

# The Underground PHP and Oracle<sup>®</sup> Manual

Release 1.2.1



CHRISTOPHER JONES AND ALISON HOLLOWAY

The Underground PHP and Oracle® Manual, Release 1.2.1

Copyright © 2006, Oracle. All rights reserved.

Authors: Christopher Jones and Alison Holloway

Contributors and acknowledgements: Vladimir Barriere, Luxi Chidambaran, Robert Clevenger, Ken Jacobs, Shoaib Lari, Simon Law, Krishna Mohan, Kevin Neel, Charles Poulsen, Roy Rossebo, Sreekumar Seshadri, Todd Trichler, Simon Watt

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

#### U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Oracle, JD Edwards, and PeopleSoft are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

# Contents

<b>Chapter 1</b> .....	<b>9</b>
<b>Introduction</b> .....	<b>9</b>
Who Should Read This Book? .....	9
Introduction to Oracle .....	9
Oracle Database Terminology .....	9
Introduction to PHP.....	10
Choosing a PHP Oracle Extension .....	11
<b>Chapter 2</b> .....	<b>13</b>
<b>PHP Oracle Extensions</b> .....	<b>13</b>
PHP Oracle Extensions .....	13
Oracle Extension.....	13
OCI8 Extension.....	13
PDO Extension .....	14
PHP Database Abstraction Libraries.....	14
ADODB.....	15
PEAR DB.....	15
PEAR MDB2 .....	15
Getting the OCI8 Extension .....	15
PHP and Oracle Installation Options .....	16
Zend Core for Oracle.....	16
PHP Version Numbering.....	17
OCI8 Function Names in PHP 5 .....	17
<b>Chapter 3</b> .....	<b>19</b>
<b>Installing Oracle Database 10g Express Edition</b> .....	<b>19</b>
Oracle Database Editions .....	19
Oracle Database XE .....	19
Installing Oracle Database XE on Linux.....	20
Installing Oracle Database XE on Debian, Ubuntu, and Kubuntu .....	21
Installing Oracle Database XE on Windows.....	22

Testing the Oracle Database XE Installation .....	24
Configuring Oracle Database XE.....	26
Setting the Oracle Database XE Environment Variables on Linux .....	26
Enabling Database Startup and Shutdown from Menus on Linux.....	26
Starting and Stopping the Oracle Listener .....	26
Starting and Stopping the Database .....	27
Enabling Remote Client Connection .....	28
<b>Chapter 4 .....</b>	<b>29</b>
<b>Using Oracle Database 10g .....</b>	<b>29</b>
Oracle Application Express.....	29
Logging In To Oracle Application Express .....	29
Creating Database Objects.....	30
Working with SQL Scripts .....	34
Creating a PL/SQL Procedure .....	36
Creating a Database User.....	38
Monitoring Database Sessions.....	40
Database Backup and Recovery .....	42
Oracle SQL Developer .....	45
Creating a Database Connection.....	46
Creating a Table.....	48
Executing a SQL Query.....	50
Running Reports .....	52
<b>Chapter 5 .....</b>	<b>55</b>
<b>Installing Apache HTTP Server .....</b>	<b>55</b>
Installing Apache HTTP Server on Linux.....	55
Starting and Stopping Apache HTTP Server .....	55
Installing Apache HTTP Server on Windows .....	56
Starting and Stopping Apache HTTP Server .....	57
<b>Chapter 6 .....</b>	<b>59</b>
<b>Installing Zend Core for Oracle .....</b>	<b>59</b>
Zend Core for Oracle.....	59
Installing Zend Core for Oracle .....	59
Configuring Zend Core for Oracle.....	61
Testing the Zend Core for Oracle Installation .....	62

<b>Chapter 7 .....</b>	<b>65</b>
<b>Installing PHP .....</b>	<b>65</b>
Installing PHP on Linux .....	65
Restart the Apache HTTP Server.....	66
Installing PHP on Windows .....	66
Restart the Apache HTTP Server.....	67
<b>Chapter 8.....</b>	<b>69</b>
<b>Installing PHP With Oracle Instant Client .....</b>	<b>69</b>
Installing Oracle Instant Client on Linux .....	69
Installing Oracle Instant Client on Windows .....	70
<b>Chapter 9 .....</b>	<b>73</b>
<b>Connecting to Oracle Using OCI8.....</b>	<b>73</b>
Oracle Connection Types .....	73
Standard Connections .....	73
Multiple Unique Connections.....	73
Persistent Connections.....	73
Oracle Database Name Connection Strings .....	74
Easy Connect String.....	74
Full Database Connection String .....	74
Database Alias .....	75
Oracle Environment Variables for Connections .....	75
Closing Oracle Connections.....	76
Connection Cleanliness with Persistent Connections .....	77
Optional Connection Parameters.....	78
Connection Character Set .....	78
Connection Privilege Level .....	79
The Transactional Behavior of Connections .....	81
Tuning Oracle Connections in PHP .....	81
Tuning Oracle Net .....	82
Other Oracle Net Optimizations .....	82
Tracing Oracle Net.....	83
Tuning Persistent Connections .....	83
Connection Management in Scalable Systems .....	85
<b>Chapter 10 .....</b>	<b>87</b>

<b>Executing SQL Statements With OCI8 .....</b>	<b>87</b>
SQL Statement Execution Steps .....	87
Query Example .....	87
Oracle Datatypes.....	88
Fetch Functions .....	88
Insert, Update, Delete, Create and Drop .....	89
Transactions .....	90
Autonomous Transactions .....	92
Handling Errors .....	93
Limiting Rows and Creating Paged Datasets .....	94
Auto-Increment Columns .....	95
Tuning SQL Statements in PHP Applications .....	96
Default Prefetch Size .....	96
Default Statement Cache Size.....	97
Using Bind Variables .....	97
Exploring SQL .....	98
Regular Expressions in SQL.....	98
Analytic Functions in SQL .....	99
<b>Chapter 11 .....</b>	<b>101</b>
<b>Using PL/SQL .....</b>	<b>101</b>
PL/SQL Overview.....	101
Blocks, Procedures, Packages and Triggers.....	102
Anonymous Block .....	102
Stored or Standalone Procedure and Function.....	102
Package .....	103
Trigger .....	103
Calling PL/SQL Procedures and Functions.....	103
Getting output from PL/SQL with DBMS_OUTPUT .....	104
Oracle Collections in PHP.....	106
Using PL/SQL Types in PHP .....	108
New Array Binding Function in PHP 5.1.2 .....	110
Using REF CURSORS for Result Sets .....	111
<b>Chapter 12 .....</b>	<b>113</b>
<b>Using Large Objects in OCI8 .....</b>	<b>113</b>

Working with LOBs .....	113
Other LOB Methods .....	114
Working with BFILEs .....	114
<b>Chapter 13 .....</b>	<b>119</b>
<b>Using XML with Oracle and PHP .....</b>	<b>119</b>
Fetching Relational Rows as XML .....	119
Using DBMS_XMLGEN .....	120
Accessing Data from Oracle over HTTP .....	120
XQuery XML Query Language .....	121
<b>Chapter 14 .....</b>	<b>123</b>
<b>Globalization .....</b>	<b>123</b>
Establishing the Environment Between Oracle and PHP .....	123
Manipulating Strings .....	124
Determining the Locale of the User .....	125
Developing Locale Awareness .....	125
Encoding HTML Pages .....	126
Specifying the Page Encoding for HTML Pages .....	126
Specifying the Encoding in the HTTP Header .....	126
Specifying the Encoding in the HTML Page Header .....	126
Specifying the Page Encoding in PHP .....	127
Organizing the Content of HTML Pages for Translation .....	127
Strings in PHP .....	127
Static Files .....	127
Data from the Database .....	127
Presenting Data Using Conventions Expected by the User .....	128
Oracle Date Formats .....	128
Oracle Number Formats .....	129
Oracle Linguistic Sorts .....	130
Oracle Error Messages .....	131
<b>Chapter 15 .....</b>	<b>133</b>
<b>Resources .....</b>	<b>133</b>
General Information and Forums .....	133
Documentation .....	133
Articles and Other References .....	134

Source and Binaries.....	134
PHP Bugs .....	135
Utilities.....	135
<b>Glossary .....</b>	<b>137</b>



## CHAPTER 1

# Introduction

This book shows how to use the PHP scripting language with the Oracle database. It covers installation of both Oracle and PHP, and shows how to use them together efficiently.

The installation and database discussion in this book highlights the Oracle Database 10g Express Edition, but everything covered in this book also applies to the other editions of the Oracle database. The PHP you write against Oracle Database 10g Express Edition can be run, without change, against all editions of the Oracle database as well.

The book contains much new material on PHP's Oracle OCI8 extension. It also incorporates several updated installation guides previously published on the Oracle Technology Network web site. The chapter on globalization comes from the *Oracle Database Express Edition 2 Day Plus PHP Developer Guide*.

We gratefully acknowledge all the Oracle staff that contributed to this book.

## Who Should Read This Book?

This book is aimed at PHP developers who are developing applications against an Oracle database. You may already be using another database and have a requirement or a preference to move to Oracle. You may be starting out with PHP database development. You may be unsure how to install PHP and Oracle. This book aims to remove any confusion.

This book is not a complete PHP or Oracle guide. It is assumed that you already have basic PHP and SQL knowledge and want best practices in using PHP against an Oracle database. Oracle documentation is freely available online and there are many other resources available. Extensive PHP documentation and resources are also online. However, for the first time, this book covers both technologies in the one document and shows how to make them work well together.

We suggest that you also read the *Oracle Database Express Edition 2 Day Plus PHP Developer Guide* which walks through building a PHP application against an Oracle database.

## Introduction to Oracle

The Oracle Database is well known for its scalability, reliability and features. It is the leading database and is available on many platforms.

There are many great books on Oracle, including Oracle's own extensive set of documentation you can read for more in-depth information on the architecture, and commands to use with Oracle. Oracle's documentation is online, refer to the Resources chapter for links.

## Oracle Database Terminology

There are some subtle differences between terminology used when describing an Oracle database, and databases from other software vendors. The following descriptions of the main

## Introduction

Oracle terms might help you to understand the Oracle terminology. Check the Glossary for more descriptions.

## Databases and Instances

An Oracle database stores and retrieves data. Each database consists of one or more data files. An Oracle *database server* consists of an Oracle *database* and an Oracle *instance*. Every time a server is started, a *system global area* (SGA) is allocated and the Oracle background processes are started. The combination of the background processes and memory buffers is called an Oracle *instance*. The instance consists of the running executable image of the Oracle software, plus a shared memory region for database block buffers and other run-time information. Some of this information is shared among all users, and some is specific to a user session. On some operating systems, like Windows, these are not “background processes”, but threads within the running Oracle image.

Although you may have more than one database per machine, typically a single Oracle database contains multiple schemas (that is, users). Multiple applications can use the same database without any conflict by using different schemas. Instead of using a CREATE DATABASE command for new applications, use the CREATE USER command to create a new schema in the database. In Oracle XE, there is a wizard to create new users in the Application Express management console.

## Tablespaces

Tablespaces are the logical units of data storage made up of one or more datafiles. Tablespaces are often created for individual applications because tablespaces can be conveniently managed. Users are assigned a default tablespace that holds all the data the users creates. A database is made up of default and DBA-created tablespaces.

## Schemas and Schema Objects

A schema is a collection of database objects. A schema is owned by a database user and has the same name as that user. Schema objects include tables, views, and indexes.

Once you have installed PHP and want to write scripts that connect to Oracle, you need to identify the schema name that contains the objects you want to interact with. The schema is the structure that contains these objects, like tables and indexes, and is the same name as the username in your connection string. So, to connect to the HR schema, you would use the username *hr*.

Some of the most common schema objects are:

- \* Tables
- \* Indexes
- \* Views

## Introduction to PHP

PHP is a hugely popular, interpreted scripting language commonly used for web applications. PHP is open source and free, and has a BSD-style license, making it corporation-friendly. PHP is perfect for rapidly developing applications both big and small, and is great for

creating Web 2.0 applications. It powers over twenty million web sites on the Internet and has a huge user community behind it. It runs on many platforms.

The language is dynamically typed and easy to use. PHP comes with many extensions offering all kinds of functionality. PHP5 introduced strong object orientated capabilities.

PHP code is commonly embedded in HTML:

```
<?php
    echo "<p>Hello</p>";
?>
<p>World</p>
```

PHP is typically installed as an Apache module, or run by the web server using FastCGI. When the PHP parser processes a file, it executes the PHP code and outputs any results. The HTML sections are echoed to the web browser verbatim.

The PHP command line interface (CLI) can also be used to run PHP scripts from an operating system shell window.

## Choosing a PHP Oracle Extension

Several PHP extensions provide PHP functions for accessing Oracle databases and there are also database abstraction layers written in PHP, which are popular. You can also use the ODBC extension.

This book discusses the OCI8 extension in detail. If you want to make full use of Oracle features and want high performance, OCI8 is the extension to use.

There are other choices available. If you want database independence, consider using the PHP Data Object (PDO) extension or ADOdb. For basic database operations, your PHP application will work with different database vendors simply by changing the extension driver.



## CHAPTER 2

# PHP Oracle Extensions

There are various ways PHP can connect to Oracle. Although this book concentrates on the OCI8 extension for PHP, it is worth knowing the alternative libraries and extensions you can use to connect your PHP applications to Oracle databases.

## PHP Oracle Extensions

The PHP Oracle extensions can be compiled into the PHP binary. The extensions are:

- \* Oracle
- \* OCI8
- \* PDO

They are all open source and included in the PHP source code.

### Oracle Extension

The extension called “Oracle” is included in PHP 3, 4 and 5.0. It is not in PHP 5.1 and is no longer maintained. This extension accesses the database using Oracle’s obsolete C code OCI7 API. New development using this extension is not recommended.

### OCI8 Extension

The OCI8 extension is included in PHP 3, 4 and 5. The OCI8 extension accesses the database using Oracle’s C code OCI8 API.

OCI8 was refactored by Zend and Oracle for stability and performance. The new code is included in PHP 5.1.2 onwards. The PHP interfaces are the same as the original OCI8 extension but there are new connection management and performance settings.

The refactored extension is designed to work with PHP4 and PHP5. If you are using PHP4 and cannot upgrade, you can replace just the original OCI8 code with the new version.

Zend have also bundled the new code into the easy to install Zend Core for Oracle (ZCO).

---

The name “OCI8” is the name for PHP’s Oracle extension, which provides PHP function calls to access an Oracle database. It is also the name for Oracle’s Call Interface API used by C programs including the PHP extension. All unqualified references to OCI8 in this book refer to the PHP extension.

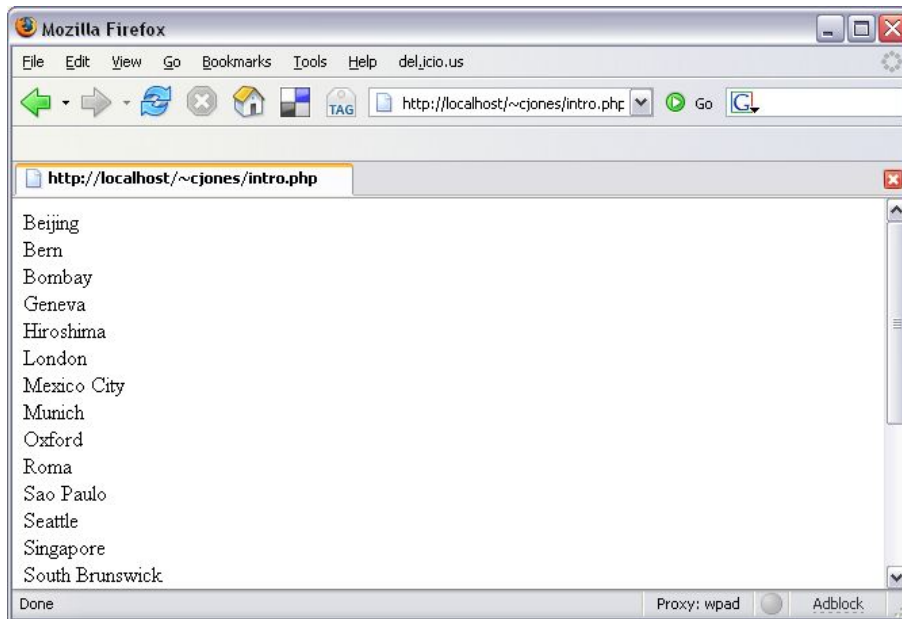
---

An example script that finds city names from the LOCATIONS table using OCI8:

*intro.php*

```
<?php
    $c = oci_connect('hr', 'hr_password', '//localhost/XE');
    $s = oci_parse($c, 'select city from locations');
    oci_execute($s);
    while ($res = oci_fetch_array($s)) {
        echo $res['CITY'] . "<br>";
    }
    oci_close($c);
?>
```

When invoked in a web browser, it connects to Oracle as the demonstration user, HR, of the Oracle XE database running on the local machine. The query is executed and a web page of results is displayed in the browser:



*Figure 2.1 PHP output in a web browser*

## PDO Extension

PHP Data Objects (PDO) is a database abstraction layer that provides PHP functions for accessing databases using a database independent interface. Each database has its own driver. PDO\_OCI provides the Oracle functionality for PDO. PDO\_OCI has no code in common with OCI8. The PDO extension is included in PHP 5.1.x onwards.

## PHP Database Abstraction Libraries

There are three main database abstraction libraries for PHP. They are written in PHP and when configured for Oracle, they use functionality provided by the OCI8 extension. The abstraction libraries are:

- \* ADOdb
- \* PEAR DB
- \* PEAR MDB2

You can freely download and use the PHP code for these libraries.

## ADOdb

The most complete and popular ADOdb extension is available from <http://adodb.sourceforge.net>. There is also an optional C extension plugin if you need extra performance.

## PEAR DB

The PEAR DB extension is available from <http://pear.php.net/package/DB> and includes the PHP extension and Application Repository.

## PEAR MDB2

The PEAR MDB2 extension is available from <http://pear.php.net/package/MDB2>. It is a new library, combining the best of PEAR DB and the PHP Metabase abstraction package, and is still being stabilized.

# Getting the OCI8 Extension

The OCI8 extension is included in PHP in various ways:

- \* Zend Core for Oracle bundle  
ZendCoreForOracle-v1.3.1-Linux-x86.tar.gz
- \* The full PHP source code of each patch release  
php-5.1.4.tar.gz
- \* The PHP Windows binaries of each patch release  
php-5.1.4-Win32.zip
- \* The two-hourly snapshots of the current source code  
php5.1-200604270230.tar.gz  
php5.1-win32-200604270230.zip
- \* As its own source code  
oci8-1.2.1.tar.gz
- \* Windows binary  
php\_oci8.dll

These last two can be used to update just the OCI8 extension in an existing PHP installation.

OCI8 is available in different ways because of the differing needs of the community. If this is your first time with PHP and Oracle, use the easy to install Zend Core for Oracle.

Many other PHP users install the full PHP source or binaries and do their own custom configuration. If they need a specific bug fix they then use PECL or CVS to get it.

PHP's source code is open source and under continual development in its CVS source code control system. (The code for the OCI8 driver is at <http://cvs.php.net/viewcvs.cgi/php-src/ext/oci8/>). Having open access to the source code means:

- \* Anyone can access CVS and pull the code. (Anyone can seek approval to contribute too.) The two-hourly PHP snap-shots are a complete view of all PHP's source in CVS at the time the snap-shot was created. You can update your PHP environment by getting this source code and recompiling. These builds may be relatively unstable because the code is flux.
- \* Each new stable version of PHP (available from <http://www.php.net/downloads.php>) is taken from the most current CVS code at the time of release. PHP is available as source code or Windows binaries.
- \* PHP Extension Community Library (PECL) source code snapshots <http://pecl.php.net/package/oci8> are taken from CVS.
- \* Windows binaries of the OCI8 DLL for various versions of PHP are automatically built from the latest CVS source. They are at [http://pecl4win.php.net/ext.php/php\\_oci8.dll](http://pecl4win.php.net/ext.php/php_oci8.dll).
- \* Zend Core for Oracle <http://www.oracle.com/technology/tech/php/zendcore/> also takes snapshots from CVS and bundles them.

The schedules of PHP releases, the PECL source snapshots, and Zend Core for Oracle are not fully synchronized.

## PHP and Oracle Installation Options

Depending what software you have already installed, and what you want to achieve, there are several configuration options you could use when installing PHP.

If your database is (or will be) installed on the same machine as the web server and PHP, you can compile your own PHP and link with Oracle's libraries. On Windows you can use PHP's pre-supplied binaries.

If your database is installed on another machine on your network, you can install Oracle Instant Client (which is just a few libraries) and compile PHP with that. On Windows you can use the Windows PHP binaries with Instant Client.

If you don't want to compile PHP yourself, install Zend Core for Oracle.

## Zend Core for Oracle

Zend Core for Oracle (ZCO) is the result of a joint project between Oracle and Zend. It is a free, pre-built release of PHP that comes enabled with the refactored OCI8 extension. ZCO is available for many platforms and has a convenient browser-based management console for configuration and getting updates.

It has a simple installer, installs on various web servers, and comes with Oracle Instant client. ZCO can connect to local and remote Oracle databases.

ZCO is free to download and use. A support package that includes support for the PHP language is available from Zend.



## PHP Version Numbering

The easiest way to check if you have the refactored OCI8 driver is to call `phpinfo()`:

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

If the OCI8 section lists seven directives (starting with `oci8.default_prefetch`) you have the refactored extension. The older version had no directives.

If you think you have found a bug in OCI8, the PHP community will ask you to test the latest PHP build before submitting a bug report. If you include the OCI8 version number in your bug report, it will be easier for the maintainer to see what the problem is.

The most accurate way to refer to a version of OCI8 is with the `phpinfo()` version number. This version is the revision number of the `oci8.c` source file.

Other bundles that include OCI8 may have their own numbers. For example, a snapshot of the OCI8 source code from PHP's CVS was bundled in PECL as "1.2 stable". The revision number of its `oci8.c` shown by `phpinfo()` is 1.293.

Zend Core for Oracle labels itself with a major version, for example, "1.3". Check the `phpinfo()` output for the OCI8 version.

## OCI8 Function Names in PHP 5

In PHP5, the OCI8 extension function names were standardized. PHP4 functions like `OCILogin()` became `oci_connect()`, `OCIParse()` became `oci_parse()` and so on. The old names still exist as aliases, so PHP4 scripts do not need to be changed.

One side effect of this function renaming is that user contributed comments in the [PHP manual](http://www.php.net/oci8) (<http://www.php.net/oci8>) are found in two differing places for the same functionality. The comments are either on the page for the old syntax, or on the page for the new syntax. Where a function has a synonym, check both manual pages for user tips.



## CHAPTER 3

# Installing Oracle Database 10g Express Edition

This chapter contains an overview of, and installation instructions for, Oracle Database 10g Express Edition (Oracle Database XE). The installation instructions are given for Linux, Windows, Debian, Ubuntu and Kubuntu.

## Oracle Database Editions

There are a number of editions of the Oracle database, each with different features, licensing options and costs. The Oracle database edition names are:

- \* Express Edition
- \* Standard Edition One
- \* Standard Edition
- \* Enterprise Edition

All the editions are built using the same code base. That is, they all have the same source code, but different features are implemented in each edition. Enterprise Edition has all the bells and whistles, whereas the Express Edition has a limited feature set, but still has all the reliability and performance of the Enterprise Edition.

You could start off with the Express Edition, and, as needed, move up to another edition as your scalability and support requirements change. You could do this without changing any of your underlying table structure or code. Just change the Oracle software and you're away.

There's a comprehensive list of the features for each Oracle edition at [http://www.oracle.com/database/product\\_editions.html](http://www.oracle.com/database/product_editions.html).

This book discusses working with Oracle Database 10g Express Edition (Oracle Database XE). This is the free edition. Free to download. Free to develop against. Free to distribute with your applications. Yes, that is free, free, free!

## Oracle Database XE

Oracle Database XE is available on 32-bit Windows and Linux platforms. Oracle Database XE is a good choice for development of PHP applications that require a free, small footprint database.

Oracle Database XE is available on the Oracle Technology Network (<http://otn.oracle.com/x>) for the following operating systems:

- \* Windows 2000 Service Pack 4 or later
- \* Windows Server 2003
- \* Windows XP Professional Service Pack 1 or later

## Installing Oracle Database 10g Express Edition

- \* Red Hat Enterprise Linux RHEL3 and RHEL4
- \* Suse SLES-9
- \* Fedora Core 4
- \* Red Flag DC Server 5.0/MIRACLE LINUX V4.0/Haansoft Linux 2006 Server (Asianux 2.0 Inside)
- \* Debian 3.1

The following libraries are required for the Linux-based operating systems:

- \* glibc release 2.3.2
- \* libaio release 0.3.96

There are a few limitations with Oracle Database XE:

- \* 4GB of data
- \* Single database instance
- \* Single CPU used
- \* 1GB RAM used

Oracle Database XE has a browser-based management interface, Oracle Application Express. Support for Oracle Database XE is through an Oracle Technology Network (<http://otn.oracle.com/>) discussion forum, which is populated by peers and product experts. You cannot buy support from Oracle for Oracle Database XE.

If you need a fully supported version for the Oracle database, you should consider Oracle Standard Edition or Enterprise Edition. You can download all the editions of the Oracle Database from the Oracle Technology Network, and use these for application development and testing, but when you go production, you'll need to pay Oracle for the license costs.

## Installing Oracle Database XE on Linux

If you do not have a version of *libaio* over release 0.3.96, you will need to install this library before you can install Oracle Database XE

The installation procedure on the *Oracle Database Developer CD for Linux x86* differs slightly to the procedure below. If you are installing from this CD, follow the instructions outlined in the document named *Start\_Here.html* at the root of the CD. To install Oracle Database XE:

1. Download the Oracle Database XE from <http://www.otn.com/xe>.
2. Log in or *su* as *root*

```
su  
Password:
```

3. Install the RPM

```
rpm -ivh oracle-xe-10.2.0.1-1.0.i386.rpm
```

Oracle Database XE installs.

## Kubuntu

## 4. Configure the database

```
/etc/init.d/oracle-xe configure
```

5. Accept the default ports of 8080 for Application Express, and 1521 for the Database Listener.
6. Enter and confirm the password for the default users.
7. Enter **Y** or **N** for whether you want the database to start automatically on reboot. The database and database listener are configured and started.

## Installing Oracle Database XE on Debian, Ubuntu, and Kubuntu

There is an Advanced Package Tool (“apt-get”) repository available on the Oracle Open Source web site for Oracle Database XE. To include this repository, add the following to the file `/etc/apt/sources.list`:

```
deb http://oss.oracle.com/debian unstable main non-free
```

`libaio` and `bc` are included in the repository, and will be installed from the repository if you don’t already have them installed.

If you download Oracle Database XE from the Oracle Technology Network (<http://www.otn.com/xe>), you need to make sure that you have already installed the `libaio` and `bc` packages. If you are using Ubuntu or Kubuntu, the `bc` package is installed by default on the desktop version, but not on the server version.

To install Oracle Database XE on Debian, Ubuntu and Kubuntu, follow these steps:

1. Log in or su as root

```
su
Password:
```

2. Install Oracle Database XE

```
# apt-get update
# apt-get install oracle-xe
```

If you haven’t added the apt-get repository, you can download Oracle Database XE from <http://www.otn.com/xe>, and run the following command to begin the install:

```
dpkg -i downloads/oracle-xe-universal_10.2.0.1-1.0_i386.deb
```

Oracle Database XE installs.

3. Configure the database

```
/etc/init.d/oracle-xe configure
```

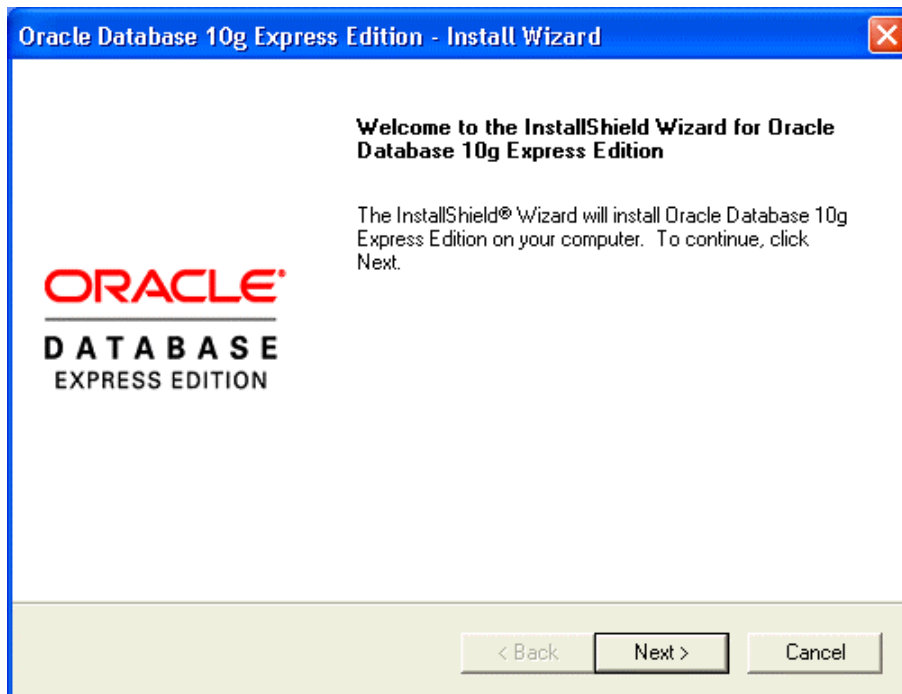
4. Accept the default ports of 8080 for Application Express, and 1521 for the Database Listener.
5. Enter and confirm the password for the default users.

6. Enter **Y** or **N** for whether you want the database to start automatically on reboot. The database and database listener are configured and started.

## Installing Oracle Database XE on Windows

To install Oracle Database XE on Windows, follow these steps:

1. Log on to Windows as a user with Administrative privileges.
2. If the `ORACLE_HOME` environment variable has been set, delete it using the **Control Panel > System** dialog.
2. Download the Oracle Database XE from <http://www.otn.com/xe>.
3. Double click on the *OracleXE.exe* file.
4. In the Oracle Database XE - Install Wizard welcome window, click **Next**.



*Figure 3-1 Oracle Database XE install welcome dialog.*

5. In the License Agreement window, select **I accept** and then click **Next**.
6. In the Choose Destination Location window, either accept the default or click Browse to select a different installation directory. Then click **Next**.

