CS 325 - SQL Reading Packet 8: "Simple Reports - Parts 1 and 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 essentially repeated from the previous packet, so that all of the report intro will be in one place for your future reference...]

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 start to 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** (which we'll informally define as a presentation of data that is **well-formatted**, **attractive**, and **self-explanatory on its own to a reader**).

One very short reminder, to start: 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
---------- ----------
Manager    2758.33333
Analyst          3000
President        5000
Sales            1400
```

clear command

We'll be discussing setting up `break`, `column`, and `compute` commands in the next reading packet. 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
```

Or, you can combine these:

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

clear breaks columns computes
```

feedback

You know that little line that follows 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` setting, changing when this feedback 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
```

...which by default shows the following value for `feedback`:

```
FEEDBACK ON for 6 or more rows
```

This means you get the feedback message only for results of 6 rows or more, but not for results with fewer rows. This is why, for a query such as:

```
select *
from   short_empl3;
```

...you get the results (including feedback) of:

```
LAST_NAME        POSITION
--------------- ----------
King            President
Jones           Manager
Blake           Manager
Raimi           Manager
Ford            Analyst
Michaels        Sales
Ward            Sales
Martin          Sales
Scott           Analyst
Turner          Sales
```

```
10 rows selected.
```

...but for a query such as:

```
select *
from   short_empl3
where  position = 'Manager';
```

...you get the results (now not including feedback) of:

```
LAST_NAME       POSITION
--------------- ----------
Jones           Manager
Blake           Manager
Raimi           Manager
```

And, here is how to set the `feedback` setting to a **different** value:

```
set feedback 3
```

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

```
show feedback
```

...which now has the result:

```
FEEDBACK ON for 3 or more rows
```

And if you now type:

```
/
```

...you'll now get the results including feedback:

```
LAST_NAME       POSITION
--------------- ----------
Jones           Manager
Blake           Manager
Raimi           Manager

3 rows selected.
```

But, queries with less than 3 rows still will not get a feedback message:

```
select *
from   short_empl3
where  position = 'Analyst';
```

...which has the results (without feedback) of:

```
LAST_NAME       POSITION
--------------- ----------
Ford            Analyst
Scott           Analyst
```

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

```
set feedback off
```

Now there will be no feedback message regardless of the number of rows -- indeed, the SQL*Plus `SQL>` prompt looks like it now goes directly after the query results!:

```
select *
from   short_empl3;
```

...now has the results (JUST this once I'm also showing the next `SQL>` prompt that you'd get running this in `SQL*Plus`, to illustrate what I mean):

```
LAST_NAME       POSITION
--------------- ----------
King            President
Jones           Manager
Blake           Manager
Raimi           Manager
Ford            Analyst
Michaels        Sales
Ward            Sales
Martin          Sales
Scott           Analyst
Turner          Sales
SQL>
```

For this packet's example purposes -- and as one would do for politeness/good practice at the end of a script -- we'll reset `feedback` back to its default value of `6` for now:

```
set feedback 6
```

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 the `pagesize` setting with:

```
show pagesize
```

...which has the result:

```
pagesize 14
```

This is the number of displayed lines, not the number of rows -- if I now re-run the `set-up-ex-tbls.sql` script:

```
start set-up-ex-tbls.sql
```

...and then run the query:

```
select *
from   short_empl3;
```

...the results are:

```
LAST_NAME        POSITION
--------------- ----------
King            President
Jones           Manager
Blake           Manager
Raimi           Manager
Ford            Analyst
Smith           Clerk
Michaels        Sales
Ward            Sales
Martin          Sales
Scott           Analyst
Turner          Sales

LAST_NAME        POSITION
--------------- ----------
Adams           Clerk
James           Clerk
Miller          Clerk

14 rows selected.
```

Notice that, if you count the lines from the first `LAST_NAME POSITION` headings until they are repeated, that is indeed 14 lines.

