

#1 Book for SQL*Plus Solutions & Syntax

2nd Edition
Updated for
Oracle Database 10g



Oracle SQL*Plus

The Definitive Guide

O'REILLY®

Jonathan Gennick

Creating HTML Reports

Beginning with Oracle8i Database Release 2 (Version 8.1.6), you can use SQL*Plus to generate HTML pages. Instead of spooling a report to a printer, you can write it to an HTML file for viewing on your corporate intranet. With a bit of creativity and some cascading style sheet (CSS) skills, you can generate some good-looking web pages. In this chapter, I'll revisit some of the character-based reports from Chapter 5 and show some ways to render those same reports as HTML pages.



CSS is a mechanism for specifying the format of an HTML document. One reason CSS works well for SQL*Plus-generated HTML is because your CSS styles can reside in a file separate from the spool file generated by SQL*Plus, greatly reducing the likelihood that you will need to modify your SQL*Plus script to effect formatting changes. To learn more about CSS, read Eric Meyer's *Cascading Style Sheets: The Definitive Guide* (O'Reilly) and *Eric Meyer on CSS* (New Riders).

Getting the Data into an HTML Table

SQL*Plus doesn't offer much in the way of formatting HTML output, and what little capability SQL*Plus does offer in this area, you're better off avoiding in favor of CSS. When writing a report as HTML, SQL*Plus places all the detail rows, and their column headings, into an HTML table. One approach to generating an HTML page with such data is to capture only that table and concatenate it with other HTML that you write yourself to generate a complete web page.

When generating HTML from SQL*Plus, keep things simple:

- Pagination doesn't apply. All "pages" are written to one HTML file. Thus, you're better off thinking in terms of one, large page. To that end, set PAGESIZE to its maximum value of 50,000.

- Line size is irrelevant. Each column value will be in its own table cell. Your browser, together with any CSS styles and/or table markup that you supply, will control the width of that table and its columns.
- Don't worry about word-wrapping. Your browser will wrap table cell values as needed depending on how the table and its columns are sized. The COLUMN command's WORD_WRAPPED is meaningless in the context of generating HTML.

Example 6-1 shows a stripped-down version of the Project Hours and Dollars Report that you saw in Examples 5-1 through 5-5. This time, the report is output in HTML form. PAGESIZE is 50000 to ensure that only one set of column headings is generated.

*Example 6-1. SQL*Plus script to generate an HTML table of report data*

```
SET ECHO OFF
SET PAGESIZE 50000
SET MARKUP HTML ON TABLE ""

--Format the columns
COLUMN employee_name HEADING "Employee Name"
COLUMN project_name HEADING "Project Name"
COLUMN hours_logged HEADING "Hours" FORMAT 9,999
COLUMN dollars_charged HEADING "Dollars|Charged" FORMAT $999,999.99

--Turn off feedback and set TERMOUT off to prevent the
--report being scrolled to the screen.
SET FEEDBACK OFF
SET TERMOUT OFF

--Execute the query to generate the report.
SPOOL middle.html
SELECT e.employee_name,
       p.project_name,
       SUM(ph.hours_logged) hours_logged,
       SUM(ph.dollars_charged) dollars_charged
FROM employee e INNER JOIN project_hours ph
  ON e.employee_id = ph.employee_id
  INNER JOIN project p
  ON p.project_id = ph.project_id
GROUP BY e.employee_id, e.employee_name,
         p.project_id, p.project_name;
SPOOL OFF
```



Remember that these report scripts are designed to be invoked from the operating system command prompt, as in:

```
sqlplus username/password @ex6-1.sql
```

Avoid invoking them interactively from the SQL*Plus prompt, as you won't be starting with a clean slate each time with respect to the various SET, COLUMN, and TTITLE commands.

The key command in Example 6-1 is the SET MARKUP command:

```
SET MARKUP HTML ON TABLE ""
```

You can break down this command as follows:

```
SET MARKUP HTML ON
```

Causes SQL*Plus to write report output into an HTML table. All output is written in HTML, but the table is all we care about for this example.

```
TABLE ""
```

Specifies that no attributes are to be written into the opening table tag. If you don't specify otherwise, SQL*Plus writes some default attributes, such as `width="90%"`, but I recommend avoiding those in favor of using CSS to format your table.

Output from the script in Example 6-1 will be written to a file named *middle.html*. The file begins with a stray `<p>` tag, which SQL*Plus perhaps generates as a crude way of putting some space before the table. Whatever the reason for the tag, it's a bit of an annoyance when using CSS to format your output. The `<p>` tag is followed by an HTML table containing report data. Column headings are wrapped in `<th>` tags in the first row of that table. The *middle.html* file then, begins as follows:

```
<p>
<table>
<tr>
<th scope="col">
Employee Name
</th>
<th scope="col">
Project Name
</th>
<th scope="col">
Hours
</th>
<th scope="col">
Dollars
<br>
Charged
</th>
</tr>
...
```

Each row of report data is written as a row in the HTML table. For example, here is the markup written to *middle.html* for the first data row:

```
...
<tr>
<td>
Marusia Churai
</td>
<td>
Corporate Web Site
```

```

</td>
<td align="right">
    20
</td>
<td align="right">
    $3,380.00
</td>
</tr>
...

