Oracle® Database Express Edition

2 Day + PHP Developer's Guide

11*g* Release 2 (11.2)

E18555-04

May 2014

ORACLE®

Oracle Database Express Edition 2 Day + PHP Developer's Guide, 11*g* Release 2 (11.2)

E18555-04

Primary Author: Simon Watt

Contributors: Christopher Jones, Simon Law, Glenn Stokol, Ligaya Turmelle, Johannes Schlüter

Contents

3 Building a Database Access Class

4 Building the AnyCo Application

5 Paging Through Employee Data

6 Showing Equipment Records by Using a REF CURSOR

7 Error handling

8 Query Performance and Prefetching

9 Inserting Data

10 Inserting Multiple Data Values

11 Using JSON and generating a JPEG image

12 Uploading and Displaying BLOBs

13 Monitoring Database Usage of the Application

14 Building Global Applications

Index

Preface

Oracle Database Express Edition 2 Day + PHP Developer's Guide introduces developers to the use of PHP to access Oracle Database Express Edition (Oracle Database XE).

This preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

Oracle Database Express Edition 2 Day + PHP Developer's Guide is an introduction to application development using PHP and Oracle Database XE.

This document assumes that you have a cursory understanding of SQL, PL/SQL, and PHP.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see these Oracle resources:

- *Oracle Database Express Edition 2 Day Developer's Guide*
- *Oracle Database SQL Language Reference*
- *Oracle Database PL/SQL Language 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.

This chapter contains the following topics:

- Purpose
- Overview of the Sample Application
- Resources

Purpose

This guide shows you how to create a web application using the PHP scripting language and Oracle Database XE.

Overview of the Sample Application

This document guides you through the development of a sample application that manages the tracking of company equipment for a fictitious company called `AnyCo Corp`. For this introduction to the PHP language and the PHP OCI8 extension which accesses the Oracle database, no PHP framework or abstraction layer is used. However, frameworks are popular and they should be evaluated when building applications. They provide functionality to do tasks the `AnyCo` application has to manually implement, and they can provide a good application design paradigm.

The AnyCo application uses employee data from the `EMPLOYEES` table in the sample HR schema provided with Oracle Database XE. See *Oracle Database Sample Schemas* for information about this schema. A new table will be created for this application to hold details about the company equipment allocated to each employee.

Figure 1–1 shows the overview of the sample application.

Figure 1–1   Overview of the Sample Application

The application will perform the following functions:

■ Establish a connection to the database using the PHP OCI8 extension. An Oracle connection pool is used to demonstrate how applications can be made scalable.

■ Query the database employee and equipment data.

■ Display and navigate through the data.

■ Show how to insert and fetch records in various ways, including fetching using a PL/SQL REF CURSOR.

■ Show how to tune PHP fetching data from SQL queries.

■ Show how to create and use a web service.

■ Upload and display an image.

■ Show how to monitor the application's use of database resources.

Resources

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

■ Oracle Database XE home page on Oracle Technology Network

http://www.oracle.com/technetwork/database/express-edition/downloads/index.html

■ Oracle Database XE Documentation Library

http://www.oracle.com/pls/xe112/homepage

■ The free book "Underground PHP and Oracle Manual" goes into detail about using PHP with Oracle Database:

http://www.oracle.com/technetwork/topics/php/underground-php-oracle-manual-098250.html

■ PHP Developer Center with resources and a PHP forum at

http://www.oracle.com/technetwork/topics/php/whatsnew/index.html

■ NetBeans IDE learning trail for PHP development at

http://netbeans.org/kb/trails/php.html

■ Oracle Database Documentation Library at

`http://www.oracle.com/technetwork/indexes/documentation/index`
`.html`

- PHP Scalability and High Availability white paper:

  `http://www.oracle.com/technetwork/topics/php/whatsnew/php-sca`
  `lability-ha-twp-128842.pdf`

- PHP Web Auditing, Authorization and Monitoring with Oracle Database

  `http://www.oracle.com/technetwork/articles/dsl/php-web-auditi`
  `ng-171451.html`

- PHP Online manual at

  `http://www.php.net/manual/en/`

2

Getting Started

This chapter explains how to install and test Oracle Database Express Edition and PHP environment.

This chapter contains the following topics:

- What You Need
- Installing Oracle Database Express Edition
- Installing Apache HTTP Server
- Testing the Apache Installation
- Installing PHP
- Testing the PHP Installation
- Installing the NetBeans IDE

What You Need

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

- Oracle Database Express Edition 11gR2
- Apache Web Server. On Linux this is commonly available in the package repository.
- PHP 5.3 or later. Several recent Linux distributions now include this version as a package.
- A text editor for editing PHP files. A code editor such as NetBeans PHP edition with a debugger is ideal, but not required.

Installing Oracle Database Express Edition

If you have not already installed Oracle Database Express Edition on your computer, you must do so. The sample data used in this tutorial is installed by default. It is the HR component of the Sample Schemas.

For information about installing Express Edition, see the installation guide for your operating system:

- *Oracle Database Express Edition Installation Guide for Microsoft Windows*
- *Oracle Database Express Edition Installation Guide for Linux x86-64*

You are encouraged to become familiar with basic concepts and operations described in *Oracle Database Express Edition Getting Started Guide*.

This section contains the following topics:

- Unlocking the HR User
- Database Resident Connection Pooling
- Starting the DRCP Pool

> **See Also:**
>
> - *Oracle Database Sample Schemas* guide for information about the HR sample schema.
> - Oracle SQL Developer web page
>
>   http://www.oracle.com/technetwork/developer-tools
>   /sql-developer/overview/index.html

Unlocking the HR User

The PHP application connects to the database as the HR user. You may need to unlock the HR account before it can be used. Use SQL*Plus or SQL Developer to unlock the HR user.

This section contains the following topics:

- Unlocking the HR User Using a Command Line
- Unlocking the HR User Using SQL Developer:

Unlocking the HR User Using a Command Line

Unlock the HR user using a command line as follows:

```
SQL> source /u01/app/oracle/product/11.2.0/xe/bin/oracle_env.sh
SQL> sqlplus system/system_password
SQL> alter user hr identified by <password-for-hr> account unlock;
```

where, *system_password* is the password you entered during database configuration, and *password-for-hr* is the password for the HR account.

Unlocking the HR User Using SQL Developer:

To unlock the sample user account using SQL Developer:

1. If you have not already created a database connection for the SYSTEM user, do so.

2. Open the database connection for the SYSTEM user.

3. In the Connection navigator under the connection for the SYSTEM user, expand the **Other Users** node.

4. Under Other Users, right-click the node for the HR user and select **Edit User**.

5. In the Create/Edit User dialog box, *uncheck* (deselect) the option **Account is Locked**.

    a. For New Password and Confirm Password, enter the password that you want for the HR user.

    b. *Uncheck* (deselect) **Password expired (User must change next login)**.

    c. *Uncheck* (deselect) **Account is Locked**.

6. Click **Apply** to alter the HR user so that the account is unlocked and not expired.

7. Click **Close** to close the dialog box.

For further information about unlocking an Oracle database account, see Chapter 7, "Managing Users and Security," in *Oracle Database Express Edition 2 Day DBA*.

> **See Also:**
>
> - Oracle Database documentation
>
>   `http://www.oracle.com/technetwork/indexes/documen`
>   `tation/index.html`

Database Resident Connection Pooling

The AnyCo sample application will use Database Resident Connection Pooling (DRCP) to show how a PHP application can scale to support many users.

PHP cannot be assumed to be thread safe and is typically run in a multi-process mode, for example with the Apache web server's pre-fork model or with FastCGI. Sharing Oracle connections between active and idle PHP processes isn't possible in the mid-tier because there is no interprocess communication. DRCP works because the sharing is handled by the database host machine. This also allows connection resources created by multiple mid-tier hosts to be shared.

For best performance it is common for PHP OCI8 applications to use "persistent" database connections. When PHP has completed an application script and sent its output to the web user's browser, the script's underlying DB connection is not closed. The connection remains cached in the still-running, now idle PHP/Apache process. It can be reused by this PHP process in any subsequent PHP script connecting with the same database credentials. This has great performance benefits. However without DRCP, when there are large numbers of PHP processes the open database connections may use a large amount of database host memory. This is despite many connections being idle due to the user's "think-time" between making web page requests, or while the PHP script runs non-database operations.

DRCP allows PHP applications to efficiently use database host memory to support large numbers of web users. DRCP allows database resources to be used by only those web users currently doing database operations. Benchmarks have shown DRCP can support tens of thousands of web users on small, commodity Linux database hosts.

Figure 2–1 Without DRCP, Idle Persistent Connections from PHP still Consume Database Resources.
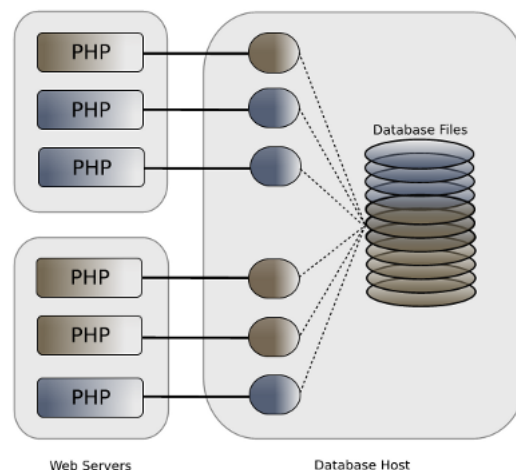
DRCP overcomes the database host memory pressure by maintaining a small pool of database server processes on the host. These can be shared by all the PHP database connections across all PHP processes and mid-tier servers when they are needed.

Figure 2–2   DRCP Architecture
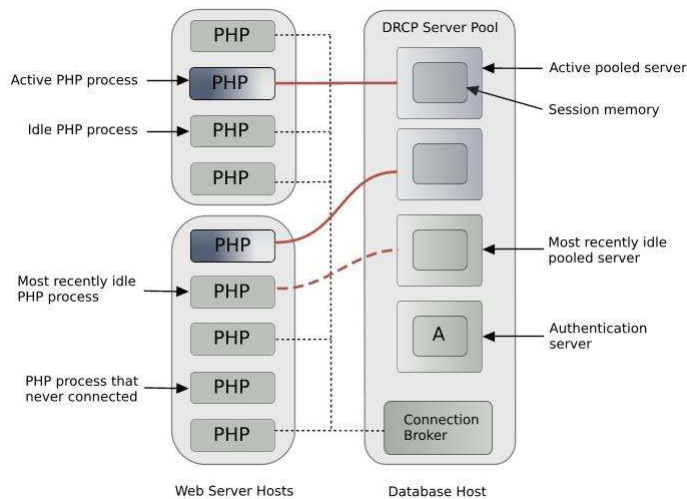
If a PHP script connects to the database but there is no pooled server process available, then it will wait until one is free. This prevents the database from being overloaded and allows applications to continue running.

Once the DRCP pool is started, applications can choose at runtime whether to use it or not. This is indicated in the PHP OCI8 connection string. Typically only short lived, similar kinds of tasks should use DRCP. Batch processes should not use the pool.

The DRCP pool can be used in two variants. The basic method is that only the processes are reused. The second method increases performance by also reusing the "session" memory associated with each process. In PHP, only "persistent" connections use the latter method. For web applications like PHP where each script is part of a single application, this session memory sharing is generally perfectly acceptable. However care must be taken that any retained session settings such as the date format do not occur unexpectedly and that they don't constitute an information security leak. DRCP allows the pool to be virtually sub-partitioned to reduce any issues like this.

PHP OCI8 applications use the `oci_pconnect()` call to create a persistent database connection. Applications can also connect to Oracle using `oci_connect()` or `oci_new_connect()`, which create non-persistent connections. The `oci_new_connect()` function always returns a new, transactionally independent connection resource each time it is called. The `oci_connect()` and `oci_pconnect()` functions will return their respective same PHP resource if a running script calls them multiple times with the same connection credentials. For each connection method, a rollback occurs at the end of each script, if necessary.

There are also differences in behavior between the three functions depending on whether DRCP is being used.

Without DRCP, a persistent connection remains open even when the PHP script has completed. A subsequent script connecting with the same credentials can immediately reuse that connection. This is fast but the database host must have enough memory to maintain the connections from each PHP process even when the processes are idle. The `oci_connect()` and `oci_new_connect()` functions don't retain the underlying connection to the database after the PHP script completes. This makes

connection slower to establish but memory use on the database host is capped by the number of active web users.

When PHP connections uses DRCP, all three OCI8 connection functions benefit from using established DRCP server processes from the DRCP pool. When each script finishes (if not earlier), its database pooled server is returned to the DRCP pool for reuse. A lightweight connection to the DRCP broker is retained, which aids re-connection performance. An `oci_pconnect()` function will reuse the process session memory, provide more efficiency and higher scalability. Each `oci_connect()` and `oci_new_connect()` call will recreate the Oracle session memory in the reused DRCP pooled process.

More information on DRCP and PHP is in the white paper:

http://www.oracle.com/technetwork/topics/php/whatsnew/php-scalability-ha-twp-128842.pdf

Starting the DRCP Pool

The DRCP pool can be controlled in SQL*Plus with the pre-supplied PL/SQL `DBMS_CONNECTION_POOL` package.

To start the pool on Oracle Linux, open a terminal window and connect as the root user:

```
# su -
```

Now, `su` to the Oracle account:

```
# su - oracle
$ source /u01/app/oracle/product/11.2.0/xe/bin/oracle_env.sh
```

Run SQL*Plus with the SYSDBA system privilege and invoke the `DBMS_CONNECTION_POOL.START_POOL()` procedure:

```
$ sqlplus / as sysdba
SQL> execute dbms_connection_pool.start_pool()
```

The pool will now run with its default parameters.

To stop the pool, run:

```
SQL> execute dbms_connection_pool.stop_pool()
```

If DRCP is running when the database is restarted, then the pool will automatically restart.

Pool parameters can be changed with `DBMS_CONNECTION_POOL.ALTER_PARAM()`, for example:

```
SQL> execute dbms_connection_pool.alter_param(null, 'MAXSIZE', '10');
```

The pool should be restarted after changing parameters.

The current pool settings can be viewed by querying Oracle's data dictionary:

```
select * from DBA_CPOOL_INFO;
```

The overall DRCP Pool statistics can be seen in

```
select * from V$CPOOL_STATS;
```

By observing the statistics over time you can decide how to tune the pool parameters.

Each DRCP database connection can specify an arbitrary "connection class". In PHP the connection class can be configured in PHP's `php.ini` initialization file or can be set at run time. See "Post PHP Installation Tasks on Windows and Linux". The connection class helps partition the DRCP pool for different use cases.

The statistics for each connection class can be seen using:

```
# select * from V$CPOOL_CC_STATS;
```

Not setting a connection class results in reduced sharing of the DRCP pool resources. For general application, if `V$CPOOL_CC_STATS` shows a large number of system generated connection class names, then check that your PHP configuration files on each mid tier server is correctly setting the connection class.

The DRCP pool is sharable across all enabled applications, including those written in PHP, Perl, and Python. Some tools like SQL*Plus are not DRCP enabled. If you use a DRCP connection with SQL*Plus you will see entries in `V$CPOOL_CC_STATS` with the class name `SHARED`. SQL*Plus will reuse the DRCP pool processes but will have to recreate each process's session memory.

When you have built the AnyCo application and run it, you can examine the monitoring views to see DRCP in action.

Installing Apache HTTP Server

The Apache HTTP server handles incoming user page requests and invokes PHP to generate the application's HTML markup.

For information about downloading, installing, and using the Apache HTTP server, see the Apache HTTP Server Project page at: `http://httpd.apache.org/`

This section contains the following topics:

- Installing Apache on Windows XP
- Installing Apache on Linux

Installing Apache on Windows XP

> **Note:** Windows XP is no longer officially supported by Microsoft. For the most recent Windows installation information, see the Apache HTTP Server Project page at: `http://httpd.apache.org/`

PHP 5.3.6 is installed using the FastCGI model in Windows. Perform the following steps to obtain Apache HTTP Server for Windows:

1. Enter the following URL in your Web browser:

   `http://httpd.apache.org/download.cgi`

2. Click the **httpd-2.2.17-win32-x86-no_ssl.msi**.

3. Save the downloaded file in a temporary directory, such as `c:\tmp` and double click to install it.

   The software will install to a directory like: `C:\Program Files\Apache Software Foundation\Apache2.2`.

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.

4. Download the Apache `mod_fcgid` FastCGI component from the following URL:

   http://httpd.apache.org/download.cgi#mod_fcgid

5. Unzip it to the installed Apache 2.2 directory.

6. Edit `C:\Program Files\Apache Software Foundation\Apache2.2\conf\httpd.conf` and add:

   ```
   LoadModule fcgid_module modules/mod_fcgid.so
   ```

   In `httpd.conf` locate the <Directory> section for htdocs and add ExecCGI to the Options:

   ```
   <Directory "C:/Program Files/Apache Software Foundation/Apache2.2/htdocs">
   ...
   Options Indexes FollowSymLinks ExecCGI
   ...
   </Directory>
   ```

You can use the **Start** menu option to start Apache. This opens a console window showing any error messages. Error messages may also be written to `C:\Program Files\Apache Software Foundation\Apache2.2\logs\error.log`.

You can also use the ApacheMonitor utility to start Apache. If you chose to install Apache as a service for all users, it will appear as an icon in your System Tray.

If you have errors, double check your `httpd.conf` file

Installing Apache on Linux

> **Note:** Some information in this section refers to an old version of Apache. For the most recent installation information, see the Apache HTTP Server Project page at: http://httpd.apache.org/

This section describes how to install Apache HTTP Server on Linux.

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.

Apache is typically already installed on Linux or directly available in package repositories.

This section contains the following topics:

- Using the Default HTTPD Package on Oracle Linux
- Manually Installing Apache on Linux
- Setting the Oracle Environment for Apache on Linux
- Setting up a User Directory for the Example Project on Linux

Using the Default HTTPD Package on Oracle Linux

1. On Oracle Linux install the `httpd` package with:

   ```
   # yum install httpd
   ```

or

```
# up2date httpd
```

2. If you will be compiling PHP manually (see later), also install the `httpd-devel` package:

```
# yum install httpd-devel
```

or

```
# up2date httpd-devel
```

3. Stop Apache as root by using:

```
# service httpd stop
```

4. To start Apache run:

```
# service httpd start
```

Manually Installing Apache on Linux

This section describes how to manually install Apache HTTP Server on Linux. 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.

Perform the following steps to install the Apache HTTP Server:

1. Download the `httpd` server from apache.org, for example, `httpd-2.2.17.tar.bz2`.

2. Go to the directory where you downloaded the `httpd-2.2.17.tar.bz2` file.

3. Log in as the root user and run these commands:

```
# tar -jxvf httpd-2.2.17.tar.bz2
# cd httpd-2.2.17
# export ORACLE_HOME=/usr/lib/oracle/app/oracle/product/11.2.0/server
# ./configure \
        --prefix=/usr/local/apache \
        --enable-module=so
# make
# make install
```

The option `--enable-module=so` allows PHP to be compiled as a Dynamic Shared Object (DSO). The `--prefix` option sets the Apache installation directory used by the command `make install`

If you don't want to install and run Apache as a privileged user, set `--prefix` to a directory such as `$HOME/apache`. Then, after installation completes, you will also need to edit `httpd.conf` and modify the `Listen` parameter to change the port that Apache listens on, because non-privileged users cannot use the default port `80`.