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

```
set pagesize 30
```

If I now re-run the previous query:

```
/
```

...now the headings are not repeated after 14 lines, because of the larger `pagesize`:

```
LAST_NAME        POSITION
--------------- ----------
King            President
Jones           Manager
Blake           Manager
Raimi           Manager
Ford            Analyst
```

```
Smith           Clerk
Michaels        Sales
Ward            Sales
Martin          Sales
Scott           Analyst
Turner          Sales
Adams           Clerk
James           Clerk
Miller          Clerk

14 rows selected.
```

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
```

Interestingly, this seems to suppress column headings completely in HSU's current version of Oracle (still the case as of Fall 2019) -- re-running the previous query:

```
/
```

...now has the result (this time including both the command and the next `SQL>` prompt for emphasis):

```
SQL> /
King            President
Jones           Manager
Blake           Manager
Raimi           Manager
Ford            Analyst
Smith           Clerk
Michaels        Sales
Ward            Sales
Martin          Sales
Scott           Analyst
Turner          Sales
Adams           Clerk
James           Clerk
Miller          Clerk

14 rows selected.

SQL>
```

For this packet's example purposes -- and as one would do for politeness/good practice at the end of a script -- we'll reset `pagesize` back to its default value of `14` for now:

```
set pagesize 14
```

linesize

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

PLEASE NOTE: this does not affect the line-wrapping that may occur in an `ssh` window if it is narrower than the line being displayed -- that will tend to override this setting. But if `linesize` is smaller than the width of one's `ssh` window, you'll see that the line-wrapping occurs based on `linesize` (and lines in a `spool`ed file should show line-wrapping based on `linesize` as well).

You can see its current value with:

```
show linesize
```

...which has the result:

```
linesize 80
```

So, right now, in a sufficiently-wide `ssh` window,

```
select *
from   empl;
```

... has the results:

```
EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE  SALARY     COMMISSION DEP
---- --------------- ---------- ---- --------- ---------- ---------- ---
7839 King            President       17-NOV-11       5000            500
7566 Jones           Manager    7839 02-APR-12       2975            200
7698 Blake           Manager    7839 01-MAY-13       2850            300
7782 Raimi           Manager    7839 09-JUN-12       2450            100
7902 Ford            Analyst    7566 03-DEC-12       3000            200
7369 Smith           Clerk      7902 17-DEC-12        800            200
7499 Michaels        Sales      7698 20-FEB-18       1600        300 300
7521 Ward            Sales      7698 22-FEB-19       1250        500 300
7654 Martin          Sales      7698 28-SEP-18       1250       1400 300
7788 Scott           Analyst    7566 09-NOV-18       3000            200
7844 Turner          Sales      7698 08-SEP-19       1500          0 300


EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE  SALARY     COMMISSION DEP
---- --------------- ---------- ---- --------- ---------- ---------- ---
7876 Adams           Clerk      7788 23-SEP-18       1100            400
7900 James           Clerk      7698 03-DEC-17        950            300
7934 Miller          Clerk      7782 23-JAN-16       1300            100

14 rows selected.
```

You can reset it with `set linesize` like this (here, I am setting it to 50 characters):

```
set linesize 50
```

And now,

```
/
```

...has the results:

```
EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE
---- --------------- ---------- ---- ---------
    SALARY COMMISSION DEP
---------- ---------- ---
7839 King            President       17-NOV-11
      5000            500

7566 Jones           Manager    7839 02-APR-12
      2975            200

7698 Blake           Manager    7839 01-MAY-13
      2850            300


EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE
---- --------------- ---------- ---- ---------
    SALARY COMMISSION DEP
---------- ---------- ---
7782 Raimi           Manager    7839 09-JUN-12
      2450            100

7902 Ford            Analyst    7566 03-DEC-12
      3000            200

7369 Smith           Clerk      7902 17-DEC-12
       800            200


EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE
---- --------------- ---------- ---- ---------
    SALARY COMMISSION DEP
---------- ---------- ---
7499 Michaels        Sales      7698 20-FEB-18
      1600        300 300

7521 Ward            Sales      7698 22-FEB-19
      1250        500 300

7654 Martin          Sales      7698 28-SEP-18
      1250       1400 300


EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE
---- --------------- ---------- ---- ---------
    SALARY COMMISSION DEP
---------- ---------- ---
7788 Scott           Analyst    7566 09-NOV-18
      3000            200

7844 Turner          Sales      7698 08-SEP-19
      1500          0 300

7876 Adams           Clerk      7788 23-SEP-18
```

```
      1100            400


EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE
---- --------------- ---------- ---- ---------
    SALARY COMMISSION DEP
---------- ---------- ---
7900 James           Clerk      7698 03-DEC-17
       950            300