```

The *middle.html* file is not a complete HTML page. Examples 6-2 and 6-3 show two additional files containing HTML markup, *first.html* and *last.html*. By wrapping the contents of *middle.html* within the contents of those other two files, you can produce a valid HTML page. On Linux/Unix systems, you can put all the pieces together with a *cat* command:

```
cat first.html middle.html last.html > proj_hours_dollars.html
```

On Windows systems, use the *type* command:

```
type first.html, middle.html, last.html > proj_hours_dollars.html
```

Example 6-2. The first.html file with markup to begin a page

```

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=WINDOWS-1252">
<title>Project Hours and Dollars Report</title>
<style type="text/css">
    table {border-collapse: collapse; width: 100%;}
    td {border: 2 solid black;}
    th {border: 2 solid black;}
</style>
<body>
<h1>Project Hours and Dollars Report</h1>

```

Example 6-3. The last.html file with markup to end a page

```

</body>
</html>

```



If you download the example scripts for this book, *first.html* and *last.html* are named *ex6-2.html* and *ex6-3.html* respectively. This is so they conform to the same naming convention as all other examples.

Figure 6-1 shows the resulting page as rendered in a browser. It's not a fancy-looking page. I'm focusing on functionality in this first example and don't want to clutter it with all sorts of HTML and CSS markup. I hope you can see, however, the potential for creativity from your ability to control the content of *first.html* and *last.html*.

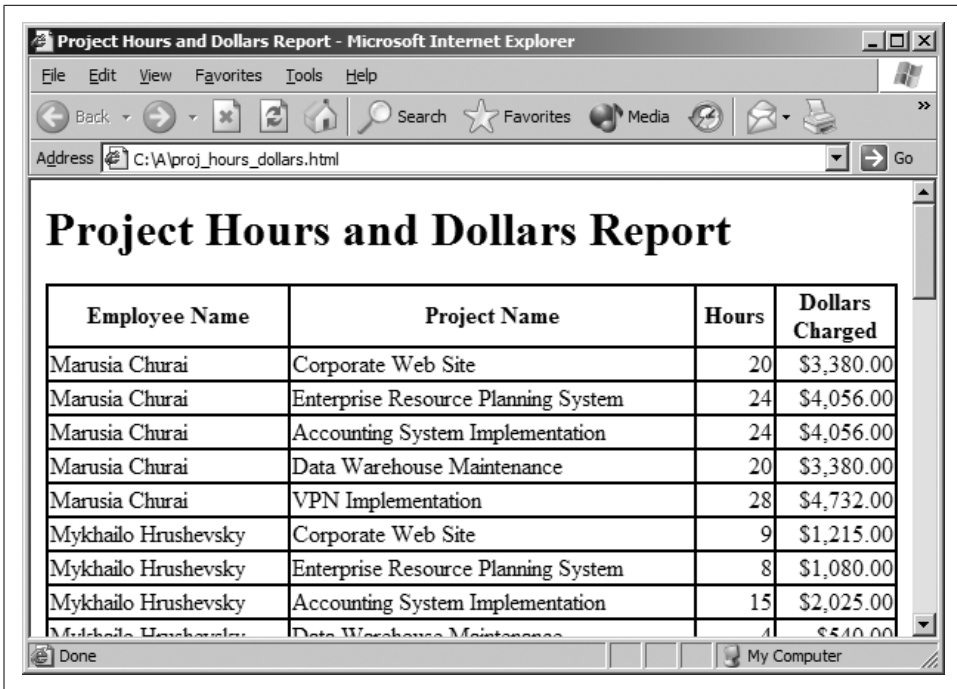


Figure 6-1. An HTML page with report content generated from SQL*Plus

Generating the Entire Page

Having SQL*Plus generate the entire HTML page is possible. SQL*Plus generates a fairly decent-looking page by default, at least in Oracle Database 10g. The one problem you'll likely encounter is that the kinds of page headings you may be used to generating for plain-text reports do not always translate well into headings for HTML reports.

Using SQL*Plus's Default Formatting

To generate a report as a complete HTML page using SQL*Plus's default HTML formatting, add the command `SET MARKUP HTML ON SPOOL ON` near the beginning of any report script. Following is the beginning of Example 5-5 (in the file `ex5-5.sql`), with the `SET MARKUP` command added as the second line:

```
SET ECHO OFF
SET MARKUP HTML ON SPOOL ON

--Setup pagesize parameters
SET NEWPAGE 0
SET PAGESIZE 55
```

```

--Set the linesize, which must match the number of equal signs used
--for the ruling lines in the headers and footers.
SET LINESIZE 61

--Setup page headings and footings
TTITLE CENTER "The Fictional Company" SKIP 3 -
      LEFT "I.S. Department" -
      RIGHT "Project Hours and Dollars Report" SKIP 1 -
      LEFT "=====
...

