5

PL/SQL Server Pages

ORACLE

Objectives

After completing this lesson, you should be able to do the following:

- **Embed PL/SQL code in Web pages (PL/SQL server pages)**
- **Explain the format of a PL/SQL server page**
- **Write the code and content for the PL/SQL server page**
- **Load the PL/SQL server page into the database as a stored procedure**
- **Run a PL/SQL server page via a URL**
- **Debug PL/SQL server page problems**

ORACLE

Objectives

In this lesson, you learn about the powerful features of PL/SQL Server Pages (PSP). Using PSP, you can embed PL/SQL in an HTML Web page.

PSP: Uses and Features

- **Uses:**
  - **If you have a large body of HTML, and want to include dynamic content or make it the front end of a database application**
  - **If most work is done using HTML authoring tools**
- **Features:**
  - **You can include JavaScript or other client-side script code in a PL/SQL server page.**
  - **PSP uses the same script tag syntax as JavaServer Pages (JSP), to make it easy to switch back and forth.**
  - **Processing is done on the server.**
  - **The browser receives a plain HTML page with no special script tags.**

ORACLE

PSP Uses and Features

You can produce HTML pages with dynamic content in several ways. One method is to create PSP. This is useful when you have a large body of HTML, and want to include dynamic content or make it the front end of a database application. If most of the work is done through an HTML authoring tool, PSP is more efficient.

You can also use the PL/SQL Web Toolkit to generate PSPs. This toolkit provides packages such as `OWA`, `htp`, and `htf` that are designed for generating Web pages. For more information, take the *Oracle AS 10g: Develop Web Pages with PL/SQL* course. This is useful when there is a large body of PL/SQL code that produces formatted output. If you use authoring tools that produce PL/SQL code for you, such as the page-building wizards in Oracle Application Server Portal, then it might be less convenient to use PSP.

Format of the PSP File

- The file must have a `.psp` extension.
- The `.psp` file can contain text, tags, PSP directives, declarations, and scriptlets.
- Typically, HTML provides the static portion of the page, and PL/SQL provides the dynamic content.

Test.psp

Format of the PSP File

It is easier to maintain the PSP file if you keep all your directives and declarations together near the beginning of a PL/SQL server page. To share procedures, constants, and types across different PL/SQL server pages, compile them into a separate package in the database by using a plain PL/SQL source file. Although you can reference packaged procedures, constants, and types from PSP scripts, the PSP scripts can only produce stand-alone procedures, not packages.

Page Directive

Specifies characteristics of the PL/SQL server page:

- What scripting language it uses
- What type of information (MIME type) it produces
- What code to run to handle all uncaught exceptions. This might be an HTML file with a friendly message, renamed to a `.psp` file.

Syntax:

```
<%@ page [language="PL/SQL"]
contentType="content type string"] [errorPage="file.psp"] %>
```

Procedure Directive

Specifies the name of the stored procedure produced by the PSP file. By default, the name is the file name without the `.psp` extension.

Syntax:

```
<%@ plsql procedure="procedure name" %>
```

Format of the PSP File (continued)

Parameter Directive

Specifies the name, and optionally the type and default, for each parameter expected by the PSP stored procedure.

Syntax:

```
<%@ plsql parameter="parameter name"
 [type="PL/SQL type"] [default="value"] %>
```

If the parameter data type is `CHARACTER`, put single quotation marks around the default value, with double quotation marks surrounding the entire default value.

Include Directive

Specifies the name of a file to be included at a specific point in the PSP file. The file must have an extension other than `.psp`. It can contain HTML, PSP script elements, or a combination of both. The name resolution and file inclusion happens when the PSP file is loaded into the database as a stored procedure, so any changes to the file after that are not reflected when the stored procedure is run.

Syntax:

```
<%@ include file="path name" %>
```

Declaration Block

Declares a set of PL/SQL variables that are visible throughout the page, not just within the next `BEGIN/END` block. This element typically spans multiple lines, with individual PL/SQL variable declarations ended by semicolons.