7934 Miller          Clerk      7782 23-JAN-16
      1300            100

14 rows selected.
```

Setting `linesize` to be longer for, say, a report with long rows that will be printed using landscape orientation (and perhaps using a smaller font size) would likely make it much more readable.

For this packet's example purposes -- and as one would do for politeness/good practice at the end of a script -- we'll reset `linesize` back to its default value of `80` for now:

```
set linesize 80
```

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 this is also a SQL*Plus setting with 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 the `newpage` setting:

```
show newpage
```

...which has the result:

```
newpage 1
```

So, right now,

```
select *
from   short_empl3;
```

...has the results (including the command and the `SQL>` prompt afterwards this time for better illustration):

```
SQL> select *
```

```
  2  from   short_empl3;

LAST_NAME       POSITION
--------------- ----------
King            President
Jones           Manager
Blake           Manager
Raimi           Manager
Ford            Analyst
Smith           Clerk
Michaels        Sales
Ward            Sales
Martin          Sales
Scott           Analyst
Turner          Sales

LAST_NAME       POSITION
--------------- ----------
Adams           Clerk
James           Clerk
Miller          Clerk

14 rows selected.

SQL>
```

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

```
set newpage 5
```

Now, re-running the previous query:

```
/
```

...has the results (again including the command and the `SQL>` prompt afterwards this time for better illustration):

```
SQL> /





LAST_NAME       POSITION
--------------- ----------
King            President
Jones           Manager
Blake           Manager
Raimi           Manager
Ford            Analyst
Smith           Clerk
Michaels        Sales
```

```
LAST_NAME        POSITION
--------------- ----------
Ward            Sales
Martin          Sales
Scott           Analyst
Turner          Sales
Adams           Clerk
James           Clerk
Miller          Clerk

14 rows selected.

SQL>
```

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

And, as this is the end of this packet, as one would do for politeness/good practice at the end of a script -- we'll reset `newpage` back to its default value of `1` for now:

```
set newpage 1
```

[the "new" simple-reports 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 can be abbreviated as `col` (that is, it is fine to use either `column` or `col`).

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*

`col` *col_to_format* `heading` *desired_heading* `format` *desired_format*

If you want blanks in a desired column heading, you should enclose the *desired_heading* 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:

```
col empl_last_name heading 'Employee|Last Name' format a20
```

...is saying, for any column named `empl_last_name`, display it with the heading

```
Employee
Last Name
```

in a 20-character-wide column.

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

```
col empl_last_name heading 'Employee|Last Name' format a20

select     *
from       empl;
```

...which has the results:

```
     Employee
EMPL Last Name            JOB_TITLE  MGR  HIREDATE      SALARY COMMISSION DEP
---- -------------------- ---------- ---- --------- ---------- ---------- ---
7839 King                 President       17-NOV-11       5000            500
7566 Jones                Manager    7839 02-APR-12       2975            200
7698 Blake                Manager    7839 01-MAY-13       2850            300
7782 Raimi                Manager    7839 09-JUN-12       2450            100
7902 Ford                 Analyst    7566 03-DEC-12       3000            200
7369 Smith                Clerk      7902 17-DEC-12        800            200
7499 Michaels             Sales      7698 20-FEB-18       1600        300 300
7521 Ward                 Sales      7698 22-FEB-19       1250        500 300
7654 Martin               Sales      7698 28-SEP-18       1250       1400 300
7788 Scott                Analyst    7566 09-NOV-18       3000            200

     Employee
EMPL Last Name            JOB_TITLE  MGR  HIREDATE      SALARY COMMISSION DEP
---- -------------------- ---------- ---- --------- ---------- ---------- ---
7844 Turner               Sales      7698 08-SEP-19       1500          0 300
7876 Adams                Clerk      7788 23-SEP-18       1100            400
7900 James                Clerk      7698 03-DEC-17        950            300
7934 Miller               Clerk      7782 23-JAN-16       1300            100

14 rows selected.
```

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

```
col empl_last_name heading Employee|Name format a20

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

...which has the results:

```
Employee
Name                     SALARY
-------------------- ----------
Smith                       800
Adams                      1100
James                       950
Miller                     1300
```

