%; }

.company
{ padding-top: 5px;
  float: right; }

.date
{ padding-top: 5px;
  float: left; }
```

4. To call the user interface functions, create the `anyco.php` file with the following text:

```
<?php

require('anyco_ui.inc');

ui_print_header('Departments');
ui_print_footer(date('Y-m-d H:i:s'));

?>
```

   The `require()` PHP command is used to include `anyco_ui.inc`. The new functions can be called to produce HTML output.

5. To test the `anyco.php` file, enter the following URL in your browser:

   On Windows:

```
http://localhost/chap3/anyco.php
```

On Linux:

```
http://localhost/~<username>/chap3/anyco.php
```

The resulting Web page is similar to the following:

The date and time appear in the page footer section.

Connecting to the Database

In this section, you will add a database connection to your Departments screen so that you can display Department data.

Follow these steps to add a database connection to your application.

To form a database connection, you use the `oci_connect()` function with three string parameters:

```
$conn = oci_connect($username, $password, $db)
```

The first and second parameters are the database user name and password, respectively. The third parameter is the database connection identifier. The `oci_connect()` function returns a connection resource needed for other OCI8 calls; it returns FALSE if an error occurs. The connection identifier returned is stored in a variable called `$conn`.

1. Edit the `anyco.php` file to add a database connection with the following parameter values:
   - Username is `hr`.
   - Password for this example is `hr`. Remember to use the actual password of your `HR` user.
   - Oracle connection identifier is `//localhost/XE`.
2. Edit the `anyco.php` file to validate that the `oci_connect()` call returns a usable database connection, write a `do_query()` function that accepts two parameters: the database connection identifier, obtained from the call to `oci_connect()`, and a query string to select all the rows from the `DEPARTMENTS` table.
3. Edit the `anyco.php` file to prepare the query for execution, add an `oci_parse()` call. The `oci_parse()` function has two parameters, the connection identifier and the query string. It returns a statement identifier needed to execute the query and fetch the resulting data rows. It returns `FALSE` if an error occurs.
4. Edit the `anyco.php` file to execute the query, add a call to the `oci_execute()` function. The oci_execute() function executes the statement associated with the statement identifier provided in its first parameter. The second parameter specifies the execution mode. `OCI_DEFAULT` is used to indicate that you do not want statements to be committed automatically. The default execution mode is `OCI_COMMIT_ON_SUCCESS`. The `oci_execute()` function returns `TRUE` on success; otherwise it returns `FALSE`.

5. Edit the `anyco.php` file to fetch all the rows for the query executed, add a `while` loop and a call to the `oci_fetch_array()` function. The `oci_fetch_array()` function returns the next row from the result data; it returns `FALSE` if there are no more rows. The second parameter of the `oci_fetch_array()` function, `OCI_RETURN_NULLS`, indicates that `NULL` database fields will be returned as PHP NULL values.

   Each row of data is returned as a numeric array of column values. The code uses a PHP `foreach` construct to loop through the array and print each column value in an HTML table cell, inside a table row element. If the item value is `NULL` then a nonbreaking space is printed; otherwise the item value is printed.

   After the edits in Steps 1 to 5, the `anyco.php` file becomes:

```
<?php // File: anyco.php

require('anyco_ui.inc');

// Create a database connection
$conn = oci_connect('hr', 'hr', '//localhost/XE');

ui_print_header('Departments');
do_query($conn, 'SELECT * FROM DEPARTMENTS');
ui_print_footer(date('Y-m-d H:i:s'));

// Execute query and display results
function do_query($conn, $query)
{
  $stid = oci_parse($conn, $query);
  $r = oci_execute($stid, OCI_DEFAULT);

  print '<table border="1">';
  while ($row = oci_fetch_array($stid, OCI_RETURN_NULLS)) {
    print '<tr>';
    foreach ($row as $item) {
      print '<td>'.
            ($item ? htmlentities($item) : ' ').'</td>';
    }
    print '</tr>';
  }
  print '</table>';
}

?>
```

6. To test the changes made to `anyco.php`, save the modified `anyco.php` file. In a browser window, enter the following URL:

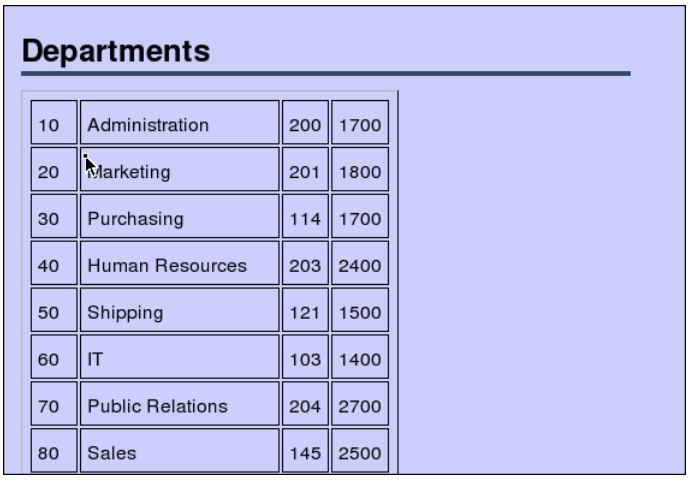
   On Windows:

   `http://localhost/chap3/anyco.php`

   On Linux:

   `http://localhost/~<username>/chap3/anyco.php`

   The page returned in the browser window should resemble the following page:



If you want to query the EMPLOYEES data, you can optionally change the query in the `do_query()` function call to:

```
do_query($conn, 'SELECT * FROM EMPLOYEES');
```

If You Have Connection Problems

Check that the username, password and connection string are valid. The connect string `'//localhost/XE'` uses the Oracle Easy Connect syntax. If you are using an Oracle Net `tnsnames.ora` file to specify the database you want to connect to, then use the network alias as the third parameter to the `oci_connect()` function.

If you are not seeing errors, check that you have set the `display_errors` directive `ON` and the `error_reporting` directive is set to `E_ALL|E_STRICT`.

If you have a PHP code problem and are not using a debugger, you can examine variables using the PHP `var_dump()` function. For example:

```
print '<pre>';
var_dump($r);
print '</pre>';
```

Other Ways to Connect

In some applications, using a persistent connection improves performance by removing the need to reconnect each time the script is called. Depending on your Apache configuration, this may cause a number of database connections to remain open simultaneously. You must balance the connection performance benefits against the overhead on the database server.

Persistent connections are made with the OCI8 `oci_pconnect()` function. Several settings in the PHP initialization file enable you to control the lifetime of persistent connections. Some settings include:

oci8.max_persistent - This controls the number of persistent connections per process.

oci8.persistent_timeout - This specifies the time (in seconds) that a process maintains an idle persistent connection.

oci8.ping_interval - This specifies the time (in seconds) that must pass before a persistent connection is "pinged" to check its validity.

For more information, see the PHP reference manual at

http://www.php.net/manual/en/ref.oci8.php

Disconnecting from the Database

The PHP engine automatically closes the database connection at the end of the script unless a persistent connection was made. If you want to explicitly close a database connection, you can call the `oci_close()` OCI function with the connection identifier returned by the `oci_connect()` call. For example:

```
<?php

$conn = oci_connect('hr', 'hr', '//localhost/XE');
...
oci_close($conn);

...

?>
```

4

Querying Data

In this chapter, you extend the Anyco HR application from Chapter 3 by adding information to the Departments page. You also implement the functionality to query, insert, update, and delete employee records in a specific department.

This chapter has the following topics:

Centralizing the Database Application Logic

In this section, you will modify your application code by moving the database access logic into separate files for inclusion in the PHP application.

1. Copy the files that you completed in Chapter 3 to a new `chap4` directory, and change to the newly created directory:

   On Windows:

   ```
   mkdir c:\program files\Apache Group\Apache2\htdocs\chap4
   cd c:\program files\Apache Group\Apache2\htdocs\chap4
   copy ..\chap3\* .
   ```

   On Linux:

   ```
   mkdir $HOME/public_html/chap4
   cd $HOME/public_html/chap4
   cp ../chap3/* .
   ```

2. Using your preferred editor, create a file called `anyco_cn.inc` that defines named constants for the database connection information. This file enables you to change connection information in one place.

   ```
   <?php // File: anyco_cn.inc

   define('ORA_CON_UN', 'hr');             // User name
   define('ORA_CON_PW', 'hr');             // Password
   define('ORA_CON_DB', '//localhost/XE'); // Connection identifier

   ?>
   ```

   For simplicity, the user name and password are written into this sample application code. For applications that will be deployed, coding the user name and

password strings directly into your application source code is not recommended. Oracle recommends that you use a more secure technique, such as implementing a dialog that prompts the user for the user name and password.

See *Oracle Database Security Guide* and the documentation for your development environment for details on security features and practices.

3. Create a file called `anyco_db.inc` that declares functions for creating a database connection, executing a query, and disconnecting from the database. Use the following logic, which includes some error handling that is managed by calling an additional function called `db_error ()`:

```
<?php  // File: anyco_db.inc

function db_connect()
{
  // use constants defined in anyco_cn.inc
  $conn = oci_connect(ORA_CON_UN, ORA_CON_PW, ORA_CON_DB);
  if (!$conn) {
    db_error(null, __FILE__, __LINE__);
  }
  return($conn);
}

function db_do_query($conn, $statement)
{
  $stid = oci_parse($conn, $statement);
  if (!$stid) {
    db_error($conn, __FILE__, __LINE__);
  }

  $r = oci_execute($stid, OCI_DEFAULT);
  if (!$r) {
    db_error($stid, __FILE__, __LINE__);
  }
 $r = oci_fetch_all($stid, $results, null, null,
                    OCI_FETCHSTATEMENT_BY_ROW);
  return($results);
}

// $r is the resource containing the error.
// Pass no argument or false for connection errors

function db_error($r = false, $file, $line)
{
  $err =  $r ? oci_error($r) : oci_error();

  if (isset($err['message'])) {
    $m = htmlentities($err['message']);
  }
  else {
    $m = 'Unknown DB error';
  }

  echo '<p><b>Error</b>: at line '.$line.' of '.$file.'</p>';
  echo '<pre>'.$m.'</pre>';

  exit;
}

?>
```

The db_do_query() function in this example uses the oci_fetch_all() OCI8 function. The oci_fetch_all() function accepts the following five parameters:

- $stid, the statement identifier for the statement executed
- $results, the output array variable containing the data returned for the query
- The null in the third parameter for the number of initial rows to skip is ignored.
- The null in the fourth parameter for the maximum number of rows to fetch is ignored. In this case, all the rows for the query are returned. For this example where the result set is not large, it is acceptable.
- The last parameter flag OCI_FETCHSTATEMENT_BY_ROW indicates that the data in the $results array is organized by row, where each row contains an array of column values. A value of OCI_FETCHSTATEMENT_BY_COLUMN causes the results array to be organized by column, where each column entry contains an array of column values for each row. Your choice of value for this flag depends on how you intend to process the data in your logic.

To examine the structure of the result array, use the PHP var_dump() function after the query has been executed. This is useful for debugging. For example:

```
print '<pre>';
var_dump($results);
print '</pre>';
```

The db_error() function accepts three arguments. The $r parameter can be false or null for obtaining connection errors, or a connection resource or statement resource to obtain an error for those contexts. The $file and $line values are populated by using __FILE__ and __LINE__, respectively, as the actual parameters to enable the error message to display the source file and line from which the database error is reported. This enables you to easily track the possible cause of errors.

The db_ error() function calls the oci_error() function to obtain database error messages.

The db_error() function calls the isset() function before printing the message. The isset() function checks if the message component of the database error structure is set, or if the error is unknown.

4. Edit anyco_ui.inc. To format the results of a single row from the DEPARTMENTS table query in an HTML table format, insert the following function:

```
function ui_print_department($dept)
{
  if (!$dept) {
    echo '<p>No Department found</p>';
  }
  else {
    echo <<<END
  <table>
  <tr>
    <th>Department<br>ID</th>
    <th>Department<br>Name</th>
    <th>Manager<br>Id</th>
    <th>Location ID</th>
  </tr>
```

```
  <tr>
END;
    echo '<td>'.htmlentities($dept['DEPARTMENT_ID']).'</td>';
    echo '<td>'.htmlentities($dept['DEPARTMENT_NAME']).'</td>';
    echo '<td>'.htmlentities($dept['MANAGER_ID']).'</td>';
    echo '<td>'.htmlentities($dept['LOCATION_ID']).'</td>';
    echo <<<END
  </tr>
  </table>
END;
  }
}
```

As noted in Chapter 3, do not prefix `END;` lines with leading spaces. If you do, the rest of the document will be treated as part of the text to be printed.

5. Edit the `anyco.php` file. Include the `anyco_ui.inc` and `anyco_db.inc` files, and call the database functions to query and display information for a department with a `department_id` of 80 by using the following code. The file becomes:

```
<?php // File: anyco.php

require('anyco_cn.inc');
require('anyco_db.inc');
require('anyco_ui.inc');

$query =
  'SELECT   department_id, department_name, manager_id, location_id
   FROM     departments
   WHERE    department_id = 80';

$conn = db_connect();

$dept = db_do_query($conn, $query);
ui_print_header('Departments');
ui_print_department($dept[0]);
ui_print_footer(date('Y-m-d H:i:s'));

?>
```

6. To test the resulting changes to the application, enter the following URL in a browser window:

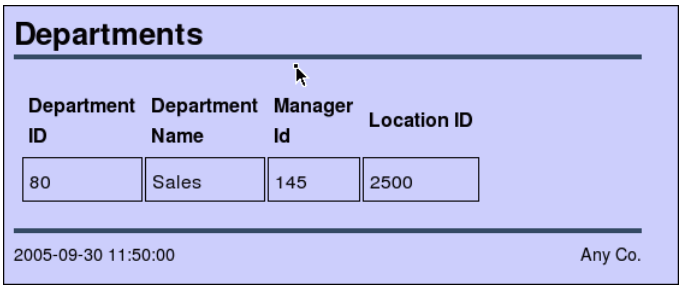
   On Windows:

   `http://localhost/chap4/anyco.php`

   On Linux:

   `http://localhost/~<username>/chap4/anyco.php`

   The page returned in the browser window should resemble the following page:



Writing Queries with Bind Variables

Using queries with values included in the WHERE clause may be useful for some situations. However, if the conditional values in the query are likely to change it is not appropriate to encode a value into the query. Oracle recommends that you use bind variables.

A bind variable is a symbolic name preceded by a colon in the query that acts as a placeholder for literal values. For example, the query string created in the `anyco.php` file could be rewritten with the bind variable `:did` as follows:

```
$query =
  'SELECT   department_id, department_name, manager_id, location_id
   FROM     departments
   WHERE    department_id = :did';
```

By using bind variables to parameterize SQL statements:

- The statement is reusable with different input values without needing to change the code.
- The query performance is improved through a reduction of the query parse time in the server, because the Oracle database can reuse parse information from the previous invocations of the identical query string.
- There is protection against "SQL Injection" security problems.
- There is no need to specially handle quotation marks in user input.

When a query uses a bind variable, the PHP code must associate an actual value with each bind variable (placeholder) used in the query before it is executed. This process is known as run-time binding.

To enable your PHP application to use bind variables in the query, perform the following changes to your PHP application code:

1. Edit the `anyco.php` file. Modify the query to use a bind variable, create an array to store the value to be associated with the bind variable, and pass the `$bindargs` array to the `db_do_query()` function:

```
<?php // File: anyco.php
...

$query =
'SELECT   department_id, department_name, manager_id, location_id
 FROM     departments
 WHERE    department_id = :did';

$bindargs = array();
// In the $bindargs array add an array containing
// the bind variable name used in the query, its value, a length
```

```
array_push($bindargs, array('DID', 80, -1));

$conn = db_connect();
$dept = db_do_query($conn, $query, $bindargs);

...
?>
```

In this example, the bind variable, called DID, is an input argument in the parameterized query, and it is associated with the value 80. Later, the value of the bind variable will be dynamically determined. In addition, the length component is passed as -1 so that the OCI8 layer can determine the length. If the bind variable was used to return output from the database an explicit size would be required.

2. Edit the `anyco_db.inc` file. Modify the `db_do_query()` function to accept a `$bindvars` array variable as a third parameter. Call the `oci_bind_by_name()` OCI8 call to associate the PHP values supplied in `$bindvars` parameter with bind variables in the query. The function becomes:

```
function db_do_query($conn, $statement, $bindvars = array())
{
  $stid = oci_parse($conn, $statement);
  if (!$stid) {
    db_error($conn, __FILE__, __LINE__);
  }

  // Bind the PHP values to query bind parameters
  foreach ($bindvars as $b) {
    // create local variable with caller specified bind value
    $$b[0] = $b[1];
    // oci_bind_by_name(resource, bv_name, php_variable, length)
    $r = oci_bind_by_name($stid, ":$b[0]", $$b[0], $b[2]);
    if (!$r) {
      db_error($stid, __FILE__, __LINE__);
    }
  }
  $r = oci_execute($stid, OCI_DEFAULT);
  if (!$r) {
    db_error($stid, __FILE__, __LINE__);
  }
  $r = oci_fetch_all($stid, $results, null, null,
       OCI_FETCHSTATEMENT_BY_ROW);
  return($results);
}
```

The binding is performed in the `foreach` loop before the `oci_execute()` is done.

For each entry in `$bindvars` array, the first element contains the query bind variable name that is used to create a PHP variable of the same name; that is, `$$b[0]` takes the value DID in `$b[0]` and forms a PHP variable called `$DID` whose value is assigned from the second element in the entry.

The `oci_bind_by_name()` function accepts four parameters: the `$stid` as the resource, a string representing the bind variable name in the query derived from the first element in the array entry, the PHP variable containing the value to be associated with the bind variable, and the length of the input value.

3. To test the results of the preceding modifications, save the `anyco.php` and `anyco_db.inc` files and enter the following URL:

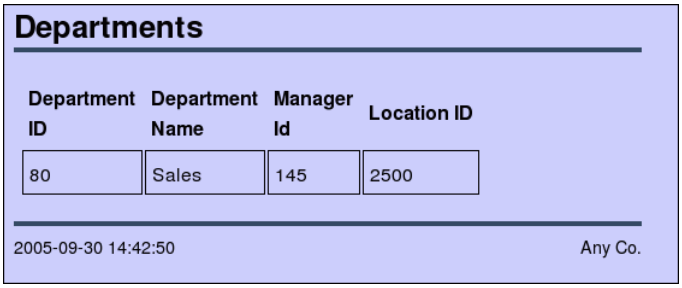
On Windows:

```
http://localhost/chap4/anyco.php
```

On Linux:

```
http://localhost/~<username>/chap4/anyco.php
```

The page returned in the browser window should resemble the following page:



Navigating Through Database Records

Adding navigation through the database records requires several important changes to the application logic. The modifications require the combination of:

- Including an HTML form to provide **Next** and **Previous** navigation buttons to step through database records.
- Detecting if the HTTP request for the page was posted by clicking the **Next** or **Previous** button.
- Tracking the last row queried by using the HTTP session state. A PHP session is started to maintain state information for a specific client between HTTP requests. The first HTTP request will retrieve the first data row and initialize the session state. A subsequent request, initiated with navigation buttons, combined with the session state from a previous HTTP request, enables the application to set variables that control the next record retrieved by the query.
- Writing a query that returns a subset of rows based on a set of conditions whose values are determined by the application state.

To add navigation through database rows, perform the following steps:

1. Edit the `anyco_ui.inc` file. Add **Next** and **Previous** navigation buttons to the Departments page. Change the `ui_print_department()` function to append a second parameter called `$posturl` that supplies the value for the form attribute `action`. After printing the `</table>` tag include HTML form tags for the **Next** and **Previous** buttons:

```
<?php // File: anyco_ui.inc
...
function ui_print_department($dept, $posturl)
{
  ...
    echo <<<END
  </tr>
  </table>
  <form method="post" action="$posturl">
  <input type="submit" value="< Previous" name="prevdept">
  <input type="submit" value="Next >"     name="nextdept">
  </form>
```

```
END;
  }
}

?>
```

2. Edit the `anyco.php` file. To detect if the **Next** or **Previous** button was used to invoke the page and track the session state, call the PHP function `session_start()`, and create a function named `construct_departments()`:

   Move and modify the database access logic into a new `construct_departments()` function, which detects if navigation has been performed, manages the session state, defines a subquery for the database access layer to process, and connects and calls a function `db_get_page_data()`. The file becomes:

```
<?php // File: anyco.php

require('anyco_cn.inc');
require('anyco_db.inc');
require('anyco_ui.inc');

session_start();
construct_departments();

function construct_departments()
{
  if (isset($_SESSION['currentdept']) &&
      isset($_POST['prevdept']) &&
      $_SESSION['currentdept'] > 1) {
    $current = $_SESSION['currentdept'] - 1;
  }
  elseif (isset($_SESSION['currentdept']) &&
          isset($_POST['nextdept'])) {
    $current = $_SESSION['currentdept'] + 1;
  }
  elseif (isset($_POST['showdept']) &&
          isset($_SESSION['currentdept'])) {
    $current = $_SESSION['currentdept'];
  }
  else {
    $current = 1;
  }

  $query = 'SELECT department_id, department_name,
                   manager_id, location_id
            FROM   departments
            ORDER BY department_id asc';

  $conn = db_connect();

  $dept = db_get_page_data($conn, $query, $current, 1);
  $deptid = $dept[0]['DEPARTMENT_ID'];

  $_SESSION['currentdept'] = $current;

  ui_print_header('Department');
  ui_print_department($dept[0], $_SERVER['SCRIPT_NAME']);
  ui_print_footer(date('Y-m-d H:i:s'));
}
```

```
?>
```

The `if` and `elseif` construct at the start of the `construct_departments()` function is used to detect if a navigation button was used with an HTTP post request to process the page, and tracks if the `currentdept` number is set in the session state. Depending on the circumstances, the variable `$current` is decremented by one when the previous button is clicked, `$current` is incremented by one when the **Next** button is clicked, otherwise `$current` is set to the current department, or initialized to one for the first time through.

A query is formed to obtain all the department rows in ascending sequence of the `department_id`. The `ORDER BY` clause is an essential part of the navigation logic. The query is used as a subquery inside the `db_get_page_data()` function to obtain a page of a number of rows, where the number of rows per page is specified as the fourth argument to the `db_get_page_data()` function. After connecting to the database, `db_get_page_data()` is called to retrieve the set of rows obtained for the specified query. The `db_get_page_data()` function is provided with the connection resource, the query string, a value in `$current` specifying the first row in the next page of data rows required, and the number of rows per page (in this case one row per page).

After `db_get_page_data()`has been called to obtain a page of rows, the value of `$current` is stored in the application session state.

Between printing the page header and footer, the `ui_print_department()` function is called to display the recently fetched department row. The `ui_print_department()` function uses `$_SERVER['SCRIPT_NAME']` to supply the current PHP script name for the `$posturl` parameter. This sets the action attribute in the HTML form, so that each **Next** or **Previous** button click calls the `anyco.php` file.

3. Edit the `anyco_db.inc` file. Implement the `db_get_page_data()` function to query a subset of rows:

```
// Return subset of records
function db_get_page_data($conn, $q1, $current = 1,
                $rowsperpage = 1, $bindvars = array())
{
  // This query wraps the supplied query, and is used
  // to retrieve a subset of rows from $q1
  $query = 'SELECT *
            FROM (SELECT A.*, ROWNUM AS RNUM
                  FROM ('.$q1.') A
                  WHERE ROWNUM <= :LAST)
            WHERE :FIRST <= RNUM';

  // Set up bind variables.
  array_push($bindvars, array('FIRST', $current, -1));
  array_push($bindvars,
             array('LAST', $current+$rowsperpage-1, -1));

  $r = db_do_query($conn, $query, $bindvars);
  return($r);
}
```

The structure of the query in the `db_get_page_data()` function enables navigation through a set (or page) of database rows.

The query supplied in `$q1` is nested as a subquery inside the following subquery:

```
SELECT A.*, ROWNUM AS RNUM FROM $q1 WHERE ROWNUM <= :LAST
```

Remember that the query supplied in `$q1` retrieves an ordered set of rows, which is filtered by its enclosing query to return all the rows from the first row to the next page size (`$rowsperpage`) of rows. This is possible because the Oracle `ROWNUM` function (or pseudocolumn) returns an integer number starting at 1 for each row returned by the query in `$q1`.

The set of rows, returned by the subquery enclosing query `$q1`, is filtered a second time by the condition in the following outermost query:

```
WHERE :FIRST <= RNUM
```

This condition ensures that rows prior to the value in `:FIRST` (the value in `$current`) are excluded from the final set of rows. The query enables navigation through a set rows where the first row is determined by the `$current` value and the page size is determined by the `$rowsperpage` value.

The `$current` value is associated with the bind variable called `:FIRST`. The expression `$current+$rowsperpage-1` sets the value associated with the `:LAST` bind variable.

4. To test the changes made to your application, save the changed files, and enter the following URL in your Web browser:

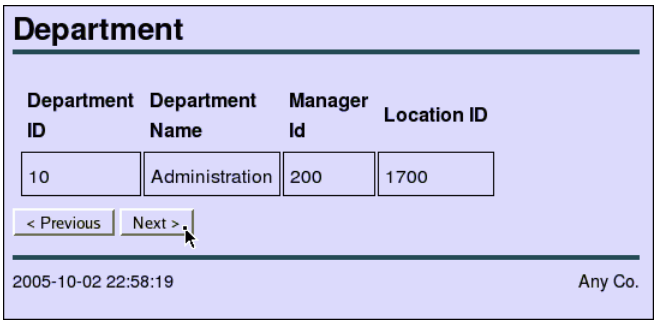
   On Windows:

   ```
   http://localhost/chap4/anyco.php
   ```

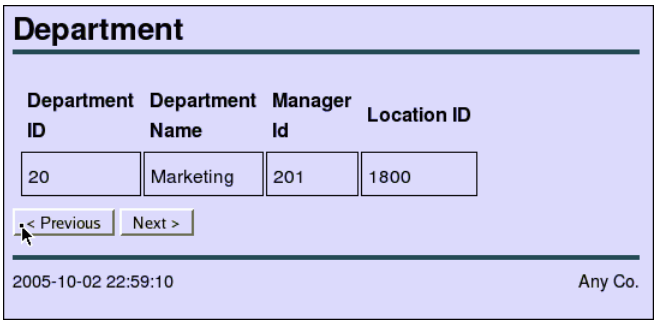
   On Linux:

   ```
   http://localhost/~<username>/chap4/anyco.php
   ```

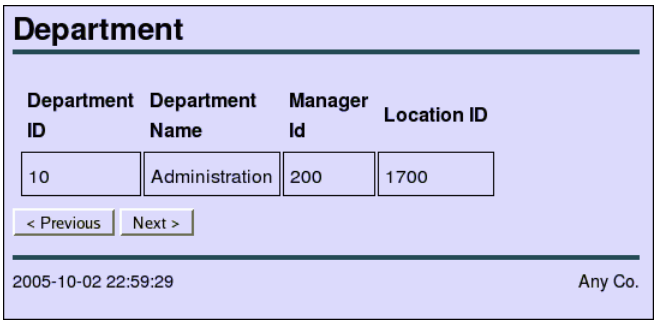
   When you request the `anyco.php` page, the first `DEPARTMENT` table record, the Administration department, is displayed:



5. To navigate to the next department record (Marketing), click **Next**:



6. To navigate back to the first department record (Administration), click **Previous**:



You may continue to test and experiment with the application by clicking **Next** and **Previous** to navigate to other records in the DEPARTMENTS table, as desired.

Note: If you navigate past the last record in the DEPARTMENTS table, an error will occur. Error handling is added in Adding Error Recovery in Chapter 5.

Extending the Basic Departments Page

The Departments page is extended to include the following additional information:

- The name of the manager of the department
- The number of employees assigned to the department
- The country name identifying the location of the department

The additional information is obtained by modifying the query to perform a join operation between the DEPARTMENTS, EMPLOYEES, LOCATIONS, and COUNTRIES tables.

To extend the Departments page, perform the following tasks:

1. Edit the anyco_ui.inc file. Modify the ui_print_department() function by replacing the Manager ID and Location ID references with the Manager Name and Location, respectively, and insert a Number of Employees field after Department Name. Make the necessary changes in the table header and data fields. The function becomes:

```
function ui_print_department($dept, $posturl)
{
  if (!$dept) {
    echo '<p>No Department found</p>';
  }
  else {
    echo <<<END
  <table>
  <tr>
    <th>Department<br>ID</th>
    <th>Department<br>Name</th>
    <th>Number of<br>Employees</th>
    <th>Manager<br>Name</th>
    <th>Location</th>
  </tr>
  <tr>
END;
  echo '<td>'.htmlentities($dept['DEPARTMENT_ID']).'</td>';
```

```
  echo '<td>'.htmlentities($dept['DEPARTMENT_NAME']).'</td>';
  echo '<td>'.htmlentities($dept['NUMBER_OF_EMPLOYEES']).'</td>';
  echo '<td>'.htmlentities($dept['MANAGER_NAME']).'</td>';
  echo '<td>'.htmlentities($dept['COUNTRY_NAME']).'</td>';
  echo <<<END
  </tr>
  </table>
  <form method="post" action="$posturl">
  <input type="submit" value="< Previous" name="prevdept">
  <input type="submit" value="Next >"     name="nextdept">
  </form>
END;
  }
}
```

There is no need to pass a `$bindargs` parameter to the `db_do_query()` call because this function does not use bind variables. The `db_do_query()` declaration will provide a default value of an empty array automatically. PHP allows functions to have variable numbers of parameters.

2. Edit the `anyco.php` file. Replace the query string in `construct_departments()` with:

```
$query =
  "SELECT d.department_id, d.department_name,
       substr(e.first_name,1,1)||'. '|| e.last_name as manager_name,
       c.country_name, count(e2.employee_id) as number_of_employees
   FROM  departments d, employees e, locations l,
         countries c, employees e2
   WHERE d.manager_id = e.employee_id
   AND d.location_id = l.location_id
   AND d.department_id = e2.department_id
   AND l.country_id = c.country_id
   GROUP BY d.department_id, d.department_name,
         substr(e.first_name,1,1)||'. '||e.last_name,
         c.country_name
   ORDER BY d.department_id ASC";
```

The query string is enclosed in double quotation marks to simplify writing this statement, which contains SQL literal strings in single quotation marks.

3. Save the changes to your files, and test the changes by entering the following URL in a Web browser:

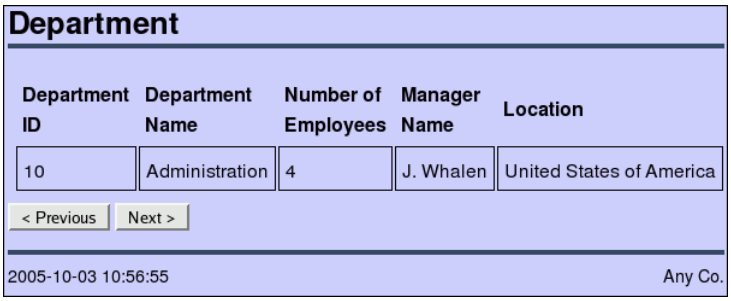
   On Windows:

   ```
   http://localhost/chap4/anyco.php
   ```

   On Linux:

   ```
   http://localhost/~<username>/chap4/anyco.php
   ```

   The Web page result should resemble the following output:


Department
Department ID
Department Name
Number of Employees
Manager Name
Location
10
Administration
4
J. Whalen
United States of America
< Previous
Next >
2005-10-03 10:56:55
Any Co.

5

Updating Data

In this chapter, you extend the Anyco HR application with forms that enable you to insert, update, and delete an employee record.

- Building the Basic Employees page
- Extending the Basic Employees Page
- Combining Departments and Employees
- Adding Error Recovery
- Further Error Handling

Building the Basic Employees page

In this section, you will extend your application to include a basic employees page.

To display employee records, perform the following tasks:

1. Create the `chap5` directory, copy the application files from `chap4`, and change to the newly created directory:

   On Windows:

   ```
   mkdir c:\program files\Apache Group\Apache2\htdocs\chap5
   cd c:\program files\Apache Group\Apache2\htdocs\chap5
   copy ..\chap4\* .
   ```

   On Linux:

   ```
   mkdir $HOME/public_html/chap5
   cd $HOME/public_html/chap5
   cp ../chap4/* .
   ```

2. Edit the `anyco.php` file. Add a `construct_employees()` function. This function constructs the employee query, calls the `db_do_query()` function to execute the query, and prints the results using the `ui_print_employees()` function:

   ```
   function construct_employees()
   {
     $query =
     "SELECT employee_id,
       substr(first_name,1,1) || '.  '|| last_name as employee_name,
       hire_date,
       to_char(salary, '9999G999D99') as salary,
       nvl(commission_pct,0) as commission_pct
      FROM   employees
   ```

```
   ORDER BY employee_id asc";

  $conn = db_connect();
  $emp = db_do_query($conn, $query);

  ui_print_header('Employees');
  ui_print_employees($emp);
  ui_print_footer(date('Y-m-d H:i:s'));
}
```

3. Edit the `anyco.php` file. Replace the call to `construct_departments()` with a call to `construct_employees()`:

```
<?php // File: anyco.php

require('anyco_cn.inc');
require('anyco_db.inc');
require('anyco_ui.inc');

session_start();
construct_employees();
...
?>
```

4. Edit the `anyco_ui.inc` file. Implement the presentation of employee data in an HTML table by adding a `ui_print_employees()` function:

```
function ui_print_employees($employeerecords)
{
  if (!$employeerecords) {
    echo '<p>No Employee found</p>';
  }
  else {
    echo <<<END
  <table>
  <tr>
    <th>Employee<br>ID</th>
    <th>Employee<br>Name</th>
    <th>Hiredate</th>
    <th>Salary</th>
    <th>Commission<br>(%)</th>
  </tr>
END;
    // Write one row per employee
    foreach ($employeerecords as $emp) {
      echo '<tr>';
      echo '<td align="right">'.
            htmlentities($emp['EMPLOYEE_ID']).'</td>';
      echo '<td>'.htmlentities($emp['EMPLOYEE_NAME']).'</td>';
      echo '<td>'.htmlentities($emp['HIRE_DATE']).'</td>';
      echo '<td align="right">'.
            htmlentities($emp['SALARY']).'</td>';
      echo '<td align="right">'.
            htmlentities($emp['COMMISSION_PCT']).'</td>';
      echo '</tr>';
    }
    echo <<<END
  </table>
END;
  }
}
```

5. Save the changes to the `anyco.php` and `anyco_ui.inc` files. Test the result of these changes by entering the following URL in your Web browser:

   On Windows:

   ```
   http://localhost/chap4/anyco.php
   ```

   On Linux:

   ```
   http://localhost/~<username>/chap4/anyco.php
   ```

   Examine the result page, and scroll down to view all the employee records displayed in the page:

Employees

Employee ID	Employee Name	Hiredate	Salary	Commission (%)
100	S. King	17-JUN-87	24,000.00	0
101	N. Kochhar	21-SEP-89	17,000.00	0
102	L. De Haan	13-JAN-93	17,000.00	0
103	A. Hunold	03-JAN-90	9,000.00	0
104	B. Ernst	21-MAY-91	6,000.00	0
105	D. Austin	25-JUN-97	4,800.00	0
106	V. Pataballa	05-FEB-98	4,800.00	0
107	D. Lorentz	07-FEB-99	4,200.00	0
108	N. Greenberg	17-AUG-94	12,000.00	0

Extending the Basic Employees Page

In this section, you will extend the basic employees page to include the ability to manipulate employee records.

To enable employee records to be manipulated, perform the following tasks:

1. Edit the `anyco.php` file. Add form handler control logic to manage the requests for showing, inserting, updating, and deleting employee records:

```
<?php // File: anyco.php

require('anyco_cn.inc');
require('anyco_db.inc');
require('anyco_ui.inc');

session_start();
// Start form handler code
if (isset($_POST['insertemp'])) {
  construct_insert_emp();
}
elseif (isset($_POST['saveinsertemp'])) {
  insert_new_emp();
}
elseif (isset($_POST['modifyemp'])) {
  construct_modify_emp();
```

```
}
elseif (isset($_POST['savemodifiedemp'])) {
  modify_emp();
}
elseif (isset($_POST['deleteemp'])) {
  delete_emp();
}
else {
  construct_employees();
}

...
```

2. Edit the `anyco.php` file. Add the `construct_insert_emp()` function:

```
function construct_insert_emp()
{
  $conn = db_connect();

  $query = "SELECT job_id, job_title
            FROM jobs
            ORDER BY job_title ASC";
  $jobs = db_do_query($conn, $query,
                      OCI_FETCHSTATEMENT_BY_COLUMN);

  $query = "SELECT sysdate FROM dual";
  $date = db_do_query($conn, $query,
                      OCI_FETCHSTATEMENT_BY_COLUMN);
  $emp = array(
    'DEPARTMENT_ID' => 10,      // Default to department 10
    'HIRE_DATE' => $date['SYSDATE'][0],
    'ALLJOBIDS' => $jobs['JOB_ID'],
    'ALLJOBTITLES' => $jobs['JOB_TITLE']
    );

  ui_print_header('Insert New Employee');
  ui_print_insert_employee($emp, $_SERVER['SCRIPT_NAME']);
  // Note: The two kinds of date used:
  // 1) SYSDATE for storing an SQL date in the database, and
  // 2) The PHP date for display in the footer of each page
  ui_print_footer(date('Y-m-d H:i:s'));
}
```

The `construct_insert_emp()` function executes two queries to obtain default data to be used to populate the Insert New Employee form, which is displayed by the `ui_print_insert_employee()` function.

The `$query` of the `JOBS` table obtains a list of all the existing job IDs and their descriptions in order to build a list for selecting a job type in the HTML form generated by the `ui_print_insert_employee()` function.

The `$query` using `SYSDATE` obtains the current database date and time for setting the default hire date of the new employee.

There are two kinds of date used in the application code, the PHP `date()` function for printing the date and time in the page footer, and the Oracle `SYSDATE` function to obtain the default date and time for displaying in the hire date field of the Employees page and to ensure that text is entered in the correct database format.

The two db_do_query() function calls provide an additional parameter value OCI_FETCHSTATEMENT_BY_COLUMN to specify that the return type for the query is an array of column values.

3. Edit the anyco.php file. Add the insert_new_emp() function to insert an employee record into the EMPLOYEES table:

```
function insert_new_emp()
{
  $newemp = $_POST;
  $statement =
    "INSERT INTO employees
        (employee_id, first_name, last_name, email, hire_date,
         job_id, salary, commission_pct, department_id)
     VALUES (employees_seq.nextval, :fnm, :lnm, :eml, :hdt, :jid,
             :sal, :cpt, :did)";

  $conn = db_connect();
  $emailid = $newemp['firstname'].$newemp['lastname'];

  $bindargs = array();
  array_push($bindargs, array('FNM', $newemp['firstname'], -1));
  array_push($bindargs, array('LNM', $newemp['lastname'], -1));
  array_push($bindargs, array('EML', $emailid, -1));
  array_push($bindargs, array('HDT', $newemp['hiredate'], -1));
  array_push($bindargs, array('JID', $newemp['jobid'], -1));
  array_push($bindargs, array('SAL', $newemp['salary'], -1));
  array_push($bindargs, array('CPT', $newemp['commpct'], -1));
  array_push($bindargs, array('DID', $newemp['deptid'], -1));

  $r = db_execute_statement($conn, $statement, $bindargs);
  construct_employees();
}
```

The return value from the db_execute_statement() function is ignored and not even assigned to a variable, because no action is performed on its result until later.

4. Edit the anyco.php file. Add the construct_modify_emp() function to build the HTML form for updating an employee record.