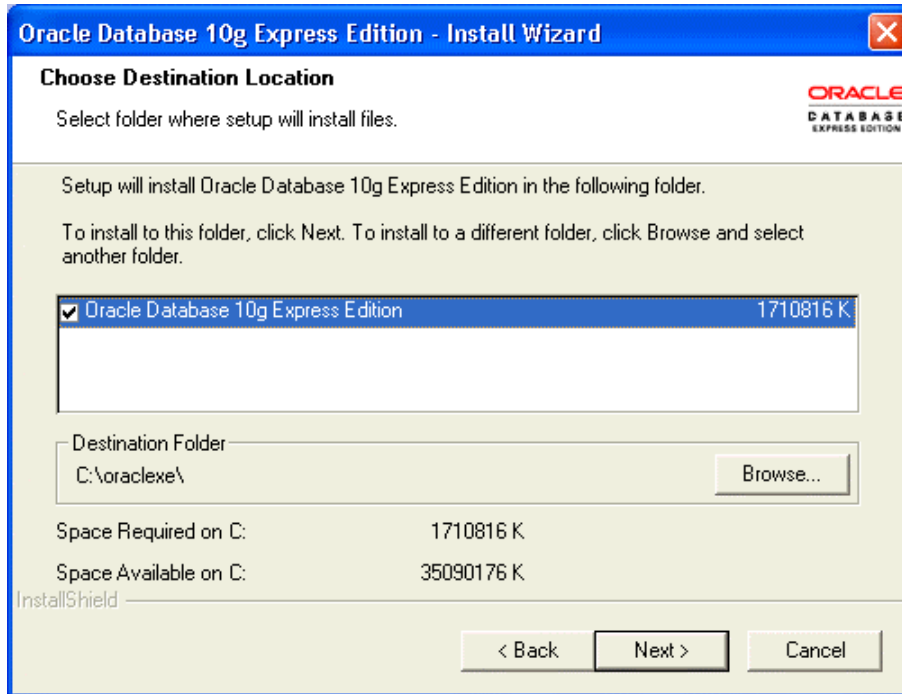


Figure 3-2 Oracle Database XE install location dialog.

7. Oracle Database XE requires a number of ports and selects a number of default ports. If these ports are already being used, you are prompted to enter another port number.
8. In the Specify Database Passwords window, enter and confirm the password to use for the SYS and SYSTEM database accounts. Then click **Next**.

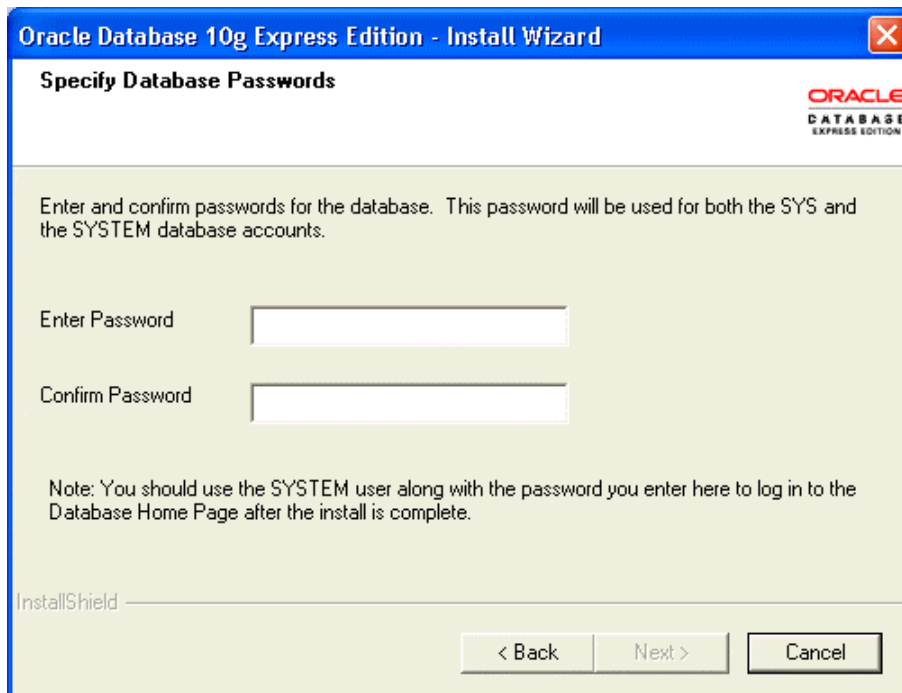
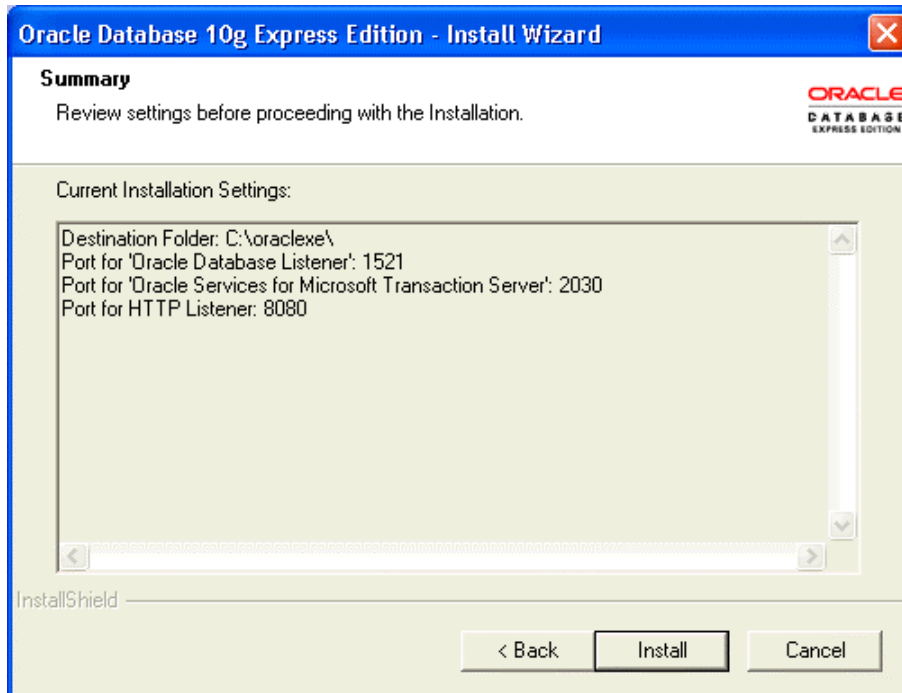


Figure 3-3 Oracle Database XE database password dialog

9. In the Summary window, review the installation settings. Then click **Install**.



*Figure 3-4 Oracle Database 10g install summary dialog.*

10. In the InstallShield Wizard Complete window, click **Launch the Database homepage** to display the Database Home Page, then click **Finish**.

## Testing the Oracle Database XE Installation

To test the installation of Oracle Database XE:

1. If you don't already have the Database homepage displayed in your web browser, open a web browser and enter:

`http://127.0.0.1:8080/apex`

The Database homepage is displayed.



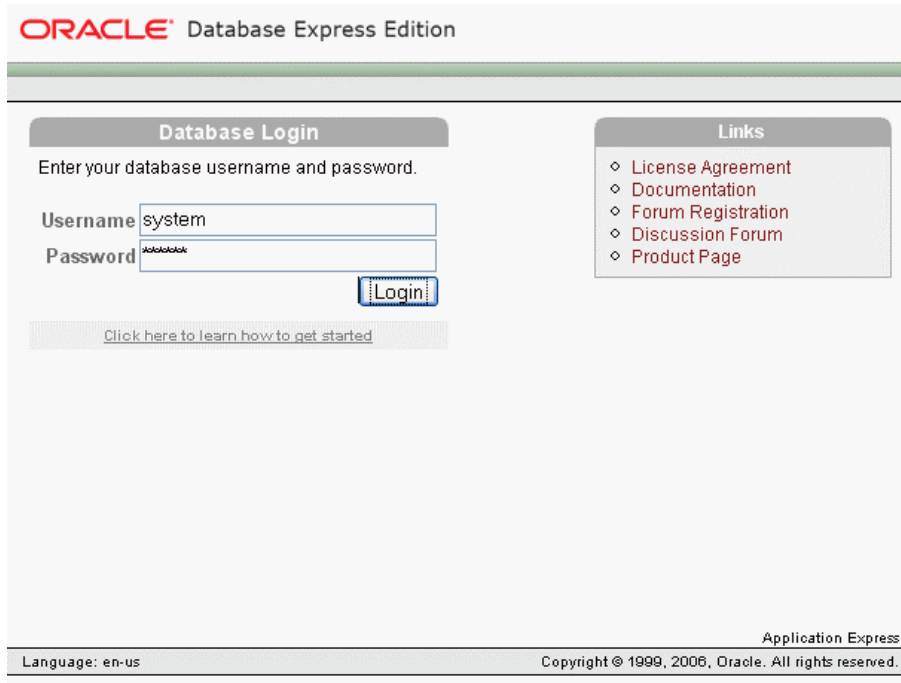


Figure 3-5 Database home page login screen

2. Log in as user **system** with the password you entered during the installation. You should now be logged into the Oracle Database homepage.

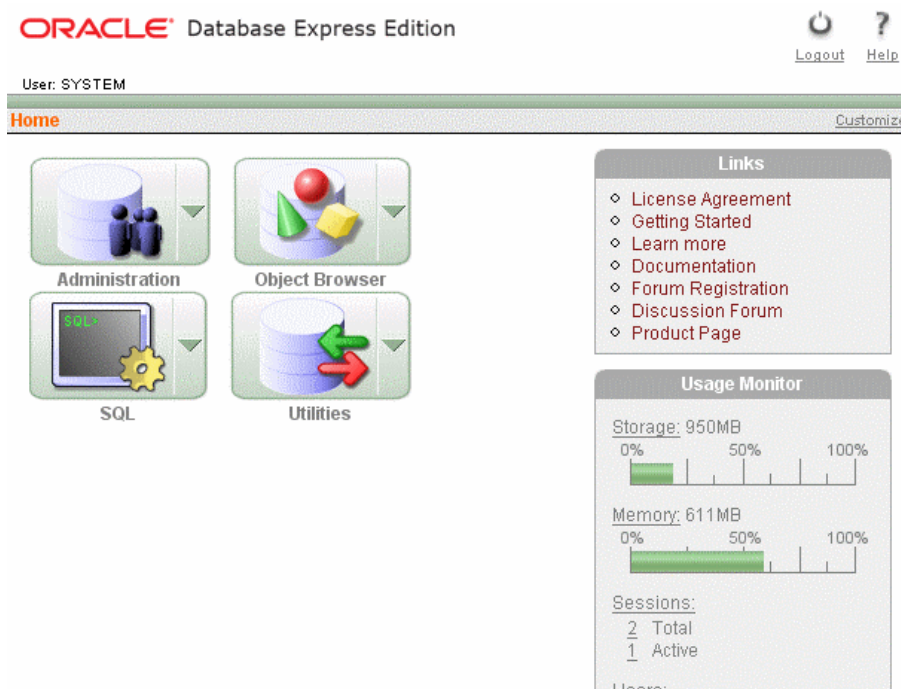


Figure 3-6 Oracle Database XE home page

## Configuring Oracle Database XE

There are a number of environment settings and configuration options you can set for Oracle Database XE. The more commonly used settings are discussed here.

### Setting the Oracle Database XE Environment Variables on Linux

On Linux platforms, there are a number of environment variables that should be set when you start up the database. A script is provided to enable you to set the Oracle environment variables when you log in. The script is located in `/usr/lib/oracle/xe/app/oracle/product/10.2.0/server/bin`, and named `oracle_env.csh` for C and tcsh shells, and `oracle_env.sh` for Bourne, Bash and Korn shells. Run the appropriate script for your shell to set your Oracle XE environment variables.

You can also add this script to your login profile to have the environment variables set up automatically when you log in. To add the script to your login profile for C and tcsh shells, add the following line to your `.login` or `.cshrc` file:

```
source
/usr/lib/oracle/xe/app/oracle/product/10.2.0/server/bin/oracle_env.csh
```

To add the script to your Bourne, Bash or Korn shell, add the following line to your `.bash_profile` or `.bashrc` file:

```
.
/usr/lib/oracle/xe/app/oracle/product/10.2.0/server/bin/oracle_env.sh
```

### Enabling Database Startup and Shutdown from Menus on Linux

You may not be able to start and stop the database using the menu on Linux platforms. This is because your user is not a member of the operating system 'dba' group. To enable this functionality, add the user name to the 'dba' group using the System Settings.

### Starting and Stopping the Oracle Listener

The database listener is an Oracle Net program that listens for and responds to requests to the database. The database listener must be running to handle these requests. After installing Oracle Database XE, the database listener should already be running, and you may have requested during the installation that the listener should be started when the operating system starts up. If you need to manually start or stop the database listener, the options and commands for this are listed below.

To start the database listener on Linux platforms, run the following command in your shell:

```
/etc/init.d/oracle-xe start
```

To stop the database listener, run the following command in your shell:

```
/etc/init.d/oracle-xe stop
```

This script also starts up the database on Linux platforms, so you do not need to manually start the database.

To start the database listener on Windows platforms, open the Services dialog using **Start > Settings > Control Panel > Administrative Tools > Services**, and select the **OracleXETNSListener** service. Right click on the Listener service, and select **Start**.

To stop the database listener on Windows platforms, use the same procedure as starting the Listener, but select **Stop** from the menu.

On Windows platforms, you need to manually start the database after starting the database listener.

## Starting and Stopping the Database

The database is another process that runs in memory, and needs to be started before Oracle Net can handle connection requests to it. After installing Oracle Database XE, the database should already be running, and you may have requested during the installation that the database should be started when the operating system starts up. If you need to manually start or stop the database, you can do this using the menu, or from the command line.

To start the database, you must log in as a user who is a member of the *dba* user group. This applies to all the methods of starting and stopping the database.

### Starting and Stopping the Database on Linux

To start up the database using the desktop, do one of the following:

- \* On Linux with Gnome: Select **Applications > Oracle Database 10g Express Edition > Start Database**.
- \* On Linux with KDE: Select **K Menu > Oracle Database 10g Express Edition > Start Database**.

To shut down the database using the desktop, do one of the following:

- \* On Linux with Gnome: Select **Applications > Oracle Database 10g Express Edition > Stop Database**.
- \* On Linux with KDE: Select **K Menu > Oracle Database 10g Express Edition > Stop Database**.

To start the database using the command line, run the following command in your shell:

```
/etc/init.d/oracle-xe start
```

To stop the database using the command line, run the following command in your shell:

```
/etc/init.d/oracle-xe stop
```

This script also starts up the database listener on Linux platforms, so you do not need to manually start the database listener.

### Starting and Stopping the Database on Windows

To start the database on Windows platforms, open the Services dialog using **Start > Settings > Control Panel > Administrative Tools > Services**, and select the **OracleServiceXE** service. Right click on the database service, and select **Start**.

To stop the database on Windows platforms, use the same procedure as starting the database, but select **Stop** from the menu.

## Starting and Stopping the Database Using SQL\*Plus

You can also use SQL\*Plus command line to start and stop the database. You can use SQL\*Plus on all operating systems.

To use SQL\*Plus to start and stop the database, you must log in as a user with the SYSDBA role. This is the SYS user in default installations, or you can use operating system authentication if you are on the local machine. To start up a database using SQL\*Plus, enter the following at the command line prompt:

```
sqlplus / as sysdba
SQL>startup
```

The database is started.

To shut down the database, you need to log in as SYSDBA, and issue the SHUTDOWN command:

```
sqlplus / as sysdba
SQL>shutdown immediate
```

The SQL\*Plus User's Guide and Reference gives you the full syntax for starting up and shutting down the database if you need more help.

## Enabling Remote Client Connection

The Oracle Database XE home page is only available from the local machine, not remotely. If you want to enable access from a remote client, you should be aware that HTTPS cannot be used (only HTTP), so your login credentials are sent in clear text, and are not encrypted, so if you don't need to set this up, it is more secure to leave it as the default setup.

To enable connection to the Oracle Database XE home page from remote machines, follow these steps:

1. Open a web browser and log into the Oracle Database XE home page (<http://127.0.0.1:8080/apex>) as *system*.
2. Select Administration from the home page.
3. Select **Manage HTTP Access** from the **Tasks** option.
4. Check the **Available from local server and remote clients** radio button, then click the **Apply Changes** button.

You can also use SQL\*Plus command line to enable access from remote clients. To use SQL\*Plus command line to change this setting, log into SQL\*Plus as *system*, and run the following command:

```
EXEC DBMS_XDB.SETLISTENERLOCALACCESS(FALSE);
```

## CHAPTER 4

# Using Oracle Database 10g

This chapter contains an overview of Oracle Application Express, and Oracle SQL Developer, two applications that you can use to perform database development and administration with Oracle Database 10g Express Edition.

## Oracle Application Express

Oracle Application Express is a browser-based interface to the Oracle Express Edition database. It is also available for download from Oracle's Technology Network (<http://otn.oracle.com>) as a standalone product to use to administer your database. Oracle Application Express also contains a database development tool, but this book does not cover that part of the product, just the database management and scripting components.

Oracle Application Express enables you to perform database administration, monitoring and maintenance. Command line utilities are also available, and we will show you both the GUI and command line tools in this book.

## Logging In To Oracle Application Express

To start and log in to Oracle Application Express:

1. Open a web browser and enter the URL <http://127.0.0.1:8080/apex>. The Oracle Application Express login screen is displayed.

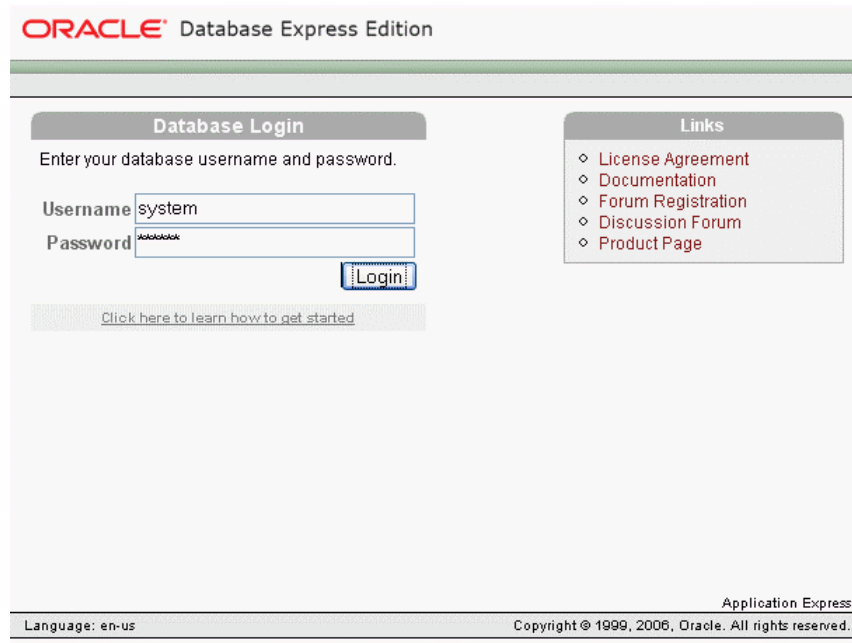


Figure 4-1 Oracle Application Express login screen.

2. Log in as **system**. The password you enter here is the password you entered when you installed Oracle. There is no default **system** password set by the Oracle Installer.

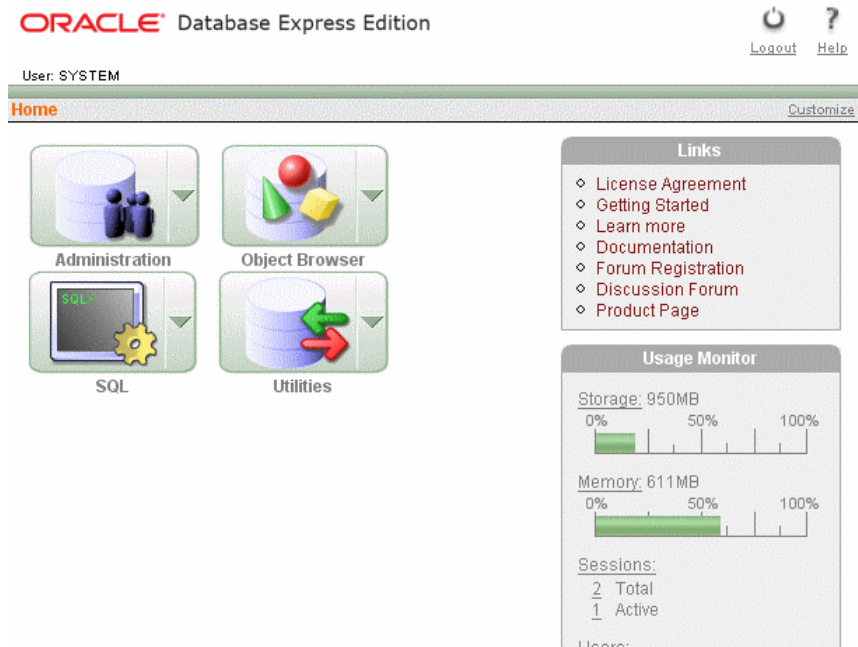


Figure 4-2 Oracle Application Express Interface

## Creating Database Objects

The Oracle Application Express Object Browser can be used to create or edit the following objects:

- \* Table
- \* View
- \* Index
- \* Sequence
- \* Collection Type
- \* Package
- \* Procedure
- \* Function
- \* Trigger
- \* Database Link
- \* Materialized View
- \* Synonym

Oracle Application Express uses wizards to guide you through creating these database objects. The following example covers creating a table, but you will see that the interface

is wizard-based and creating and editing different objects can all be performed through the Object Browser. To create a new table:

1. On the Database home page, click the **Object Browser** icon. The Object Browser page is displayed.

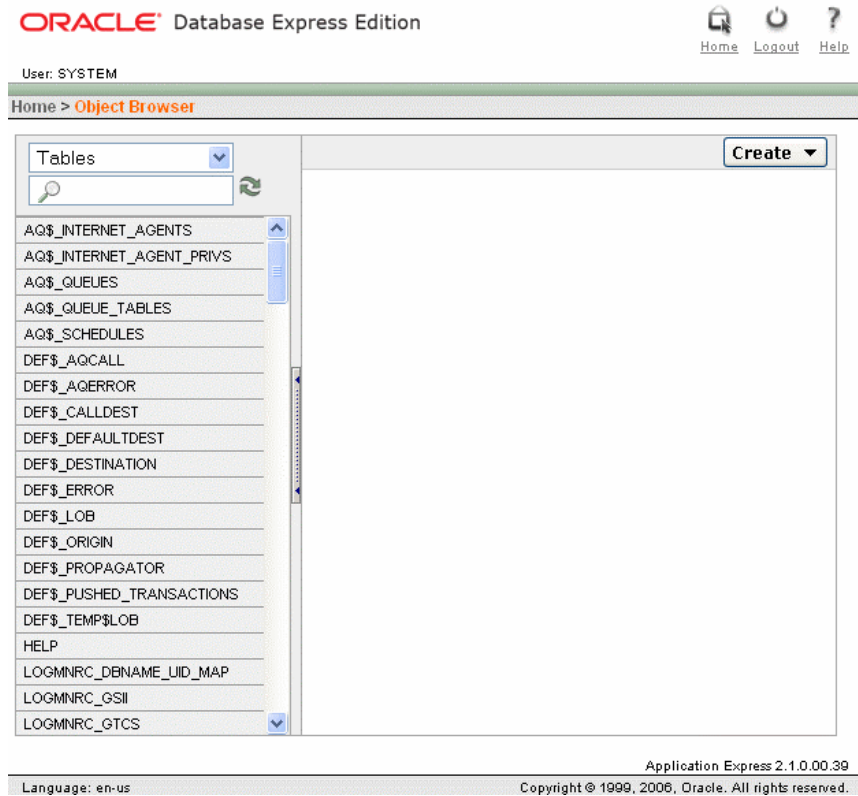


Figure 4-3 Oracle Application Express create object screen

2. Click **Create**.

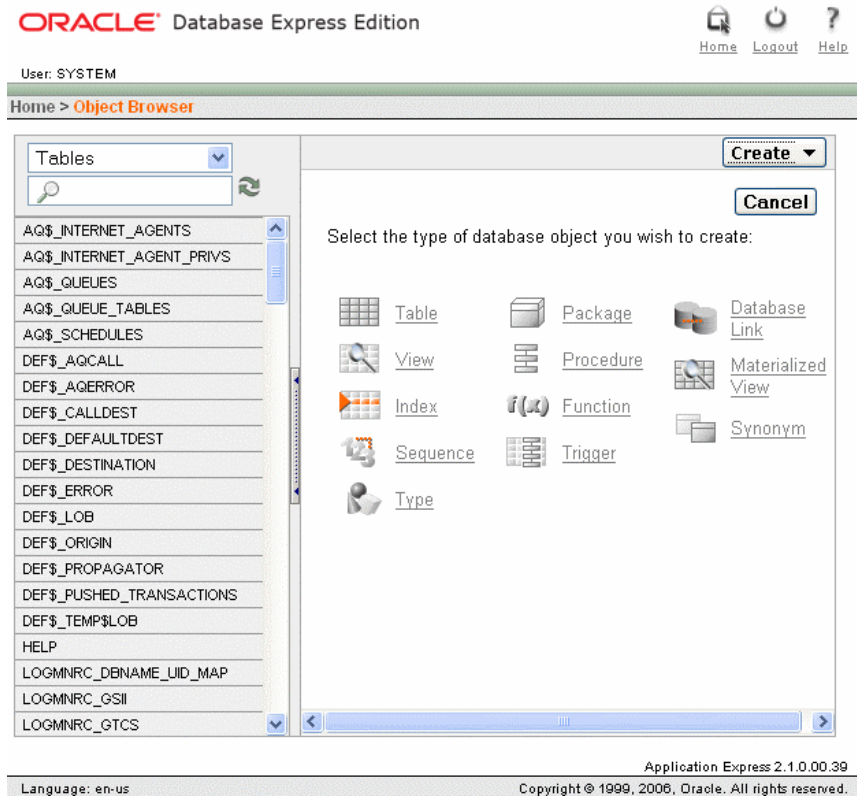


Figure 4-4 Oracle Application Express create table object screen

3. From the list of object types, select **Table**.

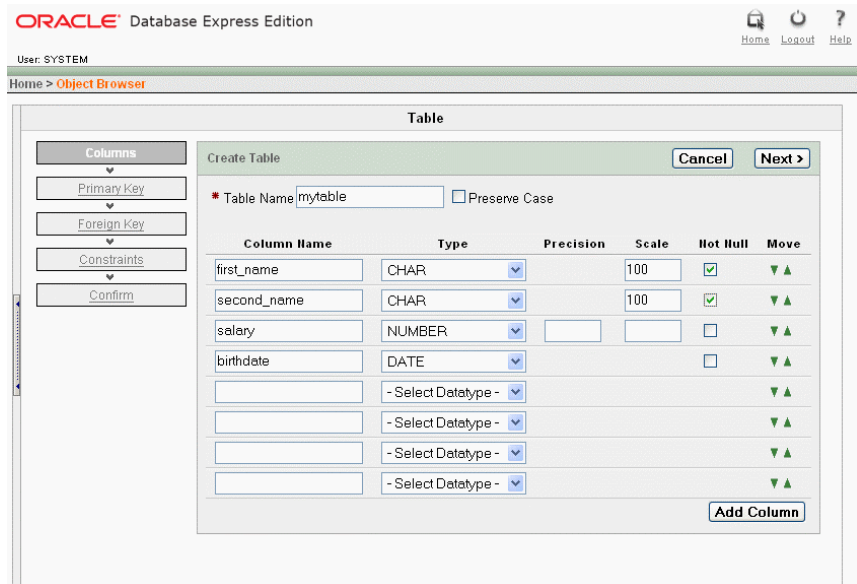


Figure 4-5 Oracle Application Express table definition screen



4. Enter a table name in the Table Name field, and details for each column. Click **Next**.
6. Define the primary key for this table (optional).
7. Add foreign keys (optional).
8. Add a constraint (optional).
9. Click **Finish**. To view the SQL used to create the table, click **SQL Syntax**.

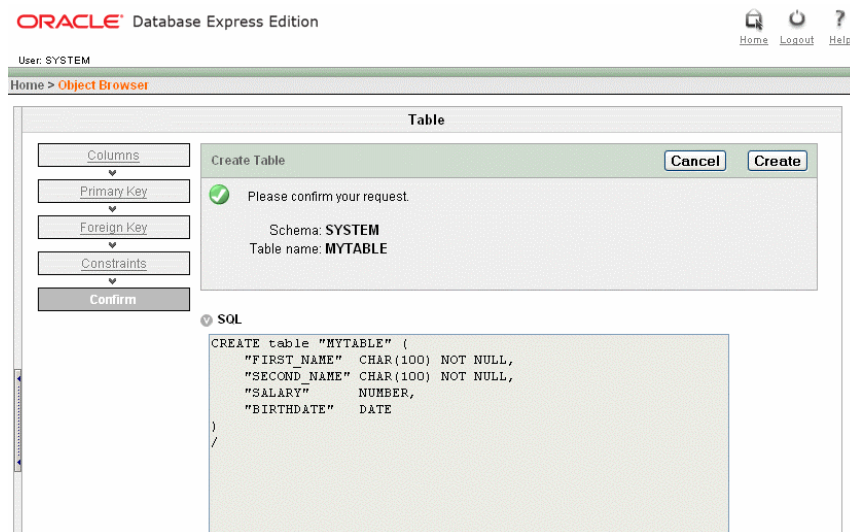


Figure 4-6 Oracle Application Express confirm create table screen.

The SQL that is generated to create the table is:

```
CREATE table "MYTABLE" (
    "FIRST_NAME" CHAR(100) NOT NULL,
    "SECOND_NAME" CHAR(100) NOT NULL,
    "SALARY" NUMBER,
    "BIRTHDATE" DATE
)
/
```

You could also run this command in SQL\*Plus command line to create the table.

10. Click **Create**. The table is created and a description of the table is displayed.

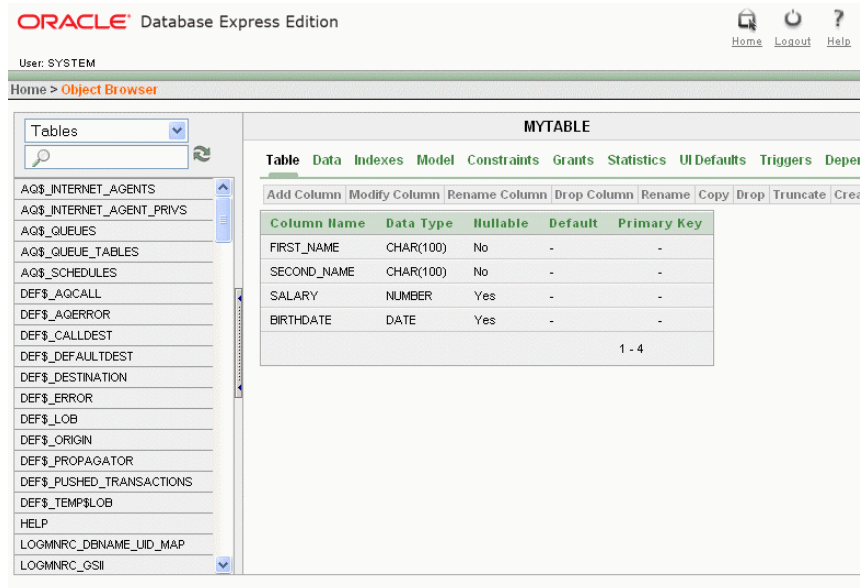


Figure 4-7 Oracle Application Express table created confirmation screen.