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

```
col empl_last_name heading Employee|Last Name format a20
```

...which results in the error message:

```
SP2-0158: unknown COLUMN option "Name"
```

This shows that double quotes work, too:

```
col empl_last_name heading "Employee|Last Name" format a20
/
```

...with the results:

```
Employee
Last Name                SALARY
-------------------- ----------
Smith                       800
Adams                      1100
James                       950
Miller                     1300
```

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

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

...which has the results:

```
Em
La     SALARY
-- ----------
Sm        800
it
h
```

```
Ad       1100
am
s

Ja        950
me

Em
La     SALARY
-- ----------
s

Mi       1300
ll
er
```

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

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

...which has the results:

```
Em
La     SALARY
-- ----------
Sm        800
Ad       1100
Ja        950
Mi       1300
```

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):

```
-- note: this is using the painting table created in SQL Reading Packet 6 -
--     Set-theoretic operations, more on modifying data, and sequences
--     (And it assumes that the painter table also created there indeed still
--     has painters with ptr_num values of 104 and 106.)

delete from painting;

insert into painting
values
(1002, 'Waterlilies', 104);

insert into painting
values
(1003, 'Yet four more', 106);

col ptg_title format a7 WOR

select *
from   painting;
```

...which has the results:

```
    PTG_ID PTG_TIT    PTR_NUM
---------- ------- ----------
      1002 Waterli        104
           lies

      1003 Yet            106
           four
           more
```

Compare this to the results you get with the (default) option `WRAPPED`:

```
col ptg_title format a7 WRAPPED
/
```

...which has the results:

```
    PTG_ID PTG_TIT    PTR_NUM
---------- ------- ----------
      1002 Waterli        104
           lies

      1003 Yet fou        106
           r more
```

```
rollback;
```

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:

```
col empl_num format a8

select empl_num, empl_last_name
from   empl
where  job_title = 'Clerk';
```

...which has the results (since we still have the `a2 TRUNCATED` format for `empl_last_name`):

```
         Em
EMPL_NUM La
-------- --
7369     Sm
7876     Ad
7900     Ja
7934     Mi
```

(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 a numeric value as 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; for example,

```
99999
```

...would right-justify numbers with no fractional places in a field of size 5,

```
99
```

...would right-justify numbers with no fractional places in a field of size 2, and

```
99999999
```

...would right-justify numbers with no fractional places in a field of size 9.

* if you want a numeric value to be formatted with a certain number of decimal places, specify that by putting the decimal in as desired; for example,

```
999.99
```

...would format a numeric column to 2 decimal places. (And here, it would do so right-justifying them in a field of size 6.)

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

```
999,999,999.99
```

Here are some examples involving salary:

```
col empl_last_name heading 'Last name' format a15
col salary heading Salary format 99999

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

...which has the results:

```
Last name       Salary
--------------- ------
Smith              800
Adams             1100
James              950
Miller            1300
```

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 you should see what I mean:

```
col salary heading Salary format 99
/
```

...which has the results:

```
Last name       Salary
--------------- ------
Smith              ###
Adams              ###
James              ###
Miller             ###
```

Now trying out formatting numeric values with a certain number of decimal places: (note that it **rounds** rather than truncates, and this is JUST in terms of display, NOT what is stored in the table!)

```
col salary heading Salary format 99999.99
/
```

...which has the results:

```
Last name          Salary
--------------- ---------
Smith              800.00
Adams             1100.00
James              950.00
Miller            1300.00
```

Now trying out formatting values over 999 with commas:

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

...which has the results:

```
Last name           Salary
--------------- ----------
Smith               800.00
Adams             1,100.00
James               950.00
Miller            1,300.00
```

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

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

...which has the results:

```
Last name            Salary
--------------- -----------
Smith               $800.00
Adams             $1,100.00
James               $950.00
```

```
Miller          $1,300.00
```

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

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

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

...which has the results:

```
Last name            Salary  Commission
--------------- ----------- -----------
Michaels          $1,600.00     $300.00
Ward              $1,250.00     $500.00
Martin            $1,250.00   $1,400.00
Turner            $1,500.00        $.00
```

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;

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

-- check out how much better these look!

select    *
from      dept_avgs
order by  dept_name;
```

...which has the results:

```
Dept Name          Dept Avg
--------------- -----------
Accounting        $1,875.00
Management        $5,000.00
Operations        $1,100.00
Research          $2,443.75
Sales             $1,566.67
```

And:

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

...which has the results:

```
Dept Name          Dept Avg
--------------- -----------
Management        $5,000.00
Research          $2,443.75
Accounting        $1,875.00
Sales             $1,566.67
Operations        $1,100.00
```

break command

The **break** command is used with queries that include 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!)

