CIS 315 - Reading Packet: "Simple Reports - Part 2"

Sources:

* Oracle9i Programming: A Primer, Rajshekhar Sunderraman, Addison Wesley.
* Classic Oracle example tables **empl** and **dept**, adapted somewhat over the years

Introduction to enhancing simple ASCII reports with the help of SQL*Plus commands

[this section is being repeated from the Week 12 posting, so that all of the report intro will be here...]

You've seen how query results are displayed by default in SQL*Plus; they are usually OK, but sometimes you'd like something that looks "nicer". "Nicer" here might mean numbers formatted to the same number of decimal places, or with a nice title, or with a complete column heading, or even without ugly line-wrapping.

So, in this section we'll talk about SQL*Plus commands you can use to change how a query's results are *displayed*, so that they are more suitable for use as a *report* (a presentation of data that is well-formatted, attractive, and self-explanatory on its own to a reader).

One very short first command: if you simply type /,

```
/
```

...in SQL*Plus, that will cause the previous *SQL* command to be re-run. (Not the previous SQL*Plus command, mind you -- the previous SQL command.) This can be handy when you are tweaking your query formatting for a report.

For example, the last SQL command I performed was querying the salary_avgs view. If I now type just

```
/
```

...I'll again see the results of that query:

```
JOB        SALARY_AVG
---------- ----------
Analyst          3000
Manager    2758.33333
President        5000
Salesman         1400
```

clear command

We'll be discussing setting up break, column, and compute commands. A report script should first make sure that some *previous* values for these are not about to mess up our results. So, it is good form to

clear any previous values for these at the beginning of a report script:

```
clear     breaks
clear     columns
clear     computes

-- compliments of S. Griffin: yes, this works, too!!!

clear breaks columns computes
```

feedback

You know that little line that follws some query results, indicating how many rows were selected? It has a name -- it is called **feedback.**

It turns out that SQL*Plus includes commands that let you tweak this feedback, changing when it appears or even turning it off altogether.

First, if you just want to know the current value for feedback, this SQL*Plus command will tell you:

```
show feedback
```

And, here is how to set the feedback to a different number:

```
set feedback 3
```

The following, then, would let you see the effects of this:

```
show feedback

-- this'll note that 3 rows were selected.

select    *
from      painter;

-- this will not note that one row was:

select    *
from      painter
where     ptr_lname = 'Monet';
```

And sometimes, for a formal report, you just want to turn feedback off:

```
set feedback off
```

pagesize

pagesize is the number of lines in a "page" (the quantum that Oracle will display before re-displaying column headings, etc.)

You can see the current value of pagesize with:

```
show pagesize
```

...and you can set the pagesize to a desired value as so (here, I am setting it to 30 lines):

```
set pagesize 30
```

One nice trick to know: if you are essentially trying to write queries to generate a flat file of data for another program, you might set the pagesize to 0 to mean that you NEVER want page breaks.

```
set pagesize 0
```

linesize

linesize is used to indicate how many characters are in a line (before line-wrapping will occur).

You can see its current value with:

```
show linesize
```

...and you can reset it with something like this (here, I am setting it to 50 characters):

```
set linesize 50
```

newpage

If you have been looking closely, you may have noticed that each query has a blank line before its column headings. It so happens that there is a name for the number of blank lines that appear before the column headings or top title (if there is one) for each page: this is called **newpage**.

(It also appears that each SQL select statement's result starts on a new "page", pagesize- and and newpage-wise.)

To see the current value of newpage:

```
show newpage
```

Here's an example of setting it (here, I am setting it to 5 lines):

```
set newpage 5
```

Now I can also admit that, oddly enough, the number of lines in a page, in practice, is actually

pagesize + newpage

...odd but true!

And, again, when your goal is to create a flat file of data, setting newpage to 0 is a very good idea.

[the "new" Week 13 lab material begins here]

column command

The SQL*Plus **column** command is used to specify column formatting when you project a column in a query.

It is important to remember, especially when you start using the **column** command, that how you choose to format something does NOT change how it is actually stored in the database -- it only changes how it appears in the current query. A **column** command is only giving display preferences.

column has many options and possibilities, and I am just demonstrating a few of the most important here. You can google to find/read up on more, if you are interested (it looks like "Oracle sqlplus column command" has some promising results...)