## Working with SQL Scripts

The SQL Console enables you to:

- \* Write SQL and PL/SQL
- \* Load and save SQL scripts
- \* Graphically build SQL

The following example guides you through creating a SQL script. This example uses the Query Builder to graphically create a SQL script to query data. To get a better set of tables to work with in this example, log in as the *hr* user. To access Query Builder:

1. On the Database home page, click the **SQL** icon.
2. Click the **Query Builder** icon.

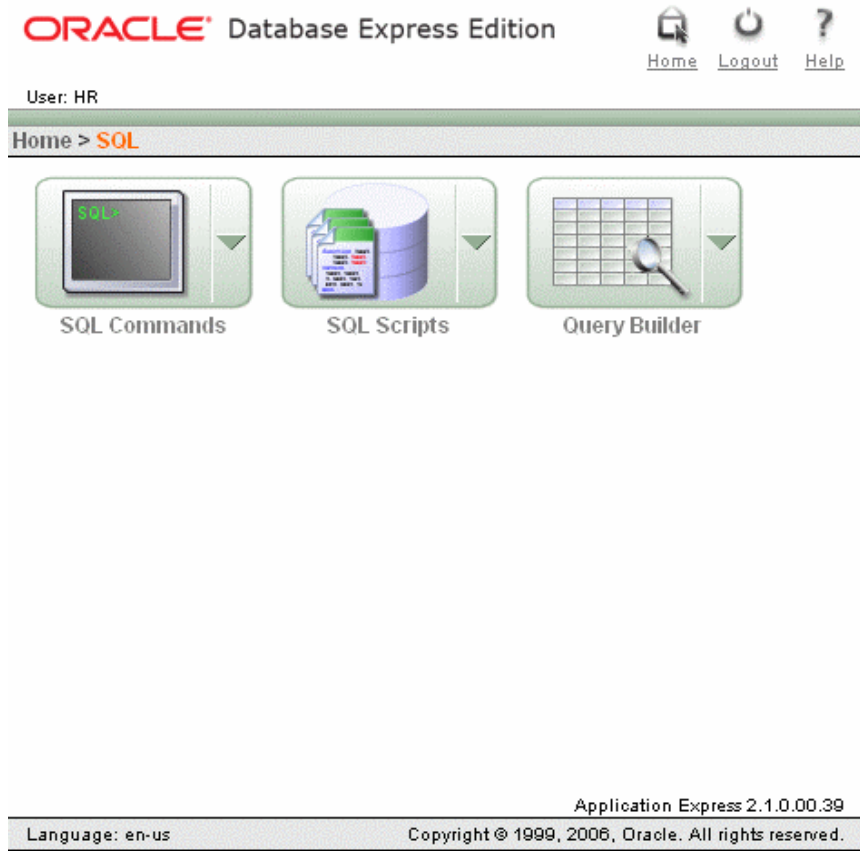


Figure 4-8 Oracle Application Express SQL options screen.

3. Select objects from the Object Selection pane. When you click on the object name, it is displayed in the Design pane.

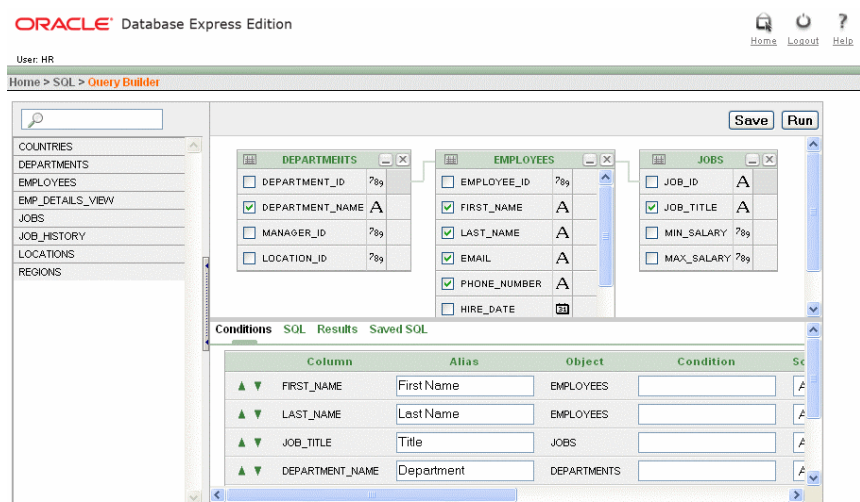


Figure 4-9 Oracle Application Express SQL query builder screen.

4. Select columns from the objects in the Design pane to select which columns to include in the query results.
5. Establish relationships between objects by clicking on the right-hand column of each table (optional).
6. Create query conditions (optional).
7. Click **Run** to execute the query and view the results.

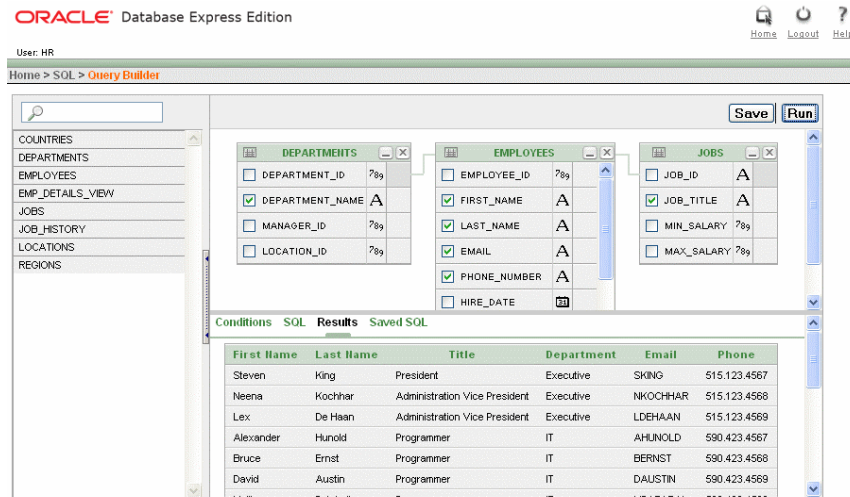


Figure 4-10 Oracle Application Express SQL query results screen.

8. You can save your query using the **Save** button.

## Creating a PL/SQL Procedure

To enter and run PL/SQL code in the SQL Commands page:

1. On the Database home page, click the **SQL** icon to display the SQL page.
2. Click the **SQL Commands** icon to display the SQL Commands page.
3. On the SQL Commands page, enter some PL/SQL code. Here is some PL/SQL code to try if you aren't familiar with PL/SQL. This procedure, *emp\_stat*, averages the salary for departments in the *hr* schema, and is encapsulated in a PL/SQL package called *emp\_sal*. You need to enter each block separately, and click **Run**.

```
create or replace package emp_sal as
procedure emp_stat;
end emp_sal;
/
```

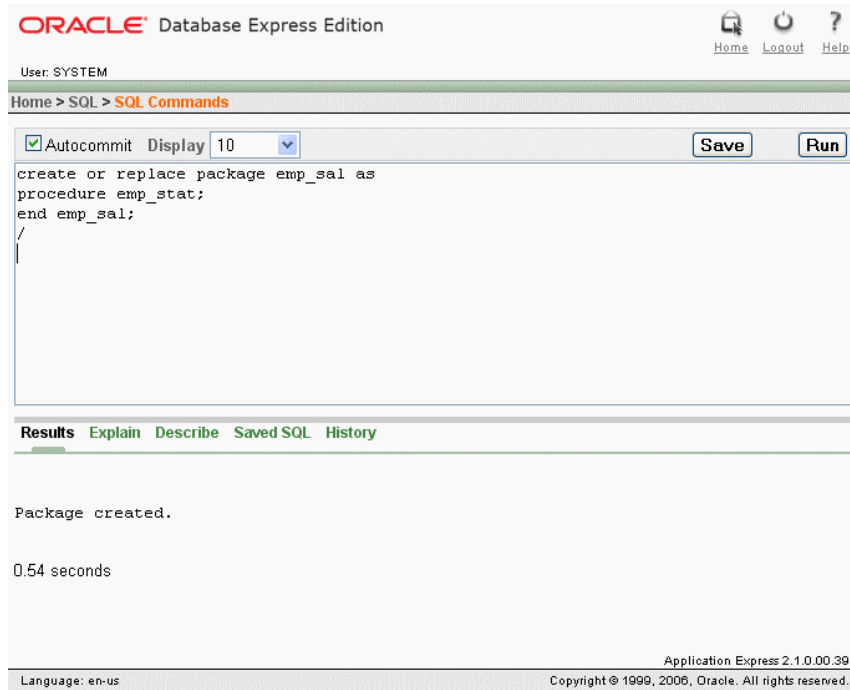


Figure 4-11 Oracle Application Express PL/SQL and SQL command screen.

Click **Run** to execute the PL/SQL. Then enter the following code:

```

create or replace package body emp_sal as
Procedure emp_stat is
TYPE EmpStatTyp is record (Dept_name varchar2(20),
Dept_avg number);
EmpStatVar EmpStatTyp;
BEGIN
DBMS_OUTPUT.PUT_LINE('Department          Avg Salary');
DBMS_OUTPUT.PUT_LINE('-----          -----');
For EmpStatVar in (select round(avg(e.salary),2)
                    a,d.department_name b
                    from departments d, employees e
                    where d.department_id=e.department_id
                    group by d.department_name)
LOOP
DBMS_OUTPUT.PUT_LINE(RPAD(EmpStatVar.b,16,' ')||
TO_CHAR(EmpStatVar.a,'999,999,999.99'));
END LOOP;

END;

end emp_sal;

```

Click **Run**. The PL/SQL code is executed.

4. You can execute the stored procedure by entering the following PL/SQL code and clicking **Run**.

```
begin
```

```
emp_sal.emp_stat;  
end;
```

5. If you want to save the PL/SQL code for future use, click **Save**.

## Creating a Database User

The Administration console is used to manage database:

- \* Storage
- \* Memory
- \* Users
- \* Activity of sessions and operations

The installation process creates an account named *system*. This account is considered an administrator account because it has DBA privileges (SYSDBA). To perform database administration such as creating new users, log into Oracle Application Express as the *system* user.

To create a database user:

1. On the Database home page, click the **Administration** icon, then click the **Database Users** icon.

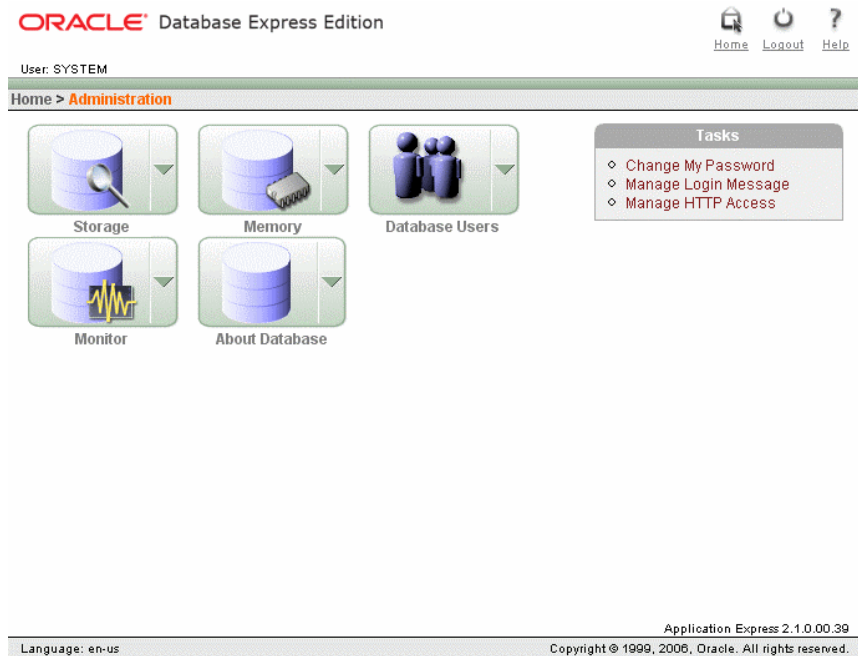
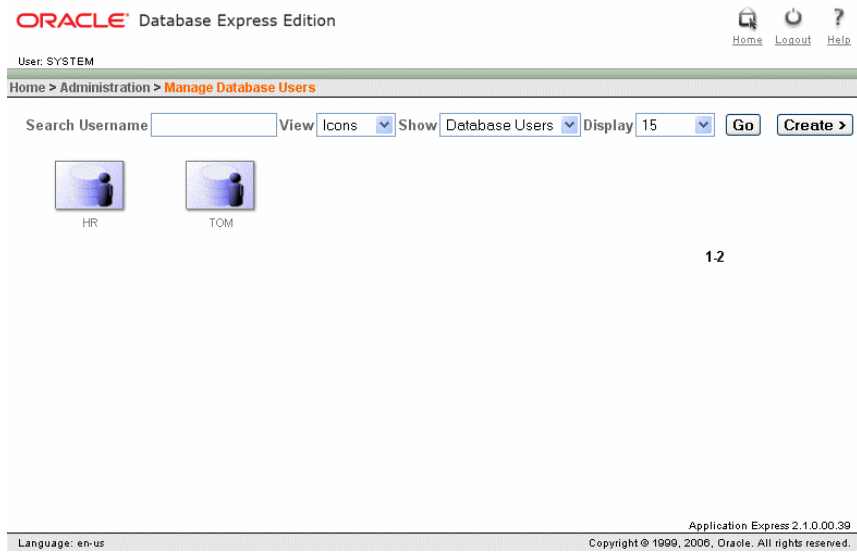


Figure 4-12 Oracle Application Express Administration screen.



1-2

Figure 4-13 Oracle Application Express Manage Database Users screen.

2. On the Manage Database Users page, click **Create**. The Create Database User page is displayed.

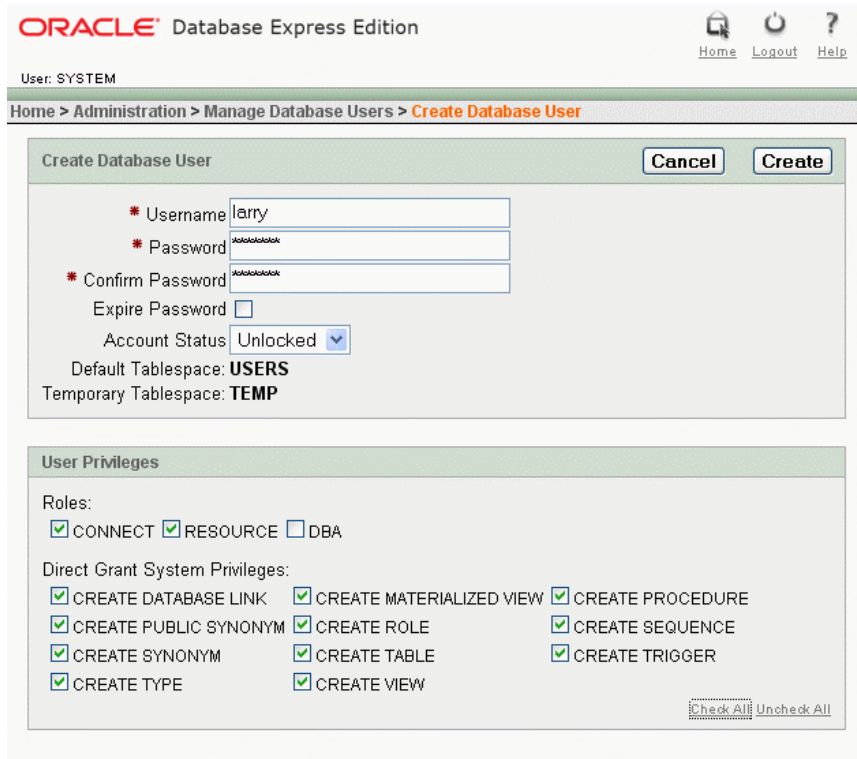
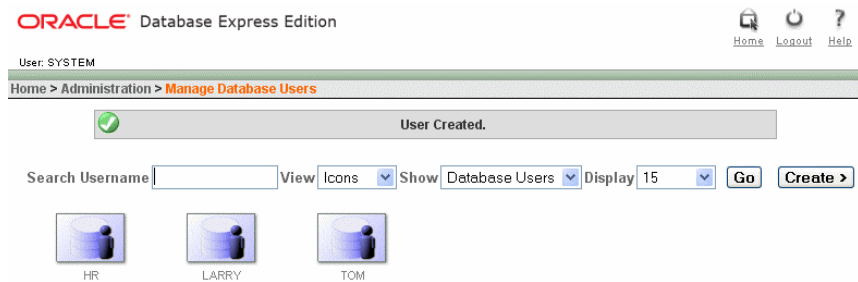


Figure 4-14 Oracle Application Express Create Database User screen.

3. Enter user information into text fields:
  - 3.1 In the Username field, enter a username.

- 3.2 In the Password and Confirm Password fields, enter a password.
  - 3.3 Grant all create object system privileges by clicking **Check All** at the lower right-hand corner of the User Privileges box.
  - 3.4 The DBA role is by default not selected. The DBA privilege, gives the user the ability to create schema objects in other users' schemas, and to create other users.
4. Click **Create**. The Manage Database Users page displays a confirmation that the user was created.



13

Figure 4-15 Oracle Application Express Manage Database Users screen.

## Monitoring Database Sessions

You can use the Oracle Database XE graphical user interface to monitor the current database sessions. This enables you to determine the users who are currently logged in to the database and what applications they are running.

You can also use the Oracle Database XE graphical user interface to kill a session—to cause it to be disconnected and its resources to be relinquished.

When you view sessions, you can view:

- \* All sessions
- \* Active sessions only
- \* Sessions that match a search string

You should be logged in as the *system* user to perform any database administration. To view the current sessions:



1. On the Database home page, click the **Administration** icon, then click **Monitor**.

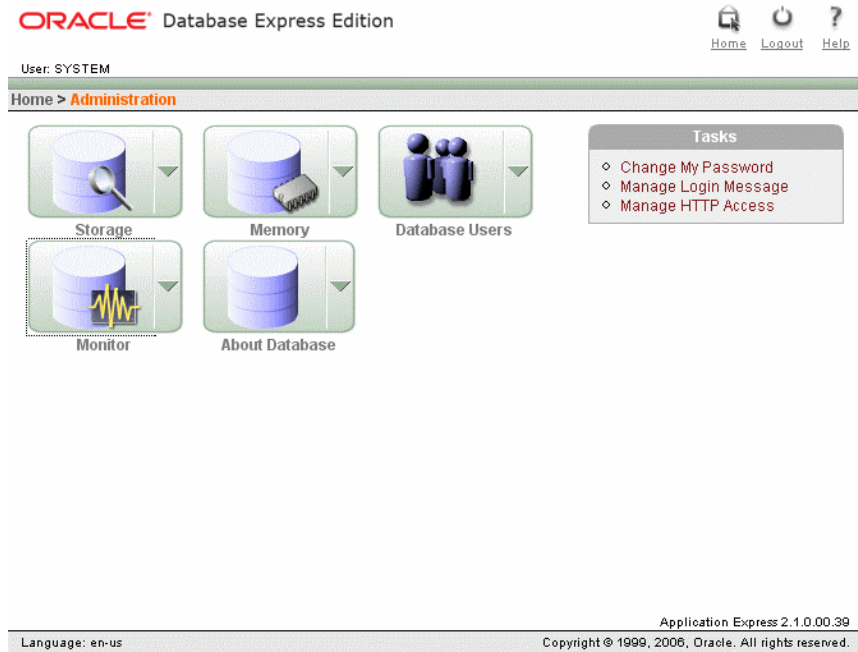


Figure 4-16 Oracle Application Express Administration screen.

3. On the Database Monitor page, click **Sessions**.

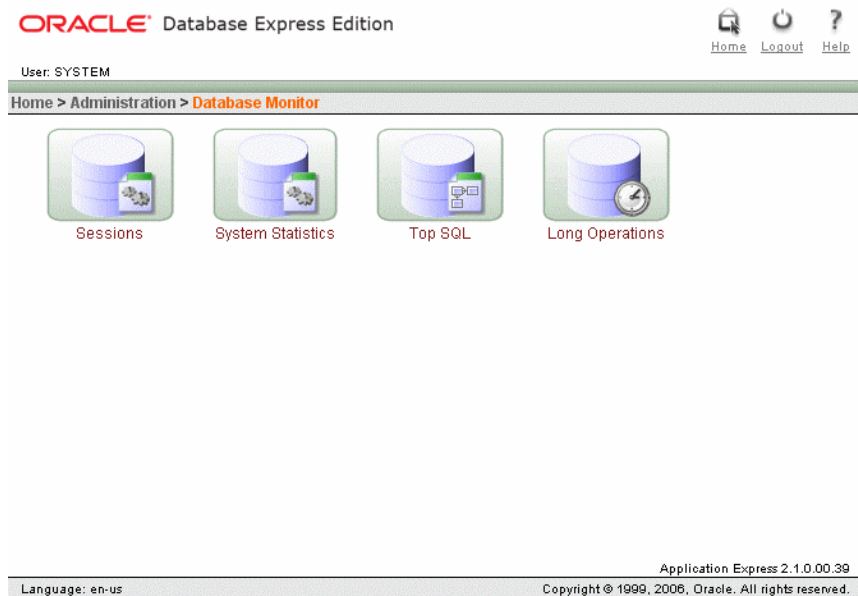
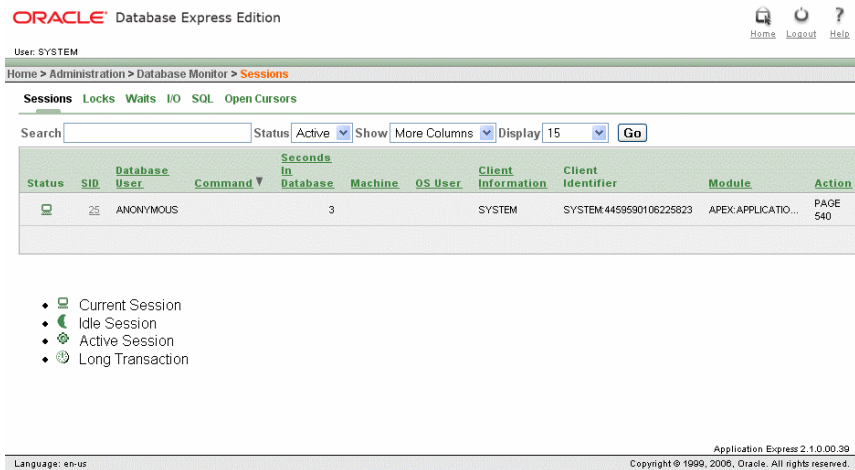


Figure 4-17 Oracle Application Express Administration Monitor screen.

The Sessions page is displayed and shows the current active sessions.



ORACLE Database Express Edition

User: SYSTEM

Home > Administration > Database Monitor > Sessions

Sessions Locks Waits I/O SQL Open Cursors

Search:  Status: Active Show: More Columns Display: 15 Go

Status	SID	Database User	Command	Seconds In Database	Machine	OS User	Client Information	Client Identifier	Module	Action
	25	ANONYMOUS		3		SYSTEM		SYSTEM:4459590106225823	APEX:APPLICATIO...	PAGE 540

- Current Session
- Idle Session
- Active Session
- Long Transaction

Application Express 2.1.10.00.39  
Language: en-us Copyright © 1999, 2006, Oracle. All rights reserved.

Figure 4-18 Oracle Application Express Sessions screen.

- (Optional) In the Status list, select **All**, and then click **Go**. The page displays all sessions, including idle sessions. (An example of an idle session is a SQL\*Plus command line session that is not currently running a command.)
- (Optional) Narrow down the sessions list by entering search text into the **Search** field and clicking **Go**. A session is shown if any of the following fields contain the search text: SID, Database User, Machine, OS User, Client Information, Client Identifier, and Module.
- (Optional) Click any of the hyperlinks above the Search field to view the following information for all sessions: Locks, Waits, Input/Output (I/O), running SQL statements, and open cursors.
- (Optional) Under the SID column, click a session ID to view the Session Details page for that session. The Session Details page enables you to kill the session.

## Database Backup and Recovery

Backing up and restoring Oracle Database XE is based on protecting the physical files that make up the database: the datafiles, the control file, the server parameter file (SPFILE), and, if in ARCHIVELOG mode, the redo log files.

In Oracle Database XE, database backup and recovery is handled by Recovery Manager (RMAN). Oracle Database XE includes backup and restore scripts that you access using menu on your desktop. These scripts perform a full backup and restore of the entire database, and store backup files in the *flash recovery area*.

Oracle Database XE implements a backup retention policy that dictates that two complete backups of the database must be retained. In ARCHIVELOG mode, all archived logs required for media recovery from either backup are also retained. The database automatically manages backups and archived logs in the flash recovery area, and deletes obsolete backups and archived logs at the end of each backup job.

The backup script performs online backups of databases in ARCHIVELOG mode and offline backups of databases in NOARCHIVELOG mode. Online backups are backups

that can run while the database is running. Offline backups are backups that run when the database is shut down.

The restore script restores the database differently depending on whether log archiving is on or off.

Log archiving on (ARCHIVELOG mode) restores the backed up database files, and then uses the online and archived redo log files to recover the database to the state it was in before the software or media failure occurred. All committed transactions that took place after the last backup are recovered, and any uncommitted transactions that were under way when the failure took place are rolled back (using undo data from the restored undo tablespace).

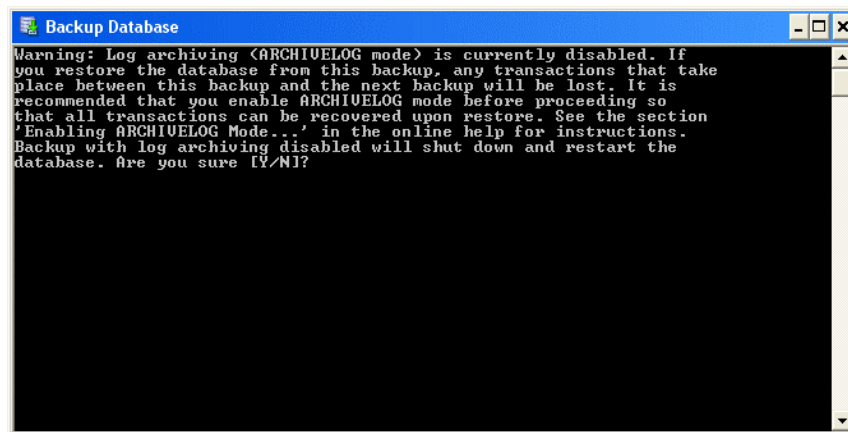
Log archiving off (NOARCHIVELOG mode) restores the database to its state at the last backup. Any transactions that took place after the last backup are lost.

## Backing Up The Database

To back up the database:

1. Log in to the Oracle Database XE host computer as a user who is a member of the *dba* user group.
2. Do one of the following:
  - On Linux with Gnome, select **Applications > Oracle Database 10g Express Edition > Backup Database**.
  - On Linux with KDE, select **K Menu > Oracle Database 10g Express Edition > Backup Database**.
  - On Windows, select **Start > Programs > Oracle Database 10g Express Edition > Backup Database**.

A console window opens so that you can interact with the backup script. If running in ARCHIVELOG mode, the script displays the following output:

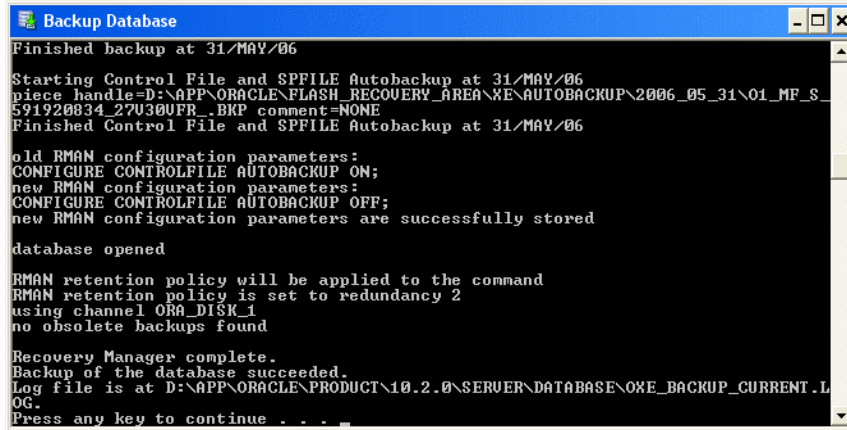


```

Warning: Log archiving (ARCHIVELOG mode) is currently disabled. If
you restore the database from this backup, any transactions that take
place between this backup and the next backup will be lost. It is
recommended that you enable ARCHIVELOG mode before proceeding so
that all transactions can be recovered upon restore. See the section
'Enabling ARCHIVELOG Mode..' in the online help for instructions.
Backup with log archiving disabled will shut down and restart the
database. Are you sure [Y/N]?
  
```

Figure 4-19 Oracle Application Express backup dialog

3. If prompted, answer **y** to confirm the database shutdown and begin the backup. After the backup is complete, the script displays the following output:



```

Backup Database
Finished backup at 31/MAY/06

Starting Control File and SPFILE Autobackup at 31/MAY/06
piece handle=D:\APP\ORACLE\FLASH_RECOVERY_AREA\XE\AUTOBACKUP\2006_05_31\01_MF_S
591920034_270300FR_BKP comment=NONE
Finished Control File and SPFILE Autobackup at 31/MAY/06

old RMAN configuration parameters:
CONFIGURE CONTROLFILE AUTOBACKUP ON;
new RMAN configuration parameters:
CONFIGURE CONTROLFILE AUTOBACKUP OFF;
new RMAN configuration parameters are successfully stored

database opened

RMAN retention policy will be applied to the command
RMAN retention policy is set to redundancy 2
using channel ORA_DISK_1
no obsolete backups found

Recovery Manager complete.
Backup of the database succeeded.
Log file is at D:\APP\ORACLE\PRODUCT\10.2.0\SERVER\DATABASE\XE_BACKUP_CURRENT.L
OG.
Press any key to continue . . .

```

Figure 4-20 Oracle Application Express backup successful message

4. Press any key to close the Backup Database window.

Logs containing the output from the backups are stored in the following locations:

```

$ORACLE_HOME/oxe_backup_current.log
$ORACLE_HOME/oxe_backup_previous.log

```

## Restoring The Database

To restore a database from a backup:

1. Log in to the Oracle Database XE host computer as a user who is a member of the *dba* user group.
2. Do one of the following:
  - On Linux with Gnome, select **Applications > Oracle Database 10g Express Edition > Restore Database**.
  - On Linux with KDE, select **K Menu > Oracle Database 10g Express Edition > Restore Database**.
  - On Windows, select **Start > Programs > Oracle Database 10g Express Edition > Restore Database**.