Syntax:

```
<%! PL/SQL declaration; [ PL/SQL declaration; ] ... %>
```

Code Block (Scriptlets)

Executes a set of PL/SQL statements when the stored procedure is run. This element typically spans multiple lines, with individual PL/SQL statements ended by semicolons. The statements can include complete blocks, or can be the bracketing parts of `IF/THEN/ELSE` or `BEGIN/END` blocks. When a code block is split into multiple scriptlets, you can put HTML or other directives in the middle, and those pieces are conditionally executed when the stored procedure is run.

Syntax:

```
<% PL/SQL statement; [ PL/SQL statement; ] ... %>
```

Expression Block

Specifies a single PL/SQL expression, such as a string, an arithmetic expression, a function call, or a combination of those things. The result is substituted as a string at that spot in the HTML page that is produced by the stored procedure. You do not need to end the PL/SQL expression with a semicolon.

Syntax:

```
<%= PL/SQL expression %>
```

Note: To identify a file as a PL/SQL server page, include a `<%@ page language="PL/SQL" %>` directive somewhere in the file. This directive is for compatibility with other scripting environments.

Development Steps for PSP

1. Create the PSP.
2. Load the PSP into the database as a stored procedure.

```
loadpsp [ -replace ]
-user username/password[@connect_string]
[ include_file_name ... ] [ error_file_name ]
psp_file_name ...
```

3. Run the PSP through a URL.

```
http://sitename/schemaname/pspname?parmname1=
value1&parmname2=value2
```

ORACLE

Steps to Create a PSP

Step 1

Create an HTML page, embedding the PL/SQL code in the HTML page.

Development Steps for PSP

Creating the PSP:

```
<%@ page language="PL/SQL" %>          ← Page directive
<%@ plsql procedure="show_table" %>    ← Procedure directive
<% -- Inventories Table Contents -- %> ← Comment
<HTML>
<HEAD><TITLE>Show Contents of Inventories </TITLE></HEAD>
<BODY>
<p><b><font face="Arial, Helvetica, Tahoma"
 size="4">INVENTORIES TABLE: </font></b></p>
<p><%
declare                                ← Scriptlet
dummy boolean;
begin
dummy := owa_util.tableprint('INVENTORIES','border');
end;
%> </p>
<p> </p><p> </p><p> </p></BODY>
</HTML>
```

ORACLE

Creating the PSP

First, create an HTML page, embedding the PL/SQL code in the HTML page. In this example, the contents of the `INVENTORIES` table are displayed in a Web page.

The page directive identifies the scripting language. The procedure directive identifies that a procedure named `show_table` will be created and stored in the database to represent this HTML page. The scriptlet executes a set of PL/SQL statements when the stored procedure is run. The result is substituted as a string at that spot in the HTML page that is produced by the stored procedure. The `owa_util.tableprint` procedure prints out the contents of a database table that is identified to the procedure through the first parameter.

Note: `owa_util.tableprint` is part of the PL/SQL Web Toolkit and is installed in the `SYS` schema.

Include Comments

To put a comment in the HTML portion of a PL/SQL server page, for the benefit of people reading the PSP source code, use the following syntax:

Syntax:

```
<%-- Comment text --%>
```

These comments do not appear in the HTML output from the PSP.

To create a comment that is visible in the HTML output, place the comment in the HTML portion and use the regular HTML comment syntax:

Syntax:

```
<!-- Comment text -->
```

Development Steps for PSP

- **Loading the PSP into the database from the operating system:**

```
>loadpsp -replace -user oe/oe show_table.psp
"show_table.psp" : procedure "show_table" created.
>
```

- **Optionally include other file names and the error file name:**

```
>loadpsp -replace -user oe/oe
banner.inc error.psp show_table.psp
"banner.inc": uploaded.
"error.psp": procedure "error" created.
"show_table.psp" : procedure "show_table" created.
>
```

ORACLE

Loading the PSP

Step 2

In the second step, you load one or more PSP files into the database as stored procedures. Each `.psp` file corresponds to one stored procedure. To perform a "`CREATE OR REPLACE`" on the stored procedures, include the `-replace` flag.

