Oracle® Application Server

mod_plsql User's Guide

10*g* Release 3 (10.1.3)

B25599-01

January 2006

ORACLE®

Oracle Application Server mod_plsql User's Guide, 10*g* Release 3 (10.1.3)

B25599-01

Primary Author: Pravin Prabhakar

Contributing Author: Peter Lubbers

Contributors: Pushkar Kapasi, Eric Lee, Thomas Van Raalte, and Nick Greenhalgh

Contents

4 Optimizing PL/SQL Performance

List of Examples

List of Figures

List of Tables

Preface

This manual describes how to install, configure, and maintain mod_plsql.

New Features

New features in this release of mod_plsql include:

- Dead database connection detection. Refer to Section 3.11.3, "Detecting Dead Database Connections in a Connection Pool" for more information.
- Definition of a tighter security model by allowing PL/SQL applications to validate requests before processing them.

New features in the previous release of mod_plsql include:

- Enhanced cache clean up algorithm allowing users to configure the time to clean up the cache.
- Enhanced OWA packages to allow better byte-packing of response data from multibyte databases resulting in fewer round-trips to the database.
- Reduced PL/SQL function call overheads in heavily called OWA package APIs like `htp.p`.
- Enhanced performance logging in mod_plsql. See "What kinds of logging facilities are available in mod_plsql?" in Appendix A, "Frequently Asked Questions" for more information.
- Enhanced security by obscuring schema passwords in the DAD configuration.
- Removed the requirement to have `/pls` in the virtual path for accessing mod_plsql. It can still be used, but is no longer required. mod_plsql has been integrated in the Oracle HTTP Server configuration and so a mod_plsql URL no longer needs to start with a "`/pls`" prefix.
- Note that OracleAS Portal DADs still need to be of the format `/pls/dad`.

Audience

This manual is intended for Oracle Application Server administrators, for using mod_plsql to deploy PL/SQL-based database applications on the World Wide Web.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive

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

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more information, you may also refer to the following manuals:

- *Oracle Database Application Developer's Guide - Fundamentals*, in the Oracle Database documentation library
- *Oracle HTTP Server Administrator's Guide*
- *Oracle Application Server Administrator's Guide*

Conventions

The following text conventions are used in this document:

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

1

Configuring mod_plsql

This chapter describes how you can set up and use mod_plsql. It contains the following sections:

1.1 Verifying Requirements

Before you run mod_plsql, you must satisfy the following requirements:

- You must have a SYS user password on the database where you plan to load PL/SQL Web Toolkit packages required by mod_plsql.
- The database to which you plan to connect mod_plsql must be up and running.
- Oracle HTTP Server mod_plsql ships with OWA package version 10.1.2.0.4. It is recommended that the OWA packages installed in the database are at least version 10.1.2.0.4.

1.2 Installing Required Packages

After installation, if you need to use Oracle HTTP Server mod_plsql with a database that is not shipped with the product, you must manually install additional required packages using the `owaload.sql` script.

> **Note:** Even if a full database export is made with the Export utility you still must reinstall mod_plsql in the new target instance by running the `OWALOAD.SQL` script as SYS. Objects in SYS are not imported with the Import/Export mechanism, and the PL/SQL toolkit has to be installed in SYS.

1. Navigate to the directory where the `owaload.sql` file is located. This directory is *`ORACLE_HOME`*`/Apache/modpsql/owa`.
2. Using SQL*Plus, login to the Oracle Database as the SYS user.
3. You can check the version of the OWA packages currently installed by running the following query:

```
select owa_util.get_version from dual;
```

- If the query succeeds, but shows a version less than 10.1.2.0.4, it is recommended that you install the newer OWA packages.
- If the query fails, you either do not have the OWA packages installed, or are running a very old version of OWA packages, and it is recommended that you install, or upgrade to the new OWA packages.

Note: To detect older OWA packages, see "How do I detect and clean up duplicate OWA packages installed in the database?" in Appendix A, "Frequently Asked Questions".

---

4. At the SQL prompt, run the following command:

```
@owaload.sql log_file
```

Table 1–1 Installing Required Packages Parameters

Elements	Description
owaload.sql	Installs the PL/SQL Web Toolkit packages into the SYS schema. It also creates public synonyms and makes the packages public so that all users in the database have access to them. Therefore, only one installation for each database is needed.
log_file	The installation log file. Make sure that you have write permissions to create the log file

5. Scan the log file for any errors.

Note: The `owaload` script checks the existing version of the OWA packages in the database and installs a new version only if:

- No OWA package exists or,
- Older OWA packages were detected. If your database already has the latest OWA packages or has a newer version installed, the `owaload` script does nothing and reports this in the log file.

---

6. Perform a manual recompile.

Note: Installing the OWA packages might invalidate all dependent objects. These packages automatically recompile on first access, but a manual recompile is recommended after the reinstallation.

After the install, check the version of the OWA packages by running "`Select owa_util.get_version from dual;`". Confirm that the version shown is 10.1.2.0.4 or later.

7. Note that public access is now granted to:
   - OWA_CUSTOM
   - OWA

- HTF
- HTP
- OWA_COOKIE
- OWA_IMAGE
- OWA_OPT_LOCK
- OWA_PATTERN
- OWA_SEC
- OWA_TEXT
- OWA_UTIL
- OWA_CACHE
- WPG_DOCLOAD
- OWA_MATCH

8. Note also that the following public synonyms are created:
   - OWA_CUSTOM for OWA_CUSTOM
   - OWA_GLOBAL for OWA_CUSTOM
   - OWA for OWA
   - HTF for HTF
   - HTP for HTP
   - OWA_COOKIE for OWA_COOKIE
   - OWA_IMAGE for OWA_IMAGE
   - OWA_OPT_LOCK for OWA_OPT_LOCK
   - OWA_PATTERN for OWA_PATTERN
   - OWA_SEC for OWA_SEC
   - OWA_TEXT for OWA_TEXT
   - OWA_UTIL for OWA_UTIL
   - OWA_INIT for OWA_CUSTOM
   - OWA_CACHE for OWA_CACHE
   - WPG_DOCLOAD for WPG_DOCLOAD
   - OWA_MATCH for OWA_MATCH

1.3 Configuring a DAD for Your PL/SQL Application

To access a Web-enabled PL/SQL application, you must first configure a PL/SQL Database Access Descriptor (DAD) for mod_plsql. A DAD is a set of values that specifies how mod_plsql connects to a database server to fulfill an HTTP request. Besides the connection details, a DAD contains important configuration parameters for various operations in the database, and for mod_plsql in general. For detailed information on mod_plsql configuration parameters, refer to the mod_plsql section in the *Oracle HTTP Server Administrator's Guide*.

1.4 Accessing mod_plsql Configuration

To configure mod_plsql, refer to the mod_plsql section in *Oracle HTTP Server Administrator's Guide*.

1.5 Diagnostic Output of mod_plsql

mod_plsql is an Oracle HTTP Server module that enables you to invoke PL/SQL applications over HTTP. As mod_plsql is an Oracle HTTP Server module, its logging is performed through Oracle HTTP Server.

Logging is controlled by the `LogLevel` parameter found in the configuration file `httpd.conf`, which is usually located at:

ORACLE_HOME`/Apache/Apache/conf`

Valid values for `LogLevel` are the following:

- `emerg`
- `alert`
- `crit`
- `error`
- `warn`
- `notice`
- `info`
- `debug`

The values build incrementally. For example, if `LogLevel` is set to `notice`, then `notice`, `warn`, `error`, `crit`, `alert`, and `emerg` messages are recorded. If the value of `LogLevel` is altered, then Oracle HTTP Server must be restarted for this change to take effect.

mod_plsql Log File Contents

The location of the mod_plsql diagnostic information is dictated by the Oracle HTTP Server parameter called `ErrorLog` found in the `httpd.conf` file. Although this parameter is called `ErrorLog`, the file it describes can contain more than just error messages. A typical value for the `ErrorLog` parameter is the following:

ORACLE_HOME`/Apache/Apache/logs/error_log`

Two types of mod_plsql messages appear in the Oracle HTTP Server error log:

- Standard mod_plsql Messages
- Performance mod_plsql Messages

Standard mod_plsql Messages

The following is an example of a standard mod_plsql message found in the Oracle HTTP Server error log:

```
[Thu Jun 30 08:34:20 2005] [warn] mod_plsql: 'PlsqlCacheCleanupSize' is
deprecated.
```

The content of the standard mod_plsql message is as follows:

- Date and time: Thu June 30 08:34:20 2005
- Message level: warn
- Indicates this message comes from mod_plsql: mod_plsql
- Message text: 'PlsqlCacheCleanupSize' is deprecated.

2

Securing Application Database Access Through mod_plsql

This chapter describes how to set up the database and PL/SQL, for best security. It covers the following topics:

- Security Considerations
- Authenticating Users Through mod_plsql
- Deauthenticating Users

> **See Also:** For more information about mod_plsql configuration parameters, refer to the *Oracle HTTP Server Administrator's Guide*.

2.1 Security Considerations

mod_plsql is a PL/SQL gateway that can be used to access any procedure to which your database account has rights. Since this also includes access to all packages granted to PUBLIC, it is the responsibility of the PL/SQL application to ensure that mod_plsql is not used to access unauthorized packages or procedures. By default, mod_plsql does not allow access to:

- Sensitive schemas. For example, `SYS.*`.
- Packages known to be sensitive, but to which PUBLIC has access. For example, `OWA_*`, `DBMS_*`, and `UTL_*` packages.
- Procedure names that contain special characters in the URL.

This default model of explicitly denying access to specific packages from being executed directly using mod_plsql is sufficient for most correctly designed PL/SQL applications. If this level of security is not adequate for your PL/SQL application, it is highly recommended that you follow one or more of the following techniques to ensure that your PL/SQL application is secure:

- Defining the PlsqlRequestValidationFunction Directive in mod_plsql
- Adding More Rules to the PlsqlExclusionList Directive in mod_plsql
- Using the AUTHORIZE Procedure in Custom Authentication
- Protecting Database Tables
- Protecting the Database from SQL Injection

2.1.1 Defining the PlsqlRequestValidationFunction Directive in mod_plsql

mod_plsql provides a DAD parameter directive called `PlsqlRequestValidationFunction` which enables you to allow or disallow further processing of a requested procedure. This is useful for implementing tight security for your PL/SQL application by blocking package or procedure calls that should not be allowed to run from the DAD.

The function defined by this parameter must have the following prototype:

```
boolean function_name (procedure_name IN varchar2)
```

When invoked, the argument '`procedure_name`' will contain the name of the procedure that the request is trying to run.

For example, if all the PL/SQL application procedures that can be called from a browser are inside the package "**mypkg**", then a simple implementation of this function can be as shown in Example 2–1:

Example 2–1 Implementation of Request Validation Function to Block Procedures

```
boolean my_validation_check (procedure_name varchar2)
is
begin
    if (upper(procedure_name) like upper('myschema.mypkg%')) then
        return TRUE;
    else
        return FALSE;
    end if;
end;
```

Tips:

- It is highly recommended that you implement this function such that it only allows requests that belong to your application, and can be called from a browser.
- As this function will be called for every request, ensure that this function is made performant. Some recommendations are:
  - Name your PL/SQL packages such that all procedures accessible from the Web are in a small set of known packages, and these packages do not define procedures that should not be accessible directly from the Web browser. With a good naming convention, the implementation of the validation function can be similar to Example 2–1.
  - If your implementation performs a table lookup to determine the packages/procedures that should be allowed to run, the performance can be further improved if you pin the cursor in the shared pool.

2.1.2 Adding More Rules to the PlsqlExclusionList Directive in mod_plsql

mod_plsql provides a DAD configuration parameter called `PlsqlExclusionList`, which can be used to disallow execution of procedures with specific patterns, directly from a browser URL. The specified pattern is case insensitive and allows a wildcard pattern of "*", which means "zero or more occurrences of any set of characters". The default patterns that are not accessible from a direct URL are `sys.*`, `dbms_*`, `utl_*`, `owa_util.*`, `owa_*`, `htp.*`, and `htf.*`. Starting from Oracle Application Server 10*g* Release 2 (10.1.2), the default built-in exclusion list remains in effect even if the user has configured additional rules using `PlsqlExclusionList`. In previous versions, if the `PlsqlExclusionList` directive was overridden in the DAD configuration, the default settings no longer applied.

In addition to the patterns that are specified with this directive, mod_plsql also disallows any URLs that contain special characters such as tabs, new lines, carriage returns, single quotation marks (`'`), reverse slash (`\`), and so on.

For details on the `PlsqlExclusionList` parameter, refer to the "mod_plsql" section in the *Oracle HTTP Server Administrator's Guide*.

Caution: Setting the `PlsqlExclusionList` directive to **#NONE#** disables all protection, and should not be used for an active Web site.

You can set the `PlsqlExclusionList` directive in the mod_plsql configuration file called `dads.conf`. This configuration file is located in the following directories:

- (UNIX) *ORACLE_HOME*`/Apache/modplsql/conf/`
- (Windows) *ORACLE_HOME*`\Apache\modplsql\conf`

Where *ORACLE_HOME* is the location of your Oracle HTTP Server installation.

To ensure the best security for user-defined PL/SQL procedures that are granted to PUBLIC, specify the user settings with the `PlsqlExclusionList` directive in the `dads.conf` file as shown in Example 2–2.

Example 2–2 Specifying the PlsqlExclusionList Directive

```
PlsqlExclusionList myschema.mypackage*
```

2.1.3 Using the AUTHORIZE Procedure in Custom Authentication

Applications using Custom Authentication can choose to deny access to restricted packages or procedures in their implementation of the **AUTHORIZE** procedure.

2.1.4 Protecting Database Tables

It can be useful to separate the schema containing your application objects (tables and procedures) from the schema against which you will run mod_plsql. This way, the user that mod_plsql connects as can be prevented from any direct access to your tables and other database objects, except through APIs that are granted to that user. Synonyms can be used to make the procedures executable without schema prefixes.

2.1.5 Protecting the Database from SQL Injection

A SQL Injection attack occurs when a malicious user is able to modify a SQL command being run against a back-end database. The modification may be to read additional data, cause a validation check to succeed when it should fail, write records, or otherwise affect the smooth running of an application.

Many applications are built on databases and often user inputs are used as part of database queries. If the application has not been coded carefully, it is sometimes possible to change the query submitted to the database by careful construction of input.

To prevent such SQL Injection attacks, wherever possible, *do not* use dynamic SQL created from any user supplied input.

If user supplied input has to be used for dynamic SQL, then the input *must* be adequately filtered by the application before using the input in any dynamic SQL. For example, if the user supplies a table name as an input parameter, then the underlying application code should ensure that the only acceptable value for this parameter is a valid table name.

2.2 Authenticating Users Through mod_plsql

mod_plsql provides different levels of authentication in addition to those provided by the Oracle HTTP Server. The Oracle HTTP Server protects documents, virtual paths and so forth, while mod_plsql protects users logging into the database or running a PL/SQL Web application.

You can enable different authentication modes, as described in Table 2–1.

Table 2–1 Authentication Modes Used with mod_plsql

Authentication Mode	Approach
Basic	Authentication is performed using basic HTTP authentication. Most applications use basic authentication.
Global OWA	Authentication is performed using the owa_custom.authorize procedure in the schema containing the PL/SQL Web Toolkit packages.
Custom OWA	Authentication is performed using packages and procedures in the user's schema (owa_customize.authorize), or if not found, in the schema containing the PL/SQL Web Toolkit packages.
PerPackage	Authentication is performed using packages and procedures in the user's schema (packageName.authorize).
Single Sign-on	Authentication is performed using Oracle Application Server Single Sign-On. Use this mode only if your application works with OracleAS Single Sign-On.

2.2.1 Basic (Database Controlled Authentication)

The module, mod_plsql, supports authentication at the database level. It uses HTTP basic authentication but authenticates credentials by using them to attempt to log on to the database. Authentication is verified against a user database account, using user names and passwords that are either:

- stored in the DAD. The end user is not required to log in. This method is useful for Web pages that provide public information.

- provided by the users by means of the browser's Basic HTTP Authentication dialog box. The user must provide a user name and password in the dialog box.

2.2.2 Oracle HTTP Server mod_plsql Basic Authentication Mode

Oracle HTTP Server has a different mechanism for the basic authentication mode. The user name and password must be stored in the DAD. Oracle HTTP Server uses HTTP basic authentication where the credentials are stored in a password file on the file system. Authentication is verified against the users listed in that file.

Basic Authentication Mode

mod_plsql supports basic authentication. Oracle HTTP Server authenticates users' credentials against a password file on the file system. This functionality is provided by a module called `mod_auth`.