A console window opens so that you can interact with the restore script. The script displays the following output:

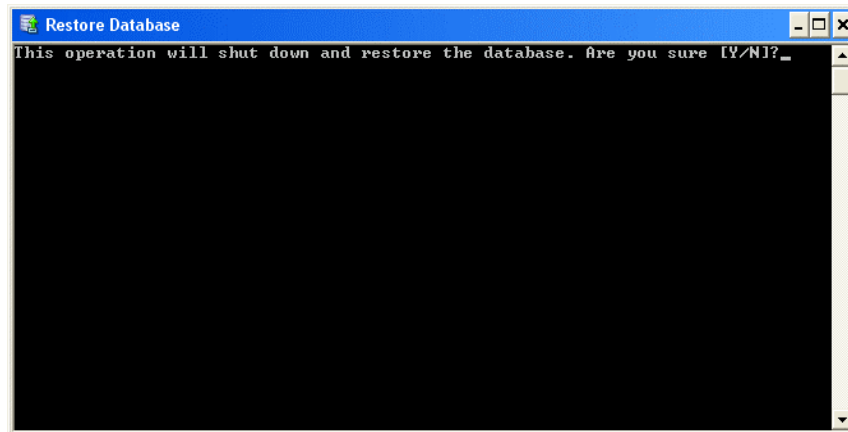


Figure 4-21 Oracle Application Express restore dialog

3. Answer **y** to confirm the database shutdown and begin the restore. After the restore is complete, the script displays the following output:

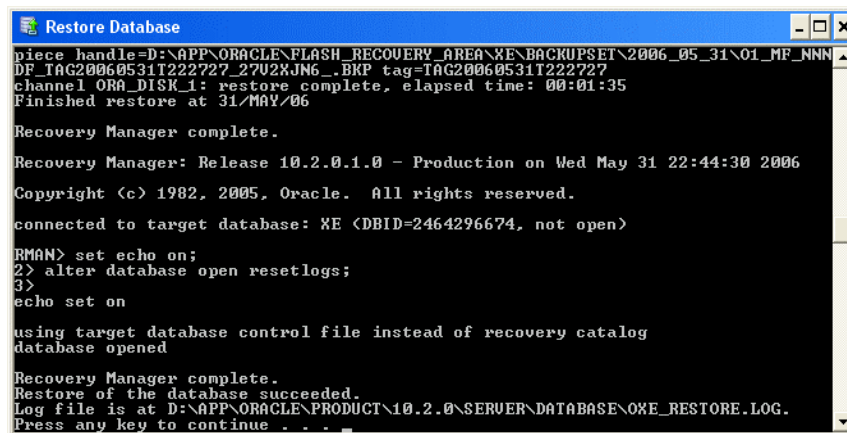


Figure 4-22 Oracle Application Express restore successful message

4. Press any key to close the Restore Database window.

A log containing the output from the restore is stored in `$ORACLE_HOME/oxe_restore.log`.

## Oracle SQL Developer

In addition to Oracle Application Express, you can also use Oracle SQL Developer for database development and maintenance. Oracle SQL Developer is a free, thick client graphical tool for database development. You can use SQL Developer to execute SQL statements, execute and debug PL/SQL statements, and to run a few SQL\*Plus commands (like DESCRIBE). SQL Developer includes some prebuilt reports, and you can create and save your own reports.

SQL Developer can connect to Oracle databases from version 9.2.0.1 onwards, and is available on Linux, Windows and Mac OSX.

You can download SQL Developer from the Oracle Technology Network at [http://www.oracle.com/technology/products/database/sql\\_developer](http://www.oracle.com/technology/products/database/sql_developer). You can also download patches, extensions, documentation, and other resources from this site. There's also a discussion forum for you to ask questions of other users, and give feedback to the SQL Developer product team.

## Creating a Database Connection

When you start SQL Developer for the first time, there are no database connections configured, so the first thing you need to do is create one. The default SQL Developer screen is shown in figure 4-23.

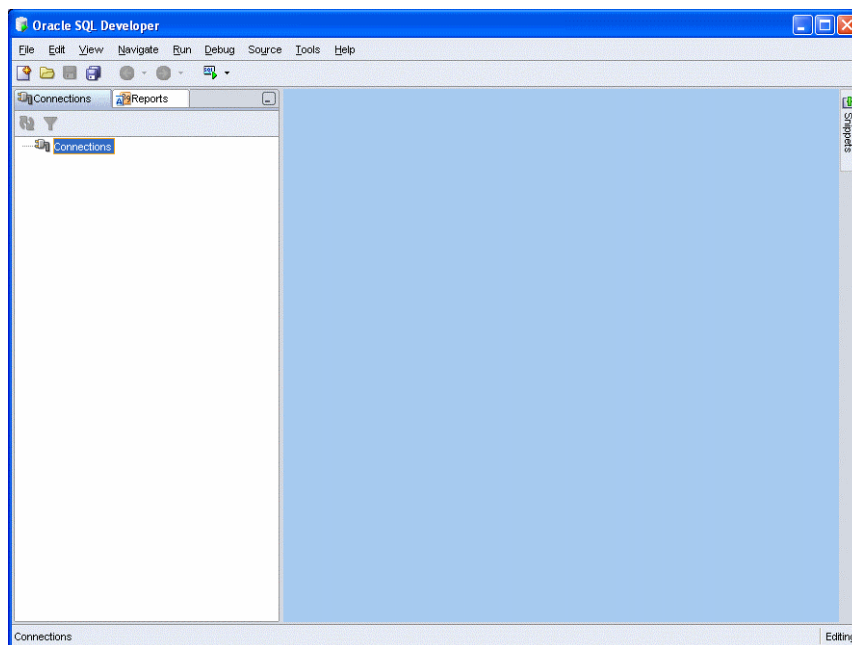


Figure 4-23 Oracle SQL Developer login screen

To create a database connection to the local Oracle Database Express Edition (Oracle XE) database:

1. Select **Connections** in the left pane, right click and select **New Database Connection**. Enter the login credentials for the Oracle XE database as **Username** *hr*, **Password** *hr*, **Hostname** *localhost*, and **SID** *XE*.

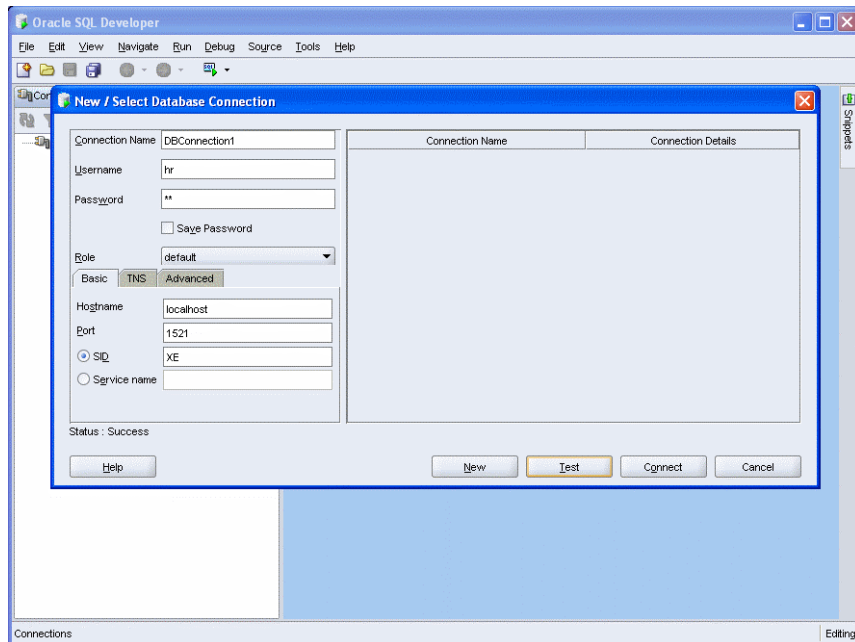


Figure 4-24 Oracle SQL Developer Connection screen

2. Click the **Test** button to test the connection. A message is displayed at the bottom left side of the dialog to tell you whether the test connection succeeded. Click the **Connect** button to save the connection and connect to it.

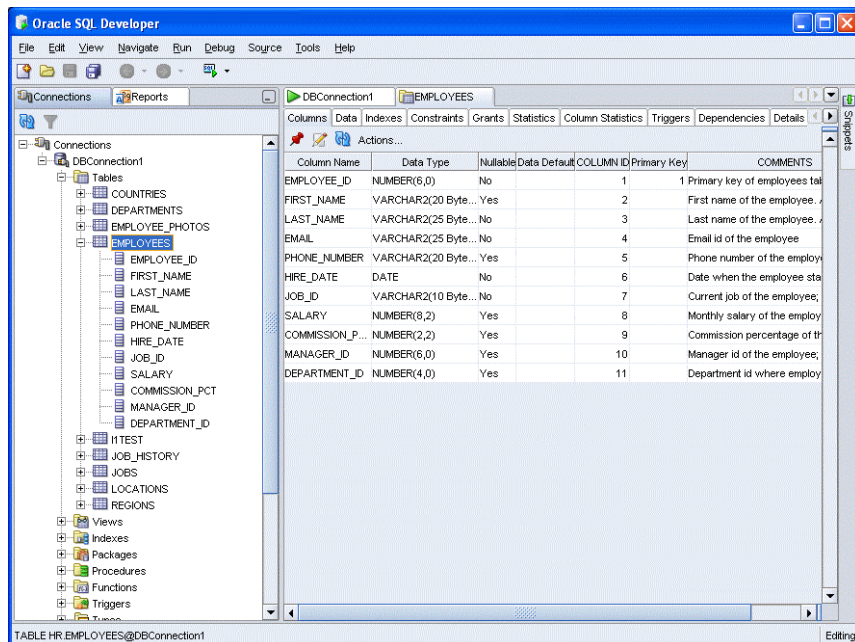


Figure 4-25 Oracle SQL Developer main screen

When you have connected to a database, you can browse through the database objects displayed in the left pane, and the right pane shows the contents of the object. In Figure 4-25, the Employees table is displayed.

## Creating a Table

You can create database objects such as tables, views, indexes, and PL/SQL procedures using SQL Developer. To create a database object, right click on the database object type you want to create, and follow the dialogs. You can use this method to create any of the database objects displayed in the left pane. For example, to create a new table:

1. Select **Tables** in the left pane, right click and select **Create Table**. The Create Table dialog is displayed. Enter the column names, types and other parameters as required.

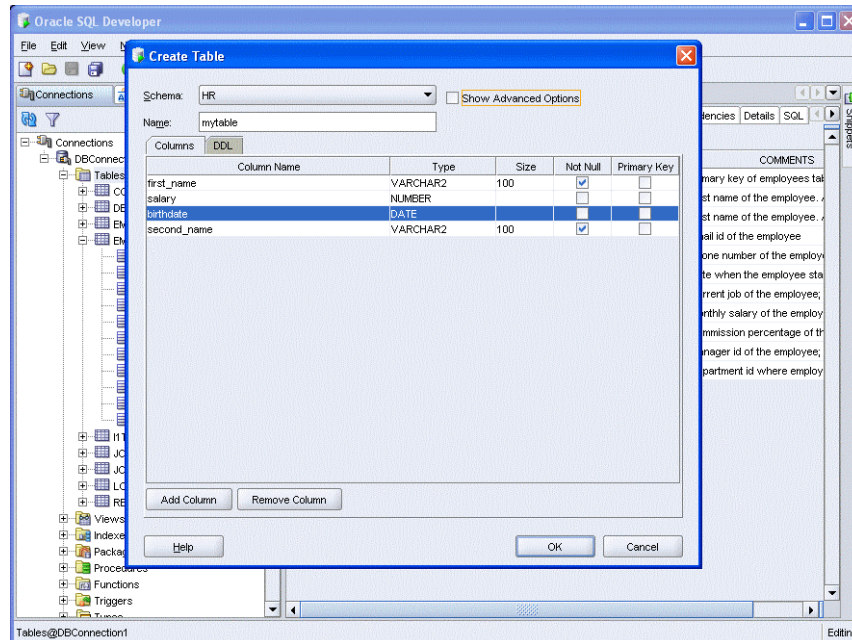


Figure 4-26 Oracle SQL Developer Create Table screen

Click the **Show Advanced Options** check box to see more advanced options like constraints, indexes, and foreign keys.



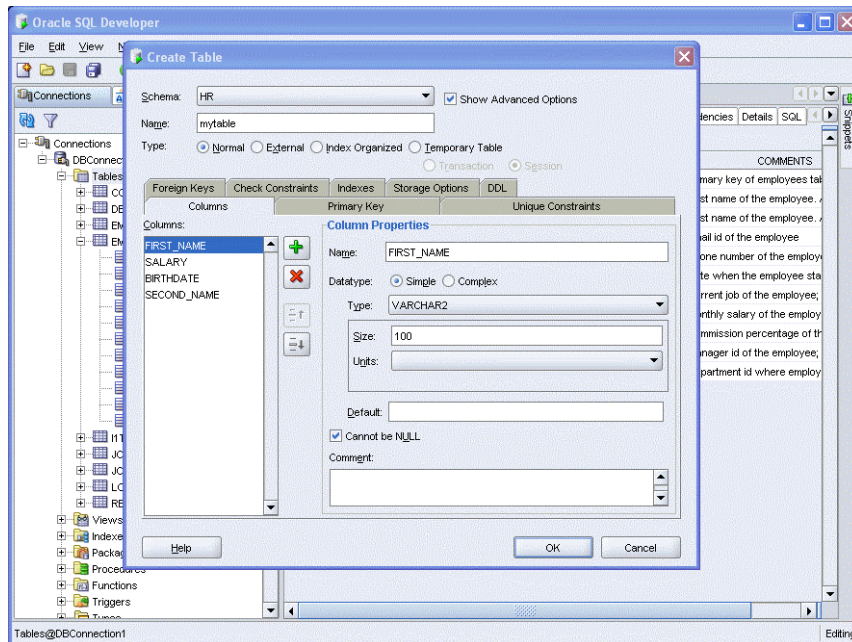


Figure 4-27 Oracle SQL Developer Advanced Create Table screen

2. Click **OK** to create the table. The new table *mytable* is now listed in the left pane.

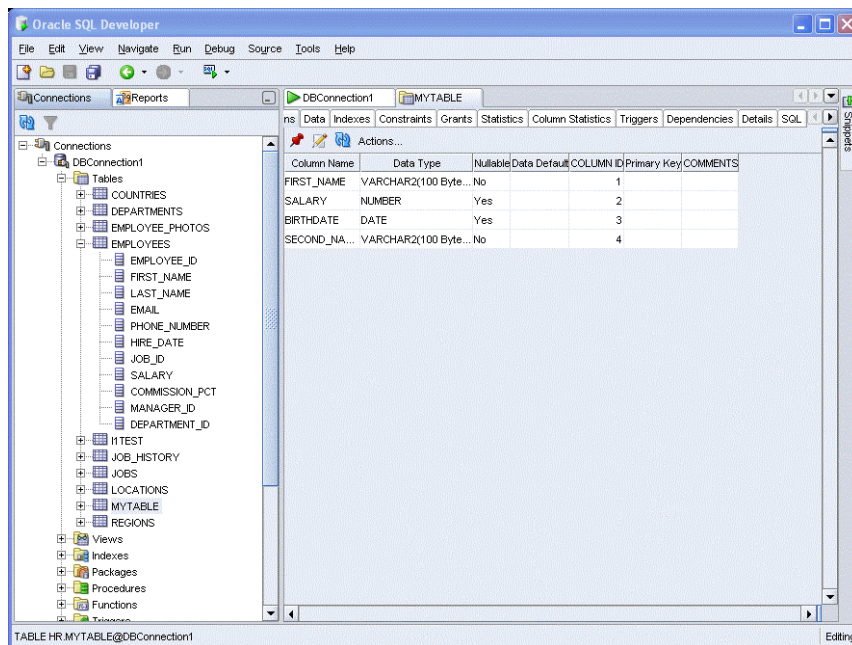


Figure 4-28 Oracle SQL Developer screen showing the new table, mytable

Click on the tabs displayed in the right pane to see the options available on the table, such as the Data tab, which enables you to add, delete, modify, sort, and filter rows in the table.

## Executing a SQL Query

The SQL Worksheet component included in SQL Developer can be used to execute SQL and PL/SQL statements. Some SQL\*Plus commands can also be executed. To execute a SQL statement in SQL Developer:

1. Select the **DBConnection1** tab in the right hand pane. This is the connection created earlier in *Creating a Database Connection*. The SQL Worksheet component is displayed, and shows an area to enter statements, and a set of tabs below that for further options.

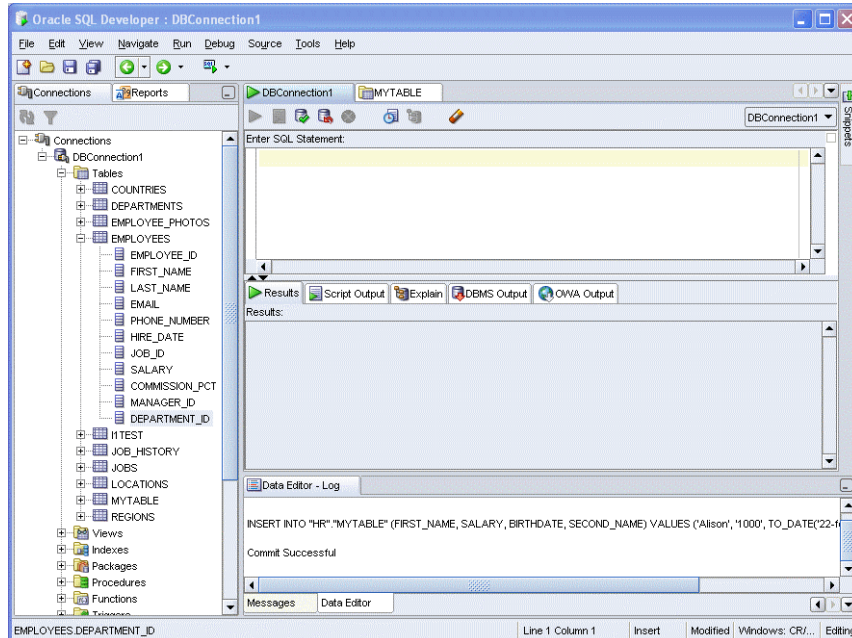


Figure 4-29 Oracle SQL Developer screen showing SQL Worksheet

2. Enter the following two statements in the SQL Worksheet:

```
DESCRIBE HR.EMPLOYEES
SELECT * FROM HR.EMPLOYEES ;
```

Click the **Run Script** icon (the second from the left in the right hand pane), or press **F5**. Both the lines of this script are run and the output is displayed in tabs below. You can view the output of the SELECT statement in the **Results** tab, and the output of the whole script (including the DESCRIBE and SELECT statements) in the **Script Output** tab. If you have used Oracle's command line tool SQL\*Plus, the output in the Script Output window is likely to be familiar to you.

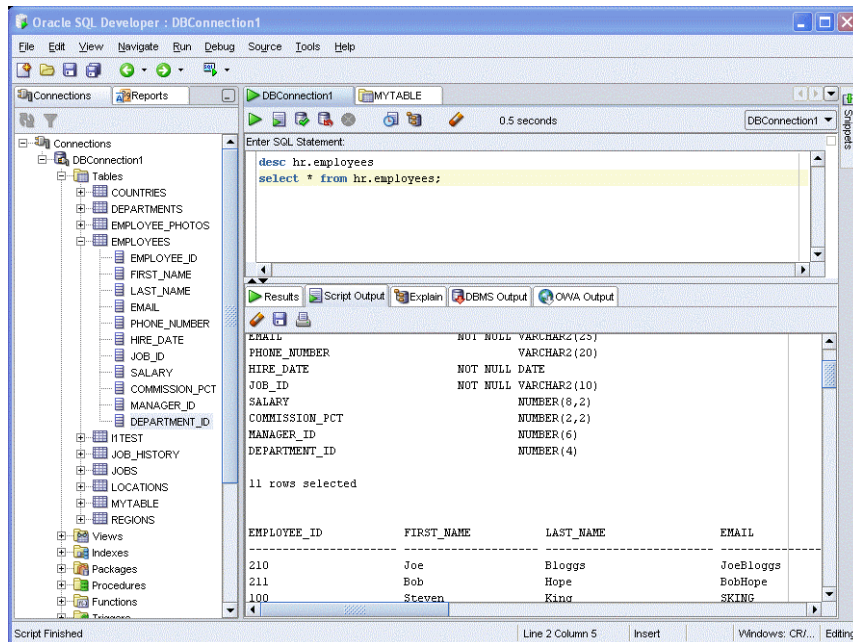


Figure 4-30 Oracle SQL Developer screen showing SQL Worksheet with output

3. If you want to execute a single line of the two-line statement, select the line you want to execute and click on the **Execute Statement** icon (the first from the left in the right hand pane), or press **F9**. In this case, the SELECT statement (second line) is selected, and the results are shown in the Results tab.

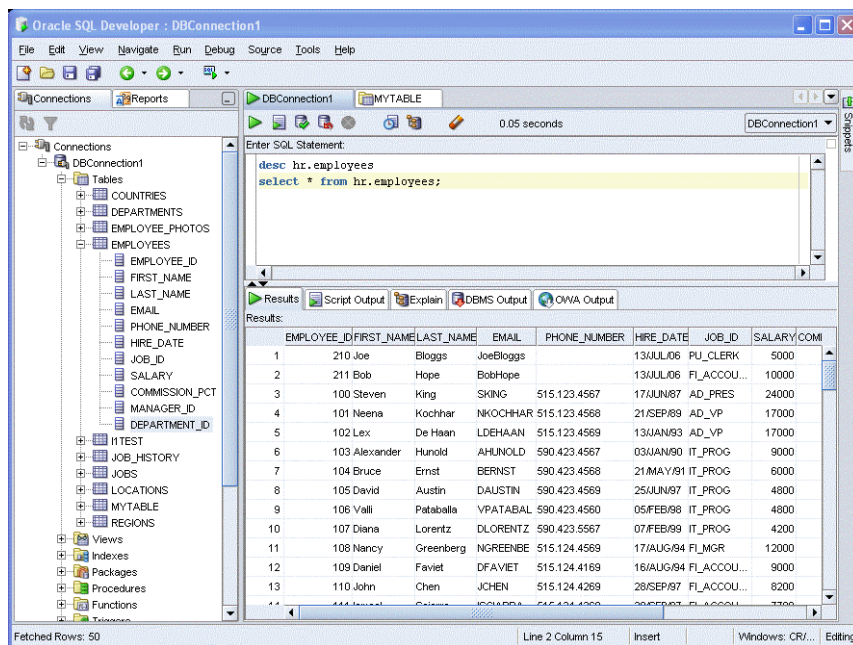


Figure 4-31 Oracle SQL Developer screen showing SQL Worksheet with output

You should also take a moment to click on the **Snippets** on the top right hand of the interface. Snippets are a handy set of SQL and PL/SQL code snippets that you can drag into the SQL Worksheet component.

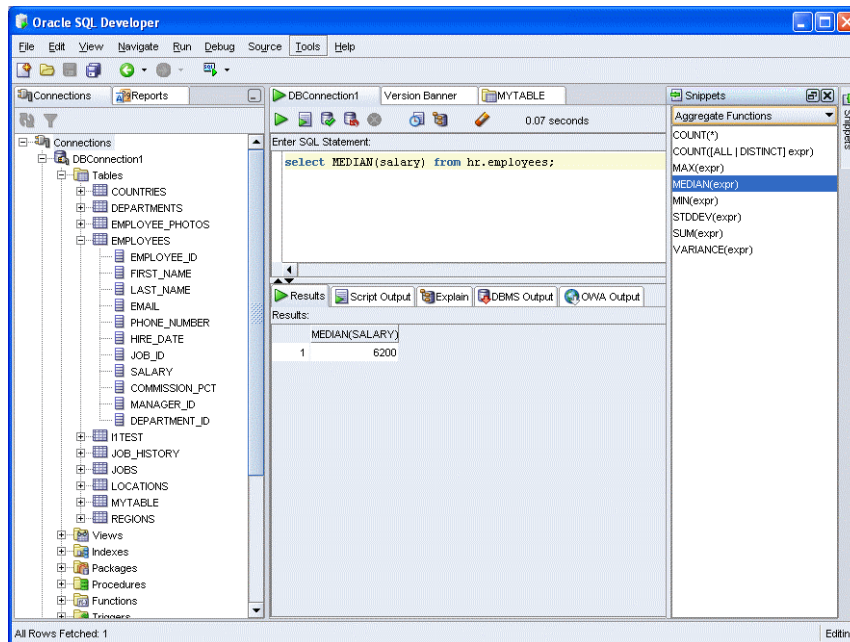


Figure 4-32 Oracle SQL Developer screen showing code snippets

## Running Reports

There are a number of reports included with SQL Developer that may be useful to you, for example, getting lists of all users in a database, all the tables owned by a user, or all the views available in the data dictionary.

You can also create your own reports (using SQL and PL/SQL), and include them in SQL Developer. To display the version numbers of the database components using one of the supplied reports:

1. Select the **Reports** tab in the left hand pane. Navigate down to **Reports > Data Dictionary Reports > About Your Database > Version Banner**. The report is run and the results are displayed in the right hand pane.

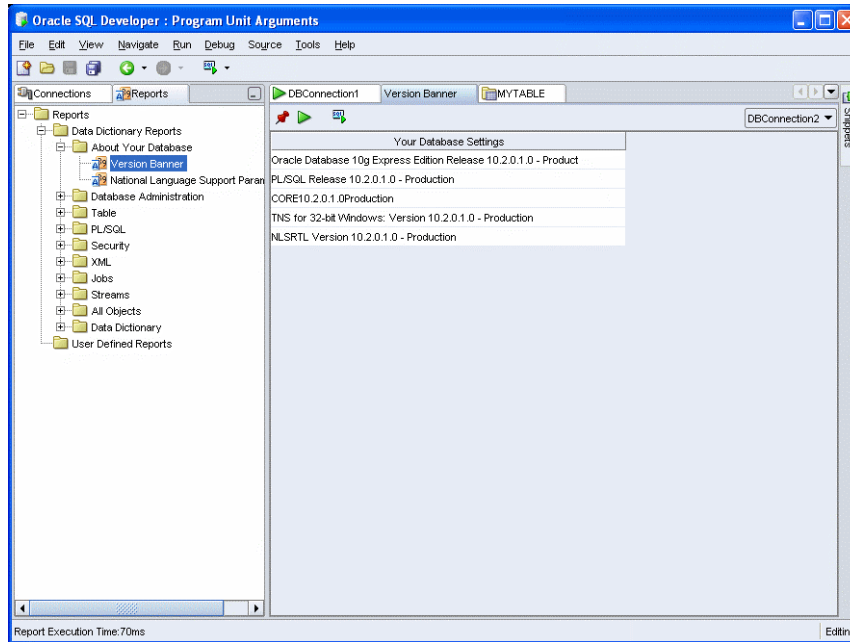


Figure 4-33 Oracle SQL Developer screen showing output from a report

2. Right click on the **Version Banner** and select **Show Properties** to see the source code that is actually executed.

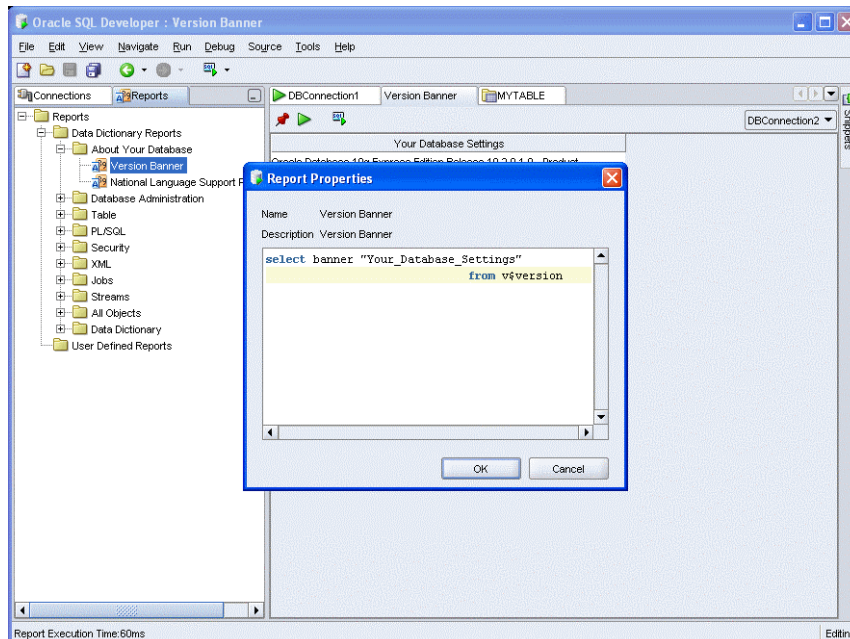


Figure 4-34 Oracle SQL Developer screen showing the source code of a report

To create your own reports, select **User Defined Reports**, and right click. A list of options for creating, editing and running your own reports is displayed.

## CHAPTER 5

# Installing Apache HTTP Server

This chapter discusses installing Apache HTTP Server on Linux and Windows. The installation was done using Red Hat Enterprise Linux AS 4.0, and Windows XP Professional Edition and uses Apache HTTP Server Release 2.0.58 available from <http://httpd.apache.org/download.cgi>.

## Installing Apache HTTP Server on Linux

The following procedure describes how to install the Apache HTTP Server on Linux.

1. Download the Apache HTTP Server from <http://httpd.apache.org/download.cgi>. The version used in this installation is *httpd-2.0.58.tar.bz2*.
2. Log in as the *root* user and execute the following commands:

```
# tar -jxvf httpd-2.0.58.tar.bz2
# cd httpd-2.0.58
# ./configure --prefix=/usr/local/apache --enable-module=so
# make
# make install
```

When configuring the web server, the option *--enable-module=so* allows PHP to be compiled as a Dynamic Shared Object (DSO). Also, the *--prefix=* option sets where Apache HTTP Server will be installed during the command "*make install*"

If you are familiar with the tar command on UNIX systems, you may be wondering why we did not need to invoke *bunzip2* to extract the tar file. Linux includes the GNU version of tar, which has a new 'j' flag to automatically uncompress a bziped tar file. If you downloaded the gzipped file, you could have used the 'z' flag instead.

---

Note: With Apache 2, you should use the default pre-fork MPM ("Multi-Processing Module"), because many of the PHP libraries are not known to be thread-safe.

---

## Starting and Stopping Apache HTTP Server

The *apachectl* script is installed in the Apache 'bin' directory. Use this script to start and stop Apache HTTP Server. To start Apache HTTP Server:

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

## Installing Apache HTTP Server

You should test that Apache has been installed properly and is started on your machine by opening your web browser to <http://localhost/> to display the Apache home page.

Now stop Apache so it can be configured for PHP:

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

---

Note: If you are using Oracle 10g Release 10.2, but not the Express Edition, you must give the “nobody” user access to the Oracle directory. With Oracle 10g Release 10.2.0.2, there is a script located in `$ORACLE_HOME/install/changePerm.sh` to do this.

---

If there are errors, they will display on your screen. Errors may also be recorded in `/usr/local/apache/logs/error_log`. If you have problems, check your `httpd.conf` and `php.ini` configuration files, and make corrections.

When you start Apache HTTP Server, you must at least have the environment variable `$ORACLE_HOME` defined. Any other required Oracle environment variables must be set before Apache HTTP Server starts as well. These are the same variables set by the `$ORACLE_HOME/bin/oracle_env.sh` or the `/usr/local/bin/oraenv` scripts.

To simplify things, you may create a script to start Apache HTTP Server, for example:

```
start_apache
```

```
#!/bin/sh
```

```
ORACLE_HOME=/usr/lib/oracle/xe/app/oracle/product/10.2.0/server
export ORACLE_HOME
echo "Oracle Home: $ORACLE_HOME"
echo Starting Apache
/usr/local/apache/bin/apachectl start
```

## Installing Apache HTTP Server on Windows

The following procedure describes how to install the Apache HTTP Server on Windows.

1. Download the Apache HTTP Server Windows binaries from <http://www.apache.org/dist/httpd/binaries/win32/>. The release used in this installation is Apache HTTP Server 2.0.58, and the downloaded file is named `apache_2.0.58-win32-x86-no_ssl.msi`.
2. Double click the downloaded file `apache_2.0.58-win32-x86-no_ssl.msi`.
3. Follow the Apache HTTP Server installation wizards. You should select to install Apache HTTP Server “for All Users, on Port 80”, because the “only for the Current User” alternative will clash with Oracle Database Express Edition’s default port 8080.