The loader logs on to the database using the specified username, password, and connect string. The stored procedures are created in the corresponding schema.

In the first example:

- The stored procedure is created in the database. The database is accessed as user `oe` with password `oe`, both when the stored procedure is created and when it is executed.
- `show_table.psp` contains the main code and text for the Web page.

In the second example:

- The stored procedure is created in the database. The database is accessed as user `oe` with password `oe`, both to create the stored procedure and when the stored procedure is executed.
- `banner.inc` is a file containing boilerplate text and script code, that is included by the `.psp` file. The inclusion happens when the PSP is loaded into the database, not when the stored procedure is executed.
- `error.psp` is a file containing code or text that is processed when an unhandled exception occurs, to present a friendly page rather than an internal error message.
- `show_table.psp` contains the main code and text for the Web page.

Loading the PSP (continued)

Include the names of all the include files (whose names do not have the `.psp` extension) before the names of the PL/SQL server pages (whose names have the `.psp` extension). Also include the name of the file specified in the `errorPage` attribute of the page directive. These file names on the `loadpsp` command line must exactly match the names specified within the PSP include and page directives, including any relative pathname such as `../include/`.

Development Steps for PSP

The `show_table` procedure is stored in the data dictionary views.

```
SQL> SELECT text
  2  FROM   user_source
  3  WHERE  name = 'SHOW_TABLE';

TEXT
------------------------------------------------------------------
PROCEDURE show_table  AS
 BEGIN NULL;
...
declare
dummy boolean;
begin
dummy := owa_util.tableprint('INVENTORIES','border');
end;
...
23 rows selected.
```

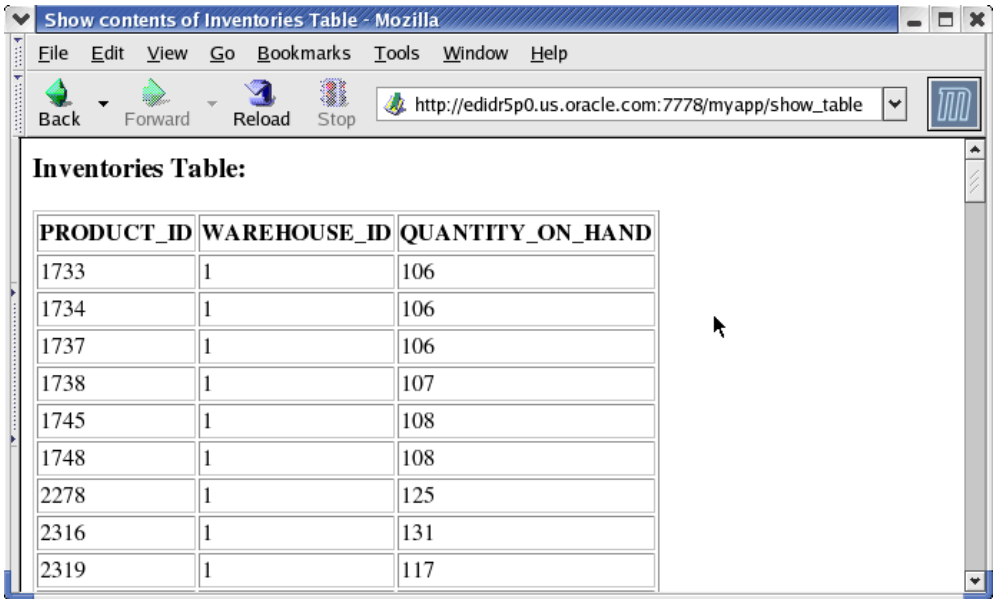
ORACLE

Loading the PSP (continued)

After the `loadpsp` utility is run, the procedure is created and stored in the database.

Development Steps for PSP

Running the PSP through a URL:



PRODUCT_ID	WAREHOUSE_ID	QUANTITY_ON_HAND
1733	1	106
1734	1	106
1737	1	106
1738	1	107
1745	1	108
1748	1	108
2278	1	125
2316	1	131
2319	1	117

Running the PSP

Step 3

For the third step, run the PSP in a browser. Identify the HTTP URL through a Web browser or some other Internet-aware client program. The virtual path in the URL depends on the way the Web gateway is configured. The name of the stored procedure is placed at the end of the virtual path.