Apache can be started with the `apachectl` script:

```
# /usr/local/apache/bin/apachectl start
```

Stop Apache with:

```
# /usr/local/apache/bin/apachectl stop
```

Setting the Oracle Environment for Apache on Linux

The Oracle environment must be set correctly before starting Apache so that PHP OCI8 works correctly. In general you should set the same variables that are set by the $ORACLE_HOME/bin/oracle_env.sh script. The necessary environment variables can be set in Apache's environment configuration file.

On Oracle Linux with the default httpd package, this is /etc/sysconfig/httpd. If you installed your own Apache using the instructions in the previous section it is /usr/local/bin/envvars. Edit the file and add these lines:

```
export ORACLE_HOME=/u01/app/oracle/product/11.2.0/xe
export LD_LIBRARY_PATH=$ORACLE_HOME/lib:$LD_LIBRARY_PATH
```

Stop and restart Apache so the environment variables are in effect.

Setting up a User Directory for the Example Project on Linux

The PHP files that will be created later need to be stored in a directory accessible by Apache. One possible location is Apache's Document root directory /var/www/html (or /usr/local/apache/htdocs, if you installed Apache manually). However, you may find it easier to give Apache access to a sub-directory of your home directory.

1. Login as your normal user and make a working directory:

    ```
    mkdir $HOME/public_html
    ```

    You will also need to let the Apache processes access your files, for example with:

    ```
    chmod 755 $HOME $HOME/public_html
    ```

2.

    a. For Oracle Linux, edit /etc/httpd/conf/httpd.conf and locate the mod_userdir.c section. Change it to:

    ```
    <IfModule mod_userdir.c>
      #
      # UserDir is disabled by default since it can confirm the presence
      # of a user name 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>
    ```

    b. If you installed Apache manually, edit /usr/local/apache/conf/httpd.conf and locate the line:

    ```
    Include conf/extra/httpd-userdir.conf
    ```

    Make sure it is uncommented by removing a leading pound sign (#), if one exists.

3. Restart Apache

    This enables the Web browser to serve files from the $HOME/public_html directory of users. For example, if you login as 'chris' then PHP files created in

$HOME/public_html would be accessible by the URL
http://localhost/~chris/

If you decide to create the PHP project files in public_html you will need to change any URLs mentioned later in this manual. For example use http://localhost/~user/ wherever the manual says to use http://localhost/.

Testing the Apache Installation

To test the Apache HTTP Server installation:

1. Start your Web browser on the computer on which you installed Apache.

2. Enter the following URL:

   http://localhost/

   Your Web browser will display a page similar to the following:

   ## It works!

   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. The port number is set by the Listen parameter in Apache's configuration file /etc/httpd/conf/httpd.conf (or /usr/local/apache/conf/httpd.conf, or C:\Program Files\Apache Software Foundation\Apache2.2\conf\httpd.conf).

   If your site uses a non-default port number you will need to change any URLs mentioned later in this manual. For example if Apache listens on port 8888, then use http://localhost:8888/ wherever the manual says to use http://localhost/.

Installing PHP

The application in this manual uses PHP 5.3 which has the OCI8 1.4 extension for Oracle Database. New features in PHP 5.3 and OCI8 1.4 are used. PHP's GD extension is used in Chapter 11, "Using JSON and generating a JPEG image."

This section contains the following topics:

- Installing PHP on Windows

- Installing PHP on Linux

- Post PHP Installation Tasks on Windows and Linux

Installing PHP on Windows

> **Note:** Some information in this section may refer to an old version of PHP. For the most recent information, see the Oracle Technology Network PHP site (http://www.oracle.com/technetwork/database/database -technologies/php/) for Downloads, Documentation (including white papers and FAQs), Community (discussion forum and blog), and other resources.

This section describes how to install PHP on Windows.

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 PHP. To install PHP, perform the following steps:

1. Download the PHP 5.3.6 zip file from the following Web site:

   http://windows.php.net/download/

   Use the non-thread safe bundle because we will install it in FastCGI mode.

2. In Windows Explorer, go to the directory where you downloaded the PHP 5.3.6 zip file.

3. Unzip the PHP package to a directory called `C:\php-5.3.6`

4. Copy `php.ini-development` to `C:\php-5.3.6\php.ini`

5. Edit `php.ini` to make the following changes:

   - Add the line `extension_dir = "C:\php-5.3.6\ext"`.

     This is the directory containing the PHP extensions.

   - Remove the semicolon from the beginning of the line

     `extension=php_oci8_11g.dll`

6. Edit `C:\Program Files\Apache Software Foundation\Apache2.2\conf\httpd.conf` and add the following lines. Make sure you use forward slashes '/' and not back slashes '\':

   ```
   FcgidInitialEnv PHPRC "c:/php-5.3.6"
   AddHandler fcgid-script .php
   FcgidWrapper "c:/php-5.3.6/php-cgi.exe" .php
   ```

   Make sure `mod_fcgid.so` is loaded and the ExecCGI option set as described previously in the section Installing Apache on Windows XP on page 2-6.

7. Restart the Apache Server so that you can test your PHP installation.

   If you have errors, double check your `httpd.conf` and `php.ini` files. Make sure you rebooted the machine after installing Oracle Database XE so that the `PATH` environment variable includes the Oracle libraries.

Installing PHP on Linux

> **Note:** Some information in this section may refer to an old version of PHP. For the most recent information, see the Oracle Technology Network PHP site (http://www.oracle.com/technetwork/database/database-technologies/php/) for Downloads, Documentation (including white papers and FAQs), Community (discussion forum and blog), and other resources.

If your Linux distribution has PHP 5.3 packages it is easiest to use them. Alternatively you can build PHP from source code.

This section contains the following topics:

- [Installing PHP and OCI8 on Oracle Linux](#)
- [Adding the OCI8 Extension to an Existing PHP Installation on Linux](#)
- [Manually Building PHP and OCI8 Together on Linux](#)

Installing PHP and OCI8 on Oracle Linux

On Oracle Linux install PHP 5.3 with:

```
# yum install php53 php53-gd
```

or

```
# up2date php53 php53-gd
```

If you don't have an Oracle Unbreakable Linux Network (ULN) subscription, you will need to install OCI8 manually as covered in the next section Adding the OCI8 Extension to an Existing PHP Installation on Linux.

If you are a subscriber to ULN then you have access to an OCI8 RPM in the **Oracle Software for Enterprise Linux 5** channel. Add this channel and then run:

```
# yum install php53-oci8-11gR2
```

or

```
# up2date php53-oci8-11gR2
```

Installing the `php53-oci8-11gR2` package will also install Oracle Instant Client libraries.

Restart Apache. If there are errors, they will display on your screen. They may also be written to `/var/log/httpd/error_log`. If you have problems, double check your `httpd.conf` and `php.ini` files.

Adding the OCI8 Extension to an Existing PHP Installation on Linux

If you have an existing PHP 5.3 installation without OCI8 you can add the latest PHP OCI8 extension by using PHP's PECL library, `http://pecl.php.net/oci8`. In general this can be used to add OCI8 to PHP 4.3.9 onwards. Note the example code in this manual requires PHP 5.3.

You will need PHP development files such as the `phpize` command. On Oracle Linux this can be found in the `php53-devel` package.

To install OCI8 perform the following steps:

1. Download and extract the latest OCI8 package, for example:
   `http://pecl.php.net/get/oci8-1.4.5.tgz`

2. Run the following commands:

```
# cd oci8-1.4.5
# phpize
# export ORACLE_HOME=/u01/app/oracle/product/11.2.0/xe
# ./configure --with-oci8
# make install
```

PHP OCI8 is built as a shared library. This makes it easy to upgrade without interfering with the rest of the PHP installation.

To get PHP to load the library, edit `/etc/php.ini` and add:

```
extension=oci8.so
```

Manually Building PHP and OCI8 Together on Linux

This section describes how to build PHP from source code on Linux. 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.

The instructions here result in a PHP binary with the OCI8 extension statically built in. A variant you might want to consider is to build PHP without OCI8, and then add OCI8 from PECL, as described in the previous section.

Perform the following steps to install PHP:

1. Download the most recent version of the PHP source code from http://www.php.net/downloads.php.

2. Login as the root user and run these commands. (They refer to PHP 5.3.6, but adapt them if you downloaded another version.)

```
# tar -jxvf php-5.3.6.tar.bz2
# cd php-5.3.6
# export ORACLE_HOME=/u01/app/oracle/product/11.2.0/xe
# ./configure \
--with-oci8 \
--with-apxs2=/usr/local/apache/bin/apxs \
--with-config-file-path=/usr/local/apache/conf
```

   If your Apache installation is not in /usr/local/apache then use the appropriate paths for your system. If you want to install PHP into a non-standard location, such as under your home directory, add the --prefix option when running 'configure'. For example, --prefix=$HOME/php53.

3. Build and install PHP:

```
# make
# make install
```

4. Copy PHP's supplied initialization development file:

```
# cp php.ini-development /usr/local/apache/conf/php.ini
```

5. Edit Apache's configuration file /usr/local/apache/conf/httpd.conf

   If a LoadModule line was not inserted by the PHP install, add it with:

```
LoadModule php5_module modules/libphp5.so
```

   Add the following lines to httpd.conf:

```
#
# This next section will call PHP for .php files
#
AddType application/x-httpd-php .php
```

6. Restart the Apache Server:

```
/usr/local/apache/bin/apachectl stop
/usr/local/apache/bin/apachectl start
```

   If there are errors, they will display on your screen. They may also be written to /usr/local/apache/logs/error_log. If you have problems, double check your httpd.conf and php.ini files.

Post PHP Installation Tasks on Windows and Linux

Once PHP is installed, make sure the `php.ini` configuration file has the following settings.

1. Setting the timezone is a new requirement of PHP 5.3.

   ```
   date.timezone = America/Los_Angeles
   ```

2. For testing it is helpful to set PHP's `display_errors` to show messages in the output text, instead of hiding them in Apache log files. In `php.ini` locate the `display_errors` setting and change it to `On` if necessary. If the directive doesn't exist, add the line:

   ```
   display_errors = On
   ```

3. The DRCP connection class should be configured in `php.ini`. The name is a user-chosen string to distinguish this application from any others that also use DRCP. If the class is not set, Oracle will not be able to share the DRCP pool effectively.

   Add a class name entry, such as:

   ```
   oci8.connection_class = ACXE
   ```

   Save the configuration file.

   If you are on Linux, make sure you have also done the steps in the earlier section Setting the Oracle Environment for Apache on Linux on page 2-9.

4. Restart Apache.

Testing the PHP Installation

This section contains the following topics:

- Checking PHP Configuration with phpinfo()
- Testing PHP Connections to Oracle

Checking PHP Configuration with phpinfo()

First, review the Apache error file `error_log` (in /var/log/httpd, /usr/local/apache/logs or C:\Program Files\Apache Software Foundation\Apache2.2\logs) to confirm there are no startup errors from Apache or PHP.

Decide where you want to create the PHP project files. This directory should be Apache accessible. For example on Linux you could use the Apache document root (/var/www/html or /usr/local/apache/htdocs) or $HOME/public_html if you configured a user directory. On Windows use the Apache document root C:\Program Files\Apache Software Foundation\Apache2.2\htdocs

In an editor create a new PHP file `pi.php` containing:

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

Load this file in a browser:

```
http://localhost/pi.php
```

Check the following:

- If you use a non default port such as 8888 change the URL to include it, for example `http://localhost:8888/pi.php`

- If your file is in `$HOME/public_html` then change the URL to include your user name, for example `http://localhost/~chris/pi.php`

- Or use both, for example `http://localhost:8888/~chris/pi.php`

You should see a page like:

If you see the text of the file echoed back it means you didn't configure Apache to send PHP files to PHP. Apache's `http.conf` file needs a line `AddType application/x-httpd-php .php` on Linux or `AddHandler fcgid-script .php` on Windows.

Correct `phpinfo()` output shows the php.ini location and if it was loaded. If it shows no `php.ini` loaded, then revisit some of the earlier steps, copy a `sample php.ini` file to the correct location and follow the steps in the section "Post PHP Installation Tasks on Windows and Linux" on page 2-14.

Scroll down to the OCI8 section. You should see

If there is no OCI8 section, check that you installed OCI8. If you installed it as a shared library, check `php.ini` has `extension=oci8.so` or `extension=php_oci8_11g.dll` on Windows, and that the `phpinfo()` output shows `extension_dir` set to the directory where the OCI8 library was installed.

Testing PHP Connections to Oracle

To check that the OCI8 extension works, create a new PHP file `testoci8.php` containing:

```php
<?php

$c = oci_connect('hr', 'welcome', 'localhost/xe');
if (!$c) {
    $m = oci_error();
    trigger_error('Could not connect to database: '. $m['message'], E_USER_ERROR);
}
$s = oci_parse($c, "SELECT * FROM employees");
if (!$s) {
    $m = oci_error($c);
    trigger_error('Could not parse statement: '. $m['message'], E_USER_ERROR);
}
$r = oci_execute($s);
if (!$r) {
    $m = oci_error($s);
    trigger_error('Could not execute statement: '. $m['message'], E_USER_ERROR);
}
$r = oci_fetch_all($s, $res);
if (!$r) {
    $m = oci_error($s);
    trigger_error('Could not fetch rows: '. $m['message'], E_USER_ERROR);
```

```
}
echo "<table border='1'>\n";
foreach ($res as $row) {
    echo "<tr>\n";
    foreach ($row as $item) {
        echo "  <td>".($item!==null?htmlentities($item,
                        ENT_QUOTES):" ")."</td>\n";
    }
    echo "</tr>\n";
}
echo "</table>\n";

?>
```

Everything between the `<?php ?>` tags will be processed by PHP and its output sent to the user's browser. Text outside of the tags will be sent to the user's browser verbatim. This includes leading and trailing white space. Files can have multiple sets of tags. Some applications use this to embed snippets of PHP inside HTML content. However a PHP-centric application will commonly use PHP echo or print statements to print out any needed HTML tags.

How the OCI8 function calls work is described in the section General Example of Running SQL in PHP OCI8 on page 3-5 and throughout the remainder of this manual.
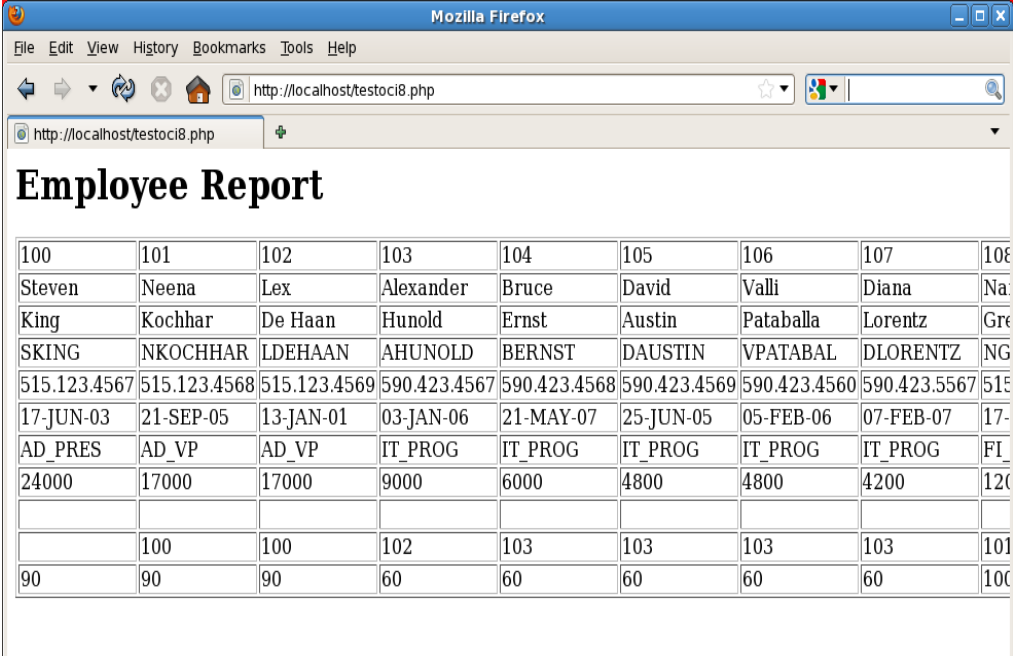
Load the following file in a browser:

```
http://localhost/testoci8.php
```

Depending how you installed Apache, you may need to use one of these alternatives:

- `http://localhost:8888/testoci8.php`
- `http://localhost/~chris/testoci8.php`
- `http://localhost:8888/~chris/testoci8.php`

You should see:

If you get a blank screen, check `php.ini` has `display_errors = On`. Reload the page and see if there was an error.

If you get an error `ORA-28000: the account is locked`, then unlock the HR account using the steps given previously in section Unlocking the HR User on page 2-2.

If you get a startup error on Linux like

`Warning: oci_connect():OciEnvNlsCreate() failed` then check that `ORACLE_HOME` was set correctly before Apache was started.

If you get an error like `ORA-12541: TNS:no listener` review the Oracle Database XE installation log and find the port that Oracle Database XE listener was installed with. For example if you used `1522` then change the connect call in `testoci8.php` to:

```
$c = oci_connect('hr', 'welcome', 'localhost:1522/xe');
```

If you get an error `ORA-01017: invalid username/password`, then change the `oci_connect()` call to use the password you assigned to `HR` in the section Unlocking the HR User on page 2-2.

Installing the NetBeans IDE

NetBeans is an extremely popular IDE for PHP web projects and has excellent coding features. A number of programming and mark-up languages, including HTML, JavaScript, and CSS editing features are supported. The latest version of NetBeans understands PHP 5.3 language constructs. NetBeans has PHP framework support, integration with tools like PHPUnit for testing, and integration with PHPDocumentor for documentation generation. It can be configured with an optional PHP debugger extension, which is very useful for PHP development. NetBeans also offers a SQL editor that works with Oracle database.

For more information and to download NetBeans, see `https://netbeans.org/`.

This section contains the following topics:

- Installing NetBeans IDE on Windows
- Installing NetBeans IDE on Linux
- Configuring NetBeans on Linux and Windows
- Using NetBeans

> **Note:** The instructions in these topics refer to NetBeans 7.0. If you plan to use a later version of NetBeans, see the appropriate information at see `https://netbeans.org/`, and especially the Documentation, Training, and Support page at `https://netbeans.org/kb/index.html`.

Installing NetBeans IDE on Windows

To install NetBeans IDE on Windows, perform the following steps:

1. Download NetBeans 7.0 with the Java SE Development Kit from the following location:

   `http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html`

2. In Windows Explorer, locate the downloaded file `jdk-6u25-nb-7_0-windows-ml.exe` and run it by double clicking it.

3. After installation, start NetBeans by navigating to **Start**, then **All Programs**, then **NetBeans**, and then **NetBeans IDE 7.0**.