## Starting and Stopping Apache HTTP Server

As part of installation, the Apache HTTP Server is started. You should now test that Apache HTTP Server has been installed properly and started on your machine by opening your web browser to <http://localhost/> to display the Apache home page.

Your system tray has an Apache Monitor control that makes it easy to stop and restart the Apache HTTP Server when needed. Alternatively, use the Apache options added to your Windows Start menu.



## CHAPTER 6

# Installing Zend Core for Oracle

## Zend Core for Oracle

Zend Core for Oracle is a fully tested and supported PHP 5 distribution that includes integration with Oracle Database 10g client libraries. It is a pre-built stack for that makes it easier to get started with PHP and Oracle, as all the hard work of installation and configuration has been done for you. Support for Zend Core for Oracle, and for PHP is provided by Zend.

There is a PHP Developer Center on the Oracle Technology Network (<http://otn.oracle.com/php>) dedicated to PHP development with Oracle. You will find a lot of useful information on this site, and some of the information in this book has been reworked and included in this book.

The collaboration between Oracle and Zend reinforces Oracle's commitment to the open source PHP community. Oracle's newly introduced and optimized OCI8 extension for PHP will be submitted back to the PHP community and integrated into Zend Core for Oracle.

Zend Core for Oracle is supported (by Zend) on the following operating systems:

- \* X86 running SLES9 or RHEL3
- \* X86 running Windows XP/2003/2000
- \* X86-64 running SLES9 or RHEL3
- \* pSeries running AIX 5.2 or 5.3

The web servers that are supported are:

- \* Apache 1.3.x
- \* Apache 2.0.x (compiled in prefork mode only)
- \* Oracle HTTP Server 10.2.0
- \* Microsoft IIS 5/6

## Installing Zend Core for Oracle

This procedure installs Zend Core for Oracle on Linux. The procedure to install on other operating systems is similar as the installer is the same on all operating systems. There are some slight differences, like the names and locations of the web server, but the installation is almost the same on all platforms.

To install Zend Core for Oracle on Linux:

## Installing Zend Core for Oracle

1. Download the Zend Core for Oracle file from the Oracle Technology Network (<http://otn.oracle.com/php>).
2. Log in or *su* as *root* if you are not already. If you are on Windows, make sure you are logged in with administrator privileges.

```
su
Password:
```

3. Extract the contents of the downloaded Zend Core for Oracle software:

```
tar -zxf ZendCoreForOracle-v1.3.1-Linux-x86.tar.gz
```

Files are extracted to a subdirectory called *ZendCoreForOracle-v1.3.1-Linux-x86*.

4. Change directory to *ZendCoreForOracle-v1.3.1-Linux-x86* and start the Zend Core for Oracle installation:

```
cd ZendCoreForOracle-v1.3.1-Linux-x86
./install
```

5. In the initial Zend Core for Oracle Installation page, click **OK**.
6. In the Zend Core for Oracle V.1 page, click **Exit**.
7. When prompted to accept the terms of the license, click **Yes**.
8. If you have an existing version of PHP installed, you are prompted to back up and overwrite the existing *php.ini* file. Click **Yes**.
9. When prompted to specify the location for installing Zend Core for Oracle, accept the default (or enter your preferred location), and click **OK**. The installer begins extracting the files required for the installation.
10. When the progress window indicates that all the software has been installed, you are prompted to "Please enter the GUI password." In the Password field, enter the password you want to use when accessing the Zend Core Console, and click **OK**.
11. When prompted to "Verify the password," enter the same password as specified in the previous step and click **OK**.
12. In the Zend Core support page, you may optionally enter your Zend network user ID and password to be able to use the Zend Core Console to track when updates to Zend Core and PHP components are available. If you have not registered, or do not want to track updates, click **No**.
13. The next page prompts you to select the web server for Zend Core installation. Select the **default Apache installed with Linux**. Click **OK**.
14. In the page confirming your web server selection, at the "Do you wish to proceed?" prompt, click **Yes**.

15. In the next installation page, you are prompted to "Please select an installation method for Apache 2.0.52." Select **Apache module** as the method, and click **OK**.
16. In the next installation page, when you are prompted to "Please select a virtual server for the Zend Core GUI," select **Main Server**, and click **OK**.
17. In the next installation page, at the "Would you like to restart the Web Server" prompt, click **Yes**.
18. In the next installation page, a notice is displayed stating that the apachectl script has been updated. Click **OK**.
19. The next installation page (containing "Thank you for installing Zend Core for Oracle") lists useful configuration commands and a web page for the administration of the Zend Core engine. Take note of the information and click **Exit**.
20. A final confirmation page is displayed. Click **OK** to finish the installation.

The Zend Core for Oracle installation is now complete.

## Configuring Zend Core for Oracle

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

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

`http://127.0.0.1/ZendCore`



Figure 6-1 Zend Core for Oracle login screen.

## Installing Zend Core for Oracle

2. Enter the GUI password that you provided during Zend Core for Oracle installation. Click the **login >>>** icon.
3. Click the **Configuration** tab to display the configuration options.
4. Click the + icon to expand the Error Handling and Logging configuration entry.
5. Set the **display\_errors** directive to **On** to enable the display of errors in the HTML script output during development.

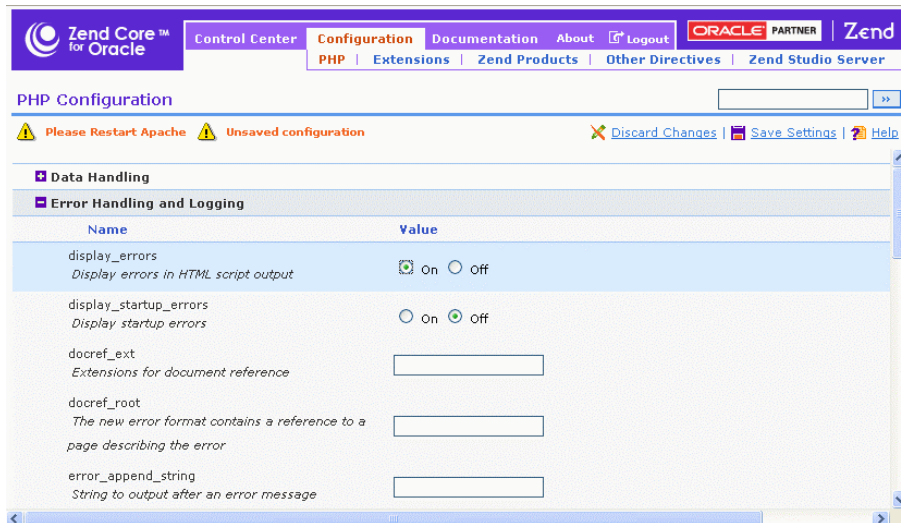


Figure 6-2 Zend Core for Oracle configuration screen.

6. Because there are unsaved changes, the "Unsaved configuration" message appears under the page header. Click **Save Settings** to save the configuration change.
7. Because you have made configuration changes, you must restart the Apache web server. Under the page header notice the "Please Restart Apache" message reminding you to do so. Click **Restart Server** to restart the Apache server. If you are using a Windows operating system, you should restart the Apache server using the Services dialog in Control Panel, or the Apache Monitor in the system tray.
8. Click **Logout** to exit the Zend Core for Oracle Administration page.

## Testing the Zend Core for Oracle Installation

To test the Zend Core for Oracle installation:

1. Create a public virtual directory as *public\_html*. Edit `APACHE_HOME/conf/httpd.conf` and remove the "#" from the following line:  

```
#UserDir public_html
```
2. As your normal user (not root), create a directory called *public\_html* in your home directory, and change directory to the newly created directory, enter the following commands in a command window:

```
cd $HOME
mkdir public_html
cd public_html
```

3. Create a file called *hello.php* that contains the following PHP code:

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

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

```
http://127.0.0.1/~<username>/hello.php
```

The line “Hello world!” appears in the browser.





## CHAPTER 7

# Installing PHP

This chapter discusses installing PHP as an Apache module on Linux and Windows. The installation was done using Red Hat Enterprise Linux AS 4.0, and Windows XP Professional Edition and uses PHP 5.1.3. PHP is available from <http://www.php.net/downloads/>.

## Installing PHP on Linux

If you don't want to compile PHP, use the pre-built Zend Core for Oracle. See the *Installing Zend Core for Oracle* chapter for information on Zend Core for Oracle. Otherwise, download the file *php-5.1.3.tar.bz2* from the PHP downloads page and install using the following steps:

1. Log in as the *root* user and execute these commands:

```
# tar -jxvf php-5.1.3.tar.bz2
# cd php-5.1.3
# export ORACLE_HOME=/usr/lib/oracle/xe/app/oracle/product/
10.2.0/server
# ./configure \
    --with-oci8=$ORACLE_HOME \
    --with-apxs2=/usr/local/apache/bin/apxs \
    --with-config-file-path=/usr/local/apache/conf \
    --enable-sigchild
# make
# make install
```

2. Copy PHP's supplied initialization file, *php.ini*:

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

For testing it is helpful to edit *php.ini* and set *display\_errors* to *On* so you see any problems in your code.

3. Edit Apache's configuration file */usr/local/apache/conf/httpd.conf* and add the following lines:

```
#
# This section will call PHP for .php, .phtml, and .phps files
#
AddType application/x-httpd-php .php
AddType application/x-httpd-php .phtml
AddType application/x-httpd-php-source .phps
#
# This is the directory containing php.ini
#
PHPIniDir "/usr/local/apache/conf"
```

4. If a *LoadModule* line was not already inserted by the PHP install, add it too:

## Installing PHP

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

## Restart the Apache HTTP Server

You must now restart the Apache Server, so that you can test your PHP installation.

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

## Installing PHP on Windows

Download the PHP 5.1.3 Zip package (not the “installer” package, which does not contain the necessary extensions). You will notice that the installation instructions here are very similar to those found in the *install.txt* file contained within the PHP archive that you downloaded. Feel free to use that as a guide; the instructions here are just a subset of the information it contains.

The following procedure describes how to install PHP on Windows.

1. Uncompress the PHP package to a directory called *C:\php-5.1.3-Win32*.
2. Copy *php.ini-recommended* to *C:\Program Files\Apache Group\Apache2\conf\php.ini*.
3. Edit *php.ini* and perform the following:
  - 3.1 Change *extension\_dir* to “*C:\php-5.1.3-Win32\ext*”, which is the directory containing *php\_oci8.dll* and the other PHP extensions.
  - 3.2 Uncomment (remove the semicolon from the beginning of the line) the line *extension=php\_oci8.dll*.
4. For testing, it is helpful to set *display\_errors* to *On*, so you see any problems in your code.
5. Edit the file *httpd.conf* and add the following lines. Make sure to use forward slashes ‘/’ instead of back slashes ‘\’:

```
#
# This will load the PHP module into Apache
#
LoadModule php5_module c:/php-5.1.3-Win32/php5apache2.dll
#
# This section will call PHP for .php, .phtml, and .phps files
#
AddType application/x-httpd-php .php
AddType application/x-httpd-php .phtml
AddType application/x-httpd-php-source .phps

#
# This is the directory containing php.ini
#
PHPIniDir "C:/Program Files/Apache Group/Apache2/conf"
```

## Restart the Apache HTTP Server

You must now restart the Apache HTTP Server, so that you can test your PHP installation. Use the Start menu option to start Apache. This opens a console window showing any error messages. They may also be recorded in *C:\Program Files\Apache Group\Apache2\logs\error.log*. If you have errors, double check your *httpd.conf* and *php.ini* files, and correct any problems.



## CHAPTER 8

# Installing PHP With Oracle Instant Client

Oracle 10g Instant Client (free download available from the Oracle Technology Network) is the easiest way for PHP to connect to a remote Oracle database, requiring installation of only three libraries.

An existing Oracle database is needed in this installation scenario, as Instant Client does not include one. Typically, the database is on another machine. If the database is local, then Oracle components will generally already be available and Instant Client is not required.

This example uses Oracle Instant Client Release 10.1.0.3 available from <http://www.oracle.com/technology/tech/oci/instantclient/instantclient.html>.

PHP is available from <http://www.php.net/downloads/>.

## Installing Oracle Instant Client on Linux

After installing Apache HTTP Server, and PHP, you can install Oracle Instant Client. To install Oracle Instant client:

1. Download the Basic and SDK Instant Client packages from the Instant Client page on the Oracle Technology Network (<http://www.oracle.com/technology/tech/oci/instantclient/instantclient.html>). Collectively, the two packages are about 35MB in size.

2. If you are using the RPMs, install the RPMs as the *root* user:

```
rpm -Uvh oracle-instantclient-basic-10.1.0.3-1.i386.rpm
rpm -Uvh oracle-instantclient-devel-10.1.0.3-1.i386.rpm
```

The first RPM puts the Oracle libraries in `/usr/lib/oracle/10.1.0.3/client/lib` and the second creates headers in `/usr/include/oracle/10.1.0.3/client`.

If you are using the ZIP files, unzip the Oracle Instant Client and the SDK packages to a directory of your choice, for example `/home/instantclient10_1`.

3. If you have installed the Instant Client RPMs, configure PHP with:

```
./configure \
--with-oci8=instantclient,/usr/lib/oracle/10.1.0.3/client/lib \
--prefix=$HOME/php --with-apxs=$HOME/apache/bin/apxs \
--enable-sigchild --with-config-file-path=$HOME/apache/conf
```

If you are using the ZIP files then change the `--with-oci8` option to the unzipped directory:

```
--with-oci8=instantclient,$HOME/instantclient10_1
```

To find out whether you are using the re-factored OCI8 extension, check the output of `phpinfo()`. The refactored OCI8 extension displays seven directives with an “oci8.” prefix. These are not present in the previous incarnation.

#### 4. Rebuild PHP:

```
make
make install
```

#### 5. Copy the PHP initialization file to the location given by `--with-config-file-path:`

```
cp php.ini-recommended $HOME/apache/conf/php.ini
```

#### 6. Set `LD_LIBRARY_PATH` to `/usr/lib/oracle/10.1.0.3/client/lib`.

If you are using a `tnsnames.ora` file to define Oracle Net service names (database aliases), set the `TNS_ADMIN` environment variable to the directory containing the file.

Set any other required Oracle globalization language environment variables, such as `NLS_LANG`. If nothing is set, the default local environment is used. See the *Globalization* chapter, for more information on globalization with PHP and Oracle.

Unset any unrequired Oracle environment variables, such as `ORACLE_HOME` and `ORACLE_SID`.

#### 7. Restart the Apache HTTP Server.

---

Note: The configuration option in older versions of PHP is `--with-oci8-instant-client`. See the configuration help using the `./configure -help` command to check which syntax you should use.

If you are using a version of PHP that is earlier than releases 4.3.11 and 5.0.5, you need to apply a PHP patch (PHP bug number 31084) to build PHP with Oracle Instant Client.

---

## Installing Oracle Instant Client on Windows

After installing Apache HTTP Server, and PHP, you can install Oracle Instant Client. To install Oracle Instant client:

1. Download the Instant Client Basic package for Windows from the Instant Client page on the Oracle Technology Network (<http://www.oracle.com/technology/tech/oci/instantclient/instantclient.html>). The zip file is about 30MB in size.
2. Create a new directory (for example, `C:\instantclient10_1`). Unzip the downloaded file into the new directory. If you want to use the minimum files required, you can unzip just the following files to the new directory:
  - `oraociei10.dll`
  - `oranzsbb10.dll`

- oci.dll
3. Edit the environment and add the location of the Oracle Instant Client files, *C:\instantclient10\_1*, to the PATH environment variable, before any other Oracle directories. For example, on Windows 2000, use **Start > Settings > Control Panel > System > Advanced > Environment Variables**, and edit PATH in the System Variables list.

If you are using a *tnsnames.ora* file to define Oracle Net service names (database aliases), copy the *tnsnames.ora* file to *C:\instantclient10\_1*, and set the user environment variable TNS\_ADMIN to *C:\instantclient10\_1*. A default service name can optionally be set in the user environment variable LOCAL.

Set any other required Oracle globalization language environment variables, such as NLS\_LANG. If nothing is set, the default local environment is used. See the *Globalization* chapter, for more information on globalization with PHP and Oracle.

Unset any unrequired Oracle environment variables, such as ORACLE\_HOME and ORACLE\_SID.

4. Restart the Apache HTTP Server.





## CHAPTER 9

# Connecting to Oracle Using OCI8

This section covers the ways in which you can connect to an Oracle database from your PHP application, including the types of Oracle connections, environment variables that may effect connections, and tuning your connections.

## Oracle Connection Types

There are three ways to connect to an Oracle database in a PHP application, using a standard connections, unique connections, or persistent connections.

### Standard Connections

For basic connection to Oracle use PHP's `oci_connect()` call:

```
$c = oci_connect($username, $password, $dbname);
```

You can call `oci_connect()` more than once in a script. If you do this and use the same username and database name, then you get a pointer to the original connection.

### Multiple Unique Connections

To get a totally independent connection use `oci_new_connect()`:

```
$c = oci_new_connect($username, $password, $dbname);
```

Each connection is separate from any other. This lets you have more than one database session open at the same time. This can be useful when you want to do database operations independently from each other.

### Persistent Connections

Persistent connections can be made with `oci_pconnect()`:

```
$c = oci_pconnect($username, $password, $dbname);
```

Persistent connections are not automatically closed at the end of a PHP script and remain open for reuse in other scripts. This makes them fast. Limits on the number of persistent connections can be set, and connections can be automatically expired to free up resources. The parameters for tuning persistent connections are discussed later in this chapter.

## Oracle Database Name Connection Strings

The “\$dbname” connection string is the name of the database that you want to attach to. It can be local or remote. Having an invalid connection string can lead to the Oracle error:

```
ORA-12514 TNS:listener does not currently know of service requested in connect descriptor
```

The database name can be one of:

- \* An Easy Connect string
- \* A full connection string
- \* A database alias

### Easy Connect String

If you are running Oracle XE on the same machine as the PHP-enabled web server, you could connect to the HR schema with:

```
$c = oci_connect('hr', 'hr_password', '//localhost/XE');
```

This uses the Easy Connect string, which is JDBC-like. The syntax is:

```
[//]hostname[:port][ /service_name]
```

The port defaults to Oracle’s standard port, 1521. The service name defaults to same name as the host computer. The “//” prefix is part of the syntax and is not a PHP line comment.

You can use this syntax to connect to Oracle8i, Oracle9i and Oracle10g databases as long as PHP is linked with Oracle 10g libraries. Zend Core for Oracle uses the appropriate Oracle libraries.

More information on the syntax can be found in the [Oracle® Database Net Services Administrator's Guide 10g Release 2 \(10.2\)](#).

### Full Database Connection String

The full Oracle Net connection string gives total flexibility over the connection.

```
$db = '(DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)
        (HOST = mymachine.mydomain)(PORT = 1521))
      (CONNECT_DATA =
        (SERVER = DEDICATED)
        (SERVICE_NAME = MYDB.MYDOMAIN)))';
```

```
$c = oci_connect($username, $password, $db);
```

When in doubt, copy the connection string used by other Oracle tools and users.

The syntax can be more complex than the example, depending on the database and Oracle Net features used. For example, by using the full syntax, you can enable Oracle Net features like load balancing and tweak packet sizes. The Easy Connect syntax doesn't allow this flexibility.

## Database Alias

You can store the full connection string in a file called *tnsnames.ora* and refer to it in PHP using an alias:

```
# tnsnames.ora
MYA = (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)
                (HOST = mymachine.mydomain)(PORT = 1521))
      (CONNECT_DATA =
        (SERVER = DEDICATED)
        (SERVICE_NAME = MYDB.MYDOMAIN)))
```

In PHP you would use the alias MYA to connect to the database:

```
$c = oci_connect($username, $password, 'MYA');
```

PHP needs to be able to find the *tnsnames.ora* file to resolve the MYA alias. The directory paths that Oracle searches for *tnsnames.ora* depend on your operating system. The search path always includes the directory specified by the TNS\_ADMIN environment variable and a default location.

In an “ORACLE\_HOME” style install the default *tnsnames.ora* is in:

```
$ORACLE_HOME/network/admin/tnsnames.ora
```

For example, in Oracle XE, \$ORACLE\_HOME is

```
/usr/lib/oracle/xe/app/oracle/product/10.2.0/server
```

If you have compiled PHP yourself and are running it with the Oracle libraries in an ORACLE\_HOME-style install, then you would typically have ORACLE\_HOME set and the default *tnsnames.ora* file will automatically be found without needing to set TNS\_ADMIN.

In Zend Core for Oracle, the ORACLE\_HOME variable is generally not set when Apache starts, so the default file will not be found. Instead, set the TNS\_ADMIN variable.

## Oracle Environment Variables for Connections

If you need to change Oracle environment settings like the language and territory conventions, or are configuring and building PHP yourself, you will need to set some environment variables in the shell that starts the web server. The OCI8 extension always needs to find Oracle libraries and message files. Not finding the files can lead to errors like “Error while trying to retrieve text for error ORA-12154”. This message means that an error ORA-12154 occurred (which is one problem, in this case a connection problem). It also means that OCI8 couldn’t even find Oracle’s message files to get the text describing the error (a second problem). The solution for both problems is often the same: to set the environment correctly.

Another potential source of problems is having multiple installations of Oracle. Finding mismatched versions of Oracle libraries and files can lead to PHP returning the Oracle error "ORA-12705: Cannot access NLS data files or invalid environment specified". The bottom line is that your environment should be set correctly and consistently. Environment variables must be set in the shell that starts Apache so when the OCI8 extension is first loaded it has access to the correct values.

Note: Do not set environment variables in a PHP script with `putenv()`. It causes hard to track errors as the behavior is not consistent for all OCI8 functions.

---

To connect to a remote database, or when PHP is linking with `ORACLE_HOME` binaries, you may need some or all of the following environment variables to be set:

- \* `ORACLE_HOME`
- \* `ORACLE_SID`
- \* `LD_LIBRARY_PATH`
- \* `NLS_LANG`
- \* `TNS_ADMIN`

The `NLS_LANG` and `TNS_ADMIN` variables are most likely to be required for Zend Core for Oracle.

Zend Core for Oracle modifies *apachectl* and adds `LD_LIBRARY_PATH`. (This may not be required in a future version of Zend Core for Oracle if Oracle links the Instant Client differently). This allows the Zend Core for Oracle GUI Console to be reused to start Apache.

If you are using a *tnsnames.ora* file and specify network aliases for the connection string with Zend Core for Oracle, you may need to do something similar with `TNS_ADMIN` so the Zend Core for Oracle Console can restart Apache correctly.

If you start Apache manually, set the environment in a calling script:

*start\_apache.sh*

```
#!/bin/sh

TNS_ADMIN=/usr/local/apache/conf
export TNS_ADMIN
echo Starting Apache
#export > /tmp/envvars
/usr/local/apache/bin/apachectl start
```

This example assumes */usr/local/apache/conf/tnsnames.ora* exists. `TNS_ADMIN` points to the directory containing the *tnsnames.ora* file.

You may come across connections that do not specify a connection string:

```
$c = oci_connect($username, $password);
```

This works when `ORACLE_SID` is set to a local database on the host running PHP.

If you configure PHP yourself and have problems, check the output from the `phpinfo()` function. Look at the Environment section and make sure the Oracle variables are set to the values you expect.

## Closing Oracle Connections

At the end of each script, connections opened with `oci_connect()` or `oci_new_connect()` are automatically closed. You can also explicitly close them at any time by calling:

```
oci_close($c);
```

Any uncommitted data is rolled back.

If a long running script only spends a small amount of time interacting with the database, you may want to close connections to free resources for other users.

Connections opened with `oci_pconnect()` will not be closed by `oci_close()`. This is similar to the way persistent resources work in other PHP extensions. Idle persistent connections can be set to expire.

The `oci_close()` function was a “no-op” prior to the re-factoring of OCI8. That is, it had no functional code, and did not actually close a connection. You could not explicitly close connections even if you wanted to! This has now changed, but you can revert to the old behavior if necessary with this *php.ini* setting:

```
oci8.old_oci_close_semantics = 1
```

`oci_close()` works by reference counting. Only when all references to the PHP connection are finished will the database connection actually be closed. In this example `$c1` and `$c2` are the one database connection (because `oci_connect` returns the same connection resource when called more than once in a script), but only at the end of script is the database connection really closed.

*close.php*

```
<?php
function do_query($c, $query)
{
    $s = oci_parse($c, $query);
    oci_execute($s);
    oci_fetch_all($s, $res);
    echo "<pre>";
    var_dump($res);
    echo "</pre>";
}
$c1 = oci_connect('hr', 'hr', '//localhost/XE');
$c2 = oci_connect('hr', 'hr', '//localhost/XE');

do_query($c1, 'select user from dual');
oci_close($c1);
do_query($c1, 'select user from dual'); // fails
do_query($c2, 'select user from dual'); // succeeds
oci_close($c2);

?>
```

In this example, the second query fails, but the third succeeds. Although the reference counting algorithm keeps the database connection physically open, the connection resource in `$c1` is not usable after `oci_close($c1)` is executed.

## Connection Cleanliness with Persistent Connections

Uncommitted data is rolled back at the end of a PHP script. For `oci_pconnect()` this means subsequent scripts reusing the connection will not see any data they shouldn't. However it is

possible for a script to change connection attributes for `oci_pconnect()` that are not reset at the end of the script:

```
$c = oci_pconnect("hr", "hr", "//localhost/XE");
do_query($c, "select sysdate from dual");
$s = oci_parse($c, "alter session set nls_territory=germany");
$r = oci_execute($s);
do_query($c, "select sysdate from dual");
```

The first time this is called, the two queries return:

```
01/MAR/06
01.03.06
```

The second date has a new format because the ALTER SESSION command changes the globalization setting. Calling the script a second time gives:

```
01.03.06
01.03.06
```

The persistent connection has retained the session settings. (The Oracle term “session” is effectively the same as the PHP term “connection”). This only happens if the same Apache process serves both HTTP requests. If a new Apache process serves the second request then it will open a new connection to the database, which will have the original default date format. Also the system will have two persistent connections left open instead of one.

Session changes like this may not be a concern. Your applications may never need to do anything like it, or all connections may need the same values anyway.

## Optional Connection Parameters

The `oci_connect()`, `oci_new_connect()` and `oci_pconnect()` functions take an optional extra two parameters:

- \* Connection character set
- \* Connection privilege level

### Connection Character Set

The character set is a string containing an Oracle character set name, for example, “ja16euc”.

```
$c = oci_connect("hr", "hr", "//localhost/XE", 'ja16euc');
```

When not specified or NULL, the NLS\_LANG environment variable setting is used. This setting determines how Oracle translates data when it is transferred from the database to PHP. If the characters sets are not equivalent some data may get converted abnormally.

It is up to your application to handle returned data correctly, perhaps by using PHP's `mb_string` functionality. Globalization is discussed in more detail in the *Globalization* chapter.

## Connection Privilege Level

One of the new features of the re-factored OCI8 extension is the ability to allow privileged SYSDBA and SYSOPER connections. Privileged connections are disabled by default. They can be enabled in *php.ini* using:

```
oci8.privileged_connect = 1
```

The SYSDBA and SYSOPER privileges give you the ability to change the state of the database, perform data recovery, and even access the database when it has not fully started. Be very careful about exposing this on customer facing web sites, that is, don't do it! It might be more useful for command line PHP scripts.

When creating a database user, you grant privileges to enable the user to connect to the database, to run queries and make updates, and to create schema objects.

The following administrative user accounts are automatically created when you install Oracle. They are both created with the password that you supplied upon configuration.

**SYSTEM**— This is the user account that you log in with to perform all administrative functions, other than starting up and shutting down the database.

**SYS**— All base tables and views for the database data dictionary are stored in the SYS schema and are critical for the operation of Oracle.

By default, the SYSDBA privilege is assigned only to user SYS. A database administrator can grant SYSDBA or SYSOPER to any user.

## Operating System Authenticated Privileged Connections

You can have the operating system perform the authentication for privileged connections based around the user that is running the web server system process. An operating system authenticated privileged connection is equivalent to the SQL\*Plus connection:

```
$ sqlplus / as sysdba
```

In PHP for “/ as sysdba” access (where no username and password is used), all these must be true:

- \* The operating system process user is run as a member of the “dba” group
- \* PHP is linked with the ORACLE\_HOME software (that is, not Instant Client, or Zend Core for Oracle) that the database is running as.
- \* The database is your default local database, for example, specified by the ORACLE\_SID environment variable

This would be typically be done by compiling and running PHP with the Oracle libraries used by the database.

Scripts that contain operating system authenticated privileged connection calls will connect successfully:

```
$c = oci_connect("/", "", null, null, OCI_SYSDBA);
```

If PHP is invoked by Apache, the library path needs to contain the same Oracle libraries. Also, theoretically, the *nobody* user must be in the privileged Oracle group, for example, in the operating system “dba” group. This is not recommended.

Similarly, AS SYSOPER access is available for members of the “oper” group. In PHP use OCI\_SYSOPER in `oci_connect()`.

On Windows, the operating system groups are called ORA\_DBA and ORA\_OPER.

## Remote Privileged Access

With Zend Core for Oracle and any other PHP based on Oracle's Instant Client, a username and password must be given when connecting. These connections are considered "remote" from the database because the libraries used by PHP are not those used by the running database.

Remote users can make privileged connections only when they have been given the appropriate Oracle access. In SQL\*Plus a privileged session would be started like:

```
$ sqlplus un/pw@sid as sysdba
```

This configuration is considered "remote" from the database. The database will not permit the (possibly physically) "remote" operating system to authorize access. An extra Oracle password file needs to be created and a password needs to be used in the database connection.

To set up a password file, check the database initialization parameter *remote\_login\_passwordfile* is EXCLUSIVE. This is the default value. To do this, log in to the operating system shell as the Oracle database software owner, and start SQL\*Plus:

```
$ sqlplus / as sysdba
```

```
SQL> show parameter remote_login_passwordfile
```

NAME	TYPE	VALUE
remote_login_passwordfile	string	EXCLUSIVE

EXCLUSIVE means the password file is only used with one database and not shared among several databases on the host.

If you need to change the value, use SQL\*Plus and enter a command like (but read up on the details):

```
SQL> alter system set remote_login_passwordfile='exclusive'  
scope=spfile sid='*';
```

From the operating system shell, create an Oracle password file:

```
$ $ORACLE_HOME/bin/orapwd file=$ORACLE_HOME/dbs/acct.pwd \  
password=secret entries=10
```

This creates a password file named *acct.pwd* that allows up to 10 privileged users with different passwords (this number can be changed later). The file is initially created with the password "secret" for users connecting with the username SYS.

To add a new user to the password file use SQL\*Plus:

```
SQL> create user c1 identified by c1pw;  
SQL> grant connect to c1;  
SQL> grant sysdba to c1;  
SQL> select * from v$pwfile_users;  
USERNAME                                SYSDBA  SYSOPER  
-----                                -
```

SYS	TRUE	TRUE
C1	TRUE	FALSE

Now in PHP you can use the following connection string:



```
$c = oci_connect("c1", "c1pw", '//localhost/XE', null, OCI_SYSDBA);
```

One feature of a privileged connection is that if you issue a `SELECT USER FROM DUAL` statement, any `OCI_SYSDBA` connection will show the user as `SYS` not `C1`. A connection made with `OCI_SYSOPER` will show a user of `PUBLIC`.