```

The SPOOL ON option to SET MARKUP causes SQL*Plus to write <html>, <head>, and <body> tags before writing data from the query to the spool file, and to close those tags before closing the file. Thus, the result from running the script is a complete web page that you can load into a browser. SQL*Plus will write a series of CSS-style definitions into the HTML heading between the <head> and </head> tags. The result, assuming you began with Example 5-5, is shown in Figure 6-2.

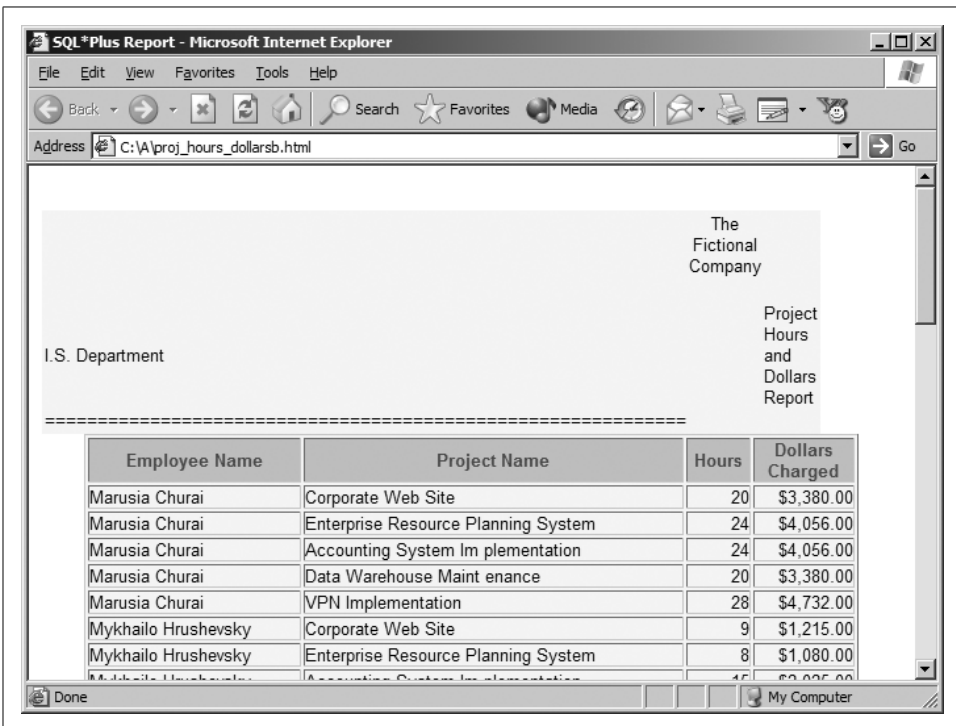


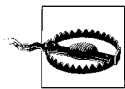
Figure 6-2. Page headers in HTML tables don't work well

The page heading in Figure 6-2 looks ugly, doesn't it? To understand why that is, you need to understand a bit about how SQL*Plus writes HTML page headings. Such headings are written to a table separate from the table used for data from the SELECT statement. This table has three columns: one for the TTITLE content following the

LEFT keyword; another for content following the CENTER keyword; and the third column for content following the RIGHT keyword.

This use of a table for page headings doesn't work as well as you might think. That long, horizontal line of equal sign (=) characters that you see in the figure is specified to be LEFT in the TTITLE command. In a plain-text report, the line extends out under the CENTER and RIGHT content. In an HTML report, however, the line can't undercut the other content because it is restricted to the leftmost column in the heading table. All those characters must display, though, and the line ends up showing the CENTER and RIGHT content way off to the extreme right-hand side of your browser window.

When generating reports for HTML, it's best to stick with simple, uncomplicated page headings. In fact, I recommend keeping everything LEFT. Use CSS to center any heading lines that you wish centered. Don't mix alignments on the same heading line. For example, avoid using LEFT and CENTER on the same line. If you must mix two alignments, keep the different parts of your heading text short, so text from one HTML table column doesn't push other columns out of their proper alignment.



Take care to follow any SKIP directive in a TTITLE immediately with an alignment directive such as LEFT. If you fail to do that, SQL*Plus seems to lose track of which cell subsequent text should fall into, and that text will be lost.

Avoid SKIPPING more than one line in a TTITLE. When you skip multiple lines, SQL*Plus inserts blank rows into the table. This is a poor way to generate vertical space in your output. Use CSS instead.

Taking Control of the Page Format

You don't have to rely on SQL*Plus's default styles to format your HTML reports. You hold the reins of power and can take almost complete control. The script in Example 6-4 generates a complete web page. Figure 6-3 shows that page as rendered using the CSS styles in Example 6-5. There's a lot to explain in this script and in the accompanying CSS stylesheet. I'll begin by explaining the SET MARKUP command:

SET MARKUP -

Begins the command.