4. From **Run Tools**, select **Plugins**.

5. Go to the **Available Plugins** tab and select all the options in the **PHP** category.

6. Click **Install** and agree to the license and location.

7. Restart the IDE when prompted.

Installing NetBeans IDE on Linux

To install NetBeans IDE on Linux, perform the following steps:

1. Download the NetBeans 7.0 PHP bundle from the following Web site:

   http://netbeans.org/downloads/

2. On Linux open a terminal window and run the installer:

   ```
   # sh netbeans-7.0-ml-php-linux.sh
   ```

3. If your JDK is not installed in a default location, specify the path when installing, for example:

   ```
   # sh netbeans-6.9-ml-php-linux.sh --javahome $HOME/jdk1.6.0_24
   ```

4. Accept the license

5. Confirm the install directory.

6. Confirm the location of your JDK.

NetBeans will install.

If you don't already have a JDK on your machine, download a bundle of NetBeans 7.0 that includes the JDK from

http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html

This bundle does not have PHP enabled. To enable PHP, go to **Tools** and run **Plugins**. Go to the **Available Plugins** tab and select all the options in the "**PHP**" category. Click **Install** and agree to the license and location. Restart the IDE when prompted.

Configuring NetBeans on Linux and Windows

Perform the following steps to configure NetBeans on Linux and Windows.

1. Start NetBeans by using the desktop icon or menu entry. Navigate to **Tools** and then **Options**.

2. In the **General** options choose your preferred browser.

3. In the **PHP** options, go to the **General** tab.

4. Set the PHP 5 interpreter to your PHP command line executable, for example `/usr/bin/php`, `/usr/local/bin/php`, or `C:\php-5.3.6\php.exe`.

Using NetBeans

Create a new **PHP Project** as follows:

1. Navigating to **File**, then **New Project**, or clicking the **New Project** Icon, or using Ctrl + Shift + N.

2. Choose **PHP** and **PHP Application**.

3. Change the Project name to `ACXE`.

4. Choose your folder where the source files will be kept. For ease of install, use a directory that Apache can access such as **/home/chris/public_html/ACXE**.

5. Set the PHP Version to PHP 5.3

6. Set the project URL to map the directory where the sources are located. For example, `http://localhost:8888/~chris/ACXE/`, if you use a non default Apache port. Do not use a framework.

7. The project will be created and an index.php file will be set to the default NetBeans template.
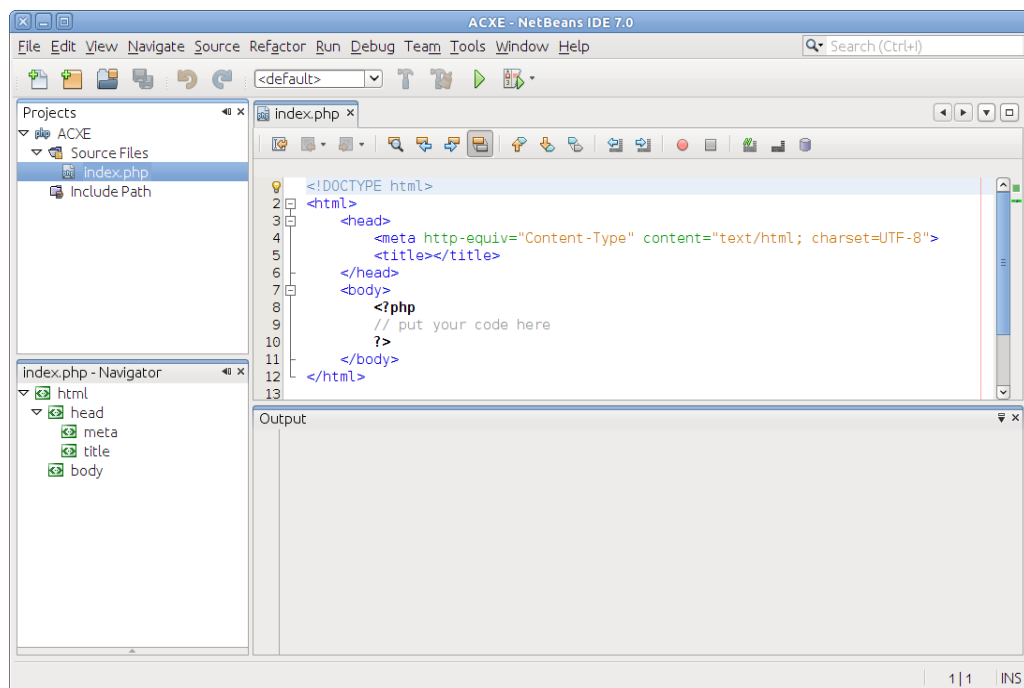
8. Put some content in `index.php` to verify it can be run. Inside the `<?php ?>` tags, replace the comment line `// put your code here` with

   ```
   echo "hi";
   ```

9. Run the file by selecting **Run Project** from **Run**, or press **F6**. A browser window will open with the newly added message:
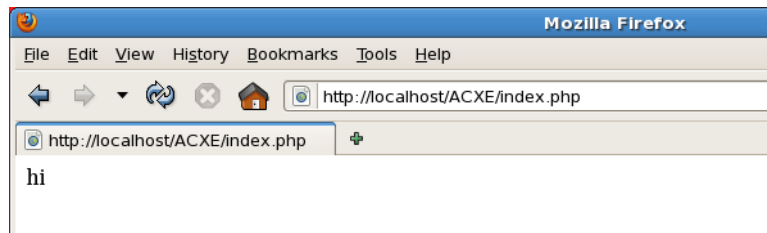
10. If the URL isn't correct and doesn't map to the file on disk, right click on the project name in the Project navigator and select **Properties**. Select **Run Configuration** and change the Project URL.

Throughout the rest of this manual you can add files to the ACXE project as needed by selecting the project name in the Project navigator and then going to **File** then **New File**, or using the **New File** icon, or Ctrl + N.

3

Building a Database Access Class

The Oracle database functionality for the AnyCo application will be abstracted into a class that handles all PHP OCI8 access.

This chapter contains the following topics:

- Connection Constants
- Creating the Db class
- General Example of Running SQL in PHP OCI8
- Running SQL with the Db Class
- Testing the Db Class

Connection Constants

Create a PHP file `ac_cred.inc.php`. The '.inc' component of the name is a common convention indicating that the file only contains definitions such as functions or classes. Giving it a final `.php` extension means that if a user somehow makes a HTTP request for that file, the web server will not send its text to the user (a security risk) but will run it as a PHP script. Because the file only contains definitions there will be no output sent to the user. This will prevent undesirable results or code exposure.

The `ac_cred.inc.php` file should initially look like:

```php
<?php

/**
 * ac_cred.inc.php: Secret Connection Credentials for a database class
 * @package Oracle
 */

/**
 * DB user name
 */
define('SCHEMA', 'hr');

/**
 * DB Password.
 *
 * Note: In practice keep database credentials out of directories
 * accessible to the web server.
 */
define('PASSWORD', 'welcome');

/**
```

```
 * DB connection identifier
 */
define('DATABASE', 'localhost/XE:pooled');

/**
 * DB character set for returned data
 */
define('CHARSET', 'UTF8');

/**
 * Client Information text for DB tracing
 */
define('CLIENT_INFO', 'AnyCo Corp.');

?>
```

To connect to an Oracle DB requires a user name, password, and a string identifying which DB to connect to. These are set as the constants SCHEMA, PASSWORD, and DATABASE using PHP's define() command. A character set is an optional but recommended connection parameter. Here UTF8 is chosen in the CHARSET constant.

Most PHP applications connect to the DB using one constant database account. In this example the database user is HR. This has some security implications that should not be discounted. Even though the file has the .php extension, in practice it is recommended to keep any files containing credentials or other sensitive information out of directories that Apache can access, and use PHP's require() command to load them. To avoid hard coding credentials in a file, some sites require applications read the values from environment variables set prior to starting Apache.

The database connection syntax used in DATABASE is Oracle's "Easy Connect" syntax. This specifies the host name where the database is running and identifies the service name of the database. Here the machine is given as localhost, meaning that PHP and the database need to be on the same machine. The service name for Oracle Database XE is XE, which is a preconfigured value. The :pooled suffix says that the connection should use the DRCP pool. If you did not start the DRCP pool in the section "Post PHP Installation Tasks on Windows and Linux" on page 2-14, then omit this suffix and change DATABASE to localhost/XE. This is the only application change that is necessary to determine whether or not to use DRCP.

A connection identifier could also be an Oracle Net tnsnames.ora alias, depending on your site standards.

The CLIENT_INFO constant will be used for end-to-end application tracing in the database. This is discussed in Chapter 13, "Monitoring Database Usage of the Application."

Creating the Db class

Create a new PHP file ac_db.inc.php to hold a database access class. Initially the file contains:

```php
<?php

/**
 * ac_db.inc.php: Database class using the PHP OCI8 extension
 * @package Oracle
 */

namespace Oracle;
```

```
require('ac_cred.inc.php');

/**
 * Oracle Database access methods
 * @package Oracle
 * @subpackage Db
 */
class Db {

    /**
     * @var resource The connection resource
     * @access protected
     */
    protected $conn = null;
    /**
     * @var resource The statement resource identifier
     * @access protected
     */
    protected $stid = null;
    /**
     * @var integer The number of rows to prefetch with queries
     * @access protected
     */
    protected $prefetch = 100;

}

?>
```

The `ac_db.inc.php` file sets the namespace to Oracle, defining the namespace of classes declared or used in the file. This avoids clashes if there are different implementations of classes in an application that happen to have the same name.

The database credentials are included with `require()`. If a required file doesn't exist a compilation error will occur. PHP also has an `include()` function that won't display an error for a missing file. Variants `require_once()` and `include_once()` can be used to prevent a sub file from being included more than once.

The `Db` class attributes will be discussed soon.

The comments are in a format that the open source tool PHPDocumentor will parse, for example `@package` defines the overall package that this file belongs to. NetBeans 7.0 can use these tags to automatically generate application documentation.

Add the following two methods into the `Db` class, between the `$prefetch` attribute and the closing brace:

```
    /**
     * Constructor opens a connection to the database
     * @param string $module Module text for End-to-End Application Tracing
     * @param string $cid Client Identifier for End-to-End Application Tracing
     */
    function __construct($module, $cid) {
        $this->conn = @oci_pconnect(SCHEMA, PASSWORD, DATABASE, CHARSET);
        if (!$this->conn) {
            $m = oci_error();
            throw new \Exception('Cannot connect to database: ' . $m['message']);
        }
        // Record the "name" of the web user, the client info and the module.
        // These are used for end-to-end tracing in the DB.
```

```
            oci_set_client_info($this->conn, CLIENT_INFO);
            oci_set_module_name($this->conn, $module);
            oci_set_client_identifier($this->conn, $cid);
    }

    /**
     * Destructor closes the statement and connection
     */
    function __destruct() {
        if ($this->stid)
            oci_free_statement($this->stid);
        if ($this->conn)
            oci_close($this->conn);
    }
```

When a PHP object instance is created its `__construct()` method will be called. The `Db` class constructor opens a connection to Oracle Database XE and keeps the connection resource in the `$conn` attribute for use when running statements. If connection doesn't succeed an error is generated. This error will be displayed to the user if PHP's `php.ini` parameter `display_errors` is `On` and sent to the Apache log files if `log_errors` is `On`. In the section "Post PHP Installation Tasks on Windows and Linux" on page 2-14, `display_errors` was set to `On` to help development. A production application should never display errors to the user because this is an information security leak.

The constructor passes the connection credentials to an `oci_pconnect()` function. The AnyCo application uses `oci_pconnect()` to create a "persistent" DRCP connection, as described in Database Resident Connection Pooling on page 2-3.

The character set is also passed to `oci_pconnect()`. It specifies the character set that data will be in when returned from Oracle to PHP. Setting it is optional but recommended. If the character set is not passed to `oci_pconnect()`, then PHP will determine the character set from the environment settings, which can be slower and may lead to an unexpected value being used.

A consequence of using the one database user name is that all statements in the application are recorded in the database as being run by HR. This makes analysis and tracing difficult or impossible. The `oci_set_client_identifier()` function allows an arbitrary string to be recorded with the connection and processed statement details in the database. By setting the identifier to the name of the web user this allows DBAs to explicitly associate an end user with database usage. The following article describes in detail where client identifiers can be used in Oracle Database:

http://www.oracle.com/technetwork/articles/dsl/php-web-auditing-171451.html

Also to aid database tracing, two other pieces of metadata are set for each connection: the Client Information and the Module Name. Chapter 13, "Monitoring Database Usage of the Application" shows where they are useful.

If a connection error occurs, an exception is thrown. The Exception class name is fully qualified. If the leading '\' was removed then an attempt to call `\Oracle\Exception` would occur, causing a run time error because a class called Exception has not been defined in the Oracle namespace. The namespace separator in PHP is a backslash(\) because it was the only feasible character available when namespaces were introduced in PHP 5.3.

The `Db` instance destructor explicitly closes any open connection. For a persistent DRCP pooled connection like shown, this returns the database server process to the DRCP pool for reuse. Because PHP variables internally use a reference counting mechanism, any variable that increases the reference count on the connection resource

must be freed before the underlying database connection will be physically closed. Here this means closing the statement resource, which is used later in this manual when the class is enhanced to run statements.

Because of PHP's reference counting mechanism, the destructor shown simply emulates the default behavior when an instance of the object is destroyed. Statement and connection resources will be terminated when variables referencing them are destroyed. This particular implementation of the destructor could therefore be omitted.

General Example of Running SQL in PHP OCI8

Running a statement in PHP OCI8 involves parsing the statement text and running it. In procedural style an INSERT would look like:

```
$c = oci_pconnect($un, $pw, $db, $cs);
$sql = "INSERT INTO mytable (c1, c2) VALUES (1, 'abc')";
$s = oci_parse($c, $sql);
oci_execute($s);
```

If a statement will be re-run in the database system with different data values, then use bind variables:

```
$c = oci_pconnect($un, $pw, $db, $cs);
$sql = "INSERT INTO mytable (c1, c2) VALUES (:c1_bv, :c2_bv)";
$s = oci_parse($c, $sql);
$c1 = 1;
$c2 = 'abc';
oci_bind_by_name($s, ":c1_bv", $c1, -1);
oci_bind_by_name($s, ":c2_bv", $c2, -1);
oci_execute($s);
```

Binding associates PHP variables with the bind identifier place holders in the SQL statement. The bind lengths are set to -1 telling PHP to infer internal buffer sizes from the lengths of the PHP values. When using oci_bind_by_name() to return data from the database (such as when assigning a PL/SQL function return value to a bind variable), the actual expected data length should be specified so enough internal space can be allocated for the PHP variable.

Bind variables are important for performance and security reasons. They allow the database to reuse statement metadata for repeated statements where only the variable values change. An alternative PHP coding style would concatenate PHP variable values into the SQL statement text. Each such statement would appear unique to the DB and caching would be reduced. This severely impacts DB performance. Also concatenation introduces SQL Injection security risks, where concatenation with malicious user input changes the semantics of the SQL statement.

In PHP, a SQL query is similar to execution but has a subsequent fetch call, of which PHP has several variants. For example to fetch all rows at once:

```
$c = oci_pconnect($un, $pw, $db, $cs);
$sql = "SELECT * FROM mytable WHERE c1 = :c1_bv AND c2 = :c2_bv";
$s = oci_parse($c, $sql);
$c1 = 1;
$c2 = 'abc';
oci_bind_by_name($s, ":c1_bv", $c1, -1);
oci_bind_by_name($s, ":c2_bv", $c2, -1);
oci_execute($s);
oci_fetch_all($s, $res, 0, -1, OCI_FETCHSTATEMENT_BY_ROW);
```

The query results would be in $res. The OCI_FETCHSTATEMENT_BY_ROW constant indicates the results will be in an array with entries for each row. The rows themselves are represented by a sub-arrays.

If the query returns a large number of rows the memory use might be undesirably large. Other PHP OCI8 functions like oci_fetch_array() could be called instead. This function returns only one row of the result set. After the script has processed the row, it could call oci_fetch_array() again to fetch the next row.

> **Note:** The bind variable name argument in an oci_bind_by_name() call does not need to have a colon prefix, but it can help visual code inspection to include it.

Running SQL with the Db Class

To make our Db class in ac_db.inc.php useful add these two methods to the class:

```
/**
 * Run a SQL or PL/SQL statement
 *
 * Call like:
 *     Db::execute("insert into mytab values (:c1, :c2)",
 *                 "Insert data", array(array(":c1", $c1, -1),
 *                                      array(":c2", $c2, -1)))
 *
 * For returned bind values:
 *     Db::execute("begin :r := myfunc(:p); end",
 *                 "Call func", array(array(":r", &$r, 20),
 *                                    array(":p", $p, -1)))
 *
 * Note: this performs a commit.
 *
 * @param string $sql The statement to run
 * @param string $action Action text for End-to-End Application Tracing
 * @param array $bindvars Binds. An array of (bv_name, php_variable, length)
 */
public function execute($sql, $action, $bindvars = array()) {
    $this->stid = oci_parse($this->conn, $sql);
    if ($this->prefetch >= 0) {
        oci_set_prefetch($this->stid, $this->prefetch);
    }
    foreach ($bindvars as $bv) {
        // oci_bind_by_name(resource, bv_name, php_variable, length)
        oci_bind_by_name($this->stid, $bv[0], $bv[1], $bv[2]);
    }
    oci_set_action($this->conn, $action);
    oci_execute($this->stid);                // will auto commit
}

/**
 * Run a query and return all rows.
 *
 * @param string $sql A query to run and return all rows
 * @param string $action Action text for End-to-End Application Tracing
 * @param array $bindvars Binds. An array of (bv_name, php_variable, length)
 * @return array An array of rows
 */
public function execFetchAll($sql, $action, $bindvars = array()) {
    $this->execute($sql, $action, $bindvars);
```

```
oci_fetch_all($this->stid, $res, 0, -1, OCI_FETCHSTATEMENT_BY_ROW);
$this->stid = null;  // free the statement resource
return($res);
}
```

These methods do the same as the previous procedural samples, with the addition of another piece of database tracing metadata called the Action, and a way to tune performance of queries, called prefetching. Prefetching is discussed later in Chapter 8, "Query Performance and Prefetching".

All the tracing metadata set in the Db class is optional, however it is easier to include it in the design instead of having to retrofit it. It can be painful to troubleshoot performance or access issues on production applications without it.

Setting the statement identifier resource $this->stid to null initiates the same internal cleanup as oci_free_statement() (used in the destructor) and also sets the attribute to null so later methods can test for validity.