## The Transactional Behavior of Connections

To see the transactional behavior of the three connection functions, use `SQL*Plus` to create a table with a single date column:

```
SQL> create table mytable (col1 date);
```

Rerun this script a few times, changing `oci_connect()` to `oci_new_connect()` and `oci_pconnect()`:

*transactions.php*

```
<?php

function do_query($c, $query)
{
    $s = oci_parse($c, $query);
    oci_execute($s, OCI_DEFAULT);
    oci_fetch_all($s, $res);
    echo "<pre>";
    var_dump($res);
    echo "</pre>";
}

$c1 = oci_connect("hr", "hr", "//localhost/XE");

$s = oci_parse($c1,
    "insert into mytable values ('" . date('j:M:y') . "')");
$r = oci_execute($s, OCI_DEFAULT); // No COMMIT occurs
do_query($c1, "select * from mytable");
$c2 = oci_connect("hr", "hr", "//localhost/XE");
do_query($c2, "select * from mytable");

?>
```

Using `oci_connect()` connection lets you query the newly inserted (but uncommitted) data because `$c1` and `$c2` refer to the same Oracle connection. Using `oci_pconnect()` is the same as `oci_connect()`.

Using `oci_new_connect()` for `$c2` gives a new connection which cannot see the uncommitted data.

## Tuning Oracle Connections in PHP

Connections to Oracle can be tuned by changing the OCI8 calls made, by changing the network configuration, and by tuning the database.

Using `oci_pconnect()` makes a big improvement in overall connection speed of frequently used applications.

## Tuning Oracle Net

There are many ways to configure connection and authentication. For example a connection:

```
$c = oci_connect("hr", "hr", "abc");
```

Could be evaluated by Oracle10g as using the Easy Connect syntax to host machine "abc" (using the default port and database service) or using a net alias "abc" configured in a *tnsnames.ora* file.

The flexibility can cause a delay in getting an error back if the connection details are invalid or a database is not operational. Both internal connection methods may be tried in sequence adding to the time delay before a PHP script gets the error. This depends on your Oracle Net and DNS settings.

How Oracle is configured to authenticate the user's credentials (here a username and password) can also have an effect.

The issue is not specific to PHP. In SQL\*Plus the connection

```
$ sqlplus hr/hr@abc
```

Would be the same.

In a basic Oracle10g installation, one way to return an error as soon as possible is to set this in your client *sqlnet.ora* file:

```
NAMES.DIRECTORY_PATH = (TNSNAMES)
SQLNET.AUTHENTICATION_SERVICES = (NONE)
```

The `DIRECTORY_PATH` value disables Easy Connect's `//hostname:port/service` syntax. Instead of using Easy Connect syntax, create a *tnsnames.ora* and use a network alias. (This may also add a small measure of security if your scripts accidentally allow arbitrary connection strings. It stops users guessing database server hostnames they shouldn't know about.)

Setting `AUTHENTICATION_SERVICES` to `NONE` stops different authentication methods being tried. Although this may prevent privileged database connections, which require operating system authorization, this, again, might be beneficial. Check the Oracle Net documentation for details and for other authentication methods and authentication-type specific timeout parameters.

The *sqlnet.ora* file should be on the machine running Apache and in the same directory as your *tnsnames.ora* file. If you are not using a *tnsnames.ora* file, set the environment variable `TNS_ADMIN` to the directory that contains the *sqlnet.ora* file.

## Other Oracle Net Optimizations

Oracle Net lets you tune a lot of other options too, including the session data unit size. You may also be able to tune your TCP/IP stack.

For sites that have a large number of connections being made, you can tune the `QUEUESIZE` option in the *listener.ora* file.

If network consumption is an issue, Oracle Net's Connection Pooling and Session Multiplexing features should be evaluated. Database resource usage of a large number of connections is potentially more likely to be significant than network consumption.

Another potential Oracle Net optimization, but one to be used with caution, is to hardcode the database host IP address in the *tnsnames.ora* file. This removes the need to do a hostname lookup. The benefit may be small on platforms that cache the IP address well. The drawback is greatly reduced reliability.

Check the *Oracle Net Services Administrator's Guide* and the *Oracle Net Services Reference* for details, and for all the wonderful variety of Oracle Net options available.

## Tracing Oracle Net

Sometimes your network is the bottleneck. If you suspect this is the case, turn on Oracle Net tracing in your client *sqlnet.ora* file and see where time is being spent. The sample *sqlnet.ora* in *\$ORACLE\_HOME/network/admin/sample* has some notes on the parameters that help. For example, with a USER level trace in *sqlnet.ora*:

```
trace_level_client = USER
trace_directory_client = /tmp
```

And the PHP code:

```
$c = oci_connect("hr", "hr", "#c"); // invalid db name
```

The trace file, for example */tmp/cli\_3232.trc*, shows:

```
...
[10-MAY-2006 09:54:58:100] nnftmlf_make_system_addrfile: system
names file is ...
[10-MAY-2006 09:55:00:854] snlinGetAddrInfo: Name resolution failed
for #c
...
```

The left hand column is the timestamp of each low level call. Here, it shows a relatively big time delay doing name resolution for the non-existent host “#c”. The cause is the configuration of the machine network name resolution.

## Tuning Persistent Connections

Persistent connections are great if the cost of opening a connection is high. What you consider high depends on your application requirements and on implementation issues such as whether the web server and database are on the same host. The drawback is the connection uses Oracle resources even when no one is accessing the application or database. And if Apache spawns a number of server processes, each of them may have its own set of connections to the database. The proliferation of connections can be controlled with *php.ini* directives.

### Maximum Number of Persistent Connections Allowed

```
oci8.max_persistent = -1
```

This parameter limits the number of persistent connections cached by each Apache process. When the limit is reached, all *oci\_pconnect()* calls are treated like *oci\_connect()* calls and are closed at the end of the script. Setting it to -1 (the default) means there is no limit.

## Timeout for Unused Persistent Connections

```
oci8.persistent_timeout = -1
```

This parameter is the length in seconds that an Apache process maintains an idle persistent connection. The expiry check happens whenever a PHP script finishes regardless of whether the script calls OCI8 functions. Setting this parameter to -1 (the default) means there is no timeout. If a connection has been expired, the next time `oci_pconnect()` is called a new connection is created.

## Pinging for Closed Persistent Connections

When a PHP script tries to use a connection it thinks is still open but the database or network has closed, it will get an error. The *ping* interval is an easy way to improve connection reliability for persistent connections.

```
oci8.ping_interval = 60
```

This parameter is the number of seconds that pass before OCI8 does a ping during `oci_pconnect()`. If the ping determines the connection is no longer usable, a new connection is transparently created. To disable pinging, set the value to -1. The default value is 60 seconds. When set to 0, PHP checks the database each time `oci_pconnect()` is called.

Regardless of the value of `oci8.ping_interval`, `oci_pconnect()` will always check an Oracle client-side setting to see if the server was known to be available the last time anything was received from the server. This is a quick operation. Setting `oci8.ping_interval` additionally physically sends a message to the server, causing a “round-trip” over the network. This is a “bad thing” for scalability.

Good application design gracefully recovers from failures. In any application there are a number of potential points of failure including the network, the hardware and user actions such as shutting down the database. Oracle itself may be configured to close idle connections and release their database resources. The database administrator may have installed user profiles with `CREATE PROFILE IDLE_TIMEOUT`, or the Oracle network layer may time out the network.

You need to balance performance (no pings) with having to handle disconnected Oracle sessions (or other changes in the Oracle environment) in your PHP code. For highest reliability and scalability it is generally recommended that you don't use `oci8.ping_interval`, but do error recovery in your application code.

Of course many smaller applications can benefit from the increased ease of use of the ping functionality.

## Apache Configuration Parameters

You can also tune Apache to kill idle processes, which will also free up Oracle resources used by persistent connections:

**MaxRequestsPerChild:** This parameter sets how many requests Apache will serve before restarting

**MaxSpareServers:** This parameter sets how many servers to keep in memory that are not handling requests

**KeepAlive:** This parameter defines whether Apache can serve a number of documents to the one user over the same HTTP connection

Ideally your *php.ini* directives work in harmony with your Apache settings and your timeouts happen predictably.

John Coggeshall's article [Improving Performance Through Persistent Connections](#) discusses Apache configuration in more depth.

## Connection Management in Scalable Systems

Oracle achieves its well-known scalability in part through a multi-threaded architecture. PHP instead has a multi-process architecture. This difference means care is required when designing scalable applications.

Using persistent connections is common for web sites that have high numbers of connections being established. Reusing a previously opened connection is significantly faster than opening a fresh one. Large sites should benchmark persistent PHP connections and Oracle Shared Servers also known as “Multi Threaded Servers” (MTS). Shared Servers reduce the numbers of processes needed to handle database requests.

Make sure that you understand the lifetime of your applications connections. Reuse connections where possible, but don't be afraid to create new connections and close them as needed. Each connection will take some Oracle memory, so overall load can be reduced if idle connections are closed with Apache process timeouts or with the *php.ini* parameters to expire persistent connections.

For sites with hundreds of connections a second, tune the cache size of an internal sequence generator, `sys.audses$`. A starting point is to change it to perhaps 10000:

```
SQL> alter sequence sys.audses$ cache 10000;
```

This is also recommended if you are using Oracle RAC (“Real Application Clusters”).

With Oracle RAC you can adjust the algorithm that decides which Oracle node handles each new connection. Set the *listener.ora* parameter `PREFER_LEAST_LOADED_NODE_<listener_name>` to OFF to use session based load balancing. This is documented in Note 220970.1 on Oracle's Metalink support site:

*Session based load balancing takes into account the number of sessions connected to each node and then distributes new connections to balance the number of sessions across the different nodes*

This can help when there is a connection “storm” and the normal allocation metrics do not get a chance to get updated fast enough.

For both RAC or non-RAC database, the `DBMS_SERVICE` package lets you specify workload management goals. This is a detailed topic; refer to Oracle's manuals for more information.

Finally, make sure that your applications are as efficient as possible. This minimizes the length of time connections are held.



## CHAPTER 10

# Executing SQL Statements With OCI8

This chapter discusses executing SQL statements using the Oracle OCI8 extension, including how statements are executed, the functions available, transactions, tuning your queries, and some tips and tricks that you might find useful in your applications.

## SQL Statement Execution Steps

Queries using the OCI8 extension follow a model familiar in the Oracle world: parse, execute and fetch. Statements like CREATE and INSERT require only parsing and executing. Parsing is really just a preparatory step, since Oracle's actual text parse can occur at the execution stage. You can optionally "bind" local values into a statement similar to the way you use "%s" print format specifiers in strings. This improves performance and security. You can also "define" where you want the results to be stored, but almost all scripts let the OCI8 fetch functions take care of this.

The possible steps are:

- \* **Parse**—Prepares a statement for execution
- \* **Bind**—Optionally lets you bind data values, for example, in the WHERE clause, for better performance and security
- \* **Define**—Optional step allowing you to specify which PHP variables will hold the results. This is not commonly used
- \* **Execute**—The database processes the statement and buffers any results
- \* **Fetch**—Gets any query results back from the database

There is no one-stop function to do all these steps in a single PHP call, but it is trivial to create one in your application and you can then add custom error handling requirements.

## Query Example

A basic query in OCI8 is:

```
<?php
$c = oci_connect("hr", "hr", "://localhost/XE");
$s = oci_parse($c, 'select city, postal_code from locations');
oci_execute($s);
print '<table border="1">';
while ($row = oci_fetch_array($s, OCI_RETURN_NULLS)) {
    print '<tr>';
    foreach ($row as $item)
```

## Executing SQL Statements With OCI8

```
        print '<td>'.$item.'</td>';
    print '</tr>';
}
print '</table>';
oci_close($c);
?>
```

In PHP, single and double quotes are used for strings. Strings with embedded quotes can be made by escaping the nested quotes with a backslash, or by using both quoting styles. The next example shows single quotes around the city name. To make the query a PHP string, it is fully enclosed in double quotes:

```
$s = oci_parse($c,
               "select * from locations where city = 'Sydney'");
```

## Oracle Datatypes

Each column has a datatype, which is associated with a specific storage format. The common built-in Oracle datatypes are:

- \* CHAR
- \* VARCHAR2
- \* NCHAR and NVARCHAR2
- \* NUMBER
- \* DATE
- \* BLOB
- \* CLOB and NCLOB
- \* BFILE
- \* XMLType

The CHAR, VARCHAR2, NUMBER, and DATE datatypes are stored directly in PHP variables. BLOB, CLOB, and BFILE datatypes use PHP descriptors and are shown in the *Using Large Objects in OCI8* chapter. XMLTypes are returned as strings. NCHAR, NVARCHAR2, and NCLOB are not supported in the OCI8 extension.

## Fetch Functions

There are a number of fetch functions, all carefully documented in the [PHP OCI8 Reference Manual](#).

- \* **oci\_fetch\_all()**—Gets all the results at once
- \* **oci\_fetch\_array()**—Gets the next row as an array of your choice
- \* **oci\_fetch\_assoc()**—Gets the next row as an associative array
- \* **oci\_fetch\_object()**—Gets a new row as an object
- \* **oci\_fetch\_row()**—Gets the next row as an integer indexed array



- \* **oci\_fetch()**—Used with `oci_result()`, which returns the result of a given field

Functions that fetch a single row need to be called repeatedly:

```
$s = oci_parse($c, "select postal_code from locations");
oci_execute($s);
while ($res = oci_fetch_array($s)) {
    echo $res['POSTAL_CODE'] . "<br>\n";
}
```

Some of the functions have optional parameters that alter their behavior, for example the options to `oci_fetch_array()` are:

- \* **OCI\_ASSOC** – Return results as an associative array
- \* **OCI\_NUM** – Return results as a numerically indexed array
- \* **OCI\_BOTH** – return results as both associative and numeric arrays
- \* **OCI\_RETURN\_NULLS** – return PHP NULL value for NULL data
- \* **OCI\_RETURN\_LOBS** – return the actual LOB data instead of an OCI- LOB resource

To fetch results in a numeric index and have null data returned using `oci_fetch_array()`:

```
$s = oci_parse($c, "select city, postal_code from locations");
oci_execute($s);
while ($res = oci_fetch_array($s, OCI_NUM+OCI_RETURN_NULLS)) {
    echo $res[0] . " - ".$res[1]."<br>\n";
}
```

Associative arrays are keyed by the uppercase column name. There is no table prefix. If you join tables where the same column name occurs with different meanings in both tables, use a column alias in the query:

```
$s = oci_parse("select region_name,
               regions.region_id as myalias,
               country_name,
               countries.region_id
               from countries
               inner join regions
               on countries.region_id = regions.region_id");
```

The eventual output of this query has column array indices `REGION_NAME`, `MYALIAS`, `COUNTRY_NAME`, and `REGION_ID`.

Some of the fetch functions don't return NULL data by default. This can be tricky when using numerically indexed arrays. The result array can appear to have fewer columns than selected, and you can't always tell which column was NULL. Either use associative arrays so the column names are directly associated with their values, or specify the `OCI_RETURN_NULLS` flag.

## Insert, Update, Delete, Create and Drop

Executing Data Manipulation Language (DDL) and Data Definition Language (DML) statements, like `CREATE` and `INSERT`, simply requires a parse and execute:

```
$s = oci_parse($c1, "create table iltest (col1 number)");  
oci_execute($s);
```

The one-off installation sections of applications should contain almost all the CREATE TABLE statements used. Applications in Oracle do not commonly need to create temporary tables at run time. Use inline views, or join tables when required instead of creating a temporary table that has no statistics for the Oracle optimizer to evaluate. Tom Kyte talks about [global temporary tables](http://asktom.oracle.com) on asktom.oracle.com

In the Oracle database, creating and dropping tables will automatically commit all uncommitted data. This cannot be changed.

For other data changing statements, `oci_execute()` will commit data by default:

```
$s = oci_parse($c1, "insert into iltest (col1) values (1)");  
oci_execute($s);
```

This can be handy for single INSERTs and UPDATEs, but transactional and performance requirements should be thought about before using the default mode everywhere. Transactions and performance are discussed more in this chapter.

## Transactions

Using transactions to protect the integrity of data is as important in PHP as any other relational database application. Except in special cases, you want either all your changes to be committed, or none of them.

OCI8's default commit behavior is like other PHP extensions and different from Oracle's standard. The default mode of `oci_execute()` is `OCI_COMMIT_ON_SUCCESS`. Also at the end of a script any uncommitted changes are automatically rolled back.

In the following example the changes are committed immediately when the `oci_execute()` call is called:

```
$s = oci_parse($c, "insert into testtable values ('my data')");  
oci_execute($s); // automatically committed
```

Unnecessarily committing or rolling back impacts database performance as it causes unnecessary network traffic ("round trips" between PHP and the database) and wasteful input and output to the database files. To maximize efficiency, use transactions where appropriate.

You specify not to auto-commit with:

```
$s = oci_parse($c, "insert into testtest values ('my data 2')");  
oci_execute($s, OCI_DEFAULT); // not committed
```

The PHP manual for `oci_execute()` describes the parameter succinctly:

*"When using OCI\_DEFAULT mode, you're creating a transaction. Transactions are automatically rolled back when you close the connection, or when the script ends, whichever is soonest. You need to explicitly call `oci_commit()` to commit the transaction, or `oci_rollback()` to abort it."*

This is fine. It allows you to optimize your scripts and keep your data integrity intact. But there is a potential drawback of this mode for sites under heavy load. OCI8 may implicitly, and unnecessarily, issue a rollback at the end of your script. This can occur in a script that

does only a query. The OCI8 extension doesn't know it was a query and so it does a rollback in case any data was changed.

Simple rules of thumb are for scripts that do multiple inserts or deletes, pass the parameter `OCI_DEFAULT` to `oci_execute()` and explicitly use `oci_commit()` when you want to commit the transaction:

```
oci_execute($s1, OCI_DEFAULT);
...
oci_execute($s2, OCI_DEFAULT);
oci_commit($c); // note: pass the connection resource
```

If you are just querying data, or the only thing your script does is insert a single row, then do not specify `OCI_DEFAULT`:

```
oci_execute($s);
```

The PHP parameter name `OCI_DEFAULT` is borrowed from Oracle's Call Interface, where the value of the C macro with the same name is actually the default value when nothing else is specified. In PHP a better name would have been `NO_AUTO_COMMIT`, but we are now stuck with the awkward usage.

Be careful mixing and matching calls with both modes in one script, since you may end up auto-committing and later also explicitly committing, or simply committing at incorrect times. Benchmarking and testing your applications in your environment is always wise.

To see exactly what calls to the Oracle database are made, turn on tracing at the top of your script with `oci_internal_debug()`. For a script that connects and does an insert:

```
<?php
  oci_internal_debug(1); // turn on tracing
  $conn = oci_connect("hr", "hr", "://localhost/XE");
  $s = oci_parse($conn, "insert into testtable values ('my data')");
  oci_execute($s, OCI_DEFAULT); // do not auto-commit
?>
```

You get output like:

```
OCI8 DEBUG: OCINlsEnvironmentVariableGet at (/tmp/php-5.1.3/ext/oci8/oci8.c:995)
OCI8 DEBUG: OCIEnvNlsCreate at (/tmp/php-5.1.3/ext/oci8/oci8.c:1151)
OCI8 DEBUG: OCIHandleAlloc at (/tmp/php-5.1.3/ext/oci8/oci8.c:1176)
OCI8 DEBUG: OCIServerAttach at (/tmp/php-5.1.3/ext/oci8/oci8.c:1185)
OCI8 DEBUG: OCIHandleAlloc at (/tmp/php-5.1.3/ext/oci8/oci8.c:1195)
OCI8 DEBUG: OCIHandleAlloc at (/tmp/php-5.1.3/ext/oci8/oci8.c:1204)
OCI8 DEBUG: OCIHandleAlloc at (/tmp/php-5.1.3/ext/oci8/oci8.c:1213)
OCI8 DEBUG: OCIAttrSet at (/tmp/php-5.1.3/ext/oci8/oci8.c:1223)
OCI8 DEBUG: OCIAttrSet at (/tmp/php-5.1.3/ext/oci8/oci8.c:1234)
OCI8 DEBUG: OCIAttrSet at (/tmp/php-5.1.3/ext/oci8/oci8.c:1244)
OCI8 DEBUG: OCIAttrSet at (/tmp/php-5.1.3/ext/oci8/oci8.c:1253)
OCI8 DEBUG: OCISessionBegin at (/tmp/php-5.1.3/ext/oci8/oci8.c:1284)
OCI8 DEBUG: OCIHandleAlloc at (/tmp/php-5.1.3/ext/oci8/oci8_statement.c:61)
```

## Executing SQL Statements With OCI8

```
OCI8 DEBUG: OCISstmtPrepare2 at (/tmp/php-5.1.3/ext/oci8/oci8_statement.c:65)
OCI8 DEBUG: OCIAttrSet at (/tmp/php-5.1.3/ext/oci8/oci8_statement.c:119)
OCI8 DEBUG: OCIAttrSet at (/tmp/php-5.1.3/ext/oci8/oci8_statement.c:128)
OCI8 DEBUG: OCIAttrGet at (/tmp/php-5.1.3/ext/oci8/oci8_statement.c:297)
OCI8 DEBUG: OCISstmtExecute at (/tmp/php-5.1.3/ext/oci8/oci8_statement.c:321)
OCI8 DEBUG: OCISstmtRelease at (/tmp/php-5.1.3/ext/oci8/oci8_statement.c:589)
OCI8 DEBUG: OCIHandleFree at (/tmp/php-5.1.3/ext/oci8/oci8_statement.c:601)
OCI8 DEBUG: OCITransRollback at (/tmp/php-5.1.3/ext/oci8/oci8.c:1400)
OCI8 DEBUG: OCISessionEnd at (/tmp/php-5.1.3/ext/oci8/oci8.c:1448)
OCI8 DEBUG: OCIHandleFree at (/tmp/php-5.1.3/ext/oci8/oci8.c:1452)
OCI8 DEBUG: OCIServerDetach at (/tmp/php-5.1.3/ext/oci8/oci8.c:1456)
OCI8 DEBUG: OCIHandleFree at (/tmp/php-5.1.3/ext/oci8/oci8.c:1460)
OCI8 DEBUG: OCIHandleFree at (/tmp/php-5.1.3/ext/oci8/oci8.c:1464)
OCI8 DEBUG: OCIHandleFree at (/tmp/php-5.1.3/ext/oci8/oci8.c:1468)
OCI8 DEBUG: OCIHandleFree at (/tmp/php-5.1.3/ext/oci8/oci8.c:1472)
```

Many of these calls just allocate local resources (“handles”) and set local state (“attributes”), but some require a “round trip” to the database.

One of these is the `OCITransRollback()` call near the end of the script. The `OCI_DEFAULT` flag said not to auto-commit and there was no explicit `oci_commit()` call. As part of PHP’s end of HTTP request shutdown at the conclusion of the script, the rollback was issued.

Note if you change to auto-commit mode you won’t see a call to `OCITransCommit()` because the commit message is piggy-backed with the execution call, thus saving a round-trip. If a script only inserts one row it is fine to auto-commit. Otherwise, do the transaction management yourself.

## Autonomous Transactions

Oracle’s procedural language for SQL, PL/SQL, allows you to do autonomous transactions, which are effectively sub-transactions. An autonomous transaction can be committed or rolled back without affecting the main transaction. This might be useful for logging data access. An audit record can be inserted even if the user decides to rollback their main change. An example is:

```
SQL> create table logtable (name varchar2(30));
SQL> create or replace procedure updatelog(name_p in varchar2)
as
  pragma autonomous_transaction;
begin
  insert into logtable (name) values(name_p);
  commit;
end;
/

SQL> insert into logtable values ('Sally');
```

```
SQL> execute updatelog('Fred');
SQL> rollback;
SQL> select name from logtable;
```

```
NAME
```

```
-----
Fred
```

You could call the PL/SQL function from PHP to log access.

## Handling Errors

The error handling of any solid application adds complexity and requires careful design. Expect the unexpected. Check all return codes.

For a start, in a production system you would turn off `display_errors` in `php.ini`. You don't want to leak internal information to web users, and you don't want your application pages to contain ugly error messages.

For development, enable the `E_STRICT` level for `error_reporting` in `php.ini` so you can catch any problems that will cause upgrade issues.

To handle errors in OCI8, using the `oci_error()` function.

For connection errors, no argument is needed:

```
$c = oci_connect("hr", "hr", "//localhost/XE");
if (!$c) {
    $e = oci_error(); // No parameter passed
    var_dump($e);
}
```

For parse errors, pass the connection resource:

```
$s = oci_parse($c, "select city from locations");
if (!$s) {
    $e = oci_error($c); // Connection handle passed
    var_dump($e);
}
```

For execution and fetching errors, pass the statement resource:

```
$rc = oci_execute($s);
if (!$rc) {
    $e = oci_error($s); // Statement handle passed
    var_dump($e);
}
```

```
$rc = oci_fetch_all($s, $results);
if (!$rc) {
    $e = oci_error($s); // Statement handle passed
    var_dump($e);
}
```

You might consider using PHP's '@' to suppress function errors. For pretty output, using PHP's output buffering functions may be a way to help hide errors from users.

```
function Connect(&$e, $un, $pw, $db)
```

## Executing SQL Statements With OCI8

```
{
    ob_start();
    $con = @oci_connect($un, $pw, $db);
    if (!$con) {
        DisplayOCIErrror();
        $e = ob_get_contents();
    }
    ob_end_clean();
    return($con);
}

function DisplayOCIErrror($r = false)
{
    if ($r)
        $err = @oci_error($r);
    else
        $err = @oci_error();
    echo "<p><b>Error</b>:</p>\n";
    echo "<pre>\n";
    if (isset($err['message'])) {
        echo htmlspecialchars($err['message']);
    }
    else {
        echo 'Unknown DB error';
    }
    echo '</pre>'. "\n";
}
```

## Limiting Rows and Creating Paged Datasets

Oracle's SQL does not have a LIMIT keyword. It isn't in the SQL standard and the vendors that use it have different implementations. There are several ways to limit the rows returned in OCI8.

The `oci_fetch_all()` function has optional arguments to specify a range of results to fetch. This is implemented by the extension, not by Oracle's native functionality. All rows preceding those you want still have to be fetched from the database.

```
$firstrow = 3;
$numrows  = 5;
oci_execute($s);
oci_fetch_all($s, $res, $firstrow, $numrows);
var_dump($res);
```

It is more efficient to let Oracle do the row selection. The canonical paging query for Oracle8i onwards is given on [asktom.oracle.com](http://asktom.oracle.com):

```
select *
from ( select a.*, rownum as rnum
      from (YOUR_QUERY_GOES_HERE -- including the order by) a
      where rownum <= MAX_ROWS )
where rnum >= MIN_ROWS
```

MIN\_ROWS is the row number of first row and MAX\_ROWS is the row number of last row to return. In PHP you might do this:

```
$mystmt = "select city from locations order by city";
$minrow = 4; // row number of first row to return
$maxrow = 8; // row number of last row to return

$pageysql = "select *
             from ( select a.*, rownum as rnum
                   from ( $mystmt ) a
                   where rownum <= :maxrow)
             where rnum >= :minrow";

$s = oci_parse($c, $pageysql);

oci_bind_by_name($s, ":maxrow", $maxrow);
oci_bind_by_name($s, ":minrow", $minrow);
oci_execute($s);
oci_fetch_all($s, $res);
```

Note that \$mystmt is not bound. Bind data is not treated as code, so you can't bind the text of the statement and expect it to be executed.

## Auto-Increment Columns

Auto increment columns in Oracle can be created using a sequence generator and a trigger. A sequence generator returns unique values when it is called. A trigger is a PL/SQL procedure that is automatically invoked at a pre-determined point. Together they can automatically insert incrementing identifiers when table rows are created.

Sequence generators are defined in the database and return Oracle numbers. Sequence numbers are generated independently of tables. Therefore, the same sequence generator can be used for more than one table or for anywhere that you want to use a unique number. Sequence generation is useful to generate unique primary keys for your data and to coordinate keys across multiple rows or tables.

You can get a new value from a sequence generator using the NEXTVAL operator in a SQL statement. This gives the next available number and increments the generator. The similar CURRVAL operator returns the current value of a sequence without incrementing the generator.

In SQL\*Plus a sequence can be created and called like:

```
SQL> CREATE SEQUENCE myseq;
SQL> SELECT myseq.nextval FROM dual;
```

```
      NEXTVAL
-----
          1
```

To create an auto-increment column on a table

```
SQL> CREATE TABLE mytable (myid number, mydata VARCHAR2(20));

SQL> CREATE TRIGGER mytrigger
      BEFORE INSERT ON mytable FOR EACH ROW
```

```
BEGIN
  SELECT myseq.nextval INTO :new.myid FROM DUAL;
END;
/
```

In PHP insert two rows:

```
$s = oci_parse($c, "insert into mytable (mydata) values ('Hello')");
oci_execute($s);
$s = oci_parse($c, "insert into mytable (mydata) values ('Bye')");
oci_execute($s);
```

Querying the table in SQL\*Plus shows the MYID values were automatically inserted:

```
SQL> select * from mytable;

  MYID MYDATA
-----
     2 Hello
     3 Bye
```

The identifier numbers start at 2 because the example query shown above when the sequence was created returned 1 and incremented the generator.

## Tuning SQL Statements in PHP Applications

Tuning is an art and a science. The database-centric approach to tuning is to first tune the application, next tune the SQL, and finally tune the database.

SQL tuning and Database tuning are covered in the Oracle documentation. SQL tuning will help make efficient use of table design, indexes and the Oracle optimizer. Database tuning maximizes throughput and allows efficient use of buffering and writing strategies.

On the OCI8 side, transaction management can help reduce round trips from PHP to the database. Consider using PL/SQL packages for complex data operations to minimize data transfers.

It is almost always more efficient to select the minimum amount of data necessary and only return rows and columns that will be used by PHP.

Efficient caching of statements by having reusable code and using bind variables are fundamental in improving database performance.

### Default Prefetch Size

You can tune PHP's overall query performance with two configuration parameters:

```
oci8.default_prefetch = 10
```

This parameter sets the number of records to be returned by Oracle when each database fetch occurs. The default value is 10. Tuning this setting higher can significantly improve performance of queries that return a large number of rows. It minimizes database server "round-trips" by returning as much data as requested each time. Testing will show the optimal size. There is no point using too large a value.

When pre-fetching is enabled, Oracle will cache the data in its client buffers and give PHP only the rows PHP itself requests. This is similar to an "array fetch", but Oracle handles all the caching.



You can also change the value at runtime with the `oci_set_prefetch()` function.

```
$s = oci_parse($c, "select * from all_objects");
oci_execute($s);
oci_set_prefetch($s, 100); // 10 is default oci8.default_prefetch
oci_fetch_all($s, $res);
```

## Default Statement Cache Size

You can set a client cache for SQL statement text with the `php.ini` directive:

```
oci8.statement_cache_size = 20
```

The default is 20 statements. Caching can be disabled by setting the value to 0.