HTML ON -

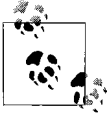
Enables HTML output.

HEAD '<title>Project Hours and Dollars Report</title>' -

<link href="ex6-5.css" rel="stylesheet" type="text/css"/>' -

Specifies content to be written between <head> and </head>. This content replaces the default set of CSS styles that SQL*Plus otherwise writes. I've

wrapped this content within single quotes, so I can write double quotes into the HTML file. The `<title>` tag specifies a page title that most browsers will display on their titlebar. The `<link>` tag causes the browser displaying the HTML page to format the page using the CSS styles defined in `ex6-5.css`.



I prefer external stylesheets. However, you can take all the styles from Example 6-5 and add them to the HEAD parameter to have them written inline in the HTML document's header.

BODY "" -

Eliminates default attributes that SQL*Plus otherwise writes into the `<body>` tag. Use CSS to format your HTML body.

TABLE 'class="detail"' -

Gives a class name to the HTML table created by SQL*Plus to hold the rows of data in the report. This eliminates the default attributes that SQL*Plus would otherwise write.

ENTMAP OFF -

Prevents SQL*Plus from translating `<` and `>` characters in the script's output stream to the HTML named character entities `<` and `>`. My approach here may be somewhat controversial, but it allows you to place HTML tags into the page title and column headings, and to good effect as you'll soon see.

SPOOL ON

Causes SQL*Plus to write `<html>`, `<head>`, and `<body>` tags before writing data from a query to a spool file and to close those tags before closing the file.

I realize that all this advice is a lot to absorb. The key here is to realize that I've specified a link to an external stylesheet, and that I've enabled the script to write HTML tags directly to the spool file. If you look at the stylesheet in Example 6-5, you'll see that the body margin is zero and that I've specified the *comic sans ms* font (not a business-like font, but I like it). The zero margin eliminates any white border that would otherwise surround my content.

Next, take a look at the page heading as specified by TTITLE:

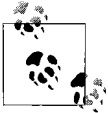
```
TTITLE LEFT "<h1>The Fictional Company</h1>" SKIP 1 -  
LEFT "<h2>Project Hours and Dollars Report</h2>"
```

Only one "page" should be in this report because PAGESIZE is set to 50000 and I know there are fewer than 50,000 rows to be returned by the report's query. The following two lines will be written to the top of the HTML page, prior to the table containing the report data:

```
<h1>The Fictional Company</h1>  
<h2>Project Hours and Dollars Report</h2>
```

This is elementary HTML markup: a top-level heading followed by one subheading. By being able to place HTML tags into a page heading, you gain the ability to format

that heading to use CSS styles. The stylesheet in Example 6-5 specifies a zero top margin for h1 and h2 to keep them close together.



In an adventurous mood? Remove the second LEFT directive from the TTITLE command and execute the script. In Oracle Database 10g Release 1 at least, the resulting report will be missing the h2 subhead. Always begin each heading line in a TTITLE with an alignment directive.

To format column headings, I place HTML markup into the HEADING text in my COLUMN commands:

```
COLUMN employee_name HEADING "<p class=left>Employee Name</p>" FORMAT A40
```

All I do here is to wrap each heading in a paragraph tag (<p>) and assign it a class. In this case, my classes are left and right, and I use those styles to align text headings to the left and numeric headings to the right. Otherwise, all headings end up centered over their columns, which doesn't always look good.

When you place HTML markup in text column headings, take care to format each column wide enough to accommodate its markup. I specified FORMAT A40 for employee_name to allow for up to 40 characters in the <p> tag. Were I to specify FORMAT A9, my heading text would be truncated to nine characters, with the result that the incomplete tag <p class= would be written to my HTML file. When generating HTML reports, don't worry about using FORMAT to control the width of a column. Instead, use FORMAT to ensure enough width for your entire heading, including any HTML markup that you add.



I have not experienced truncation problems with headings of numeric columns. Thus, I can use FORMAT 9,999 for hours_logged without fear that my heading markup, which is much longer than just five characters, will be truncated.

The script in Example 6-4 writes two tables into the HTML file, one for the headings and the other for the data. Example 6-5 specifies the following two styles for these tables:

```
table {background: black; color: white; width: 100%;}  
table.detail {background: #e0e0e0; color: black}
```

*Example 6-4. SQL*Plus script to generate a complete web page*

```
SET ECHO OFF  
SET PAGESIZE 50000  
SET MARKUP -  
  HTML ON -  
  HEAD '<title>Project Hours and Dollars Report</title> -  
      <link href="ex6-5.css" rel="stylesheet" type="text/css"/>' -  
  BODY "" -  
  TABLE 'class="detail"' -
```