Using `METHOD=GET`, the URL may look like this:

```
http://sitename/DAD/pspname?parmname1=value1&parmname2=value2
```

Using `METHOD=POST`, the URL does not show the parameters:

```
http://sitename/DAD/pspname
```

The `METHOD=GET` format is more convenient for debugging and allows visitors to pass exactly the same parameters when they return to the page through a bookmark.

The `METHOD=POST` format allows a larger volume of parameter data, and is suitable for passing sensitive information that should not be displayed in the URL.

Printing the Table Using a Loop

- **To print the results of a multirow query, use a loop:**

```
<% FOR item IN (SELECT * FROM some_table) LOOP %>
  <TR>
  <TD><%= item.col1 %></TD>
  <TD><%= item.col2 %></TD>
  </TR>
<% END LOOP; %>
```

- **Alternatively, use `OWA_UTIL.TABLEPRINT` or `OWA_UTIL.CELLSPRINT` procedures from the PL/SQL Web Toolkit.**

ORACLE

Printing the Content of a Table

You can iterate through each row of the result set, printing the appropriate columns using HTML list or table tags. Following is an example of a list:

```
<%@ page language="PL/SQL" %>
<%@ plsql procedure="show_customers" %>
<HTML>
<HEAD><TITLE>Show Contents of Customers (using a loop)
</TITLE></HEAD>
<BODY>
<UL>
<% for item in (select customer_id, cust_first_name,
                credit_limit, cust_email
                from customers order by credit_limit) loop %>
<LI>
ID = <%= item.customer_id %><BR>
Name = <%= item.cust_first_name %><BR>
Credit = <%= item.credit_limit %><BR>
Email = <I><%= item.cust_email %></I><BR>
<% end loop; %>
</UL>
</BODY>
</HTML>
```

Specifying a Parameter

- **Include the parameter directive in the `.psp` file.**
  - **Syntax:**

```
<%@ plsql parameter="parameter name"
 [type="PL/SQL type"] [default="value"] %>
```

  - **Example:**

```
<%@ plsql parameter="mincredit" type="NUMBER"
default="3000" %>
```

- **Assign the parameter a value through the URL call:**

```
http://edidr5p0.us.oracle.com/DAD
/show_customers_hc?mincredit=3000
```

ORACLE

Specifying a Parameter

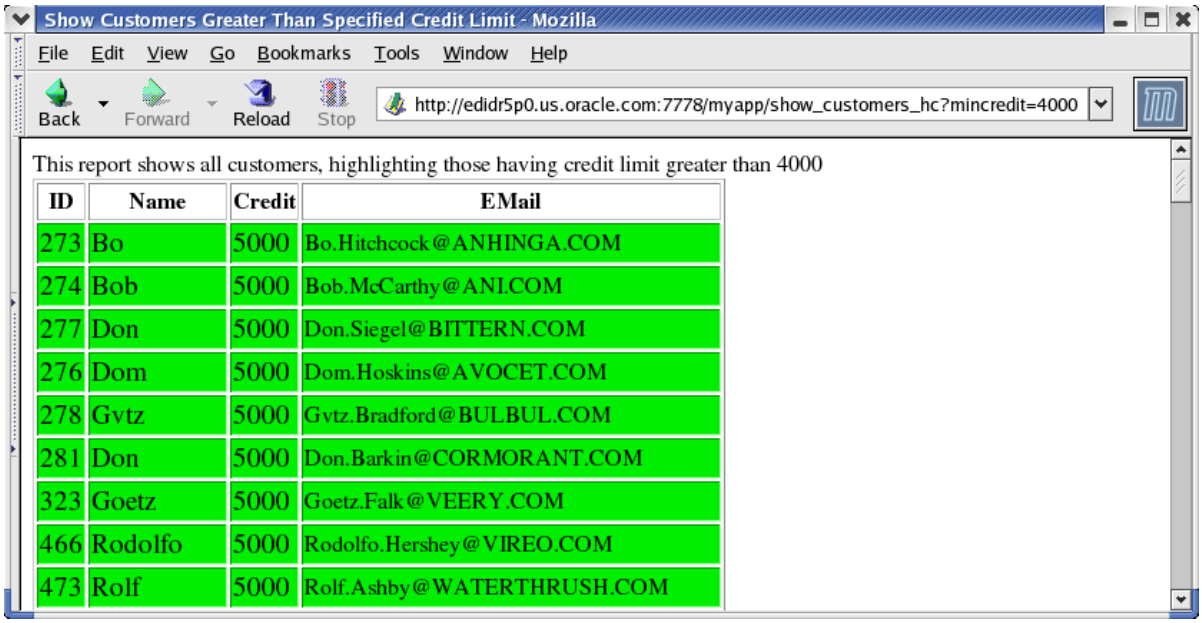
You can pass parameters to the PSP by identifying the parameter name and value in the URL call.