The client-side statement cache is in addition to the standard server side statement cache. The client cache means even the text of the statement does not need to be transmitted to the database, further reducing network traffic and database server load. The cache is per-Oracle session so this feature is most likely to be useful when persistent connections are used.

Like many tuning options, there is a time/memory trade-off when tweaking these parameters. The statement cache also means slightly more load is put on the PHP host, offsetting the reduced load on the database host.

## Using Bind Variables

Bind variables are just like “%s” print format specifiers. They let you re-execute a statement with different values for the variables and get different results.

Binding is highly recommended. It can improve overall database throughput. Oracle can reuse any cached execution plan for the statement, even if someone else originally executed it.

Bind variables are also an important way to prevent SQL injection security attacks. SQL injection may occur when SQL statements are hard coded text concatenated with user input:

```
$s = oci_parse($c, "select ".$col." from mytable");
```

If the user input is not carefully checked, then it may be possible for a malicious user to execute a SQL statement of their choice instead of the one you intended. Over the years, extra functionality was added to PHP in an attempt to work around this kind of problem. One of the notorious features is “magic quotes”, which will go away in PHP 6 because of the confusion caused.

Bind variables give automatic protection against SQL injection attacks and you don’t need to resort to work arounds. User data is always treated as data and never as part of the SQL statement.

```
$s = oci_parse($c,
    "select last_name from employees where employee_id = :eidbv");
$myeid = 101;
oci_bind_by_name($s, ":EIDBV", $myeid);
oci_execute($s);
oci_fetch_all($s, $res);
echo "Last name is: ". $res['LAST_NAME'][0] . "<br>\n";
```

```
// No need to re-parse
```

```
$myeid = 102;
```

```
oci_execute($s);
oci_fetch_all($s, $res);
echo "Last name is: ". $res['LAST_NAME'][0] . "<br>\n";
```

A lot of the older documentation uses “&” in calls to `oci_bind_by_name()`. Don’t do this. Since the recent PHP call-by-reference clean up, this syntax has been deprecated. It may even cause you problems.

The bind data needs to be accessible when `oci_execute()` is called. Using a local variable in a sub function may cause a scope problem.

As well as “IN” binds, which pass data into Oracle, there are also “OUT” binds that return values. These are mostly used to return values from PL/SQL procedures and functions. If the PHP variable associated with an OUT bind does not exist, you need to specify the otherwise optional length parameter. Another case when the length should be specified is when returning numbers. By default in OCI8, numbers are converted to and from strings when they are bound. This means the length parameter should also be passed to `OCIBindByName()` when returning a number:.

```
oci_bind_by_name($s, ":MB", $mb, 10);
```

There is also an optional fifth parameter, which is the datatype. This mostly used for binding LOBS and result sets as shown in a later chapter.

There is one case where you might decide not to use bind variables. When queries contain bind variables, the optimizer does not have any information about the value you may eventually use when the statement is executed. If your data is highly skewed, you might want to hard code values. But if the data is derived from user input be sure to sanitize it.

## Exploring SQL

Do yourself a favor and explore SQL and PL/SQL. Make maximum reuse of functionality that already exists. Tom Kyte's popular [asktom.oracle.com](http://asktom.oracle.com) has a lot of very useful information.

Oracle’s general guideline is to let the database manage data and to transfer the minimum amount across the network. Avoid shipping data from the database to PHP for unnecessary post processing. Data is a core asset of your business. It should be treated consistently across your applications. Keeping a thin interface between your application layer and the database is also good programming practice.

Some examples of useful database features are regular expressions and analytic functions. Many more features including, [autonomous transactions](#) and spatial functionality are left for you to investigate further.

## Regular Expressions in SQL

Regular expressions were first introduced in Oracle 10g Release 10.1, and extended in Release 10.2. The following query gets both spellings of Stephen:

```
select first_name, last_name
from employees
where regexp_like(first_name, '^Ste(v|ph)en$');
```

Oracle’s regular expressions conform to POSIX regular expression standard and the Unicode Regular Expression Guidelines.

## Analytic Functions in SQL

Oracle's Analytic functions are a useful tool to compute aggregate values based on a group of rows. Here is an example of a correlation. CORR() returns the coefficient of correlation of a set of number pairs. You must be connected as the SYSTEM user in this particular example, since it depends on table data not available to HR:

```
select max_extents, corr(max_trans, initial_extent)
from all_tables
group by max_extents;
```

Other analytic functions allow you to get basic information like standard deviations or do tasks such as ranking (for Top-N or Bottom-N queries) or do linear regressions.



## CHAPTER 11

# Using PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. It is a server-side, stored procedural language that is easy-to-use, seamless with SQL, portable, and secure. PL/SQL enables you to mix SQL statements with procedural constructs. PHP can call PL/SQL blocks to make use of existing database functionality, and can use it to efficiently send and return data. You can create stored procedures, functions and packages so your applications can be extended infinitely.

## PL/SQL Overview

There are a number of pre-supplied PL/SQL packages to make application development easier. Packages exist for full text indexing, queuing, change notification, job scheduling and TCP access, just to name a few. When deciding whether to write PHP on the client or PL/SQL in the server, consider your skill levels in the languages, the cost of data transfer across the network and the re-usability of the code. If you write in PL/SQL, all your Oracle applications in any tool or client language can reuse the functionality. Some functionality should only ever be in the database, such as data change triggers. In Oracle, you can create these to be fired when an event such as a data insert or a user logon occurs.

A PL/SQL block has three basic parts:

- \* A declarative part (DECLARE)
- \* An executable part (BEGIN ... END)
- \* An exception-handling (EXCEPTION) part that handles error conditions

For example:

```
declare
  sal_1 pls_integer;
begin
  select salary into sal_1 from employees where employee_id = 191;
  dbms_output.put_line('Salary is ' || sal_1);
exception
  when no_data_found then
    dbms_output.put_line('No results returned');
end;
```

You can run this in many tools, including PHP. In Oracle's SQL\*Plus it is run by entering the text at the prompt and finishing with a single "/" to tell SQL\*Plus to run the code. If you turn on SET SERVEROUTPUT beforehand, then SQL\*Plus will display the output messages after execution. Other tools have different ways of indicating the end of the statements and how to switch serveroutput on:

```
SQL> set serveroutput on
SQL> declare
```

## Using PL/SQL

```
2   sal_1 pls_integer;
3   begin
4   select salary into sal_1
5   from employees
6   where employee_id = 191;
7   dbms_output.put_line('Salary is ' || sal_1);
8   exception
9   when no_data_found then
10      dbms_output.put_line('No results returned');
11  end;
12  /
Salary is 2500
```

Only the executable part of a PL/SQL block is required. The optional declarative part is written first, where you define types, variables, and similar items. Errors that occur during execution can be dealt with in the exception-handling part.

## Blocks, Procedures, Packages and Triggers

PL/SQL code is generally categorized as one of the following:

- \* Anonymous block
- \* Stored procedure or function
- \* Package
- \* Trigger

Procedures, functions, and variables in packages can be used from other packages, procedures, or functions.

### Anonymous Block

An anonymous block is a PL/SQL block that appears in your application and is not named or stored in the database. The previous example is an anonymous block. In many applications, PL/SQL blocks can appear wherever SQL statements can appear. A PL/SQL block groups related declarations and statements. Because these blocks are not stored in the database, they are generally for one-time use either in a script or as SQL text dynamically submitted to the Oracle server.

### Stored or Standalone Procedure and Function

A stored procedure or function is a PL/SQL block that Oracle stores in the database and can be called by name from an application. Functions return a value when executed. When you create a stored procedure or function, Oracle stores its parsed representation in the database for efficient reuse. Procedures can be created in SQL\*Plus like:

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

6 /

If you have errors creating a PL/SQL block, use the SQL\*Plus SHOW ERRORS command to display any error messages

## Package

Typically, stored procedures and functions are encapsulated into packages. This helps minimize recompilation of dependent objects when a body of code changes. The package specification defines the signatures of the functions and procedures. If that definition is unchanged, code that invokes it will not need to be recompiled.

```
SQL> create or replace package mypack as
  2   function myfunc(i_p in number) return varchar2;
  3   end mypack;
  4   /
```

Package created.

```
SQL> create or replace package body mypack as
  2   function myfunc(i_p in number) return varchar2 as
  3   d_l varchar2(20);
  4   begin
  5       select mydata into d_l from mytab where myid = i_p;
  6       return d_l;
  7   end;
  8 end mypack;
  9   /
```

Package body created.

## Trigger

A database trigger is a stored procedure associated with a database table, view, or event. The trigger can be called after the event, to record it, or take some follow-up action. The trigger can be called before the event, to prevent erroneous operations or fix new data so that it conforms to business rules. An example of a trigger was shown in the previous chapter, *Executing SQL Statements with OCI8*, under the heading *Auto-Increment Columns*.

## Calling PL/SQL Procedures and Functions

To invoke a previously created PL/SQL procedure, use the SQL CALL statement like:

```
<?php
  $c = oci_connect('hr', 'hr', '//localhost/XE');
  $s = oci_parse($c, "call myproc('mydata', 123)");
  oci_execute($s);
?>
```

You can also use an anonymous block. In this example the block contains a single procedure call but you could include any number of other PL/SQL statements.

```
<?php
```

## Using PL/SQL

```
$c = oci_connect('hr', 'hr', '//localhost/XE');
$s = oci_parse($c, "begin myproc('mydata', 456); end;");
oci_execute($s);
?>
```

Note there is a semi-colon after “end”, which is different to the way SQL statements are written.

Calling a PL/SQL function needs a bind variable to hold the return value:

```
<?php
$c = oci_connect('hr', 'hr', '//localhost/XE');
$s = oci_parse($c,
    "begin :r := mypack.myfunc(123); end;");
oci_bind_by_name($s, ':r', $r, 20);
oci_execute($s);
echo "Name is: ".$r;
?>
```

This example also shows how to call a function inside a package.

There is extensive Oracle documentation on PL/SQL. The standard 3<sup>rd</sup> party text on PL/SQL is [Oracle PL/SQL Programming](#), an O’Reilly book by Steven Feuerstein.

## Getting output from PL/SQL with DBMS\_OUTPUT

PL/SQL executes in the database, and there is no way to prompt for input or give asynchronous output to the client host while it executes.

In many cases the synchronous output of the DBMS\_OUTPUT package is very practical for reporting or debugging. Other PL/SQL applications get output by logging messages to database tables or use packages like UTL\_FILE and DBMS\_PIPE to asynchronously display output to separate terminal windows. With DBMS\_OUTPUT, client tools must explicitly fetch the script output after the PL/SQL application completes.

In Oracle 10g Release 10.2, size limitations on DBMS\_OUTPUT were lifted and you are likely to see it being used more frequently for output. In Release 10.2, the maximum line length is 32K bytes and a theoretically infinite number of lines can be returned.

This is the sample DBMS\_OUTPUT code that Oracle ships with the Oracle JDeveloper PHP Extension. It uses PHP4-style OCI8 function calls, letting the code work in PHP4 and PHP5. It binds parameters for the DBMS\_OUTPUT procedure calls.

```
/**
 * Enable or disable DBMS_OUTPUT
 *
 * Turning DBMS_OUTPUT on tells the Oracle Server to buffer
 * DBMS_OUTPUT calls. After a SQL statement or PL/SQL block
 * completes {@link DbmsOutputFetch()} needs to be called to
 * fetch and display the output.
 *
 * @return bool
 * @param resource $con the connection from OCILogon
 * @param bool $p optional flag. Pass true if DBMS_OUTPUT
 * should be enabled in the Oracle Server and false if it
 * should be disabled.
 */
function DbmsOutputSet($con, $p = true)
```



```

{
  if ($p)
    $stmt = 'BEGIN DBMS_OUTPUT.ENABLE(NULL); END;';
  else
    $stmt = 'BEGIN DBMS_OUTPUT.DISABLE(); END;';

  if (!$con)
    return(false);

  $r = false;
  $stid = @OCIParse($con, $stmt);
  if ($stid) {
    $r = @OCIExecute($stid);
    @OCIFreeStatement($stid);
  }
  return $r;
}

/**
 * Fetch DBMS_OUTPUT and return it as an array of lines
 *
 * If no DBMS_OUTPUT is available, then the array contains
 * the single entry "No DBMS_OUTPUT". DBMS_OUTPUT will
 * be available only if {@link DbmsOutputSet()} has
 * been called.
 *
 * @return mixed array of DBMS_OUTPUT lines, or false on
 *         error
 * @param resource $con the connection from OCILogon
 */
function DbmsOutputFetch($con)
{
  $res = false;

  if ($con) {
    $stid = @OCIParse($con,
      'BEGIN DBMS_OUTPUT.GET_LINE(:LN, :ST); END;');
    if (!$stid) {
      DisplayOCIErr($con);
    }
    else {
      if (!@OCIBindByName($stid, ':LN', $ln, 255) ||
        !@OCIBindByName($stid, ':ST', $st, 1)) {
        DisplayOCIErr($stid);
      }
      else {
        $res = array();
        for ($cnt = 0;
          ($succ = @OCIExecute($stid)) && !$st; $cnt++) {
          $res[] = $ln;
        }
        if ($succ && $cnt == 0)
          $res[] = 'No DBMS_OUTPUT';
        if (!$succ) {

```

## Using PL/SQL

```
        DisplayOCIErrror($stid);
        $res = false;
    }
}
@OCIFreeStatement($stid);
}
}
return ($res);
}
```

The `DisplayOCIErrror()` function was given in the previous chapter, *Executing SQL Statements With OCI8*, under the heading *Handling Errors*.

The `:LN` line variable is arbitrarily bound as size 255. This was the limit prior to Oracle 10g Release 10.2. You really don't want every single application to be binding 32K, especially if the database is running in Oracle's multi-threaded server (MTS) mode. If you bind 32K, then Oracle has to ship 32K buffers around and it is easy to slow performance or get memory errors.

To use the `DbmsOutputFetch()`, enable buffering before executing your PL/SQL block. Only when `oci_execute()` returns can the output be fetched.

```
$c = oci_connect("hr", "hr", "//localhost/XE");
DbmsOutputSet($c, true);
$s = oci_parse($c,
    "begin dbms_output.put_line('Hello, world!'); end;");
oci_execute($s);

$output = DbmsOutputFetch($c); // now we can get output
foreach ($output as $line)
    echo "$line<br>";
```

If you expect large amounts of output, you may want to write your own code to stream results as they are fetched from the database instead of returning them in one array from `DbmsOutputFetch()`.

## Oracle Collections in PHP

Many programming techniques use collection types such as arrays, bags, lists, nested tables, sets, and trees. To support these techniques in database applications, PL/SQL provides the datatypes `TABLE` and `VARRAY`, which allow you to declare index-by tables, nested tables, and variable-size arrays.

A collection is an ordered group of elements, all of the same type. Each element has a unique subscript that determines its position in the collection. Collections work like the arrays found in most third-generation programming languages. Also, collections can be passed as parameters. So, you can use them to move columns of data into and out of database tables or between client-side applications and stored subprograms.

The Oracle manual says:

*“A collection is an ordered group of elements, all of the same type.”*

Collections are effectively arrays. Similar to LOBs, collections are manipulated by methods on a collection resource, which is allocated with `oci_new_collection()`.

In a simple email address book demonstration (created by Charles Poulsen from Oracle), two VARRAYs are created, one for an array of people's names, and one for an array of email addresses. VARRAYs (short for variable-size arrays), use sequential numbers as subscripts to access a fixed number of elements.

```
SQL> drop table emails;

SQL> create table emails (
    user_id      varchar2(10),
    friend_name  varchar2(20),
    email_address varchar2(20));

SQL> create or replace type email_array as
    varray(100) of varchar2(20);
/

SQL> create or replace type friend_array as
    varray(100) of varchar2(20);
/

SQL> create or replace procedure update_address_book(
    p_user_id      in varchar2,
    p_friend_name  friend_array,
    p_email_addresses email_array)
is
begin
    delete from emails where user_id = p_user_id;
    for i in 1 .. p_email_addresses.count loop
        insert into emails
            (user_id, friend_name, email_address)
            values (p_user_id, p_friend_name(i),
                p_email_addresses(i));
    end loop;
end update_address_book;
/
```

The `update_address_book()` procedure loops over all elements of the address collection and inserts each one.

In PHP we create collection variables and use the `append()` method to add elements to each array. By binding as `OCI_B_NTY` (“Named Type”) we can pass collections to the PL/SQL procedure arguments. The following PHP code creates a collection of names and a collection of email addresses. These collections are bound to the arguments of the PL/SQL `address_book()` call. When this executes, the names and email addresses are inserted into the database.

```
$user_name      = 'cjones';
$friends_names = array('alison', 'aslam');
$friends_emails = array('alison@example.com', 'aslam@example.com');

$friend_coll = oci_new_collection($c, 'FRIEND_ARRAY');
$email_coll  = oci_new_collection($c, 'EMAIL_ARRAY');

for ($i=0; $i < count($friends_names); $i++) {
    $friend_coll->append($friends_names[$i]);
```

## Using PL/SQL

```
$email_coll->append($friends_emails[$i]);
}

$s = oci_parse($c,
    "begin update_address_book(:name, :friends, :emails); end;");

oci_bind_by_name($s, ':name', $user_name);
oci_bind_by_name($s, ':friends', $friend_coll, -1,
    OCI_B_NTY);
oci_bind_by_name($s, ':emails', $email_coll, -1,
    OCI_B_NTY);

oci_execute($s);
```

Other PHP collection methods allow accessing or copying data in the collection.

## Using PL/SQL Types in PHP

Sometime you have to call PL/SQL procedures that are designed for interacting with other PL/SQL code. For example, the Oracle Text CTX\_THES package procedures return Oracle PL/SQL types that are not easily accessible in OCI8. PL/SQL types differ from SQL types that are created with the CREATE TYPE command. Oracle Text is described in the Oracle marketing datasheets like this:

*“Oracle Text uses standard SQL to index, search, and analyze text and documents stored in the Oracle database, in files, and on the web. Oracle Text can perform linguistic analysis on documents, as well as search text using a variety of strategies including keyword searching, context queries, Boolean operations, pattern matching, mixed thematic queries, HTML/XML section searching, and so on. It can render search results in various formats including unformatted text, HTML with term highlighting, and original document format. Oracle Text supports multiple languages and uses advanced relevance-ranking technology to improve search quality. Oracle Text also offers advanced features like classification, clustering, and support for information visualization metaphors.”*

Let's set up an example package with a similar interface to CTX\_THES. This example, *ctx.sql*, just returns random data in the parameter:

```
-- Package "SuppliedPkg" simulates Oracle Text's CTX_THES.
-- It has a procedure that returns a PL/SQL type.
create or replace package SuppliedPkg as
    type SuppliedRec is record (
        id    number,
        data  varchar2(100)
    );
    type SuppliedTabType is table of SuppliedRec
        index by binary_integer;
    procedure SuppliedProc(p_p in out nocopy SuppliedTabType);
end SuppliedPkg;
/
```

```

create or replace package body SuppliedPkg as
  procedure SuppliedProc(p_p in out nocopy SuppliedTabType) is
  begin
    -- Create some random results
    p_p.delete;
    for i in 1..5 loop
      p_p(i).id := i;
      p_p(i).data := 'Random: ' || i ||
        (1+ABS(MOD(dbms_random.random,100000)));
    end loop;
  end SuppliedProc;
end SuppliedPkg;
/

```

Run the file *ctx.sql* in SQL\*Plus:

```
$ sqlplus hr/hr@//localhost/XE @ctx.sql
```

This is the “fixed” part of the problem, representing the pre-supplied functionality. It seems impossible to call `SuppliedProc()` and return its data to PHP. Since you can’t change `SuppliedProc()`, you can create a helper function in PL/SQL to convert the PL/SQL type `SuppliedTabType` to a pair of SQL types. Create *myproc.sql* and use SQL\*Plus to run it like the previous script:

```

-- Create a wrapper procedure that calls the pre-supplied
-- SuppliedProc() and converts its output to SQL types.

create or replace type MyIdRec as table of number;
/
create or replace type MyDataRec as table of varchar2(100);
/

create or replace procedure MyProc
  (p_id IN OUT MyIdRec, p_data IN OUT MyDataRec)
as
  l_results SuppliedPkg.SuppliedTabType;
begin

  -- get results from existing procedure
  SuppliedPkg.SuppliedProc(l_results);

  -- copy to a type we can pass back to PHP
  p_id.delete;
  p_data.delete;
  for i in 1..l_results.count loop
    p_id.extend;
    p_id(i) := l_results(i).id;
    p_data.extend;
    p_data(i) := l_results(i).data;
  end loop;

end MyProc;
/

```

Now you can call `MyProc()` from within *ctx.php*:

## Using PL/SQL

```
<?php
$c = oci_connect("hr", "hr", "//localhost/XE");
$s = oci_parse($c, 'begin MyProc(:res_id, :res_data); end;');
$res_id = oci_new_collection($c, 'MYIDREC');
$res_data = oci_new_collection($c, 'MYDATAREC');
oci_bind_by_name($s, ':res_id', $res_id, -1, OCI_B_NTNY);
oci_bind_by_name($s, ':res_data', $res_data, -1, OCI_B_NTNY);
oci_execute($s);
for ($i = 0; $i < $res_id->size(); $i++) {
    $id = $res_id->getElem($i);
    $data = $res_data->getElem($i);
    echo "Id: $id, Data: $data<br>";
}
?>
```

This allocates two collections and binds them as the parameters to MyProc(). After MyProc() has been called, the collection method getElem() is used to access each value returned.

## New Array Binding Function in PHP 5.1.2

PHP 5.1.2 improved collection support with a new function oci\_bind\_array\_by\_name(). Coupled with a helper PL/SQL function, this can be very efficient for insertion. We can bind a PHP array containing all data and send it to the database with a single oci\_execute().

```
SQL> drop table mytab;
SQL> create table mytab(name varchar2(20));

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

SQL> create or replace package body mypkg as
    cursor cur is select name from mytab;
    procedure myproc(p1 in out arrtype) is
    begin
        for i in 1 .. p1.count loop
            insert into mytab values (p1(i));
        end loop;
    end myproc;
end mypkg;
/
```

To insert a PHP array into MYTAB, use:

```
$s = oci_parse($c, "BEGIN mypkg.myproc(:c1); END;");
$array = array("abc", "def", "ghi", "jkl", "mno");
oci_bind_array_by_name($s, ":c1", $array, 5, -1, SQLT_CHR);
oci_execute($s, OCI_DEFAULT);
oci_commit();
```

The `oci_bind_array_by_name()` function is similar to `oci_bind_by_name()`. It takes not just the upper data length, but also the number of elements in the array. In this example, the number of elements is 5, and the data length is given as -1, meaning use the actual length of the character data. The inserted data is:

```
SQL> select * from mytab;
```

```
NAME
-----
abc
def
ghi
jkl
mno
```

A number of other Oracle types can be bound. A source of information is from reading the automated tests under the OCI8 tree in CVS. These are also bundled with the PHP source code.

## Using REF CURSORS for Result Sets

REF CURSORS let you return a set of query results to PHP. You can store a set of query results in a variable. In PHP you bind an OCI\_B\_CURSOR variable to a PL/SQL procedure call, and retrieve the rows of its result set in a fetch loop.

As an example, we create a PL/SQL package with a procedure that accepts a job identifier and queries the employees who perform that job. The procedure returns a REF CURSOR containing the employee identifiers and salaries.

In the package specification, the REF CURSOR type and signature of the procedure is given. The package body contains the code of the function.

```
SQL> CREATE OR REPLACE PACKAGE cv_types AS
  TYPE EmpInfoTyp is REF CURSOR;
  PROCEDURE EmpInfoRpt (jid IN VARCHAR, emp_cv IN OUT EmpInfoTyp);
END cv_types;
/
```

```
SQL> CREATE OR REPLACE PACKAGE BODY cv_types AS
PROCEDURE EmpInfoRpt (jid IN VARCHAR, emp_cv IN OUT EmpInfoTyp) AS
  BEGIN
    OPEN emp_cv FOR SELECT EMPLOYEE_ID, SALARY
                    FROM EMP_DETAILS_VIEW WHERE JOB_ID=jid;
  END;
END;
/
```

In PHP we would use this PL/SQL procedure like:

```
$job_id = 'SA_MAN';
$stmt = "BEGIN cv_types.EmpInfoRpt(:jid, :rc); END;";

$s = oci_parse($c, $stmt);
oci_bind_by_name($s, ':jid', $job_id);
```

## Using PL/SQL

```
$refcur = oci_new_cursor($c); // pass the connection
oci_bind_by_name($s, ':rc', $refcur, -1, OCI_B_CURSOR);

// Execute the call to EmpInfoRpt()
oci_execute($s);

// Execute and fetch from the cursor
oci_execute($refcur);
echo '<table border="1">';
while($row = oci_fetch_assoc($refcur)) {
    echo '<tr>';
    foreach ($row as $c) {
        echo "<td>$c</td>";
    }
    echo '</tr>';
}
echo '</table>';
```

This shows that \$refcur is already a prepared statement when it is fetched. It is simply executed like a normal query.



## CHAPTER 12

# Using Large Objects in OCI8

Oracle Character Large Object (CLOB) and Binary Large Object (BLOB) types can contain very large amounts of data. They can be used for table columns and PL/SQL variables. There are various ways to specify them for optimal Oracle storage. A pre-supplied DBMS\_LOB package makes manipulation in PL/SQL easy.

Oracle also has a BFILE type for large objects stored outside the database.

## Working with LOBs

In PHP, LOBs are manipulated using a descriptor. To show this, in SQL\*Plus create a table that has a BLOB column:

```
SQL>create table mybtab (blobid number, blobdata blob);
```

PHP code to insert data into this table is:

```
$myblobid = 123;
$myv = 'a very large amount of binary data';

$llob = oci_new_descriptor($c, OCI_D_LOB);
$s = oci_parse($c,
    'INSERT INTO mybtab (blobid, blobdata) '
    . 'VALUES(:myblobid, EMPTY_BLOB()) '
    . 'RETURNING blobdata INTO :blobdata');
oci_bind_by_name($s, ':MYBLOBID', $myblobid);
oci_bind_by_name($s, ':BLOBDATA', $llob, -1, OCI_B_BLOB);
oci_execute($s, OCI_DEFAULT);

$llob->save($myv);
oci_commit($c);
```

The RETURNING clause returns the Oracle LOB locator of the new row. By binding as OCI\_B\_BLOB, the PHP descriptor in \$llob references this locator. The \$llob->save() method then stores the data in \$myv into the BLOB column. The OCI\_DEFAULT flag is used for oci\_execute() so the descriptor remains valid for the save method. The commit concludes the insert and makes the data visible to other database users.

If the application uploads LOB data using a web form, it can be inserted directly from the upload directory with \$llob->import(\$filename). PHP's maximum allowed size for uploaded files is set in *php.ini* using the upload\_max\_filesize parameter.

When fetching a LOB, OCI8 returns the LOB descriptor, and the data is retrieved by using a load() or read() method:

```
$query = 'SELECT blobdata FROM mybtab WHERE blobid = 123';
$stmt = oci_parse($c, $query);
oci_execute($stmt);
$arr = oci_fetch_assoc($stmt);
```

```
$data = $arr['BLOBDATA']->load();
```

The PHP manual recommends using `read()` instead of `load()` because it allows the data size of data to be limited. This can help avoid the PHP process abruptly terminating when it reaches its allocated maximum memory size, set with `memory_limit` in `php.ini`.

Using CLOBs is almost identical to using BLOBs. The bind type becomes `OCI_B_CLOB`, and the table must obviously contain a CLOB column.

## Other LOB Methods

A number of other methods on the LOB descriptor allow seeking to a specified offset, exporting data directly to file, erasing data, and copying or comparing a LOB.

This code snippet shows seeking to the 10<sup>th</sup> position in the result descriptor, and then storing the next 50 bytes in `$result`:

```
$arr['BLOBDATA']->seek(10, OCI_SEEK_SET);  
$result = $arr['BLOBDATA']->read(50);
```

The full list of LOB methods is:

- `OCI-Lob->close`: Close a LOB descriptor
- `OCI-Lob->eof`: Test for LOB end-of-file
- `OCI-Lob->erase`: Erases a specified part of the LOB
- `OCI-Lob->export`, `OCI-Lob->writeToFile`: Write a LOB to a file
- `OCI-Lob->flush`: Flushes buffer of the LOB to the server
- `OCI-Lob->free`: Frees resources associated with the LOB
- `OCI-Lob->getBuffering`: Returns current state of buffering for the LOB
- `OCI-Lob->import`, `OCI-Lob->saveFile`: Loads data from a file to a LOB
- `OCI-Lob->load`: Returns LOB contents
- `OCI-Lob->read`: Returns part of the LOB
- `OCI-Lob->rewind`: Moves the LOB's internal pointer back to the beginning
- `OCI-Lob->save`: Saves data to the LOB
- `OCI-Lob->seek`: Sets the LOB's internal position pointer
- `OCI-Lob->setBuffering`: Changes LOB's current state of buffering
- `OCI-Lob->size`: Returns size of LOB
- `OCI-Lob->tell`: Returns current pointer position
- `OCI-Lob->truncate`: Truncates a LOB
- `OCI-Lob->write`: Writes data to the LOB
- `OCI-Lob->writeTemporary`: Writes a temporary LOB

Several PHP functions also work on LOB descriptors:

- `oci_lob_copy`: Copies a LOB
- `oci_lob_is_equal`: Compare two LOB locators for equality

Check the PHP manual for usage.

## Working with BFILES

A BFILE is an Oracle large object (LOB) datatype for files stored outside the database. BFILES are a handy way for using relatively static, externally created content. They are also useful for loading text or binary data into Oracle tables.

In SQL and PL/SQL, a BFILE is accessed via a locator, which is simply a pointer to the external file. There are numerous pre-supplied functions that operate on BFILE locators.

To show how BFILEs work in PHP this section creates a sample application that accesses and displays a single image. The image will not be loaded into the database but the picture description is loaded so it can be queried. The BFILE allows the image to be related to the description. Also the application could be extended in future to use PL/SQL packages to read and manipulate the image.

In this example, the image data is not loaded and printed in PHP. Instead, the browser is redirected to the image URL of the external file. This significantly reduces the amount of data that needs to be handled by the application.