Our Db::execute() method allows us to write our INSERT statement as:

```
$db = new \Oracle\Db("Test Example", "Chris");
$sql = "INSERT INTO mytable (c1, c2) VALUES (:c1_bv, :c2_bv)";
$c1 = 1;
$c2 = 'abc';
$db->execute($sql, "Insert Example", array(array(":c1_bv", $c1, -1),
                                           array(":c2_bv", $c2, -1)));
```

The query example would be:

```
$db = new \Oracle\Db("Test Example", "Chris");
$sql = "SELECT * FROM mytable WHERE c1 = :c1_bv AND c2 = :c2_bv";
$c1 = 1;
$c2 = 'abc';
$res = $db->execFetchAll($sql, "Query Example",
                         array(array(":c1_bv", $c1, -1),
                               array(":c2_bv", $c2, -1)));
```

The Db instance creation uses a fully qualified namespace description.

The bind variables are encapsulated in an array of arrays. Each sub-array describes one bind variable.

As coded, the Db class automatically commits each time oci_execute() it is called. This has performance and transactional consistency implications if the class is to be reused in future applications. To make Db more general purpose you could consider changing Db::execute() to do:

```
...
oci_execute($this->stid, OCI_NO_AUTO_COMMIT);
...
```

In this case you would need to add commit and rollback methods to the Db class that call oci_commit() and oci_rollback() respectively. The examples in this manual don't require these changes. Note that in PHP any oci_connect() or any oci_pconnect() call that uses the same connection credentials will reuse the same underlying connection to the database. So if an application creates two instances of Db, they will share the same transaction state. Rolling back or committing one instance will affect transactions in the other. The oci_new_connect() function is different and will create its own new connection each time it is called.

Testing the Db Class

Test the `Db` class by creating a new PHP file called `test_db.php`:

```php
<?php

// test_db.php

require('ac_db.inc.php');

$db = new \Oracle\Db("test_db", "Chris");
$sql = "SELECT first_name, phone_number FROM employees ORDER BY employee_id";
$res = $db->execFetchAll($sql, "Query Example");
// echo "<pre>"; var_dump($res); echo "</pre>\n";

echo "<table border='1'>\n";
echo "<tr><th>Name</th><th>Phone Number</th></tr>\n";
foreach ($res as $row) {
    $name = htmlspecialchars($row['FIRST_NAME'], ENT_NOQUOTES, 'UTF-8');
    $pn   = htmlspecialchars($row['PHONE_NUMBER'], ENT_NOQUOTES, 'UTF-8');
    echo "<tr><td>$name</td><td>$pn</td></tr>\n";
}
echo "</table>";

?>
```

The `require()` command includes the content of `ac_db.inc.php` giving the script access to the `Db` class.

The module name parameter for the `Db` instance creation is set to the file name base `test_db`. This allows anyone doing database tracing to identify where the connection was initiated from. The connection identifier is arbitrarily set to a fictitious user's name. The Action parameter to `$db->execFetchAll()` is set to the operation in the file.

No bind variables are passed in this example so the optional bind parameter is not specified in the `$db->execFetchAll()` method call. The definition of `Db::execFetchAll()` sets the bind variable list to an empty array when there is no final argument and therefore won't attempt to bind any data.

The query results are returned in `$res` as an array of row data. You can see the array structure by un-commenting the `var_dump()` function, which is useful for simple PHP debugging. The `$res` array is iterated over in a `foreach()` loop which process each row in turn. The two columns in each row's sub-array are accessed by `$row['FIRST_NAME']` and `$row['PHONE_NUMBER']`. By default, columns in Oracle database tables are case insensitive. They will be returned to PHP as upper case array indices. If the table had been created in Oracle with case a sensitive column name like

```sql
CREATE TABLE mytab ("MyCol" NUMBER);
```

then in PHP you would need to use a case sensitive array index `$row['MyCol']`.

In `test_db.php`, the returned data is processed with `htmlspecialchars()` to make sure that any text that happens to look like HTML is treated as displayable text and not as HTML markup. This escaping of output is very important for security in web applications to make sure there are no cross-site scripting (XSS) security issues.

The exact `htmlspecialchars()` options you use would depend on context. PHP also has an `htmlentities()` function that might be useful. The character set should match the HTML page character set. The AnyCo application will do this.

Load `test_db.php` in a browser: `http://localhost/test_db.php`. Or, in NetBeans, right click on the file in the **Projects** navigator and select **Run**.

It displays:

Name	Phone Number
Steven	515.123.4567
Neena	515.123.4568
Lex	515.123.4569
Alexander	590.423.4567
Bruce	590.423.4568
David	590.423.4569
Valli	590.423.4560
Diana	590.423.5567
Nancy	515.124.4569
Daniel	515.124.4169

If you have problems connecting, resolve any PHP interpreter errors. Make sure all methods are located inside the class definition braces. Review the section Testing PHP Connections to Oracle on page 2-16 for other common problems.

4

Building the AnyCo Application

This chapter contains the following topics:

A Cascading Style Sheet

To start creating the AnyCo application, create a cascading style sheet file `style.css`. It contains:

```css
/* style.css */

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

table {
    border-collapse: collapse;
    margin: 5px;
}

tr:nth-child(even) {background-color: #FFFFFF}
tr:nth-child(odd) {background-color: #EDF3FE}

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

#header {
    font-weight: bold;
    font-size: 160%;
    text-align: center;
    border-bottom: solid #334B66 4px;
    margin-bottom: 10px;
}
```

```
#menu {
    position: absolute;
    left: 5px;
    width: 180px;
    display: block;
    background-color: #dddddd;
}
#user {
    font-size: 90%;
    font-style:italic;
    padding: 3px;
}

#content {
    margin-left: 200px;
}
```

This gives a simple styling to the application, keeping a menu to the left hand side of the main content. Alternate rows of table output are colored differently. See Figure 1–1 in Chapter 1, "Introducing PHP with Oracle Database XE."

An Application Class for Sessions

For the AnyCo application we will create two classes, Session and Page, to give some reusable components.

The Session class is where web user authentication will be added. It also provides the components for saving and retrieving web user "session" information on the mid-tier, allowing the application to be stateful. PHP sessions are not directly related to Oracle sessions which were discussed in the DRCP overview. Data such as starting row number of the currently displayed page of query results can be stored in the PHP session. The next HTTP request can retrieve this value from the session storage and show the next page of results.

Create a new PHP file called `ac_equip.inc.php` initially containing:

```php
<?php

/**
 * ac_equip.inc.php: PHP classes for the employee equipment example
 * @package Equipment
 */
namespace Equipment;

/**
 * URL of the company logo
 */
//define('LOGO_URL', 'http://localhost/ac_logo_img.php');

/**
 * @package Equipment
 * @subpackage Session
 */
class Session {
    /**
     *
     * @var string Web user's name
     */
    public $username = null;
    /**
```

```
 *
 * @var integer  current record number for paged employee results
 */
public $empstartrow = 1;
/**
 *
 * @var string CSRF token for HTML forms
 */
public $csrftoken = null;

}

?>
```

The file starts with a namespace declaration, `Equipment` in this case.

The commented out LOGO_URL constant will be described later in the Chapter 12, "Uploading and Displaying BLOBs."

The `$username` attribute will store the web user's name. The `$empstartrow` attribute stores the first row number of the currently displayed set of employees. This allows employee data to be "paged" through with **Next** and **Previous** buttons as shown in Figure 1–1, "Overview of the Sample Application". The `$csrftoken` value will be described in the Chapter 9, "Inserting Data."

Add two authentication methods to the Session class:

```
/**
 * Simple authentication of the web end-user
 *
 * @param string $username
 * @return boolean True if the user is allowed to use the application
 */
public function authenticateUser($username) {
    switch ($username) {
        case 'admin':
        case 'simon':
            $this->username = $username;
            return(true);  // OK to login
        default:
            $this->username = null;
            return(false); // Not OK
    }
}

/**
 * Check if the current user is allowed to do administrator tasks
 *
 * @return boolean
 */
public function isPrivilegedUser() {
    if ($this->username === 'admin')
        return(true);
    else
        return(false);
}
```

The `authenticateUser()` method implements extremely unsophisticated and insecure user authentication. Typically PHP web applications do their own user authentication. Here only `admin` and `simon` will be allowed to use the application. For more information on authentication refer to

http://www.oracle.com/technetwork/articles/mclaughlin-phpid1-091467.html

The `isPrivilegedUser()` method returns a boolean value indicating if the current user is considered privileged. In the AnyCo application this will be used to determine if the user can see extra reports and can upload new data. Only the AnyCo "admin" will be allowed to do these privileged operations.

Providing a Stateful Web Experience with PHP Sessions

PHP can store session values that appear persistent as users move from HTML page to HTML page. By default the session data is stored in a file on the PHP server's disk. The session data is identified by a unique cookie value, or a value passed in the URL if the user has cookies turned off. The cookie allows PHP to associate its local session storage with the correct web user.

PHP sessions allow user HTTP page requests to be handled seamlessly by random mid-tier Apache processes while still allowing access to the current session data for each user. PHP allow extensive customization of session handling, including ways to perform session expiry and giving you ways to store the session data in a database. Refer to the PHP documentation for more information.

To store, fetch and clear the session values in the AnyCo application, add these three methods to the Session class:

```
/**
 * Store the session data to provide a stateful web experience
 */
public function setSession() {
    $_SESSION['username']   = $this->username;
    $_SESSION['empstartrow'] = (int)$this->empstartrow;
    $_SESSION['csrftoken']  = $this->csrftoken;
}

/**
 * Get the session data to provide a stateful web experience
 */
public function getSession() {
    $this->username = isset($_SESSION['username']) ?
        $_SESSION['username'] : null;
    $this->empstartrow = isset($_SESSION['empstartrow']) ?
        (int)$_SESSION['empstartrow'] : 1;
    $this->csrftoken = isset($_SESSION['csrftoken']) ?
        $_SESSION['csrftoken'] : null;
}

/**
 * Logout the current user
 */
public function clearSession() {
    $_SESSION = array();
    $this->username = null;
    $this->empstartrow = 1;
    $this->csrftoken = null;
}
```

These reference the superglobal associative array `$_SESSION` which gives access to PHP's session data. When any of the Session attributes change, the AnyCo application will call `setSession()` to record the changed state. Later when another application request starts processing, its script will call the `getSession()` method to retrieve the

saved attribute values. The ternary "?:" tests will use the session value if there is one, or else use a hardcoded default.

Finally, add the following method to the Session class to aid CSRF protection in HTML forms. This will be described in the section CSRF example with ac_add_one.php on page 9-5 in Chapter 9, "Inserting Data."

```
/**
 * Records a token to check that any submitted form was generated
 * by the application.
 *
 * For real systems the CSRF token should be securely,
 * randomly generated so it can't be guessed by a hacker
 * mt_rand() is not sufficient for production systems.
 */
public function setCsrfToken() {
    $this->csrftoken = mt_rand();
    $this->setSession();
}
```

Adding a Page Class

A Page class will provide methods to output blocks of HTML output so each web page of the application has the same appearance.

Add the new Page class to the ac_equip.inc.php file after the closing brace of the Session class, but before the PHP closing tag '?>'. The class initially looks like:

```
/**
 * @package Equipment
 * @subpackage Page
 */
class Page {
    /**
     * Print the top section of each HTML page
     * @param string $title The page title
     */
    public function printHeader($title) {
        $title = htmlspecialchars($title, ENT_NOQUOTES, 'UTF-8');
        echo <<<EOF
<!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=utf-8">
  <link rel="stylesheet" type="text/css" href="style.css">
  <title>$title</title>
</head>
<body>
<div id="header">
EOF;
// Important: don't have white space on the 'EOF;' line before or after the tag

        if (defined('LOGO_URL')) {
            echo '<img src="' . LOGO_URL . '" alt="Company Icon"> ';
        }
        echo "$title</div>";
    }
    /**
     * Print the bottom of each HTML page
```

```
     */
    public function printFooter() {
        echo "</body></html>\n";
    }

}
```

The `printHeader()` method prints the HTML page prologue, includes the style sheet, and prints the page title.

A PHP 'heredoc' is used to print the big block of HTML content. The variable $title in the text will be expanded and its value displayed. The closing tag 'EOF;' must be at the start of the line and also not have any trailing white space. Otherwise the PHP parser will treat the rest of the file as part of the string text and will produce a random parsing error when it encounters something that looks like a PHP variable.

A logo will also be displayed in the header when LOGO_URL is defined in a later example, remember it is currently commented out at the top of `ac_equip.inc.php`.

The `printFooter()` methods simply ends the HTML page body. A general application could augment this to display content that should be printed at the bottom of each page, such as site copyright information.

The AnyCo application has a left hand navigation menu. Add a method to the Page class to print this:

```
    /**
     * Print the navigation menu for each HTML page
     *
     * @param string $username The current web user
     * @param type $isprivilegeduser True if the web user is privileged
     */
    public function printMenu($username, $isprivilegeduser) {
        $username = htmlspecialchars($username, ENT_NOQUOTES, 'UTF-8');
        echo <<<EOF
<div id='menu'>
<div id='user'>Logged in as: $username </div>
<ul>
<li><a href='ac_emp_list.php'>Employee List</a></li>
EOF;
        if ($isprivilegeduser) {
            echo <<<EOF
<li><a href='ac_report.php'>Equipment Report</a></li>
<li><a href='ac_graph_page.php'>Equipment Graph</a></li>
<li><a href='ac_logo_upload.php'>Upload Logo</a></li>
EOF;
        }
        echo <<<EOF
<li><a href="index.php">Logout</a></li>
</ul>
</div>
EOF;
    }
```

The user name and privileged status of the user will be passed in to customize the menu for each user. These values will come from the Session class.

Later chapters in this manual will create the PHP files referenced in the links. Clicking those link without having the files created will give an expected error.

The three classes: Db, Session, and Page, used by the AnyCo application are now complete.

The Application Login Page

The start page of the AnyCo application is the login page. Create a new PHP file called `index.php`. In NetBeans replace the existing contents of this file. The `index.php` file should contain:

```php
<?php

/**
 * index.php: Start page for the AnyCo Equipment application
 *
 * @package Application
 */

session_start();
require('ac_equip.inc.php');

$sess = new \Equipment\Session;
$sess->clearSession();

if (!isset($_POST['username'])) {
    $page = new \Equipment\Page;
    $page->printHeader("Welcome to AnyCo Corp.");
    echo <<< EOF
<div id="content">
<h3>Select User</h3>
<form method="post" action="index.php">
<div>
<input type="radio" name="username" value="admin">Administrator<br>
<input type="radio" name="username" value="simon">Simon<br>
<input type="submit" value="Login">
</div>
</form>
</div>
EOF;
// Important: don't have white space on the 'EOF;' line before or after the tag
    $page->printFooter();
} else {
    if ($sess->authenticateUser($_POST['username'])) {
        $sess->setSession();
        header('Location: ac_emp_list.php');
    } else {
        header('Location: index.php');
    }
}

?>
```

The `index.php` file begins with a `session_start()` call. This must occur in code that wants to use the `$_SESSION` superglobal and should be called before any output is created.

An instance of the Session class is created and any existing session data is discarded by the `$sess->clearSession()` call. This allows the file to serve as a logout page. Any time `index.php` is loaded, the web user will be logged out of the application.

The bulk of the file is in two parts, one creating an HTML form and the other processing it. The execution path is determined by the PHP superglobal `$_POST`. The first time this file is run `$_POST['username']` won't be set so the HTML form along

with the page header and footer will be displayed. The form allows the web user login as `Administrator` or `Simon`.

The submission action target for the form is `index.php` itself. So after the user submits the form in their browser, this same PHP file is run. Since the submission method is "`post`", PHP will populate the superglobal `$_POST` with the form values. This time the second branch of the '`if`' statement will be run.

The user is then authenticated. The radio button input values '`admin`' and '`simon`' are the values that will be passed to `$sess->authenticateUser()`. A valid user will be recorded in the session data. PHP then sends back an HTTP header causing a browser redirect to `ac_emp_list.php`. This file will be created in the next section.

If the user is not validated by `$sess->authenticateUser()` then the login form is redisplayed.

Note that scripts should not display text before a `header()` call is run.

To run the application as it stands, load `index.php` in a browser. In NetBeans, use Run->Run Project, or press **F6**. The browser will show:

Welcome to AnyCo Corp.

Select User

○ Administrator
○ Simon

[Login]

5

Paging Through Employee Data

This chapter creates the main display page of the AnyCo application as shown in Figure 1–1, "Overview of the Sample Application". It will show five employee records at a time and allow you to page through the list of employees.

This chapter contains the following topics:

- Creating the Employee Listing
- Running the Employee List

Creating the Employee Listing

Create a new PHP file `ac_emp_list.php` initially containing:

```php
<?php

/**
 * ac_emp_list.php: list of employees
 * @package Employee
 */

define('NUMRECORDSPERPAGE', 5);

session_start();
require('ac_db.inc.php');
require('ac_equip.inc.php');

$sess = new \Equipment\Session;
$sess->getSession();
if (!isset($sess->username) || empty($sess->username)) {
    header('Location: index.php');
    exit;
}

$page = new \Equipment\Page;
$page->printHeader("AnyCo Corp. Employees List");
$page->printMenu($sess->username, $sess->isPrivilegedUser());
printcontent($sess, calcstartrow($sess));
$page->printFooter();

// Functions

?>
```

The `NUMRECORDSPERPAGE` constant determines how many employee records to show.

After the `$sess->getSession()` call retrieves the stored session data there is some basic validation to confirm the user is authorized to view this page. Here the only requirement is that the `username` is set. If it isn't, the browser is redirected to the login page, `index.php`. Here is where a production application would do more validation, perhaps checking a timestamp and forcing users to re-login after a certain idle period. The user name could have been encrypted, making it harder for a hacker to view session data or to impersonate a session.

The body of the file prints the HTML page header, menu, content and footer of the page.

This file will show PHP's traditional procedural style instead of continuing the object oriented approach previously used. Under the `Functions` comment add the function to print the page content:

```
/**
 * Print the main body of the page
 *
 * @param Session $sess
 * @param integer $startrow The first row of the table to be printed
 */
function printcontent($sess, $startrow) {
    echo "<div id='content'>";

    $db = new \Oracle\Db("Equipment", $sess->username);
    $sql = "SELECT employee_id, first_name || ' ' || last_name AS name,
            phone_number FROM employees ORDER BY employee_id";
    $res = $db->execFetchPage($sql, "Equipment Query", $startrow,
            NUMRECORDSPERPAGE);
    if ($res) {
        printrecords($sess, ($startrow === 1), $res);
    } else {
        printnorecords();
    }

    echo "</div>";  // content
    // Save the session, including the current data row number
    $sess->empstartrow = $startrow;
    $sess->setSession();
}
```

This runs a query on the `EMPLOYEES` table. The `Db::execFetchPage()` method is similar to `Db::execFetchAll()` and will be shown in a moment. If there are records to display then `printrecords()` will show them, else `printnorecords()` will display a message that there was nothing to show. The final stage of printing the content is to update the session with the new starting row number.