The basic format for the **column** command is:

`column` *col_to_format* `heading` *desired_heading* `format` *desired format*

If you want blanks in a desired column heading, you should enclose the column in single or double quotes; if you want all of a heading to show, be sure to format it wide enough for all of that heading to fit! You can also specify that a heading print across multiple lines by putting in | in the heading where you want the next heading-line to begin.

column command - non-numeric columns

You specify the format of the column based on the type of data in that column, For **varchar2**, **char**, and **date** data, you use format **a** followed by how many characters wide you want that column to be displayed with.

So, the **column** command below is saying, for any column named **empl_last_name**, display it with the heading

```
Employee
Last Name
```

in a 25-character-wide column:

```
column empl_last_name heading 'Employee|Last Name' format a25
```

Try this to see how the **column** command affects how this query's results are displayed:

```
set linesize 80

select     *
from       empl;
```

If you don't have blanks in the heading, you don't have to have single quotes around it:

```
column empl_last_name heading Employee|Name format a25
/
```

...but you MUST have quotes if a column heading has a space (this will FAIL:)

```
column empl_last_name heading Employee|Last Name format a25
```

This shows that double quotes work, too:

```
column empl_last_name heading "Employee|Last Name" format a25
/
```

What do you think happens if you deliberately format an alphanumeric column too narrowly? Try this and see:

```
column empl_last_name heading 'Employee|Last Name' format a2
/
```

...but if you put TRUNCATED or TRU after a format, it will behave differently; try this to see how it behaves differently:

```
column empl_last_name heading 'Employee|Last Name' format a2 TRUNCATED
/
```

Putting WORD WRAPPED or WOR has a slightly different effect -- the following will demonstrate the difference (the default is actually named WRAPPED, shown here to demonstrate the difference):

```
insert into painting
values
('Waterlilies', '108');

insert into painting
values
('Yet four more', '108');

column ptg_title format a7 WOR

select *
from   painting;

column ptg_title format a7 WRAPPED
/
```

What if you just want to, say, format a column so that it is wide enough for its entire heading, but you don't want to specify a different heading? Then just don't put in a **heading** part:

```
column empl_num format a8
```

(Note that the empl_num column is actually declared to be char(4), so you do need **a** in its format...)

column command - numeric columns

For a numeric column, you do NOT use **a** in its format. Instead, you specify a numeric format pattern. There are many options for this, too, but here are a few basics:

* to format an integer to a certain width, express the format as that many 9's. It will then be right-justified in a field of that size.

```
99999
99
99999999
```

* if you want a floating-point value to be formatted with a certain number of decimal places, specify that by putting the decimal in as desired:

```
999.99
```

...would format a column to 2 decimal places.

* you can even include commas if you'd like large numbers to be formatted with them:

```
999,999,999.99
```

Here are some examples involving salary:

```
column salary heading Salary format 99999
select empl_last_name, dept_num, salary
from   empl;
```

Be careful -- Oracle behaves very differently if you format a numeric column to be too narrow than it does if you format a non-numeric column to be too narrow! Try this, and see what I mean:

```
column salary heading Salary format 99
/
```

Now format to a certain number of decimal places: (note that it rounds rather than truncates...)

```
column salary heading Salary format 99999.99
/
```

Now format values over 999 with commas:

```
column salary heading Salary format 99,999.99
/
```

Oh, and you can include a dollar sign, if you'd like:

```
column salary heading 'Salary' format $99,999.99
/
```

You can also ask to give one column the same format as another using **like**, as so:

```
column salary heading Salary format $99,999.99
column commission like salary heading 'Commission'
```

```
select     empl_last_name, salary, commission
from       empl
where      job_title = 'Salesman';
```

Views can work very nicely in reports:

```
drop view dept_avgs;

create view dept_avgs(dept_name, dept_avg) as
select      dept_name, avg(salary)
from        empl e, dept d
where       e.dept_num = d.dept_num
group by   dept_name;

column dept_avg heading "Dept Avg" format $99,999.99
column dept_name heading "Dept Name"

-- check out how much better these look!

select    *
from      dept_avgs
order by  dept_name;

select     *
from       dept_avgs
order by   dept_avg desc;
```

break command

The **break** command is used with queries including an **order-by** clause to get "prettier" ordered-row table displays. (And let's face it: the rows in reports should always be ordered in a way that makes sense for that report!)