2.2.3 Global OWA, Custom OWA, and Per Package (Custom Authentication)

Custom authentication enables applications to authenticate users within the application itself, not at the database level. Authorization is performed by invoking a user-written authorization function.

Custom authentication is done using OWA_CUSTOM and cannot be combined with dynamic username/password authentication. Custom authentication needs to have a static username/password stored in the DAD configuration file. mod_plsql uses this DAD username/password to login to the database. Once mod_plsql is logged in, authentication control is passed back to the application, by calling an application-level PL/SQL hook. This callback function is implemented by the application developers. The value returned by the callback function determines the success or failure of authentication. The value `TRUE` means success and `FALSE` means failure.

Depending on the kind of custom authentication needed, you can place the authentication function in different locations:

- Global OWA enables you to invoke the same authentication function for all users and procedures.
- Custom OWA enables you to invoke a different authentication function for each user and for all procedures.
- Per Package authentication enables you to invoke the authentication function for all users, but only for procedures in a specific package or for anonymous procedures.

For example, when using Custom OWA, an authorization function might verify that a user has logged in as user **guest** with password **welcome**, or it might check the user's IP address to determine access.

Table 2–2 summarizes the parameter values.

Table 2–2 Custom Authentication Modes and Callback Functions

Mode	Access Control Scope	Callback Function
Global OWA	All packages	`owa_custom.authorize` in the OWA package schema.
Custom OWA	All packages	`owa_custom.authorize` in the user's schema, or, if not found, in the OWA package schema.

Table 2–2 (Cont.) Custom Authentication Modes and Callback Functions

Mode	Access Control Scope	Callback Function
Per package	Specified package	`packageName.authorize` in the user's schema, or `anonymous.authorize` is called.

2.3 Deauthenticating Users

For DADs using dynamic authentication (no username/password in the DAD), mod_plsql allows users to log off (clear HTTP authentication information) programmatically through a PL/SQL procedure without having to exit all browser instances. This feature is supported on Netscape 3.0 or higher and on Microsoft Internet Explorer. On other browsers, the user may have to exit the browser to deauthenticate.

Deauthentication can be done programatically by creating your own logout procedure, which simulates a logout and redirects the user to a sign-off page.

Create or replace procedure `MyLogOffProc` as follows:

```
BEGIN
   -- Open the HTTP header
   owa_util.mime_header('text/html', FALSE, NULL);

   -- Send a cookie to logout
   owa_cookie.send('WDB_GATEWAY_LOGOUT', 'YES', path=>'/');

   -- Close the HTTP header
   owa_util.http_header_close;

   -- Generate the page
   htp.p('You have been logged off from the WEBSITE');
   htp.anchor( 'http://www.abc.com', 'click here');
   htp.p('<BR>bye');
END;
```

Another method of deauthentication is to add `/logmeoff` after the DAD in the URL. For example:

```
http://www.abc.com:2000/pls/myDAD/logmeoff
```

3

Understanding mod_plsql

mod_plsql provides support for deploying PL/SQL-based database applications on the World Wide Web. It is part of Oracle HTTP Server, which ships with Oracle Application Server and Oracle Database.

As part of the Oracle HTTP Server, it is the job of mod_plsql to interpret a URL sent by a Web browser to a Web server, call the appropriate PL/SQL subprograms to treat the browser request, then return the generated response to the browser. Typically, mod_plsql responds to a Web browser HTTP request by constructing an HTML page to display. There are additional uses for mod_plsql, of which two are listed subsequently:

- Transfer files from a client machine to or from Oracle Database. You can upload and download text files or binary files.
- Perform custom user authentication in Web applications.

As a plug-in to Oracle HTTP Server, mod_plsql causes stored procedures to be executed in response to HTTP requests. For each URL that is processed, mod_plsql either uses a database session from its connection pool, or creates a new session on the fly and pools it. For mod_plsql to invoke the appropriate database PL/SQL procedure in a URL-processing session, you must first configure a virtual path and associate that path with a Database Access Descriptor (DAD).

A DAD is a named set of configuration values that specify the information necessary to create a session for a specific database and a specific database user/password. This includes the database service name and the Globalization Support setting (for example, language) for the session. Refer to Section 3.2, "Database Access Descriptors (DADs)" for more information.

To develop the stored procedures that are executed by mod_plsql at runtime, you use the PL/SQL Web Toolkit: a set of PL/SQL packages that can be used to obtain information about an HTTP request; specify HTTP response headers, such as cookies, content-type, and mime-type, for HTTP headers; set cookies; and generate standard HTML tags for creating HTML pages. Refer to *Oracle Application Server PL/SQL Web Toolkit Reference* for more information.

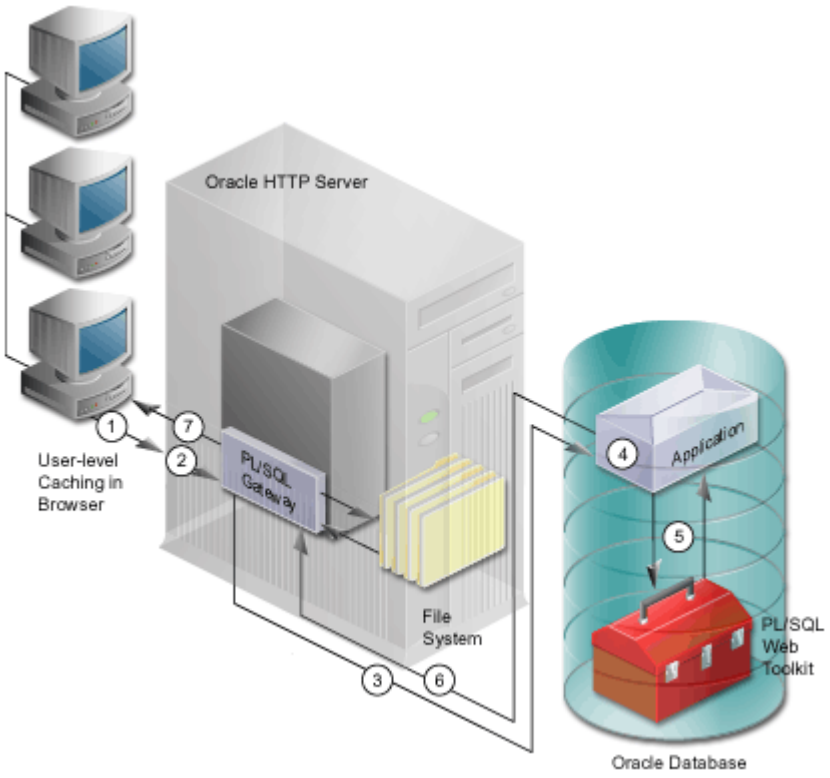
This chapter discusses the following topics:

3.1 Processing Client Requests

mod_plsql is an Oracle HTTP Server plug-in that communicates with the database. It maps browser requests into database stored procedure calls over a SQL*Net connection. It is often indicated by a `/pls` virtual path.

The following scenario provides an overview of what steps occur when a server receives a client request:

Figure 3–1 Overview of the Process When a Server Receives a Client Request



1. The Oracle HTTP Server receives a request containing a virtual path, which is configured to be serviced by mod_plsql.
2. The Oracle HTTP Server routes the request to mod_plsql.
3. By using the configuration information stored in your DAD, mod_plsql connects to the database. The request is forwarded by mod_plsql to the Oracle Database.
4. mod_plsql prepares the call parameters, and invokes the PL/SQL procedure in the application.

5. The PL/SQL procedure generates an HTML page using data and the PL/SQL Web Toolkit accessed from the database.
6. The response is returned to mod_plsql.
7. The Oracle HTTP Server sends the response to the client browser.

The procedure that mod_plsql invokes returns the HTTP response to the client. To simplify this task, mod_plsql includes the PL/SQL Web Toolkit, which contains a set of packages called the owa packages. Use these packages in your stored procedure to get information about the request, construct HTML tags, and return header information to the client. Install the toolkit in a common schema so that all users can access it.

3.2 Database Access Descriptors (DADs)

Each mod_plsql request is associated with a Database Access Descriptor (DAD), a set of configuration values used for database access. A DAD specifies information such as:

- the database alias (Oracle Net service name).
- a connect string, if the database is remote.
- a procedure for uploading and downloading documents.

You can also specify username and password information in a DAD. If they are not specified, the user is prompted to enter a username and password when the URL is invoked.

> **See Also:** *Oracle HTTP Server Administrator's Guide* for descriptions of the DAD parameters and an overview of the mod_plsql configuration files.

3.3 Invoking mod_plsql

To invoke mod_plsql in a Web browser, input the URL in the following format:

```
protocol://hostname[:port]/DAD_location/[[!][schema.][package.]proc_name[?query_string]]
```

Table 3–1 Invoking mod_plsql Parameters

Parameter	Description
protocol	Either `http` or `https`. For SSL, use `https`.
hostname	The machine where the Web server is running.
port (optional)	The port at which the Web server is listening. If omitted, port 80 is assumed.
DAD location	A virtual path to handle PL/SQL requests that you have configured in the Web server. The DAD location can contain only ASCII characters.
! character (optional)	Indicates to use the flexible parameter passing scheme. See Section 3.6.2, "Flexible Parameter Passing" for more information.
schema (optional)	The database schema name. If omitted, name resolution for `package.proc_name` occurs based on the database user that the URL request is processed as.

Table 3–1 (Cont.) Invoking mod_plsql Parameters

Parameter	Description
package (optional)	The package that contains the PL/SQL stored procedure. If omitted, the procedure is standalone.
proc_name	The PL/SQL stored procedure to run. This must be a procedure and not a function. It can accept only IN arguments.
?query_string (optional)	The parameters for the stored procedure. The string follows the format of the GET method. For example: ■ Multiple parameters are separated with the `&` character. Space characters in the values to be passed in are replaced with the `+` character. ■ If you use HTML forms to generate the string (as opposed to generating the string yourself), the formatting is done automatically. ■ The HTTP request may also choose the HTTP POST method to post data to mod_plsql. See "POST, GET, and HEAD Methods" on page 3-4 for more information.

Example 3–1 Invoking A Procedure That Does Not Take Arguments

```
http://www.acme.com:9000/pls/mydad/mypackage.myproc
```

The Web server running on `www.acme.com` and listening at port `9000` handles the request. When the Web server receives the request, it passes the request to mod_plsql. This is because the `/pls/mydad` indicates that the Web server is configured to invoke mod_plsql. It then uses the DAD associated with `/pls/mydad` and runs the `myproc` procedure stored in `mypackage`.

Example 3–2 Invoking A Procedure That Takes Arguments

```
http://www.acme.com:9000/pls/mydad/mypackage.myproc?a=v&b=1
```

The Web server running on `www.acme.com` and listening at port `9000` handles the request. When the Web server receives the request, it uses the DAD associated with `/pls/mydad` and runs the `myproc` procedure stored in `mypackage`, and passes two arguments, `a` and `b`, with the values `v`, and `1` to the procedure.

Example 3–3 Invoking the Default Procedure Stored in the DAD Configuration

```
http://www.acme.com:9000/pls/mydad
```

The Web server running on `www.acme.com` and listening at port `9000` handles the request. When the Web server receives the request, it uses the DAD associated with `/pls/mydad` and invokes the default procedure configured in the DAD. For example, if the configuration parameter `PlsqlDefaultPage` in the DAD `/pls/mydad` is set to `myschema.mypackage.myproc`, then the procedure `myschema.mypackage.myproc` is invoked for the request.

In this example, the default home page for the `mydad` DAD (as specified in the DAD Configuration) is displayed.

POST, GET, and HEAD Methods

The POST, GET, and HEAD methods in the HTTP protocol instruct browsers on how to pass parameter data (usually in the form of name-value pairs) to applications. The parameter data is generated by HTML forms.

mod_plsql applications can use any of the methods. Each method is as secure as the underlying transport protocol (HTTP or HTTPS).

- When using the POST method, parameters are passed in the request body. Generally, if you are passing large amounts of parameter data to the server, use the POST method.
- When using the GET method, parameters are passed using a query string. The limitation of this method is that the length of the value in a name-value pair cannot exceed the maximum length for the value of an environment variable, as imposed by the underlying operating system. In addition, operating systems have a limit on how many environment variables you can define.
- When using the HEAD method, it has the same functionality as the GET method. The difference is that only the HTTP status line and the HTTP headers are passed back. No content data is streamed back to the browser. This is useful for monitoring tools in which you are only interested if the request is processed correctly.
- Mixed Mode - In mod_plsql you can pass some of the parameters in a query string and the remaining ones as POST data. For example, if you have a procedure `foo` (a varchar2, b number), and want to pass values "v" and "1" to 'a' and 'b' respectively, you could do so in three ways to create URLs:
  - All values are specified as part of the query string.

    ```
    http://host:port/pls/DAD/foo?a=v&b=1
    ```

  - All values are specified as part of the POST data.

    ```
    http://host:port/pls/DAD/foo, POST data="a=v&b=1"
    ```

  - Some of the parameters are specified in the URL and the rest in the POST data.

    ```
    http://host:port/pls/DAD/foo?a=v, POST data="b=1"
    ```

Note: POST data is generated as part of the input fields on a HTML form. You should not create the POST string manually in the PL/SQL procedure, or in the URL. The Submit operation of the HTML form will generate a *POST* request and pass the value to your procedure.

3.4 Transaction Mode

After processing a URL request for a procedure invocation, mod_plsql performs a rollback if there were any errors. Otherwise, it performs a commit. This mechanism does not allow a transaction to span across multiple HTTP requests. In this stateless model, applications typically maintain state using HTTP cookies or database tables.

3.5 Supported Data Types

Because HTTP supports character streams only, mod_plsql supports the following subset of PL/SQL data types:

- NUMBER
- VARCHAR2
- TABLE OF NUMBER
- TABLE OF VARCHAR2

Records are not supported.

3.6 Parameter Passing

mod_plsql supports:

- Parameter passing by name

  Each parameter in a URL that invokes procedure or functions identified by a unique name. Overloaded parameters are supported. See Section 3.6.1, "Parameter Passing by Name (Overloaded Parameters)" for more information.

- Flexible parameter passing

  Procedures are prefixed by a ! character. See Section 3.6.2, "Flexible Parameter Passing" for more information.

- Large (up to 32K) parameters passing

  See Section 3.6.3, "Large Parameter Passing" for more information.

Note: mod_plsql handles multi-value variables by storing the values in a PL/SQL table. This enables you to be flexible about how many values the user can pick, and it makes it easy for you to process the user's selections as a unit. Each value is stored in a row in the PL/SQL table, starting at index 1. The first value (in the order that it appears in the query string) of a variable that has multiple values is placed at index 1, the second value of the same variable is placed at index 2, and so on. The PL/SQL application should not rely on the ordering of the arguments passed by mod_plsql, as it can change without notice. If the order of the values in the PL/SQL table is significant to your procedure, you need to modify your PL/SQL application to do the ordering internally.

If you do not have variables with multiple values, the order in which the variables appear does not matter, because their values are passed to the procedure's parameters by name, and not by position.

The PL/SQL tables used as parameters in the mod_plsql environment must have a base type of `VARCHAR2`. Oracle can convert `VARCHAR2` to other data types such as `NUMBER`, `DATE`, or `LONG`. The maximum length of a `VARCHAR2` variable is 32K.

If you cannot guarantee that at least one value will be submitted to the PL/SQL table (for example, the user can select no options), use a hidden form element to provide the first value. Not providing a value for the PL/SQL table produces an error, and you cannot provide a default value for a PL/SQL table.

3.6.1 Parameter Passing by Name (Overloaded Parameters)

Overloading allows multiple subprograms (procedures or functions) to have the same name, but differ in the number, order, or the datatype family of the parameters. When you call an overloaded subprogram, the PL/SQL compiler determines which subprogram to call based on the data types passed.

PL/SQL enables you to overload local or packaged subprograms. Standalone subprograms cannot be overloaded.

You must give parameters different names for overloaded subprograms that have the same number of parameters. Because HTML data is not associated with datatypes, mod_plsql does not know which version of the subprogram to call.