*Example 6-4. SQL*Plus script to generate a complete web page (continued)*

```
ENTMAP OFF -
SPOOL ON

--Set-up page headings and footings
TTITLE LEFT "<h1>The Fictional Company</h1>" SKIP 1 -
      LEFT "<h2>Project Hours and Dollars Report</h2>"

--Format the columns
COLUMN employee_name HEADING "<p class=left>Employee Name</p>" FORMAT A40
COLUMN project_name HEADING "<p class=left>Project Name</p>" FORMAT A40
COLUMN hours_logged HEADING "<p class=right>Hours</p>" FORMAT 9,999
COLUMN dollars_charged HEADING "<p class=right>Dollars|Charged</p>" -
      FORMAT $999,999.99

--Turn off feedback and set TERMOUT off to prevent the
--report being scrolled to the screen.
SET FEEDBACK OFF
SET TERMOUT OFF

--Execute the query to generate the report.
SPOOL proj_hours_dollars.html
SELECT e.employee_name,
       p.project_name,
       SUM(ph.hours_logged) hours_logged,
       SUM(ph.dollars_charged) dollars_charged
FROM   employee e INNER JOIN project_hours ph
      ON e.employee_id = ph.employee_id
      INNER JOIN project p
      ON p.project_id = ph.project_id
GROUP BY e.employee_id, e.employee_name,
         p.project_id, p.project_name;
SPOOL OFF
EXIT
```

Example 6-5. CSS styles to format the output from Example 6-4

```
body {margin: 0; font-family: comic sans ms;
      background: black;}
table {background: black; color: white; width: 100%;}
table.detail {background: #eeeeee; color: black}
p.left {text-align: left}
p.right {text-align: right}
th {padding-left: 5px; padding-right: 5px; text-decoration: underline}
td {padding-left: 5px; padding-right: 5px;}
h1 {margin-bottom: 0;}
h2 {margin-top: 0}
```

There is no way to specify a class for the heading table, so I control that table's format through the table style, which sets the background and margin such that the top of the page is white text on a black background that fills the width of the browser window. There is only one other table in the document, the table that holds the data,

and for that table I can specify a class, which gives me a way to distinguish between the two tables. The `table.detail` style inherits the 100% width from `table`, and further specifies that the detail area of the page is to be black on a light-gray background.

And there you have it: a credible-looking web page in Figure 6-3 that is generated entirely via SQL*Plus.

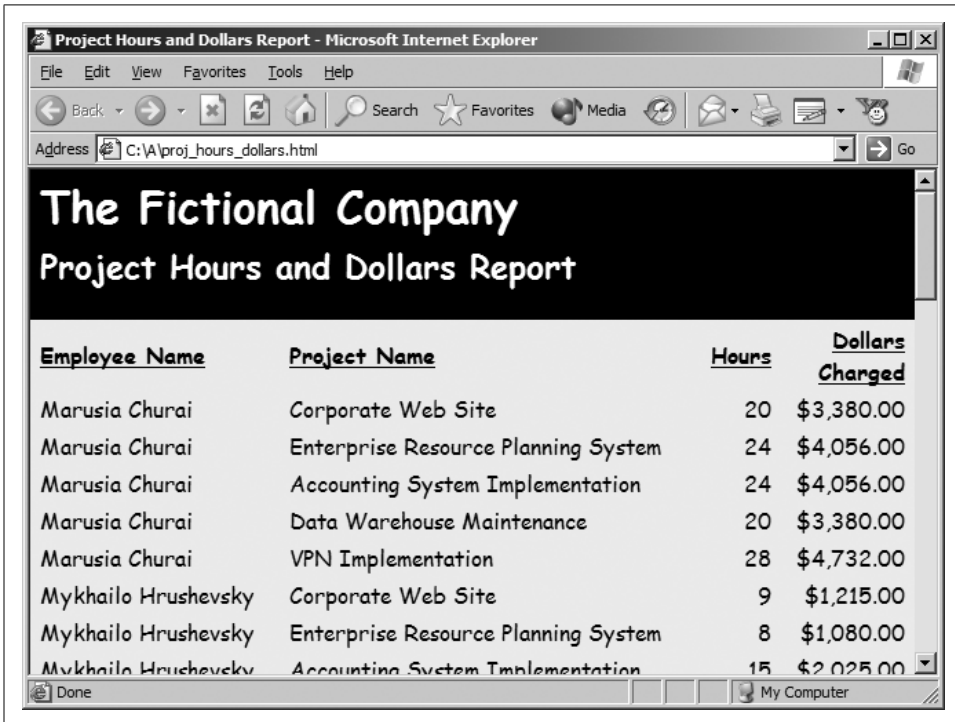


Figure 6-3. Output from Example 6-4 rendered using styles from Example 6-5

Another Approach to Headers

That SQL*Plus writes all page headers into an HTML table bothers me. In the previous section's example, that table represents unwanted markup. The associated `<table>` tag also includes some unwanted attribute settings. One of those, `width="90%"`, can cause you some grief depending on what it is that you want to accomplish in the way of formatting. An alternative, helpful approach is to dispense with using `TTITLE` altogether and use the `PROMPT` command to write page headings directly into the HTML output stream.