Notice this query and its results:

```
select    dept_num, empl_last_name, salary
from      empl
order by  dept_num;

column dept_num heading 'Dept' format a4
/
```

See how the dept_num is repeated in consecutive rows? Well, all **break** does is make such a display "prettier" by only displaying the FIRST value when several rows have the SAME value. That is, try the following to see what I mean:

```
-- this BREAK causes only the "first" dept_num in a several consecutive to
-- display;

break on dept_num
/
```

You can even specify that you'd like 1 or more blank lines between each different dept_num:

```
-- I can get blank lines between each broken-into section:

break on dept_num skip 1
/
```

Only one **break** command can be in effect at a time; so put ALL of the columns you want to "break" on in a single break command...! Consider this:

```
column mgr heading Mgr
select    dept_num, mgr, empl_last_name, salary
from      empl
order by  dept_num, mgr;

-- can have the break effect on more than one column at a time ---
-- BUT only 1 break command can be in effect at one time, so
-- put ALL the columns you want to break on in a single break command

break on dept_num on mgr skip 1
/

-- and to NOT get the skip after each manager? (thanks to C. McLain)

break on dept_num skip 1 on mgr
/

break on dept_num skip 1 on mgr skip 2
/
```

You might remember that a SQL*Plus command is only supposed to be on ONE line. If a SQL*Plus command is getting too long -- and a break command can get long! -- you can CONTINUE to the next line (you can ask sqlplus to pretend it isn't a new line yet) by using a - at the end of the line:

```
break on dept_num -
on mgr skip 3
/
```

compute command

compute only makes sense when used with **break**. It just lets you specify that you'd like some computation to be done for the rows with the same value of something you are **break**'ing on...!

Study the results of executing the following to see what the **compute** is causing to happen here:

```
break on dept_num skip 1 on mgr
compute avg min max of salary on dept_num
/
```

By the way, you can type simply **compute** or **break** to see the current definition for these that you are using.

```
-- 'compute' will show you your current compute definition
```

```
compute

-- and 'break' will show you your current break definition

break
```

You know how there can only be one **break** command in effect at a time? You can have multiple **compute** commands -- but only 1 per column! If you try to put in a 2nd compute on the same column, the new version with replace the old.

```
compute count of empl_last_name on dept_num
/

-- TWO computes in effect now:

compute

-- does this one replace earlier?

compute count of salary on dept_num
/

-- yes!

compute
```

Here are a few other compute-related options students have let me know about:

```
-----------------------------------------------------------------------
-- to customize how your compute results are labeled:
--
-- label option for compute command: (compliments of Mr. Serrano)
--

column dept_num format a5
break on dept_num skip 1
compute sum label 'total' of salary on dept_num

--
select    dept_num, empl_last_name, salary
from      empl
order by  dept_num;
-----------------------------------------------------------------------

-----------------------------------------------------------------------
-- to get a "grand" (overall) computation:
--    (compliments of L. Holden)
--
-- "Breaking and computing "on report" provides a grand total for
--    an entire report.... See code below, it computes a total of
--    employees by department and a grand total of all employees:"

break on dept_num skip 1 on REPORT
compute count of empl_num on dept_num
compute count label Total of empl_num on REPORT
```

```
column dept_num format a7
column empl_num format a7
set pagesize 53

select dept_num, empl_num
from empl
order by dept_num;
```

top and bottom titles

You can specify top titles or bottom titles for each "page" using **ttitle** and **btitle.** Here's how you can see the current values set for these:

```
show ttitle
show btitle
```

...and here are examples showing how you can specify top and bottom titles:

```
-- want a TITLE aTOP each page? ttitle

ttitle 'Beautiful|Three Line|Top Title'

-- want a BOTTOM title? btitle

btitle 'Gorgeous Two-line|Bottom Title'

/
```

GOOD REPORT SCRIPT ETIQUETTE

Once you change any of these display settings, they stay changed until you change them again, or until you exit your SQL*Plus session. So, if you run a script, and then type in additional commands at the SQL> prompt, those additional commands will have whatever display settings were made in that script!