Consider the following:

```
col dept_num heading 'Dept' format a4
col empl_last_name heading 'Last name' format a15
col salary heading Salary format $99,999.99

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

...which has the results:

```
Dept Last name            Salary
---- --------------- -----------
100  Miller            $1,300.00
100  Raimi             $2,450.00
200  Scott             $3,000.00
200  Jones             $2,975.00
200  Ford              $3,000.00
200  Smith               $800.00
300  Martin            $1,250.00
300  Ward              $1,250.00
300  Blake             $2,850.00
300  Michaels          $1,600.00
300  James               $950.00

Dept Last name            Salary
---- --------------- -----------
300  Turner            $1,500.00
400  Adams             $1,100.00
500  King              $5,000.00

14 rows selected.
```

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 several consecutive to
-- display;

break on dept_num
/
```

...which has the results:

```
Dept Last name            Salary
---- --------------- -----------
100  Miller            $1,300.00
     Raimi             $2,450.00
200  Scott             $3,000.00
     Jones             $2,975.00
     Ford              $3,000.00
     Smith               $800.00
300  Martin            $1,250.00
     Ward              $1,250.00
     Blake             $2,850.00
     Michaels          $1,600.00
     James               $950.00

Dept Last name            Salary
---- --------------- -----------
300  Turner            $1,500.00
400  Adams             $1,100.00
500  King              $5,000.00

14 rows selected.
```

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 using skip:

break on dept_num skip 1
/
```

...which has the results:

```
Dept Last name            Salary
---- --------------- -----------
100  Miller            $1,300.00
     Raimi             $2,450.00

200  Scott             $3,000.00
     Jones             $2,975.00
     Ford              $3,000.00
     Smith               $800.00

300  Martin            $1,250.00
     Ward              $1,250.00
     Blake             $2,850.00

Dept Last name            Salary
```

```
---- --------------- -----------
300  Michaels         $1,600.00
     James              $950.00
     Turner           $1,500.00

400  Adams            $1,100.00

500  King             $5,000.00


14 rows selected.
```

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:

```
col mgr heading Mgr

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

...which has the results:

```
Dept Mgr  Last name            Salary
---- ---- --------------- -----------
100  7782 Miller            $1,300.00
     7839 Raimi             $2,450.00

200  7566 Scott             $3,000.00
     7566 Ford              $3,000.00
     7839 Jones             $2,975.00
     7902 Smith               $800.00

300  7698 Michaels          $1,600.00
     7698 James               $950.00
     7698 Turner            $1,500.00

Dept Mgr  Last name            Salary
---- ---- --------------- -----------
300  7698 Ward              $1,250.00
     7698 Martin            $1,250.00
     7839 Blake             $2,850.00

400  7788 Adams             $1,100.00

500       King              $5,000.00


14 rows selected.
```

To break on `dept_num` AND `mgr`, add `on mgr` to the `break` command:

```
-- 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
/
```

...which has the results:

```
Dept Mgr  Last name            Salary
---- ---- --------------- -----------
100  7782 Miller            $1,300.00

     7839 Raimi             $2,450.00

200  7566 Scott             $3,000.00
          Ford              $3,000.00

     7839 Jones             $2,975.00

     7902 Smith               $800.00


Dept Mgr  Last name            Salary
---- ---- --------------- -----------
300  7698 Michaels          $1,600.00
          James               $950.00
          Turner            $1,500.00
          Ward              $1,250.00
          Martin            $1,250.00

     7839 Blake             $2,850.00

400  7788 Adams             $1,100.00

500        King             $5,000.00

14 rows selected.
```

And to NOT get the skip after each manager? Put the `skip 1` immediately after the `on_dept_num` part, instead of at the end of the `break` command (thanks to C. McLain):