Example 6-6 shows a new version of the Project Hours and Dollars Report that uses `PROMPT` to write page headings to the HTML stream. Figure 6-4 shows the result as

rendered using the stylesheet in Example 6-7. This example introduces the concept of using <div> tags to format different sections of your HTML page.

Example 6-6. A script that uses PROMPT to write HTML page headings

```
SET ECHO OFF
SET PAGESIZE 50000
SET MARKUP -
  HTML ON -
  HEAD '<title>Project Hours and Dollars Report</title> -
  <link href="ex6-7.css" rel="stylesheet" type="text/css"/>' -
  BODY "" -
  TABLE 'class="detail"' -
  ENTMAP OFF -
  SPOOL ON

--Format the columns
COLUMN employee_name HEADING "<p class=left>Employee Name</p>" FORMAT A40
COLUMN project_name HEADING "<p class=left>Project Name</p>" FORMAT A40
COLUMN hours_logged HEADING "<p class=right>Hours</p>" FORMAT 9,999
COLUMN dollars_charged HEADING "<p class=right>Dollars Charged</p>" -
  FORMAT $999,999.99

--Turn off feedback and set TERMOUT off to prevent the
--report being scrolled to the screen.
SET FEEDBACK OFF
SET TERMOUT OFF

--Execute the query to generate the report.
SPOOL proj_hours_dollars.html
PROMPT <div class="top"> -
  <h1>The Fictional Company</h1> -
  <h2>Project Hours and Dollars Report</h2> -
  </div> -
  <div class="bottom">
SELECT e.employee_name,
       p.project_name,
       SUM(ph.hours_logged) hours_logged,
       SUM(ph.dollars_charged) dollars_charged
FROM employee e INNER JOIN project_hours ph
  ON e.employee_id = ph.employee_id
  INNER JOIN project p
  ON p.project_id = ph.project_id
GROUP BY e.employee_id, e.employee_name,
         p.project_id, p.project_name;
PROMPT </div>
SPOOL OFF
END
```

Example 6-7. CSS styles to format the output from Example 6-6

```
body {margin: 0; font-family: comic sans ms; background: black;}
div {margin-left: 5px; margin-top: 5px; margin-right: 5px;}
div.top {background: white; color: black;}
```

Example 6-7. CSS styles to format the output from Example 6-6 (continued)

```
padding-bottom: 5px; text-align: center;}
div.bottom {background: #eeeeee; color: black;}
table.detail {position: relative; top: -1.5em;}
p.left {text-align: left;}
p.right {text-align: right;}
th {padding-left: 5px; padding-right: 5px; text-decoration: underline;}
td {padding-left: 5px; padding-right: 5px;
padding-top: 0; padding-bottom: 0;}
h1 {margin-bottom: 0;}
h2 {margin-top: 0; margin-bottom: 0;}
```

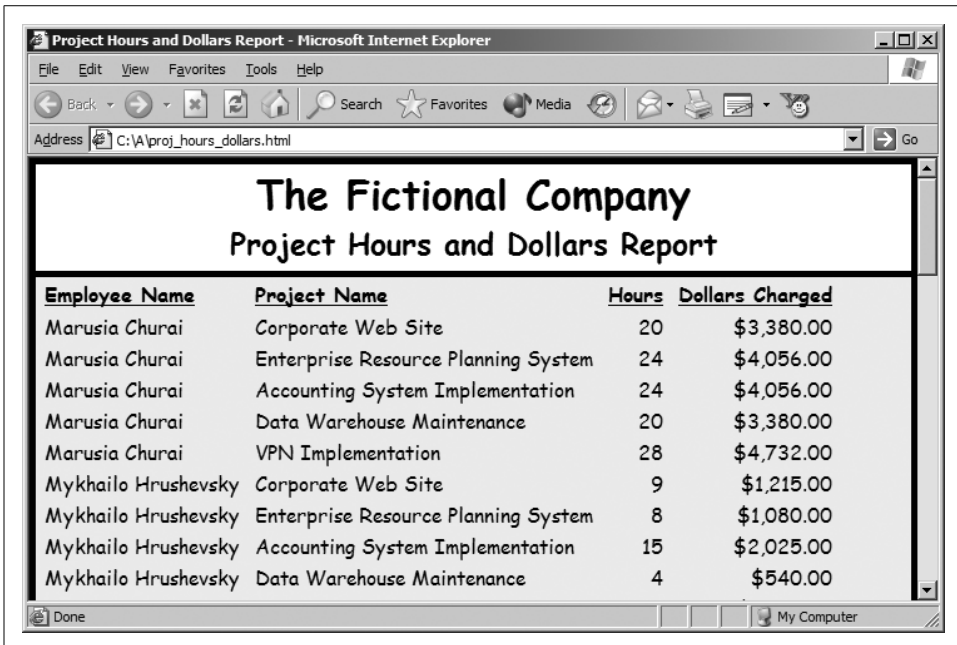


Figure 6-4. Output from Example 6-6 rendered using styles from Example 6-7



Getting consistently sized black borders around and between the page heading and report content is something I was unable to accomplish when those page headings were embedded within an HTML table. Your mileage may vary, but I found that the table and its 90% width got in the way of what I wanted to do.

