Table of Contents

I am starting with the numeric functions. the function we use the most is the Round function, following by Floor and Ceiling. We also use the Rand to generate random numbers.

These demos are all run with literals- such as shown here. Instead of including these selects, I have used the expression as the column header for the result. I have adjusted column widths to fit the page.

```
  SQRT(64)
----------
         8
```

1. Math functions: Abs, Power, Sign, Sqrt

I am discussing Abs, Sign, Power, Sqrt, and Mod here because they are pretty simple. Oracle also includes functions to do trig calculations; we won't be using those.

Demo 01: **ABS** returns the absolute value of the argument

```
  ABS(12)     ABS(-12)       ABS(0)    ABS(125.85)   ABS(-74.15)
----------   ----------   ----------   -----------   -----------
        12           12            0        125.85         74.15
```

Demo 02: SIGN returns only 0, 1, or –1;

if the argument is 0, sign returns 0;
if the argument is positive, sign returns +1;
if the argument is negative, sign returns -1.

```
       SIGN (0)          SIGN (12)        SIGN (-12)    SIGN (12.32)
---------------   ---------------   --------------- ---------------
              0                 1                -1               1
```

Demo 03: **POWER** (a, b) is used to calculate a raised to the b power

```
       POWER (3, 2)          POWER (10, 3)        POWER (10, -3)      POWER (4.5, 3.2)
--------------------   --------------------   --------------------   --------------------
                   9                   1000                   .001             123.106234
```

Demo 04: **SQRT** returns the square root of its argument; the argument must be a non-negative number or you get an error

```
  SQRT(64)    SQRT(68.56)   SQRT(ABS(-679.34))      SQRT(0)
----------   -----------   ------------------   ----------
         8    8.28009662           26.0641516            0
```

Demo 05: **MOD (a, b)** returns the remainder after a is divided by b. Mod makes the most sense when used with integers.

If mod (a, 1) = 0 then a is an integer
If mod (a, 2) = 0 then a is even;
If mod (a, 2) = 1 or -1 then a is odd
Note the result when the second argument is 0.

If you have a time duration expressed as minutes, you can determine the number of full hours as trunc(theTime/60) and minutes as mod (theTime, 60)

```
  MOD(18, 12)   MOD(18, 6)    MOD(6, 18)    MOD(18.6, 12)  MOD(18, 6.4)   MOD(18.6, 6.4)
------------  ------------  ------------   --------------  ------------  ---------------
           6             0             6              6.6           5.2              5.8

   MOD(10,3)   MOD(-10,3)   MOD(10,-3)  MOD(-10,-3)
------------ ------------ ------------ ------------
           1           -1            1           -1

  MOD(128,1) MOD(128.002,1) MOD(128.00,1)  MOD(35,2) MOD(368,2)
------------ -------------- ------------- ---------- ----------
           0           .002             0          1          0

    MOD(0,3)    MOD(10,0)     MOD(0,0)
------------ ------------ ------------
           0           10            0
```

2. Rounding functions

The next set of numeric functions returns a value close to the argument.

Demo 06: **ROUND** returns the number rounded at the specified precision. The precision defaults to 0. You can have a negative precision. Round rounds up at .5

```
ROUND(45.678,0)   ROUND(45.2,0)   ROUND(46.5,0)   ROUND(45.678,2)
---------------   -------------   -------------   ---------------
             46              45              47             45.68

ROUND(45.55,1)   ROUND(-46.5,0)   ROUND(345.67,-2)   ROUND(45,-2)
--------------  --------------   ----------------   ------------
          45.6              -47                300              0
```

Demo 07: **TRUNC** returns the number truncated at the specified precision

```
TRUNC(45.678,0)   TRUNC(45.678,2)   TRUNC(45.55,0)   TRUNC(-45.55,0)
---------------   ---------------   --------------   ---------------
             45             45.67               45               -45

TRUNC(345.67,-2)   TRUNC(399.99,-2)   TRUNC(399.99,-5)
----------------   ----------------   ----------------
             300                300                  0
```

Demo 08: **CEIL** returns the smallest integer greater than or equal to the argument

```
  CEIL(10)   CEIL(10.2)   CEIL(-10.5)   CEIL(10.8)
----------   ----------   -----------   ----------
        10           11           -10           11
```

Demo 09: **FLOOR** returns the largest integer less than or equal to the argument

```
 FLOOR(10)   FLOOR(10.5)   FLOOR(-10.5)   FLOOR(10.8)
----------   -----------   ------------   -----------
        10            10            -11            10
```

Comparison of Round, Ceiling, Floor, and Truncate. Suppose you had calculated a numeric value and you needed to display it with 2 digits after the decimal. How would you want to display the value 25.0279? This often comes up in issues such as calculating sales tax- do we round up or round down? The decision about which one is correct is a business decision- not a programming decision. But we can use the following small table to see our choices in terms of SQL

Demo 10: Comparison of Round, Ceil ,Floor and Trunc

```
Create table z_tst_numerics ( id integer, val_1 float);
insert into z_tst_numerics values (1,  25.0034);
insert into z_tst_numerics values (2,  25.0079);
insert into z_tst_numerics values (3,  25.0279) ;
insert into z_tst_numerics values (4,  25.4239) ;
insert into z_tst_numerics values (5, -25.0279) ;
insert into z_tst_numerics values (6, -25.4239) ;
```