```
function construct_modify_emp()
{
  $empid = $_POST['emprec'];
  $query =
    "SELECT employee_id, first_name, last_name, email, hire_date,
            salary, nvl(commission_pct,0) as commission_pct
     FROM   employees
     WHERE  employee_id = :empid";

  $conn = db_connect();
  $bindargs = array();
  array_push($bindargs, array('EMPID', $empid, -1));

  $emp = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_ROW,
                                    $bindargs);

  ui_print_header('Modify Employee ');
  ui_print_modify_employee($emp[0], $_SERVER['SCRIPT_NAME']);
  ui_print_footer(date('Y-m-d H:i:s'));
}
```

5. Edit the `anyco.php` file. Add the `modify_emp()` function to update the employee record in the `EMPLOYEES` table, using the update form field values:

```
function modify_emp()
{
  $newemp = $_POST;
  $statement =
    "UPDATE employees
     SET   first_name = :fnm, last_name = :lnm, email = :eml,
           salary = :sal, commission_pct = :cpt
     WHERE employee_id = :eid";

  $conn = db_connect();
  $bindargs = array();
  array_push($bindargs, array('EID', $newemp['empid'], -1));
  array_push($bindargs, array('FNM', $newemp['firstname'], -1));
  array_push($bindargs, array('LNM', $newemp['lastname'], -1));
  array_push($bindargs, array('EML', $newemp['email'], -1));
  array_push($bindargs, array('SAL', $newemp['salary'], -1));
  array_push($bindargs, array('CPT', $newemp['commpct'], -1));

  $r = db_execute_statement($conn, $statement, $bindargs);
  construct_employees();
}
```

6. Edit the `anyco.php` file. Add the `delete_emp()` function to delete an employee record from the `EMPLOYEES` table:

```
function delete_emp()
{
  $empid = $_POST['emprec'];
  $statement = "DELETE FROM employees
                WHERE employee_id = :empid";

  $conn = db_connect();
  $bindargs = array();
  array_push($bindargs, array('EMPID', $empid, 10));
  $r = db_execute_statement($conn, $statement, $bindargs);

  construct_employees();
}
```

7. Edit the `anyco.php` file. In the `construct_employees()` function, modify the `db_do_query()` call to supply `OCI_FETCHSTATEMENT_BY_ROW` as the last parameter, and provide `$_SERVER['SCRIPT_NAME']` as second parameter in the `ui_print_employees()` call. The function becomes:

```
function construct_employees()
{
  $query =
  "SELECT employee_id,
    substr(first_name,1,1) || '.  '|| last_name as employee_name,
    hire_date,
    to_char(salary, '9999G999D99') as salary,
    nvl(commission_pct,0) as commission_pct
   FROM   employees
   ORDER BY employee_id asc";

  $conn = db_connect();
  $emp = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_ROW);
```

```
  ui_print_header('Employees');
  ui_print_employees($emp, $_SERVER['SCRIPT_NAME']);
  ui_print_footer(date('Y-m-d H:i:s'));
}
```

8. Edit the `anyco_db.inc` file. Add `$resulttype` as a third parameter to the `db_do_query()` call. Replace the last parameter value, `OCI_FETCHSTATEMENT_BY_ROW`, in the `oci_fetch_all()` call with a variable, so that callers can choose the output type.

```
function db_do_query($conn, $statement, $resulttype,
                     $bindvars = array())
{
  $stid = oci_parse($conn, $statement);

  ...

  $r = oci_fetch_all($stid, $results, null, null, $resulttype);
  return($results);
}
```

9. Edit the `anyco_db.inc` file. Inside the `db_get_page_data()` function, insert `OCI_FETCHSTATEMENT_BY_ROW` as the third parameter value in the `db_do_query()` call:

```
function db_get_page_data($conn, $q1, $current = 1,
                          $rowsperpage = 1, $bindvars = array())
{

  ...

  $r = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_ROW, $bindvars);
  return($r);
}
```

10. Edit the `anyco_db.inc` file. Add a `db_execute_statement()` function to execute data manipulation statements:

```
function db_execute_statement($conn, $statement, $bindvars = array())
{
  $stid = oci_parse($conn, $statement);
  if (!$stid) {
    db_error($conn, __FILE__, __LINE__);
  }

  // Bind parameters
  foreach ($bindvars as $b) {
    // create local variable with caller specified bind value
    $$b[0] = $b[1];
    $r = oci_bind_by_name($stid, ":$b[0]", $$b[0], $b[2]);
    if (!$r) {
      db_error($stid, __FILE__, __LINE__);
    }
  }

  $r = oci_execute($stid);
  if (!$r) {
    db_error($stid, __FILE__, __LINE__);
  }
  return($r);
```

```
}
```

11. Edit the `anyco_ui.inc` file. Change the `ui_print_employees()` function to produce an HTML form containing the employee rows. The function becomes:

```
function ui_print_employees($employeerecords, $posturl)
{
  if (!$employeerecords) {
    echo '<p>No Employee found</p>';
  }
  else {
    echo <<<END
  <form method="post" action="$posturl">
  <table>
  <tr>
    <th> </th>
    <th>Employee<br>ID</th>
    <th>Employee<br>Name</th>
    <th>Hiredate</th>
    <th>Salary</th>
    <th>Commission<br>(%)</th>
  </tr>
END;
    // Write one row per employee
    foreach ($employeerecords as $emp) {
      echo '<tr>';
      echo '<td><input type="radio" name="emprec" value="'.
        htmlentities($emp['EMPLOYEE_ID']).'"></td>';
      echo '<td align="right">'.
        htmlentities($emp['EMPLOYEE_ID']).'</td>';
      echo '<td>'.htmlentities($emp['EMPLOYEE_NAME']).'</td>';
      echo '<td>'.htmlentities($emp['HIRE_DATE']).'</td>';
      echo '<td align="right">'.
        htmlentities($emp['SALARY']).'</td>';
      echo '<td align="right">'.
        htmlentities($emp['COMMISSION_PCT']).'</td>';
      echo '</tr>';
    }
    echo <<<END
  </table>
  <input type="submit" value="Modify" name="modifyemp">
  <input type="submit" value="Delete" name="deleteemp">
    
  <input type="submit" value="Insert new employee"
         name="insertemp">
  </form>
END;
  }
}
```

A radio button is displayed in the first column of each row to enable you to select the record to be modified or deleted.

12. Edit the `anyco_ui.inc` file. Add the `ui_print_insert_employee()` function to generate the form to input new employee data:

```
function ui_print_insert_employee($emp, $posturl)
{
  if (!$emp) {
    echo "<p>No employee details found</p>";
  }
```

```
  else {
    $deptid = htmlentities($emp['DEPARTMENT_ID']);
    $hiredate = htmlentities($emp['HIRE_DATE']);

    echo <<<END
  <form method="post" action="$posturl">
  <table>
    <tr>
      <td>Department ID</td>
      <td><input type="text" name="deptid" value="$deptid"
                 size="20"></td>
    </tr>
    <tr>
      <td>First Name</td>
      <td><input type="text" name="firstname" size="20"></td>
    </tr>
    <tr>
      <td>Last Name</td>
      <td><input type="text" name="lastname" size="20"></td>
    </tr>
    <tr>
      <td>Hiredate</td>
      <td><input type="text" name="hiredate" value="$hiredate"
                 size="20"></td>
    </tr>
    <tr>
      <td>Job</td>
       <td><select name="jobid">
END;
    // Write the list of jobs
    for ($i = 0; $i < count($emp['ALLJOBIDS']); $i++)
    {
      echo '<option
             label="'.htmlentities($emp['ALLJOBTITLES'][$i]).'"'.
           ' value="'.htmlentities($emp['ALLJOBIDS'][$i]).'">'.
           htmlentities($emp['ALLJOBTITLES'][$i]).'</option>';
    }
    echo <<<END
      </select>
      </td>
    </tr>
    <tr>
      <td>Salary</td>
      <td><input type="text" name="salary" value="1"
                 size="20"></td>
    </tr>
    <tr>
      <td>Commission (%)</td>
      <td><input type="text" name="commpct" value="0"
                 size="20"></td>
    </tr>
  </table>
    <input type="submit" value="Save" name="saveinsertemp">
    <input type="submit" value="Cancel" name="cancel">
  </form>
END;
  }
}
```

13. Edit the `anyco_ui.inc` file. Add the `ui_print_modify_employee()` function to generate the form to update an employee record:

```
function ui_print_modify_employee($empdetails, $posturl)
{
  if (!$empdetails) {
    echo '<p>No Employee record selected</p>';
  }
  else {
    $fnm = htmlentities($empdetails['FIRST_NAME']);
    $lnm = htmlentities($empdetails['LAST_NAME']);
    $eml = htmlentities($empdetails['EMAIL']);
    $sal = htmlentities($empdetails['SALARY']);
    $cpt = htmlentities($empdetails['COMMISSION_PCT']);
    $eid = htmlentities($empdetails['EMPLOYEE_ID']);

    echo <<<END
  <form method="post" action="$posturl">
  <table>
    <tr>
      <td>Employee ID</td>
      <td>$eid</td></tr>
    <tr>
      <td>First Name</td>
      <td><input type="text" name="firstname" value="$fnm"></td>
    </tr>
    <tr>
      <td>Last Name</td>
      <td><input type="text" name="lastname" value="$lnm"></td>
    </tr>
    <tr>
      <td>Email Address</td>
      <td><input type="text" name="email" value="$eml"></td>
    </tr>
    <tr>
      <td>Salary</td>
      <td><input type="text" name="salary" value="$sal"></td>
    </tr>
    <tr>
      <td>Commission (%)</td>
      <td><input type="text" name="commpct" value="$cpt"></td>
    </tr>
  </table>
  <input type="hidden" value="{$empdetails['EMPLOYEE_ID']}"
         name="empid">
  <input type="submit" value="Save" name="savemodifiedemp">
  <input type="submit" value="Cancel" name="cancel">
  </form>
END;
  }
}
```

14. Save the changes to your Anyco application files, and test the changes by entering the following URL in your Web browser:

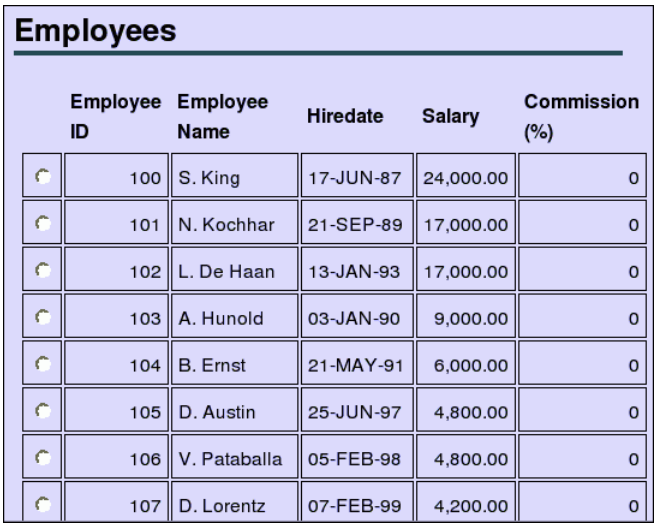
    On Windows:

    `http://localhost/chap5/anyco.php`

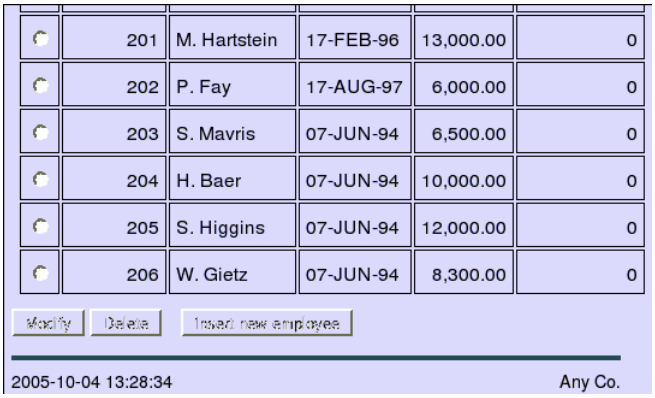
    On Linux:

    `http://localhost/~<username>/chap5/anyco.php`

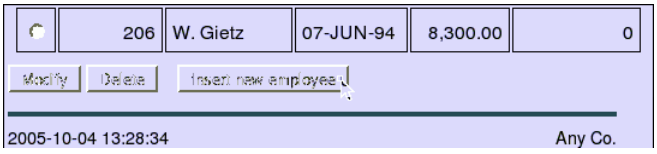
The list of all employees is displayed with a radio button in each row.



Scroll to the bottom of the Employees page to view the **Modify**, **Delete**, and **Insert new employee** buttons:



15. To insert a new employee record, click **Insert new employee**:



16. When you create or modify employee records, you will see that the database definitions require the salary to be greater than zero, and the commission to be less than 1. The commission will be rounded to two decimal places. In the Insert New Employee page, the Department ID field contains 10 (the default), Hiredate contains the current date (in default database date format), Salary contains 1, and Commission (%) contains 0. Enter the following field values:

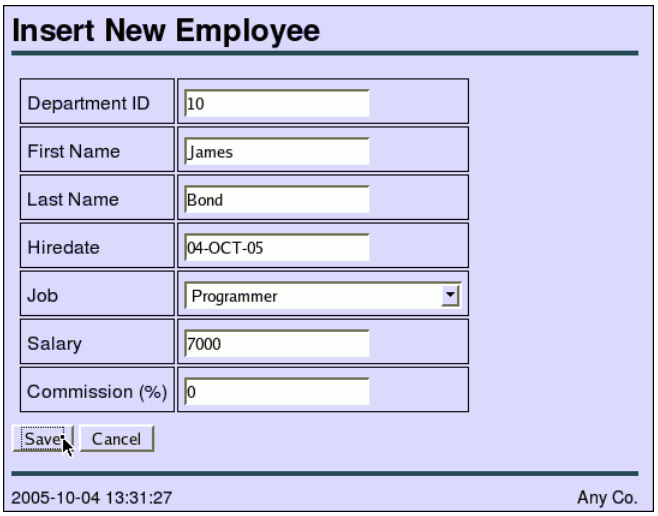
    First Name: James

    Last Name: Bond

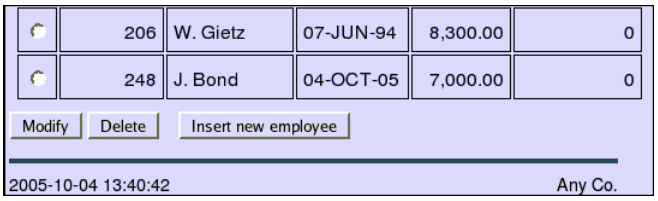
    Job: Select Programmer from the list.

    Salary: replace the 1 with 7000

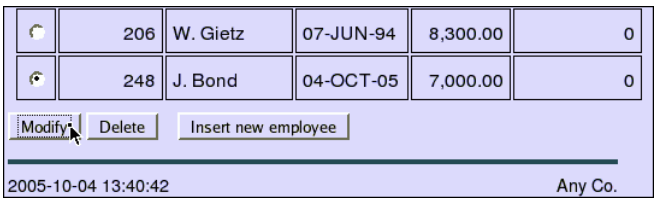
Click **Save**.



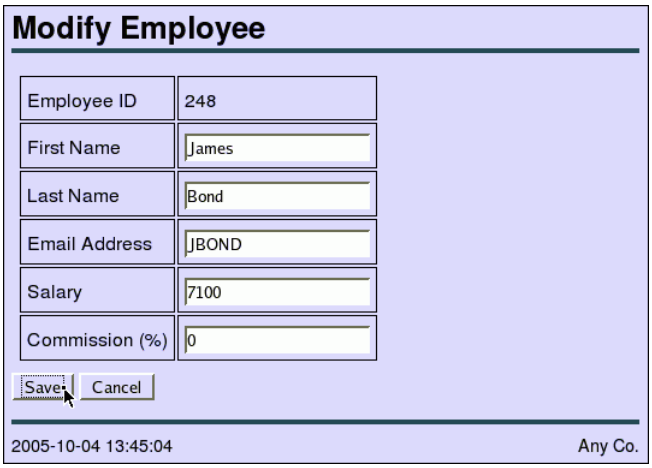
17. When the new employee record is successfully inserted, the Web page is refreshed with the form listing all employees. Scroll the Web page to the last record and check that the new employee record is present. The employee ID assigned to the new record on your system may be different than the one shown in the following example:



18. To modify the new employee record, select the radio button next to the new employee record, and click **Modify**:



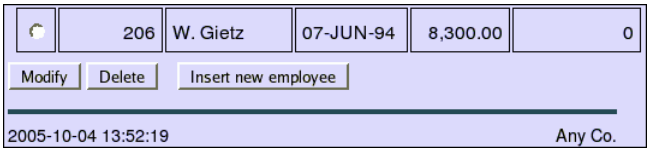
19. In the Modify Employee page, modify the Email Address field to JBOND, increase the Salary to 7100, and click **Save**:



20. Successfully updating the employee record causes the Employees page to be redisplayed. Scroll to the last employee record and confirm that the salary for James Bond is now 7,100:

21. To remove the new employee record, select the radio button for the new employee record, and click **Delete**:

On successful deletion, the deleted row does not appear in the list of employee records redisplayed in the Employees page:



Combining Departments and Employees

In this section, you will modify your application to enable access to both Employees and Departments pages.

To combine the Departments and Employees pages, perform the following tasks:

1. Edit the `anyco.php` file. Modify the query in the `construct_employees()` function to include a `WHERE` clause to compare the `department_id` with a value in a bind variable called `:did`. This makes the page display employees in one

department at a time. Get the deptid session parameter value to populate the bind variable:

```
$query =
 "SELECT employee_id,
        substr(first_name,1,1) || '.  '|| last_name as employee_name,
        hire_date,
        to_char(salary, '9999G999D99') as salary,
        nvl(commission_pct,0) as commission_pct
  FROM   employees
  WHERE  department_id = :did
  ORDER BY employee_id asc";