Example 6-6 uses the following PROMPT command to write a good bit of markup to the HTML file:

```
PROMPT <div class="top"> -
      <h1>The Fictional Company</h1> -
```

```
<h2>Project Hours and Dollars Report</h2> -  
</div> -  
  
<div class="bottom">
```

I use one PROMPT command because the output from each PROMPT is followed by a `
` tag. One unwanted `
` is more than sufficient. The markup here writes out a header (h1) and subheader (h2), wraps those in a `<div>` classified as “top,” and begins a new `<div>` for the report detail. The detail `<div>` is closed by another PROMPT command that immediately follows the SELECT statement.

The stylesheet in Example 6-7 centers the h1 and h2 content and makes it black text on a white background. All of this is accomplished through the `div.top` style. A five-pixel margin is set for the left, top, and right side of each `<div>`. The black page background shows through that margin, giving the black borders that you see around and between the two sections of the page. The `<div>` containing the report detail is set to display black text on a light-gray background. These settings give the basic form to the page.

One unusual aspect of table formatting in this example is that the `table.detail` style specifies a negative margin, placing the top of the report detail higher on the page than it would otherwise belong. The intent is to compensate for the unwanted `
` and `<p>` tags that SQL*Plus writes just in front of the detail table, and the value of `-1.5em` is one that I found through experimentation. I think the results are pleasing, but you can omit that negative margin setting and live with a bit of extra blank space in front of the detail.

You might look at my approach in this section, that of using PROMPT to write out arbitrary markup, as a bit of a hack. And it is a bit of a hack. However, it’s a hack that offends my sense of elegance less than the idea of placing my page headings into a table, and it enables the use of `<div>` tags, which gives better control over the look and feel of the resulting web page.

Master/Detail Reports in HTML

The last HTML reporting challenge that I want to talk about is that of generating master/detail reports such as the Project Hours and Dollars Detail report generated by Example 5-8. For these reports, you cannot avoid the use of TTITLE. You do want to define a header that repeats throughout the report because you need a way to indicate when the master record changes.

Example 6-8 recreates the Project Hours and Dollars Detail report in HTML form. Figure 6-5 shows the result, and as you probably expect by now, Example 6-9 shows

the CSS stylesheet used to produce the layout that you see in the figure. Pay particular attention to the following aspects of this report example:

- The page heading, the one that occurs once at the top of the HTML page, is written to the HTML stream using PROMPT commands, just as in Example 6-6. PROMPT commands also write the <div> tags.
- TTITLE is used to specify the line that prints for each new master record. This line is written out as a level-3 heading using the <h3> tag. (Unfortunately, it's written into the leftmost cell of a three-column table.)
- Similar to what was done in the previous section, the <div> containing the page title is formatted as white text on black, while the <div> containing the report detail is formatted as black text on light gray.
- To distinguish the detail lines from the master lines, the detail table has been indented a bit and it has been given a white background. See the table.detail style in Example 6-9.
- The detail table has been given a slight, negative top margin to position each set of detail rows close to their respective master heading. This is optional, but the results are pleasing when rendered in Microsoft Internet Explorer.

This report certainly isn't the last word in the HTML formatting of master/detail reports. It's a good example, though, along with the other reports in this chapter, of how creative you can get using the combination of SQL*Plus, HTML, and CSS. (See Figure 6-5.)

*Example 6-8. SQL*Plus script to write a master/detail report in HTML*

```
SET ECHO OFF
SET PAGESIZE 50000
SET NEWPAGE 1
SET MARKUP -
  HTML ON -
  HEAD '<title>Project Hours and Dollars Detail</title> -
      <link href="ex6-9.css" rel="stylesheet" type="text/css"/>' -
  BODY "" -
  TABLE 'class="detail"' -
  ENTMAP OFF -
  SPOOL ON

--Set up the heading to use for each new employee
TTITLE LEFT "<h3>Employee: " FORMAT 9999 emp_id_var " " emp_name_var</h3>

--Format the columns
COLUMN employee_id NEW_VALUE emp_id_var NOPRINT
COLUMN employee_name NEW_VALUE emp_name_var NOPRINT
COLUMN project_id HEADING "<p class=right>Proj ID</p>" FORMAT 9999
COLUMN project_name HEADING "<p class=left>Project Name</p>" -
      FORMAT A26
COLUMN time_log_date HEADING "<p class=left>Date</p>" FORMAT A30
COLUMN hours_logged HEADING "<p class=right>Hours</p>" FORMAT 9,999
```