Specifying a Parameter (continued)

The example below creates a parameter named `mincredit`. There is also some conditional processing to highlight values that are greater than a specified price.

```
<%@ page language="PL/SQL" %>
<%@ plsql procedure="show_customers_hc" %>
<%@ plsql parameter="mincredit" type="NUMBER" default="3000"
%>
<%! color varchar2(7); %>
<HTML>
<HEAD><TITLE>Show Customers Greater Than Specified Credit
Limit</TITLE></HEAD>
<BODY>
<P>This report shows all customers, highlighting those having
credit limit is greater than <%= mincredit %>.
<TABLE BORDER>
<TR>
<TH>ID</TH>
<TH>Name</TH>
<TH>Credit</TH>
<TH>Email  </TH>
</TR>
<%
for item in (select * from customers
             order by credit_limit desc) loop
  if item.credit_limit > mincredit then
    color := '#white';
  else
    color := '#green';
  end if;
%>
<TR BGCOLOR="<%= color %>">
<TD><BIG><%= item.customer_id %></BIG></TD>
<TD><BIG><%= item.cust_first_name %></BIG></TD>
<TD><BIG><%= item.credit_limit %></BIG></TD>
<TD><%= item.cust_email %></TD>
</TR>
<% end loop; %>
</TABLE>
</BODY>
</HTML>
```

Specifying a Parameter



ID	Name	Credit	EMail
273	Bo	5000	Bo.Hitchcock@ANHINGA.COM
274	Bob	5000	Bob.McCarthy@ANI.COM
277	Don	5000	Don.Siegel@BITTERN.COM
276	Dom	5000	Dom.Hoskins@AVOCET.COM
278	Gvtz	5000	Gvtz.Bradford@BULBUL.COM
281	Don	5000	Don.Barkin@CORMORANT.COM
323	Goetz	5000	Goetz.Falk@VEERY.COM
466	Rodolfo	5000	Rodolfo.Hershey@VIREO.COM
473	Rolf	5000	Rolf.Ashby@WATERTHRUSH.COM

ORACLE

Specifying a Parameter (continued)

You passed `mincredit=4000` as the parameter along with the URL. The output shows all the records and highlights those having a credit limit greater than 4,000.