For example, although PL/SQL enables you to define two procedures using the same parameter names for the procedures, an error occurs if you use this with mod_plsql.

```
-- legal PL/SQL, but not for mod_plsql
CREATE PACKAGE my_pkg AS
  PROCEDURE my_proc (val IN VARCHAR2);
  PROCEDURE my_proc (val IN NUMBER);
END my_pkg;
```

To avoid the error, name the parameters differently. For example:

```
-- legal PL/SQL and also works for mod_plsql
CREATE PACKAGE my_pkg AS
  PROCEDURE my_proc (valvc2 IN VARCHAR2);
  PROCEDURE my_proc (valnum IN NUMBER);
END my_pkg;
```

The URL to invoke the first version of the procedure looks similar to:

```
http://www.acme.com:9000/pls/mydad/my_pkg.my_proc?valvc2=input
```

The URL to invoke the second version of the procedure looks similar to:

```
http://www.acme.com:9000/pls/mydad/my_pkg.my_proc?valnum=34
```

Overloading and PL/SQL Arrays

If you have overloaded PL/SQL procedures where the parameter names are identical, but the data type is `owa_util.ident_arr` (a table of varchar2) for one procedure and a scalar type for another procedure, mod_plsql can still distinguish between the two procedures. For example, if you have the following procedures:

```
CREATE PACKAGE my_pkg AS
  PROCEDURE my_proc (val IN VARCHAR2); -- scalar data type
  PROCEDURE my_proc (val IN owa_util.ident_arr); -- array data type
END my_pkg;
```

Each of these procedures has a single parameter of the same name, `val`.

When mod_plsql gets a request that has only one value for the `val` parameter, it invokes the procedure with the scalar data type.

Example 3–4 Sending a URL to Execute the Scalar Version of a Procedure

Send the following URL to execute the scalar version of the procedure:

```
http://www.acme.com:9000/pls/mydad/my_proc?val=john
```

When mod_plsql gets a request with more than one value for the `val` parameter, it then invokes the procedure with the array data type.

Example 3–5 Sending a URL to Execute the Array Version of a Procedure

Send the following URL to execute the array version of the procedure:

```
http://www.acme.com:9000/pls/mydad/my_proc?val=john&val=sally
```

To ensure that the array version executes, use hidden form elements on your HTML page to send dummy values that are checked and discarded in your procedure.

3.6.2 Flexible Parameter Passing

You can have HTML forms from which users can select any number of elements. If these elements have different names, you would have to create overloaded procedures to handle each possible combination. Alternatively, you could insert hidden form elements to ensure that the names in the query string are consistent each time, regardless of what elements the user chooses. mod_plsql makes this operation easier by supporting flexible parameter passing to handle HTML forms where users can select any number of elements.

To use flexible parameter passing for a URL-based procedure invocation, prefix the procedure with an exclamation mark (!) in the URL. You can use two or four parameters. The two parameter interface provides improved performance with mod_plsql. The four parameter interface is supported for compatibility.

3.6.2.1 Two Parameter Interface

```
procedure [proc_name]
     (name_array IN [array_type],
     value_array IN  [array_type]);
```

Table 3–2 Two Parameter Interface Parameters

Parameter	Description
`proc_name` (required)	The name of the PL/SQL procedure that you are invoking.
`name_array`	The names from the query string (indexed from 1) in the order submitted.
`value_array`	The values from the query string (indexed from 1) in the order submitted.
`array_type` (required)	Any PL/SQL index-by table of varchar2 type (Example, owa.vc_arr).

Example 3–6 Two Parameter Interface

If you send the following URL:

```
http://www.acme.com:9000/pls/mydad/!scott.my_proc?x=john&y=10&z=doe
```

The exclamation mark prefix (!) instructs mod_plsql to use flexible parameter passing. It invokes procedure `scott.myproc` and passes it the following two arguments:

```
name_array ==> ('x', 'y', 'z')
value_array ==> ('john', '10', 'doe')
```

> **Note:** When using this style of Flexible Parameter Passing, the procedure must be defined with the parameters `name_array` and `value_array`. The datatypes of these arguments should match the datatypes shown in the example.

3.6.2.2 Four Parameter Interface

The four parameter interface is supported for compatibility.

```
procedure [proc_name]
     (num_entires IN NUMBER,
```

```
name_array  IN  [array_type],
value_array IN  [array_type],
reserved in [array_type]);
```

Table 3–3 Four Parameter Interface Parameters

Parameter	Description
`proc_name` (required)	The name of the PL/SQL procedure that you are invoking.
`num_entries`	The number of name_value pairs in the query string
`name_array`	The names from the query string (indexed from 1) in the order submitted.
`value_array`	The values from the query string (indexed from 1) in the order submitted.
`reserved`	Not used. It is reserved for future use.
`array_type` (required)	Any PL/SQL index-by table of varchar2 type (Example, owa.vc_arr).

Example 3–7 Four Parameter Interface

If you send the following URL, where the query_string has duplicate occurrences of the name "x":

```
http://www.acme.com:9000/pls/mydad/!scott.my_pkg.my_proc?x=a&y=b&x=c
```

The exclamation mark prefix (!) instructs mod_plsql to use flexible parameter passing. It invokes procedure `scott.my_pkg.myproc` and passes it the following arguments:

```
num_entries ==> 3
name_array ==> ('x', 'y', 'x');
value_array ==> ('a', 'b', 'c')
reserved ==> ()
```

> **Note:** When using this style of Flexible Parameter Passing, the procedure must be defined with the parameters `num_entries`, `name_array`, `value_array`, and `reserved`. The datatypes of these arguments should match the datatypes shown in the example.

3.6.3 Large Parameter Passing

The values passed as scalar arguments and the values passed as elements to the index-by table of varchar2 arguments can be up to 32K in size.

For example, when using flexible parameter passing (described in Section 3.6.2, "Flexible Parameter Passing"), each name or value in the `query_string` portion of the URL gets passed as an element of the `name_array` or `value_array` argument to the procedure being invoked. These names or values can be up to 32KB in size.

3.7 File Upload and Download

mod_plsql enables you to:

- Upload and download files as raw byte streams without any character set conversions. The files are uploaded into the document table. A primary key is passed to the PL/SQL upload handler routine so that it can retrieve the appropriate table row.
- Specify one document table for each DAD so that uploaded files from different applications are not mixed. Also, the Direct BLOB (Binary Large Object) Download feature enables you to download content from any database table.
- Provide access to files in these tables through a URL format that doesn't use query strings, for example:

  ```
  http://www.acme.com:9000/pls/mydad/docs/cs250/lecture1.htm
  ```

  This is required to support uploading a set of files that have relative URL references to each other.
- Upload multiple files for each form submission.
- Upload files into LONG RAW and BLOB types of columns in the document table.

This section discusses the following:

3.7.1 Document Table Definition

You can specify the document storage table for each DAD. The document storage table must have the following definition:

```
CREATE TABLE [table_name] (
     NAME           VARCHAR2(256) UNIQUE NOT NULL,
     MIME_TYPE      VARCHAR2(128),
     DOC_SIZE       NUMBER,
     DAD_CHARSET    VARCHAR2(128),
     LAST_UPDATED   DATE,
     CONTENT_TYPE   VARCHAR2(128),
     [content_column_name] [content_column_type]
     [ , [content_column_name] [content_column_type]]
);
```

Users can choose the `table_name`. The `content_column_type` type must be either LONG RAW or BLOB.

The `content_column_name` depends on the corresponding `content_column_type`:

- If the content_column_type is LONG RAW, the `content_column_name` must be CONTENT.
- If the content_column_type is BLOB, the `content_column_name` must be BLOB_CONTENT.

An example of legal document table definition is:

```
CREATE TABLE MYDOCTABLE (
  NAME               VARCHAR(256)   UNIQUE NOT NULL,
  MIME_TYPE          VARCHAR(128),
  DOC_SIZE           NUMBER,
  DAD_CHARSET        VARCHAR(128),
  LAST_UPDATED       DATE,
  CONTENT_TYPE       VARCHAR(128),
  CONTENT            LONG RAW,
  BLOB_CONTENT       BLOB ;
);
```

3.7.1.1 Semantics of the CONTENT Column

The contents of the table are stored in a content column. There can be more than one content column in a document table. However, for each row in the document table, only one of the content columns is used. The other content columns are set to NULL.

3.7.1.2 Semantics of the CONTENT_TYPE Column

The `content_type` column tracks in which content column the document is stored. When a document is uploaded, mod_plsql sets the value of this column to the type name.

For example, if a document was uploaded into the BLOB_CONTENT column, then the `CONTENT_TYPE` column for the document is set to the string 'BLOB'.

3.7.1.3 Semantics of the LAST_UPDATED Column

The `LAST_UPDATED` column reflects a document's creation or last modified time. When a document is uploaded, mod_plsql sets the `LAST_UPDATED` column for the document to the database server time.

If an application then modifies the contents or attributes of the document, it must also update the `LAST_UPDATED` time.

mod_plsql uses the `LAST_UPDATED` column to check and indicate to the HTTP client (browser) if the browser can use a previously cached version of the document. This reduces network traffic and improves server performance.

3.7.1.4 Semantics of the DAD_CHARSET Column

The `DAD_CHARSET` column keeps track of the character set setting at the time of the file upload. This column is reserved for future use.

3.7.2 Old Style Document Table Definition

For backward capability with the document model used by older releases of WebDB 2.x, mod_plsql also supports the following old definition of the document storage table where the CONTENT_TYPE, DAD_CHARSET and LAST_UPDATED columns are not present.

```
/* older style document table definition (DEPRECATED) */
CREATE TABLE [table_name]
(
    NAME         VARCHAR2(128),
    MIME_TYPE    VARCHAR2(128),
    DOC_SIZE     NUMBER,
    CONTENT      LONG RAW
);
```

3.7.3 Configuration Parameters for Document Upload/Downloading

The following configuration parameters in the DAD affect a document upload/download operation:

- "PlsqlDocumentTablename"
- "PlsqlDocumentPath (Document Access Path)"
- "PlsqlDocumentProcedure (Document Access Procedure)"
- "PlsqlUploadAsLongRaw"

Example 3–8 Parameters for Document Upload/Download

If the configuration for these parameters in a DAD is as follows:

```
PlsqlDocumentTablename   scott.my_document_table
PlsqlUploadAsLongRaw   html
PlsqlDocumentPath   docs
PlsqlDocumentProcedure   scott.my_doc_download_procedure
```

then:

- mod_plsql will retrieve data from, or store to a database table called **my_document_table** in the **scott** schema.
- All file extensions except `.html` will be uploaded to the document table as BLOBs. All files with `.html` extension will be uploaded as *Long Raw*.
- All URLs which have the keyword **docs** immediately following the DAD location will result in invocation of the procedure **scott.my_doc_download_procedure**.

  Typically, this procedure will call **wpg_docload.download_file** to initiate a file download for a file whose name is based on the URL specification.

A simple example with the preceding configuration is:

```
http://www.acme.com:9000/pls/dad/docs/index.html
```

This results in downloading of the file `index.html` from the *Long Raw* column of the database table **scott.my_document_table**. Note that the application procedure has full control on the file download to initiate, and has the flexibility to define a more complex `PlsqlDocumentProcedure` that implements file-level access controls and versioning.

Note: The application defined procedure **scott.my_doc_download_procedure** has to be defined without arguments, and should rely on the CGI environment variables to process the request.

3.7.3.1 PlsqlDocumentTablename

The `PlsqlDocumentTablename` parameter specifies the table for storing documents when file uploads are performed through this DAD.

Syntax:

```
PlsqlDocumentTablename  [document_table_name]

PlsqlDocumentTablename  my_documents
```

or,

```
PlsqlDocumentTablename  scott.my_document_table
```

3.7.3.2 PlsqlDocumentPath (Document Access Path)

The `PlsqlDocumentPath` parameter specifies the path element to access a document. The `PlsqlDocumentPath` parameter follows the DAD name in the URL. For example, if the document access path is `docs`, then the URL would look similar to:

```
http://www.acme.com:9000/pls/mydad/docs/myfile.htm
```

The `mydad` is the DAD name and `myfile.htm` is the file name.

Syntax:

```
PlsqlDocumentPath  [document_access_path_name]
```

3.7.3.3 PlsqlDocumentProcedure (Document Access Procedure)

The `PlsqlDocumentProcedure` procedure is an application-specified procedure. It has no parameters and processes a URL request with the document access path. The document access procedure calls `wpg_docload.download_file(filename)` to download a file. It knows the filename based on the URL specification. For example, an application can use this to implement file-level access controls and versioning. An example of this is in Section 3.7.7, "File Download".

Syntax:

```
PlsqlDocumentProcedure  [document_access_procedure_name]
```

Example 3–9 PlsqlDocumentProcedure

```
PlsqlDocumentProcedure  my_access_procedure
```

or,

```
PlsqlDocumentProcedure  scott.my_pkg.my_access_procedure
```

3.7.3.4 PlsqlUploadAsLongRaw

The DAD parameter, `PlsqlUploadAsLongRaw`, configures file uploads based on their file extensions. The value of a `PlsqlUploadAsLongRaw` DAD parameter is a *one-entry-for-each-line* list of file extensions. Files with these extensions are uploaded by mod_plsql into the content column of `LONG RAW` type in the document table. Files with other extensions are uploaded into the BLOB content column.

The file extensions can be text literals (jpeg, gif, and so on) or an asterisk (*) matches any file whose extension has not been listed in the `PlsqlUploadAsLongRaw` setting.

Syntax:

```
PlsqlUploadAsLongRaw  [file_extension]
PlsqlUploadAsLongRaw *
```

`[file_extension]` is an extension for a file (with or without the '.' character, for example, 'txt' or '.txt') or the wildcard character *.

Example 3–10 PlsqlUploadAsLongRaw

```
PlsqlUploadAsLongRaw  html
PlsqlUploadAsLongRaw  txt
PlsqlUploadAsLongRaw  *
```

3.7.4 File Upload

To send files from a client machine to a database, create an HTML page that contains:

- A FORM tag whose `enctype` attribute is set to `multipart/form-data` and whose `action` attribute is associated with a mod_plsql procedure call, referred to as the "action procedure."
- An INPUT element whose type and name attributes are set to file. The `INPUT type="file"` element enables a user to browse and select files from the file system.

When a user clicks **Submit**, the following events occur:

1. The browser uploads the file specified by the user as well as other form data to the server.
2. mod_plsql stores the file contents in the database in the document storage table. The table name is derived from the `PlsqlDocumentTablename` DAD setting.
3. The action procedure specified in the `action` attribute of the FORM is run (similar to invoking a mod_plsql procedure without file upload).

> **Note:** The parsing of HTML documents is deprecated in mod_plsql. mod_plsql used to parse the content of an HTML file when it was uploaded, and identified other files that the HTML document was referring to. This information was then stored into a table. The table name was constructed by appending the name of the document table with "part". This functionality was found to be not of use to customers and has been deprecated, starting in version 9.0.4 of mod_plsql.

The following example shows an HTML form that lets a user select a file from the file system to upload. The form contains other fields to provide information about the file.

```
<html>
   <head>
      <title>test upload</title>
   </head>
   <body>
   <FORM enctype="multipart/form-data"
      action="pls/mydad/write_info"
      method="POST">
      <p>Author's Name:<INPUT type="text" name="who">
      <p>Description:<INPUT type="text" name="description"><br>
      <p>File to upload:<INPUT type="file" name="file"><br>
      <p><INPUT type="submit">
   </FORM>
   </body>
</html>
```

When a user clicks **Submit** on the form:

1. The browser uploads the file listed in the `INPUT type="file"` element.
2. The `write_info` procedure then runs.
3. The procedure writes information from the form fields to a table in the database and returns a page to the user.

Note: The action procedure does not have to return anything to the user, but it is a good idea to let the user know whether the Submit succeeded or failed, as shown subsequently.

```
procedure write_info (
     who         in varchar2,
     description in varchar2,
     file        in varchar2) as
begin
     insert into myTable values (who, description, file);
     htp.htmlopen;
     htp.headopen;
     htp.title('File Uploaded');
     htp.headclose;
     htp.bodyopen;
     htp.header(1, 'Upload Status');
     htp.print('Uploaded ' || file || ' successfully');
     htp.bodyclose;
     htp.htmlclose;
end;
```

The filename obtained from the browser is prefixed with a generated directory name to reduce the possibility of name conflicts. The "action procedure" specified in the form renames this name. So, for example, when `/private/minutes.txt` is uploaded, the name stored in the table by the mod_plsql is `F9080/private/minutes.txt`. The application can rename this in the called stored procedure. For example, the application can rename it to `scott/minutes.txt`.

See Also: RFC 1867, "Form-Based File Upload in HTML" (IETF)

3.7.5 Specifying Attributes (Mime Types) of Uploaded Files

In addition to renaming the uploaded file, the stored procedure can alter other file attributes. For example, the form in the example from Section 3.7.4, "File Upload" could display a field for allowing the user to input the uploaded document's Multipurpose Internet Mail Extension (MIME) type.

The MIME type can be received as a parameter in `write_info`. The document table would then store the mime type for the document instead of the default mime type that is parsed from the multipart form by mod_plsql when uploading the file.