*Example 6-8. SQL*Plus script to write a master/detail report in HTML (continued)*

```
COLUMN dollars_charged HEADING "<p class=right>Dollars|Charged</p>" -
      FORMAT $999,999.99

--Breaks and computations
BREAK ON employee_id SKIP PAGE NODUPLICATES -
      ON employee_name NODUPLICATES -
      ON project_id SKIP 2 NODUPLICATES -
      ON project_name NODUPLICATES

--Turn off feedback and set TERMOUT off to prevent the
--report being scrolled to the screen.
SET FEEDBACK OFF
SET TERMOUT OFF

--Execute the query to generate the report.
SPOOL hours_detail.html
PROMPT <div class="top"> -
      <h1>The Fictional Company</h1> -
      <h2>Project Hours and Dollars Detail</h2> -
      </div><div class="bottom">
SELECT e.employee_id,
       e.employee_name,
       p.project_id,
       p.project_name,
       TO_CHAR(ph.time_log_date,'dd-Mon-yyyy') time_log_date,
       ph.hours_logged,
       ph.dollars_charged
FROM   employee e INNER JOIN project_hours ph
       ON e.employee_id = ph.employee_id
       INNER JOIN project p
       ON p.project_id = ph.project_id
ORDER BY e.employee_id, p.project_id, ph.time_log_date;
SPOOL OFF
EXIT
```

Example 6-9. CSS styles to format the output from Example 6-8

```
body {margin: 0; font-family: comic sans ms; background: white;}
div {margin: 10px;}
div.top {background: black; color: white;
         padding-bottom: 5px; text-align: center;}
div.bottom {background: silver; color: black;}
table.detail {position: relative; top: -1em;
             margin-left: 2em; background-color: white;}
p.left {text-align: left;}
p.right {text-align: right;}
th {padding-left: 5px; padding-right: 5px;
    text-decoration: underline;}
td {padding-left: 5px; padding-right: 5px;
    padding-top: 0; padding-bottom: 0;}
h1 {margin-bottom: 0;}
```

Example 6-9. CSS styles to format the output from Example 6-8 (continued)

```
h2 {margin-top: 0; margin-bottom: 0;}
```

```
h3 {margin-top: 0; margin-bottom: 1.25em;}
```

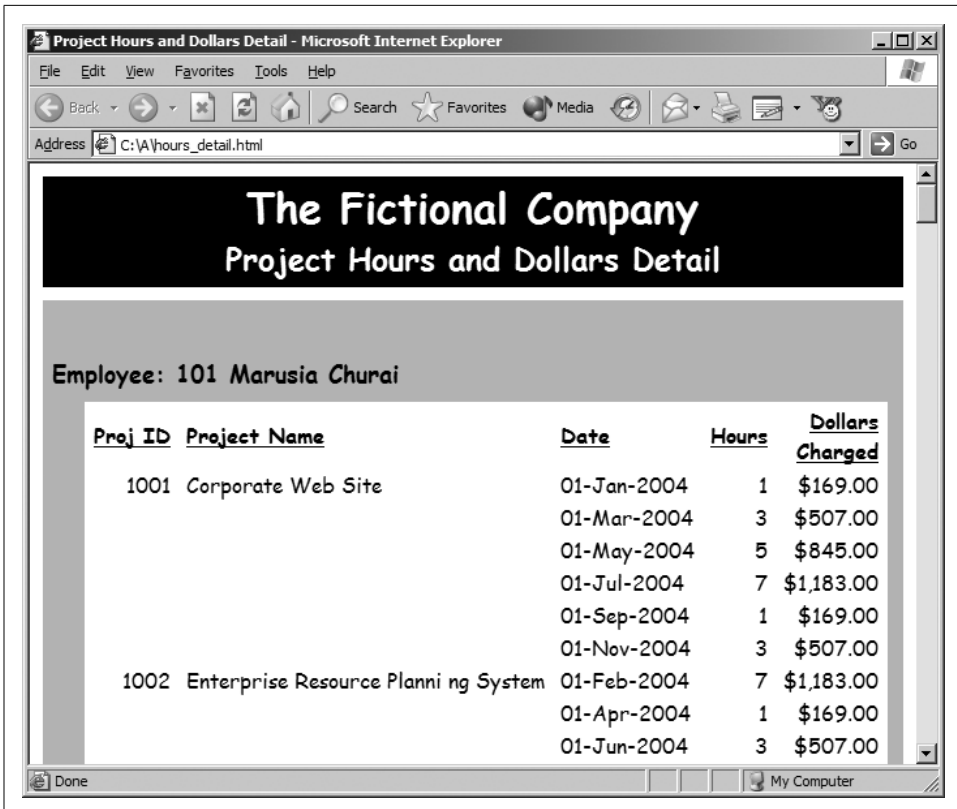


Figure 6-5. Example 6-8's master/detail report rendered using styles from Example 6-9