$deptid = $_SESSION['deptid'];
```

2. Edit the anyco.php file. In the construct_employees() function, update the call to the db_do_query() function to pass the bind information:

```
$conn = db_connect();

$bindargs = array();
array_push($bindargs, array('DID', $deptid, -1));

$emp = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_ROW, $bindargs);
```

3. Edit the anyco.php file. In the construct_departments() function, save the department identifier in a session parameter:

```
$_SESSION['currentdept'] = $current;
$_SESSION['deptid'] = $deptid;
```

This saves the current department identifier from the Departments page as a session parameter, which is used in the Employees page.

4. Edit the anyco.php file. Create a function get_dept_name() to query the department name for printing in the Departments and Employees page titles:

```
function get_dept_name($conn, $deptid)
{
  $query =
    'SELECT department_name
     FROM   departments
     WHERE  department_id = :did';

  $conn = db_connect();
  $bindargs = array();
  array_push($bindargs, array('DID', $deptid, -1));
  $dn = db_do_query($conn, $query,OCI_FETCHSTATEMENT_BY_COLUMN, $bindargs);

  return($dn['DEPARTMENT_NAME'][0]);
}
```

5. Edit the anyco.php file. Modify the construct_employees() function to print the department name in the Employees page heading:

```
$deptname = get_dept_name($conn, $deptid);
ui_print_header('Employees: '.$deptname);
```

6. Edit the anyco.php file. Modify the construct_departments() function to print the department name in the Departments page heading:

```
$deptname = get_dept_name($conn, $deptid);
```

```
    ui_print_header('Department: '.$deptname);
```

7. Edit the `anyco.php` file. Modify the `construct_insert_emp()` function so that the default department is obtained from the session parameter passed in the `$emp` array to the `ui_print_insert_employee()` function. The function becomes:

```
function construct_insert_emp()
{
  $deptid = $_SESSION['deptid'];

  $conn = db_connect();
  $query = "SELECT job_id, job_title FROM jobs ORDER BY job_title ASC";
  $jobs = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_COLUMN);
  $query = "SELECT sysdate FROM dual";
  $date = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_COLUMN);
  $emp = array(
    'DEPARTMENT_ID' => $deptid,
    'HIRE_DATE' => $date['SYSDATE'][0],
    'ALLJOBIDS' => $jobs['JOB_ID'],
    'ALLJOBTITLES' => $jobs['JOB_TITLE']
    );
  ui_print_header('Insert New Employee');
  ui_print_insert_employee($emp, $_SERVER['SCRIPT_NAME']);
  ui_print_footer(date('Y-m-d H:i:s'));
}
```

8. Edit the `anyco.php` file. Modify the final `else` statement in the HTML form handler. The handler becomes:

```
// Start form handler code
if (isset($_POST['insertemp'])) {
  construct_insert_emp();
}
elseif (isset($_POST['saveinsertemp'])) {
  insert_new_emp();
}
elseif (isset($_POST['modifyemp'])) {
  construct_modify_emp();
}
elseif (isset($_POST['savemodifiedemp'])) {
  modify_emp();
}
elseif (isset($_POST['deleteemp'])) {
  delete_emp();
}
elseif (   isset($_POST['showemp'])
        || isset($_POST['prevemp'])
        || isset($_POST['showemp'])) {
  construct_employees();
}
elseif (   isset($_POST['nextdept'])
        || isset($_POST['prevdept'])
        || isset($_POST['firstdept'])
        || isset($_POST['showdept'])) {
  construct_departments();
}
else {
  construct_departments();
}
```

9. Edit the `anyco_ui.php` file. In the `ui_print_department()` function, change the HTML form to enable it to call the Employees page:

```
...
<form method="post" action="$posturl">
<input type="submit" value="First" name="firstdept">
<input type="submit" value="< Previous" name="prevdept">
<input type="submit" value="Next >" name="nextdept">
   
<input type="submit" value="Show Employees" name="showemp">
</form>
...
```

10. Edit the `anyco_ui.php` file. In the `ui_print_employees()` function, change the HTML form to enable it to call the Departments page:

```
...
</table>
<input type="submit" value="Modify" name="modifyemp">
<input type="submit" value="Delete" name="deleteemp">
  
<input type="submit" value="Insert new employee" name="insertemp">
  
<input type="submit" value="Return to Departments" name="showdept">
</form>
...
```

11. Save the changes to your PHP files. In your browser, test the changes by entering the following URL:

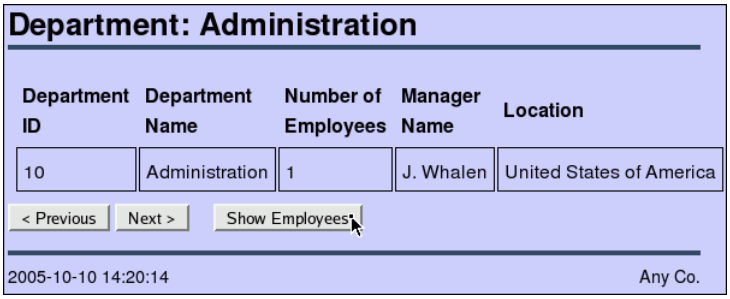
    On Windows:

    `http://localhost/chap5/anyco.php`

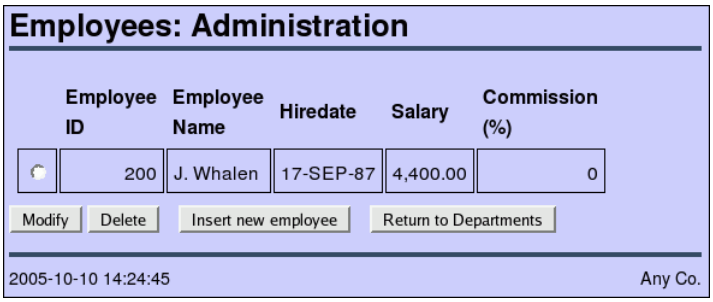
    On Linux:

    `http://localhost/~<username>/chap5/anyco.php`

    The Departments page is displayed.



    To display a list of employees in the department, click the **Show Employees** button.



You can return to the Departments page by clicking the **Return to Departments** button. Experiment by navigating to another department and listing its employees to show the process of switching between the Departments and Employees pages.

Adding Error Recovery

Error management is always a significant design decision. In production systems, you might want to classify errors and handle them in different ways. Fatal errors could be redirected to a standard "site not available" page or home page. Data errors for new record creation might return to the appropriate form with invalid fields highlighted.

In most production systems, you would set the `display_errors` configuration option in the `php.ini` file to `off`, and the `log_errors` configuration option to `on`.

You can use the PHP output buffering functionality to trap error text when a function is executing. Using `ob_start()` prevents text from displaying on the screen. If an error occurs, the `ob_get_contents()` function allows the previously generated error messages to be stored in a string for later display or analysis.

Now you change the application to display error messages and database errors on a new page using a custom error handling function. Errors are now returned from the `db*` functions keeping them silent.

1. Edit the `anyco_db.inc` file. Change the `db_error()` function to return the error information in an array structure, instead of printing and quitting. The function becomes:

```
function db_error($r = false, $file, $line)
{
  $err =  $r ? oci_error($r) : oci_error();

  if (isset($err['message'])) {
    $m = htmlentities($err['message']);
    $c = $err['code'];
  }
  else {
    $m = 'Unknown DB error';
    $c = null;
  }

  $rc = array(
    'MESSAGE' => $m,
    'CODE'    => $c,
    'FILE'    => $file,
    'LINE'    => $line
    );
  return $rc;
}
```

2. Edit the anyco_db.inc file. For every call to the db_error() function, assign the return value to a variable called $e and add a return false; statement after each call:

```
if (<error test>)
{
  $e = db_error(<handle>, __FILE__, __LINE__);
  return false;
}
```

Make sure to keep the <error test> and <handle> parameters the same as they are currently specified for each call. Remember that the __FILE__ and __LINE__ constants help to pinpoint the location of the failure during development. This is useful information to log for fatal errors in a production deployment of an application.

3. Edit the anyco_db.inc file. Add a $e parameter to every function to enable the return of error information. Use the & reference prefix to ensure that results are returned to the calling function. Each function declaration becomes:

```
function db_connect(&$e) {...}

function db_get_page_data($conn, $q1, $currrownum = 1, $rowsperpage = 1,
                          &$e, $bindvars = array()) {...}

function db_do_query($conn, $statement, $resulttype, &$e,
                     $bindvars = array()) {...}

function db_execute_statement($conn, $statement, &$e,
                              $bindvars = array()) {...}
```

4. Edit the anyco_db.inc file. In the db_get_page_data() function, change the call to the db_do_query() function to pass down the error parameter $e:

```
$r = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_ROW, $e, $bindvars);
```

5. Edit the anyco_db.inc file. Add an @ prefix to all oci_* function calls. For example:

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

The @ prefix prevents errors from displaying because each return result is tested. Preventing errors from displaying can hide incorrect parameter usage, which may hinder testing the changes in this section.

6. Edit the anyco.php file. Create a function to handle the error information:

```
function handle_error($message, $err)
{
  ui_print_header($message);
  ui_print_error($err, $_SERVER['SCRIPT_NAME']);
  ui_print_footer(date('Y-m-d H:i:s'));
}
```

7. Edit the anyco.php file. Modify all calls to db_* functions to include the additional error parameter:

Steps 8 to 15 show the complete new functions, so the code changes in this step can be skipped.

- Change all db_connect() calls to db_connect($err).

- Change all `db_do_query()` calls and insert a `$err` parameter as the fourth parameter. For example, the call in `construct_employees()` becomes:

```
$emp = db_do_query($conn, $query,
                   OCI_FETCHSTATEMENT_BY_ROW, $err, $bindargs);
```

  Change the other four db_do_query() calls in anyco.php remembering to keep the existing parameter values of each specific call.

- Change the `db_get_page_data()` call and insert a `$err` parameter as the fifth parameter:

```
$dept = db_get_page_data($conn, $query, $current, 1, $err);
```

- Change the `db_execute_statement()` calls and insert a `$err` parameter as the third parameter, for example:

```
$r = db_execute_statement($conn, $statement, $err, $bindargs);
```

8. Edit the `anyco.php` file. Modify the `construct_departments()` function to handle errors returned. The function becomes:

```
function construct_departments()
{
  if (isset($_SESSION['currentdept']) && isset($_POST['prevdept']) &&
           $_SESSION['currentdept'] > 1)
     $current = $_SESSION['currentdept'] - 1;
  elseif (isset($_SESSION['currentdept']) && isset($_POST['nextdept']))
     $current = $_SESSION['currentdept'] + 1;
  elseif (isset($_POST['showdept']) && isset($_SESSION['currentdept']))
     $current = $_SESSION['currentdept'];
  else
     $current = 1;

  $query =
    "SELECT d.department_id, d.department_name,
              substr(e.first_name,1,1)||'. '|| e.last_name as manager_name,
              c.country_name, count(e2.employee_id) as number_of_employees
     FROM   departments d, employees e, locations l,
            countries c, employees e2
     WHERE  d.manager_id    = e.employee_id
     AND    d.location_id   = l.location_id
     AND    d.department_id = e2.department_id
     AND    l.country_id    = c.country_id
     GROUP BY d.department_id, d.department_name,
                substr(e.first_name,1,1)||'. '||e.last_name, c.country_name
     ORDER BY d.department_id ASC";

  $conn = db_connect($err);

  if (!$conn) {
    handle_error('Connection Error', $err);
  }
  else {
    $dept = db_get_page_data($conn, $query, $current, 1, $err);
    if ($dept === false) {
      // Use === so empty array at end of fetch is not matched
      handle_error('Cannot fetch Departments', $err);
    } else {

      if (!isset($dept[0]['DEPARTMENT_ID']) && $current > 1) {
        // no more records so go back one
```

```
        $current--;
        $dept = db_get_page_data($conn, $query, $current, 1, $err);
      }

      $deptid = $dept[0]['DEPARTMENT_ID'];

      $_SESSION['deptid'] = $deptid;
      $_SESSION['currentdept'] = $current;

      $deptname = get_dept_name($conn, $deptid);
      ui_print_header('Department: '.$deptname);
      ui_print_department($dept[0], $_SERVER['SCRIPT_NAME']);
      ui_print_footer(date('Y-m-d H:i:s'));
    }
  }
}
```

9. Edit the `anyco.php` file. Modify the `construct_employees()` function to handle errors. The function becomes:

```
function construct_employees()
{
  $query =
    "SELECT employee_id,
            substr(first_name,1,1) || '.  '|| last_name as employee_name,
            hire_date,
            to_char(salary, '9999G999D99') as salary,
            nvl(commission_pct,0) as commission_pct
     FROM   employees
     WHERE  department_id = :did
     ORDER BY employee_id asc";

  $deptid = $_SESSION['deptid'];

  $conn = db_connect($err);

  if (!$conn) {
    handle_error('Connection Error', $err);
  }
  else {
    $bindargs = array();
    array_push($bindargs, array('DID', $deptid, -1));
    $emp = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_ROW, $err,
    $bindargs);

    if (!$emp) {
      handle_error('Cannot fetch Employees', $err);
    }
    else {
      $deptname = get_dept_name($conn, $deptid);
      ui_print_header('Employees: '.$deptname);
      ui_print_employees($emp, $_SERVER['SCRIPT_NAME']);
      ui_print_footer(date('Y-m-d H:i:s'));
    }
  }
}
```

10. Edit the `anyco.php` file. Modify the `construct_insert_emp()` function to handle errors. The function becomes:

```
function construct_insert_emp()
{
  $deptid = $_SESSION['deptid'];
  $conn = db_connect($err);
  if (!$conn) {
    handle_error('Connection Error', $err);
  }
  else {
    $query = "SELECT job_id, job_title FROM jobs ORDER BY job_title ASC";
    $jobs = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_COLUMN, $err);
    $query = "SELECT sysdate FROM dual";
    $date = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_COLUMN, $err);

    $emp = array(
      'DEPARTMENT_ID' => $deptid,
      'HIRE_DATE' => $date['SYSDATE'][0],
      'ALLJOBIDS' => $jobs['JOB_ID'],
      'ALLJOBTITLES' => $jobs['JOB_TITLE']
      );

    ui_print_header('Insert New Employee');
    ui_print_insert_employee($emp, $_SERVER['SCRIPT_NAME']);
    ui_print_footer(date('Y-m-d H:i:s'));
  }
}
```

11. Edit the `anyco.php` file. Modify the `insert_new_emp()` function to handle errors. The function becomes:

```
function insert_new_emp()
{
  $statement =
    'INSERT INTO employees
                (employee_id, first_name, last_name, email, hire_date,
                job_id, salary, commission_pct, department_id)
     VALUES (employees_seq.nextval, :fnm, :lnm, :eml, :hdt,
            :jid, :sal, :cpt, :did)';

  $newemp = $_POST;

  $conn = db_connect($err);
  if (!$conn) {
    handle_error('Connect Error', $err);
  }
  else {
    $emailid = $newemp['firstname'].$newemp['lastname'];

    $bindargs = array();
    array_push($bindargs, array('FNM', $newemp['firstname'], -1));
    array_push($bindargs, array('LNM', $newemp['lastname'], -1));
    array_push($bindargs, array('EML', $emailid, -1));
    array_push($bindargs, array('HDT', $newemp['hiredate'], -1));
    array_push($bindargs, array('JID', $newemp['jobid'], -1));
    array_push($bindargs, array('SAL', $newemp['salary'], -1));
    array_push($bindargs, array('CPT', $newemp['commpct'], -1));
    array_push($bindargs, array('DID', $newemp['deptid'], -1));

    $r = db_execute_statement($conn, $statement, $err, $bindargs);
    if ($r) {
      construct_employees();
    }
```

```
    else {
      handle_error('Cannot insert employee', $err);
    }
  }
}
```

12. Edit the `anyco.php` function. Modify the `construct_modify_emp()` function to handle errors. The function becomes:

```
function construct_modify_emp()
{
  if (!isset($_POST['emprec'])) { // User did not select a record
    construct_employees();
  }
  else {
    $empid = $_POST['emprec'];

    $query =
      "SELECT employee_id, first_name, last_name, email, hire_date,
              salary, nvl(commission_pct,0) as commission_pct
       FROM   employees
       WHERE  employee_id = :empid";

    $conn = db_connect($err);
    if (!$conn) {
      handle_error('Connect Error', $err);
    }
    else {
      $bindargs = array();
      array_push($bindargs, array('EMPID', $empid, -1));

      $emp = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_ROW, $err,
             $bindargs);

      if (!$emp) {
        handle_error('Cannot find details for employee '.$empid, $err);
      }
      else {
        ui_print_header('Modify Employee ');
        ui_print_modify_employee($emp[0], $_SERVER['SCRIPT_NAME']);
        ui_print_footer(date('Y-m-d H:i:s'));
      }
    }
  }
}
```

13. Edit the `anyco.php` file. Change the `modify_emp()` function to handle errors. The function becomes:

```
function modify_emp()
{
  $newemp = $_POST;

  $statement =
    "UPDATE employees
     SET    first_name = :fnm, last_name = :lnm, email = :eml,
            salary = :sal, commission_pct = :cpt
     WHERE  employee_id = :eid";

  $conn = db_connect($err);
  if (!$conn) {
```

```
    handle_error('Connect Error', $err);
  }
  else {
    $bindargs = array();
    array_push($bindargs, array('EID', $newemp['empid'], -1));
    array_push($bindargs, array('FNM', $newemp['firstname'], -1));
    array_push($bindargs, array('LNM', $newemp['lastname'], -1));
    array_push($bindargs, array('EML', $newemp['email'], -1));
    array_push($bindargs, array('SAL', $newemp['salary'], -1));
    array_push($bindargs, array('CPT', $newemp['commpct'], -1));

    $r = db_execute_statement($conn, $statement, $err, $bindargs);

    if (!$r) {
      handle_error('Cannot update employee '.$newemp['empid'], $err);
    }
    else {
      construct_employees();
    }
  }
}
```

14. Edit the `anyco.php` file. Modify the `delete_emp()` function to handle errors. The function becomes:

```
function delete_emp()
{
  if (!isset($_POST['emprec'])) { // User did not select a record
    construct_employees();
  }
  else {
    $empid = $_POST['emprec'];

    $conn = db_connect($err);
    if (!$conn) {
      handle_error('Connection Error', $err);
    }
    else {
      $statement = "DELETE FROM employees WHERE employee_id = :empid";
      $bindargs = array();
      array_push($bindargs, array('EMPID', $empid, -1));
      $r = db_execute_statement($conn, $statement, $err, $bindargs);

      if (!$r) {
        handle_error("Error deleting employee $empid", $err);
      }
      else {
        construct_employees();
      }
    }
  }
}
```

15. Edit the `anyco.php` file. Modify the `get_dept_name()` function to handle errors. The function becomes:

```
function get_dept_name($conn, $deptid)
{
  $query =
    'SELECT department_name
     FROM   departments
```

```
        WHERE  department_id = :did';

    $conn = db_connect($err);
    if (!$conn) {
      return ('Unknown');
    }
    else {
      $bindargs = array();
      array_push($bindargs, array('DID', $deptid, -1));
      $dn = db_do_query($conn, $query, OCI_FETCHSTATEMENT_BY_COLUMN,
                        $err, $bindargs);
      if ($dn == false)
        return ('Unknown');
      else
        return($dn['DEPARTMENT_NAME'][0]);
    }
  }
```

16. Edit the `anyco_ui.inc` file. Add a new function `ui_print_error()`:

```
function ui_print_error($message, $posturl)
{
  if (!$message) {
    echo '<p>Unknown error</p>';
  }
  else {
    echo "<p>Error at line {$message['LINE']} of "
         ."{$message['FILE']}</p>";  // Uncomment for debugging
    echo "<p>{$message['MESSAGE']}</p>";
  }
  echo <<<END
  <form method="post" action="$posturl">
  <input type="submit" value="Return to Departments" name="showdept">
END;
}
```

17. Save the changes to your application files. Test the changes by entering the following URL in your browser:

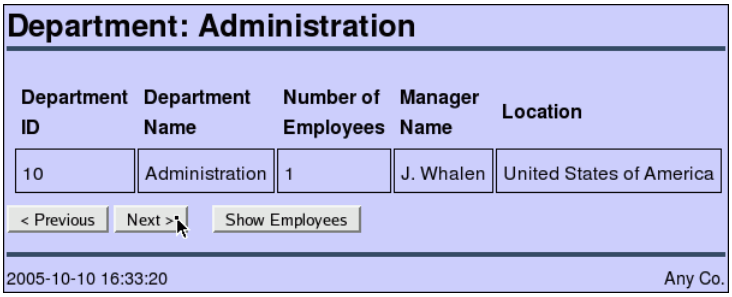
    On Windows:

    `http://localhost/chap5/anyco.php`

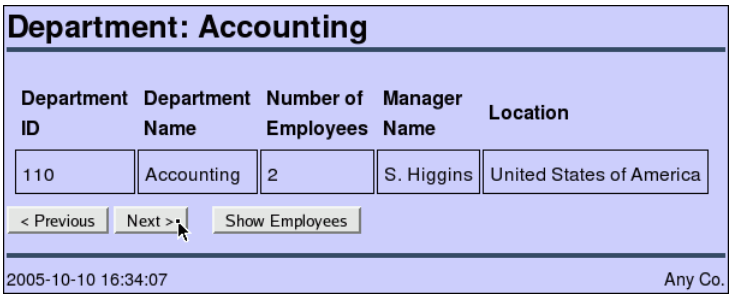
    On Linux:

    `http://localhost/~<username>/chap5/anyco.php`

    The Departments page is displayed:



18. Click **Next** to navigate to the last department record, the Accounting department with ID 110. Try to navigate past the last department record by clicking **Next**.



The error handling prevents navigation past the last department record.

19. If a new employee is inserted with a salary of 0, or the department ID is changed to one that does not exist, the new error page is shown with the heading "Cannot insert employee".

Further Error Handling

Specific Oracle errors can be handled individually. For example, if a new employee record is created by clicking the **Insert new employee** button on the Employees page, and the Department ID is changed to a department that does not exist, you can trap this error and display a more meaningful message:

1. Edit the `anyco.php` file. Change the error handling in the `insert_new_emp()` function:

```
$r = db_execute_statement($conn, $statement, $err, $bindargs);
if ($r) {
  construct_employees();
}
else {
  if ($err['CODE'] == 2291) {  // Foreign key violated
    handle_error("Department {$newemp['deptid']} does not yet exist",
    $err);
  }
  else {
    handle_error('Cannot insert employee', $err);
  }
}
```

2. Save the changes to your application files. Test the changes by entering the following URL:

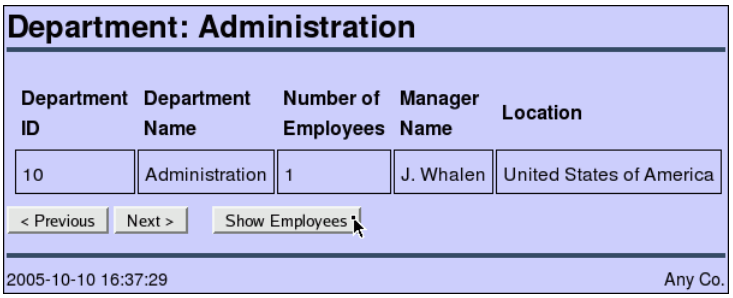
   On Windows:

   `http://localhost/chap5/anyco.php`

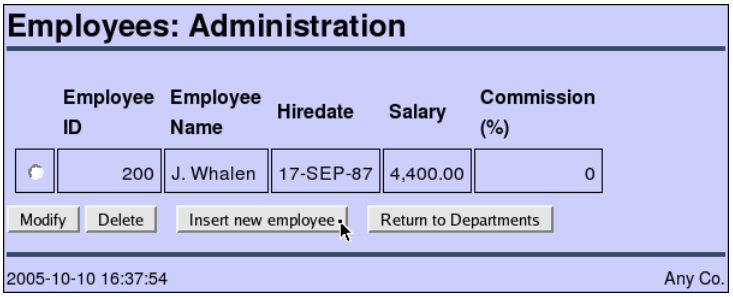
   On Linux:

   `http://localhost/~<username>/chap5/anyco.php`

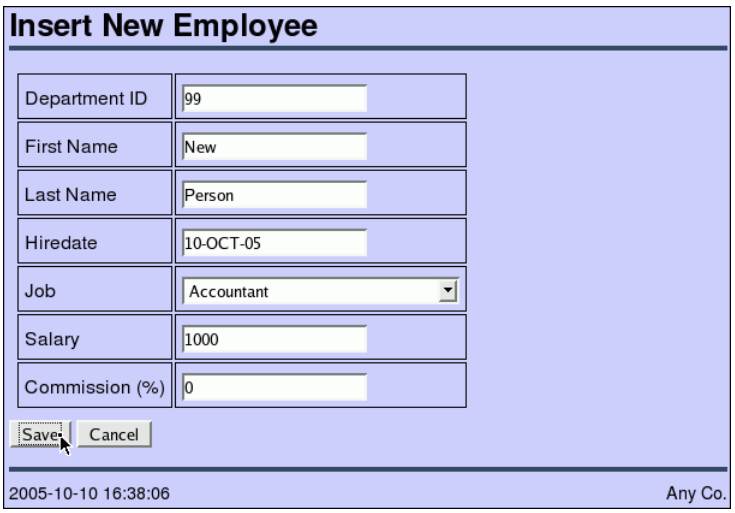
3. In the Departments page, click **Show Employees**.



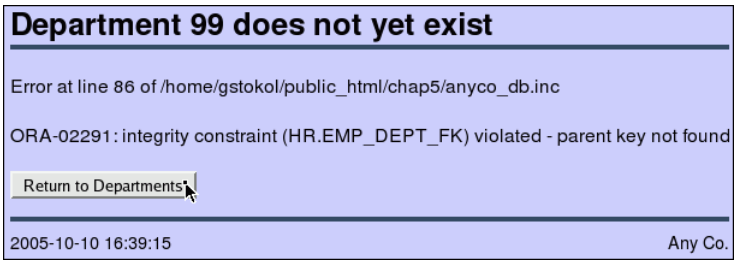
4. In the Employees page, click **Insert new employee**.



5. In the Insert New Employee page, enter employee details as shown, setting the Department ID to 99, and click **Save**.



The following error page is displayed:



You can click **Return to Departments** to return to the Departments page and then click **Show Employees** to verify that the new employee record has not been added to the Administration department.

6

Executing Stored Procedures and Functions

This chapter shows you how to run stored procedures and functions using PHP and Oracle Database Express Edition (Oracle Database XE). It has the following topics:

- Using PL/SQL to Capture Business Logic
- Using PL/SQL Ref Cursors to Return Result Sets

The Anyco application is extended with a PL/SQL function to calculate remuneration for each employee, and is further extended with a PL/SQL procedure to return a REF CURSOR of employee records.

Using PL/SQL to Capture Business Logic

Oracle PL/SQL procedures and functions enable you to store business logic in the database for any client program to use. They also reduce the amount of data that must be transferred between the database and PHP.

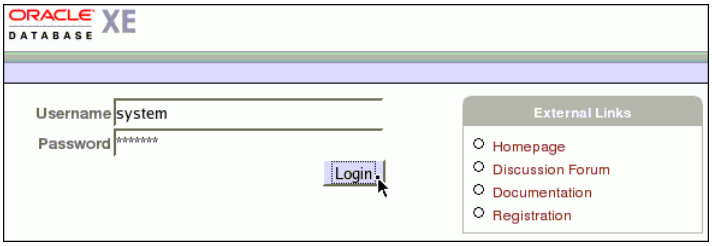
In this section, you will create a PL/SQL stored function to calculate and display the total remuneration for each employee.

To display the total remuneration of each employee, perform the following steps:

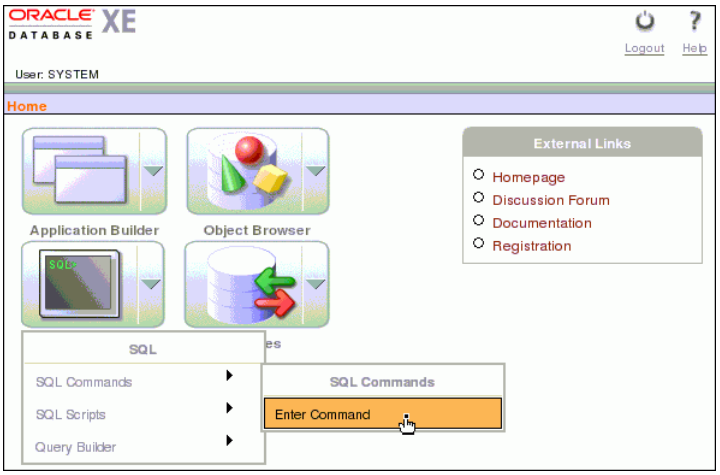
1. In a browser, enter the URL for your Oracle Database Express Edition home page:

   ```
   http://localhost:8080/htmldb
   ```

2. At the login screen, in the Username field enter `system`, and in the Password field enter `manager` (or the password you entered at the prompt during configuration of Oracle Database XE). Click **Login**.



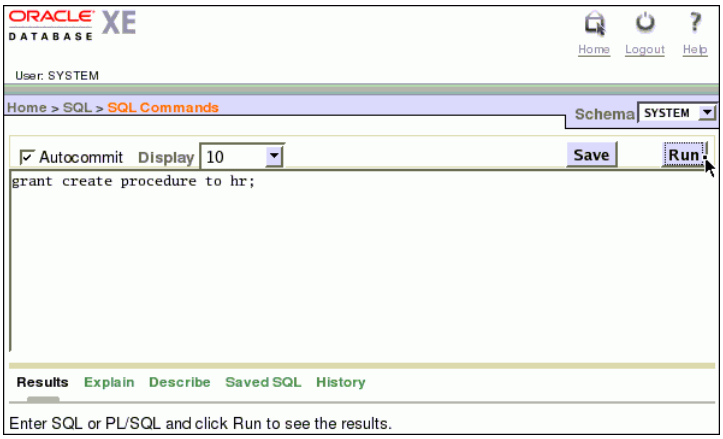
3. In the Home page, click the arrow on the **SQL** icon, move the mouse over **SQL Commands**, and click **Enter Command**:



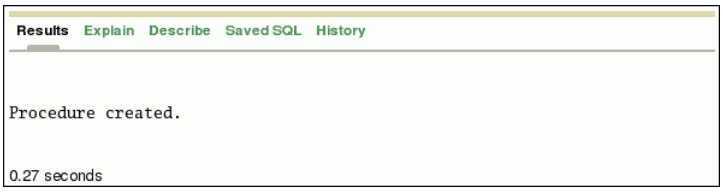
4. In the SQL Commands page, to assign the `create procedure` privilege to the `HR` user, enter the following `grant` command:

   ```
   grant create procedure to hr;
   ```

   Click **Run**:

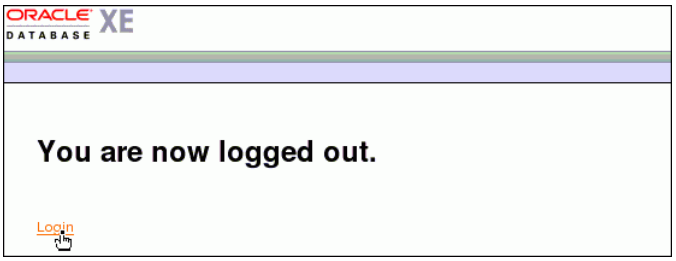


   A message similar to the following appears in the Results section below the text area where the command was entered:

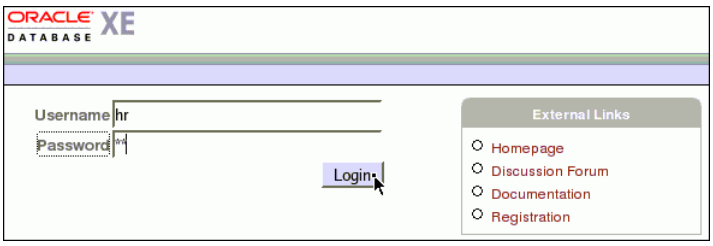


5. Click the **Logout** link to terminate the HTMLDB session.

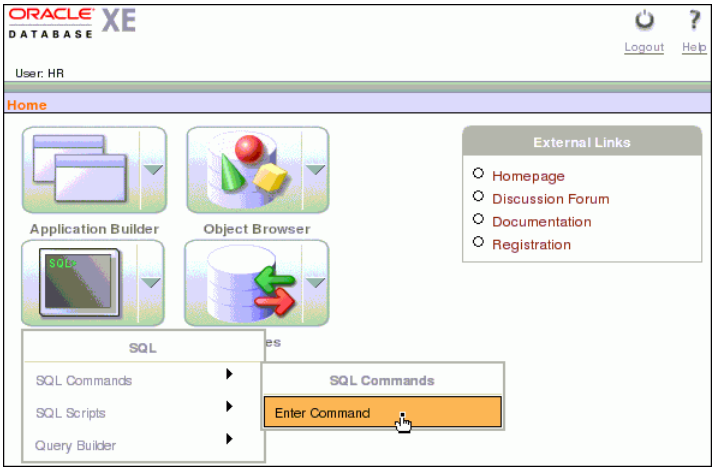
6. In the Logout Confirmation page, click the **Login** link:



7. In the Oracle Database XE Login page, enter `hr` in the Username and Password fields. Click **Login**:



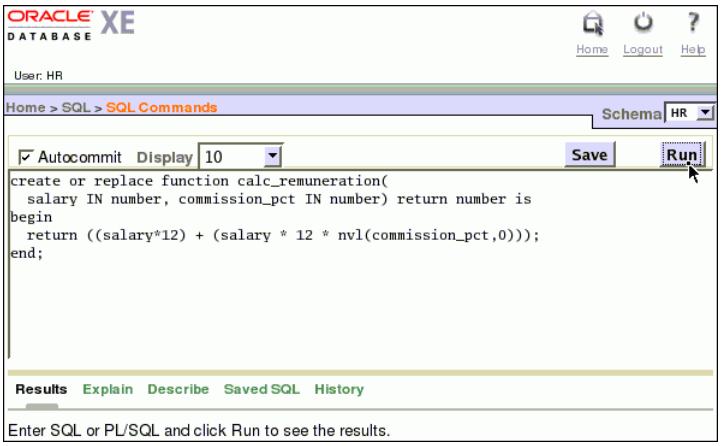
8. In the Home page, click the arrow on the **SQL** icon, move the mouse over **SQL Commands**, and click **Enter Command**:



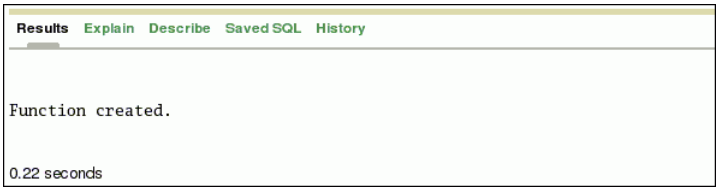
9. In the SQL Commands page, enter the following text to create a `calc_remuneration()` function:

```
create or replace function calc_remuneration(
  salary IN number, commission_pct IN number) return number is
begin
  return ((salary*12) + (salary * 12 * nvl(commission_pct,0)));
end;
```

Click **Run**:



In the results window, confirm that the function is created:



10. Create the `chap6` directory, copy the application files from `chap5`, and change to the newly created directory:

    On Windows:

    ```
    mkdir c:\program files\Apache Group\Apache2\htdocs\chap6
    cd c:\program files\Apache Group\Apache2\htdocs\chap6
    copy ..\chap5\* .
    ```

    On Linux:

    ```
    mkdir $HOME/public_html/chap6
    cd $HOME/public_html/chap6
    cp ../chap5/* .
    ```

11. Edit the `anyco.php` file. Modify the query in the `construct_employees()` function to call the PL/SQL function for each row returned:

    ```
    $query =
     "SELECT employee_id,
             substr(first_name,1,1) || '. '|| last_name as employee_name,
             hire_date,
             to_char(salary, '9999G999D99') as salary,
             nvl(commission_pct,0) as commission_pct,
             to_char(calc_remuneration(salary, commission_pct),'9999G999D99')
               as remuneration
      FROM employees
      WHERE department_id = :did
      ORDER BY employee_id ASC";
    ```

12. Edit the `anyco_ui.inc` file. In the `ui_print_employees()` function, add a `Remuneration` column to the table, and modify the `foreach` loop to display the remuneration field for each employee:

    ```
    echo <<<END
       <form method="post" action="$posturl">
    ```

```
    <table>
    <tr>
      <th> </th>
      <th>Employee<br>ID</th>
      <th>Employee<br>Name</th>
      <th>Hiredate</th>
      <th>Salary</th>
      <th>Commission<br>(%)</th>
      <th>Remuneration</th>
    </tr>
END;