The call to `printcontent()` at the top level uses `calcstartrow()` to decide which row number to start at. Add this function to `ac_emp_list.php`:

```
/**
 * Return the row number of the first record to display.
 *
 * The calculation is based on the current position
 * and whether the Next or Previous buttons were clicked
 *
 * @param Session $sess
 * @return integer The row number that the page should start at
 */
function calcstartrow($sess) {
    if (empty($sess->empstartrow)) {
```

```
            $startrow = 1;
        } else {
            $startrow = $sess->empstartrow;
            if (isset($_POST['prevemps'])) {
                $startrow -= NUMRECORDSPERPAGE;
                if ($startrow < 1) {
                    $startrow = 1;
                }
            } else if (isset($_POST['nextemps'])) {
                $startrow += NUMRECORDSPERPAGE;
            }
        }
        return($startrow);
}
```

The rows will be displayed with a form having **Next** and **Previous** buttons. The calculation for the starting row depends on which button was clicked and whereabouts in the data set the user has got to.

Add `printrecords()` to `ac_emp_list.php` to show any fetched records:

```
/**
 * Print the Employee records
 *
 * @param Session $sess
 * @param boolean $atfirstrow True if the first array entry is the first table row
 * @param array $res Array of rows to print
 */
function printrecords($sess, $atfirstrow, $res) {
    echo <<< EOF
        <table border='1'>
        <tr><th>Name</th><th>Phone Number</th><th>Equipment</th></tr>
EOF;
    foreach ($res as $row) {
        $name = htmlspecialchars($row['NAME'], ENT_NOQUOTES, 'UTF-8');
        $pn   = htmlspecialchars($row['PHONE_NUMBER'], ENT_NOQUOTES, 'UTF-8');
        $eid  = (int)$row['EMPLOYEE_ID'];
        echo "<tr><td>$name</td>";
        echo "<td>$pn</td>";
        echo "<td><a href='ac_show_equip.php?empid=$eid'>Show</a> ";
        if ($sess->isPrivilegedUser()) {
            echo "<a href='ac_add_one.php?empid=$eid'>Add One</a>";
            echo "<a href='ac_add_multi.php?empid=$eid'> Add Multiple</a>\n";
        }
        echo "</td></tr>\n";
    }
    echo "</table>";
    printnextprev($atfirstrow, count($res));
}
```

This function's logic is similar to that shown in `test_db.php`. Remember the 'EOF;' token must be at the start of the line and not have any trailing white space.

Privileged users see extra links to issue pieces of equipment to each employee. At the end of the HTML table any **Next** and **Previous** buttons are shown by calling `printnextprev()`.

Add `printnextprev()` to `ac_emp_list.php`:

```
/**
 * Print Next/Previous buttons as needed to page through the records
 *
```

```
 * @param boolean $atfirstrow True if the first array entry is the first table row
 * @param integer $numrows Number of rows the current query retrieved
 */
function printnextprev($atfirstrow, $numrows) {
    if (!$atfirstrow || $numrows == NUMRECORDSPERPAGE) {
        echo "<form method='post' action='ac_emp_list.php'><div>";
        if (!$atfirstrow)
            echo "<input type='submit' value='< Previous' name='prevemps'>";
        if ($numrows == NUMRECORDSPERPAGE)
            echo "<input type='submit' value='Next >' name='nextemps'>";
        echo "</div></form>\n";
    }
}
```

The `printnextprev()` logic handles the boundary cases including

- not displaying a **Previous** button on the first page

- not showing a **Next** button when a full page wasn't displayed.

Finally, add `printnorecords()` to `ac_emp_list.php` to display a message when there are no records to show:

```
/**
 * Print a message that there are no records
 *
 * This can be because the table is empty or the final page of results
 * returned no more records
 */
function printnorecords() {
    if (!isset($_POST['nextemps'])) {
        echo "<p>No Records Found</p>";
    } else {
        echo <<<EOF
            <p>No More Records</p>
            <form method='post' action='ac_emp_list.php'>
            <input type='submit' value='< Previous' name='prevemps'></form>
EOF;
    }
}
```

Note: The `EOF;` token must be at the start of a line and not have trailing white space.

There are two cases here, one where the table has no rows, and the other when the user is paging through the table and clicking **Next** gives no more data to display. This latter case will occur when the number of rows in the table is a multiple of `NUMRECORDSPERPAGE`.

Before we can run the application we need to create the `Db::execFetchPage()` method. In the file `ac_db.inc.php` add a new method to the `Db` class:

```
    /**
     * Run a query and return a subset of records.  Used for paging through
     * a resultset.
     *
     * The query is used as an embedded subquery.  Don't permit user
     * generated content in $sql because of the SQL Injection security issue
     *
```

```
 * @param string $sql The query to run
 * @param string $action Action text for End-to-End Application Tracing
 * @param integer $firstrow The first row number of the dataset to return
 * @param integer $numrows The number of rows to return
 * @param array $bindvars Binds. An array of (bv_name, php_variable, length)
 * @return array Returns an array of rows
 */
public function execFetchPage($sql, $action, $firstrow = 1, $numrows = 1,
$bindvars = array()) {
    //
    $query = 'SELECT *
        FROM (SELECT a.*, ROWNUM AS rnum
              FROM (' . $sql . ') a
              WHERE ROWNUM <= :sq_last)
        WHERE :sq_first <= RNUM';

    // Set up bind variables.
    array_push($bindvars, array(':sq_first', $firstrow, -1));
    array_push($bindvars, array(':sq_last', $firstrow + $numrows - 1, -1));
    $res = $this->execFetchAll($query, $action, $bindvars);
    return($res);
}
```

Oracle database doesn't have a `LIMIT` clause to return a subset of rows so nesting the caller's query is needed. PHP's `array_push()` function appends the extra bind variables used for the start and end row numbers in the outer query to any bind variables for the caller's query.

Because the SQL text is concatenated watch out for SQL injection issues. Never pass user input into this function.

Running the Employee List

Save all the files and run the application. Login first as `Simon`. You will see:

Click **Logout**. Re-login as `Administrator`. You will see:

Use the **Next** and **Previous** buttons to page through the data.

Try changing NUMRECORDSPERPAGE to see the effect on paging.

6

Showing Equipment Records by Using a REF CURSOR

This chapter creates the report run by clicking the **Show** link next to an employees name on the **AnyCo Corp. Employees List** page from the previous chapter.

The previous chapter showed how to fetch data from a SQL query. This chapter shows how to use a `REF CURSOR` in PHP. The `REF CURSOR` will fetch the names of the equipment that have been issued to an employee.

This chapter contains the following topics:

- Introduction to PL/SQL Packages and Package Bodies
- Introduction to PL/SQL Stored Procedures
- Introduction to REF CURSORs
- The Equipment Table
- Calling the REF CURSOR in PHP

Introduction to PL/SQL Packages and Package Bodies

A PL/SQL package stores related items as a single logical entity. A package is composed of two distinct pieces:

- The **package specification** defines what is contained in the package; it is analogous to a header file in a language such as C++. The specification defines all public items. The specification is the published interface to a package.

- The **package body** contains the code for the procedures and functions defined in the specification, and the code for private procedures and functions that are not declared in the specification. This private code is only visible within the package body.

The package specification and body are stored as separate objects in the data dictionary and can be seen in the `user_source` view. The specification is stored as the `PACKAGE` type, and the body is stored as the `PACKAGE BODY` type.

While it is possible to have a specification without a body, as when declaring a set of public constants, it is not possible to have a body with no specification.

Introduction to PL/SQL Stored Procedures

A stored procedure is a named set of PL/SQL statements designed to perform an action. Stored procedures are stored inside the database. They define a programming interface for the database rather than allowing the client application to interact with

database objects directly. Stored procedures are typically used for data validation or to encapsulate large, complex processing instructions that combine several SQL queries.

Stored functions have a single return value parameter. Unlike functions, procedures may or may not return values.

Introduction to REF CURSORs

Using `REF CURSOR`s is one of the most powerful, flexible, and scalable ways to return query results from an Oracle Database to a client application.

A `REF CURSOR` is a PL/SQL data type whose value is the memory address of a query work area on the database. In essence, a `REF CURSOR` is a pointer or a handle to a result set on the database.

`REF CURSOR`s have the following characteristics:

- A `REF CURSOR` refers to a memory address on the database. Therefore, the client must be connected to the database during the lifetime of the `REF CURSOR` in order to access it.

- A `REF CURSOR` involves an additional database round-trip. While the `REF CURSOR` is returned to the client, the actual data is not returned until the client opens the `REF CURSOR` and requests the data. Note that data is not be retrieved until the user attempts to read it.

- A `REF CURSOR` is not updatable. The result set represented by the `REF CURSOR` is read-only. You cannot update the database by using a `REF CURSOR`.

- A `REF CURSOR` is not backward scrollable. The data represented by the `REF CURSOR` is accessed in a forward-only, serial manner. You cannot position a record pointer inside the `REF CURSOR` to point to random records in the result set.

- A `REF CURSOR` is a PL/SQL data type. You create and return a `REF CURSOR` inside a PL/SQL code block.

The Equipment Table

This manual's example scenario is that AnyCo Corp issues each employee various pieces of equipment to do their job. An `EQUIPMENT` table will hold the equipment names and which employee it was issued to.

In SQL*Plus connect as the `HR` user and run the following script:

```
sqlplus hr/welcome@localhost/XE

CREATE TABLE equipment(
    id          NUMBER PRIMARY KEY,
    employee_id REFERENCES employees(employee_id) ON DELETE CASCADE,
    equip_name  VARCHAR2(20) NOT NULL);

CREATE SEQUENCE equipment_seq;
CREATE TRIGGER equipment_trig BEFORE INSERT ON equipment FOR EACH ROW
BEGIN
    :NEW.id := equipment_seq.NEXTVAL;
END;
/
```

The PL/SQL sequence and trigger assign a unique key to each new equipment record as it is inserted.

If you run these statements in a SQL editor, such as in NetBeans, omit the trailing slash ('/') in the `CREATE TRIGGER` statement. The slash is SQL*Plus's end-of-statement indicator and is not part of the statement that is run by the database.

Create some sample data:

```
-- Sample Data
INSERT INTO equipment (employee_id, equip_name) VALUES (100, 'pen');
INSERT INTO equipment (employee_id, equip_name) VALUES (100, 'telephone');
INSERT INTO equipment (employee_id, equip_name) VALUES (101, 'pen');
INSERT INTO equipment (employee_id, equip_name) VALUES (101, 'paper');
INSERT INTO equipment (employee_id, equip_name) VALUES (101, 'car');
INSERT INTO equipment (employee_id, equip_name) VALUES (102, 'pen');
INSERT INTO equipment (employee_id, equip_name) VALUES (102, 'paper');
INSERT INTO equipment (employee_id, equip_name) VALUES (102, 'telephone');
INSERT INTO equipment (employee_id, equip_name) VALUES (103, 'telephone');
INSERT INTO equipment (employee_id, equip_name) VALUES (103, 'computer');
INSERT INTO equipment (employee_id, equip_name) VALUES (121, 'computer');
INSERT INTO equipment (employee_id, equip_name) VALUES (180, 'pen');
INSERT INTO equipment (employee_id, equip_name) VALUES (180, 'paper');
INSERT INTO equipment (employee_id, equip_name) VALUES (180, 'cardboard box');
COMMIT;
```

In SQL*Plus create a procedure as `HR`:

```
CREATE OR REPLACE PROCEDURE get_equip(eid_p IN NUMBER, RC OUT SYS_REFCURSOR) AS
BEGIN
    OPEN rc FOR SELECT   equip_name
                FROM     equipment
                WHERE    employee_id = eid_p
                ORDER BY equip_name;
END;
/
```

In PHP this procedure can be called by running an anonymous PL/SQL block:

```
BEGIN get_equip(:id, :rc); END;
```

The `:id` bind variable is used similarly to binds shown before. It passes a value from a PHP variable into the database for the `WHERE` clause of `get_equip()`. The bind variable `:rc` is different and will hold the query results returned from `equip_name()` as explained in a few moments.

Calling the REF CURSOR in PHP

To display an employee's list of equipment create a new PHP file `ac_show_equip.php`:

```php
<?php

/**
 * ac_show_equip.php: Show an employee's equipment
 * @package ShowEquipment
 */

session_start();
require('ac_db.inc.php');
require('ac_equip.inc.php');

$sess = new \Equipment\Session;
$sess->getSession();
if (!isset($sess->username) || empty($sess->username)
```

```
        || !isset($_GET['empid'])) {
    header('Location: index.php');
    exit;
}
$empid = (int) $_GET['empid'];

$page = new \Equipment\Page;
$page->printHeader("AnyCo Corp. Show Equipment");
$page->printMenu($sess->username, $sess->isPrivilegedUser());
printcontent($sess, $empid);
$page->printFooter();

// Functions

?>
```

This is similar in structure to `ac_emp_list.php`. This time the verification test after `$sess->getSession()` also checks for an employee identifier. This value is passed in as a URL parameter from the `printrecords()` function in `ac_emp_list.php`:

```
...
<a href='ac_show_equip.php?empid=$eid'>Show</a>
...
```

The identifier value is accessed in `ac_show_equip.php` via PHP's `$_GET` superglobal array. If the array entry is not set then the assumption is that `ac_show_equip.php` was called incorrectly and the user is redirected to the login page, `index.php`.

The `$_GET['empid']` value is cast to an integer to minimize potential SQL injection issues. Although we will bind the value, it is better to consistently filter all user input. If `$_GET['empid']` contained alphabetic text for some reason, PHP's casting rules will result in the number `0` being stored in `$empid`. If the text had a numeric prefix then `$empid` would be that number, but at least the following text would have been discarded.

Before we get to the main content of the file, add a small helper function `getempname()` in the Functions section of `ac_show_equip.php`:

```
/**
 * Get an Employee Name
 *
 * @param Db $db
 * @param integer $empid
 * @return string An employee name
 */
function getempname($db, $empid) {
    $sql = "SELECT first_name || ' ' || last_name AS emp_name
        FROM employees
        WHERE employee_id = :id";
    $res = $db->execFetchAll($sql, "Get EName", array(array(":id", $empid, -1)));
    $empname = $res[0]['EMP_NAME'];
    return($empname);
}
```

This takes the employee identifier that the script was invoked for and looks up the matching employee name. An exercise for the reader is to handle the case when the query doesn't return any rows.

Now add the main `printcontent()` function to `ac_show_equip.php`:

```
/**
```

```
 * Print the main body of the page
 *
 * @param Session $sess
 * @param integer $empid Employee identifier
 */
function printcontent($sess, $empid) {
    echo "<div id='content'>\n";
    $db = new \Oracle\Db("Equipment", $sess->username);
    $empname = htmlspecialchars(getempname($db, $empid), ENT_NOQUOTES, 'UTF-8');
    echo "$empname has: ";

    $sql = "BEGIN get_equip(:id, :rc); END;";
    $res = $db->refcurExecFetchAll($sql, "Get Equipment List",
            "rc", array(array(":id", $empid, -1)));
    if (empty($res['EQUIP_NAME'])) {
        echo "no equipment";
    } else {
        echo "<table border='1'>\n";
        foreach ($res['EQUIP_NAME'] as $item) {
            $item = htmlspecialchars($item, ENT_NOQUOTES, 'UTF-8');
            echo "<tr><td>$item</td></tr>\n";
        }
        echo "</table>\n";
    }
    echo "</div>";  // content
}
```

This calls a new method `Db::refcurExecFetchAll()` which will return an array of records, printed in our traditional loop.

The REF CURSOR bind parameter `:rc` needs to be bound specially. Since the bind variable name could be arbitrarily chosen or located anywhere in the statement text, its name is passed separately into `refcurExecFetchAll()` and it is not included in the array of normal bind variables.

Now create the `refcurExecFetchAll()` method by editing `ac_db.inc.php` and adding this to the `Db` class:

```
/**
 * Run a call to a stored procedure that returns a REF CURSOR data
 * set in a bind variable.  The data set is fetched and returned.
 *
 * Call like Db::refcurexecfetchall("begin myproc(:rc, :p); end",
 *                            "Fetch data", ":rc", array(array(":p", $p, -1)))
 * The assumption that there is only one refcursor is an artificial
 * limitation of refcurexecfetchall()
 *
 * @param string $sql A SQL string calling a PL/SQL stored procedure
 * @param string $action Action text for End-to-End Application Tracing
 * @param string $rcname the name of the REF CURSOR bind variable
 * @param array  $otherbindvars Binds. Array (bv_name, php_variable, length)
 * @return array Returns an array of tuples
 */
public function refcurExecFetchAll($sql, $action, $rcname,
$otherbindvars = array()) {
    $this->stid = oci_parse($this->conn, $sql);
    $rc = oci_new_cursor($this->conn);
    oci_bind_by_name($this->stid, $rcname, $rc, -1, OCI_B_CURSOR);
    foreach ($otherbindvars as $bv) {
        // oci_bind_by_name(resource, bv_name, php_variable, length)
        oci_bind_by_name($this->stid, $bv[0], $bv[1], $bv[2]);
```

```
        }
        oci_set_action($this->conn, $action);
        oci_execute($this->stid);
        oci_execute($rc); // run the ref cursor as if it were a statement id
        oci_fetch_all($rc, $res);
        $this->stid = null;
        return($res);
    }
```

The REF CURSOR bind parameter in $rcname is bound to a cursor created with oci_new_cursor(), not to a normal PHP variable. The type OCI_B_CURSOR must specified.

After setting the tracing "action" text, the PL/SQL statement is run. In this example it calls get_equip() which opens and returns the cursor for the query. The REF CURSOR in $rc can now be treated like a PHP statement identifier as if it had been returned from an oci_parse() call. It is then fetched from. The query results are returned in $res to the function caller.

Save all files and run the application in a browser. Login as either Simon or Administrator. Click the **Show** link next to **Steven King**. The equipment he has is displayed:

7

Error handling

Error handling in web applications should occur at many levels, protecting against everything from invalid user input right through to database errors. To make the user experience smooth, PHP errors should never be displayed to the web user. They should be captured in mid-tier log files and the user should instead be given the chance to retry or do another task. In a production system the php.ini `display_errors` setting should be `Off`.

This chapter contains the following topics:

- Database Errors
- Displaying a Custom Error Message

Database Errors

At the database level, it is recommended to check all PHP OCI8 errors.

In `ac_db.inc.php`, currently the only error checking occurs at connection time in `__construct()`:

```
...
if (!$this->conn) {
    $m = oci_error();
    throw new \Exception('Cannot connect to database: ' . $m['message']);
}
...
```

The `oci_error()` function returns an associative array, one element of which includes the text of the Oracle error message.