Demo 11: Comparison query

```
Select id
, val_1
, ROUND(val_1,2)                  as "round"
, TRUNC(val_1,2)                  as "trunc"
, CEIL(val_1 * 100.00)/100.00     as "ceil"
, FLOOR(val_1* 100.00)/100.00     as "floor"
From z_tst_numerics
;
```

```
ID   VAL_1       round       trunc       ceil         floor
---- ---------- ---------- ----------  ----------  ----------
1       25.0034      25        25           25.01       25
2       25.0079      25.01     25           25.01       25
3       25.0279      25.03     25.02        25.03       25.02
4       25.4239      25.42     25.42        25.43       25.42
5      -25.0279     -25.03    -25.02       -25.02      -25.03
6      -25.4239     -25.42    -25.42       -25.42      -25.43
```

Demo 12: Using a negative precision of -1 with round gives you values rounded off to the nearest 10 dollars.

```
Select
  quoted_price as price
, round(quoted_price, -1) as Price_10
, round(quoted_price, -2) as Price_100
From oe_order_details
;
Selected rows
```

```
     PRICE   PRICE_10  PRICE_100
---------- ---------- ----------
        15         20          0
      14.5         10          0
      4.25          0          0
     75.99         80        100
      25.5         30          0
       150        150        200
    149.99        150        100
     12.95         10          0
       300        300        300
       225        230        200
```

3. Random values- pseudo-random values

Strictly speaking the technique to produce random values in Oracle is not a function, but it is an Oracle supplied technique and it makes sense to discuss it here.

We use the term random numbers to refer to a series of numbers that are produced by some process where the next number to be generated cannot be predicted. We also assume with random numbers that if we generate a set of 10,000 random integers between 1 and 10, then each of the values 1 through 10 should occur approximately the same number of times. Of course, we are not quite saying what "approximately" the same number of times really means.

There are discussions about what "truly" random means and for some applications there are random numbers that are based on using atmosphere noise or the degradation of atomic particles.

For many programming purposes, pseudo-random numbers are good enough. Pseudo-random numbers are generated by an algorithm which is provided with a seed value to get the series of values started. If you give the generator the same seed it will produce the same series of numbers. This is very good for test purposes, so that you can run your program repeatedly against the same series of test data. The pseudo-random number generators also work without a seed, in which case the seed is generally based on the computer time- that means each time you run the generator you get a different set of numbers. This is also good for testing programs.

Many pseudo-random number generators allow you to also specify the range of values you want as output- for example integers between 1 and 100, or floating point numbers between -10 and +10.

One use for random numbers is to create test values to insert into a table. Generating 100 or 100,000 rows of random test data can be handled with random functions.

3.1. DBMS_RANDOM

DBMS_RANDOM is a supplied Oracle package for generating pseudo random numbers or strings. For the rest of this discussion, I'll use the term "random numbers "to refer to pseudo-random numbers generated by the Oracle DBMS_RANDOM package.

Demo 13: Let's start with an example. Generating random values - run this several times

```
Select dbms_random.value
From dual;
```

These are some typical values returned by the query- the query returns a single value.

```
0.87564450157828969250026606645729508674
```

```
0.03293085802772244969039334971831440769
```

```
0.5715936725133017963338688533991109245
```

```
0.35305488184218107629434141212297758639
```

```
0.17470228115667371240150793918577648334
```

```
0.23249386861943613971605269193433067208
```

You can specify a Low and a High parameter for a range of values. The values will range from Low (and can equal Low) up to High (but not equal to High)

Demo 14: Generating random values between 10 and 25 and converting to an integer

```
Select Floor(dbms_random.value(10, 26))
From dual;
```

I ran that query 10 times and got the following values.

```
21
12
21
24
17
12
14
19
14
22
```

There are 16 values between 10 and 25 (including the endpoints). If I use the expression in the previous demo to generate 16000 values, I would expect to get each values approximate 1000 times. Using a loop technique I generated those numbers and counted them. The results were are shown here. Note that the number 26 did not occur at all. Also note that the values 10 and 25 occurred about the same number of times.

number	10	11	12	13	14	15	16	17
count	1028	1046	10006	10004	990	953	1023	1035

number	18	19	20	21	22	23	24	25
count	993	966	978	1070	997	980	957	974

You may see people using the round function - such as round(dbms_random.value(10,25)). Using that expression and generating 16000 values, I get a count of 544 for value 10 and 522 for value 25 and the other values were all between 1032 and 1109. This certainly does not give the numbers at the end points the same chance as the others.

Demo 15: Random values from -10 to +10

```
Select floor(dbms_random.value(-10, +11 ))
From dual;
```

Demo 16: Random values from 0.05 to 0.08

```
Select round(dbms_random.value(50, 81)/1000 , 2)
From dual;
-- several runs
```

```
0.07
0.05
0.08
0.08
```

Demo 17: I could assign the random value to a variable

```
variable num number;
exec :num :=  dbms_random.value;
print :num;
```

```
       NUM
----------
.715401877
```