3.7.6 Uploading Multiple Files

To send multiple files in a single submit, the upload form must include multiple <INPUT type="file" name="file"> elements. If more than one file INPUT element defines `name` to be of the same name, then the action procedure must declare that parameter name to be of type `owa.vc_arr`. The names defined in the file INPUT elements could also be unique, in which case, the action procedure must declare each of them to be of varchar2. For example, if a form contained the following elements:

```
<INPUT type="file" name="textfiles">
<INPUT type="file" name="textfiles">
<INPUT type="file" name="binaryfile">
```

As a result, the action procedure must contain the following parameters:

```
procedure handle_text_and_binary_files(textfiles IN owa.vc_arr, binaryfile IN
```

varchar2).

3.7.7 File Download

After you have uploaded files to the database, you can download them from the database in three different ways, as described here:

- Define a PL/SQL procedure that calls `wpg_docload.download_file(file_name)` to download file `file_name`.
- Define a virtual path (`PlsqlDocumentPath`) for document downloads in the DAD configuration, and associate a user-defined procedure with that path (`PlsqlDocumentProcedure`). When mod_plsql detects, immediately after the DAD name, the virtual path specified by `PlsqlDocumentPath`, it automatically invokes the user-defined procedure (`PlsqlDocumentProcedure`), which in turn must call `wpg_docload.download_file(file_name)` to initiate download of file `file_name`. This user-defined procedure should have a prototype such that it be invoked without being passed any additional arguments.

  For example, if the DAD "mydad" specifies that `PlsqlDocumentPath` as "`docs`" and `PlsqlDocumentProcedure` is configured `as "myschema.pkg.process_download"`, then the procedure "`myschema.pkg.process_download`" is invoked by mod_plsql whenever the URL is of the format `http://www.acme.com:9000/pls/mydad/docs/myfile.htm`.

  An example implementation of process_download is:

```
procedure process_download is
v_filename varchar2(255);
begin
     -- getfilepath() uses the SCRIPT_NAME and PATH_INFO cgi
     -- environment variables to construct the full path name of
     -- the file URL, and then returns the part of the path name
     -- following '/docs/'
     v_filename := getfilepath;
     select name into v_filename from plsql_gateway_doc
                     where UPPER(name) = UPPER(v_filename);
     -- now we call docload.download_file to initiate
     -- the download.
     wpg_docload.download_file(v_filename);
     exception
        when others then
           v_filename := null;
end process_download;
```

- Use the Direct Binary Large Object (BLOB) Download mechanism to download a BLOB from any database table. You do this by calling a PL/SQL procedure that streams the standard HTTP headers, such as mime-type and content-length, and then invokes `wpg_docload.download_file(blob_name)` to download BLOB `b,ob_name`, as shown here:

  1. Create a stored procedure that calls `wpg_docload.download_file(blob)` where `blob` is of data type BLOB. Since mod_plsql has no information about the contents in the BLOB, you must supply them.
  2. Setup the Content-Type and other headers.

     In the following example, the procedure uses the name from the argument to select a BLOB from a table and initiates the Direct BLOB download:

```
create or replace procedure download_blob(name in varchar2) is
```

```
myblob blob;
begin
```

– Select the BLOB out of mytable using the name argument.

```
select blob_data into myblob from mytable where blob_name = name;
```

– Setup headers which describes the content.

```
owa_util.mime_header('text/html', FALSE);
htp.p('Content-Length: ' || dbms_lob.getlength(myblob));
owa_util.http_header_close;
```

– Initiate Direct BLOB download.

```
wpg_docload.download_file(myblob);
end;
```

The structure of the `mytable` table is as follows:

```
create table mytable
(
blob_name varchar2(128),
blob_data blob
);
```

When document downloads are initiated but the "Direct BLOB download" is not used, then the argument passed to `wpg_docload.download_file` should be able to uniquely identify the filename to download from the document table. While streaming back the content for such documents, mod_plsql will generate the HTTP response headers based on the information stored in other columns of the document table entry for the filename. The MIME_TYPE, DOC_SIZE, and LAST_UPDATED columns will be used to add response headers of "Content-Type", "Content-Length", and "If-Modified-Since" headers respectively.

Note: Every time you call the `wpg_docload.download_file` API from a procedure, a file download operation is initiated by mod_plsql. During such operations, no other HTML content generated by the procedure is passed back to the browser.

Downloading Documents with Multibyte Characters

If you are using mod_plsql on a Windows platform, and are running against a multibyte database that contains the special character 0x5c as part of a multibyte character set (Japanese or Korean), you must edit the DAD used to access the application, by modifying the file `dads.conf`. To do this:

1. Open the file *`ORACLE_HOME`*`/Apache/modplsql/conf/dads.conf`.
2. Locate the DAD used to access the application.
3. Add the line "`WindowsFileConversion Off`" to this DAD entry.
4. Save the file.
5. Restart Oracle HTTP Server.

If you do not update the DAD configuration, you will experience failures while downloading documents which contain 0x5c in the filename. For example:

- In OracleAS Portal, you will see the download error:

```
Error: Document not found (WWC-46000)
```

- When using mod_plsql against your own PL/SQL application, file downloads will result in the error:

```
HTTP-404 Not Found
```

3.8 Path Aliasing (Direct Access URLs)

Path Aliasing enables applications using mod_plsql to provide direct reference to its objects using simple URLs. The Path Aliasing functionality is a generalization of how the document download functionality is provided. The following configuration parameters in the DAD are used for Path Aliasing:

- PlsqlPathAlias
- PlsqlPathAliasProcedure

For Example, if the configuration for these parameters in a DAD is as follows:

```
PlsqlPathAlias   myalias
PlsqlPathAliasProcedure   scott.my_path_alias_procedure
```

then, all URLs that have the keyword **myalias** immediately following the DAD location will invoke the procedure `scott.my_path_alias_procedure`. Based on the URL specification, this procedure can initiate an appropriate response.

Note: The application defined procedure `scott.my_path_alias_procedure` has to be defined to take one argument of type `varchar2` called **p_path**. This argument will receive everything following the keyword used in `PlsqlPathAlias`.

For example, in the preceding configuration, the URL:

`http://www.acme.com:9000/pls/dad/myalias/MyFolder/MyItem`

will result in the procedure **scott.my_path_alias_procedure** receiving the argument **MyFolder/MyItem**.

3.9 Common Gateway Interface (CGI) Environment Variables

The OWA_UTIL package provides an API to get the values of CGI environment variables. The variables provide context to the procedure being executed through mod_plsql. Although mod_plsql is not operated through CGI, the PL/SQL application invoked from mod_plsql can access these CGI environment variables.

The list of CGI environment variables is as follows:

- HTTP_AUTHORIZATION
- DAD_NAME
- DOC_ACCESS_PATH
- HTTP_ACCEPT
- HTTP_ACCEPT_CHARSET
- HTTP_ACCEPT_LANGUAGE

- HTTP_COOKIE
- HTTP_HOST
- HTTP_PRAGMA
- HTTP_REFERER
- HTTP_USER_AGENT
- PATH_ALIAS
- PATH_INFO
- HTTP_ORACLE_ECID
- DOCUMENT_TABLE
- REMOTE_ADDR
- REMOTE_HOST
- REMOTE_USER
- REQUEST_CHARSET (refer to Section 3.9.2.1, "REQUEST_CHARSET CGI Environment Variable")
- REQUEST_IANA_CHARSET (refer to Section 3.9.2.2, "REQUEST_IANA_CHARSET CGI Environment Variable")
- REQUEST_METHOD
- REQUEST_PROTOCOL
- SCRIPT_NAME
- SCRIPT_PREFIX
- SERVER_NAME
- SERVER_PORT
- SERVER_PROTOCOL

A PL/SQL application can get the value of a CGI environment variable using the owa_util.get_cgi_env interface.

Syntax:

```
owa_util.get_cgi_env(param_name in varchar2) return varchar2;
```

param_name is the name of the CGI environment variable. param_name is case-insensitive.

3.9.1 Adding and Overriding CGI Environment Variables

The `PlsqlCGIEnvironmentList` DAD parameter is a *one-entry-for-each-line* list of name and value pairs that can override any environment variables or add new ones. If the name is one of the original environment variables (as listed in Section 3.9, "Common Gateway Interface (CGI) Environment Variables"), that environment variable is overridden with the given value. If the name is not in the original list, a new environment variable is added into the list with that same name and value given in the parameter.

Note: Refer to the *Oracle HTTP Server Administrator's Guide* for information about the mod_plsql Configuration Files.

If no value is specified for the parameter, then the value is obtained from the Oracle HTTP Server. With Oracle HTTP Server, you can pass the DOCUMENT_ROOT CGI Environment variable by specifying:

```
PlsqlCGIEnvironmentList DOCUMENT_ROOT
```

New environment variables passed in through this configuration parameter are available to the PL/SQL application through the owa_util.get_cgi_env interface.

Example 3–11 PlsqlCGIEnvironmentList with Environment Variable Overrides

```
PlsqlCGIEnvironmentList SERVER_NAME=myhost.mycompany.com
PlsqlCGIEnvironmentList REMOTE_USER=testuser
```

This example overrides the SERVER_NAME and the REMOTE_USER CGI environment variables with the given values since they are part of the original list.

Example 3–12 PlsqlCGIEnvironmentList with New Environment Variable

```
PlsqlCGIEnvironmentList MYENV_VAR=testing
PlsqlCGIEnvironmentList SERVER_NAME=
PlsqlCGIEnvironmentList REMOTE_USER=user2
```

This example overrides the SERVER_NAME and the REMOTE_USER variables. The SERVER_NAME variable is deleted since there is no value given to it. A new environment variable called MYENV_VAR is added since it is not part of the original list. It is assigned the value of "testing".

3.9.2 PlsqlNLSLanguage

For mod_plsql, the Globalization Support setting is controlled by the DAD level setting of `PlsqlNLSLanguage`. If PlsqlNLSLanguage is not configured at the DAD level, the Globalization Support configuration is picked up from the environment setting of the Oracle NLS_LANG parameter. For details on this parameter, refer to the "mod_plsql" section in the *Oracle HTTP Server Administrator's Guide*.

3.9.2.1 REQUEST_CHARSET CGI Environment Variable

The CGI environment variable REQUEST_CHARSET is set based on the setting of `PlsqlNLSLanguage`. If `PlsqlNLSLanguage` is not configured at the DAD level, the Globalization Support setting is picked up from the environment setting of the Oracle NLS_LANG parameter.

The PL/SQL application can access this information by a function call. For example:

```
owa_util.get_cgi_env('REQUEST_CHARSET');
```

3.9.2.2 REQUEST_IANA_CHARSET CGI Environment Variable

This is the IANA (Internet Assigned Number Authority) equivalent of the REQUEST_CHARSET CGI environment variable. IANA is an authority that globally coordinates the standards for charsets on the Internet.

The PL/SQL application can access this information by a function call. For example:

```
owa_util.get_cgi_env('REQUEST_IANA_CHARSET');
```

3.10 Using Caching with PL/SQL Based Web Applications

Caching can improve the performance of PL/SQL based Web applications. To improve performance, you can cache Web content generated by PL/SQL procedures in the middle tier and decrease the database workload.

This section covers the techniques used in caching, including the following:

- Using the Validation Technique - An application asks the server if the page has been modified since it was last presented.
- Using the Expires Technique - Based upon a specific time period, the PL/SQL based Web application determines the page will be cached, or should be generated again.
- System- and User-level Caching with PL/SQL Based Web Applications - This is valid whether you are using the Validation Technique or the Expires Technique. The level of caching is determined by whether a page is cached for a particular user or for every user in the system.

These techniques and levels are implemented using `owa_cache` packages located inside the PL/SQL Web Toolkit.

See Also: *Oracle Application Server PL/SQL Web Toolkit Reference*

3.10.1 Using the Validation Technique

In general, the validation technique basically asks the server if the page has been modified since it was last presented. If it has not been modified, the cached page will be presented to the user. If the page has been modified, a new copy will be retrieved, presented to the user and then cached.

There are two methods which use the Validation Technique: Last-Modified method, and the Entity Tag method. The next two sections show how these techniques are used in the HTTP protocol. Although the PL/SQL Gateway does not use the HTTP protocol, many of the same principles are used.

3.10.1.1 Last-Modified

When a Web page is generated using the HTTP protocol, it contains a **Last-Modified** Response Header. This header indicates the date, relative to the server, of the content that was requested. Browsers save this date information along with the content. When subsequent requests are made for the URL of the Web page, the browser then:

1. Determines if it has a cached version.
2. Extracts the date information.
3. Generates the Request Header **If-Modified-Since**.
4. Sends the request the server.

Cache-enabled servers look for the **If-Modified-Since** header and compare it to the date of their content. If the two match, an HTTP Response status header such as "HTTP/1.1 304 Not Modified" is generated, and no content is streamed. After receiving this status code, the browser can reuse its cache entry because it has been validated.

If the two do not match, an HTTP Response header such as "HTTP/1.1 200 OK" is generated and the new content is streamed, along with a new **Last-Modified Response h**eader. Upon receipt of this status code, the browser must replace its cache entry with the new content and new date information.

3.10.1.2 Entity Tag Method

Another validation method provided by the HTTP protocol is the **ETag** (Entity Tag) Response and Request header. The value of this header is a string that is opaque to the browser. Servers generate this string based on their type of application. This is a more generic validation method than the **If-Modified-Since** header, which can only contain a date value.

The **ETag** method works very similar to the Last Modified method. Servers generate the ETag as part of the Response Header. The browser stores this opaque header value along with the content that is steamed back. When the next request for this content arrives, the browser passes the **If-Match** header with the opaque value that it stored to the server. Because the server generated this opaque value, it is able to determine what to send back to the browser. The rest is exactly like the **Last-Modified** validation method as described earlier.

3.10.1.3 Using the Validation Technique for mod_plsql

Using HTTP validation caching as a framework, the following is the Validation Model for mod_plsql.

PL/SQL based Web applications that want to control the content being served should use this type of caching. This technique offers some moderate performance gains. One example of this would be a Web application that serves dynamic content that could change at any given time. In this case, the Web application needs full control over what is being served. Validation caching always asks the Web application whether the cached content is stale or not before serving it back to the browser.

Figure 3–2 shows the use of the validation technique for mod_plsql.

1. The Oracle HTTP Server receives a PL/SQL procedure request from a client server. The Oracle HTTP Server routes the request to mod_plsql.
2. mod_plsql prepares the request.
3. mod_plsql invokes the PL/SQL procedure in the Web application. mod_plsql passes the usual Common Gateway Interface (CGI) environment variables to the Web application.
4. The PL/SQL procedure generates content to pass back. If the PL/SQL procedure decides that the generated content is cacheable, it calls the `owa_cache` procedure from the PL/SQL Web Toolkit to set the tag and cache level:

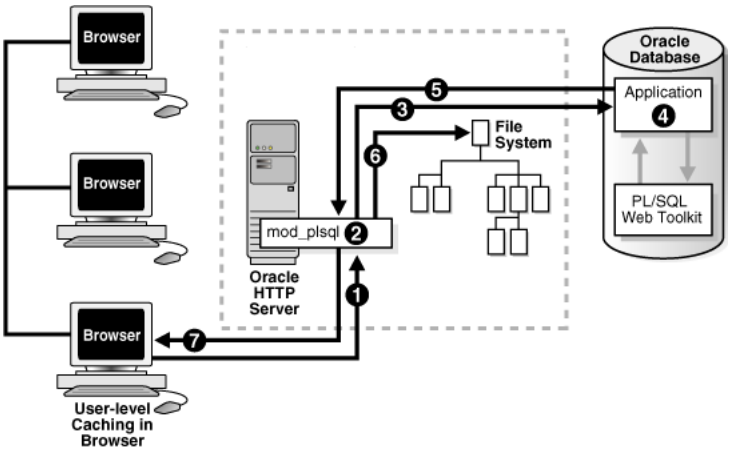
   ```
   owa_cache.set_cache(p_etag, p_level);
   ```

Table 3–4 Validation Model Parameters

Parameter	Description
`set_cache` procedure	Sets up the headers to notify mod_plsql that the content being streamed back can be cached. Then, the mod_plsql caches the content on the local file system along with the tag and caching level information as it is streamed back to the browser.
`p_etag`	The string that the procedure generates to tag the content.
`p_level`	The caching level: `SYSTEM` for system level or `USER` for user level.

1. The HTML is returned to mod_plsql.
2. mod_plsql stores the cacheable content in its file system for the next request.
3. The Oracle HTTP Server sends the response to the client browser.