    // Write one row per employee
    foreach ($employeerecords as $emp) {
      echo '<tr>';
      echo '<td><input type="radio" name="emprec"
                value="'.htmlentities($emp['EMPLOYEE_ID']).'"></td>';
      echo '<td align="right">'.htmlentities($emp['EMPLOYEE_ID']).'</td>';
      echo '<td>'.htmlentities($emp['EMPLOYEE_NAME']).'</td>';
      echo '<td>'.htmlentities($emp['HIRE_DATE']).'</td>';
      echo '<td align="right">'.htmlentities($emp['SALARY']).'</td>';
      echo '<td align="right">'.htmlentities($emp['COMMISSION_PCT']).'</td>';
      echo '<td align="right">'.htmlentities($emp['REMUNERATION']).'</td>';
      echo '</tr>';
    }
```

13. Save the changes to your application files. In a browser, enter the following URL to test the application:

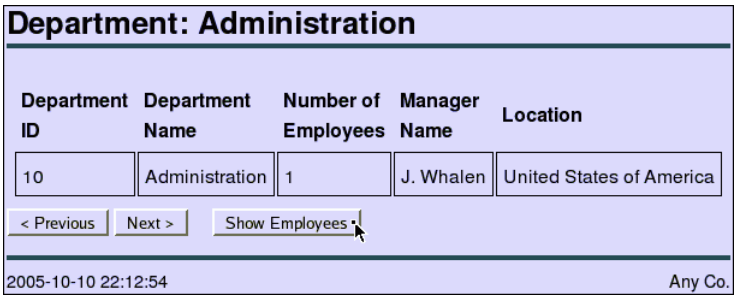
    On Windows:

    ```
    http://localhost/chap6/anyco.php
    ```

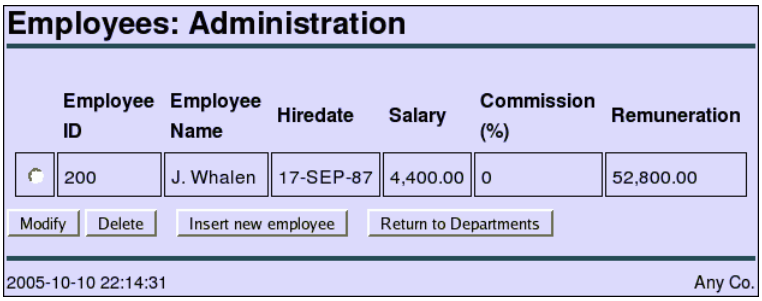
    On Linux:

    ```
    http://localhost/~<username>/chap6/anyco.php
    ```

14. In the Departments page, click **Show Employees**.



    In the Employees page for the department, the employee remuneration is displayed in the last column:



Using PL/SQL Ref Cursors to Return Result Sets

Data sets can be returned as REF CURSORS from PL/SQL blocks in a PHP script. This can be useful where the data set requires complex functionality.

A REF CURSOR in PL/SQL is a type definition that is assigned to a cursor variable. It is common to declare a PL/SQL type inside a package specification for reuse in other PL/SQL constructs, such as a package body.

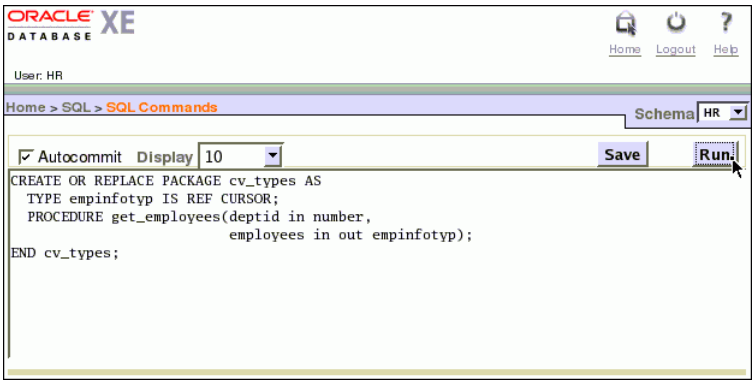
In this section, you will use a REF CURSOR to retrieve the employees for a specific department.

To create a PL/SQL package specification and body, with a REF CURSOR to retrieve employees for a specific department, perform the following steps:

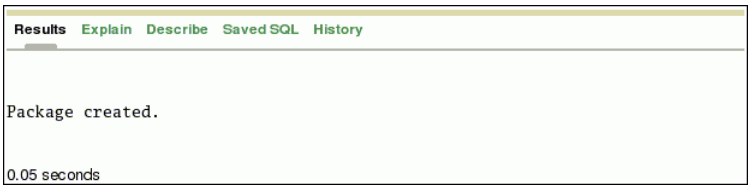
1. In the SQL Commands page, as the `HR` user, create the following PL/SQL package specification:

```
CREATE OR REPLACE PACKAGE cv_types AS
  TYPE empinfotyp IS REF CURSOR;
  PROCEDURE get_employees(deptid in number,
                          employees in out empinfotyp);
END cv_types;
```

   Click **Run**:



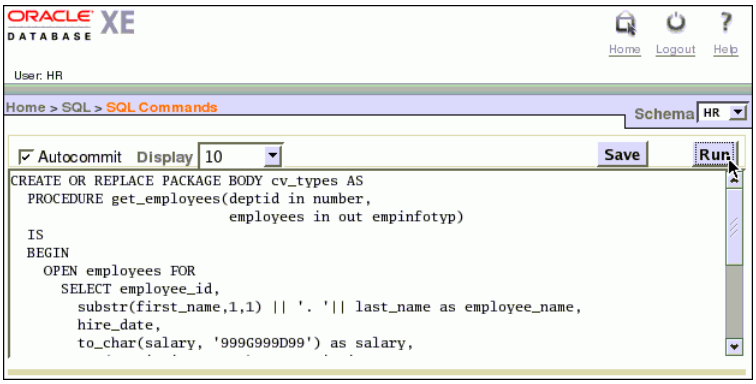
   In the Results section, confirm that the package specification is successfully created:



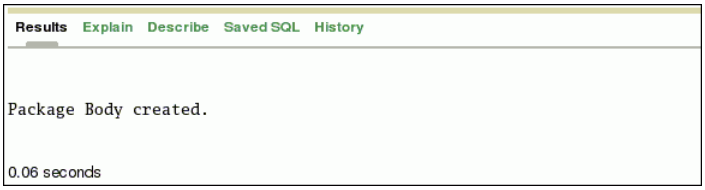
2. In the SQL Commands page, as the `HR` user, create the PL/SQL package body (implementation):

```
CREATE OR REPLACE PACKAGE BODY cv_types AS
  PROCEDURE get_employees(deptid in number,
                          employees in out empinfotyp)
  IS
  BEGIN
    OPEN employees FOR
      SELECT employee_id,
        substr(first_name,1,1) || '. '|| last_name as employee_name,
        hire_date,
        to_char(salary, '999G999D99') as salary,
        NVL(commission_pct,0) as commission_pct,
        to_char(calc_remuneration(salary, commission_pct),
              '9999G999D99') as remuneration
      FROM employees
      WHERE department_id = deptid
      ORDER BY employee_id ASC;
  END get_employees;
END cv_types;
```

Click **Run**:



In the Results section, confirm that the package body is successfully created:



3. Edit the `anyco_db.inc` file. Create a new PHP function that calls the PL/SQL packaged procedure:

```
// Use ref cursor to fetch employee records
// All records are retrieved - there is no paging in this example
function db_get_employees_rc($conn, $deptid, &$e)
{
  // Execute the call to the stored procedure
  $stmt = "BEGIN cv_types.get_employees($deptid, :rc); END;";
  $stid = @oci_parse($conn, $stmt);
  if (!$stid) {
    $e = db_error($conn, __FILE__, __LINE__);
    return false;
  }
```

```
  $refcur = oci_new_cursor($conn);
  if (!$stid) {
    $e = db_error($conn, __FILE__, __LINE__);
    return false;
  }
  $r = @oci_bind_by_name($stid, ':RC', $refcur, -1, OCI_B_CURSOR);
  if (!$r) {
    $e = db_error($stid, __FILE__, __LINE__);
    return false;
  }
  $r = @oci_execute($stid);
  if (!$r) {
    $e = db_error($stid, __FILE__, __LINE__);
    return false;
  }
  // Now treat the ref cursor as a statement resource
  $r = @oci_execute($refcur, OCI_DEFAULT);
  if (!$r) {
    $e = db_error($refcur, __FILE__, __LINE__);
    return false;
  }
  $r = @oci_fetch_all($refcur, $employeerecords, null, null,
                      OCI_FETCHSTATEMENT_BY_ROW);
  if (!$r) {
    $e = db_error($refcur, __FILE__, __LINE__);
    return false;
  }
  return ($employeerecords);
}
```

The `db_get_employees_rc()` function executes the following anonymous (unnamed) PL/SQL block:

```
BEGIN cv_types.get_employees($deptid, :rc); END;
```

The PL/SQL statement inside the BEGIN END block calls the stored PL/SQL package procedure `cv_types.et_employees()`. This returns an `OCI_B_CURSOR` REF CURSOR bind variable in the PHP variable `$refcur`.

The `$refcur` variable is treated as a statement handle that is used for execute and fetch operations.

4. Edit the `anyco.php` file. In the `construct_employees()` function, remove the query text and the bind arguments. The function becomes:

```
function construct_employees()
{
  $deptid = $_SESSION['deptid'];
  $conn = db_connect($err);
  if (!$conn) {
    handle_error('Connection Error', $err);
  }
  else {
    $emp = db_get_employees_rc($conn, $deptid, $err);

    if (!$emp) {
      handle_error('Cannot fetch Employees', $err);
    }
    else {
      $deptname = get_dept_name($conn, $deptid);
```

```
            ui_print_header('Employees: '.$deptname);
            ui_print_employees($emp, $_SERVER['SCRIPT_NAME']);
            ui_print_footer(date('Y-m-d H:i:s'));
          }
        }
      }
```

5. Save the changes to your application files. In a browser, enter the following URL to test the application:

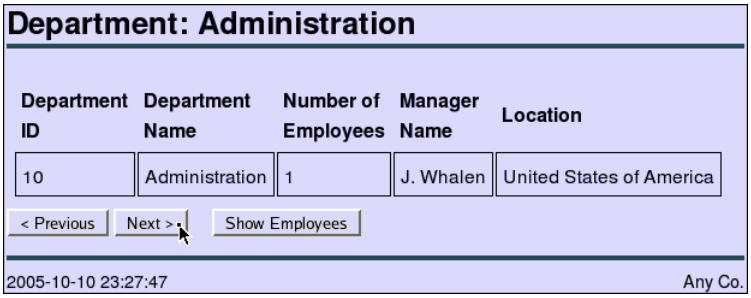
   On Windows:

   ```
   http://localhost/chap6/anyco.php
   ```

   On Linux:

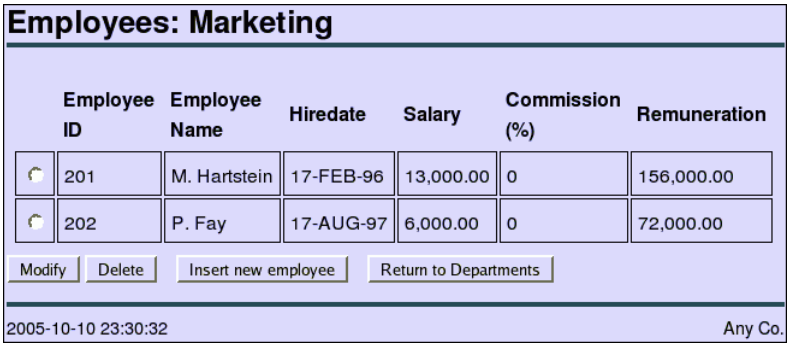
   ```
   http://localhost/~<username>/chap6/anyco.php
   ```

6. In the Departments page, click **Next** to navigate to the Marketing department page.



7. In the Marketing department page, click **Show Employees**.

   In the Employees page for the Marketing department, the employee records remuneration is displayed in the last column:



8. In the SQL Commands page, to log out of the HR database session, click the **Logout** link.

7

Loading Images

This chapter shows you how to change the application to upload a JPEG image for new employee records and display it on the Employees page. It has the following topics:

- Using BLOBs to Store and Load Employee Images
- Resizing Images

Using BLOBs to Store and Load Employee Images

In this section, you will modify your application code to enable a photo to be stored in the record of an employee.

To enable images of employees to be stored in the employee records, perform the following tasks:

1. Create the `chap7` directory, copy the application files from `chap6`, and change to the newly created directory:

   On Windows:

   ```
   mkdir c:\program files\Apache Group\Apache2\htdocs\chap7
   cd c:\program files\Apache Group\Apache2\htdocs\chap7
   copy ..\chap6\* .
   ```

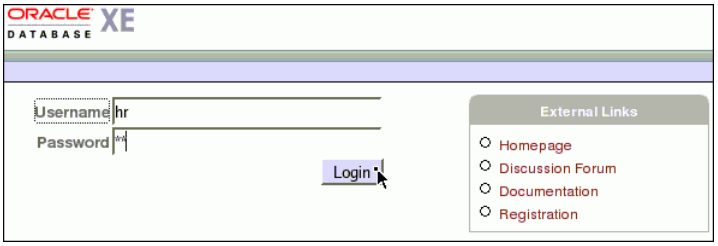
   On Linux:

   ```
   mkdir $HOME/public_html/chap7
   cd $HOME/public_html/chap7
   cp ../chap6/* .
   ```

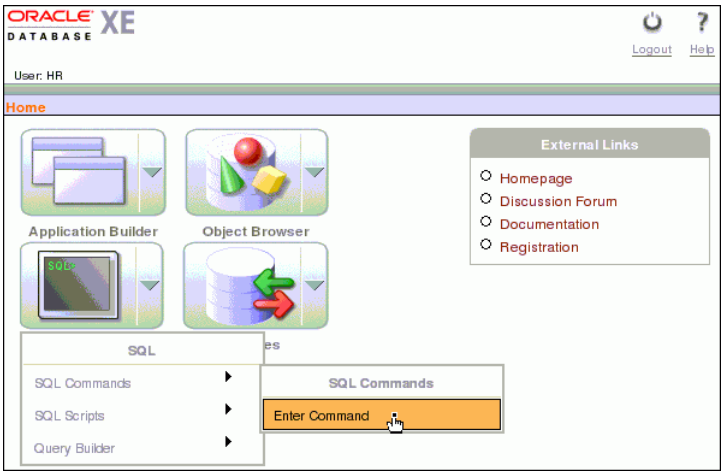
2. In a browser, enter the URL to access the Oracle Database Express Edition HTMLDB Web page:

   ```
   http://localhost:8080/htmldb
   ```

3. In the Oracle Database XE Login page, enter `hr` in the Username and Password fields. Click **Login**:



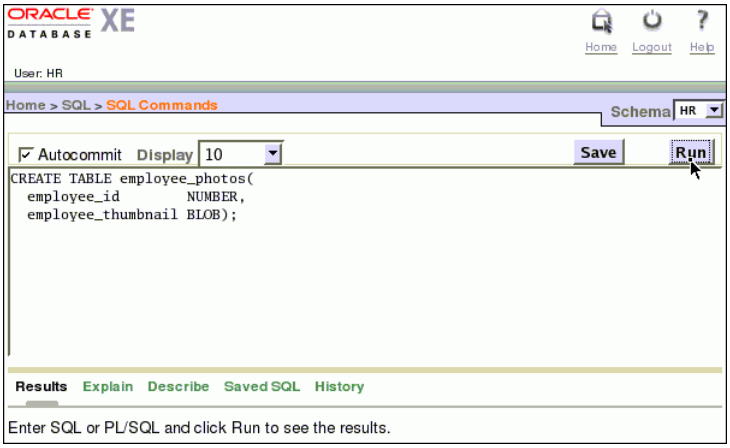
4. In the Home page, to create a new table for storing employee images, click the arrow on the **SQL** icon, highlight **SQL Commands**, and click **Enter Command**:



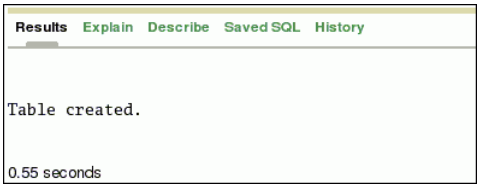
5. In the **SQL Commands** page, enter the following `CREATE TABLE` statement:

```
CREATE TABLE employee_photos(
  employee_id        NUMBER,
  employee_thumbnail BLOB);
```

Click **Run**:



6. In the Results section below the command text area, confirm that the table is successfully created:



The `HR` user must have the `CREATE TABLE` privilege to perform this command. If you get an "insufficient privileges" error message, then log out as the `HR` user, log in as `system` with password `manager`, and execute the following `GRANT` command:

```
GRANT create table TO hr;
```

Then log in as HR again to execute the CREATE TABLE statement.

7. Edit the anyco_ui.inc file. Add a Photograph column to the EMPLOYEES table in the ui_print_employees() function:

```
<th>Commission<br>(%)</th>
<th>Remuneration</th>
<th>Photograph</th>
```

The data for the Photograph column is populated with an <img> tag whose src attribute is defined as a URL reference to a new anyco_im.php file, which will display the image for each employee record.

8. Edit the anyco_ui.inc file. Add code in the ui_print_employees() function to generate an <img> tag referencing the anyco_im.php file with the employee identifier as a parameter:

```
  echo '<td align="right">'
         .htmlentities($emp['REMUNERATION']).'</td>';
  echo '<td><img src="anyco_im.php?showempphoto='.$emp['EMPLOYEE_ID']
         .'" alt="Employee photo"></td>';
```

9. Edit the anyco_ui.inc file. To enable images to be uploaded when a new employee record is created, add an enctype attribute to the <form> tag in the ui_print_insert_employee() function:

```
<form method="post" action="$posturl" enctype="multipart/form-data">
```

At the bottom of the form add an upload field with an input type of file:

```
<tr>
  <td>Commission (%)</td>
  <td><input type="text" name="commpct" value="0" size="20"></td>
</tr>
<tr>
  <td>Photo</td>
  <td><input type="file" name="empphoto"></td>
</tr>
```

10. Create the anyco_im.php file. This file accepts an employee identifier as a URL parameter, reads the image from the Photograph column for that employee record, and returns the thumbnail image to be displayed:

```
<?php    // anyco_im.php

require('anyco_cn.inc');
require('anyco_db.inc');
construct_image();

function construct_image()
{
  if (!isset($_GET['showempphoto'])) {
    return;
  }

  $empid = $_GET['showempphoto'];

  $conn = db_connect($err);

  if (!$conn) {
    return;
  }
```

```
  $query =
    'SELECT employee_thumbnail
     FROM   employee_photos
     WHERE  employee_id = :eid';

  $stid = oci_parse($conn, $query);
  $r = oci_bind_by_name($stid, ":eid", $empid, -1);
  if (!$r) {
    return;
  }
  $r = oci_execute($stid, OCI_DEFAULT);
  if (!$r) {
    return;
  }

  $arr = oci_fetch_row($stid);
  if (!$arr) {
    return;                     // photo not found
  }

  $result = $arr[0]->load();

  // If any text (or whitespace!) is printed before this header is sent,
  // the text is not displayed. The image also is not displayed properly.
  // Comment out the "header" line to see the text and debug.
  header("Content-type: image/JPEG");
  echo $result;
}

?>
```

The `construct_image()` function uses the `OCI-Lob->load()` function to retrieve the Oracle LOB data, which is the image data. The PHP `header()` function sets the MIME type in the HTTP response header to ensure the browser interprets the data as a JPEG image.

If you want to display other image types, then the `Content-type` needs to be changed accordingly.

11. Edit the `anyco_db.inc` file. Add a new function `db_insert_thumbnail()` to insert an image into the `EMPLOYEE_PHOTOS` table:

```
function db_insert_thumbnail($conn, $empid, $imgfile, &$e)
{
  $lob = oci_new_descriptor($conn, OCI_D_LOB);
  if (!$lob) {
    $e = db_error($conn, __FILE__, __LINE__);
    return false;
  }

  $insstmt =
    'INSERT INTO employee_photos (employee_id, employee_thumbnail)
     VALUES(:eid, empty_blob())
     RETURNING employee_thumbnail into :etn';

  $stmt = oci_parse($conn, $insstmt);
  $r = oci_bind_by_name($stmt, ':etn', $lob, -1, OCI_B_BLOB);
  if (!$r) {
    $e = db_error($stid, __FILE__, __LINE__);
    return false;
```

```
  }
  $r = oci_bind_by_name($stmt, ':eid', $empid, -1);
  if (!$r) {
    $e = db_error($stid, __FILE__, __LINE__);
    return false;
  }
  $r = oci_execute($stmt, OCI_DEFAULT);
  if (!$r) {
    $e = db_error($stid, __FILE__, __LINE__);
    return false;
  }

  if (!$lob->savefile($imgfile)) {
    $e = db_error($stid, __FILE__, __LINE__);
    return false;
  }
  $lob->free();

  return true;
}
```

To tie the new EMPLOYEE_PHOTOS and EMPLOYEES tables together, you must use the same employee id in both tables.

12. Edit the anyco_db.inc file. Change the $bindvars parameter in the db_execute_statement() function to &$bindvars so that OUT bind variable values are returned from the database. At the bottom of the function, add a loop to set any return bind values:

```
function db_execute_statement($conn, $statement, &$e, &$bindvars = array())
{
  ...