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

...which has the results:

```
Dept Mgr  Last name            Salary
---- ---- --------------- -----------
100  7782 Miller            $1,300.00
     7839 Raimi             $2,450.00

200  7566 Scott             $3,000.00
          Ford              $3,000.00
     7839 Jones             $2,975.00
     7902 Smith               $800.00

300  7698 Michaels          $1,600.00
```

```
          James                $950.00
          Turner             $1,500.00

Dept Mgr  Last name             Salary
---- ---- --------------- -----------
300  7698 Ward               $1,250.00
          Martin             $1,250.00
     7839 Blake              $2,850.00

400  7788 Adams              $1,100.00

500     King                 $5,000.00


14 rows selected.
```

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 - (a single dash) at the end of the line:

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

...which has the results:

```
Dept Mgr  Last name             Salary
---- ---- --------------- -----------
100  7782 Miller             $1,300.00
     7839 Raimi              $2,450.00


200  7566 Scott              $3,000.00
          Ford               $3,000.00
     7839 Jones              $2,975.00
     7902 Smith                $800.00


300  7698 Michaels           $1,600.00

Dept Mgr  Last name             Salary
---- ---- --------------- -----------
300  7698 James                $950.00
          Turner             $1,500.00
          Ward               $1,250.00
          Martin             $1,250.00
     7839 Blake              $2,850.00


400  7788 Adams              $1,100.00


500       King               $5,000.00
```

```
14 rows selected.
```

compute command

The SQL*Plus **compute** command 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` command is causing to happen here:

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

...which has the results:

```
Dept Mgr  Last name            Salary
---- ---- --------------- -----------
100  7782 Miller            $1,300.00
     7839 Raimi             $2,450.00
**** ****                 -----------
avg                         $1,875.00
mini                        $1,300.00
maxi                        $2,450.00

200  7566 Scott             $3,000.00
          Ford              $3,000.00
     7839 Jones             $2,975.00
     7902 Smith               $800.00

Dept Mgr  Last name            Salary
---- ---- --------------- -----------
**** ****                 -----------
avg                         $2,443.75
mini                          $800.00
maxi                        $3,000.00

300  7698 Michaels          $1,600.00
          James               $950.00
          Turner            $1,500.00
          Ward              $1,250.00
          Martin            $1,250.00
     7839 Blake             $2,850.00

Dept Mgr  Last name            Salary
---- ---- --------------- -----------
**** ****                 -----------
avg                         $1,566.67
mini                          $950.00
maxi                        $2,850.00

400  7788 Adams             $1,100.00
```

```
**** ****                 -----------
avg                         $1,100.00
mini                        $1,100.00
maxi                        $1,100.00


Dept Mgr  Last name            Salary
---- ---- --------------- -----------
500       King              $5,000.00
**** ****                 -----------
avg                         $5,000.00
mini                        $5,000.00
maxi                        $5,000.00


14 rows selected.
```

See how, each time a `break` occurs on `dept_num`, `compute` causes the average, minimum, and maximum of the `salary` for the rows with that `dept_num` to be computed and displayed?

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
```

...which has the results:

```
COMPUTE avg LABEL 'avg' minimum LABEL 'minimum' maximum LABEL 'maximum' OF salary ON dept_num
```

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

break
```

...which has the results:

```
break on dept_num skip 1 nodup
          on mgr nodup
```

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
/
```

...which has the results:

```
Dept Mgr  Last name            Salary
---- ---- --------------- -----------
100  7782 Miller            $1,300.00
     7839 Raimi             $2,450.00
**** **** --------------- -----------
avg                         $1,875.00
coun                    2
```

```
mini                          $1,300.00
maxi                          $2,450.00

200  7566 Scott               $3,000.00
          Ford                $3,000.00
     7839 Jones               $2,975.00

Dept Mgr  Last name                Salary
---- ---- --------------- -----------
200  7902 Smith                 $800.00
**** **** --------------- -----------
avg                           $2,443.75
coun                    4
mini                            $800.00
maxi                          $3,000.00

300  7698 Michaels            $1,600.00
          James                 $950.00
          Turner              $1,500.00
          Ward                $1,250.00