Figure 3–2 Validation Technique



3.10.1.4 Second Request Using the Validation Technique

Using the Validation Technique for mod_plsql, a second request is made by the client browser for the same PL/SQL procedure.

Figure 3–3 shows the second request using the Validation Technique.

1. mod_plsql detects that it has a cached content for the request.
2. mod_plsql forwards the same tag and caching level information (from the first request) to the PL/SQL procedure as part of the CGI environment variables.
3. The PL/SQL procedure uses these caching CGI environment variables to check if the content has changed. It does so by calling the following `owa_cache` functions from the PL/SQL Web Toolkit:

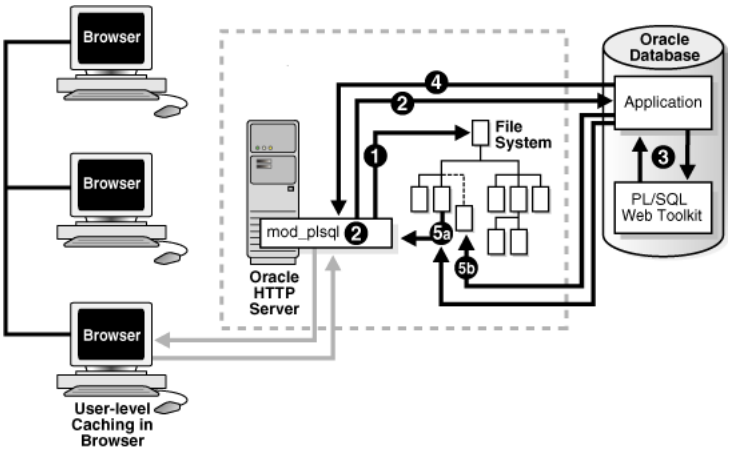
   ```
   owa_cache.get_etag;
   owa_cache.get_level;
   ```

   These owa functions get the tag and caching level.
4. The Web application sends the caching information to mod_plsql.
5. Based on that information determines whether the content needs to be regenerated or can be served from the cache.

   a. If the content is still the same, the procedure calls the `owa_cache.set_not_modified` procedure and generates no content. This causes mod_plsql to use its cached content. The cached content is directly streamed back to the browser.

   b. If the content has changed, it generates the new content along with a new tag and caching level. mod_plsql replaces its stale cached copy with a new one and updates the tag and caching level information. The newly generated content is streamed back to the browser.

Figure 3–3 Validation Technique-Second Request



3.10.2 Using the Expires Technique

In the validation model, mod_plsql always asks the PL/SQL procedure if it can serve the content from the cache. In the expires model, the procedure pre-establishes the content validity period. Therefore, mod_plsql can serve the content from its cache without asking the procedure. This further improves performance because no interaction with the database is required.

This caching technique offers the best performance. Use if your PL/SQL based Web application is not sensitive to serving stale content. One example of this is an application that generates news daily. The news can be set to be valid for 24 hours. Within the 24 hours, the cached content is served back without contacting the application. This is essentially the same as serving a file. After 24 hours, mod_plsql will again fetch new content from the application.

Assume the same scenario described for the Validation model, except the procedure uses the Expires model for caching.

Figure 3–4 shows the use of the expires technique for mod_plsql.

1. The Oracle HTTP Server receives a PL/SQL Server Page request from a client server. The Oracle HTTP Server routes the request to mod_plsql.
2. The request is forwarded by mod_plsql to the Oracle Database.
3. mod_plsql invokes the PL/SQL procedure in the application and passes the usual Common Gateway Interface (CGI) environment variables to the application.
4. The PL/SQL procedure generates content to pass back. If the PL/SQL procedure decides that the generated content is cacheable, it calls the `owa_cache` procedure from the PL/SQL Web Toolkit to set the validity period and cache level:

```
owa_cache.set_expires(p_expires, p_level);
```

Table 3–5 Expires Model Parameters

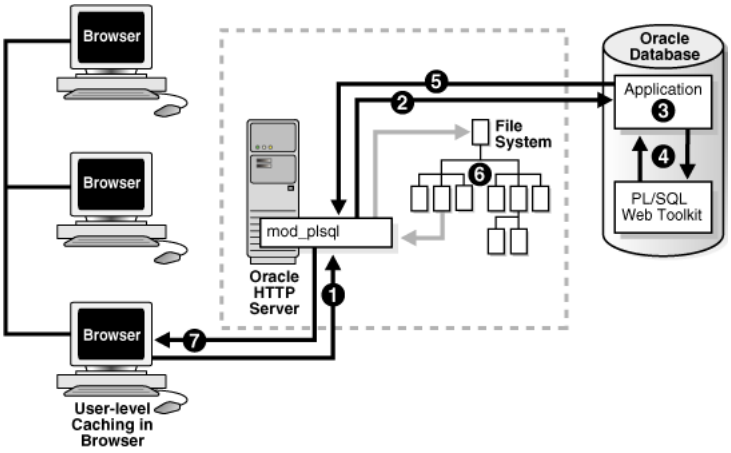
Parameter	Description
`set_expires` procedure	Sets up the headers to notify mod_plsql that Expires caching is being used. mod_plsql then caches the content to the file system along with the validity period and caching level information.
`p_expires`	Number of minutes that the content is valid.

Table 3–5 (Cont.) Expires Model Parameters

Parameter	Description
`p_level`	Caching level.

1. The HTML is returned to mod_plsql.
2. mod_plsql stores the cacheable content in its file system for the next request.
3. The Oracle HTTP Server sends the response to the client browser.

Figure 3–4 The Expires Technique



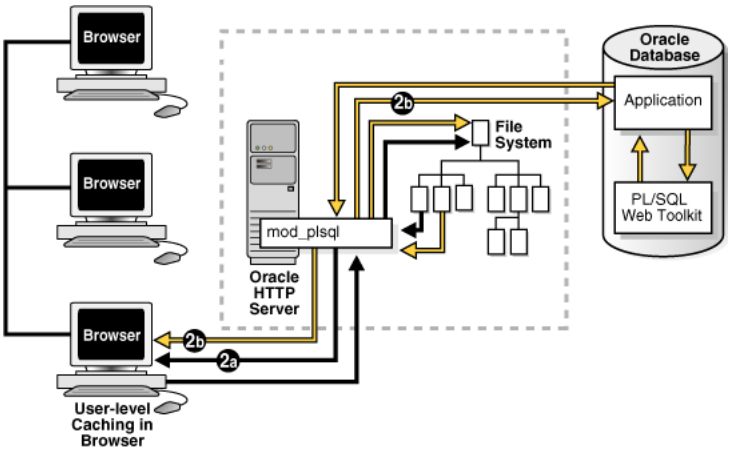
Second Request Using the Expires Technique

Using the same expires model explained earlier, a second request is made by the client browser for the same PL/SQL procedure.

Figure 3–5 shows the second request using the Expires Technique.

1. mod_plsql detects that it has a cached copy of the content that is expires-based.
2. mod_plsql checks the content's validity by taking the difference between the current time and the time this cache file was created.
   a. If this difference is within the validity period, the cached copy is still fresh and will be used without any database interaction. The cached content is directly streamed back to the browser.
   b. If the difference is not within the validity period, the cached copy is stale. mod_plsql invokes the PL/SQL procedure and generates new content. The procedure then decides whether to use expires-based caching again. If so, it also determines the validating period for this new content. The newly generated content is streamed back to the browser.

Figure 3–5 The Expires Technique-Second Request



3.10.3 System- and User-level Caching with PL/SQL Based Web Applications

A PL/SQL procedure determines whether generated content is system-level content or user-level. This helps the PL/SQL Gateway cache to store less redundant files if more than one user is looking at the same content. It decides this by:

- For **system-level** content, the procedure passes the string `SYSTEM` as the caching level parameter to the `owa_cache` functions (`set_cache` for validation model or `set_expires` for expires model). This is for every user that shares the cache.

  By using system-level caching, you can save both space in your file system and time for all users in the system. One example of this would be a Web application that generates content that is intended for everybody using the Web application. By caching the content with the system-level setting, only one copy of the content is cached in the file system. Furthermore, every user on that system benefits since the content is served directory from the cache.

- For **user-level** content, it passes the string `USER` as the parameter for the caching level. This is for a specific user that is logged in. The stored cache is unique for that user. Only that user can use the cache. The type of user is determined by the authentication mode. Refer to the following table for the different types of users.

Table 3–6 Type of User Determined by Authentication Mode

Authentication Mode	Type of User
Single Sign On (SSO)	Lightweight user
Basic	Database user
Custom	Remote user

For example, if no user customizes a PL/SQL based Web application, then the output can be stored in a system-level cache. There will be only one cache copy for every user on the system. User information is not used since the cache can be used by multiple users.

However, if a user customizes the application, a user-level cache is stored for that user only. All other users still use the system level cache. For a user-level cache hit, the user information is a criteria. A user-level cache always overrides a system-level cache.

See Also: Section 2.2, "Authenticating Users Through mod_plsql" for more information on authentication modes.

PL/SQL Web Toolkit Functions (owa_cache package)

Your decision whether to use the Validation technique or the Expires technique determines which `owa_cache` functions to call.

The `owa_cache` package contains procedures to set and get special caching headers and environment variables. These allow developers to use the PL/SQL Gateway cache more easily. This package should already be installed in your database.

Table 3–7 lists the primary functions to call.

Table 3–7 Primary owa_cache Functions

owa Functions	Purpose
`owa_cache.set_cache (p_etag IN varchar2, p_level IN varchar2)`	Validation Model - Sets up the headers. ■ `p_etag` parameter tags the generated content. ■ `p_level` parameter is the caching level to use.
`owa_cache.set_not_modified`	Validation Model - Sets up the headers to notify mod_plsql to use the cached content. Only used when a validation -based cache hit occurs.
`owa_cache.get_level`	Validation Model - Gets the caching level, `USER` or `SYSTEM`. Returns null if the cache is not hit.
`owa_cache.get_etag`	Validation Model - Gets the tag associated with the cached content. Returns null if the cache is not hit.
`owa_cache.set_expires(p_ expires IN number, p_level IN varchar2)`	Expires Model - Sets up the headers. ■ `p_expires` parameter is the number of minutes the content is valid. ■ `p_level` parameter is the caching level to use.

3.11 Performance Tuning Areas for mod_plsql

When tuning mod_plsql to improve the performance of PL/SQL based Web applications, it is important to be familiar with the internal working of mod_plsql. This section presents a basic overview of some mod_plsql functionality.

This section covers the following topics:

- Connection Pooling with mod_plsql
- Closing Pooled Database Sessions
- Detecting Dead Database Connections in a Connection Pool

3.11.1 Connection Pooling with mod_plsql

On UNIX platforms, the Database Server connection pooling logic supplied with mod_plsql can be best explained with an example. Consider the following typical scenario:

1. The Oracle HTTP Server listener is started. There are no database connections in the connection pool maintained by mod_plsql.
2. A browser makes a mod_plsql request (R1) for Database Access Descriptor (DAD) D1.

3. One of the Oracle HTTP Server processes (`httpd` process P1) starts servicing the request R1.
4. mod_plsql in process P1 checks its connection pool and finds that there are no database connections in its pool for that user request.
5. Based on the information in DAD D1, mod_plsql in process P1 opens a new database connection, services the PL/SQL request, and adds the database connection to its pool.
6. From this point on, all subsequent requests to process P1 for DAD D1 can now make use of the database connection pooled by mod_plsql.
7. If a request for DAD D1 gets picked up by another process (process P2), then mod_plsql in process P2 opens its own database connection, services the request, and adds the database connection to its pool.
8. From this point on, all subsequent requests to process P2 for DAD D1 can now make use of the database connection pooled by mod_plsql.
9. Now, assume that a request R2 is made for DAD D2 and this request gets routed to process P1.
10. mod_plsql in process P1 does not have any database connections pooled for DAD D2, and a new database session is created for DAD D2 and pooled after servicing the request. Process P1 now has two database connections pooled, one for DAD D1 and another for DAD D2.

    The important details in the example shown in steps 1-10 are:

    a. Each Oracle HTTP Server process serves all types of requests, such as static files requests, servlet requests, and mod_plsql requests. There is no control on which Oracle HTTP Server process services the next request.
    b. One Oracle HTTP Server process cannot use or share the connection pool created by another process.
    c. Each Oracle HTTP Server process pools at most one database connection for each DAD.
    d. User sessions are switched within a pooled database connection for a DAD. For DADs based on Oracle Application Server Single Sign-On (SSO), proxy authentication is used to switch the user session. For non-SSO users, using HTTP basic authentication with the username and password not in the DAD, users are re-authenticated on the same connection.
    e. Multiple DADs may point to the same database instance, but database connections are not shared across DADs even within the same process.
    f. Unused DADs do not result in any database connections.

In the worst-case scenario, the total number of database connections pooled by mod_plsql is a factor of the total number of active DADs multiplied by the number of Oracle HTTP Server (`httpd`) processes running at any given time for a single Oracle HTTP Server instance. If you have configured the Oracle HTTP Server processes to a high number, you need to configure the back-end database to handle a corresponding number of database sessions, and remember that this configuration value needs to be multiplied times the number of Oracle HTTP Server instances that use the back-end database.

For example, if there are three Oracle HTTP Server instances configured to spawn a maximum of 50 `httpd` processes each, plus two active DADs, you need to set up the

database to allow 300 (`3*50*2`) sessions. This number does not include any sessions that are needed to allow other Web applications to connect.

On Windows platforms, the Oracle HTTP Server runs as a single process. On such systems, the mod_plsql connection pool is shared across threads, and the total number of database connections is a factor of the number of concurrent requests for each DAD. Due to the sharing of database connections across threads, Section 4.3.4, "Two-Listener Strategy" does not apply to Windows systems.

3.11.2 Closing Pooled Database Sessions

Pooled database sessions are closed under the following circumstances:

1. When a pooled connection has been used to serve a configured number of requests.

   By default, each connection pooled by mod_plsql is used to service a maximum of 1000 requests, and then the database connection is shut down and reestablished on the next mod_plsql request. This is done to make sure that any resource leaks in the PL/SQL based Web application, or in the Oracle client server side, do not adversely affect the system. Change the default value of `1000` by tuning the DAD configuration parameter `PlsqlMaxRequestsPerSession`.

2. When a pooled connection has been idle for an extended period of time.

   By default, the cleanup thread in mod_plsql cleans up each pooled connection after 15 minutes of idle time. The value for idle session timeout is controlled by the configuration setting `PlsqlIdleSessionCleanupInterval`. If your site is accessing mod_plsql content less frequently, the idle session cleanup happens more frequently, and users will establish database connections that are not being used too often. In such situations, consider increasing the default setting of `PlsqlIdleSessionTimeoutInterval` for better performance. Note that keeping pooled database connections open for a longer time means additional load on the database to allow for more sessions.

3. On UNIX systems, when the Oracle HTTP Server process goes down.

   On UNIX systems, the Oracle HTTP Server configuration parameter `MaxRequestsPerChild` governs when an Oracle HTTP Server process will be shut down. For example, if this parameter is set to `5000`, each Oracle HTTP Server process would serve exactly 5000 requests before it is shut down. Oracle HTTP Server processes could also start up and shut down as part of Oracle HTTP Server maintenance based on the configuration parameters `MinSpareServers`, `MaxSpareServers`, and `MaxClients`.

   An incorrect configuration of Oracle HTTP Server could result in a setup where Oracle HTTP Server processes are started up and shut down heavily, resulting in ineffective mod_plsql connection pooling. For best performance, configure Oracle HTTP Server such that each Oracle HTTP Server process remains active for a certain period of time, thus preventing the processes from shutting down.

   > **See Also:** The section "mod_plsql" in the *Oracle HTTP Server Administrator's Guide.*

4. When mod_plsql detects dead connections in its connection pool.

   See Section 3.11.3, "Detecting Dead Database Connections in a Connection Pool" for more information.

3.11.3 Detecting Dead Database Connections in a Connection Pool

mod_plsql maintains a pool of connections to the database, and reuses established database connections for subsequent requests. If there is no response from a database connection in a connection pool, mod_plsql detects this, discards the dead connection, and creates a fresh database connection for subsequent requests.

The dead database connection detection feature of mod_plsql eliminates the occurrence of random errors when a database node or instance goes down. This feature is also extremely useful in high availability configurations like Real Application Cluster (RAC). If a node in an RAC cluster has gone down, mod_plsql detects this and immediately starts servicing requests using the other RAC nodes.

By default, when a RAC node or database instance goes down and mod_plsql had previously pooled connections to the node, the first mod_plsql request which uses a dead connection in its pool will result in a failure response of HTTP-503 being sent back to the end-user. This failure is then used by mod_plsql to trigger the detection and removal of all dead connections in its pool. mod_plsql pings all connection pools that were created before the node failure, and this ping operation is performed at the time of processing the next request that uses a pooled connection. If the ping operation fails, the database connection is discarded, and a new connection is created and processed.