  $r = @oci_execute($stid);
  if (!$r) {
    $e = db_error($stid, __FILE__, __LINE__);
    return false;
  }
  $outbinds = array();
  foreach ($bindvars as $b) {
    $outbinds[$b[0]] = $$b[0];
  }
  $bindvars = $outbinds;
  return true;
}
```

13. Edit the anyco.php file. Change the INSERT statement in the insert_new_emp() function so that it returns the new employee identifier in the bind variable :neweid. This value is inserted with the image into the new EMPLOYEE_PHOTOS table.

```
$statement =
  'INSERT INTO employees
              (employee_id, first_name, last_name, email, hire_date,
              job_id, salary, commission_pct, department_id)
   VALUES (employees_seq.nextval, :fnm, :lnm, :eml, :hdt,
          :jid, :sal, :cpt, :did)
   RETURNING employee_id into :neweid';
```

Also in the `insert_new_emp()` function, add a call to the `array_push()` function to set a new bind variable `NEWEID` at the end of the list of `array_push()` calls:

```
array_push($bindargs, array('CPT', $newemp['commpct'], -1));
array_push($bindargs, array('DID', $newemp['deptid'], -1));
array_push($bindargs, array('NEWEID', null, 10));
```

Because the value of `NEWID` is being retrieved with the RETURNING clause in the INSERT statement, its initial value is set to `NULL`. The length is set to 10 to allow enough digits in the return value.

14. Edit the `anyco.php` file. In the `insert_new_emp()` function, add a call between the `db_execute_statement()` and `construct_employees()` calls to insert the thumbnail image:

```
$r = db_execute_statement($conn, $statement, $err, $bindargs);
if ($r) {
  $r = db_insert_thumbnail($conn, $bindargs['NEWEID'],
                           $_FILES['empphoto']['tmp_name'], $e);
  construct_employees();
}
```

15. In a browser, enter the following application URL:

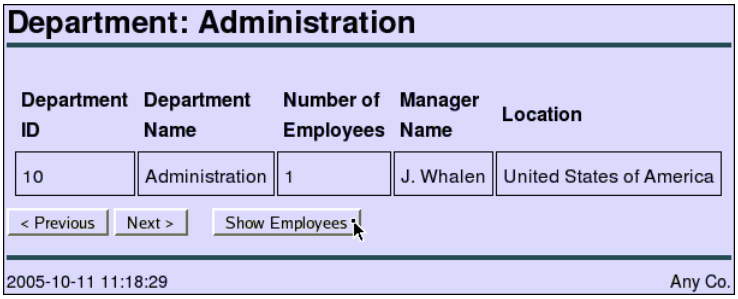
    On Windows:

    `http://localhost/chap7/anyco.php`

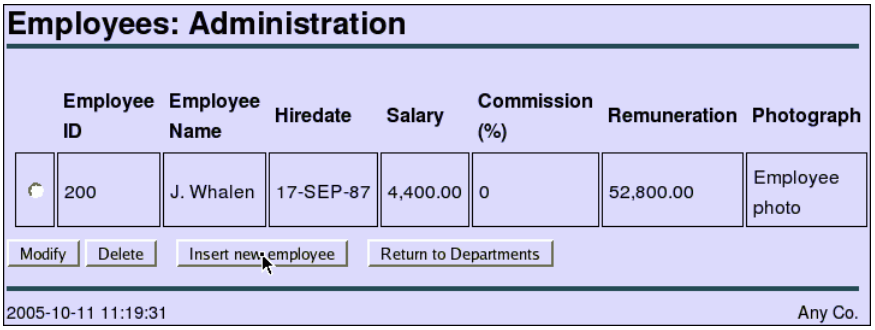
    On Linux:

    `http://localhost/~<username>/chap7/anyco.php`

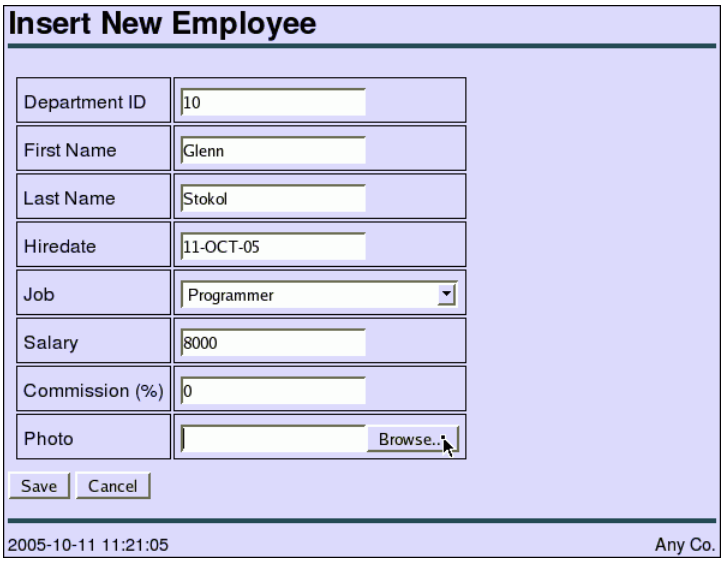
16. In the Departments page, click **Show Employees** to navigate to the Employees page:



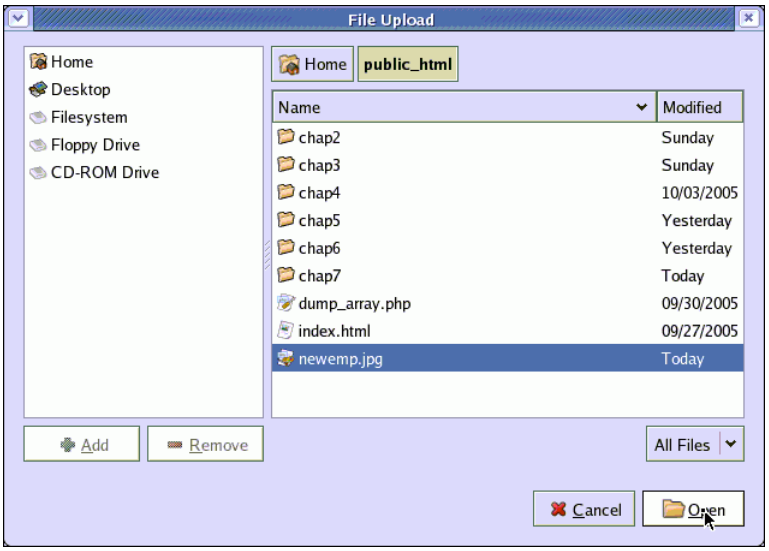
17. In the Employees page, to insert a new employee record click **Insert new employee**:



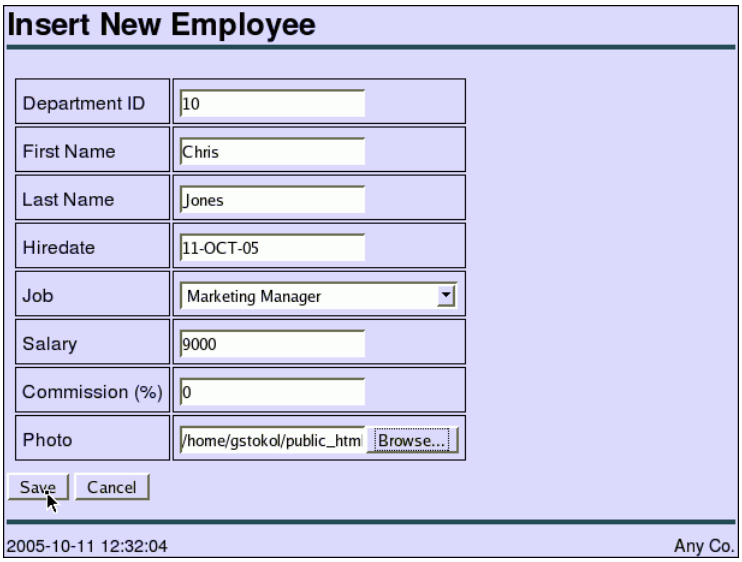
18. The Insert New Employee form allows you to choose a thumbnail image on your system to be uploaded to the database. Enter your own values in the fields or use the values as shown. Click **Browse**:



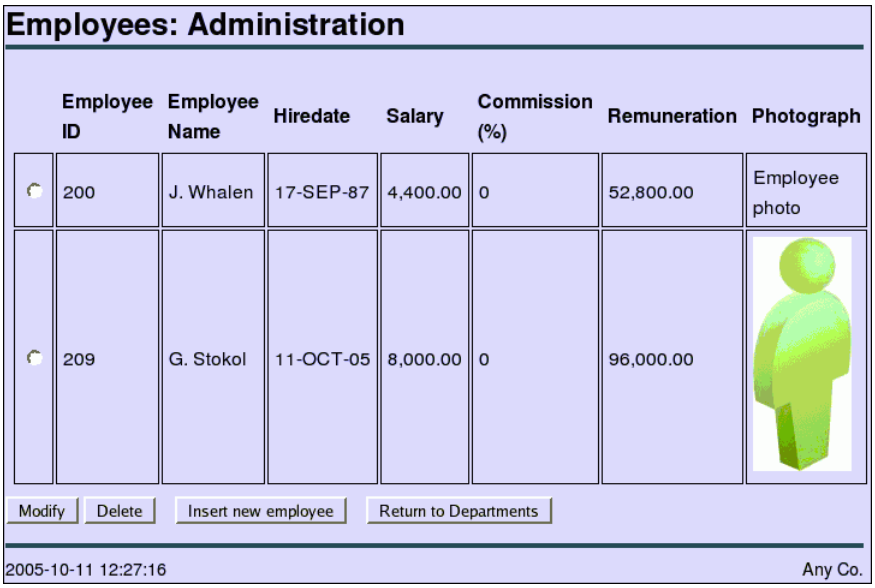
19. In the File Upload window, browse for and select a JPEG image file, and click **Open**:



20. In the Insert New Employee page, click **Save**:



The Employees page is displayed with the new employee record, including the image, which is displayed at its original size:



Resizing Images

In this section, you will further modify your application code to create a thumbnail image from a supplied image, and store the thumbnail image in the record of an employee.

You can use the PHP GD graphicsextension to resize employee images.

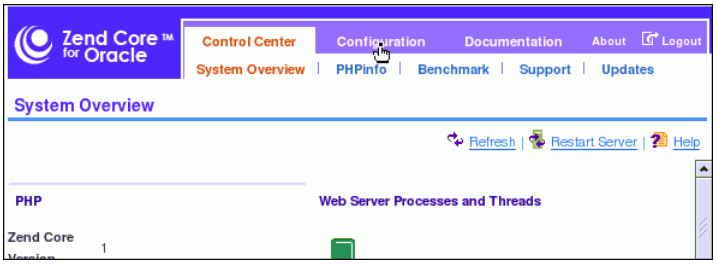
1. To turn on the graphic extension, enter the following URL in your browser to access the Zend Core for Oracle Console:

   ```
   http://localhost/ZendCore
   ```

2. At the login screen, in the Password field enter the password you provided when Zend Core for Oracle was installed, and click the login (>>>) icon.

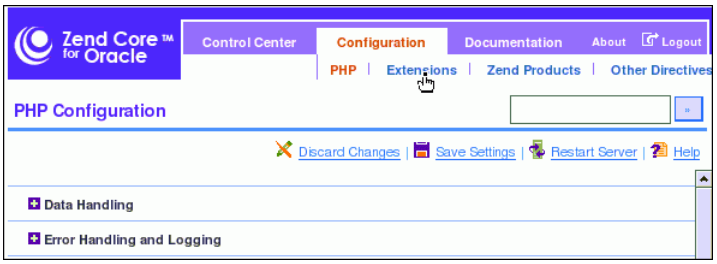
Copyright, 2005, Zend Technologies Ltd.

3. In the Console page, click the **Configuration** tab.



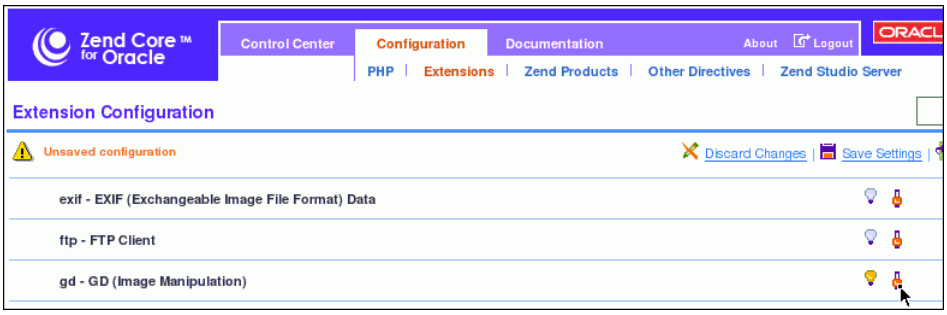
Copyright, 2005, Zend Technologies Ltd.

4. In the Configuration tab page, click the **Extensions** subtab.



Copyright, 2005, Zend Technologies Ltd.

5. In the Extension subtab page, expand the Zend Core Extensions tree control. Locate the **gd - GD (Image Manipulation)** entry and change its switch to `on` or `enabled`.



6. In the Extension subtab page, to save the configuration changes, click **Save Setting**.
7. In the Extension subtab page, to restart the Web server, click **Restart Server**.
8. To log out of the Zend Core for Oracle Console, click **Logout**.
9. Edit the `anyco_db.inc` file. To resize the image to create a thumbnail image, add the following code before the call to `$lob->savefile($imgfile)` in the `db_insert_thumbnail()` function:

```
$r = oci_execute($stmt, OCI_DEFAULT);
if (!$r) {
  $e = db_error($stid, __FILE__, __LINE__);
  return false;
}

// Resize the image to a thumbnail
define('MAX_THUMBNAIL_DIMENSION', 100);
$src_img = imagecreatefromjpeg($imgfile);
list($w, $h) = getimagesize($imgfile);
if ($w > MAX_THUMBNAIL_DIMENSION || $h > MAX_THUMBNAIL_DIMENSION)
{
  $scale =  MAX_THUMBNAIL_DIMENSION / (($h > $w) ? $h : $w);
  $nw = $w * $scale;
  $nh = $h * $scale;

  $dest_img = imagecreatetruecolor($nw, $nh);
  imagecopyresampled($dest_img, $src_img, 0, 0, 0, 0, $nw, $nh, $w, $h);