This can be startling to unwary users, so, at the end of a report script (any script that modifies the display settings), you SHOULD "clean up", setting the display settings back to their "default" values.

Ms. Koyuncu noted that you could easily put these "cleanup" commands into their own script, and then just call that script at the end of your report script. That would be very slick indeed.

```
--*******************************************************---
-- AT THE END OF A REPORT SCRIPT, YOU *SHOULD*****
-- clean up when done (so as to not shock user with their
-- next query)

-- better to put the below lines into another cleanup
--    script you can call frequently! (thanks to T. Koyuncu)
-- @ cleanup

clear breaks
clear columns
clear computes
```

```
set space  1
set feedback      6
set pagesize      14
set linesize      80
set newpage       1
set heading       on

-- to turn off titles set!
ttitle off
btitle off
```

flat file example

As a little bonus, here is an example of creating *almost* a comma-separated flat file of data from a database (which actually appears to work properly when I tried if in Fall 2009! 8-)):

```
--*****************************************************---
-- quick flat file example:
--*****************************************************---

-- aha! space is # of spaces BETWEEN columns; default is 1

set space 0

set newpage 0
set linesize 80
set pagesize 0
set echo off
set feedback off
set space 0

set newpage 0
set linesize 80
set pagesize 0
set echo off
set feedback off
set heading off

spool  flat_empl_data.txt

select  empl_last_name || ',' || salary
from    empl;

-- don't forget to spool off, or results file may be empty or
-- incomplete;

spool off

-- AT THE END OF A REPORT SCRIPT, YOU *SHOULD*****
-- clean up when done (so as to not shock user with their
-- next query)

clear breaks
clear columns
clear computes
```

```
set space  1
set feedback      6
set pagesize      14
set linesize      80
set newpage       1
set heading       on

-- to turn off titles set!
ttitle off
btitle off
```

Some useful string- and date- and time-related functions

This section discusses some Oracle functions related to strings, dates, and times that can be handy in creating more-readable/"prettier" queries and reports. It is not an exhaustive coverage; the goal is to give you some idea of the possibilities (so you can explore further as inspiration strikes you).

Reminder: concatenation

We'll start with a reminder of a string operation we have already discussed and practiced: concatenation! (Why? well, your project's final milestone is coming up, and several well-formatted reports are required, and concatenation can definitely help in producing readable, attractive reports!)

Hopefully, then, you recall that || can be used to combine one or more string literals or columns, projecting the combined result as a single column. So, for example, the following query projects a single column, combining each employee last name, a ', $', and employee salary:

```
select empl_last_name || ', $' || salary "Pay Info"
from empl
order by empl_last_name;
```

Assuming that I've restored the `empl` table to its usual 14 rows, the above query will result in:

```
Pay Info
-----------------------------------------------------
Adams, $1100
Blake, $2850
Ford, $3000
James, $950
Jones, $2975
King, $5000
Martin, $1250
Michaels, $1600
Miller, $1300
Raimi, $2450
Scott, $3000

Pay Info
-----------------------------------------------------
Smith, $800
Turner, $1500
Ward, $1250
```

```
14 rows selected.
```

When creating a report, concatenation can frequently be used to create more-readable results. As just a few examples:

* if you have first and last names for people, and you wish to display them alphabetically (as in a class role, or a phone directory), it looks good to concatenate them last name first, with a comma in-between

```
select last_name || ', ' || first_name "Name"
from ...
where ...
order by last_name;
```

...which might look like:

```
Name
-----------------------------------
Adams, Annie
Cartwright, Josh
Zeff, Pat
```

* ...although for a mailing list, or name tags, etc., you'd probably prefer to have the first name first, and maybe you'd even order them by first name:

```
select first_name || ' ' || last_name "Attendees"
from ...
where ...
order by last_name;
```

...which might look like:

```
Attendees
-----------------------------------
Annie Adams
Josh Cartwright
Pat Zeff
```

* and many combinations of street, city, state, and zip columns are possible:

```
select city || ', ' ||  state || '  ' || zip
from ...
where ...

select zip || '-' || city
from ...
where ...

select state || ': ' || city
from ...
```

```
where ...
```

...etc., and these can be ordered by city and then zip, by state and then city and then zip, by zip, by some other column (such as last name or department or salary or hiredate), etc., depending on what's appropriate for that query.