Using an HTML Form to Call a PSP

1. **Create an HTML form.**
2. **Call the PSP from the form.**

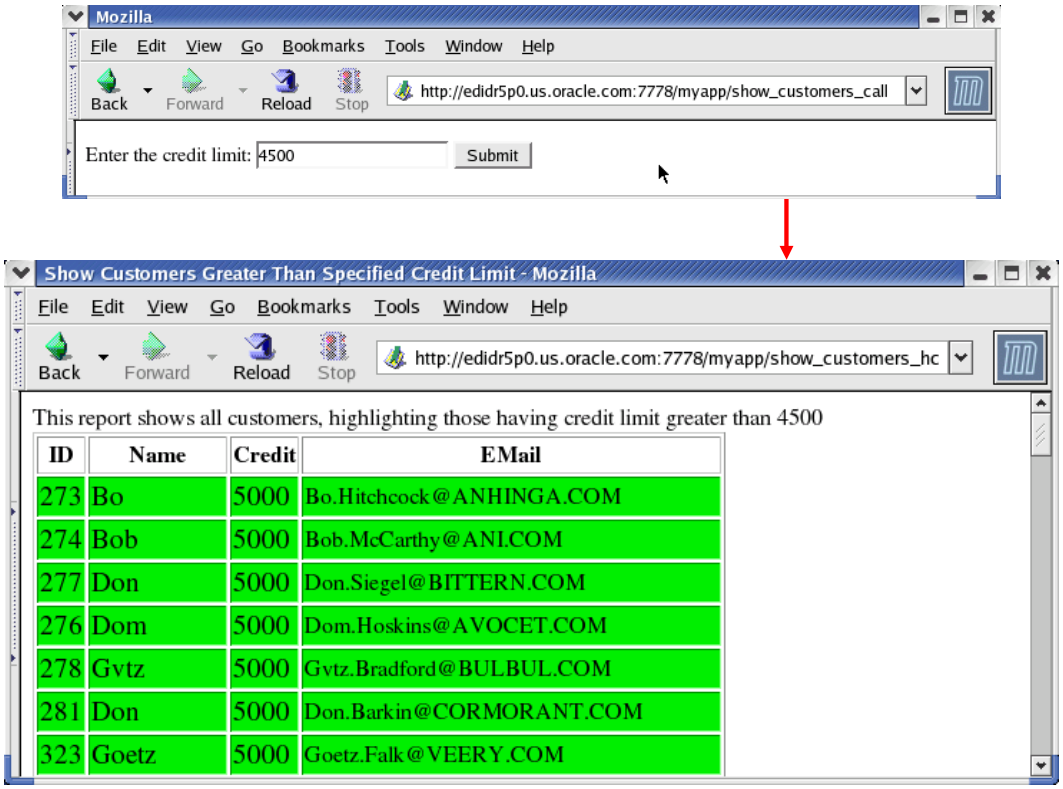
```
<%@ page language="PL/SQL" %>
<%@ plsql procedure="show_customer_call" %>
<%@ plsql parameter="mincredit" type="NUMBER" default=
"3000" %>                                          ← 1
<html>
<body>
<form method="POST" action="show_customers_hc">←  2
<p>Enter the credit limit:
<input type="text" name="mincredit">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

ORACLE

Calling a PSP from an HTML Form

Create an HTML form that calls the PSP. To avoid coding the entire URL of the stored procedure in the `ACTION= attribute` of the form, make the form a PSP file so that it goes in the same directory as the PSP file it calls.

Using an HTML Form to Call a PSP



ORACLE

Calling a PSP from an HTML Form (continued)

Initially, you are calling the HTML form that accepts the credit limit from the user. After submitting the HTML form, call the PSP, which shows all the records and highlight all the records having a credit limit greater than the value submitted by the user.

Debugging PSP Problems

- **Code the PL/SQL syntax and PSP directive syntax correctly. It will not compile with syntax errors.**
- **Run the PSP file by requesting its URL in a Web browser. An error might indicate that the file is not found.**
- **When the PSP script is run, and the results come back to the browser, use standard debugging techniques to check for and correct wrong output.**
- **Use `htp.p('string')` to print information to the screen.**

ORACLE

Debugging PSP Problems

The first step is to code PL/SQL syntax and PSP directive syntax correctly. It will not compile with syntax errors.

- Use semicolons to terminate lines if required.
- If required, enclose a value with quotation marks. You may need to enclose a value that is within single quotation marks (needed by PL/SQL) inside double quotation marks (needed by PSP).
- Mistakes in the PSP directives are usually reported through PL/SQL syntax messages. Check that your directives use the right syntax, that directives are closed properly, and that you are using the right element (declaration, expression, or code block) depending on what goes inside it.
- PSP attribute names are case sensitive. Most are specified in all lowercase; `contentType` and `errorPage` must be specified as mixed-case.

Run the PSP file by requesting its URL in a Web browser.

- Request the right virtual path, depending on the way the Web gateway is configured. Typically, the path includes the host name, optionally a port number, the schema name, and the name of the stored procedure (with no `.psp` extension).
- If you use the `-replace` option when compiling the file, the old version of the stored procedure is erased. You may want to test new scripts in a separate schema until they are ready, then load them into the production schema.
- If you copied the file from another file, remember to change any procedure name directives in the source to match the new file name.

Debugging PSP Problems (continued)

- If you receive one file-not-found error, make sure to request the latest version of the page the next time. The error page may be cached by the browser. You may need to press [Shift] and click Reload in the browser to bypass its cache.

When the PSP script is run, and the results come back to the browser, use standard debugging techniques to check for and correct wrong output. The tricky part is to set up the interface between different HTML forms, scripts, and CGI programs so that all the right values are passed into your page. The page may return an error because of a parameter mismatch.

- To see exactly what is being passed to your page, use `METHOD=GET` in the calling form so that the parameters are visible in the URL.
- Make sure that the form or CGI program that calls your page passes the correct number of parameters, and that the names specified by the `NAME=attributes` on the form match the parameter names in the PSP file. If the form includes any hidden input fields, or uses the `NAME= attribute` on the Submit or Reset buttons, then the PSP file must declare equivalent parameters.
- Make sure that the parameters can be cast from string into the correct PL/SQL types. For example, do not include alphabetic characters if the parameter in the PSP file is declared as a `NUMBER`.
- Make sure that the query string of the URL consists of name-value pairs, separated by equal signs, especially if you are passing parameters by constructing a hard-coded link to the page.
- If you are passing a lot of parameter data, such as large strings, you may exceed the volume that can be passed with `METHOD=GET`. You can switch to `METHOD=POST` in the calling form without changing your PSP file.
- You can display text or variables by putting the following in your code:

  ```
  htp.p(' My Var: ' || my_var);
  ```

  When you run the program, the information is displayed on the screen.

Summary

In this lesson, you should have learned how to:

- **Define PL/SQL server pages**
- **Explain the format of a PL/SQL server page**
- **Write the code and content for the PL/SQL server page**
- **Load the PL/SQL server page into the database as a stored procedure**
- **Run a PL/SQL server page via a URL**
- **Debug PL/SQL server page problems**

ORACLE

Summary

You can use PL/SQL embedded in HTML and store the code as a PL/SQL server page (PSP)in the database. The three steps for creating a PSP are:

1. Create the PSP.
2. Load the PSP into the database as a stored procedure.
3. Run the PSP in a browser.

Practice Overview

This practice covers the following topics:

- **Creating a PSP**
- **Loading a PSP**
- **Running the PSP through the browser**

ORACLE

Practice Overview

In this practice, you write and deploy a PSP that retrieves order information. You will also write and deploy a PSP that retrieves customer information where Customer ID is passed as a parameter.

Practice 5

Note: The instructor needs to set up a DAD for the class.

1. Create a PL/SQL server page to display order information. Name the procedure as `show_orders`. Display the following fields:

```
ORDER_ID
ORDER_MODE
CUSTOMER_ID
ORDER_STATUS
ORDER_TOTAL
TAX
SALES_REP_ID
```

Note: `TAX` should be displayed using the `calc_c` function created in Lesson 4.

   a. Use the `lab_05_01.psp` file containing the HTML code. After creating the PSP, load it from the operating system

   b. Request the `show_orders` PSP from your browser.

2. Create a PL/SQL server page to display the following customer information:

```
CUSTOMER_ID
CUST_FIRST_NAME
CUST_LAST_NAME
CREDIT_LIMIT
CUST_EMAIL
```

   a. Use the `lab_05_02a.psp` file containing the HTML code. Name the procedure `show_cust`.

   b. Use a parameter to pass `CUSTOMER_ID` and then display information for that customer.

   c. Use an HTML form to call the PSP. Modify the `lab_05_02b.psp` file and add the necessary details to call the PSP.