  imagejpeg($dest_img, $imgfile);  // overwrite file with new thumbnail

  imagedestroy($src_img);
  imagedestroy($dest_img);
}

if (!$lob->savefile($imgfile)) {
...
```

The `imagecreatefromjpeg()` function reads the JPEG file and creates an internal representation used by subsequent GD functions. Next, new dimensions are calculated with the longest side no larger than 100 pixels. A template image with the new size is created using the `imagecreatetruecolor()` function. Data from the original image is sampled into it with the `imagecopyresampled()` function to create the thumbnail image. The thumbnail image is written back to the original file and the internal representations of the images are freed.

The existing code in the `db_insert_thumbnail()` function uploads the image file to the database as it did in the previous implementation.

10. Enter the following URL in your browser to test the changes in your application:

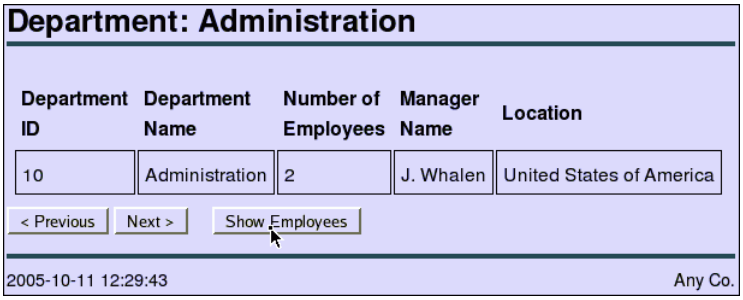
    On Windows:

    ```
    http://localhost/chap7/anyco.php
    ```

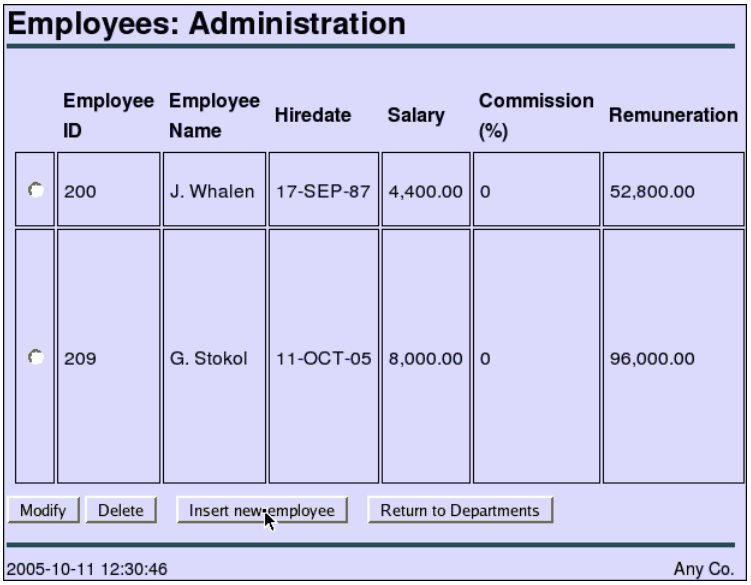
    On Linux:

    ```
    http://localhost/~<username>/chap7/anyco.php
    ```

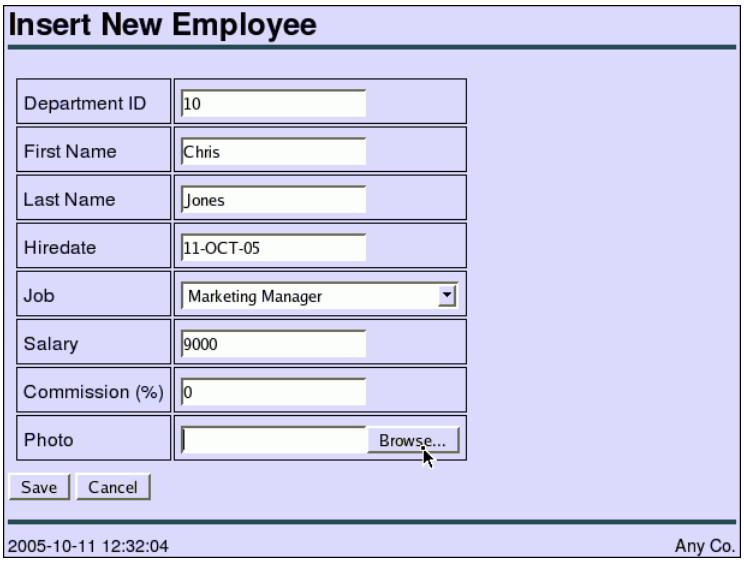
11. In the Departments page, navigate to the Employees page by clicking **Show Employees**:



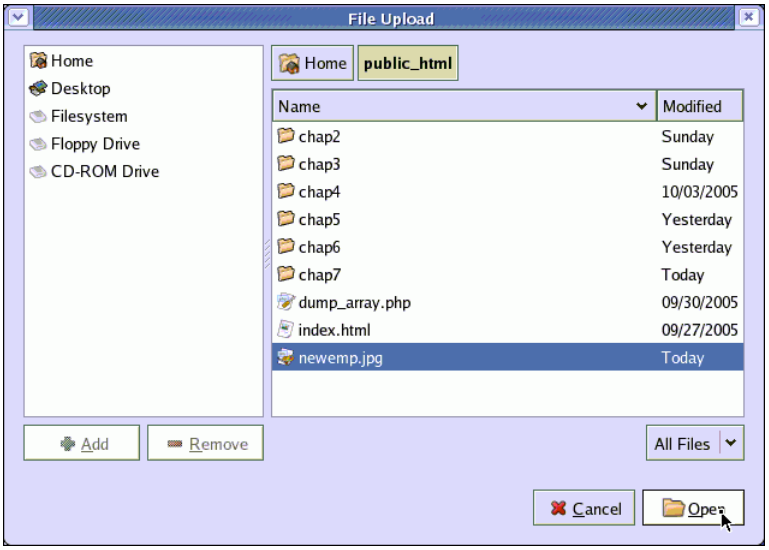
12. In the Employees page, to insert a new employee record, click **Insert new employee**:



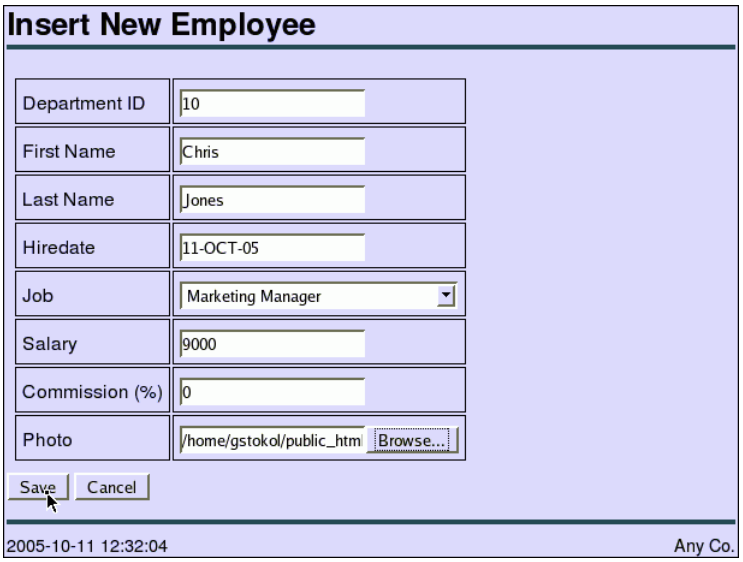
13. Enter the new employee details or use the values shown. To browse for an employee image, click **Browse**:



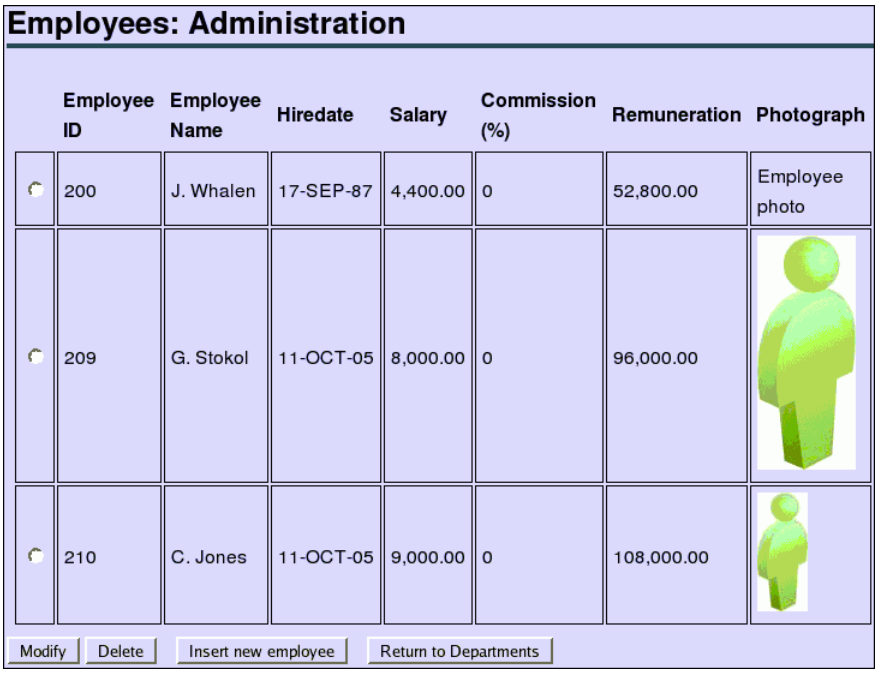
14. Locate and select a JPEG image with a size larger than 100 pixels, and click **Open**:



15. In the Insert New Image page, click **Save**:



The Employees page shows the new uploaded JPEG image with a reduced image size, compared to the image loaded before including the image resize code:



8

Building Global Applications

This chapter discusses global application development in a PHP and Oracle Database Express environment. It addresses the basic tasks associated with developing and deploying global Internet applications, including developing locale awareness, constructing HTML content in the user-preferred language, and presenting data following the cultural conventions of the locale of the user.

Building a global Internet application that supports different locales requires good development practices. A locale refers to a national language and the region in which the language is spoken. The application itself must be aware of the locale preference of the user and be able to present content following the cultural conventions expected by the user. It is important to present data with appropriate locale characteristics, such as the correct date and number formats. Oracle Database Express is fully internationalized to provide a global platform for developing and deploying global applications.

This chapter has the following topics:

Establishing the Environment Between Oracle and PHP

Correctly setting up the connectivity between the PHP engine and the Oracle database is first step in building a global application, it guarantees data integrity across all tiers. Most internet based standards support Unicode as a character encoding, in this chapter we will focus on using Unicode as the character set for data exchange.

Zend Core for Oracle is an Oracle OCI application, and rules that apply to OCI also apply to PHP. Oracle locale behavior (including the client character set used in OCI applications) is defined by the `NLS_LANG` environment variable. This environment variable has the form:

```
<language>_<territory>.<character set>
```

For example, for a German user in Germany running an application in Unicode, `NLS_LANG` should be set to

```
GERMAN_GERMANY.AL32UTF8
```

The language and territory settings control Oracle behaviors such as the Oracle date format, error message language, and the rules used for sort order. The character set AL32UTF8 is the Oracle name for UTF-8.

For information on the `NLS_LANG` environment variable, see the Oracle Database Express Edition installation guides.

When Zend Core for Oracle is installed on Apache, you can set `NLS_LANG` in `/etc/profile`:

```
export NLS_LANG GERMAN_GERMANY.AL32UTF8
```

If Zend Core for Oracle is installed on Oracle HTTP Server, you must set `NLS_LANG` as an environment variable in `$ORACLE_HOME/opmn/conf/opmn.xml`:

```
<ias-component id="HTTP_Server">
  <process-type id="HTTP_Server" module-id="OHS">
    <environment>
      <variable id="PERL5LIB"
       value="D:\oracle\1012J2EE\Apache\Apache\mod_perl\site\5.6.1\lib"/>
      <variable id="PHPRC" value="D:\oracle\1012J2EE\Apache\Apache\conf"/>
      <variable id="NLS_LANG" value="german_germany.al32utf8"/>
    </environment>
    <module-data>
      <category id="start-parameters">
        <data id="start-mode" value="ssl-disabled"/>
      </category>
    </module-data>
    <process-set id="HTTP_Server" numprocs="1"/>
  </process-type>
</ias-component>
```

You must restart the Web listener to implement the change.

Manipulating Strings

PHP was designed to work with the ISO-8859-1 character set. To handle other character sets, specifically multibyte character sets, a set of "MultiByte String Functions" is available. To enable these functions, open the Zend Core for Oracle console and go to the Configuration tab.

Navigate to the Extensions subtab and expand the Zend Core Extensions tree control.

Your application code should use functions such as `mb_strlen()` to calculate the number of characters in strings. This may return different values than `strlen()`, which returns the number of bytes in a string.

Once you have enabled the mbstring extension and restarted the Web server, several configuration options become available. You can change the behavior of the standard PHP string functions by setting `mbstring.func_overload` to one of the "Overload" settings.

For more information, see the PHP mbstring reference manual at

http://www.php.net/mbstring

Determining the Locale of the User

In a global environment, your application should accommodate users with different locale preferences. Once it has determined the preferred locale of the user, the application should construct HTML content in the language of the locale and follow the cultural conventions implied by the locale.

A common method to determine the locale of a user is from the default ISO locale setting of the browser. Usually a browser sends its locale preference setting to the HTTP server with the Accept Language HTTP header. If the Accept Language header is NULL, then there is no locale preference information available, and the application should fall back to a predefined default locale.

The following PHP code retrieves the ISO locale from the Accept-Language HTTP header through the `$_SERVER` Server variable.

```
$s = $_SERVER["HTTP_ACCEPT_LANGUAGE"]
```

Developing Locale Awareness

Once the locale preference of the user has been determined, the application can call locale-sensitive functions, such as date, time, and monetary formatting to format the HTML pages according to the cultural conventions of the locale.

When you write global applications implemented in different programming environments, you should enable the synchronization of user locale settings between the different environments. For example, PHP applications that call PL/SQL procedures should map the ISO locales to the corresponding `NLS_LANGUAGE` and `NLS_TERRITORY` values and change the parameter values to match the locale of the user before calling the PL/SQL procedures. The PL/SQL UTL_I18N package contains mapping functions that can map between ISO and Oracle locales.

Table 8–1 shows how some commonly used locales are defined in ISO and Oracle environments.

Table 8–1 Locale Representations in ISO, SQL, and PL/SQL Programming Environments

Locale	Locale ID	NLS_LANGUAGE	NLS_TERRITORY
Chinese (P.R.C.)	zh-CN	SIMPLIFIED CHINESE	CHINA
Chinese (Taiwan)	zh-TW	TRADITIONAL CHINESE	TAIWAN
English (U.S.A)	en-US	AMERICAN	AMERICA
English (United Kingdom)	en-GB	ENGLISH	UNITED KINGDOM
French (Canada)	fr-CA	CANADIAN FRENCH	CANADA
French (France)	fr-FR	FRENCH	FRANCE
German	de	GERMAN	GERMANY
Italian	it	ITALIAN	ITALY
Japanese	ja	JAPANESE	JAPAN
Korean	ko	KOREAN	KOREA
Portuguese (Brazil)	pt-BR	BRAZILIAN PORTUGUESE	BRAZIL
Portuguese	pt	PORTUGUESE	PORTUGAL

Table 8–1 (Cont.) Locale Representations in ISO, SQL, and PL/SQL Programming

Locale	Locale ID	NLS_LANGUAGE	NLS_TERRITORY
Spanish	es	SPANISH	SPAIN

Encoding HTML Pages

The encoding of an HTML page is important information for a browser and an Internet application. You can think of the page encoding as the character set used for the locale that an Internet application is serving. The browser must know about the page encoding so that it can use the correct fonts and character set mapping tables to display the HTML pages. Internet applications must know about the HTML page encoding so they can process input data from an HTML form.

Instead of using different native encodings for the different locales, Oracle recommends that you use UTF-8 (Unicode encoding) for all page encodings. This encoding not only simplifies the coding for global applications, but it also enables multilingual content on a single page.

Specifying the Page Encoding for HTML Pages

You can specify the encoding of an HTML page either in the HTTP header, or in HTML page header.

Specifying the Encoding in the HTTP Header

To specify HTML page encoding in the HTTP header, include the Content-Type HTTP header in the HTTP specification. It specifies the content type and character set. The Content-Type HTTP header has the following form:

```
Content-Type: text/html; charset=utf-8
```

The charset parameter specifies the encoding for the HTML page. The possible values for the charset parameter are the IANA names for the character encodings that the browser supports.

Specifying the Encoding in the HTML Page Header

Use this method primarily for static HTML pages. To specify HTML page encoding in the HTML page header, specify the character encoding in the HTML header as follows:

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
```

The charset parameter specifies the encoding for the HTML page. As with the Content-Type HTTP Header, the possible values for the charset parameter are the IANA names for the character encodings that the browser supports.

Specifying the Page Encoding in PHP

You can specify the encoding of an HTML page in the Content-Type HTTP header in PHP by setting the `default_charset` configuration variable as follows:

```
default_charset = UTF-8
```

This can be found in the Zend Core for Oracle Console in the Configuration tab. Choose the PHP subtab and expand the Data Handling tree control. After entering a value, save the configuration settings and restart the Web server.

This setting does not imply any conversion of outgoing pages. Your application must ensure that the server-generated pages are encoded in UTF-8.

Organizing the Content of HTML Pages for Translation

Making the user interface available in the local language of the user is a fundamental task in globalizing an application. Translatable sources for the content of an HTML page belong to the following categories:

- Text strings included in the application code
- Static HTML files, images files, and template files such as CSS
- Dynamic data stored in the database

Strings in PHP

You should externalize translatable strings within your PHP application logic, so that the text is readily available for translation. These text messages can be stored in flat files or database tables depending on the type and the volume of the data being translated.

Static Files

Static files such as HTML and GIF files are readily translatable. When these files are translated, they should be translated into the corresponding language with UTF-8 as the file encoding. To differentiate the languages of the translated files, stage the static files of different languages in different directories or with different file names.

Data from the Database

Dynamic information such as product names and product descriptions is typically stored in the database. To differentiate various translations, the database schema holding this information should include a column to indicate the language. To select the desired language, you must include a WHERE clause in your query.

Presenting Data Using Conventions Expected by the User

Data in the application must be presented in a way that conforms to the expectation of the user. Otherwise, the meaning of the data can be misinterpreted. For example, the date '12/11/05' implies '11th December 2005' in the United States, whereas in the United Kingdom it means '12th November 2005'. Similar confusion exists for number and monetary formats of the users. For example, the symbol '.' is a decimal separator in the United States; in Germany this symbol is a thousand separator.

Different languages have their own sorting rules. Some languages are collated according to the letter sequence in the alphabet, some according to the number of stroke counts in the letter, and some languages are ordered by the pronunciation of the words. Presenting data not sorted in the linguistic sequence that your users are accustomed to can make searching for information difficult and time consuming.

Depending on the application logic and the volume of data retrieved from the database, it may be more appropriate to format the data at the database level rather than at the application level. Oracle Database XE offers many features that help to refine the presentation of data when the locale preference of the user is known. The following sections provide examples of locale-sensitive operations in SQL.

Oracle Date Formats

The three different date presentation formats in Oracle Database XE are standard, short, and long dates. The following examples illustrate the differences between the short date and long date formats for both the United States and Germany.

```
SQL> alter session set nls_territory=america nls_language=american;

Session altered.


SQL> select employee_id EmpID,
  2  substr(first_name,1,1)||'.'||last_name "EmpName",
  3  to_char(hire_date,'DS') "Hiredate",
  4  to_char(hire_date,'DL') "Long HireDate"
  5  from employees
  6* where employee_id <105;

     EMPID EmpName                   Hiredate   Long HireDate
---------- ------------------------- ---------- ----------------------------
       100 S.King                    06/17/1987 Wednesday, June 17, 1987
       101 N.Kochhar                 09/21/1989 Thursday, September 21, 1989
       102 L.De Haan                 01/13/1993 Wednesday, January 13, 1993
       103 A.Hunold                  01/03/1990 Wednesday, January 3, 1990
       104 B.Ernst                   05/21/1991 Tuesday, May 21, 1991

SQL> alter session set nls_territory=germany nls_language=german;

Session altered.

SQL> select employee_id EmpID,
  2  substr(first_name,1,1)||'.'||last_name "EmpName",
  3  to_char(hire_date,'DS') "Hiredate",
  4  to_char(hire_date,'DL') "Long HireDate"
  5  from employees
  6* where employee_id <105;

     EMPID EmpName                   Hiredate Long HireDate
---------- ------------------------- -------- ------------------------------
       100 S.King                    17.06.87 Mittwoch, 17. Juni 1987
       101 N.Kochhar                 21.09.89 Donnerstag, 21. September 1989
       102 L.De Haan                 13.01.93 Mittwoch, 13. Januar 1993
       103 A.Hunold                  03.01.90 Mittwoch, 3. Januar 1990
       104 B.Ernst                   21.05.91 Dienstag, 21. Mai 1991
```

Oracle Number Formats

The following examples illustrate the differences in the decimal character and group separator between the United States and Germany.

```
SQL> alter session set nls_territory=america;

Session altered.

SQL> select employee_id EmpID,
  2  substr(first_name,1,1)||'.'||last_name "EmpName",
  3  to_char(salary, '99G999D99') "Salary"
  4  from employees
  5* where employee_id <105
```

```
     EMPID EmpName                    Salary
---------- ------------------------- ----------
       100 S.King                     24,000.00
       101 N.Kochhar                  17,000.00
       102 L.De Haan                  17,000.00
       103 A.Hunold                    9,000.00
       104 B.Ernst                     6,000.00

SQL> alter session set nls_territory=germany;

Session altered.

SQL> select employee_id EmpID,
  2  substr(first_name,1,1)||'.'||last_name "EmpName",
  3  to_char(salary, '99G999D99') "Salary"
  4  from employees
  5* where employee_id <105

     EMPID EmpName                    Salary
---------- ------------------------- ----------
       100 S.King                     24.000,00
       101 N.Kochhar                  17.000,00
       102 L.De Haan                  17.000,00
       103 A.Hunold                    9.000,00
       104 B.Ernst                     6.000,00
```

Oracle Linguistic Sorts

Spain traditionally treats *ch*, *ll* as well as *ñ* as unique letters, ordered after *c*, *l* and *n* respectively. The following examples illustrate the effect of using a Spanish sort against the employee names Chen and Chung.

```
SQL> alter session set nls_sort=binary;

Session altered.

SQL> select employee_id EmpID,
  2         last_name "Last Name"
  3  from employees
  4  where last_name like 'C%'
  5* order by last_name

     EMPID Last Name
---------- -------------------------
       187 Cabrio
       148 Cambrault
       154 Cambrault
       110 Chen
       188 Chung
       119 Colmenares

6 rows selected.

SQL> alter session set nls_sort=spanish_m;

Session altered.

SQL> select employee_id EmpID,
  2         last_name "Last Name"
```

```
  3  from employees
  4  where last_name like 'C%'
  5* order by last_name

    EMPID Last Name
---------- -------------------------
       187 Cabrio
       148 Cambrault
       154 Cambrault
       119 Colmenares
       110 Chen
       188 Chung

6 rows selected.
```

Oracle Error Messages

The NLS_LANGUAGE parameter also controls the language of the database error messages being returned from the database. Setting this parameter prior to submitting your SQL statement ensures that the language-specific database error messages will be returned to the application.

Consider the following server message:

```
ORA-00942: table or view does not exist
```

When the NLS_LANGUAGE parameter is set to French, the server message appears as follows:

```
ORA-00942: table ou vue inexistante
```

For more discussion of globalization support features in Oracle Database Express Edition, see "Working in a Global Environment" in *Oracle Database Express Edition 2 Day Developer Guide* .

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Z