Dept Mgr  Last name                Salary
---- ---- --------------- -----------
300  7698 Martin              $1,250.00
     7839 Blake               $2,850.00
**** **** --------------- -----------
avg                           $1,566.67
coun                    6
mini                            $950.00
maxi                          $2,850.00

400  7788 Adams               $1,100.00
**** **** --------------- -----------
avg                           $1,100.00

Dept Mgr  Last name                Salary
---- ---- --------------- -----------
coun                    1
mini                          $1,100.00
maxi                          $1,100.00

500       King                $5,000.00
**** **** --------------- -----------
avg                           $5,000.00
coun                    1
mini                          $5,000.00
maxi                          $5,000.00


14 rows selected.
```

See how, now, each time a `break` occurs on `dept_num`, the first `compute` causes the average, minimum, and maximum of the `salary` for the rows with that `dept_num` to be computed and displayed, and the second `compute` causes the count of rows with that `dept_num` with non-null `empl_last_name` to be computed and displayed?

And we can see that two computes are indeed in effect:

```
-- TWO computes in effect now:

compute
```

...which has the results:

```
COMPUTE avg LABEL 'avg' minimum LABEL 'minimum' maximum LABEL 'maximum' OF salary ON dept_num
COMPUTE count LABEL 'count' OF empl_last_name ON dept_num
```

And, to seek to show that one `compute` can replace another:

```
-- does this compute on salary replace the current compute on salary?

compute count of salary on dept_num
/
```

...which has the results:

```
Dept Mgr  Last name            Salary
---- ---- --------------- -----------
100  7782 Miller            $1,300.00
     7839 Raimi             $2,450.00
**** **** --------------- -----------
coun                    2           2

200  7566 Scott             $3,000.00
          Ford              $3,000.00
     7839 Jones             $2,975.00
     7902 Smith               $800.00
**** **** --------------- -----------
coun                    4           4

Dept Mgr  Last name            Salary
---- ---- --------------- -----------

300  7698 Michaels          $1,600.00
          James               $950.00
          Turner            $1,500.00
          Ward              $1,250.00
          Martin            $1,250.00
     7839 Blake             $2,850.00
**** **** --------------- -----------
coun                    6           6

400  7788 Adams             $1,100.00

Dept Mgr  Last name            Salary
---- ---- --------------- -----------
**** **** --------------- -----------
coun                    1           1

500       King              $5,000.00
**** **** --------------- -----------
coun                    1           1
```

```
14 rows selected.
```

See, in the above results, how the computations of average, minimum, maximum salary after each `dept_num` break are now replaced with the computation of count of salaries (the number of rows with non-null salary) after each `dept_num`?

```
-- and for further confirmation:

compute
```

...which has the results:

```
COMPUTE count LABEL 'count' OF empl_last_name ON dept_num
COMPUTE count LABEL 'count' OF salary ON dept_num
```

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

This first example shows that you can change the **label** for a `compute`'s computation -- that is, you can change what text is displayed along with the computation. Below, the `sum` computation's label is changed to `total` (bold formatting added for emphasis):

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

col 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;
-------------------------------------------------------------------
```

...which has the results:

```
Dept  Last name            Salary
----- --------------- -----------
100   Miller            $1,300.00
      Raimi             $2,450.00
***** --------------- -----------
count               2
total                   $3,750.00

200   Scott             $3,000.00
      Jones             $2,975.00
      Ford              $3,000.00
      Smith               $800.00
***** --------------- -----------

Dept  Last name            Salary
----- --------------- -----------
```

```
count              4
total                   $9,775.00

300   Martin            $1,250.00
      Ward              $1,250.00
      Blake             $2,850.00
      Michaels          $1,600.00
      James               $950.00
      Turner            $1,500.00
***** --------------- -----------
count              6

Dept  Last name            Salary
----- --------------- -----------
total                   $9,400.00

400   Adams             $1,100.00
***** --------------- -----------
count              1
total                   $1,100.00

500   King              $5,000.00
***** --------------- -----------
count              1
total                   $5,000.00

14 rows selected.
```

And it turns out that you can a final/overall computation for all of the rows at the end of a query using `compute`'s `on report` clause. Here is an example (bold formatting added for emphasis):

```
-------------------------------------------------------------------
-- 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
col dept_num heading "Dept" format a8
col empl_num heading "Empl Num" format a8
set pagesize 53

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

...which has the results:

```
Dept     Empl Num
-------- --------
100          7934
```

```
             7782
******** --------
count           2

200          7788
             7566
             7902
             7369
******** --------
count           4

300          7654
             7521
             7698
             7499
             7900
             7844
******** --------
count           6

400          7876
******** --------
count           1

500          7839
******** --------
count           1

         --------
Total          14

14 rows selected.
```

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'

/
```

...which has the results: HERE PLEASE in progress!

```
Thu Nov 21                                                         page    1
                                 Beautiful
                                 Three Line
                                 Top Title

Dept     Empl Num
-------- --------
100          7934
             7782
******** --------
count           2

200          7788
             7566
             7902
             7369
******** --------
count           4

300          7654
             7521
             7698
             7499
             7900
             7844
******** --------
count           6

400          7876
******** --------
count           1

500          7839
******** --------
count           1

         --------
Total          14









                               Gorgeous Two-line
                                  Bottom Title

14 rows selected.
```

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 a comma-separated flat file of data from a database (which indeed appeared to work properly when I put these into a SQL script and ran that script in November 2019):

```
--****************************************************---
-- 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 is 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 November 21st:

```
select sysdate
from dual;
```

....then the result would be:

```
SYSDATE
---------
21-NOV-19
```

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            NOVEMBER

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):

```
col 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            November

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):

```
col mon_year format a8
col long_version format a29
col brief_versn format a17

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 2019-11-21, the above results in:

```
YEAR MON_YEAR NUM_VERS LONG_VERSION                  BRIEF_VERSN
---- -------- -------- ----------------------------- -----------------
2019 Nov 2019 11-21-19 Thursday , November  21, 2019 THU - Nov 21 - 19
```

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 10:18 pm on Thursday, November 21st:

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

...the result was:

```
UGLY
--------------------------------------------------------------------------------
5 21 325 Thursday , Nov 2019 - 10 22 18 19 PM
```

a few more examples of date-related operations and functions

function `to_date`

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 November 21, 2019, then:

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

...results in:

```
SYSDATE+1
```

```
---------
22-NOV-19
```

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-18' + 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-18') + 1
from dual;
```

...results in:

```
TO_DATE('
---------
01-JAN-19
```

function `next_day`

`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 November 21, 2019 was a Thursday, then:

```
select next_day('21-Nov-2019', 'TUESDAY') nxt_tues,
       next_day('21-Nov-2019', 'MONDAY') nxt_mon,
       next_day('21-Nov-2019', 'FRIDAY') nxt_fri
from dual;
```

...results in:

```
NXT_TUES  NXT_MON   NXT_FRI
--------- --------- ---------
26-NOV-19 25-NOV-19 22-NOV-19
```

functions `add_months` and `months_between`

`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-19', 1) one_mth_later,
       months_between('15-Apr-19', '15-Jan-19') diff1,
       months_between('15-Apr-19', '01-Jun-19') diff2
from dual;
```

...results in:

```
ONE_MTH_L      DIFF1      DIFF2
--------- ---------- ----------
28-FEB-19          3 -1.5483871
```

A few string-related functions

function `initcap`

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

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

...results in:

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

functions `lower` and `upper`

`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
```

functions `lpad` and `rpad`

`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

```
col dots format a12 tru
col huh format a15 tru
col right_justif format a12 tru

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:

```
col "Hiredate" format a28

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

...which results in:

```
Hiredate
----------------------------
     Thursday - November  11
     Monday   - April     12
     Wednesday- May       13
     Saturday - June      12
     Monday   - December  12
     Monday   - December  12
     Tuesday  - February  18
     Friday   - February  19
     Friday   - September 18
     Friday   - November  18
     Sunday   - September 19

Hiredate
----------------------------
     Sunday   - September 18
     Sunday   - December  17
     Saturday - January   16
     Thursday - November  19

15 rows selected.
```

functions `ltrim` and `rtrim`

`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

```
col nicer format a30

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, when run on 2019-11-21, resulted in:

```
LFTCH RTCHO NICER
----- ----- -----------------------------
Hi       Hi Thursday, November 21, 2019
```

functions `length` and `substr`

`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 argument is omitted, it returns the rest of the string starting at the given position)

```
col abb1 format a3
col rest format a13

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.