Note: If after node failure, multiple mod_plsql requests come in concurrently, and mod_plsql has not yet detected the first dead connection, there could be multiple failures at that instant.

See Also: Section 3.11.2, "Closing Pooled Database Sessions" for information on other scenarios where mod_plsql closes dead database connections even when no mod_plsql request is made to the server.

mod_plsql provides two configuration options for tuning the dead database connection detection feature:

- Specifying the Option to Detect Dead Database Connections
- Specifying the Timeout Period

3.11.3.1 Specifying the Option to Detect Dead Database Connections

mod_plsql corrects connections after it detects a failure that could be caused by a database node going down. This is controlled by the `PlsqlConnectionValidation` parameter. For details on the PlsqlConnectionValidation parameter, refer to the "mod_plsql" section in the *Oracle HTTP Server Administrator's Guide*.

See Also: Table 4–1, " Database Access Descriptor (DAD) Parameters Recommended Setting Summary"

3.11.3.2 Specifying the Timeout Period

When the `PlsqlConnectionValidation` parameter is set to `Automatic` or `AlwaysValidate`, mod_plsql attempts to test pooled database connections.

You can specify the timeout period for mod_plsql to test a bad database connection in a connection pool. This is controlled by the `PlsqlConnectionTimeout` parameter, which specifies the maximum time mod_plsql should wait for the test request to complete before it assumes that a connection is not usable.

For details on the PlsqlConnectionTimeout parameter, refer to the "mod_plsql" section in the *Oracle HTTP Server Administrator's Guide*.

> **See Also:** Table 4–1, " Database Access Descriptor (DAD) Parameters Recommended Setting Summary"

3.12 Restrictions in mod_plsql

The following restrictions exist in mod_plsql:

- The maximum length of the HTTP cookie header is 32000 bytes. Values higher than this generate an error.
- The maximum length of any single cookie within the HTTP cookie is 3990. Values higher than this generate an error. This limit is due to the OCI array bind limit of strings in arrays.
- There is a hard maximum cookie limit in mod_plsql that limits the number of cookies being set at any given time. That limit is set to 20. Anything over 20 will be dropped.
- The PL/SQL Gateway does not support calling procedures with OUT parameters to be called from a Web interface. Doing this may result in ORA-6502 errors. The recommended approach is not to call any procedure that has OUT variables in it. However, the current architecture will let you modify a value as long as the modified value does not exceed the length that was passed in. Existing applications that encounter this problem need to be modified in one of the following ways:
  - Implement wrappers for procedures with OUT parameters so that such procedures are not invoked directly through a browser URL.
  - Create a local variable that gets assigned the value of the parameter being passed in, and is then used for all internal changes.
- The total number of name value pairs that can be passed to a PL/SQL procedure is 2000.
- mod_plsql limits the size of a single parameter that can be passed to a procedure to 32512 bytes.
- It is not possible to use identical DAD locations in different virtual hosts.
- The maximum value allowed for the `PlsqlCacheMaxSize` and `PlsqlCacheTotalSize` parameters is 4294967296 bytes (4 Gigabytes). If you specify a value higher than this, mod_plsql issues a warning and sets the value to 4 GB internally.

4

Optimizing PL/SQL Performance

This chapter discusses the techniques for improving PL/SQL performance in Oracle HTTP Server.

This chapter contains the following sections:

4.1 PL/SQL Performance in Oracle HTTP Server - Overview

This section describes several techniques to improve the performance of PL/SQL based Web applications in Oracle HTTP Server.

Table 4–1 lists recommendations for Database Access Descriptor (DAD) parameters and settings. By default, these DAD parameters are specified in the file `dads.conf`. On UNIX systems, this is in the *ORACLE_HOME*`/Apache/modplsql/conf` directory. On Windows systems, by default, this file is in the *ORACLE_HOME*`\Apache\modplsql\conf` directory. The file `dads.README` in this directory describes the DAD parameters in detail.

Table 4–1 Database Access Descriptor (DAD) Parameters Recommended Setting Summary

Parameter	Recommended Setting
`PlsqlAlwaysDescribeProcedure`	Set this to `off` for best performance. Default Value: `off`
`PlsqlDatabaseConnectString`	For newer DADs, use the `ServiceNameFormat`. Use the `SIDFormat` only for backward compatibility. **Note:** For HA configurations of the database, it is recommended that the connect string parameter gets resolved through an LDAP lookup.
`PlsqlFetchBufferSize`	For multibyte character sets like Japanese or Chinese, setting this to `256` should provide better performance Default Value: `200`

Table 4–1 (Cont.) Database Access Descriptor (DAD) Parameters Recommended Setting Summary

Parameter	Recommended Setting
`PlsqlIdleSessionCleanupInterval`	Increasing this parameter allows pooled database connections to remain available, in the pool, for the specified time Default Value: `15` (minutes) **See Also:** Section 3.11.3, "Detecting Dead Database Connections in a Connection Pool"
`PlsqlLogEnable`	This parameter should be set to `Off` unless recommended by Oracle support for debugging purposes Default Value: `off`
`PlsqlConnectionValidation`	This parameter specifies the option mod_plsql should use to detect dead connections in its connection pool. The different options are: ▪ `Automatic` - mod_plsql tests all pooled database connections created before the detection of a failure that could indicate an instance failure. ▪ `ThrowAwayOnFailure` - mod_plsql discards all pooled database connections created before the detection of a failure that could indicate an instance failure. ▪ `AlwaysValidate` - mod_plsql always tests all pooled database connections before issuing a request. **Important:** Since this option has an associated performance overhead for each request, this should be used with caution. ▪ `NeverValidate` - mod_plsql will never ping any pooled database connection. Default Value: `Automatic` **See Also:** Section 3.11.3.1, "Specifying the Option to Detect Dead Database Connections".
`PlsqlConnectionTimeout`	This parameter specifies the timeout (in milliseconds) for testing a connection pooled in mod_plsql. Default Value: `10000` **See Also:** Section 3.11.3.2, "Specifying the Timeout Period".
`PlsqlMaxRequestsPerSession`	If the PL/SQL based Web application does not leak resources or memory, this parameter can be set to a higher value (for example, 5000). Default Value: `1000` **See Also:** Section 3.11.2, "Closing Pooled Database Sessions" and Section 4.3.2, "Connection Pooling Tips and Oracle HTTP Server Configuration".
`PlsqlNLSLanguage`	Set this parameter to match the database Globalization Support parameters to eliminate overheads in character set conversions in Oracle Net Services
`PlsqlSessionStateManagement`	Set this parameter to `StatelessWithFastResetPackageState` if the database is 8.1.7.2 or later.

Table 4–2 lists mod_plsql caching options and the sections that describe these caching options.

Table 4–2 Caching Options

Option	Description
Expires Technique	Best performance - for content that changes predictably. See Also: Section 3.10.2, "Using the Expires Technique"
Validation technique	Good performance - for content that changes unpredictably. See Also: Section 3.10.1, "Using the Validation Technique"
System-level caching	Improves performance by caching one copy for everyone on system. See Also: Section 3.10.3, "System- and User-level Caching with PL/SQL Based Web Applications"

See Also:

- The appendix on performance metrics in the *Oracle Application Server Performance Guide* for information on mod_plsql metrics
- The section "mod_plsql" in the *Oracle HTTP Server Administrator's Guide* for details on the DAD Parameters shown in Table 4–1.
- *Oracle Application Server PL/SQL Web Toolkit Reference*

4.2 Process-Based and Thread-Based Operation in Oracle HTTP Server

This section describes PL/SQL performance issues that apply on platforms where the Oracle HTTP Server is process-based and thread-based. On a process-based Oracle HTTP Server, such as those running on UNIX-based platforms, each process servers all types of HTTP requests, including servlets and PL/SQL, static files. On a thread-based Oracle HTTP Server, such as Windows-based platforms, there is just one Oracle HTTP Server process with multiple threads within the process; individual threads can be used serve all types of HTTP requests.

Note: In some cases in this chapter we make references to performance optimizations that apply for PL/SQL based Web applications where the distinction between platforms, either process-based or thread based is significant.

4.3 Performance Tuning Issues in mod_plsql

While using mod_plsql, the areas that affect performance and scalability are:

- PL/SQL Based Web Application Development Considerations and Programming Tips
- Connection Pooling Tips and Oracle HTTP Server Configuration
- Tuning the Number of Database Sessions
- Two-Listener Strategy
- Overhead Problems
- The Flexible Parameter Passing (four-parameter) Overhead

See Also: Section 3.11, "Performance Tuning Areas for mod_plsql"

4.3.1 PL/SQL Based Web Application Development Considerations and Programming Tips

PL/SQL Gateway users should consider the following topics when developing PL/SQL based Web applications:

1. **Manage the use of Database Access Descriptors (DADs)**

   Try to restrict the number of DADs that each Oracle HTTP Server node uses.

   > **Note:** Performance is not affected if there are DADs that are not being used.

2. **Use of Nested Tables**

   PL/SQL provides the ability to create tables. To build PL/SQL tables, you build a table that gives the datatype of the table, as well as the index of the table. The index of the table is the binary integer ranging from -2147483647 to +2147483647. This table index option is known as *sparsity*, and allows meaningful index numbers such as customer numbers, employee number, or other useful index keys. Use PL/SQL tables to process large amounts of data.

   PL/SQL provides `TABLE` and `VARRAY` (variable size array) collection types. The `TABLE` collection type is called a nested table. Nested tables are unlimited in size and can be sparse, which means that elements within the nested table can be deleted using the `DELETE` procedure. Variable size arrays have a maximum size and maintain their order and subscript when stored in the database. Nested table data is stored in a system table that is associated with the nested table. Variable size arrays are suited for batch operations in which the application processes the data in batch array style. Nested tables make for efficient queries by storing the nested table in a storage table, where each element maps to a row in the storage table.

3. **Use procedure naming overloading with caution**

   PL/SQL based Web applications should use the procedure name overloading feature with caution. It is best if procedure name overloading is avoided by having multiple procedures with different names.

4. **Consider rewriting applications where there is significant overhead in determining the type parameters**

   PL/SQL based Web applications should be aware of the overhead in trying to execute procedures where the URL does not provide enough details to know about the type of the parameter, such as scalar or array. In such cases, the first attempt to execute a procedure fails and the procedure signature needs to be described before it can be executed by mod_plsql.

   See Also: Section 4.3.5, "Overhead Problems"

5. **Use procedures with 2-parameter style flexible parameter passing**

   Procedures should make use of the more performant 2-parameter style flexible parameter passing rather than the 4-parameter style parameter passing.

See Also:

- *Oracle Application Server PL/SQL Web Toolkit Reference*
- Section 4.3.6, "The Flexible Parameter Passing (four-parameter) Overhead"

4.3.2 Connection Pooling Tips and Oracle HTTP Server Configuration

Consider the following topics when configuring connection pooling with Oracle HTTP Server:

1. Using the default connections pooling and setting values for `PlsqlMaxRequestsPerSession`

   Creating new database connections is an expensive operation and it is best if every request does not have to open and close it own database connections. The optimal technique is to make sure that database connections opened in one request are reused in subsequent requests. In some rare situations, where a database is accessed very infrequently and performance is not a major concern, connection pooling can be disabled. For example, if the administrator accesses a site infrequently to perform some administration tasks, then the DAD used to access the administrator applications can choose to disable connection pooling. To disable connection pooling, set the DAD parameter `PlsqlMaxRequestsPerSession` to the value 1.

   Note: Setting `PlsqlMaxRequestsPerSession` to the value 1 reduces the number of available database sessions and may impact performance.

2. On UNIX systems, Oracle HTTP Server configuration should be properly tuned so that once processes are started up, the processes remain up for a while. Otherwise, the connection pooling in mod_plsql is rendered useless. The Oracle HTTP Server listener should not have to continually start up and shut down processes. A proper load analysis should be performed of the site to determine what the average load on the Web site. The Oracle HTTP Server configuration should be tuned such that the number of `httpd` processes can handle the average load on the system. In addition, the configuration parameter `MaxClients` in the `httpd.conf` file should be able to handle random load spikes as well.

3. On UNIX systems, Oracle HTTP Server processes should be configured so that processes are eventually killed and restarted. This is required to manage any possible memory leaks in various components accessed through the Oracle HTTP Server. This is specifically required in mod_plsql to ensure that any database session resource leaks do not cause a problem. Make sure that `MaxRequestsPerChild` configuration parameter is set to a high number. For PL/SQL based Web applications, this should not be set to `0`.

4. For heavily loaded sites, the Oracle HTTP Server configuration parameter `KeepAlive` should be disabled. This ensures that each process is available to service requests from other clients as soon as a process is done with servicing the current request. For sites which are not heavily loaded, and where it is guaranteed that the number of Oracle HTTP Server processes are always greater than the number of simultaneous requests to the Oracle HTTP Server listener, enabling the `KeepAlive` parameter results in performance improvements. In such cases, make sure to tune the `KeepAliveTimeout` parameter appropriately.

5. You may want to lower the value of `Timeout` in the Oracle HTTP Server configuration. This ensures that Oracle HTTP Server processes are freed up earlier if a client is not responding in a timely manner. Do not set this value too low, otherwise slower responding clients could time out.
6. Most Web sites have many static image files, which are displayed in each screen for a consistent user interface. Such files rarely change and you can reduce a considerable load on the system by tagging each image served by the Oracle HTTP Server listener with `mod_expires`. You should also consider front-ending your Web site with Oracle Application Server Web Cache.
   - How do I know if the Web site can benefit from the use of `mod_expires`?
     - Use Netscape, or any browser that enables you to view page caching information, and visit several heavily accessed Web pages on the site. On each page, right click the mouse and select `View Info` from the pop up menu (or the equivalent command for your browser). If the top panel in the page information window lists many different images and static content, then the site could benefit from the use of `mod_expires`.
     - You can also check the Oracle HTTP Server access logs to see what percentage of requests result in HTTP 304 (Not Modified) status. Use the `grep` utility to search for `304` in the `access_log` and divide this resulting number of lines by the total number of lines in the `access_log`. If this percentage is high, then the site could benefit from the use of `mod_expires`.
   - How do I tag static files with the Expires header?
     - Locate the `Location` directive used to serve your static image files. Add the `ExpiresActive` and `ExpiresDefault` directives to it.

       ```
       Alias /images/ "/u01/app/oracle/myimages/"
       <Directory "/u01/app/oracle/myimages/">
            AllowOverride None
            Order allow, deny
            Allow from all
            ExpiresActive On
            ExpiresDefault A2592000
       </Directory>
       ```

     - The browser caches all static files served off the `/images` path for 30 days from now. Refer to the *Oracle HTTP Server Administrator's Guide* for more details.

       If you are using Oracle Application Server Web Cache, these files can be cached in memory with the use of the `Surrogate-Control` header. For example:

       ```
       Alias /images/ "/u01/app/oracle/myimages/"
       <Directory "/u01/app/oracle/myimages/">
            AllowOverride None
            Order allow, deny
            Allow from all
            ExpiresActive On
            ExpiresDefault A2592000
            <Files *>
               Header set Surrogate-Control 'max-age=2592000'
            </Files>
       </Directory>
       ```

Refer to the *Oracle Application Server Web Cache Administrator's Guide* for more details on the `Surrogate-Control` header.

- How do I know if the static files are being tagged with the `Expires` header?
  - Using Netscape, or the browser of your choice, clean up all the cached files in the browser.
  - Visit a Web page that should have images tagged with the `Expires` header. Right click the mouse on the page and select **View Info**, from the pop up menu. or use the equivalent command for your browser.
  - In the top panel of the page information, select an image that should be tagged with the `Expires` header.
  - Review the information displayed in the bottom panel. The `Expires` header should be set to a valid date. If this entry is `No date given`, then the file is not being tagged with the `Expires` header.

4.3.3 Tuning the Number of Database Sessions

Consider the following topics when tuning the number of database sessions:

1. The `processes` and `sessions` parameters in the Oracle `init$SID.ora` configuration file should be set so that Oracle is able to handle the maximum number of database sessions. This number should be proportional to the number of DADs times the maximum number of Oracle HTTP Server processes, times the number of Oracle HTTP Server instances.
2. Using a two-listener strategy or using a shared server reduces the number of database sessions. See Section 4.3.4, "Two-Listener Strategy".
3. On UNIX platforms, the connection pool is not shared across Oracle HTTP Server processes. For this reason, it is recommended that the application use as few DADs as possible.
4. Front ending your Oracle HTTP Server with Oracle Application Server Web Cache reduces the requirement to have a high number of processes for your HTTP configuration, resulting in lesser number of database sessions.