Reminder: date-related function: sysdate

We've already seen one date-related function: **sysdate**. You may recall that this function returns the current date:

```
insert into empl(empl_num, empl_last_name, job_title, mgr, hiredate, salary,
                 dept_num)
values
('6745', 'Zeff', 'Analyst', '7566', sysdate, 3000, '200');
```

...and the hiredate for Zeff will be the date that this insertion was performed. And sysdate can be used in a select as well -- this simply projects the current date for each row in the "dummy" table dual, which only has one column and one row, and so simply projects the current date. So if I run the following on April 27th:

```
select sysdate
from dual;
```

....then the result would be:

```
SYSDATE
---------
27-APR-09
```

Date- and time-related function: to_char

Now, for some additional functions. Oracle function **to_char** expects a date or a number and a format string, and it returns a character-string version of the given date or number based on that given format.

A complete coverage of all of the possibilities for the format string is beyond the scope of this introduction, but you can easily find out more on the Web. Here are a few examples, though, to give you some ideas of the the possibilities:

For example, this will project just the month of the given date, projecting that month as the entire name of that month:

```
select empl_last_name, to_char(hiredate, 'MONTH') "MONTH HIRED"
from empl;
```

...resulting in:

```
EMPL_LAST_NAME  MONTH HIR
--------------- ---------
King            NOVEMBER
Jones           APRIL
Blake           MAY
```

```
Raimi           JUNE
Ford            DECEMBER
Smith           DECEMBER
Michaels        FEBRUARY
Ward            FEBRUARY
Martin          SEPTEMBER
Scott           NOVEMBER
Turner          SEPTEMBER

EMPL_LAST_NAME  MONTH HIR
--------------- ---------
Adams           SEPTEMBER
James           DECEMBER
Miller          JANUARY
Zeff            APRIL

15 rows selected.
```

If you'd like the month with an uppercase first letter and lowercase letter for the rest, use the format string `'Month'` (and here we'll use a column command, too, to get a non-chopped heading):

```
column hiremonth heading "Month Hired" format a11

select empl_last_name "Last Name", to_char(hiredate, 'Month') hiremonth
from empl;
```

...resulting in:

```
Last Name       Month Hired
--------------- -----------
King            November
Jones           April
Blake           May
Raimi           June
Ford            December
Smith           December
Michaels        February
Ward            February
Martin          September
Scott           November
Turner          September

Last Name       Month Hired
--------------- -----------
Adams           September
James           December
Miller          January
Zeff            April

15 rows selected.
```

These format examples could easily get a bit long-winded, so here are a few more examples all in one query (and some of these also show how you can include some literals in the format strings, too):

```
select to_char(sysdate, 'YYYY') year,
       to_char(sysdate, 'Mon YYYY') mon_year,
       to_char(sysdate, 'MM-DD-YY') num_version,
       to_char(sysdate, 'Day, Month DD, YYYY') long_version,
       to_char(sysdate, 'DY - Mon DD - YY') brief_versn
from   dual;
```

Granted, sometimes you get surprises -- when run on 4-27-09, the above results in:

```
YEAR MON_YEAR NUM_VERS LONG_VERSION                  BRIEF_VERSN
---- -------- -------- ---------------------------- -----------------
2009 Apr 2009 04-27-09 Monday , April     27, 2009   MON - Apr 27 - 09
```

I think the "gaps" are based on including the space needed for the "longest" weekday and month names; there are string functions you can use to get rid of such spaces, which we'll discuss shortly, for times when you don't want those gaps.