To allow Apache to serve the image, edit `httpd.conf` and map a URL to the directory containing the file. For example if the file is `/tmp/cj.jpg` add:

```
Alias /tmp/ "/tmp/"
<Directory "/tmp/">
    Options None
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

(Using `/tmp` like this isn't recommended for anything except testing!) Restart Apache and use a browser to check that `http://localhost/tmp/cj.jpg` loads the picture in `/tmp/cj.jpg`.

In Oracle, create a DIRECTORY alias for `/tmp`. This is Oracle's pointer to the operating system and forms part of each BFILE. The directory must be on the same machine that the database server runs on. In SQL\*Plus run:

```
SQL> connect system/yourpassword@//localhost/XE
SQL> create directory TestDir AS '/tmp';
SQL> grant read on directory TestDir to hr;
SQL> connect hr/hr@//localhost/XE
SQL> create table FileTest (
            FileNum number primary key,
            FileDesc varchar2(30),
            Image bfile);
```

This gives the HR user access to the `/tmp` directory and creates a table `FileTest` containing a file number identifier, a text description of the file, and the BFILE itself. The image data is not loaded into this table; the BFILE in the table is a pointer to the file on your filesystem.

PHP code to insert the image name into `FileTest` looks like:

```
<?php
    $fnum = 1;
    $fdsc = "Some description to search";
    $name = "cj.jpg";
    $c = oci_connect("hr", "hr", "//localhost/XE");
    $s = oci_parse($c,
        "insert into FileTest (FileNum, FileDesc, Image)
        values (:fnum, :fdsc, bfilename('TESTDIR', :name))");
    oci_bind_by_name($s, ":fnum", $fnum, -1, SQLT_INT);
    oci_bind_by_name($s, ":fdsc", $fdsc, -1, SQLT_CHR);
    oci_bind_by_name($s, ":name", $name, -1, SQLT_CHR);
    oci_execute($s, OCI_DEFAULT);
```

## Using Large Objects in OCI8

```
oci_commit($c);
?>
```

The `bfilename()` constructor inserts into the BFILE-type column using the TESTDIR directory alias created earlier. Bind variables are used for efficiency and security.

This new BFILE can be queried back in PHP:

```
<?php
$c = oci_connect("hr", "hr", "//localhost/XE");
$s = oci_parse($c,
    "select Image from FileTest where FileNum = :fnum");
$fnum = 1;
oci_bind_by_name($s, ":fnum", $fnum);
oci_execute($s);
$row = oci_fetch_assoc($s);
$bf = $row['IMAGE']; // This is a BFILE descriptor
echo "<pre>"; var_dump($bf); echo "</pre>";
?>
```

For simplicity, the query condition is the file number of the new record. In real life it might use a regular expression on the FileDesc field like:

```
select Image from FileTest
where regexp_like(FileDesc, 'somepattern')
```

The PHP script returns a BFILE descriptor. Now what? In this example the file name is needed so the browser can redirect to a page showing the image. Unfortunately there is no direct method in PHP to get the filename from the descriptor. But if you have (or create) an Oracle procedure you could pass it the BFILE descriptor and get the name.

Instead of executing the query in PHP and using PL/SQL to find the filename, a more efficient method is to create a PL/SQL anonymous block that does both the query and extracts the filename from its returned BFILE. The previous PHP query code can be replaced with:

```
<?php
$c = oci_connect("hr", "hr", "//localhost/XE");
$s = oci_parse($c,
    'declare '
    . ' b_l bfile;'
    . ' da_l varchar2(255);'
    . 'begin '
    . 'select image into b_l from filetest where filenum = :fnum;'
    . 'dbms_lob.filegetname(b_l, da_l, :name);'
    . 'end;');
$fnum = 1;
oci_bind_by_name($s, ":fnum", $fnum);
oci_bind_by_name($s, ":name", $name, 255, SQLT_CHR);
oci_execute($s);

header("Location: http://localhost/tmp/$name");
?>
```

The filename `cj.jpg` is returned in `$name` courtesy of the `:name` bind variable argument to the `DBMS_LOB.FILEGETNAME()` function. The `header()` function redirects the user to the image. If you have problems, comment out the `header()` call and check that `$name` is valid. If

any text is printed before the header() is output, the HTTP headers will not be correct and the image will not display.

The application can be improved by putting the PL/SQL into a stored procedure. This lets Oracle parse and optimize the code at creation time. Also, if you're concerned about protecting intellectual property, you can obfuscate the code with Oracle's "wrap" utility.

To create a PL/SQL procedure MyGetFileName() in SQL\*Plus:

```
SQL> connect hr/hr@//localhost/XE
SQL> create or replace procedure MyGetFileName
  (fnum_p in number, name_p out varchar2) as
  bf_l bfile;
  da_l varchar2(255);
begin
  select image into bf_l from FileTest where FileNum = fnum_p;
  dbms_lob.filegetname(bf_l, da_l, name_p);
end;
/
```

Back in PHP the filename is easily fetched from the database with an anonymous block. The browser is redirected to the image similar to the previous example:

```
<?php
  $c = oci_connect("hr", "hr", "//localhost/XE");
  $s = oci_parse($c, "begin MyGetFileName(:fnum, :name); end;");
  $fnum = 1;
  oci_bind_by_name($s, ":fnum", $fnum);
  oci_bind_by_name($s, ":name", $name, 255, SQLT_CHR);
  oci_execute($s);
  header("Location: http://localhost/tmp/$name");
?>
```

BFILES are easy to work with in PL/SQL because the presupplied DBMS\_LOB package has a number of useful functions. For example DBMS\_LOB.LOADFROMFILE() reads BFILE data from the file system into a PL/SQL BLOB or CLOB. This could be loaded into a BLOB table column, manipulated in PL/SQL, or even returned to PHP using oci8's LOB features. Another example is DBMS\_LOB.FILEEXISTS(), which can be used to check whether the FileTest table contains references to images that do not exist.

BFILES are very useful for many purposes including loading images into the database, but BLOBs may be better in some circumstances. Changes to BFILE locators can be rolled back or committed but since the files themselves are outside the database, BFILE data does not participate in transactions. You can have dangling references to BFILES because Oracle doesn't check the validity of BFILES until the data is explicitly read (this allows you to pre-create BFILES or to change the physical data on disk). BFILE data files are read-only and cannot be changed within Oracle. Finally, BFILES need to be backed up manually. Because of these points, there might be times you should use BLOBs to store images inside the database to ensure data and application consistency but BFILES are there if you want them.



## CHAPTER 13

# Using XML with Oracle and PHP

Both Oracle and PHP5 have excellent XML capabilities, allowing lots of scope for processing information. All editions of Oracle contain what is known as “XML DB”, the XML capabilities of the database. When tables are created, XML can be stored in linear LOB format, or according to the structure of your XML schema.

This chapter covers the basics of returning XML data from Oracle to PHP. There is a good article *Using PHP5 with Oracle XML DB* by Yuli Vasiliev in the July/August 2005 Oracle Magazine that discusses it more and covers PHP’s XML functionality.

## Fetching Relational Rows as XML

One useful feature of XDB is that relational SQL tables can automatically be retrieved as XML.

```
$query =
'SELECT XMLELEMENT("Employees",
  XMLELEMENT("Name", employees.last_name),
  XMLELEMENT("Id", employees.employee_id)) as result
FROM employees
WHERE employee_id > 200';

$s = oci_parse($c, $query);
oci_execute($s);
while ($row = oci_fetch_row($s))
  foreach ($row as $item)
    echo htmlentities($item)." ";
```

The output is:

```
<Employees><Name>Hartstein</Name><Id>201</Id></Employees>
<Employees><Name>Fay</Name><Id>202</Id></Employees>
<Employees><Name>Mavris</Name><Id>203</Id></Employees>
<Employees><Name>Baer</Name><Id>204</Id></Employees>
<Employees><Name>Higgins</Name><Id>205</Id></Employees>
<Employees><Name>Gietz</Name><Id>206</Id></Employees>
```

---

Tip: Watch out for the quoting of XML queries. The XML functions can have embedded double-quotes. This is the exact opposite of standard SQL queries, which can have embedded single quotes.

---

There are a number of other XML functions that can be similarly used. See the Oracle Database SQL Reference.

## Using DBMS\_XMLGEN

Another way to create XML from relational data is to use the PL/SQL package DBMS\_XMLGEN(). Queries that use DBMS\_XMLGEN() return a CLOB column, so the initial result needs to be treated in PHP as a LOB descriptor. There is effectively no length limit for CLOBs. The following example queries the first name of employees in department 30 and stores the XML marked-up output in \$mylob:

```
$query = "select dbms_xmlgen.getxml('
        select first_name
        from employees
        where department_id = 30') xml
        from dual";

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

The value in \$mylob is:

```
<?xml version="1.0"?>
<ROWSET>
  <ROW>
    <FIRST_NAME>Den</FIRST_NAME>
  </ROW>
  <ROW>
    <FIRST_NAME>Alexander</FIRST_NAME>
  </ROW>
  <ROW>
    <FIRST_NAME>Shelli</FIRST_NAME>
  </ROW>
  <ROW>
    <FIRST_NAME>Sigal</FIRST_NAME>
  </ROW>
  <ROW>
    <FIRST_NAME>Guy</FIRST_NAME>
  </ROW>
  <ROW>
    <FIRST_NAME>Karen</FIRST_NAME>
  </ROW>
</ROWSET>
```

## Accessing Data from Oracle over HTTP

XML DB allows you to access data directly via HTTP, FTP or WebDAV. The Oracle Network listener will handle all these requests. As an example of HTTP, use the PL/SQL DBMS\_XDB package to create a resource, which here is just some simple text:

```
SQL> declare
```



```

    res boolean;
begin
  begin
    -- delete if it already exists
    dbms_xdb.deleteResource('/public/test1.txt');
  exception
  when others then
    null;
  end;
  res := dbms_xdb.createResource('/public/test1.txt',
                                'the text to store');
end;
/

```

For testing only, remove access control on the public resource:

```

SQL> connect system/yourpassword
SQL> alter user anonymous identified by anonymous account unlock;

```

Sometimes it takes a few minutes for this all to take effect, but the file can now be accessed from a browser (or PHP application) using:

<http://localhost:8080/public/test1.php>

There is extensive Oracle documentation on XDB.

## XQuery XML Query Language

Oracle's support for XQuery was introduced in Oracle 10 Release 2. Unfortunately, to keep the footprint of Oracle XE small, XQuery is not available there, but it is in the other editions of Oracle.

A basic XQuery to return the records in the EMPLOYEES table is:

```
for $i in ora:view("employees") return $i
```

This XQuery syntax is in a special SELECT:

```

SELECT COLUMN_VALUE
FROM XMLTABLE('for $i in ora:view("employees") return $i')

```

The different quoting styles used by SQL and XQuery needs careful attention in PHP. It can be coded:

```

<?php
    $c = oci_connect("hr", "hr", "://localhost/XE");

    $xq = 'for $i in ora:view("employees") return $i';
    $query = 'SELECT COLUMN_VALUE FROM XMLTABLE(\''.$xq.'\')';

    $s = oci_parse($c, $query);
    oci_execute($s);
    while ($row = oci_fetch_row($s))
        foreach ($row as $item)
            echo htmlentities($item)." ";
?>

```

## Using XML with Oracle and PHP

Table rows are automatically wrapped in tags and returned:

```
<ROW>
  <EMPLOYEE_ID>100</EMPLOYEE_ID>
  <FIRST_NAME>Steven</FIRST_NAME>
  <LAST_NAME>King</LAST_NAME>
  <EMAIL>SKING</EMAIL>
  <PHONE_NUMBER>515.123.4567</PHONE_NUMBER>
  <HIRE_DATE>1987-06-17</HIRE_DATE>
  <JOB_ID>AD_PRES</JOB_ID>
  <SALARY>24000</SALARY>
  <DEPARTMENT_ID>90</DEPARTMENT_ID>
</ROW>
...
<ROW>
  <EMPLOYEE_ID>206</EMPLOYEE_ID>
  <FIRST_NAME>William</FIRST_NAME>
  <LAST_NAME>Gietz</LAST_NAME>
  <EMAIL>WGIETZ</EMAIL>
  <PHONE_NUMBER>515.123.8181</PHONE_NUMBER>
  <HIRE_DATE>1994-06-07</HIRE_DATE>
  <JOB_ID>AC_ACCOUNT</JOB_ID>
  <SALARY>8300</SALARY>
  <MANAGER_ID>205</MANAGER_ID>
  <DEPARTMENT_ID>110</DEPARTMENT_ID>
</ROW>
```

You can also use RETURNING CONTENT to return a single document node:

```
$xq = 'for $i in ora:view("hr","locations")/ROW return $i/CITY';
$query = 'SELECT XMLQUERY(\''.$xq.'\' RETURNING CONTENT) FROM DUAL';
```

Since this result is treated as a single SQL query row, there are restrictions on its length.

## CHAPTER 14

# Globalization

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

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

## Establishing the Environment Between Oracle and PHP

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

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

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

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

```
GERMAN_GERMANY.AL32UTF8
```

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

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

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

```
export NLS_LANG GERMAN_GERMANY.AL32UTF8
```

Some globalization values can be changed per connection.

```
$s = oci_parse($c,  
    "alter session set nls_territory=germany nls_language=german");
```

## Glossary

```
oci_execute($s);
```

After executing this, Oracle error messages will be in German and the default date format will have changed.

---

Caution: Be careful changing the globalization settings when using a persistent connection. The next time the connection is used, the altered values will still be in effect.

---

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

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

You must restart the Web listener to implement the change.

## Manipulating Strings

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

Navigate to the Extensions sub tab and expand the Zend Core Extensions tree control.

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

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

For more information, see the PHP `mbstring` reference manual at <http://www.php.net/mbstring>.

## Determining the Locale of the User

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

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

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

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

## Developing Locale Awareness

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

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

Table 14-1 shows how some commonly used locales are defined in ISO and Oracle 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

Locale	Locale ID	NLS_LANGUAGE	NLS_TERRITORY
Portuguese (Brazil)	pt-BR	BRAZILIAN PORTUGUESE	BRAZIL
Portuguese	pt	PORTUGUESE	PORTUGAL
Spanish	es	SPANISH	SPAIN

*Table 14-1 Locale Representations in ISO, SQL, and PL/SQL Programming Environments*

## 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 (Internet Assigned Numbers Authority) names for the character encodings that the browser supports.

## Specifying the Page Encoding in PHP

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

```
default_charset = UTF-8
```

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

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

## Organizing the Content of HTML Pages for Translation

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

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

## Strings in PHP

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

## Static Files

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

## Data from the Database

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

## Presenting Data Using Conventions Expected by the User

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

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

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

### Oracle Date Formats

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

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

Session altered.

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

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

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

Session altered.

```
SQL> select employee_id EmpID,
```



```

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

```

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

## Oracle Number Formats

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

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

Session altered.

```

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

```

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

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

Session altered.

```

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

```

EMPID	EmpName	Salary
100	S.King	24.000,00
101	N.Kochhar	17.000,00
102	L.De Haan	17.000,00
103	A.Hunold	9.000,00

## Oracle Linguistic Sorts

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

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

Session altered.

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

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

6 rows selected.

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

Session altered.

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

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

6 rows selected.

## Oracle Error Messages

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

Consider the following server message:

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

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

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

For more discussion of globalization support features in Oracle Database Express Edition, see *[Working in a Global Environment](#)* in the *[Oracle Database Express Edition 2 Day Developer Guide](#)*.



## CHAPTER 15

# Resources

This chapter gives links to documentation, resources and articles discussed in this book, and to other web sites of interest.

## General Information and Forums

- \* [PHP Developer Center on Oracle Technology Network \(OTN\)](http://www.oracle.com/technology/tech/php/index.html)  
<http://www.oracle.com/technology/tech/php/index.html>
- \* [OTN PHP Discussion Forum](http://www.oracle.com/technology/forums/php.html)  
<http://www.oracle.com/technology/forums/php.html>
- \* [Blog: Christopher Jones on OPAL](http://blogs.oracle.com/opal/)  
<http://blogs.oracle.com/opal/>
- \* [Blog: Alison Holloway on PHP](http://blogs.oracle.com/alison/)  
<http://blogs.oracle.com/alison/>
- \* [AskTom](http://asktom.oracle.com) - General Oracle language and application design help  
<http://asktom.oracle.com>
- \* [Oracle Metalink](http://metalink.oracle.com) - Oracle Support website  
<http://metalink.oracle.com>
- \* [Oracle Database Editions](http://www.oracle.com/database/product_editions.html)  
[http://www.oracle.com/database/product\\_editions.html](http://www.oracle.com/database/product_editions.html)

## Documentation

- \* [OCI8 documentation](http://www.php.net/oci8)  
<http://www.php.net/oci8>
- \* [Oracle Database Express Edition 2 Day Plus PHP Developer Guide](http://download.oracle.com/docs/cd/B25329_01/doc/appdev.102/b25317/toc.htm)  
[http://download.oracle.com/docs/cd/B25329\\_01/doc/appdev.102/b25317/toc.htm](http://download.oracle.com/docs/cd/B25329_01/doc/appdev.102/b25317/toc.htm)
- \* [Oracle Database Express Edition 10g Release 2 \(10.2\) Documentation](http://www.oracle.com/pls/xe102/homepage) (Oracle Database XE)  
<http://www.oracle.com/pls/xe102/homepage>
- \* [Oracle 10.2 Database Documentation](http://www.oracle.com/pls/db102/homepage) - mostly a superset of XE documentation  
<http://www.oracle.com/pls/db102/homepage>
- \* [Connection Pooling in Oracle Net](http://st-doc.us.oracle.com/10/102/network.102/b14212/intro.htm#i454806)  
<http://st-doc.us.oracle.com/10/102/network.102/b14212/intro.htm#i454806>
- \* [Session Multiplexing](http://st-doc.us.oracle.com/10/102/network.102/b14212/intro.htm#i442084)  
<http://st-doc.us.oracle.com/10/102/network.102/b14212/intro.htm#i442084>

## Resources

- \* [Oracle Net Services Administrator's Guide](http://www.oracle.com/pls/db102/to_toc?pathname=network.102%2Fb14212%2Ftoc.htm&remark=portal+%28Administration%29)  
[http://www.oracle.com/pls/db102/to\\_toc?pathname=network.102%2Fb14212%2Ftoc.htm&remark=portal+%28Administration%29](http://www.oracle.com/pls/db102/to_toc?pathname=network.102%2Fb14212%2Ftoc.htm&remark=portal+%28Administration%29)
- \* [Oracle Net Services Reference](http://www.oracle.com/pls/db102/to_toc?pathname=network.102%2Fb14213%2Ftoc.htm&remark=portal+%28Administration%29)  
[http://www.oracle.com/pls/db102/to\\_toc?pathname=network.102%2Fb14213%2Ftoc.htm&remark=portal+%28Administration%29](http://www.oracle.com/pls/db102/to_toc?pathname=network.102%2Fb14213%2Ftoc.htm&remark=portal+%28Administration%29)
- \* [Oracle Database Net Services Administrator's Guide 10g Release 2 \(10.2\)](http://download-west.oracle.com/docs/cd/B19306_01/network.102/b14212/naming.htm#ABC524382SRI12)  
[http://download-west.oracle.com/docs/cd/B19306\\_01/network.102/b14212/naming.htm#ABC524382SRI12](http://download-west.oracle.com/docs/cd/B19306_01/network.102/b14212/naming.htm#ABC524382SRI12)
- \* [Using PHP5 with Oracle XML DB](http://www.oracle.com/technology/oramag/oracle/05-jul/o45php.html)  
<http://www.oracle.com/technology/oramag/oracle/05-jul/o45php.html>
- \* [Oracle Database SQL Reference](http://download-west.oracle.com/docs/cd/B19306_01/server.102/b14200/toc.htm)  
[http://download-west.oracle.com/docs/cd/B19306\\_01/server.102/b14200/toc.htm](http://download-west.oracle.com/docs/cd/B19306_01/server.102/b14200/toc.htm)

## Articles and Other References

- \* [Oracle PHP Troubleshooting FAQ](http://www.oracle.com/technology/tech/php/htdocs/php_troubleshooting_faq.html)  
[http://www.oracle.com/technology/tech/php/htdocs/php\\_troubleshooting\\_faq.html](http://www.oracle.com/technology/tech/php/htdocs/php_troubleshooting_faq.html)
- \* [Oracle PL/SQL Programming](http://www.amazon.com/gp/product/0596009771/104-2601327-4619903) by Steven Feuerstein, O'Reilly Press. Currently in its 4th edition.  
<http://www.amazon.com/gp/product/0596009771/104-2601327-4619903>
- \* [Improving Performance Through Persistent Connections](http://www.oracle.com/technology/pub/articles/oracle_php_cookbook/coggeshall_persist.html) by John Coggeshall  
[http://www.oracle.com/technology/pub/articles/oracle\\_php\\_cookbook/coggeshall\\_persist.html](http://www.oracle.com/technology/pub/articles/oracle_php_cookbook/coggeshall_persist.html)
- \* [Autonomous transactions in Oracle](http://asktom.oracle.com/%7Etkyte/autonomous/index.html)  
<http://asktom.oracle.com/%7Etkyte/autonomous/index.html>
- \* [Using PHP 5 with Oracle XML DB](http://www.oracle.com/technology/oramag/oracle/05-jul/o45php.html) by Yuli Vasiliev  
<http://www.oracle.com/technology/oramag/oracle/05-jul/o45php.html>
- \* [An Overview on Globalizing Oracle PHP Applications](http://www.oracle.com/technology/tech/php/pdf/globalizing_oracle_php_applications.pdf)  
[http://www.oracle.com/technology/tech/php/pdf/globalizing\\_oracle\\_php\\_applications.pdf](http://www.oracle.com/technology/tech/php/pdf/globalizing_oracle_php_applications.pdf)

## Source and Binaries

- \* [Zend Core for Oracle](http://www.oracle.com/technology/tech/php/zendcore/)  
<http://www.oracle.com/technology/tech/php/zendcore/>
- \* [PHP versioned releases \(source and Windows binaries\)](http://www.php.net/downloads.php)  
<http://www.php.net/downloads.php>
- \* [OC18 source code in CVS](http://cvs.php.net/viewcvs.cgi/php-src/ext/oci8/)  
<http://cvs.php.net/viewcvs.cgi/php-src/ext/oci8/>

- \* [Snapshots of PHP's source code \(and Windows binaries\)](http://snaps.php.net/)  
<http://snaps.php.net/>
- \* [PECL OCI8 source snapshot](http://pecl.php.net/package/oci8)  
<http://pecl.php.net/package/oci8>
- \* [Windows OCI8 DLLs matching PECL oci8 snapshots](http://pecl4win.php.net/ext.php/php_oci8.dll)  
[http://pecl4win.php.net/ext.php/php\\_oci8.dll](http://pecl4win.php.net/ext.php/php_oci8.dll)
- \* [Oracle Instant Client](http://www.oracle.com/technology/tech/oci/instantclient/)  
<http://www.oracle.com/technology/tech/oci/instantclient/>

## PHP Bugs

- \* [PHP bug system](http://bugs.php.net/search.php)  
<http://bugs.php.net/search.php>

## Utilities

- \* [PHP Extension for JDeveloper](http://www.oracle.com/technology/products/jdev/htdocs/partners/addins/exchange/php/index.html)  
<http://www.oracle.com/technology/products/jdev/htdocs/partners/addins/exchange/php/index.html>





# Glossary

## **Anonymous Block**

A PL/SQL block that appears in your application and is not named or stored in the database. In many applications, PL/SQL blocks can appear wherever SQL statements can appear. A PL/SQL block groups related declarations and statements. Because these blocks are not stored in the database, they are generally for one-time use.

## **Binding**

A method of including data in SQL statements that allows SQL statements to be efficiently reused with different data.

## **BFILE**

The BFILE datatype stores unstructured binary data in operating-system files outside the database. A BFILE column or attribute stores a file locator that points to an external file containing the data.

## **BLOB**

The BLOB datatype stores unstructured binary data in the database.

## **CHAR**

The CHAR datatype stores fixed-length character strings.

## **CLOB and NCLOB**

The CLOB and NCLOB datatypes store up to 8 terabytes of character data in the database. CLOBs store database character set data, and NCLOBs store Unicode national character set data.

## **Collection Type**

A collection is an ordered group of elements, all of the same type. Each element has a unique subscript that determines its position in the collection. PL/SQL datatypes TABLE and VARRAY enable collection types such as arrays, bags, lists, nested tables, sets and trees.

## **Connection String**

The string used to identify which database to connect to.

## **Data Dictionary**

A set of tables and views that are used as a read-only reference about the database.

## **Database**

A database stores and retrieves data. Each database consists of one or more data files. Although you may have more than one database per machine, typically a single Oracle database contains multiple schemas. A schema is often equated with a user. Multiple applications can use the same database without any conflict by using different schemas.

### **Database Link**

A pointer that defines a one-way communication path from an Oracle Database server to another database server. A database link connection allows local users to access data on a remote database.

### **Database Name**

The name of the database. In the PHP world, this is the text used in `oci_connect()` calls. Also see Easy Connect.

### **Datatype**

Each column value and constant in a SQL statement has a datatype, which is associated with a specific storage format, constraints, and a valid range of values. When you create a table, you must specify a datatype for each of its columns. For example, NUMBER, or DATE

### **DATE**

The DATE datatype stores point-in-time values (dates and times) in a table.

### **DDL**

SQL's Data Definition Language. SQL statements that define the database structure or schema, like CREATE, ALTER, and DROP.

### **DML**

SQL's Data Manipulation Language. SQL statements that define or manage the data in the database, like SELECT, INSERT, UPDATE and DELETE.

### **Easy Connect**

A simple hostname/database name string that is used to identify which database to connect to.

### **Index**

Indexes are optional structures associated with tables. Indexes can be created to increase the performance of data retrieval.

### **Instance**

The Oracle Instance is the running component of an Oracle database server. When an Oracle database is started, a system global area (SGA) is allocated and Oracle background processes are started. The combination of the background processes and memory buffers is called an Oracle instance.

### **Instant Client**

The Oracle Instant Client is a small set of libraries, which allow you to connect to an Oracle Database. A subset of the full Oracle Client, it requires minimal installation but has full functionality. Instant Client is downloadable from OTN and is usable and distributable for free. An Oracle Instant Client install does not require to be in an Oracle Home.

### **LOB**

A large object. LOBS may be persistent (stored in the database) or temporary. See CLOB, BLOB and BFILE.

**LOB Locator**

A “pointer” to LOB data.

**Materialized View**

A materialized view provides access to table data by storing the results of a query in a separate schema object. Unlike an ordinary view, which does not take up any storage space or contain any data, a materialized view contains the rows resulting from a query against one or more base tables or views.

**NCHAR and NVARCHAR2**

NCHAR and NVARCHAR2 are Unicode datatypes that store Unicode character data.

**NUMBER**

The NUMBER datatype stores fixed and floating-point numbers.

**Packages**

A package is a group of related procedures and functions, along with the cursors and variables they use, stored together in the database for continued use as a unit.

**Procedures and Functions**

A PL/SQL procedure or function is a schema object that consists of a set of SQL statements and other PL/SQL constructs, grouped together, stored in the database, and run as a unit to solve a specific problem or perform a set of related tasks.

**Object Privilege**

A right to perform a particular action on a specific schema object. Different object privileges are available for different types of schema objects. The privilege to delete rows from the DEPARTMENTS table is an example of an object privilege.

**ORACLE\_HOME install**

An Oracle Client or Oracle Database install. These installs contain all required software in a directory hierarchy. This set of directories includes binaries, utilities, configuration scripts, demonstration scripts and error message files for each component of Oracle. Any program using Oracle typically requires the ORACLE\_HOME environment variable to be set to the top level installation directory.

**Oracle Net**

The networking component of Oracle that connects client tools such as PHP to local or remote databases. The Oracle Net listener is a process that handles connection requests from clients and passes them to the target database.

**OTN**

The Oracle Technology Network is Oracle’s free repository of articles on Oracle technologies. It also hosts many discussion forums, including one on PHP.

**Package**

A group of PL/SQL procedures, functions, and variable definitions stored in the Oracle database. Procedures, functions, and variables in packages can be called from other packages, procedures, or functions.

## **PEAR**

The PHP Extension and Application Repository (PEAR) is a repository for reusable packages written in PHP.

## **PECL**

The PHP Extension Community Library (PECL) is a repository of PHP extensions that can be linked into the PHP binary.

## **PHP**

A popular, interpreted scripting language commonly used for web applications. PHP is a recursive acronym for “PHP: Hypertext Preprocessor”

## **Php.ini**

The configuration file used by PHP. Many (but not all) options that are set in php.ini can also be set at runtime using `ini_set()`.

## **PL/SQL**

Oracle's procedural language extension to SQL. It is a server-side, stored procedural language that enables you to mix SQL statements with procedural constructs. With PL/SQL, you can create and run PL/SQL program units such as procedures, functions, and packages. PL/SQL program units generally are categorized as anonymous blocks, stored functions, stored procedures, and packages.

## **Prepared Statement**

A pre-parsing step for SQL statements. This term is often used with non-Oracle databases. In Oracle it is called parsing.

## **Regular Expression**

A pattern used to match data. Oracle has several functions that accept regular expressions.

## **Schema**

A schema is a collection of database objects. A schema is owned by a database user and has the same name as that user. Schema objects are the logical structures that directly refer to the database's data. Schema objects include structures like tables, views, and indexes.

## **Sequence**

A sequence (a sequential series of numbers) of Oracle integers of up to 38 digits defined in the database.

## **SID**

The system identifier is commonly used to mean the database name alias in the connection string. SID also stands for session identifier, the unique number assigned to each database user session when they connect to the database.

## **SQL\*Plus**

The traditional command line tool for executing SQL statements available with all Oracle databases. Although recently superceded by GUI tools like Oracle's free SQL Developer, SQL\*Plus remains hugely popular. It is also convenient to show examples using SQL\*Plus.

**Stored Procedures and Functions**

A PL/SQL block that Oracle stores in the database and can be called by name from an application. Functions are different than procedures in that functions return a value when executed. When you create a stored procedure or function, Oracle parses the procedure or function, and stores its parsed representation in the database.

**Synonym**

A synonym is an alias for any table, view, materialized view, sequence, procedure, function, package, type, Java class schema object, user-defined object type, or another synonym.

**SYS**

An administrative user account name, which has access to all base tables and views for the database data dictionary.

**SYSDBA**

A system privilege that, by default, is assigned only to the SYS user. It enables SYS to perform high-level administrative tasks such as starting up and shutting down the database.

**SYSOPER**

Similar to SYSDBA, but with a limited set of privileges that allows basic administrative tasks without having access to user data.

**SYSTEM**

An administrative user account name that is used to perform all administrative functions other than starting up and shutting down the database.

**System privilege**

The right to perform a particular action, or to perform an action on any schema objects of a particular type. For example, the privileges to create tables and to delete the rows of any table in a database.

**Table**

Tables are the basic unit of data storage. Database tables hold all user-accessible data. Each table has columns and rows.

**Tablespace**

Tablespaces are the logical units of data storage made up of one or more datafiles. Tablespaces are often created for individual applications because tablespaces can be conveniently managed. Users are assigned a default tablespace that holds all the data the users creates. A database is made up of default and DBA-created tablespaces.

**Temporary LOB**

See LOB.

**Tnsnames.ora**

The Oracle Net configuration file used for connecting to a database. The file maps an alias to a local or remote database and allows various configuration options for connections. The alias is used in the PHP connection string. TNS stands for Transparent Network Substrate.

### **Transaction**

A sequence of SQL statements whose changes are either all committed, or all rolled back.

### **Trigger**

A stored procedure associated with a database table, view, or event. The trigger can be called after the event, to record it, or take some follow-up action. The trigger can be called before the event, to prevent erroneous operations or fix new data so that it conforms to business rules.

### **User**

A database user is often equated to a schema. Each user connects to the database with a username and secret password, and has access to tables, and so on, in the database.

### **VARCHAR and VARCHAR2**

These datatypes store variable-length character strings. The names are currently synonyms but VARCHAR2 is recommended to ensure maximum compatibility of applications in future.

### **View**

Views are customized presentations of data in one or more tables or other views. A view can also be considered a stored query. Views do not actually contain data. Rather, they derive their data from the tables on which they are based, referred to as the base tables of the views.

### **XMLType**

XMLType is a datatype that can be used to store XML data in table columns.