Note: This is only beneficial when Oracle HTTP Server is front-ended with OracleAS Web Cache and OracleAS Web Cache caches static content. To test that OracleAS Web Cache is caching static content, see item 6 in Section 4.3.2, "Connection Pooling Tips and Oracle HTTP Server Configuration".

4.3.4 Two-Listener Strategy

On platforms where the Oracle HTTP Server is process-based, such as all UNIX-based platforms, each process serves all types of HTTP requests, including servlets, PLSQL, static files, and CGI. In a single Oracle HTTP Server listener setup, each `httpd` process maintains its own connection pool to the database. The maximum number of database sessions is governed by the setting in `httpd.conf` configuration file for `StartServers`, `MinSpareServers`, and `MaxSpareServers`, plus the load on the system. This architecture does not allow for tuning the number of database sessions based on the number of mod_plsql requests. To tune the number of database sessions based on the number of mod_plsql requests, install a separate HTTP listener for mod_plsql requests only. This approach greatly reduces the number of database sessions that are needed to serve mod_plsql requests.

For example, assume a main Oracle HTTP Server listener is running on port 7777 of `mylsnr1.mycompany.com`. First, you can install another Oracle HTTP Server listener on port 8888 on `mylsnr2.mycompany.com`. Next, redirect all mod_plsql requests made to `mylsnr1.mycompany.com:7777` to the second listener on `mylsnr2.mycompany.com:8888`. Review the following steps:

1. To redirect all PL/SQL requests for `mylsnr1.mycompany.com:7777` to `mylsnr2.mycompany.com:8888`, make the following configuration changes:

   a. For the Oracle HTTP Server listener running on Port 7777, edit *ORACLE_HOME*`/Apache/modplsql/conf/plsql.conf` file. Comment out the following line by putting a # in front of the line:

   ```
   #LoadModule plsql_module...
   ```

   b. Copy the DAD location used to service PL/SQL requests in `mylsnr1.mycompany.com` to the configuration file *ORACLE_HOME*`/Apache/modplsql/conf/dads.conf` in `mylsnr2.mycompany.com`.

   Comment out the DAD location configuration parameters on `mylsnr1.mycompany.com` by prefixing the line with a "#" character.

   ```
   #<Location /pls/portal>
   #...
   #</Location>
   ```

   c. Configure this listener to forward all mod_plsql requests for this DAD location to the second listener by adding the following line in `dads.conf`:

   ```
   ProxyPass /pls/portal http://mylsnr2.mycompany.com:8888/pls/portal
   ```

   Repeat the configuration procedures for all DAD Locations.

2. Because the PL/SQL procedures generate URLs that are displayed in the browser, it is important that all URLs are constructed without any references to the internal mod_plsql listener on `mylsnr2.mycompany.com:8888`. Depending on how the URLs are being generated in the PL/SQL based Web application, there are three options:

   - If the URLs are hard-coded into the application, make sure that they are always generated using the hard-coded values as HOST=mylsnr1.mycompany.com and PORT=7777. No change would be required for this scenario.
   - If the PL/SQL based Web applications always use the CGI environment variables `SERVER_NAME` and `SERVER_PORT`, then it is easy to change the configuration of the listener on `mylsnr2.mycompany.com`. Edit the file and change the lines `ServerName` and `Port` in the *ORACLE_HOME*`/Apache/conf/httpd.conf` file for the second listener as follows:

     ```
     ServerName mylsnr1.mycompany.com (was mylsnr2.mycompany.com)
     Port 7777                        (was 8888)
     ```

   - If the URLs are being generated using the CGI environment variable `HTTP_HOST`, you need to override the CGI environment variables for the Oracle HTTP Server listener running on Port 8888. Add the following lines to the *ORACLE_HOME*`/Apache/modplsql/conf/dads.conf` file for each DAD to override the default CGI environment variables `HOST`, `SERVER_NAME`, and `SERVER_PORT`:

     ```
     PlsqlCGIEnvironmentList  SERVER_NAME  mylsnr1.mycompany.com
     ```

```
PlsqlCGIEnvironmentList  SERVER_PORT  7777
PlsqlCGIEnvironmentList  HOST mylsnr1.us.oracle.com:7777
```

In all cases, the intent is to fool the application to generate URLs as if there never was a second listener.

3. Test the setup and make sure that you can access all the DADs without any problems.
4. In this setup, the main listener `mylsnr1.mycompany.com` can be configured based on the total load on the Oracle HTTP Server listener. The second listener on `mylsnr2.mycompany.com` can be fine-tuned based on just the mod_plsql requests being made.

4.3.5 Overhead Problems

While executing some of the stored procedures, mod_plsql may incur a `Describe` overhead, which would result in two extra round trips to the database for a successful execution. This has performance implications.

4.3.5.1 The Describe Overhead

In order to execute PL/SQL procedures, mod_plsql needs to know about the datatype of the parameters being passed in. Based on this information, mod_plsql binds each parameter either as an array or as a scalar. One way to know the procedure signature is to describe the procedure before executing it. However, this approach is not efficient because every procedure has to be described before execution. To avoid the describe overhead, mod_plsql looks at the number of parameters passed for each parameter name. It uses this information to assume the datatype of each variable. The logic is simply that if there is a single value being passed, then the parameter is a scalar, otherwise it is an array. This works for most cases but fails if there is an attempt to pass a single value for an array parameter or pass multiple values for a scalar. In such cases, the first attempt to execute the PL/SQL procedure fails. mod_plsql issues a `Describe` call to get the signature of the PL/SQL procedure and binds each parameter based on the information retrieved from the `Describe` operation. The procedure is re-executed and results are sent back.

This `Describe` call occurs transparently to the procedure, but internally mod_plsql has encountered two extra round trips, one for the failed execute call and the other for the describe call.

4.3.5.2 Avoiding the Describe Overhead

You can avoid performance problems with the following:

- Use flexible parameter passing.
- Always ensure that you pass multiple values for arrays. For single values, you can pass dummy values that are ignored by the procedure.
- Use the following workaround, which defines a two-parameter style procedure which defaults the unused variables.

  1. Define a scalar equivalent of your procedure, which internally calls the original procedure. For example, the original package could be similar to the following example:

```
CREATE OR REPLACE PACKAGE testpkg AS
 TYPE myArrayType is TABLE of VARCHAR2(32767) INDEX BY binary_ integer;
 PROCEDURE arrayproc (arr myArrayType);
END testpkg;
```

```
/
```

2. If you are making URL calls like `/pls/.../testpkg.arrayproc? arr= 1`, change the specification to be similar to the following:

```
CREATE OR REPLACE PACKAGE testpkg AS
 TYPE myArrayType is TABLE of VARCHAR2( 32767) INDEX BY binary_integer;
  PROCEDURE arrayproc (arr varchar2);
  PROCEDURE arrayproc (arr myArrayType);
END testpkg;
/
```

3. The procedure `arrayproc` should be similar to:

```
CREATE OR REPLACE PACKAGE BODY testpkg AS
PROCEDURE arrayproc (arr varchar2) IS
  localArr myArrayType;
BEGIN
  localArr( 1) := arr;
  arrayproc (localArr);
END arrayproc;
```

4.3.6 The Flexible Parameter Passing (four-parameter) Overhead

Round-trip overhead exists if a PL/SQL procedure is using the older style four-parameter interface. The PL/SQL Gateway first tries to execute the procedure by using the two-parameter interface. If this fails, the PL/SQL Gateway tries the four-parameter interface. This implies that all four-parameter interface procedures experience one extra round-trip for execution.

- Avoiding the flexible parameter passing overhead

  To avoid this overhead, it is recommended that you write corresponding wrappers that use the two-parameter interface and internally call the four-parameter interface procedures. Another option is to change the specification of the original procedure to default to the parameters that are not passed in the two-parameter interface. The four-parameter interface has been provided only for backward compatibility and will be deprecated in the future.

- Using flexible parameters and the exclamation mark

  The flexible parameter passing mode in Oracle HTTP Server expects the PL/SQL procedure to have the exclamation mark before the procedure name. Due to performance implications of the auto-detect method used in Oracle HTTP Server, the exclamation mark is now required for flexible parameter passing in Oracle HTTP Server. In Oracle HTTP Server, each procedure is described completely before being executed. The procedure `Describe` call determines the signature of the procedure and requires around-trip to the database. The PL/SQL Gateway in Oracle HTTP Server avoids this round trip by having end-users explicitly indicate the flexible parameter passing convention by adding the exclamation mark before the procedure.

4.4 Tuning File System Cache to Improve Caching Performance

You can configure and use a File System Cache to improve the performance of OracleAS Portal applications and generic PL/SQL based Web applications.

This section covers the following topics:

4.4.1 Introducing File System Cache Tuning

This section covers mod_plsql related File System Cache tuning options. Cache contents are cached using Operating System supplied file system calls; the cached contents are not stored in the mod_plsql memory space. Using the mod_plsql File System Cache, the contents of the cache may be in memory when the Operating System supports, and the system is configured to use features such as memory disk (some UNIX platforms support memory disk based fast storage).

The information in this section can improve the performance of PL/SQL based Web applications when mod_plsql is configured to use the File System Cache. For example, OracleAS Portal uses the File System Cache, and therefore, OracleAS Portal performance should improve when the File System Cache is properly tuned.

Table 4–3 lists the cache related parameters that you can set for mod_plsql. Set these parameters in the `cache.conf` file that is available on UNIX in the directory, *ORACLE_HOME*`/Apache/modplsql/conf`, and on Windows, this is found in the directory, *ORACLE_HOME*`\Apache\modplsql\conf`.

> **Note:** The file `cache.README` in the `conf` directory includes a full description of each parameter, and provides examples showing how to set parameter values.

Table 4–3 mod_plsql cache.conf Configuration Parameter Summary

Parameter	Description
`PlsqlCacheCleanupTime`	Sets the interval for running cache cleanup routines. Default: `Everyday 23:00` (run cleanup routine daily at 11PM local time) **See Also:** Section 4.4.5, "Configuring Cache Cleanup"
`PlsqlCacheDirectory`	Defines the directory that holds the mod_plsql cache. Default: On UNIX systems, the default directory for the error log is: *ORACLE_HOME*`/Apache/modplsql/cache` On Windows systems, the default directory is: *ORACLE_HOME*`\Apache\modplsql\cache` **See Also:** Section 4.4.3, "Configuring File System Cache to Reside on a Faster File System"
`PlsqlCacheEnable`	Enables the file system cache. Default: `On` **See Also:** Section 4.4.2, "Enabling File System Cache"

Table 4–3 (Cont.) mod_plsql cache.conf Configuration Parameter Summary

Parameter	Description
`PlsqlCacheMaxAge`	Controls the aging, in days for the cache contents. Default: `30` (days) **See Also:** Section 4.4.4.2, "Setting the Days of Aging for Cache with PlsqlCacheMaxAge"
`PlsqlCacheMaxSize`	Sets the maximum size, in bytes, for an individual file stored in the cache. Default: `1048576` (1 Megabyte) **See Also:** Section 4.4.4.3, "Setting the Maximum File Size for a Cache File with PlsqlCacheMaxSize"
`PlsqlCacheTotalSize`	Limits the total size of the cache. The value is specified in bytes. Default: `20971520` (20 Megabytes) **See Also:** Section 4.4.4.1, "Setting the Total Cache Size with PlsqlCacheTotalSize"

4.4.2 Enabling File System Cache

The `cache.conf` parameter `PlsqlCacheEnable` enables mod_plsql caching. For maximum performance, enable `PlsqlCacheEnable` by setting the value of this parameter to `On`.

Note: Only applications that support PLSQL caching, such as Oracle Portal, will benefit by setting `PlsqlCacheEnable` to `On`.

4.4.3 Configuring File System Cache to Reside on a Faster File System

This section describes how to configure a File System Cache to reside on a separate disk. When you use File System Cache and store the cache on a faster separate disk, performance should improve for all types of Web applications using File System Cache, including OracleAS Portal and generic PL/SQL based Web applications.

When you configure File System Cache, the cache can reside either on a separate physical disk or in a memory disk.

To set up a File System Cache on a separate disk:

1. Assume that the file system for the cache resides at the location:

   On UNIX: `/u01/cache`

   On Windows: `E:\cache`

2. Update the file:

   On UNIX: *`ORACLE_HOME`*`/Apache/modplsql/conf/cache.conf`

   On Windows: *`ORACLE_HOME`*`\Apache\modplsql\conf\cache.conf`

3. Change the cache parameter `PlsqlCacheDirectory`:

   On UNIX: `PlsqlCacheDirectory /u01/cache`

   On Windows: `PlsqlCacheDirectory E:\cache`

4. Restart Oracle HTTP Server for the configuration changes to take effect.

4.4.4 Resizing File System Cache

This section covers the following topics:

- Setting the Total Cache Size with PlsqlCacheTotalSize
- Setting the Days of Aging for Cache with PlsqlCacheMaxAge
- Setting the Maximum File Size for a Cache File with PlsqlCacheMaxSize

4.4.4.1 Setting the Total Cache Size with PlsqlCacheTotalSize

The default installation sets the mod_plsql file system cache size to 2097152 bytes (20 Megabytes). If your PL/SQL application does not make use of the OWA_CACHE packages, or uses them to cache small amounts of content, then the default setting should be sufficient. If your PL/SQL application caches a lot of content in the mod_plsql file system cache, you should consider specifying a higher value.

To control the cache size, set the `PlsqlCacheTotalSize` parameter in the file `cache.conf`. On UNIX systems, this file is located under *`ORACLE_HOME`*`/Apache/modplsql/conf` directory. On Windows systems, this file is located under *`ORACLE_HOME`*`\Apache\modplsql\conf`.

You need to set the cache size high enough to achieve a high cache hit ratio. Try to set the cache size large enough so that frequently accessed content stays cached. It is also important to limit the amount of disk space, so that the cache size does not grow too large. Correct tuning for the cache size provides enough cache to hold all frequently accessed content while preventing the cache size from growing too large, since a very large cache is inefficient to search.

The value for `PlsqlCacheTotalSize` is specified as a number of bytes. 1MB equals 1048576 bytes. This setting is a soft limit on the amount of cache allocated. In some cases, the cache size may grow beyond this limit until the next cleanup operation. Therefore, the hard limit on the cache size is the underlying physical hard disk size. When this limit is reached, no cache content can be written out to disk until space is available.

> **Note:** The maximum value allowed for `PlsqlCacheTotalSize` is 4294967296 bytes (4 Gigabytes).

To determine a reasonable cache size, do the following:

1. Turn on mod_plsql performance logging by setting the `LogLevel` in `httpd.conf` to the `info` level to enable mod_plsql logging.
2. Monitor the `error_log` on a daily basis. On UNIX systems, the default directory for the error log is: *`ORACLE_HOME`*`/Apache/Apache/log`. On Windows systems, the default directory is: *`ORACLE_HOME`*`\Apache\Apache\log`.

The mod_plsql `error_log` entries have the form:

```
[info] mod_plsql: cachecleanup deleted=2571 max_age=96,2178852b
kept=1042,25585368b time=128s limit=25600000b
```

where:

`deleted` is the number of cache files that got deleted during the cleanup process.

`max_age` is the number of cache files and total size that got deleted because they haven't been used for some time.

`kept` is the number of cache files and total size that was kept after the cleanup process.

`time` is the amount of time to perform the cleanup.

`limit` is the total cache size. This is the value of the `PlsqlCacheTotalSize` setting.

Interpret the entries in the error log as follows:

- If a high number of files are being deleted when compared to the number of files that were kept, this is a clear indication that your cache size is too small. You probably need to increase the size of the cache.
- If a low number of files being deleted when compared to the number of files that were kept is observed, this is an indication that your cache size is probably too big. If you have enough disk space, you can chose to leave it as it or you can decrease the size of the cache to reclaim some disk space.

4.4.4.2 Setting the Days of Aging for Cache with PlsqlCacheMaxAge

Using the `PlsqlCacheMaxAge` parameter, you can control the "staleness" of cache content. The value for parameter is specified in units of days. The default value for this parameter is 30 (days). This means cache content is kept in the cache if it is less than 30 days old. After 30 days, the content is considered for deletion during the cleanup process.

The `max_age` information in mod_plsql `error_log` shows cache file aging information. If your site is a highly dynamic site, it would make sense to configure this setting to a lower value, since the older cache content will usually not be used again and, therefore, the lower value does not affect the cache hit ratio. If the site contains many static pages, it would make sense to increase the value of `PlsqlCacheMaxAge` so that the cleanup process does not deliberately delete the cache content.