Left as an extra exercise for the reader is to improve the error handling in the `Db` class. The rest of this tutorial is not dependent on any changes in this regard. Evaluate each PHP OCI8 call and decide where to check return values. Call `oci_error()` to get the text of the message. For a connection error, do not pass an argument to `oci_error()`, as shown above. Unlike connection errors where `oci_error()` takes no argument, to check errors from `oci_parse()` pass the connection resource to `oci_error()`:

```
$stid = oci_parse($conn, $sql);
if (!$stid) {
    $m = oci_error($conn)
    ...
}
```

For `oci_execute()` errors pass the statement handle:

```
$r = oci_execute($stid);
if (!$r) {
```

```
        $m = oci_error($stid)
        ...
}
```

Displaying a Custom Error Message

Simulate an error in `ac_show_equip.php` by editing `getempname()` and throwing an exception in `printcontent()`. PHP will give a run time error when it reaches that call:

```
function printcontent($sess, $empid) {
    echo "<div id='content'>\n";
    $db = new \Oracle\Db("Equipment", $sess->username);
    $empname = htmlspecialchars(getempname($db, $empid), ENT_NOQUOTES, 'UTF-8');
    echo "$empname has: ";

    throw new Exception;

    $sql = "BEGIN get_equip(:id, :rc); END;";

    ...
```

Run the application in a browser and click the **Show** link for **Steven King**. Because we set **display_errors** to **On** for development purposes we see the error displayed in the content area:
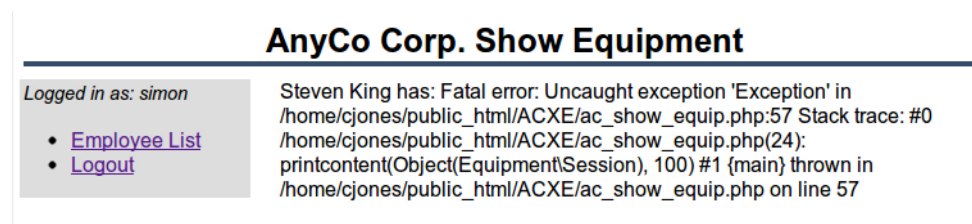
The error is printed after the initial part of the page showing the user name is printed. In a production site with `display_errors` set to `Off`, the user would see just this partial section content being displayed, which is not ideal. To prevent this, PHP's output buffering can be used.

Edit `ac_show_equip.php` and modify where `printcontent()` is called. Wrap the call in a PHP try-catch block, changing it to:

```
...
$page->printMenu($sess->username, $sess->isPrivilegedUser());
ob_start();
try {
    printcontent($sess, $empid);
} catch (Exception $e) {
    ob_end_clean();
    echo "<div id='content'>\n";
    echo "Sorry, an error occurred";
    echo "</div>";
}
ob_end_flush();
$page->printFooter();
...
```

The `ob_start()` function captures all subsequently generated output in a buffer. Other PHP `ob_*` functions allow that buffer to be discarded or flushed to the browser. In the

code above, the `ob_end_clean()` call in the exception handler will discard the "Steven King has:" message so a custom error message can be printed.

Run the application again to see the following error:

AnyCo Corp. Show Equipment

Logged in as: simon

Sorry, an error occurred

- Employee List
- Logout
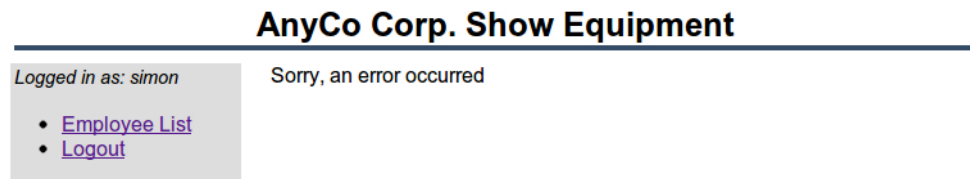
If you don't like using object-oriented code, an alternative to throwing and catching an exception would be to return a boolean from `printcontent()` and handle the error manually. If you want to stop execution you can use PHP's `trigger_error()`.

Edit the `printcontent()` function in `ac_show_equip.php` and change the temporary line:

```
throw new Exception;
```

to

```
trigger_error('Whoops!', E_USER_ERROR);
```

To catch and handle PHP errors like `E_USER_ERROR`, you can use PHP's `set_error_handler()` function which allows an error handler function to be registered.

At the top of `ac_show_equip.php` add a call to `set_error_handler()`:

```
...
session_start();
set_error_handler("ac_error_handler");

require('ac_db.inc.php');
require('ac_equip.inc.php');
...
```

Also add the called function:

```
/**
 * Error Handler
 *
 * @param integer $errno Error level raised
 * @param string $errstr Error text
 * @param string $errfile File name that the error occurred in
 * @param integer $errline File line where the error occurred
 */
function ac_error_handler($errno, $errstr, $errfile, $errline) {
    error_log(sprintf("PHP AnyCo Corp.: %d:  %s in %s on line %d",
            $errno, $errstr, $errfile, $errline));
    header('Location: ac_error.html');
    exit;
}
```

This records the message in the Apache log file and redirects to an error page. Create that error page in a new HTML file `ac_error.html`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
        "http://www.w3.org/TR/html4/loose.dtd">
<html>
```

```
<!-- ac_error.html: a catch-all error page -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title></title>
</head>

<body bgcolor="#ffffff">
<h1>AnyCo Corp. Error Page</h1>
<p>We're sorry an error occurred.<p>
<p><a href="index.php">Login</a></p>

</body>
</html>
```

Run the application and login. Click **Show** to see an employee's equipment. The error page is shown:

Locate the Apache error log on your system, for example in `/var/log/httpd/error_log` on Oracle Linux. The log will contain message generated by PHP:

```
[Wed Apr 27 13:06:09 2011] [error] [client 127.0.0.1] PHP AnyCo Corp.: 256:
Whoops! in /home/chris/public_html/ACXE/ac_show_equip.php on line 71, referer:
http://localhost/~chris/ACXE/ac_emp_list.php
```

Remove or comment out the temporary `trigger_error()` call in `printcontent()` before continuing with the next chapter.

```
...
// trigger_error('Whoops!', E_USER_ERROR);
...
```

8

Query Performance and Prefetching

This chapter contains the following topics:

- Prefetching Overview
- The Employee Report Page
- Running the Equipment Report
- REF CURSOR Prefetching

Prefetching Overview

This section shows how the performance of fetching query rows can be tuned in PHP.

Prefetching is the way that PHP OCI8 reduces network roundtrips to the database when fetching query results. By retrieving batches of rows, there is better database and network efficiency.

Prefetching is enabled by default in PHP OCI8. When the first row is initially retrieved from the database, up to the configured limit (100 by default) extra rows up will be returned and stored in an internal buffer local to the PHP process. Any of the PHP OCI8 `oci_fetch_*` functions called in a script will internally use data from that buffer until it is exhausted, at which point another round trip to the database occurs and a further batch of rows is returned. The way the `oci_fetch_*` functions return data to the caller doesn't change regardless of the prefetch value in effect.

The default prefetch value can be set with `oci8.default_prefetch` in the `php.ini` configuration file, or it can be set at run time with `oci_set_prefetch()`.

So far the AnyCo application has used `oci_fetch_all()`. For a change, this chapter will show the other commonly used function, `oci_fetch_array()`. When this is called in a loop, it iterates through all rows in the query result set. For bigger data sets, fetching one row at a time prevents a large amount of memory being needed to hold the whole result set.

The action and benefits of prefetching would not be changed if `oci_fetch_all()` was used. Prefetching is handled in the Oracle client libraries at a layer below PHP.

The Employee Report Page

Create a new PHP file `ac_report.php` that generates a report of all employees and the equipment issued to them. The file initially looks like:

```
<?php

/**
```

```
 * ac_report.php: Full report of all employees and their equipment
 * @package Report
 */

session_start();
require('ac_db.inc.php');
require('ac_equip.inc.php');

$sess = new \Equipment\Session;
$sess->getSession();
if (!isset($sess->username) || empty($sess->username)
        || !$sess->isPrivilegedUser()) {
    header('Location: index.php');
    exit;
}

$page = new \Equipment\Page;
$page->printHeader("AnyCo Corp. Equipment Report");
$page->printMenu($sess->username, $sess->isPrivilegedUser());
printcontent($sess);
$page->printFooter();

// Functions

?>
```

In the `Functions` section add the `printcontent()` function:

```
/**
 * Print the main body of the page
 *
 * @param Session $sess
 */
function printcontent($sess) {
    echo "<div id='content'>";
    $db = new \Oracle\Db("Equipment", $sess->username);

    $sql = "select first_name || ' ' || last_name as emp_name, equip_name
        from employees left outer join equipment
        on employees.employee_id = equipment.employee_id
        order by emp_name, equip_name";

    // Change the prefetch value to compare performance.
    // Zero will be slowest. The system default is 100
    $db->setPrefetch(200);

    $time = microtime(true);
    $db->execute($sql, "Equipment Report");
    echo "<table>";
    while (($row = $db->fetchRow()) != false) {
        $empname = htmlspecialchars($row['EMP_NAME'], ENT_NOQUOTES, 'UTF-8');
        $equipname = htmlspecialchars($row['EQUIP_NAME'], ENT_NOQUOTES, 'UTF-8');
        echo "<tr><td>$empname</td><td>$equipname</td></tr>";
    }
    echo "</table>";
    $time = microtime(true) - $time;
    echo "<p>Report generated in " . round($time, 3) . " seconds\n";
    echo "</div>";  // content
}
```

The structure is basically similar to the layout shown in previous chapters.

The `$db->setPrefetch()` call is used to set the prefetch value. The `microtime()` calls are used to show how long the report took to generate.

A new `Db::fetchRow()` method is used to get one row at a time. It is called in a loop after the query has been run.

Edit `ac_db.inc.php` and add the `setPrefetch()` and `fetchRow()` methods to the `Db` class:

```
/**
 * Set the query prefetch row count to tune performance by reducing the
 * number of round trips to the database.  Zero means there will be no
 * prefetching and will be slowest.  A negative value will use the php.ini
 * default value.  Some queries such as those using LOBS will not have
 * rows prefetched.
 *
 * @param integer $pf The number of rows that queries should prefetch.
 */
public function setPrefetch($pf) {
    $this->prefetch = $pf;
}

/**
 * Fetch a row of data.  Call this in a loop after calling Db::execute()
 *
 * @return array An array of data for one row of the query
 */
public function fetchRow() {
    $row = oci_fetch_array($this->stid, OCI_ASSOC + OCI_RETURN_NULLS);
    return($row);
}
```

The `OCI_ASSOC` flag tells PHP to return the results in an associative array, using the column names as the array keys. The `OCI_RETURN_NULLS` flag tells PHP to return an array entry for null data values. The value will be an empty string. This ensures that the array for each row has the same number of entries.

Running the Equipment Report

Save all the files and run the Application as Administrator. From the left hand navigation menu select **Equipment Report**. It shows all employees and the equipment they have been issued.

AnyCo Corp. Equipment Report

Logged in as: admin

- Employee List
- Equipment Report
- Equipment Graph
- Upload Logo
- Logout

Adam Fripp	computer
Alana Walsh	
Alberto Errazuriz	
Alexander Hunold	computer
Alexander Hunold	telephone
Alexander Khoo	
Alexis Bull	
Allan McEwen	
Alyssa Hutton	
Amit Banda	
Anthony Cabrio	

At the bottom is the amount of time taken to generate the query output. For this amount of data and because PHP and the database are not separated by a network, the time will be small:

William Gietz	
William Smith	
Winston Taylor	cardboard box
Winston Taylor	paper
Winston Taylor	pen

Report generated in 0.024 seconds

To show the effect of turning off prefetching, edit `ac_report.php` and change the prefetch setting to `0`:

```
$db->setPrefetch(0);
```

This means that each row of data that PHP OCI8 gets from the Oracle Client libraries initiates a roundtrip request to the database server. No extra rows are prefetched.

Re-run the report. The elapsed time should be longer.

Winston Taylor	cardboard box
Winston Taylor	paper
Winston Taylor	pen

Report generated in 0.043 seconds

For a small system like this there might be some test variability and the values may be too small to be reliable. Re-run several times or change the query to return more rows if this is the case.

REF CURSOR Prefetching

Prefetching can also be used when fetching records from `REF CURSORS`. To make the `REF CURSOR` prefetch value changeable in the `Db` class, edit `ac_db.inc.php` and add the following lines before the `REF CURSOR` execution in `Db::refcurExecFetchAll()`:

```
if ($this->prefetch >= 0) {
    oci_set_prefetch($rc, $this->prefetch);  // set on the REFCURSOR
}
```

This prefetch size is set on the `REF CURSOR`, not the top level statement. The function will look as follows:

```
public function refcurExecFetchAll($sql, $action, $rcname,
$otherbindvars = array()) {
    $this->stid = oci_parse($this->conn, $sql);
    $rc = oci_new_cursor($this->conn);
    oci_bind_by_name($this->stid, $rcname, $rc, -1, OCI_B_CURSOR);
    foreach ($otherbindvars as $bv) {
        // oci_bind_by_name(resource, bv_name, php_variable, length)
        oci_bind_by_name($this->stid, $bv[0], $bv[1], $bv[2]);
    }
    oci_set_action($this->conn, $action);
    oci_execute($this->stid);
    if ($this->prefetch >= 0) {
```

```
        oci_set_prefetch($rc, $this->prefetch);  // set on the REFCURSOR
    }
    oci_execute($rc); // run the ref cursor as if it were a statement id
    oci_fetch_all($rc, $res);
    return($res);
}
```

With your own applications, testing will show the optimal prefetch size for your queries. There is no benefit in using too large a value. Conversely, because Oracle dynamically allocates space, there is little to be gained by making the value too small.

It is unlikely that you want to turn pre-fetching completely off. The only case would be in PHP code that gets a REF CURSOR, fetches some data from it, and then passes the cursor back to a PL/SQL procedure which fetches the remaining data. If prefetching occurred when PHP fetches records from the REF CURSOR, but those prefetched rows were not returned to the script via an oci_fetch_* call, those rows would be "lost" and would not be available to the second PL/SQL procedure.

> **Note:**   PHP must be linked with Oracle Database 11gR2 libraries for prefetching from REF CURSOR to work. When using earlier versions each requested REF CURSOR row required a roundtrip to the database, reducing performance of the system.

9

Inserting Data

To enable pieces of equipment to be assigned to employees the AnyCo application will have an HTML form. The form allows administrators to assign a piece of equipment to a specific employee.

This chapter contains the following topics:

- Building the Insert Form
- Running the Single Insert Form
- CSRF example with ac_add_one.php

Building the Insert Form

Create a new PHP file `ac_add_one.php`. Initially the file looks like:

```php
<?php

/**
 * ac_add_one.php: Add one piece of equipment to an employee
 * @package Application
 */
session_start();
require('ac_db.inc.php');
require('ac_equip.inc.php');

$sess = new \Equipment\Session;
$sess->getSession();
if (!isset($sess->username) || empty($sess->username)
        || !$sess->isPrivilegedUser()
        || (!isset($_GET['empid']) && !isset($_POST['empid']))) {
    header('Location: index.php');
    exit;
}
$empid = (int) (isset($_GET['empid']) ? $_GET['empid'] : $_POST['empid']);

$page = new \Equipment\Page;
$page->printHeader("AnyCo Corp. Add Equipment");
$page->printMenu($sess->username, $sess->isPrivilegedUser());
printcontent($sess, $empid);
$page->printFooter();

// Functions

?>
```

The process flow of operation will be similar to `index.php`. The first time `ac_add_one.php` is run an HTML input form will be displayed. When the user submits the form, `ac_add_one.php` is invoked again, which will insert the data into the database.

The privileges required by this function include checks that an employee id is set in either the `$_GET['empid']` or `$_POST['empid']` superglobals. When `ac_add_one.php` is first called (see `printrecords()` in `ac_emp_list.php`), the employee id is passed as a URL parameter and will be in the `$_GET` superglobal. When the form (that will shortly be shown) in `ac_add_one.php` is submitted, the employee identifier will be in `$_POST`.

Add the `printcontent()` function to `ac_add_one.php`:

```
/**
 * Print the main body of the page
 *
 * @param Session $sess
 * @param integer $empid Employee identifier
 */
function printcontent($sess, $empid) {
    echo "<div id='content'>\n";
    $db = new \Oracle\Db("Equipment", $sess->username);
    if (!isset($_POST['equip']) || empty($_POST['equip'])) {
        printform($sess, $db, $empid);
    } else {
        /* if (!isset($_POST['csrftoken'])
                || $_POST['csrftoken'] != $sess->csrftoken) {
            // the CSRF token they submitted doesn't match the one we sent
            header('Location: index.php');
            exit;
        } */
        $equip = getcleanequip();
        if (empty($equip)) {
            printform($sess, $db, $empid);
        } else {
            doinsert($db, $equip, $empid);
            echo "<p>Added new equipment</p>";
            echo '<a href="ac_show_equip.php?empid='
                    . $empid . '">Show Equipment</a>' . "\n";
        }
    }
    echo "</div>";  // content
}
```

The `printcontent()` function contains the logic to decide if the HTML form should be printed or the user-entered data should be inserted. The commented-out CSRF token code will be discussed below.

Also in `ac_add_one.php` add the `printform()` function:

```
/**
 * Print the HTML form for entering new equipment
 *
 * @param Session $sess
 * @param Db $db
 * @param integer $empid Employee identifier
 */
function printform($sess, $db, $empid) {
    $empname = htmlspecialchars(getempname($db, $empid), ENT_NOQUOTES, 'UTF-8');
    $empid = (int) $empid;
    $sess->setCsrfToken();
    echo <<<EOF
```

```
Add equipment for $empname
<form method='post' action='${_SERVER["PHP_SELF"]}'>
<div>
    Equipment name <input type="text" name="equip"><br>
    <input type="hidden" name="empid" value="$empid">
    <input type="hidden" name="csrftoken" value="$sess->csrftoken">
    <input type="submit" value="Submit">
</div>
</form>
EOF;
}
```

> **Note:** The `EOF;` token must be at the start of a line and not have trailing white space.

This simple form prompts the user for a value. The CSRF token will be described later.

Add the `getcleanequip()` function to ac_add_one.php:

```
/**
 * Perform validation and data cleaning so empty strings are not inserted
 *
 * @return string The new data to enter
 */
function getcleanequip() {
    if (!isset($_POST['equip'])) {
        return null;
    } else {
        $equip = $_POST['equip'];
        return(trim($equip));
    }
}
```

This implementation strips any leading or trailing white space from the entered data.

The general mantra for basic web application security is to filter input and escape output. The `getcleanequip()` function filters input. The data could be sanitized in other ways here. You may decide that you don't want HTML tags to be accepted. You can strip such tags by using one of PHP's input filters. For example, if you wanted, you could change:

```
        $equip = $_POST['equip'];
```

to

```
        $equip = filter_input(INPUT_POST, 'equip', FILTER_SANITIZE_STRING);
```

This would remove HTML tags, leaving other text in place.

In ac_add_one.php, valid data is inserted by doinsert(). Add the code for this function to the file:

```
/**
 * Insert a piece of equipment for an employee
 *
 * @param Db $db
 * @param string $equip Name of equipment to insert
 * @param string $empid Employee identifier
 */
function doinsert($db, $equip, $empid) {
```

```
        $sql = "INSERT INTO equipment (employee_id, equip_name) VALUES (:ei, :nm)";
        $db->execute($sql, "Insert Equipment", array(array("ei", $empid, -1),
                                                array("nm", $equip, -1)));
}
```

This uses the existing `Db::execute()` method in `ac_db.inc.php` with familiar bind variable syntax. Note that the `Db` class automatically commits each time `oci_execute()` is called as discussed earlier in the section Running SQL with the Db Class on page 3-6.

Finally, to complete `ac_add_one.php`, add the helper function `getempname()`:

```
/**
 * Get an Employee Name
 *
 * @param Db $db
 * @param integer $empid
 * @return string An employee name
 */
function getempname($db, $empid) {
    $sql = "SELECT first_name || ' ' || last_name AS emp_name
        FROM employees
        WHERE employee_id = :id";
    $res = $db->execFetchAll($sql, "Get EName", array(array("id", $empid, -1)));
    $empname = $res[0]['EMP_NAME'];
    return($empname);
}
```

This is identical to the function of the same name in `ac_show_equip.php`.

Similar functionality to the `ac_show_equip.php` form could be used to delete or update records, remembering the limitations of a stateless web architecture means that rows can't be locked in one HTML page and changed in another.

Running the Single Insert Form

Run the AnyCo application and log in as `Administrator`. Click the **Add One** link next to Steven King. The equipment input form is displayed:

Enter a new piece of equipment, `paper`, and click **Submit**. The new data is inserted. The updated list can be seen by clicking **Show** link next to Steven King.

CSRF example with ac_add_one.php

The form is currently prone to CSRF attacks, where another site can take advantage of you being logged in and cause you to submit data or do some other privileged operation.

To show this, create a new HTML page called `hack.html`:

```
<html>
<!-- hack.html: Show issues with CSRF -->
<body>
<h1>Make Millions!</h1>
<form method='post' action='http://localhost/ac_add_one.php'>
<div>
    Do you dream of being rich?<br>
    <input type="hidden" name="equip" value="fish">
    <input type="hidden" name="empid" value="100">
    <input type="submit" value="Win">
</div>
</form>

</body>
</html>
```

Change the HTML form action URL to match your system.

Run the AnyCo application in a browser and login as `Administrator`. In a new browser tab or window, open the following file:

`http://localhost/hack.html`

Ostensibly to the person looking at the page it has nothing to do with the AnyCo application.

Click the **Win** button. This calls the AnyCo application and causes the bogus equipment name `fish` to be inserted into the equipment list of employee `100` (which is Steven King). The inserted value can be seen on the subsequent **Show Equipment** page:

AnyCo Corp. Show Equipment

Logged in as: admin

- Employee List
- Equipment Report
- Equipment Graph
- Upload Logo
- Logout

Steven King has:

fish
paper
pen
telephone
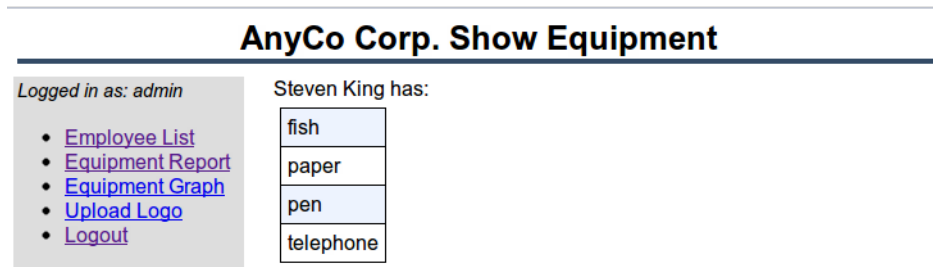
Now edit `ac_add_one.php` and enable CSRF protection by removing the comments for the check in `printcontent()`:

```
...
} else {
    if (!isset($_POST['csrftoken'])
            || $_POST['csrftoken'] != $sess->csrftoken) {
        // the CSRF token they submitted doesn't match the one we sent
        header('Location: index.php');
        exit;
    }
    $equip = getcleanequip();
...
```

The form in `ac_add_one.php` includes a generated Cross-Site Request Forgery token as a hidden field. The value is also stored in the user session. The CSRF check in `printcontent()` will verify that the token in the submitted form matches PHP's stored session value.

Save the file and run the AnyCo application again, logging in as `Administrator`. In a new browser tab or window, open the following file:

`http://localhost/hack.html`

Now click **Win**.

This time the CSRF protection in `printcontent()` doesn't find a CSRF token in the submitted form and redirects to the login page, index.php, which logs out. Log back in again to the AnyCo application and check that Steven King's equipment list is unchanged, with no second entry for `fish`. For hack.html to be successful it would have to know the value of the `csrftoken` field that gets stored in the PHP session when the `ac_add_one.php` generates the real entry form.

CSRF protection is just one of many kinds of security restrictions that web applications should enforce. You should do a thorough security evaluation of any code you deploy on the web.

Many of the popular PHP frameworks provide assistance to reduce the amount of effort required in producing a secure application. For example they may provide a more secure implementation of CSRF token generation than the one in the AnyCo Session class.

10

Inserting Multiple Data Values

PHP OCI8 can insert arrays of characters or integers in one call. This reduces network traffic and database system overhead when inserting multiple values into a table.

This chapter contains the following topics:

- Creating the Multiple Insert Form
- Running the Multiple Insert Form

Creating the Multiple Insert Form

The example in this chapter shows a form allowing three data values to be inserted in one operation.

The array insert is done using a PL/SQL bulk FORALL command. Login to SQL*Plus as HR and create a PL/SQL package:

```
CREATE OR REPLACE PACKAGE equip_pkg AS
    TYPE arrtype IS TABLE OF VARCHAR2(20) INDEX BY PLS_INTEGER;
    PROCEDURE insert_equip(eid_p IN NUMBER, eqa_p IN arrtype);
END equip_pkg;
/

CREATE OR REPLACE PACKAGE BODY equip_pkg AS
    PROCEDURE insert_equip(eid_p IN NUMBER, eqa_p IN arrtype) IS
    BEGIN
        FORALL i IN INDICES OF eqa_p
            INSERT INTO equipment (employee_id, equip_name)
                                 VALUES (eid_p, eqa_p(i));
    END insert_equip;
END equip_pkg;
/
```

The insert_equip() procedure accepts an array of equipment names and inserts them in to the EQUIPMENT table.

Create a new PHP file ac_add_multi.php and copy the contents of ac_add_one.php to it. Carefully make the following changes to convert it to handle an array of values.

In the HTML form in ac_add_multi.php, change the one input field from:

```
<div>
    Equipment name <input type="text" name="equip"><br>
    <input type="hidden" name="empid" value="$empid">
...
```

to three input fields:

```
...
<div>
    Equipment name <input type="text" name="equip[]"><br>
    Equipment name <input type="text" name="equip[]"><br>
    Equipment name <input type="text" name="equip[]"><br>
    <input type="hidden" name="empid" value="$empid">
...
```

Note the `[]` tokens to return an array, which were not needed in `ac_add_one.php`.

Replace the `getcleanequip()` function in `ac_add_multi.php` so it handles the array of returned form values:

```
/**
 * Perform validation and data cleaning so empty strings are not inserted
 *
 * @return array The array of new data to enter
 */
function getcleanequip() {
    if (!isset($_POST['equip'])) {
        return array();
    } else {
        $equiparr = array();
        foreach ($_POST['equip'] as $v) {      // Strip out unset values
            $v = trim($v);
            if (!empty($v))
                $equiparr[] = $v;
        }
        return($equiparr);
    }
}
```

This loops along each of the array entries and only returns non empty strings.

Finally, replace `doinsert()` in `ac_add_multi.php` with:

```
/**
 * Insert an array of equipment values for an employee
 *
 * @param Db $db
 * @param array $equiparr array of string values to be inserted
 * @param string $empid Employee identifier
 */
function doinsert($db, $equiparr, $empid) {
    $arraybinds = array(array("eqa", $equiparr, SQLT_CHR));
    $otherbinds = array(array("eid", $empid, -1));
    $sql = "BEGIN equip_pkg.insert_equip(:eid, :eqa); END;";
    $db->arrayInsert($sql, "Insert Equipment List", $arraybinds, $otherbinds);
}
```

This uses a new `arrayInsert()` method in the `Db` class to call the PL/SQL `insert_equip()` procedure. The data value arrays needs to be bound differently from normal scalar PHP OCI8 binds, so the bind parameters to `arrayInsert()` are separated into two kinds.

Edit ac_db.inc.php and add the new method:

```
    /**
     * Insert an array of values by calling a PL/SQL procedure
     *
     * Call like Db::arrayinsert("begin myproc(:arn, :p); end",
     *                           "Insert stuff",
```

```
 *                                       array(array(":arn", $dataarray, SQLT_CHR)),
 *                                       array(array(":p", $p, -1)))
 *
 * @param string $sql PL/SQL anonymous block
 * @param string $action Action text for End-to-End Application Tracing
 * @param array $arraybindvars Bind variables. An array of tuples
 * @param array $otherbindvars  Bind variables. An array of tuples
 */
public function arrayInsert($sql, $action, $arraybindvars,
$otherbindvars = array()) {
    $this->stid = oci_parse($this->conn, $sql);
    foreach ($arraybindvars as $a) {
        // oci_bind_array_by_name(resource, bv_name,
        // php_array, php_array_length, max_item_length, datatype)
        oci_bind_array_by_name($this->stid, $a[0], $a[1],
        count($a[1]), -1, $a[2]);
    }
    foreach ($otherbindvars as $bv) {
        // oci_bind_by_name(resource, bv_name, php_variable, length)
        oci_bind_by_name($this->stid, $bv[0], $bv[1], $bv[2]);
    }
    oci_set_action($this->conn, $action);
    oci_execute($this->stid);                // will auto commit
    $this->stid = null;
}
```

Binding in `Db::arrayInsert()` is similar to the example previously shown in this manual. The `oci_bind_array_by_name()` function takes slightly different arguments, since the number of elements in data array must now be passed in. In the AnyCo application `oci_bind_array_by_name` is being used only for inserting data from PHP so the maximum data length parameter can be passed as `-1`. This tells PHP to use the actual value lengths. The single `oci_execute()` call inserts all the data items into the database.

Running the Multiple Insert Form

Save the files and run the AnyCo application in a browser. Log in as `Administrator` and click the **Add Multiple** link for Steven King.

Add some data items such as `Computer`, `Monitor`, and `Keyboard`.

Click **Submit** and then click **Show** next to Steven King to check that the data items are inserted.

Array binding also works for fetching data. PL/SQL procedures using the efficient `BULK COLLECT` syntax can return data to PHP in one OCI8 `oci_execute()` call. For retrieving data from Oracle the `oci_bind_array_by_name()` call would need to know how many items and what the maximum data size is so PHP can allocate the memory correctly.

11

Using JSON and generating a JPEG image

This chapter shows how the JSON serialization format can be used for transferring data from a simple web service. The web service is called by a client that creates an image using PHP's GD extension.

This chapter contains the following topics:

- Creating a Simple Web Service Returning JSON
- Creating a JPEG image

Creating a Simple Web Service Returning JSON

Create a new PHP file `ac_get_json.php` containing:

```php
<?php

/**
 * ac_get_json.php: Service returning equipment counts in JSON
 * @package WebService
 */

require('ac_db.inc.php');

if (!isset($_POST['username'])) {
    header('Location: index.php');
    exit;
}

$db = new \Oracle\Db("Equipment", $_POST['username']);

$sql = "select equip_name, count(equip_name) as cn
        from equipment
        group by equip_name";
$res = $db->execFetchAll($sql, "Get Equipment Counts");

$mydata = array();
foreach ($res as $row) {
    $mydata[$row['EQUIP_NAME']] = (int) $row['CN'];
}

echo json_encode($mydata);

?>
```

Note there is no authentication in this web service. It is "external" to the AnyCo application. All it requires is a `username` entry in the `POST` data.

The file queries the AnyCo Corp. equipment allocation and uses PHP's `json_encode()` to return the statistics in JSON format. The output returned by the web service is something like this, depending on which data you currently have in the `EQUIPMENT` table:

```
{"cardboard box":1,"pen":4,"computer":2,"telephone":3,"paper":3,"car":1}
```

Creating a JPEG image

Create a new PHP file `ac_graph_img.php` to call the web service and create a graph. The file initially contains:

```php
<?php

/**
 * ac_graph_img.php: Create a JPEG image of the equipment allocation statistics
 *
 * Don't have any text or white space before the "<?php" tag because it will
 * be incorporated into the image stream and corrupt the picture.
 *
 * @package Graph
 */

define('WEB_SERVICE_URL', "http://localhost/ac_get_json.php");

session_start();
require('ac_equip.inc.php');

$sess = new \Equipment\Session;
$sess->getSession();
if (!isset($sess->username) || empty($sess->username)
    || !$sess->isPrivilegedUser()) {
    header('Location: index.php');
    exit;
}
$data = callservice($sess);
do_graph("Equipment Count", 600, $data);

// Functions

?>
```

Change the web service URL to match your system.

To this file add the `callservice()` function:

```php
/**
 * Call the service and return its results
 *
 * @param Session $sess
 * @return array Equipment name/count array
 */
function callservice($sess) {
    // Call the web "service" to get the Equipment statistics
    // Change the URL to match your system configuration
    $calldata = array('username' => $sess->username);
    $options = array(
        'http' => array(
            'method'  => 'POST',
            'header'  => 'Content-type: application/x-www-form-urlencoded',
```

```
                          'content' => http_build_query($calldata)
                   )
              );
              $ctx = stream_context_create($options);
              $result = file_get_contents(WEB_SERVICE_URL, false, $ctx);
              if (!$result) {
                   $data = null;
              } else {
                   $data = json_decode($result, true);

                   // Sort an array by keys using an anonymous function
                   uksort($data, function($a, $b) {
                        if ($a == $b)
                             return 0;
                        else
                             return ($a < $b) ? -1 : 1;
                        });
                   }
              return($data);
}
```

This uses the PHP streams functionality to request the URL and get the statistics. The stream context includes the username as a post variable, which is required by the service.

The data is decoded from the JSON format and the array is sorted by name order. The second argument to PHP's uksort() function is an anonymous function that does the data comparison.

Edit ac_graph_img.php and add the function to create the image:

```
/**
 * Draw a bar graph, with bars projecting horizontally
 *
 * @param string $title The Graph's title
 * @param type $width Desired image width in pixels
 * @param array $items Array of (caption, value) tuples
 */
function do_graph($title, $width, $items) {
    $border = 50;              // border space around bars
    $caption_gap = 4;          // space between bar and its caption
    $bar_width = 20;           // width of each bar
    $bar_gap = 40;             // space between each bar
    $title_font_id = 5;        // font id for the main title
    $bar_caption_font_id = 5;  // font id for each bar's title

    // Image height depends on the number of items
    $height = (2 * $border) + (count($items) * $bar_width) +
        ((count($items) - 1) * $bar_gap);

    // Find the horizontal distance unit for one item
    $unit = ($width - (2 * $border)) / max($items);

    // Create the image and add the title
    $im = ImageCreate($width, $height);
    if (!$im) {
        trigger_error("Cannot create image<br>\n", E_USER_ERROR);
    }
    $background_col = ImageColorAllocate($im, 255, 255, 255); // white
    $bar_col = ImageColorAllocate($im, 0, 64, 128);          // blue
    $letter_col = ImageColorAllocate($im, 0, 0, 0);          // black
```

```
        ImageFilledRectangle($im, 0, 0, $width, $height, $background_col);
        ImageString($im, $title_font_id, $border, 4, $title, $letter_col);

        // Draw each bar and add a caption
        $start_y = $border;
        foreach ($items as $caption => $value) {
            $end_x = $border + ($value * $unit);
            $end_y = $start_y + $bar_width;
            ImageFilledRectangle($im, $border, $start_y, $end_x, $end_y, $bar_col);
            ImageString($im, $bar_caption_font_id, $border,
                $start_y + $bar_width + $caption_gap, $caption, $letter_col);
            $start_y = $start_y + ($bar_width + $bar_gap);
        }

        // Output the complete image.
        // Any text, error message or even white space that appears before this
        // (including any white space before the "<?php" tag) will corrupt the
        // image data.  Comment out the "header" line to debug any issues.
        header("Content-type: image/jpg");
        ImageJpeg($im);
        ImageDestroy($im);
}
```

This function uses PHP's GD extension to create the graph. The default GD fonts are a bit clunky but new ones can be added. The output is a JPEG stream so the PHP file can be called anywhere in a web page's HTML code where you would otherwise include an image file.

In the AnyCo application, the image can be integrated by creating a new file ac_graph_page.php:

```
<?php

/**
 * ac_graph_page.php: Display a page containing the equipment graph
 * @package Graph
 */

session_start();
require('ac_equip.inc.php');

$sess = new \Equipment\Session;
$sess->getSession();
if (!isset($sess->username) || empty($sess->username)
        || !$sess->isPrivilegedUser()) {
    header('Location: index.php');
    exit;
}

$page = new \Equipment\Page;
$page->printHeader("AnyCo Corp. Equipment Graph");
$page->printMenu($sess->username, $sess->isPrivilegedUser());

echo <<<EOF
<div id='content'>
<img src='ac_graph_img.php' alt='Graph of office equipment'>
</div>
EOF;

$page->printFooter();
```

```
?>
```

> **Note:** The 'EOF;' token must be at the start of a line and not have trailing white space.

The image is included in a normal HTML `img` tag.

Load the AnyCo application in a browser and log in as `Administrator`. Click the **Equipment Graph** link in the left hand navigation menu. The graph is displayed.
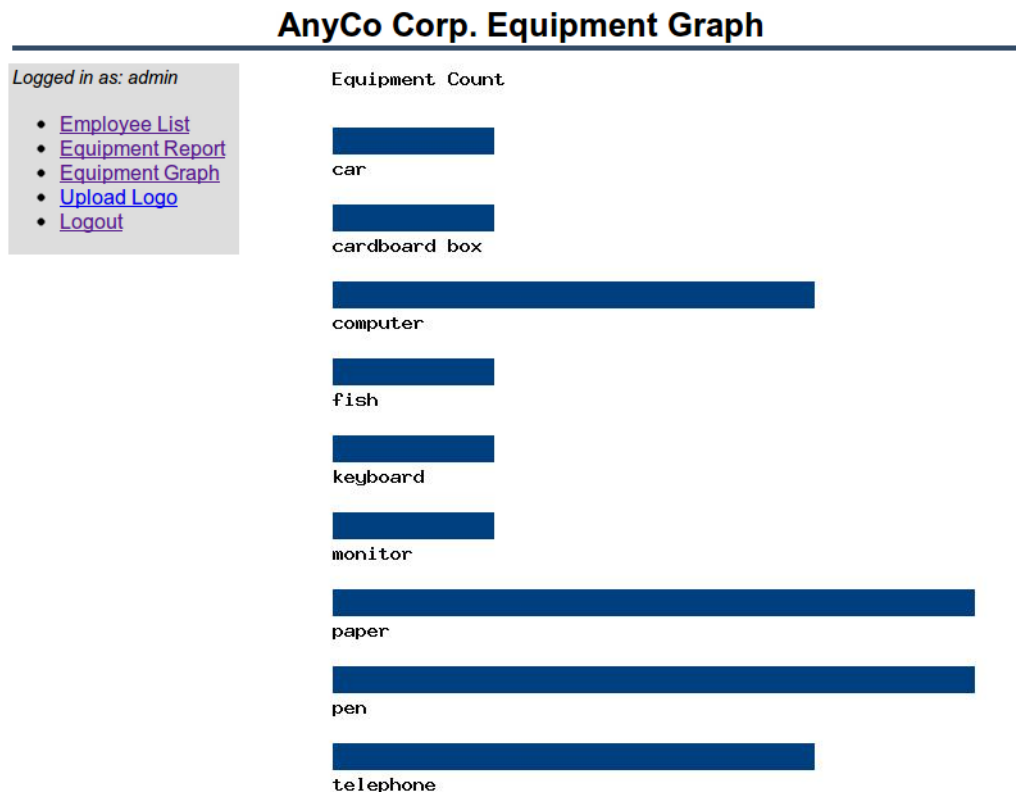
If the image doesn't display, it might be a problem in `ac_graph_img.php` due to text such as an error message or even because of white space before the `<?php` tag. This text will be included in the image stream and make the picture invalid. To help debug this kind of problem you could comment out the `$session` checks and also the `header()` call in `ac_graph_img.php`. Then to show the raw data of the image stream load the following link in a browser:

```
http://localhost/ac_graph_img.php
```

The JSON format is often used to efficiently transfer data between a browser and a PHP server. The `ac_get_json.php` web service could be used directly in many of the available JSON graphics libraries.

12

Uploading and Displaying BLOBs

This chapter contains the following topics:

Creating a Table to Store the Logo

The PHP OCI8 extension easily allows LOB data to be manipulated. A BLOB will be used in the AnyCo application to store a company logo which will be displayed on each web page.

In SQL*Plus create a table PICTURES to store the logo:

```
CREATE TABLE pictures (id NUMBER, pic BLOB);

CREATE SEQUENCE pictures_seq;
CREATE TRIGGER pictures_trig BEFORE INSERT ON pictures FOR EACH ROW
BEGIN
    :NEW.id := pictures_seq.NEXTVAL;
END;
/
```

Uploading Images in PHP OCI8

Create a new PHP file ac_logo_upload.php. The initial contents are:

```php
<?php

/**
 * ac_logo_upload.php: Upload a new company logo
 * @package Logo
 */

session_start();
require('ac_db.inc.php');
require('ac_equip.inc.php');

$sess = new \Equipment\Session;
$sess->getSession();
if (!isset($sess->username) || empty($sess->username)
```

```
        || !$sess->isPrivilegedUser()) {
    header('Location: index.php');
    exit;
}

$page = new \Equipment\Page;
$page->printHeader("AnyCo Corp. Upload Logo");
$page->printMenu($sess->username, $sess->isPrivilegedUser());
printcontent($sess);
$page->printFooter();

// Functions

?>
```

Add the `printcontent()` function:

```
/**
 * Print the main body of the page
 *
 * @param Session $sess
 */
function printcontent($sess) {
    echo "<div id='content'>";
    if (!isset($_FILES['lob_upload'])) {
        printform();
    } else {
        $blobdata = file_get_contents($_FILES['lob_upload']['tmp_name']);
        if (!$blobdata) {
            // N.b. this test could be enhanced to confirm the image is a JPEG
            printform();
        } else {
            $db = new \Oracle\Db("Equipment", $sess->username);
            $sql = 'INSERT INTO pictures (pic)
                    VALUES(EMPTY_BLOB()) RETURNING pic INTO :blobbind';
            $db->insertBlob($sql, 'Insert Logo BLOB', 'blobbind', $blobdata);
            echo '<p>New logo was uploaded</p>';
        }
    }
    echo "</div>";  // content
}
```

This is in the now familiar two part structure with an HTML form and a form-handler. The INSERT statement uses a bind value to represent the BLOB. The new Db class `insertBlob()` will associate the BLOB data with the bind variable and commit the record. The uploaded image will be added to the PICTURES table.

Complete `ac_logo_upload.php` by adding the form function `printform()`:

```
/**
 * Print the HTML form to upload the image
 *
 * Adding CSRF protection is an exercise for the reader
 */
function printform() {
    echo <<<EOF
Upload new company logo:
<form action="ac_logo_upload.php" method="POST" enctype="multipart/form-data">
<div>
    Image file name: <input type="file" name="lob_upload">
    <input type="submit" value="Upload"
```

```
</div>
<form
EOF;
}
```

> **Note:** The 'EOF;' token must be at the start of a line and not have trailing white space.

When this form is submitted the PHP web server will be able to access uploaded `BLOB` data in the temporary file `$_FILES['lob_upload']['tmp_name']`, as seen in `printcontent()`.

PHP has various options controlling locations and upper sizes of files, refer to the PHP documentation. The AnyCo application will use the default values.

Edit `ac_db.inc.php` and add the `insertBlob()` method to the `Db` class:

```
/**
 * Insert a BLOB
 *
 * $sql = 'INSERT INTO BTAB (BLOBID, BLOBDATA)
 *         VALUES(:MYBLOBID, EMPTY_BLOB()) RETURNING BLOBDATA INTO :BLOBDATA';
 * Db::insertblob($sql, 'do insert for X', myblobid',
 * $blobdata, array(array(":p", $p, -1)));
 *
 * $sql = 'UPDATE MYBTAB SET blobdata = EMPTY_BLOB()
 *         RETURNING blobdata INTO :blobdata';
 * Db::insertblob($sql, 'do insert for X', 'blobdata', $blobdata);
 *
 * @param string $sql An INSERT or UPDATE statement that returns a LOB locator
 * @param string $action Action text for End-to-End Application Tracing
 * @param string $blobbindname Bind variable name of the BLOB in the statement
 * @param string $blob BLOB data to be inserted
 * @param array $otherbindvars Bind variables. An array of tuples
 */
public function insertBlob($sql, $action, $blobbindname, $blob,
$otherbindvars = array()) {
    $this->stid = oci_parse($this->conn, $sql);
    $dlob = oci_new_descriptor($this->conn, OCI_D_LOB);
    oci_bind_by_name($this->stid, $blobbindname, $dlob, -1, OCI_B_BLOB);
    foreach ($otherbindvars as $bv) {
        // oci_bind_by_name(resource, bv_name, php_variable, length)
        oci_bind_by_name($this->stid, $bv[0], $bv[1], $bv[2]);
    }
    oci_set_action($this->conn, $action);
    oci_execute($this->stid, OCI_NO_AUTO_COMMIT);
    if ($dlob->save($blob)) {
        oci_commit($this->conn);
    }
}
```

The `insertBlob()` method accepts a final option parameter for normal bind variables. This is not used when it is called in `printcontent()` in `ac_logo_upload.php`.

The `BLOB` is bound as a special type, similar to how a `REF CURSOR` was bound in the Chapter 6, "Showing Equipment Records by Using a REF CURSOR." PHP OCI8 also has a `OCI_B_CLOB` constant which can be used for binding `CLOB`s. The `LOB` descriptor is an instance of PHP OCI8's `OCI-Lob` class, which has various methods for uploading

and reading data. When `oci_execute()` is processed on the SQL `INSERT` statement the `OCI_NO_AUTO_COMMIT` flag is used. This is because the database transaction must remain open until the `$dlob->save()` method inserts the data. Finally, an explicit `oci_commit()` commits the `BLOB`.

Run the AnyCo application in a browser and log in Administrator. Click the **Upload Logo** link in the left hand menu. Locate a JPEG image on your computer and select it. The next section of this chapter will display the image in the page header with the title, so choose an image of 15 to 20 pixels in height.

AnyCo Corp. Upload Logo

Logged in as: admin

- Employee List
- Equipment Report
- Equipment Graph
- Upload Logo
- Logout

Upload new company logo:
Image file name: [Choose File] oraclelogo.jpg   [Upload]

Click the **Upload** button.

Fetching the Logo and Creating an Image

Displaying the logo is similar in concept to how the graph image was displayed in the previous chapter. However since the `BLOB` is already in JPEG format the GD extension is not required.

Create a new PHP file `ac_logo_img.php`. The file contains:

```php
<?php

/**
 * ac_logo_img.php: Create a JPEG image of the company logo
 *
 * Don't have any text or white space before the "<?php" tag because it will
 * be incorporated into the image stream and corrupt the picture.
 *
 * @package Logo
 */

session_start();
require('ac_db.inc.php');
require('ac_equip.inc.php');

$sess = new \Equipment\Session;
$sess->getSession();
if (isset($sess->username) && !empty($sess->username)) {
    $username = $sess->username;
} else { // index.php during normal execution, or other external caller
    $username = "unknown-logo";
}

$db = new \Oracle\Db("Equipment", $username);
$sql = 'SELECT pic FROM pictures WHERE id = (SELECT MAX(id) FROM pictures)';
$img = $db->fetchOneLob($sql, "Get Logo", "pic");

header("Content-type: image/jpg");
echo $img;
```

```
?>
```

This queries the most recent logo and sends it back as a JPEG stream. If the image appears corrupted, comment out the `header()` and `echo` function calls and check if any text or white space is being emitted by the script.

The user name check differs from those used in previous sections. The logo is displayed on all pages including the login page before the web user name is known. Because `Db` accepts a user name for end-to-end tracing, `ac_logo_img.php` uses a bootstrap user name `unknown-logo`.

Edit `ac_db.inc.php` and add the `fetchOneLob()` method to the `Db` class:

```
/**
 * Runs a query that fetches a LOB column
 * @param string $sql A query that include a LOB column in the select list
 * @param string $action Action text for End-to-End Application Tracing
 * @param string $lobcolname The column name of the LOB in the query
 * @param array $bindvars Bind variables. An array of tuples
 * @return string The LOB data
 */
public function fetchOneLob($sql, $action, $lobcolname, $bindvars = array()) {
    $col = strtoupper($lobcolname);
    $this->stid = oci_parse($this->conn, $sql);
    foreach ($bindvars as $bv) {
        // oci_bind_by_name(resource, bv_name, php_variable, length)
        oci_bind_by_name($this->stid, $bv[0], $bv[1], $bv[2]);
    }
    oci_set_action($this->conn, $action);
    oci_execute($this->stid);
    $row = oci_fetch_array($this->stid, OCI_RETURN_NULLS);
    $lob = null;
    if (is_object($row[$col])) {
        $lob = $row[$col]->load();
        $row[$col]->free();
    }
    $this->stid = null;
    return($lob);
}
```

The `oci_fetch_array()` options could have included the `OCI_RETURN_LOBS` flag to indicate the data should be returned as a PHP string. The code here shows the column being returned as a locator instead. This shows how a locator can be operated on, here using the `load()` to read all the data and `free()` method to free up resources. If you had an application with very large data, the locator `read()` method could be used to process the `LOB` in chunks, which would be a memory efficient way of processing large data streams.

Unlike `insertBlob()` which bound using the `OCI_B_BLOB` type and was therefore specific for `BLOB`s, the `fetchOneLob()` can be used for both `BLOB` and `CLOB` data.

If an application processes multiple images (or chunks of an image) sequentially in a loop, for example:

```
while (($img = $db->fetchOneLob($sql, "Get Logo", "pic")) != null ) {
    dosomething($img);
}
```

then you can reduce PHP's peak memory usage by explicitly un-setting `$img` at the foot of the loop:

```
dosomething($img);
$unset($img);
```

This allows the memory allocated for the current $img to be reused for the next image data stream. Otherwise the original image memory is only freed after PHP constructs the second image and is ready to assign it to $img. This optimization is not needed by the AnyCo application.

Displaying the Logo

To display an uploaded logo in the AnyCo application, edit ac_equip.inc.php and un-comment the LOGO_URL definition:

```
define('LOGO_URL', 'http://localhost/ac_logo_img.php');
```

Make sure the URL is correct for your environment.

The logo is displayed in Page::printHeader(). Every standard page of the application will show the logo. Rerun the application to verify this:

Keeping images in the database allows the complete application data to be backed up and shared across all applications. However for performance you could consider implementing a caching technique that writes the logo to disk so it can be streamed directly without requiring the overhead of database access. The upload form could regenerate the disk file each time a new image is uploaded.

13

Monitoring Database Usage of the Application

This chapter contains the following topics:

Overview of Metadata

Throughout the implementation of the AnyCo application, metadata values were used for Oracle's end-to-end application tracing features. Values set were the:

- client identifier

- client information

- module

- action

The client identifier held a value uniquely associated with an end user. The other three values were effectively a descending hierarchy of data about the application's tasks.

The metadata values are semi-arbitrary text strings. Oracle records the metadata values and makes them available in certain database features, such as in the list of currently open connections. How your DBA uses those features and how your application is designed will determine what values an application should set.

In the AnyCo application the client identifier was set to `simon` or `admin`, depending on which web user was logged into the application. The client information was always set to 'AnyCo Corp.' Refer to `CLIENT_INFO` in `ac_db.inc.php`. The module was set when each page created an instance of the `Db` class. By choice, the module name used was always `Equipment`, indicating that this set of files was related to manipulating employees' equipment. The action varied with each SQL statement being run.

Viewing Metadata

To see where the metadata values are used, login to the AnyCo application and navigate through several pages.

On Windows, from the **Start** menu, select **Programs** (or **All Programs**), then **Oracle Database 11g Express Edition**, and then **Go To Database Administration Page**.

On Linux, click the **Application** menu (on Gnome) or the **K** menu (on KDE), then point to **Oracle Database 11g Express Edition**, and then **Go To Database Administration Page**. Navigate to the **Sessions** page and login as the `SYSTEM` user. This shows

You may see multiple entries. Apache will create a number of processes, any one of which might handle any of the HTTP requests. As you navigate through the application pages, different Apache processes handle the page requests.

Drill down by clicking the SID number for one of the AnyCo sessions:

Client and Application information is also available on this page.

The Oracle Database XE administration tool shows the SQL statement and statistics about its execution. If you see any poorly tuned or heavily used statements the end-to-end tracing metadata will let you locate the source PHP files for easy review and re-design. For statements, the metadata in effect at first execution of each unique statement is the value recorded.

The accuracy of the metadata is reliant upon the consistency of use in the applications that connect to the database.

Detailed information on how the client identifier can be used is in the technical article:

http://www.oracle.com/technetwork/articles/dsl/php-web-auditing-171451.htm
l

More Uses of Metadata

The Oracle Database Express Edition administration pane is a simplified view of all the information Oracle database records about connections and statement execution. Various standard database administration views such as V$SESSION and V$SQLAREA will also contain the tracing metadata. You can write your own queries or use other tools to present the information.

The client identifier metadata can be used to restrict data access. In Oracle Database XE you could manually augment each SQL statement to restrict access by testing the client identifier:

```
select * from equipment
where sys_context('userenv', 'client_identifier') = 'admin';
```

If a web user with another client identifier was logged in, the WHERE condition would evaluate false and no rows would be returned. Only the Administrator would be able to see data.

In the Enterprise Edition of Oracle Database, the Virtual Private Database feature supports creation of policy rules which will automatically restrict access to data. This removes the need for every SQL statement to be modified. The client identifier is also recorded in the audit log when auditing is enabled.

When using the client identifier for enforcing security, is a very important to have application code integrity. Is it imperative to ensure that there is no omission or impersonation of the client identifier.

Metadata and Persistent Connections

The AnyCo application uses PHP OCI8 persistent connections that are kept open even when the PHP processes are not processing scripts. The metadata values are not reset at the end of a user script and they are visible in the Session screen when the PHP process and the database connection are idle. Also the current values will remain in effect if any subsequent oci_pconnect() handled by the PHP process doesn't explicitly set them again.

Using non-persistent connections, calls to oci_connect() or oci_new_connect() won't have this behavior, since those connections are always closed after each PHP script completes.

In a busy system with little idle time, the left-over metadata for persistent connections is generally not an issue. The problem areas for DBAs are busy connections, not idle ones. Not un-setting the metadata gives maximum performance because it avoids a round trip between PHP and the database. This would slow down the whole system so it is not recommended. However, you could forcefully clear the metadata by adding this to the Db destructor:

```
$this->stid = oci_parse($this->conn,
            "begin
                dbms_session.clear_identifier;
                dbms_application_info.set_client_info('');
                dbms_application_info.set_module('', '');
            end;");
```

```
oci_execute($this->stid);
```

If you try this, first restart the web server to close all existing PHP persistent database connections.

No solution is perfect. If the PHP process crashes it won't be able to clear the values or notify the database to close a non-persistent connection.

14

Building Global Applications

This chapter discusses global application development in a PHP and Oracle Database 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 is fully internationalized to provide a global platform for developing and deploying global applications.

This chapter contains the following topics:

- Establishing the Environment Between Oracle and PHP
- Manipulating Strings
- Determining the Locale of the User
- Developing Locale Awareness
- Encoding HTML Pages
- Organizing the Content of HTML Pages for Translation
- Presenting Data Using Conventions Expected by the User

Establishing the Environment Between Oracle and PHP

Correctly setting up the connectivity between the PHP engine and the Oracle database is the 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.

PHP uses Oracle's C language OCI interface, 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 Portuguese user in Brazil running an application in Unicode, `NLS_LANG` should be set to

```
BRAZILIAN PORTUGUESE_BRAZIL.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 installation guides.

When PHP is installed on Oracle Linux's Apache, you can set `NLS_LANG` in `/etc/sysconfig/httpd`:

```
export NLS_LANG='BRAZILIAN PORTUGUESE_BRAZIL.AL32UTF8'
```

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, you must enable PHP's `mbstring` extension.

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

The PHP `intl` extension which wraps the ICU library is also popular for manipulating strings, see

http://www.php.net/intl

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 14–1 shows how some commonly used locales are defined in ISO and Oracle environments.

Table 14–1   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
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 by setting the PHP configuration variable as follows:

```
default_charset = UTF-8
```

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, image 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 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 thousands 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 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 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 Portuguese users in Brazil.

```
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_language = 'BRAZILIAN PORTUGUESE' nls_territory =
'BRAZIL';

Sessão alterada.

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        17/6/2003   terça-feira, 17 de junho de 2003
101   N.Kochhar     21/9/2005   quarta-feira, 21 de setembro de 2005
102   L.De Haan     13/1/2001   sábado, 13 de janeiro de 2001
103   A.Hunold      3/1/2006    terça-feira, 3 de janeiro de 2006
104   B.Ernst       21/5/2007   segunda-feira, 21 de maio de 2007
```

Oracle Number Formats

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

```
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=brazil;

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
```

> **Note:**   If the decimal and thousands separators used by Oracle are
> not '.' and ',' respectively, then you may see PHP errors when doing
> arithmetic on returned data values. This is because PHP will not
> correctly convert a string variable containing digits into an integer or
> float variable if the separators can't be parsed in PHP style. To avoid
> this problem you can set the format explicitly with:
>
> ```
> alter session set nls_numeric_characters = '.,'
> ```

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 `BRAZILIAN PORTUGUESE`, the server message appears as follows:

```
ORA-00942: a tabela ou view não existe
```

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

Index

Symbols

A

W

Web browser
   testing Apache installation on Windows,   2-10