Here is a summary of some of the available date-related format strings for use in a **to_char** format string:

```
'MM'        - month number
'MON'       - the first 3 letters of the month name, all-uppercase
'Mon'       - the first 3 letters of the month name, mixed case
'MONTH'     - the entire month name, all-uppercase
'Month'     - the entire month name, mixed case
'DAY'       - fully spelled out day of the week, all-uppercase
'Day'       - fully spelled out day of the week, mixed case
'DY'        - 3-letter abbreviation of the day of the week, all-uppercase
'Dy'        - 3-letter abbreviation of the day of the week, mixed case
'DD'        - date of the month, written as a 2-digit number
'YY'        - the last two digits of the year
'YYYY'      - the year written out in four digits
```

even:

```
'D'         - number of date's day in the current week (Sunday is 1)
'DD'        - number of date's day in the current month
'DDD'       - number of date's day in the current year
```

Now, why did I say that **to_char** was a time-related function as well? Because, although it is not obvious, you can store both a date and a time in a column of type DATE -- and you can then project the time information of a given date with format strings such as:

```
'HH12'      - hours of the day (1-12)
'HH24'      - hours of the day (0-23)
'MI'        - minutes of the hour
'SS'        - seconds of the minute
'AM'        - displays AM or PM depending on the time
```

...and when I ran the following at about 11:05 pm on Monday, April 27th:

```
select to_char( sysdate, 'D DD DDD Day, Mon YYYY - HH12 HH24 MI SS AM') "UGLY"
```

```
from dual;
```

...the result was:

```
UGLY
-----------------------------------------
2 27 117 Monday   , Apr 2009 - 11 23 05 30 PM
```

a few more examples of date-related operations and functions

Have you noticed yet that the Oracle Date type supports + and -? If you add a number to a date, the result is the date that results from adding that number of days to that date! If run on April 27th, 2009, then:

```
select sysdate + 1
from dual;
```

...results in:

```
SYSDATE+1
---------
28-APR-09
```

Now, you'll find that this addition or subtraction will work fine with a column declared to be a date -- but what if, for whatever reason, you want to add or subtract from a date literal? (Or if you want to use some date function given a date literal?) You'll find that the string that you use for insertion will not work:

-- FAILS!!
```
select '31-DEC-08' + 1
from dual;
```

...with the error message:

```
ERROR at line 1:
ORA-01722: invalid number
```

But:

to_date - expects a date-string, and returns the corresponding date

...can allow you to do this: (and this example now demonstrates how, yes, the month and year boundaries are indeed handled reasonably):

```
select to_date('31-DEC-08') + 1
from dual;
```

...results in:

```
TO_DATE('
```

```
---------
01-JAN-09
```

next_day - expects a date and a string representing the day of the week, and returns the date of the next date after the given date that is on that day of the week

If you remember that April 27, 2009 was a Monday, then:

```
select next_day('27-Apr-2009', 'TUESDAY') nxt_tues,
       next_day('27-Apr-2009', 'MONDAY') nxt_mon,
       next_day('27-Apr-2009', 'FRIDAY') nxt_fri
from dual;
```

...results in:

```
NXT_TUES  NXT_MON   NXT_FRI
--------- --------- ---------
28-APR-09 04-MAY-09 01-MAY-09
```

add_months - expects a date and a number of months, and results in the date that many months from the given date;
months_between - expects two dates, and returns the number of months between those two dates (positive if the first date is later than the second, negative otherwise)

```
select add_months('30-Jan-09', 1) one_mth_later,
       months_between('15-Apr-09', '15-Jan-09') diff1,
       months_between('15-Apr-09', '01-Jun-09') diff2
from dual;
```

...results in:

```
ONE_MTH_L      DIFF1      DIFF2
--------- ---------- ----------
28-FEB-09          3 -1.5483871
```

A few string-related functions

initcap - expects a string, and returns a string with an initial uppercase letter

```
select initcap('SILLY') looky
from dual;
```

...results in:

```
LOOKY
-----
Silly
```

lower - expects a string, and returns an all-lowercase version of your string
upper - expects a string, and returns an all-uppercase version of your string

```
select lower(empl_last_name), upper(empl_last_name)
```

```
from empl
where job_title = 'President';
```