4.4.4.3 Setting the Maximum File Size for a Cache File with PlsqlCacheMaxSize

Using the `PlsqlCacheMaxSize` parameter, you can specify the maximum size for individual files in the cache. Using this parameter prevents the case in which one cache file fills up the entire cache.

The default value for this parameter is 1048576 (bytes). In general, set this parameter to a value that represents about 1-3% of the total cache size.

Note: The maximum value allowed for `PlsqlCacheMaxSize` is 4294967296 bytes (4 Gigabytes).

4.4.5 Configuring Cache Cleanup

The cache cleanup parameter determines the frequency in which the File System Cache is examined and, if necessary, cleaned up. The cache cleanup parameter, `PlsqlCacheCleanupTime` is specified in the `cache.conf` file. The frequency can be set to daily, weekly, or monthly. When specifying weekly cleanup, it is possible to specify the day of the week and the time of the day.

The default mod_plsql setting of `PlsqlCacheCleanupTime` is daily at 11PM local time. Therefore, by default, every night at 11PM, the cleanup routine runs. When you select the monthly frequency, the cleanup occurs on the first Saturday of each month.

Configuring this parameter correctly is important since cleaning up too often can result in a lower cache hit ratio and when cleaning does not occur often enough, the cache's disk usage may be excessive.

Monitor the cleanup activities using the entries in the mod_plsql `error_log`; then tune the cleanup parameter, `PlsqlCacheCleanupTime` by analyzing the entries.

```
[info] mod_plsql: cachecleanup deleted=2571 max_age=96,2178852b
kept=1042,25585368b time=128s limit=25600000b
```

Note the following:

- Seeing a large number for the cleanup time can be an indication that the cleanup frequency is set too low. When the log indicates that the cleanup operation is busy examining or deleting many cache files, increasing the cleanup frequency should decrease the time spent in the cleanup operation.
- If a high number files are being deleted during the cleanup operation because of "staleness", this is an indication that the cleanup frequency is too low. In this case, increase the frequency so that the cleanup can actively delete "stale" cache content more frequently.

4.5 Oracle HTTP Server Directives

To improve PL/SQL performance in Oracle HTTP Server, you need to tune the Oracle HTTP Server directives appropriately for your configuration.

See Also:

- The section "Configuring Oracle HTTP Server Directives" in the *Oracle Application Server Performance Guide*.
- *Oracle HTTP Server Administrator's Guide*.Chapter 3, "Managing Server Processes" and Chapter 4, "Managing the Network Connection".

A

Frequently Asked Questions

What is mod_plsql?

mod_plsql is an Oracle HTTP Server plug-in that communicates with the database by mapping browser requests into database stored procedure calls over a SQL*Net connection. It is generally indicated by a `/pls` virtual path. The mod_plsql gateway provides support for building and deploying PL/SQL-based applications on the Web. PL/SQL stored procedures can retrieve data from database tables and generate HTTP responses containing formatted data and HTML code to display in a Web browser. See the *Oracle Database Application Developer's Guide - Fundamentals* for more information.

What is the PL/SQL Web Toolkit?

The PL/SQL Web Toolkit enables you to develop Web applications as PL/SQL procedures stored in an Oracle database server. Packages in the toolkit define procedures, functions, and data types that you can use in your stored procedures. See the *Oracle Database Application Developer's Guide - Fundamentals* for more information.

How do I find the version of mod_plsql?

You can determine the version of mod_plsql by executing the `oversioncheck` script on the mod_plsql binary.

On UNIX platforms, issue the following command:

```
ORACLE_HOME/Apache/Apache/bin/oversioncheck ORACLE_HOME/Apache/modplsql/bin/modplsql.so
```

On Windows platforms, issue the following command:

```
ORACLE_HOME\Apache\Apache\bin\oversioncheck ORACLE_HOME\bin\modplsql.dll
```

How do I find the version of the OWA packages?

1. Use SQL*Plus and connect as any user to the database.
2. Execute the following command:

   ```
   select owa_util.get_version from dual;
   ```

   This should show the version of the OWA packages. For example, 10.1.2.0.4.

If this query fails, you are having a very old version of OWA packages that does not have versioning. It is recommended that you upgrade to a newer version.

How do I install the OWA packages?

See Section 1.2, "Installing Required Packages" for more information.

How do I uninstall the OWA packages?

OWA packages can be uninstalled by performing the following tasks:

1. Navigate to the directory from where the OWA packages were installed. For example:

   ```
   cd ORACLE_HOME/Apache/modplsql
   ```

2. Use SQL*Plus to connect as the owner the OWA packages. This user should be the SYS user, unless you have an old version of the OWA packages.
3. Invoke the script `owadins.sql` to uninstall the OWA packages.

How do I detect and clean up duplicate OWA packages installed in the database?

The following SQL query can be used to determine the location of the OWA packages:

```
SELECT OWNER, OBJECT_TYPE
FROM   DBA_OBJECTS
WHERE  OBJECT_NAME = 'OWA'
```

You will see the following results:

```
SQL>

1  SELECT OWNER, OBJECT_TYPE
2  FROM DBA_OBJECTS
3* WHERE OBJECT_NAME = 'OWA'

OWNER  OBJECT_TYPE
-----  -----------
SYS    PACKAGE
SYS    PACKAGE BODY
PUBLIC SYNONYM
```

If you see more lines than shown in the preceding SQL query, it means that older OWA packages exist in other schemas, which may cause issues for mod_plsql users. In such situations, uninstall all versions of the OWA packages from the database, and reinstall the OWA packages that ship with the product.

I am getting HTTP error codes while accessing PL/SQL procedures through mod_plsql.

mod_plsql logs detailed error messages to the Oracle HTTP Server file *ORACLE_HOME*`/Apache/Apache/logs/error_log`. Scan this file to understand the problem. For more information on mod_plsql logging, see "What kinds of logging facilities are available in mod_plsql?".

All my PL/SQL procedures return a "Document contains no data" error in Netscape, or a blank page in Internet Explorer.

This problem could occur if you have duplicate OWA Packages installed in the database. See "How do I detect and clean up duplicate OWA packages installed in the database?" for more information.

I have a performant PL/SQL procedure, but some of my HTTP requests through mod_plsql take more than 15 seconds.

The most common reason for this problem is that the middle-tier character set does not match that of the back-end database, and `HTTPKeepAlive` is enabled in Oracle HTTP Server. This kind of misconfiguration causes an invalid *Content-Length* to be sent back to the browser, causing the browser to detect the end of the response stream only when the `KeepAliveTimeout` interval causes the stream to be closed. To solve the problem,

ensure that the `PlsqlNLSLanguage` parameter in the DAD matches that of the database.

If this is not the reason for the problem, there could be performance issues with your PL/SQL application. You can confirm this by:

- Enabling performance logging for mod_plsql. See the section "What kinds of logging facilities are available in mod_plsql?" for more information.
- Running the URL that has performance problems, a few times.
- Scanning the `error_log` for the time reported in "**dbProcTime**". This time denotes the time taken by the target PL/SQL procedure as viewed from the database. If this is high, you will need to debug your PL/SQL application using standard database profiling techniques.

Can I use mod_plsql to run applications on my own database?

Yes. But before you can run your applications, you need to install the OWA packages into your database. See Section 1.2, "Installing Required Packages".

How do I configure mod_plsql?

Please refer to the *Oracle HTTP Server Administrator's Guide*.

How do I create a DAD for mod_plsql?

For manual configuration of DADs, refer to the mod_plsql section in the *Oracle HTTP Server Administrator's Guide*.

What authentication modes are available in mod_plsql?

See Chapter 2, "Securing Application Database Access Through mod_plsql".

What is the mod_plsql Cleanup Thread?

mod_plsql starts a thread in each httpd process. The job of this thread is to clean up idle database sessions and the file system cache. This thread is called the **Cleanup Thread**.

What kind of database connection pooling is present in mod_plsql?

Refer to Chapter 4, "Optimizing PL/SQL Performance".

How does mod_plsql clean up database sessions?

mod_plsql cleans up unused database sessions based on the configuration setting of `PlsqlIdleSessionCleanupInterval`. Besides this, the configuration directive `PlsqlMaxRequestsPerSession` governs how many requests will be serviced from a pooled database session. Finally, database sessions are closed when HTTPD processes are shut down.

What happens when pooled database connections exist in mod_plsql and the database is restarted?

See Section 3.11.3, "Detecting Dead Database Connections in a Connection Pool" for more information.

How does mod_plsql clean up cached content in the file system?

The cleanup thread scans the file system cache based on the configuration of `PlsqlCacheCleanupTime`. The default cleanup time is everyday at 11 P.M. local time.

Can I invoke mod_plsql without a "/pls" prefix in the URL?

Yes. Since mod_plsql uses the Oracle HTTP Server's `Location` directive, you can configure any virtual path to be serviced by mod_plsql.

How can I improve PL/SQL and mod_plsql performance?

Refer to Chapter 4, "Optimizing PL/SQL Performance".

What kinds of logging facilities are available in mod_plsql?

- By default, mod_plsql logs alerts, warnings, and errors to the Oracle HTTP Server `error_log` file *ORACLE_HOME*`/Apache/Apache/logs/error_log`. The amount of information logged by mod_plsql is controlled by the setting of Oracle HTTP Server' `LogLevel` parameter in `httpd.conf`. By default, this is configured to `warn`.
- You can also enable performance logging for mod_plsql on a per-request basis as follows:

  1. Edit *ORACLE_HOME*`/Apache/Apache/conf/httpd.conf` and set `LogLevel` to `info` (default is `warn`).
  2. Restart Oracle HTTP Server using the following command:

     ```
     ORACLE_HOME/opmn/bin/opmnctl restartproc type=ohs
     ```

  3. Issue some URLs to mod_plsql and verify that the file *ORACLE_HOME*`/Apache/Apache/logs/error_log` starts showing entries as follows:

     ```
     [Tue Apr 01 14:54:49 2003] [info] mod_plsql: [perf] 130.35.92.145
     /pls/app/htp.p status=200 user=scott reqTime=21ms connSU=(null),0ms
     connRO=(null),0ms connNSSO=HIT,1ms procTime=17ms sessionTidyTime=0ms
     cache=(null) cookie=(null),0ms pageCalls=0,0ms bytes=5 describe=No,0ms
     streamTime=0ms pid=175 sessFile=(null) userFile=834\0855 sysFile=470\5949
     cacheLevel=(null) cacheTime=0ms dbProcTime=15ms
     id=1049237685:130.35.92.145:373:1 spid=(null) qs=(null) requestTrace=(null)
     cookieLen=0 cookieValue=(null) reqUserTime=16ms assertUser=(null)
     subid=(null) authLevel=(null) oraError=0
     ```

- If you are an OracleAS Portal customer, you can additionally choose to enable Portal session cookie logging by adding the following configuration parameter to your Portal DAD:

  ```
  PlsqlInfoLogging InfoDebug
  ```

  This is in addition to the previous setting of `LogLevel` to `info` in `httpd.conf`.
- Finally, you can enable debug logging in mod_plsql. This is the highest level of logging and is not recommended for active sites.

> **Caution: This mode of logging should be enabled only at the request of Oracle Support.**

In this mode, debug messages are logged to Oracle HTTP Server's `error_log` file and additional mod_plsql specific logs are created under *ORACLE_HOME*`/Apache/modplsql/logs`. Log location is configurable using the `PlsqlLogDirectory` directive in *ORACLE_HOME*`/Apache/modplsql/conf/plsql.conf`. To enable debug level logging:

1. Edit *ORACLE_HOME*`/Apache/modplsql/conf/plsql.conf` and set `PlsqlLogEnable` to **On** (default is **Off**).
2. Restart Oracle HTTP Server using the following command:

```
ORACLE_HOME/opmn/bin/opmnctl restartproc type=ohs
```

See Also: Section 1.5, "Diagnostic Output of mod_plsql"

What kind of DMS metrics are available for mod_plsql?

For information on Dynamic Monitoring Service, refer to the appendix on performance metrics in the *Oracle Application Server Performance Guide*.

What considerations should I have in mod_plsql for High Availability?

For high availability, mod_plsql based applications should be aware of the following things:

- The mod_plsql configuration parameter `PlsqlDatabaseConnectString` should use a connect string format of `NetServiceNameFormat` so that name resolution happens through an LDAP lookup of Oracle Internet Directory. This enables you to configure the database `host:port:service_name` information in a central repository, which makes it easier to add or remove RAC nodes when required.
- mod_plsql automatically detects dead database connections. See Section 3.11.3, "Detecting Dead Database Connections in a Connection Pool" for more information.

What considerations should I have in mod_plsql when the database is separated by a firewall?

If a firewall exists between the middle tier running mod_plsql, and the back end database, the idle session cleanup interval in mod_plsql should be configured lower than the idle session cleanup interval of the firewall. This ensures that the firewall never closes a connection established by mod_plsql.

Note: mod_plsql idle session cleanup interval can be configured using the parameter `PlsqlIdleSessionCleanupInterval` in *ORACLE_HOME*`/Apache/modplsql/conf/plsql.conf`. The default value is 15 minutes.

How do I assert a different hostname, port, or request_protocol to the PL/SQL application?

- In situations where your Oracle HTTP Server instance is front-ended by Web Cache or a Load Balancing Router, there is a need to assert the hostname and port for the site to be that of the Web Cache or the LBR. In such situations, it is recommended that you use the Oracle HTTP Server configuration directives `ServerName` and `Port` to do the assertion. If for some reason, you do not wish to assert the hostname and port at the Oracle HTTP Server level, you can use the

mod_plsql configuration directive `PlsqlCGIEnvironmentList` to assert a different hostname and port to only the PL/SQL applications running under mod_plsql. For example:

- `PlsqlCGIEnvironmentList SERVER_NAME=lbr.us.oracle.com`

  Consider using Oracle HTTP Server's `ServerName` directive in `httpd.conf` instead.

- `PlsqlCGIEnvironmentList SERVER_PORT=9999`

  Consider using Oracle HTTP Server's `Port` directive in `httpd.conf` instead.

- `PlsqlCGIEnvironmentList HTTP_HOST=myservername.us.oracle.com:9999`

  Combination of `SERVER_NAME:SERVER_PORT`.

- Similarly, in cases where your site is accessed externally as an SSL, but is internally running in non-SSL mode (with an SSL accelerator in between), you might want to assert the REQUEST_PROTOCOL as HTTPS so that the PL/SQL application generates SSL links instead of non-SSL links. For example:

```
PlsqlCGIEnvironmentList  REQUEST_PROTOCOL=https
```

How do I disable access to procedure names that have a specific pattern?

Refer to Section 2.1.2, "Adding More Rules to the PlsqlExclusionList Directive in mod_plsql" for more information.

I see the error "HTTP-503 ORA-12154" in the file ORACLE_HOME/Apache/Apache/conf/error_log. What does this mean?

This error means that mod_plsql is unable to connect to the database.

Ensure that:

1. The database is up and running.
2. The username and password information in the DAD is correct.
3. The middle tier is able to connect to the database using the `PlsqlDatabaseConnectString` parameter in the DAD.

In most situations, the problem occurs because SQL*Net is not able to resolve the connect string parameter using the configuration information under *`ORACLE_HOME`*`/network/admin`.

- For entries configured with `TNSFormat` or `NetServiceNameFormat`, validate the connect string information by using `tnsping dad_connect_string`. For example:

  ```
  tnsping "cn=iasdb,cn=oraclecontext"
  ```

  or

  ```
  tnsping iasdb.us.oracle.com
  ```

- For entries configured with `SIDFormat` and `ServiceNameFormat`, ensure that the hostname, port, and SID/service_name information match for the database listener. After verifying this, confirm that SQL*Plus can connect to the database using the DAD username, password, and connect string.

If this does not work, refer to the Oracle SQL*Net documentation on how to troubleshoot this further.

> **Note:** The default connect string parameter in the DAD is configured to get resolved through an LDAP lookup in Oracle Internet Directory. If you make changes to `ldap.ora`, you will need to restart Oracle HTTP Server for the changes to be accessible to mod_plsql.

Why do URLs of the formats /DAD/package.procedure() or /DAD/package.procedure(123) not work when invoking mod_plsql?

mod_plsql does not support URLs of the following formats:

- `/DAD/package.procedure()`
- `/DAD/package.procedure(123)`

For example:

`http://www.acme.com:9000/pls/mydad/mypackage.myproc()` or
`http://www.acme.com:9000/pls/mydad/mypackage.myproc(123)`

For details about the supported URL format, refer to Section 3.3, "Invoking mod_plsql".

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

K

L

M

O

P

R

S

T

U

V

W