...results in:

```
LOWER(EMPL_LAST UPPER(EMPL_LAST
--------------- ---------------
king            KING
```

lpad - "left pad" - expects a string, a desired length, and a padding character, and returns a string that is the given string padded on the left with the given padding character to result in a string with the desired length

rpad - "right pad" - expects a string, a desired length, and a padding character, and returns a string that is the given string padded on the right with the given padding character to result in a string with the desired length

```
select lpad(empl_last_name, 12, '.') dots, rpad(empl_last_name, 15, '?') huh,
       lpad(empl_last_name, 12, ' ') right_justifd
from empl;
```

...results in:

```
DOTS         HUH             RIGHT_JUSTIF
------------ --------------- ------------
........King King??????????         King
.......Jones Jones??????????        Jones
.......Blake Blake??????????        Blake
.......Raimi Raimi??????????        Raimi
........Ford Ford??????????         Ford
.......Smith Smith??????????        Smith
....Michaels Michaels???????     Michaels
........Ward Ward??????????         Ward
......Martin Martin?????????       Martin
.......Scott Scott??????????        Scott
......Turner Turner?????????       Turner

DOTS         HUH             RIGHT_JUSTIF
------------ --------------- ------------
.......Adams Adams??????????        Adams
.......James James??????????        James
......Miller Miller?????????       Miller
........Zeff Zeff??????????         Zeff

15 rows selected.
```

And, of course, if a function returns a string, then a call to that function can be used wherever a string is permitted, including within another function call:

```
select lpad( to_char(hiredate, 'Day'), 14, ' ') ||
       to_char(hiredate, '- Month YY') "Hiredate"
from empl;
```

...which results in:

```
Hiredate
----------------------------
     Sunday   - November  91
     Tuesday  - April     91
     Wednesday- May       91
     Sunday   - June      91
     Tuesday  - December  91
     Monday   - December  90
     Wednesday- February  91
     Friday   - February  91
     Saturday - September 91
     Saturday - November  91
     Sunday   - September 91

Hiredate
----------------------------
     Monday   - September 91
     Tuesday  - December  91
     Thursday - January   92
     Monday   - April     09

15 rows selected.
```

ltrim - expects a string, returns that string with any leading blanks (blanks starting the string) removed
rtrim - expects a string, returns that string with any trailing banks (blanks ending the string) removed

```
select ltrim('   Hi   ') lftchop, rtrim('   Hi   ') rtchop,
       rtrim(to_char(sysdate, 'Day')) || ', ' || rtrim(to_char(sysdate, 'Month'))
          || ' ' || to_char(sysdate, 'DD, YYYY') nicer
from dual;
```

...which results in:

```
LFTCH RTCHO NICER
----- ----- ----------------------------
Hi       Hi Monday, April 27, 2009
```

length - expects a string, and returns the number of character in that string (its length)
substr - expects a string, the position to start at in that string (where the first character is position 1), and how long a substring is desired, and returns the substring of that length starting at that position.
(if the 3rd argment is omitted, it returns the rest of the string starting at the given position)

```
select empl_last_name,
       length(empl_last_name) length,
       substr(empl_last_name, 1, 3) abb1,
       substr(empl_last_name, 3) rest
from empl;
```

...which results in:

```
EMPL_LAST_NAME      LENGTH ABB REST
--------------- ---------- --- -------------
```

```
King                      4 Kin ng
Jones                     5 Jon nes
Blake                     5 Bla ake
Raimi                     5 Rai imi
Ford                      4 For rd
Smith                     5 Smi ith
Michaels                  8 Mic chaels
Ward                      4 War rd
Martin                    6 Mar rtin
Scott                     5 Sco ott
Turner                    6 Tur rner

EMPL_LAST_NAME      LENGTH ABB REST
--------------- ---------- --- -------------
Adams                     5 Ada ams
James                     5 Jam mes
Miller                    6 Mil ller
Zeff                      4 Zef ff

15 rows selected.
```

Again, please note: this is not an exhaustive list of the additional functions that Oracle provides. But it hopefully gives you an idea of the rich set of possibilities available.